જાવામાં એક અથવા વધારે ચલ નીચે જણાવેલી વાક્યરચના પ્રમાણે ઘોષિતવિધાન (declaration statement) દ્વારા ઘોષિત કરી શકાય :

<type-name> {variable-names};

અહીં વાક્યરચનામાં ઉપયોગમાં લીધેલી પદ્ધતિ નીચે પ્રમાણે છે :

- કોણીય કૌંસ < > ઉપયોગકર્તા દ્વારા જણાવવાના ઘટકનો નિર્દેશ કરે છે.
- છગડિયા કૌંસ { } અલ્પવિરામથી અલગ કરેલા ઘટકોની યાદીનો નિર્દેશ કરે છે.

અહીં {variable-names} ચલનાં નામોની યાદીનો નિર્દેશ કરે છે. જ્યારે યાદીમાં એક કરતાં વધારે ઘટક હોય, ત્યારે અલ્પવિરામથી જુદી પાડવામાં આવે છે.

<type-name>ને ચલના ડેટાપ્રકાર સૂચવતા ચાવીરૂપ શબ્દ (keyword) વડે બદલવામાં આવે છે. તે ચલનું કદ નક્કી કરવા તેમાં કયા પ્રકારની કિંમતો તે ધરાવી શકે તેમજ તેના ઉપર કાર્યો કરી શકાય તે જણાવવા માટે વપરાય છે. જ્યારે કમ્પ્યૂટર ચલઘોષિત વિધાનનો (variable declaration statement) અમલ કરે છે, ત્યારે ચલ માટે અલગ મેમરી સુયોજિત (સેટ) કરે છે અને તે મેમરી સાથે ચલનું નામ જોડે છે.

ચલનાં કેટલાંક ઉદાહરણ નીચે આપેલાં છે :

int marks;

double amount, interest;

float rate;

char grade;

boolean isPass;

ચલનું નામ વ્યાખ્યાયિત કરવા માટે આપણે નીચે જણાવેલા કેટલાક નિયમોને અનુસરવાની જરૂર છે :

- ચલના નામની શરૂઆત કોઈ મૂળાક્ષર, અન્ડરસ્કોર (_) અથવા ડૉલરના ચિહ્ન ($)થી કરવી જોઈએ. પહેલા અક્ષર પછી તેમાં અંક, મૂળાક્ષર, $ અને અન્ડરસ્કોર હોઈ શકે.

  કેટલાક માન્ય ચલનાં નામ : birth_date, result, CallCost, top5students, date, amount$, $price

  કેટલાક અમાન્ય ચલનાં નામ : 4me, %discount, birth date

- ચલનાં નામમાં વચ્ચે ખાલી જગ્યા (space) માન્ય નથી. આથી, birth date એ માન્ય ચલનું નામ નથી.
- ચલ તરીકે જાવાનો આરક્ષિત શબ્દ (reserved word) ન હોઈ શકે. આરક્ષિત શબ્દનો જાવામાં વિશિષ્ટ ઉપયોગ હોય છે અને પ્રોગ્રામર દ્વારા અન્ય હેતુ માટે તેનો ઉપયોગ કરી શકાતો નથી. કેટલાક આરક્ષિત શબ્દનાં ઉદાહરણ class, public, static અને while છે.

### ચલનું નામ આપવાની માર્ગદર્શિકા

- ચલનું નામ અર્થપૂર્ણ પસંદ કરો. જાવા ચલનાં નામ કોઈ પણ લંબાઈનાં હોઈ શકે છે.
- જ્યારે ચલનાં નામમાં વધારે શબ્દો હોય જેમકે 'balance amount' તો નીચે જણાવેલામાંથી કોઈ પણ એક પદ્ધતિને અનુસરો :
  - પહેલા શબ્દ સિવાયના દરેક શબ્દનો પહેલો અક્ષર કેપિટલ રાખો. આ રીતને અમુક સમયે કેમલકેસ (camel case) કહેવામાં આવે છે કારણકે નામની વચ્ચેના કેપિટલ અક્ષરો ઊંટની ખૂંધ જેવા લાગે છે. ઉદાહરણ : balanceAmount, birthDate

- શબ્દોને અન્ડરસ્કોરથી છૂટા પાડો. ઉદાહરણ : balance_amount, birth_date

- યાદ રાખો કે જાવા કેસ-સેન્સિટિવ (case-sensitive) છે. આથી કેપિટલ અને સ્મોલ અક્ષરો જુદા ગણવામાં આવે છે. આથી ચલનાં નામ balance અને Balance જુદા છે.

- પ્રણાલિકાગત, ક્લાસ (class)નાં નામ કેપિટલ (upper case)થી શરૂ કરવામાં આવે છે, જ્યારે ચલનાં અને મેથડનાં નામ સ્મોલ (lower case)થી શરૂ કરવામાં આવે છે.

ચલને ઘોષિત કરવા માટેની સારી પ્રોગ્રામિંગની શૈલી નીચે મુજબ છે :

- ચલને ઘોષિત કરવાના વિધાનમાં ફક્ત એક જ ચલ આપો.

- દરેક ચલને ઘોષિત કરવા સાથે પ્રોગ્રામમાં તેના હેતુની સમજ માટે કોમેન્ટનો સમાવેશ કરો.

- અગત્યના ચલ ફંક્શનની શરૂઆતમાં ઘોષિત કરો. જે ચલ ફંક્શનના સમગ્ર તર્ક માટે અગત્યના ન હોય, તેને તે પહેલી વખત વપરાય ત્યારે ઘોષિત કરો.

જાવામાં ત્રણ પ્રકારના ચલ હોય છે : (1) ઇન્સ્ટન્સ વેરિયેબલ (Instance variable) (2) ક્લાસ વેરિયેબલ (class variable) અને (3) લોકલ વેરિયેબલ (local variable). ફંક્શનમાં ઘોષિત કરેલા ફંક્શન પેરામીટર (વિધેય પ્રાચલો) અને વેરિયેબલ (ચલ) એ લોકલ વેરિયેબલ છે. આપણે ઇન્સ્ટન્સ વેરિયેબલ અને ક્લાસ વેરિયેબલ વિશે પછી અભ્યાસ કરીશું.

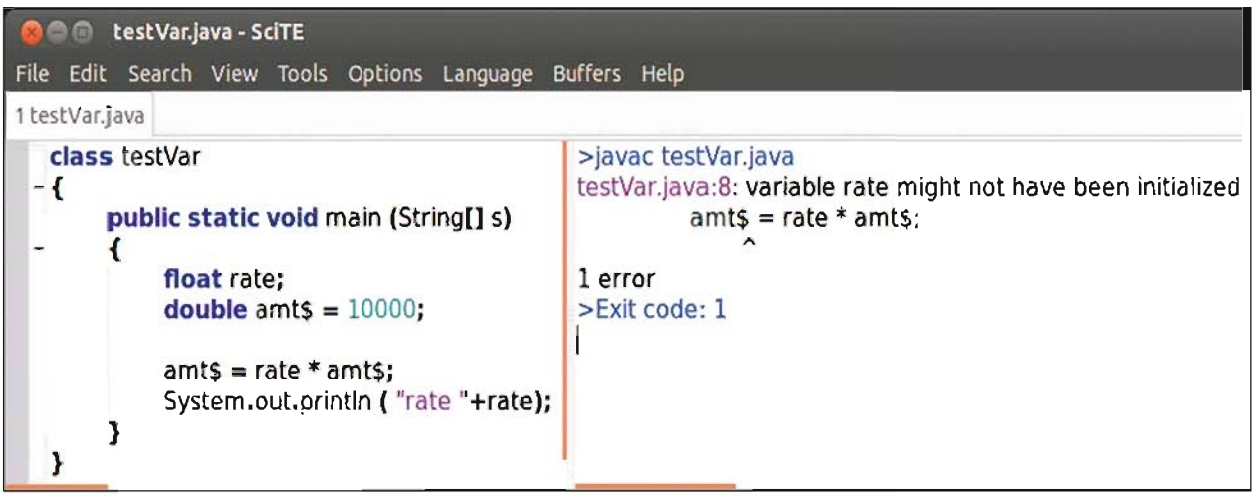
```
int marksObtained, totalMarks=100, counter=0;

boolean isPass = true;
```

સામાન્ય રીતે ચલને ઘોષિત કરવાની વાક્યરચના નીચે પ્રમાણે હોય છે :

<type name> {variable [= <value>,]};

અહીં ચોરસ કૌંસ [ ]માં દર્શાવેલ ઘટક વૈકલ્પિક હોય છે.

અહીં નોંધ કરો કે લોકલ વેરિયેબલને પૂર્વનિર્ધારિત કિંમત સાથે આરંભિક કિંમત આપી શકતા નથી. આવા ચલનો ઉપયોગ કરતાં પહેલા કોઈ કિંમત આપવાની જવાબદારી પ્રોગ્રામરની રહે છે. આકૃતિ 7.6માં જાવાપ્રોગ્રામમાં ચલનો ઉપયોગ દર્શાવ્યો છે.




આકૃતિ 7.6 : જાવાપ્રોગ્રામમાં લોકલ વેરિયેબલ

અહીં તમે જોઈ શકશો કે 'amt$ = rate * amt$' વિધાન વાપરતાં પહેલાં લોકલ વેરિયેબલ 'rate'ને કોઈ કિંમત આપી નથી. જ્યારે આપણે આ પ્રોગ્રામને કંપાઇલ કરીશું, ત્યારે આકૃતિ 7.6માં દર્શાવ્યા પ્રમાણે તે પ્રોગ્રામમાં ભૂલ (error) દર્શાવશે.

## લિટરલ (Literals)

અચળ કિંમત માટે વાપરવામાં આવતું નામ લિટરલ તરીકે ઓળખાય છે. જાવામાં સંખ્યા (number), અક્ષર (character), સ્ટ્રિંગ (string - જે શાબ્દિક લખાણ છે) અને બુલિયન કિંમતો માટે અલગ-અલગ પ્રકારનાં લિટરલ હોય છે.

## ન્યુમરિક લિટરલ (Numeric Literals)

પૂર્ણાંક અને અપૂર્ણાંક સંખ્યાઓ જણાવવા માટે ન્યુમરિક લિટરલ વપરાય છે. ઉદાહરણ : 157 અને 17.42 ન્યુમરિક લિટરલ છે.

**ઇન્ટિજર લિટરલ (Integer literals)** એ લિટરલ છે, જે પૂર્ણ સંખ્યા (whole number) છે. જાવામાં દશાંશ (base 10), અષ્ટાંકી (base-8) અને સોળ-અંકી (base-16) અને યુનિકોડ (unicode) ઇન્ટિજર લિટરલ છે.

સામાન્ય પૂર્ણાંક સંખ્યાઓ જેમકે 4, 157, 17777 અને -32 એ તેના કદ પ્રમાણે byte, short અથવા int ડેસિમલ ઇન્ટિજર લિટરલ છે. int કરતાં મોટા ડેસિમલ ઇન્ટિજર લિટરલ આપોઆપ long પ્રકારના બની જાય છે. આપણે નાની સંખ્યાને સંખ્યાની પાછળ L અથવા l લખીને long બનાવી શકીએ છીએ. (દા.ત. 4L એ long integer છે, જેની કિંમત 4 છે). ઋણ પૂર્ણાંક સંખ્યાઓ આગળ ઋણનું ચિહ્ન (-) લખવામાં આવે છે, દા.ત. -45.

અષ્ટાંકી સંખ્યા ફક્ત 0થી 7 અંકનો ઉપયોગ કરે છે. જાવામાં ન્યુમરિક લિટરલ આગળ 0 (શૂન્ય) લખવાથી તે સંખ્યા અષ્ટાંકી ગણવામાં આવે છે. ઉદાહરણ તરીકે, 045 લિટરલ એ અષ્ટાંકી પૂર્ણાંક છે, જેની દશાંશકિંમત 37 છે.

સોળ-અંકી સંખ્યા 16 અંકનો ઉપયોગ કરે છે, જેમાં 0થી 9 અંક અને અક્ષર A, B, C, D, E અને F હોય છે. આ સંદર્ભમાં કેપિટલ અક્ષર કે નાના અક્ષર અદલબદલ કરી શકાય છે. અક્ષર Aથી F અનુક્રમે 10 થી 15 સંખ્યા રજૂ કરે છે. જાવામાં સોળ-અંકી લિટરલ લખવા માટે સંખ્યાની શરૂઆત 0x અથવા OXથી થાય છે. સોળ-અંકી લિટરલનાં ઉદાહરણ 0x45 અથવા 0xFF7A છે. કેરેક્ટર લિટરલમાં સોળ-અંકી સંખ્યા પણ મનસ્વી (arbitrary) યુનિકોડ અક્ષર રજૂ કરવા વપરાય છે.

યુનિકોડ લિટરલ \u પછી ચાર સોળ-અંકી અંક વડે બનેલો હોય છે. ઉદાહરણ તરીકે, '\u00E9' કેરેક્ટર લિટરલ છે, જે તીવ્ર ઉચ્ચારણના સ્વરભાર સાથેનો "e" યુનિકોડ અક્ષર છે.

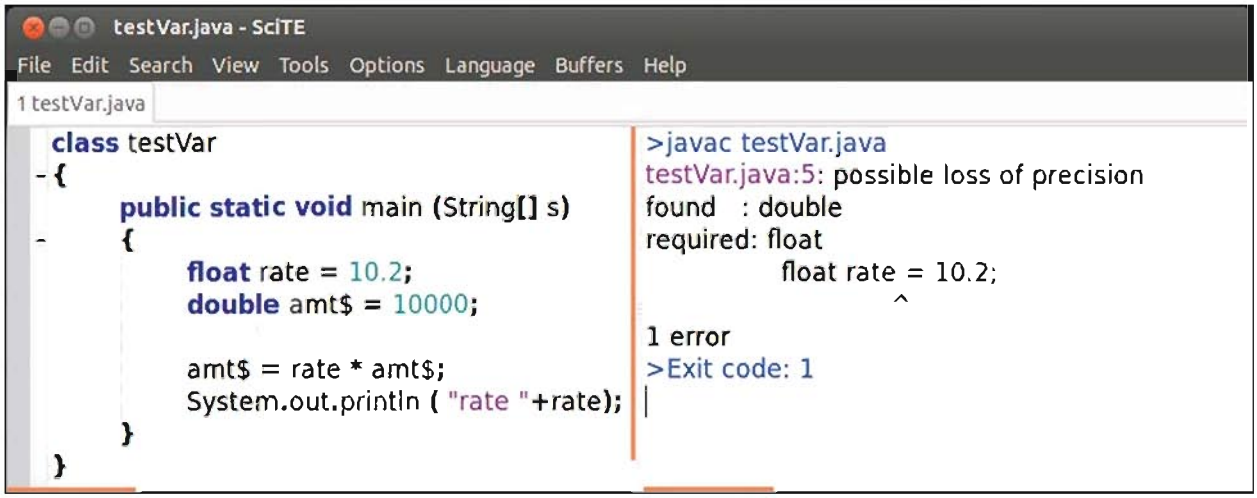
**જાવા 7** દ્વિઅંકી સંખ્યા, જેમાં 0 અને 1 અંક વપરાય છે, ને રજૂ કરવા માટે સંખ્યાની પહેલા 0b (અથવા 0B) લખવામાં આવે છે. ઉદાહરણ : 0b10110 અથવા 0b101011001011.

રિયલ નંબર લિટરલને ફ્લોટિંગ-પોઇન્ટ લિટરલ (Floating Point Literals) પણ કહેવામાં આવે છે. આ સંખ્યા બે પ્રકારની સંકેતલિપિ વડે લખી શકાય છે, પ્રમાણભૂત (standard) અને વૈજ્ઞાનિક (scientific).

પ્રમાણભૂત સંકેતલિપિમાં પૂર્ણાંક ભાગ અને અપૂર્ણાંક ભાગને દશાંશચિહ્ન (.)થી અલગ કરવામાં આવે છે. ઉદાહરણ તરીકે 12.37.

વૈજ્ઞાનિક સંકેતલિપિમાં સંખ્યા પછી અક્ષર e (અથવા E) અને ચિહ્નિત પૂર્ણાંક ઘાતાંક (Signed Integer Exponent) લખવામાં આવે છે. ઉદાહરણ તરીકે 1.3e12 અને 12.3737e-108. અહીં "e12" અને "e-108" એ 10નો ઘાતાંક રજૂ કરે છે. આથી, 1.3e12 એટલે 1.3 વખત$10^{12}$ અને 12.3737e-108 એટલે 12.3737 વખત $10^{-108}$. વૈજ્ઞાનિક માળખું અતિ મોટી સંખ્યા અને ખૂબ જ નાની સંખ્યા જણાવવા માટે વાપરી શકાય.

જાવામાં ફ્લોટિંગ-પોઇન્ટ લિટરલનો પૂર્વનિર્ધારિત પ્રકાર double છે. લિટરલનો પ્રકાર float કરવા માટે સંખ્યાની પાછળ "F" અથવા "f" ઉમેરવો પડે છે. ઉદાહરણ તરીકે, "1.2F" નો અર્થ 1.2 લિટરલનો પ્રકાર float છે.




આકૃતિ 7.7 : કંપાઈલેશન એરર

આકૃતિ 7.7નું નિરીક્ષણ કરો જેમાં આપણે float પ્રકારના ચલને લિટરલ 10.2ની કિંમત આપેલી છે, જેથી આપણે કંપાઈલેશન એરર મેળવીએ છીએ. આ પ્રોગ્રામ સફળતાપૂર્વક ચલાવવા માટે આપણે એ વિધાનને "float rate = 10.2f" લખવું પડે. સંખ્યાના અંતમાં લખેલ પ્રત્યય F પહેલી કે બીજી એબીસીડીમાં લખી શકાય છે.

### બુલિયન લિટરલ (Boolean Literals)

બુલિયન પ્રકાર માટે સ્પષ્ટ રીતે બે લિટરલ છે : true અને false. આ લિટરલ અવતરણ વિના ટાઈપ કરવામાં આવે છે. તે ચલ નથી પણ કિંમત છે. બુલિયન કિંમતો મોટે ભાગે શરતી પદાવલીઓમાં જોવા મળે છે. c ભાષામાં 0ને false તરીકે અને બિન-શૂન્ય કિંમતને true તરીકે ગણવામાં આવે છે. જાવામાં true અને false લિટરલ કોઈ આંકડાકીય કિંમત સાથે સંકળાયેલ નથી.

### કેરેક્ટર લિટરલ (Character Literals)

કેરેક્ટર લિટરલને એક અવતરણચિહ્ન વચ્ચે સમાવાયેલા એક અક્ષર દ્વારા જણાવવામાં આવે છે. ઉદાહરણ : 'a', '#' અને '3'. અક્ષરને 16 બીટ યુનિકોડ કેરેક્ટર તરીકે સંગ્રહ કરવામાં આવે છે. અમુક વિશિષ્ટ અક્ષરો માટે વિશેષ લિટરલ છે જે "escape character" તરીકે બૅક્સ્લેશ (\)નો ઉપયોગ કરે છે. કોષ્ટક 7.2માં વિશેષ કોડ કે જેના દ્વારા છાપી ન શકાય તેવા અક્ષરો તથા યુનિકોડ કેરેક્ટર સેટમાં આવેલા અક્ષરોની યાદી આપવામાં આવી છે.

| Escape code | Meaning |
|---|---|
| \n | New line |
| \t | Tab |
| \b | Backspace |
| \r | Carriage return |
| \f | Form feed (New page) |
| \\ | Back slash character |
| \' | Single quote character |
| \" | Double quote character |
| \ddd | Character represented by three octal digits ( d: 0 to 7) |
| \xdd | Character represented by two hexadecimal digits (d: 0 to 9, a to f) |
| \udddd | Character represented by Unicode number dddd (d: hexadecimal digit) |

કોષ્ટક 7.2 : એસ્કેપકોડ

## સ્ટ્રિંગ લિટરલ (String Literals)

જાવામાં અન્ય બધા ડેટાપ્રકારો "primitive" ડેટાને બદલે ઑબ્જેક્ટ રૂપમાં છે. આપણે હમણાં થોડા સમય માટે વિશે વિચારતા નથી. જોકે અહીં આપણે પૂર્વવ્યાખ્યાયિત ઑબ્જેક્ટ કે જે ઘણો અગત્યનો છે, તેના વિશે વિચારીશું, જે સ્ટ્રિંગ (string) પ્રકાર છે. સ્ટ્રિંગ એ અક્ષરોની એક શ્રેણી છે.

આપણે કોડલિસ્ટિંગ 7.1માં "Balance:" સ્ટ્રિંગ લિટરલ વાપર્યો છે. સ્ટ્રિંગ લિટરલ એ બે અવતરણચિહ્ન વચ્ચે અક્ષરોની એક શ્રેણી છે. સ્ટ્રિંગની અંદર વિશિષ્ટ અક્ષરો કોષ્ટક 7.2માં દર્શાવ્યા પ્રમાણે ઊંધા સ્લેશચિહ્ન વાપરીને રજૂ કરી શકાય છે.

ઉદાહરણ તરીકે, Many "Congratulations!" સ્ટ્રિંગ આપવા માટે આપણે સ્ટ્રિંગ લિટરલ "Many, \"Congratulations!\"" ટાઇપ કરવું પડે.

આ જ પ્રમાણે, "This string brought to you by Java\u2122" સ્ટ્રિંગમાં યુનિકોડની શ્રેણી \u2122 ટ્રેડમાર્કનું ચિહ્ન (™) બનાવશે.

## કૉમેન્ટ (Comments)

પ્રોગ્રામમાં કૉમેન્ટ (comment) ફક્ત માનવ-વાચકો માટે જ છે. તેની કમ્પ્યૂટર દ્વારા સંપૂર્ણપણે અવગણના થાય છે. દરેક વ્યક્તિ દ્વારા પ્રોગ્રામ સમજવો સરળ બનાવવા માટે કૉમેન્ટ લખવી અગત્યની છે.

જાવામાં કૉમેન્ટના નીચે લખેલ પ્રકાર હોય છે :

- **એક-લીટી કૉમેન્ટ :** તેની શરૂઆત ડબલસ્લેશ (//) થી થાય છે અને તે લીટીના અંત સુધી વિસ્તૃત થાય છે. કમ્પ્યૂટર // અને તેના પછી એ જ લીટીના તમામ લખાણની અવગણના કરે છે.
- **બહુ-લીટી કૉમેન્ટ :** તેની શરૂઆત /* થી અને અંત */ થી થાય છે. આ પ્રકારની કૉમેન્ટનો ઉપયોગ સામાન્ય રીતે એક કરતાં વધારે લીટીની કૉમેન્ટ હોય ત્યારે થાય છે. હકીકતમાં /* અને */ વચ્ચે શબ્દસમૂહ કે એક અથવા વધારે લીટીની કૉમેન્ટ કોઈ પણ રાખી શકે છે. બે સીમારેખાઓ /* અને */ વચ્ચેનું તમામ લખાણ કૉમેન્ટ ગણવામાં આવે છે. કૉમેન્ટનું નેસ્ટિંગ (nesting) કરી શકાતું નથી, એટલેકે એક કૉમેન્ટની અંદર બીજી કૉમેન્ટ ન હોઈ શકે.
- **દસ્તાવેજીકરણ કૉમેન્ટ :** આ પ્રકારની કૉમેન્ટ /** થી શરૂ થાય છે અને */ થી તેનો અંત આવે છે. તેનો ઉપયોગ કોડમાંથી API ડોક્યુમેન્ટેશન બનાવવા માટે થાય છે. આ વિશિષ્ટ પ્રકારની કૉમેન્ટ છે, જે javadoc સિસ્ટમ માટે વપરાય છે. javadocની ચર્ચા આ પુસ્તકની મર્યાદાબહાર છે.

પ્રોગ્રામમાં કૉમેન્ટ સિવાય તમામે જાવાની વાક્યરચના (syntax)ને અનુસરવું પડે છે.

## પદાવલી (Expressions)

પદાવલીઓ પ્રોગ્રામિંગના અનિવાર્ય ભાગ છે. પદાવલીના પાયાના ઘટકો લિટરલ (જેવા કે 674, 3.14, true અને 'X'), ચલ અને ફંક્શન કૉલ છે. યાદ રહે કે વિધેય એ એક સબરૂટિન છે, જે કિંમત પરત કરે છે.

સાદી પદાવલી એક લિટરલ, ચલ કે ફંક્શનકોલ હોઈ શકે. વધારે જટિલ પદાવલીઓ સાદી પદાવલીઓને પ્રક્રિયકો વાપરી ભેગા કરીને બનાવી શકાય છે.

પ્રક્રિયકોમાં (operators) ગાણિતિક પ્રક્રિયકો ( +, -, *, /, %), સરખામણી-પ્રક્રિયકો (<, >, =, ...), તાર્કિક પ્રક્રિયકો (and, or, not...)નો સમાવેશ થાય છે. જ્યારે પદાવલીમાં થોડા પ્રક્રિયકો હોય, ત્યારે અગ્રતાક્રમ (precedence)નો પ્રશ્ન થાય છે કે ગણતરી કરવા માટે કઈ રીતે પ્રક્રિયકોનાં જૂથ બનાવવાં. ઉદાહરણ તરીકે, "A + B * C" પદાવલીમાં પહેલાં B*Cની ગણતરી કરવામાં આવે છે અને તેનું પરિણામ Aમાં ઉમેરવામાં આવે છે. આપણે અહીં સરવાળા (+) કરતાં ગુણાકાર (*) ને વધારે અગ્રતાક્રમ આપીએ છીએ. જો આપણે ઇચ્છતા હોય તે પૂર્વનિર્ધારિત અગ્રતાક્રમ ન હોય, તો આપણે જૂથ સ્પષ્ટ રીતે જણાવવા માટે કૌંસનો ઉપયોગ કરી શકીએ. ઉદાહરણ તરીકે,

જો આપણે "(A + B) * C" પદાવલીની કિંમત શોધવી હોય, તો પહેલાં Aને Bનો સરવાળો કરી તેના પરિણામનો C સાથે ગુણાકાર કરવામાં આવે.

### પ્રક્રિયક (Operators)

પદાવલી બનાવવા માટે વપરાતાં વિશિષ્ટ ચિહ્નો પ્રક્રિયક છે. જાવા અનેક પ્રકારના પ્રક્રિયક પૂરા પાડે છે. આ પ્રકરણમાં આપણે નીચે જણાવેલા પ્રક્રિયકોની ચર્ચા કરીશું :

- અંકગણિતીય પ્રક્રિયક (Arithmetic operators)
- તુલનાત્મક પ્રક્રિયક (Comparison operators)
- તાર્કિક પ્રક્રિયક (Logical operators)
- શરતી પ્રક્રિયક (Conditional operator)
- એસાઇન્મેન્ટ પ્રક્રિયક (Assignment operator)

### અંકગણિતીય પ્રક્રિયક (Arithmetic Operators)

જાવામાં મૂળભૂત અંકગણિતીય પ્રક્રિયક સરવાળો (+), બાદબાકી (–), ગુણાકાર (*), ભાગાકાર (/) અને મૉડ્યુલસ (%) છે. આ બધા પ્રક્રિયક બાયનરી છે, તેમાં બે સંકાર્ય (operand) હોય છે. પ્રક્રિયક + અને – યુનરી (unary) (ફક્ત એક જ સંકાર્ય સાથે) તરીકે પણ વપરાય છે. આ બધા જ પ્રક્રિયકો તમામ પ્રકારના આંકડાકીય ડેટા ઉપર લાગુ કરી શકાય છે, જેમકે byte, short, int, long, float અને double. તે char પ્રકારની કિંમત સાથે પણ વાપરી શકાય છે. આ સંદર્ભમાં તે પૂર્ણાંક સંખ્યા તરીકે ગણવામાં આવે છે. જ્યારે char પ્રકારના ડેટા હોય, ત્યારે તેની કિંમત unicode સંખ્યા પ્રમાણે ગણવામાં આવે છે, જ્યારે તે અંકગણિતીય પ્રક્રિયક સાથે વાપરવામાં આવે. કોષ્ટક 7.3માં અંકગણિતીય પ્રક્રિયકોનું વર્ણન આપેલું છે :

| પ્રક્રિયક | અર્થ | ઉદાહરણ | પરિણામ |
|---|---|---|---|
| + | સરવાળો | 2 + 8 | 10 |
| - | બાદબાકી | 2 - 8 | -6 |
| * | ગુણાકાર | 2 * 8 | 16 |
| / | ભાગાકાર | 8 / 2 | 4 |
| % | મૉડ્યુલસ<br>(ભાગાકાર પછી શેષ જણાવે છે, ભાગફળ પૂર્ણાંક સંખ્યા હોય છે.) | 8 % 3<br>25.8 % 7 | 2<br>4.8 |

કોષ્ટક 7.3 : અંકગણિતીય પ્રક્રિયકો

બાયનરી અંકગણિતીય પ્રક્રિયા પછી પરિણામનો ડેટાપ્રકાર નીચે પ્રમાણે હોય છે :

- જો બન્ને સંકાર્ય (operands) એક જ પ્રકારના ડેટા હોય, તો પરિણામનો ડેટાપ્રકાર પણ સંકાર્યના પ્રકાર સમાન જ રહેશે.
  - જો બન્ને સંકાર્ય પૂર્ણાંક હશે તો પરિણામ પૂર્ણાંક રહેશે. ઉદાહરણ તરીકે 9/2નું પરિણામ 4.5 નહીં, પણ 4 છે. પૂર્ણાંક સંખ્યાના ભાગાકારનું પરિણામ પણ પૂર્ણાંક કિંમત રહેશે અને શેષ છોડી દેવામાં આવે છે.
  - જો બંને સંકાર્ય અપૂર્ણાંક હોય, તો પરિણામ અપૂર્ણાંક રહેશે. ઉદાહરણ તરીકે, 9f/2f નું પરિણામ 4.5 છે.

- જ્યારે બન્ને સંકાર્ય અલગ-અલગ પ્રકારના ડેટા હોય, ત્યારે :
  - સૌપ્રથમ જે ડેટાપ્રકારની નાની રેન્જ છે, તેને મોટી રેન્જના ડેટા પ્રકારમાં બદલી નાખવામાં આવે છે, જેથી બન્ને સંકાર્યનો સમાન પ્રકાર બની રહે. આ પ્રકારના પરિવર્તનને પ્રમોશન (promotion) પણ કહેવામાં આવે છે.
  - હવે, પદાવલીનું પરિણામ મોટી રેન્જના સંકાર્ય સમાન બની રહેશે.
  - ઉદાહરણ : 4 + 3.5નું પરિણામ 7.5 મળશે. 9/2.0નું પરિણામ 4.5 મળશે અને 9f/2નું પરિણામ 4.5 મળશે.

**યાદ રાખવા જેવા મુદ્દાઓ :**

- મોડ્યુલસ પ્રક્રિયક %ના કિસ્સામાં, જો પહેલો સંકાર્ય ઋણ હોય તો, પરિણામ ઋણ છે.
- જાવામાં % પ્રક્રિયક સાથે અપૂર્ણાંક ડેટાપ્રકાર પણ વાપરી શકીએ છીએ. પરિણામ પૂર્ણાંક ભાગફળ પછી રહેલી શેષ છે. આથી, 25.8 % 7ના જવાબ માટે પૂર્ણાંક ભાગફળ 3 છે અને શેષ 25.8 – (3*7) = 4.8 છે.
- પ્રક્રિયક +નો ઉપયોગ શાબ્દિક લખાણ (string)ને જોડવા (concatenate) માટે પણ કરી શકાય છે.

જ્યારે બે સંકાર્યમાંથી એકનો પ્રકાર string હોય અને તેના ઉપર + પ્રક્રિયક લાગુ કરવામાં આવે, ત્યારે બીજા સંકાર્યનો ડેટા-પ્રકાર આપોઆપ string બની જાય છે. આ પણ ગર્ભિત (implicit) પ્રકારનું રૂપાંતર છે. આ પ્રકારનું રૂપાંતર આપણે balance રકમ છાપતા સમયે '"Balance :" + balance' પદાવલીમાં કોડલિસ્ટિંગ 7.1માં જોઈ શકીએ છીએ. આકૃતિ 7.8માં દર્શાવેલા કોડલિસ્ટિંગમાં પણ તેનો ઉપયોગ કરેલો છે.

ArithOperators.java - SciTE

File Edit Search View Tools Options Language Buffers Help

1 ArithOperators.java

```
class ArithOperators
{
    public static void main (String[] args)
    {
        short x = 6;
        int y = 4;
        float a = 12.5f;  // suffix f to explicitly specify floating-point literal
        float b = 7.2f;   // suffix f for floating-point literal

        System.out.println ("x is " + x + ", y is " + y);
        System.out.println ("x + y = " + (x + y));
        System.out.println ("x - y = " + (x - y));
        System.out.println ("x / y = " + (x / y));
        System.out.println ("x % y = " + (x % y));
        x = -6;
        System.out.println ("x % y = " + (x % y));
        y=-4;
        System.out.println ("x % y = " + (x % y));
        x=6; y=-4;
        System.out.println ("x % y = " + (x % y));

        System.out.println ("a is " + a + ", b is " + b);
        System.out.println ("a / b = " + (a / b));
        System.out.println ("a / x = " + (a / x));
        System.out.println ("a % x = " + (a%x));
        System.out.println ("a % b = " + (a%b));
    } // end main()
} // end class ArithOperators
```

```
>javac ArithOperators.java
>Exit code: 0
>java -cp . ArithOperators
x is 6, y is 4
x + y = 10
x - y = 2
x / y = 1
x % y = 2
x % y = -2
x % y = -2
x % y = 2
a is 12.5, b is 7.2
a / b = 1.7361112
a / x = 2.0833333
a % x = 0.5
a % b = 5.3
>Exit code: 0
```

**આકૃતિ 7.8 : અંકગણિતીય પ્રક્રિયકોનો ઉપયોગ દર્શાવતો જાવાપ્રોગ્રામ**

### આકૃતિ 7.8માં દર્શાવેલા પ્રોગ્રામની સમજૂતી (Explanation of Program Shown in Figure 7.8)

- આપણે main() મેથડની શરૂઆતમાં ચાર ચલ વ્યાખ્યાયિત કરીએ છીએ. જેમાં x અને y એ પૂર્ણાંક પ્રકારના (short અને int પ્રકાર) અને a તથા b અપૂર્ણાંક પ્રકારના (float પ્રકાર) છે.

- તમને યાદ હશે કે અપૂર્ણાંક પ્રકારના લિટરલ (જેમકે 12.5)નો પૂર્વનિર્ધારિત પ્રકાર double હોય છે. આથી તેનો પ્રકાર float તરીકે ગણવા માટે લિટરલ 12.5 અને 7ના અંતમાં (suffix) f લખવામાં આવેલ છે.
- system.out.println() મેથડ પ્રમાણભૂત નિર્ગમ એકમ ઉપર માત્ર સંદેશો છાપે છે. આ મેથડમાં માત્ર એક જ આર્ગ્યુમેન્ટ, એક સ્ટ્રિંગ, હોય છે, પણ આપણે આ શાબ્દિક લખાણ સાથે જોડવા માટે +નો ઉપયોગ કરી શકીએ છીએ. પહેલાં જણાવ્યું હતું તે પ્રમાણે જાવા સંખ્યાનું સ્ટ્રિંગમાં રૂપાંતર કરે છે અને પછી તે બંનેને જોડે છે.
- જ્યારે x=6 અને y=–4 હોય, ત્યારે x%yનું પરિણામ 2 મળે છે, કારણકે પ્રથમ સંકાર્ય ધન છે.
- જ્યારે a=12.5 અને x=6 હોય ત્યારે a%xનું પરિણામ 0.5 મળે છે. અહીં ભાગાકારનું પૂર્ણાંક ભાગફળ 2 અને શેષ 0.5 છે. એ જ રીતે જો a=12.5 અને b=7.2 હોય, તો a%bમાં પૂર્ણાંક ભાગફળ 1 અને શેષ 5.3 છે.
- System.out.println ("a / x = " + (a / x)); વિધાનમાં મિશ્રિત પ્રકારનાં સંકાર્ય જુઓ. અહીં, ચલ a એ float છે અને x ચલ int છે, ભાગાકાર પછી તેનું પરિણામ float છે. પદાવલીની ગણતરી સમયે કિંમતોને ઉચ્ચ ડેટા-પ્રકાર સંકાર્યમાં રૂપાંતર કરવામાં આવેલ છે.

એ જ પ્રોગ્રામના System.out.println મેથડના બીજા callમાં રહેલી પદાવલી x+yના કૌંસને દૂર કરીને પ્રયોગ કરી જુઓ, એટલે કે આકૃતિ 7.8ના કોડલિસ્ટિંગની 11મી લીટીમાં રહેલી પદાવલી "x + y = " + (x + y) ને પદાવલી "x + y = " + x + y વડે બદલો અને આકૃતિ 7.9માં દર્શાવેલા પરિણામનું વિશ્લેષણ કરો.

ArithOperators.Java - SciTE

File Edit Search View Tools Options Language Buffers Help

1 ArithOperators.java

```
class ArithOperators
{
    public static void main (String[] args)
    {
        short x = 6;
        int y = 4;
        float a = 12.5f;  // suffix f to explicitly specify floating-point literal
        float b = 7.2f;   // suffix f for floating-point literal

        System.out.println ("x is " + x + ", y is " + y);
        System.out.println ("x + y = " + x + y);
        System.out.println ("x - y = " + (x - y));
        System.out.println ("x / y = " + (x / y));
        System.out.println ("x % y = " + (x % y));
        x = -6;
        System.out.println ("x % y = " + (x % y));
        y=-4;
        System.out.println ("x % y = " + (x % y));
        x=6; y=-4;
        System.out.println ("x % y = " + (x % y));

        System.out.println ("a is " + a + ", b is " + b);
        System.out.println ("a / b = " + (a / b));
        System.out.println ("a / x = " + (a / x));
        System.out.println ("a % x = " + (a%x));
        System.out.println ("a % b = " + (a%b));
    } // end main()
} // end class ArithOperators
```

```
>javac ArithOperators.java
>Exit code: 0
>java -cp . ArithOperators
x is 6, y is 4
x + y = 64
x - y = 2
x / y = 1
x % y = 2
x % y = -2
x % y = -2
x % y = 2
a is 12.5, b is 7.2
a / b = 1.7361112
a / x = 2.0833333
a % x = 0.5
a % b = 5.3
>Exit code: 0
```

**આકૃતિ 7.9 : આકૃતિ 7.8ના કોડલિસ્ટિંગની 11મી લીટીમાં રહેલ (x+y)માંથી કૌંસ ( ) કાઢી નાખવાની અસર**

## ઇન્ક્રિમેન્ટ અને ડિક્રિમેન્ટ પ્રક્રિયક (Increment and Decrement Operators)

એકપદી પ્રક્રિયક ++ અને --ને અનુક્રમે વૃદ્ધિ પ્રક્રિયક (ઇન્ક્રિમેન્ટ ઑપરેટર) અને હ્રાસપ્રક્રિયક (ડિક્રિમેન્ટ ઑપરેટર) કહેવામાં આવે છે. ++ પ્રક્રિયક ચલમાં 1 ઉમેરે છે અને -- પ્રક્રિયક ચલમાંથી 1 બાદ કરે છે.

આ પ્રક્રિયકો કોઈ પણ પ્રકારના પૂર્ણાંક ચલ અને char પ્રકારના ચલ ઉપર પણ વાપરી શકાય છે. જો x એ પૂર્ણાંક ચલ હોય, તો આપણે x++, ++x, x--, --x નો ઉપયોગ એક પદાવલી તરીકે અથવા કોઈ મોટી પદાવલીના ભાગ તરીકે કરી શકીએ છીએ.

જ્યારે ચલના નામ પાછળ ++ અથવા -- પ્રક્રિયકનો ઉપયોગ કરવામાં આવે, ત્યારે તેને પોસ્ટ-ઇન્ક્રિમેન્ટ અથવા પોસ્ટ-ડિક્રિમેન્ટ કહેવાય છે. આવા સંજોગોમાં પદાવલીની ગણતરી કરતાં સમયે ચલની જૂની કિંમત વપરાય છે અને તે પછી ચલની કિંમત વધારવામાં કે ઘટાડવામાં આવે છે. ઉદાહરણ તરીકે, ધારો કે y=4+x++; વિધાનના અમલ પહેલાં ચલ xની કિંમત 3 છે. અહીં પદાવલીની જમણી બાજુની ગણતરી કરતાં સમયે xની જૂની કિંમત વપરાશે અને તે પછી xની કિંમતની વૃદ્ધિ (increment) કરવામાં આવશે. આથી, આપણને પરિણામ xની કિંમત 4 અને yની કિંમત 7 મળશે.

જ્યારે ચલનાં નામ પહેલાં ++ અથવા -- પ્રક્રિયકનો ઉપયોગ કરવામાં આવે, ત્યારે તેને પ્રી-ઇન્ક્રિમેન્ટ અથવા પ્રી-ડિક્રિમેન્ટ કહેવાય છે. આ કિસ્સામાં ચલની કિંમત પહેલાં વધારવામાં કે ઘટાડવામાં આવે છે અને તે પછી આ નવી કિંમત પદાવલીની ગણતરીમાં વપરાય છે. ઉદાહરણ તરીકે, y=4+ ++x વિધાનમાં xની જૂની કિંમત 3 છે, તો પહેલા xની કિંમત વધારવામાં આવશે અને આ નવી કિંમતનો ઉપયોગ પદાવલીની જમણી બાજુની ગણતરીમાં થશે. આથી xની કિંમત 4 અને yની કિંમત 8 થશે.

જ્યારે આ પ્રક્રિયક ફક્ત એક જ પદના વિધાનમાં વાપરવામાં આવે, ત્યારે ચલની પહેલાં કે પછીમાં કોઈ તફાવત નથી. ઉદાહરણ તરીકે, વિધાન x++; અને ++x; એ સ્ટેન્ડએલોન સ્ટેટમેન્ટ (standalone statement) છે.

## તુલનાત્મક પ્રક્રિયક (Comparison Operators)

તુલનાત્મક પ્રક્રિયક (comparison operators)ને સંબંધાત્મક પ્રક્રિયક (relational operators) પણ કહેવાય છે. જાવામાં તુલનાત્મક પ્રક્રિયકો ==, !=, <, >, <=, અને >= છે. આ બધા પ્રક્રિયકોનો અર્થ નીચે મુજબ છે :

| | |
|---|---|
| A == B | A અને B "સમાન" છે ? |
| A != B | A અને B "સમાન નથી" ? |
| A < B | B "કરતાં A નાનો" છે ? |
| A > B | B "કરતાં A મોટો" છે ? |
| A <= B | B "કરતાં A નાનો અથવા સમાન" છે ? |
| A >= B | B "કરતાં A મોટો અથવા સમાન" છે ? |

આ પ્રક્રિયકો કોઈ પણ પ્રકારના આંકડાકીય (numeric) પ્રકારોની અને char પ્રકારની કિંમતોની સરખામણી કરવા માટે વપરાય છે. char પ્રકાર માટે તેની યુનિકોડ આંકડાકીય કિંમત સરખામણીમાં વપરાય છે.

તુલનાત્મક પ્રક્રિયકના ઉપયોગથી પદાવલીનું પરિણામ બુલિયન મળે છે, એટલે કે પરિણામ true અથવા false હોય છે. આથી આવી પદાવલીઓ બુલિયન-કિંમત ધરાવતી પદાવલી (boolean-valued expression) પણ કહેવાય છે.

આપણે બુલિયન-કિંમત ધરાવતી પદાવલીઓ કોઈ બુલિયન ચલને પણ આપી શકીએ (assign), જે રીતે કોઈ સંખ્યા આંકડાકીય ચલને (numeric variable).

== અને != પ્રક્રિયકો બુલિયન-કિંમતોની સરખામણી કરવા માટે થાય છે. ઉદાહરણ તરીકે,

```
boolean bothPositive;  bothPositive = ((x > 0) == (y > 0));
```

સામાન્ય રીતે, તુલનાત્મક પ્રક્રિયકોનો ઉપયોગ if વિધાનો અને લૂપ (loops)માં થાય છે.

### તાર્કિક પ્રક્રિયકો (Logical Operators)

તાર્કિક પ્રક્રિયકો બુલિયન ઑપરેટર (boolean operators) પણ કહેવાય છે, કારણકે તે બુલિયન પ્રકારના સંકાર્ય ઉપર કાર્ય કરે છે. જાવામાં AND, OR, XOR અને NOT જેવાં તાર્કિક કાર્યો કરવા માટે અનુક્રમે &&, ||, ^ અને ! પ્રક્રિયકો વપરાય છે.

તાર્કિક AND કાર્ય માટે બે બુલિયન કિંમતોને જોડવા માટે બુલિયન પ્રક્રિયક && વપરાય છે. પદાવલી A && Bનું પરિણામ true છે, માત્ર જ્યારે બંને સંકાર્ય A અને B true હોય. ઉદાહરણ તરીકે, જો બંને x બરાબર 0 અને y બરાબર 0 હોય તો જ (x == 0) && (y == 0) ની કિંમત true મળે.

તાર્કિક OR માટે બુલિયન પ્રક્રિયક || (બે ઊભી લીટી અક્ષરો) છે. જો Aની કિંમત true હોય અથવા Bની કિંમત true હોય અથવા બન્ને true હોય તો પદાવલી A || Bની કિંમત true મળે છે. જો A અને B બંને false હોય, તો જ પદાવલીનું પરિણામ false મળે. ઉદાહરણ તરીકે, જો x અને y બંને 0 બરાબર ન હોય તો જ (x == 0) && (y == 0)નું પરિણામ false મળે.

તાર્કિક NOT માટે બુલિયન પ્રક્રિયક ! છે અને તે એકપદી (unary) પ્રક્રિયક છે. તેનું પરિણામ પૂરકમાં (complement) પરિણમે છે. જો સંકાર્ય true હોય, તો પરિણામ false મળે અને એથી વિપરીત રીતે પણ.

આ ઉપરાંત, તાર્કિક કાર્ય XOR (exclusive OR) માટે પ્રક્રિયક ^ છે. તે તેના પ્રક્રિયકો અલગ-અલગ હોય, તો જ true કિંમત પરત કરે છે અન્યથા false પરત કરે છે. આથી જ્યારે બંને સંકાર્યની એક સમાન બુલિયન કિંમત હોય ત્યારે પરિણામ false મળે છે. ઉદાહરણ તરીકે, (x == 0) ^ (y == 0) ની કિંમત true મળશે, જો x અથવા yમાંથી ફક્ત એકની કિંમત શૂન્ય હોય.

### શૉર્ટસર્કિટિંગ (Short Circuiting)

જ્યારે શરતી પ્રક્રિયકો AND અથવા OR (&& અને ||) વાપરવામાં આવે છે, ત્યારે જાવા બીજા સંકાર્યની ગણતરી કરશે નહીં, જ્યાં સુધી પરિણામ મેળવવા તે જરૂરી ન હોય. જો &&ના કિસ્સામાં પહેલું સંકાર્ય false હોય, તો બીજા સંકાર્યની ગણતરી કરવાની જરૂર નથી. એ જ પ્રમાણે ||ના કિસ્સામાં જો પહેલું સંકાર્ય true હોય, તો બીજા સંકાર્યની ગણતરી કરવાની જરૂર નથી.

**ઉદાહરણ :** પદાવલી (x != 0) && (y/x > 1) લો. જો xની કિંમત 0 હોય, તો આપેલી પદાવલીના ભાગ (x !=0)ની કિંમત false થશે. આપણે જાણીએ છીએ કે તાર્કિક પ્રક્રિયક &&નું પરિણામ false મળશે જ્યારે કોઈ પણ એક પ્રક્રિયકની કિંમત false હોય, આથી આપેલી પદાવલીના બાકીના ભાગ (y/x > 1)ની ગણતરી કરવાની જરૂર નથી. અહીં ટૂંકો રસ્તો લઈને ગણતરી કરેલી છે અને શૂન્ય વડે ભાગાકાર ટાળેલ છે. શૉર્ટસર્કિટિંગ વિના (ઉપર જણાવેલો ટૂંકો રસ્તો ન લેવાથી), અમલ કરતાં સમયે 'શૂન્ય વડે ભાગાકાર' (Division by Zero) ભૂલ આવી શકે.

### શરતી પ્રક્રિયક (Conditional Operator)

જાવામાં શરતી પ્રક્રિયક ત્રિપદી પ્રક્રિયક (ternary operator) છે જેમાં ત્રણ સંકાર્ય હોય છે. તે ત્રણ સંકાર્ય જુદા પાડવા માટે પદાવલીમાં બે ચિહ્નો ? અને : નો ઉપયોગ કરે છે. તેનું સ્વરૂપ નીચે પ્રમાણે હોય છે :

*<boolean-expression> ? <expression1> : <expression2>*

અહીં, પ્રથમ સંકાર્ય બુલિયન પદાવલી છે, અને તેની સૌપ્રથમ ગણતરી કરવામાં આવે છે. જો તેની કિંમત true હોય તો આખી પદાવલીની કિંમત એ બીજા સંકાર્ય expression1ની કિંમત સમાન થશે, અન્યથા ત્રીજા સંકાર્ય expression2 સમાન થશે.

**ઉદાહરણ :** વિધાન ધ્યાનમાં લો : next = (N % 2 == 0) ? (N/2) : (3*N+1);

ધારો કે Nની કિંમત 8 છે. પ્રથમ સંકાર્ય (N % 2 == 0) ની કિંમત true છે આથી જમણી બાજુની પદાવલીની કિંમત પદાવલી (N/2)ની કિંમત બરાબર થશે એટલે કે 4 થશે. જો Nની કિંમત એકી સંખ્યા હોય તો પહેલી પદાવલીની કિંમત false થશે અને (3*N+1)ની કિંમત nextને એસાઇન (assign) કરવામાં આવશે. અહીં નોંધ કરો કે આ ઉદાહરણમાં કૌંસની જરૂર નથી પણ તે સમીકરણને વાંચવામાં સરળ બનાવે છે.

### એસાઈન્મેન્ટ (Assignment)

જાવામાં જે પદાવલી એસાઇન્મેન્ટ પ્રક્રિયક (=) ધરાવતી હોય, તેને સામાન્ય રીતે એસાઇન્મેન્ટ સ્ટેટમેન્ટ (assignment statement - સોંપણી કરતું વિધાન) કહેવામાં આવે છે.

આપણે એક વખત ચલ ઘોષિત કરીએ (declare) તે પછી આપણે તેને કિંમત એસાઇન્મેન્ટ પ્રક્રિયક '=' વડે આપી શકીએ છીએ. જાવામાં કોઈ ચલની અંદર ડેટા મેળવવાની એક રીત એસાઇન્મેન્ટ સ્ટેટમેન્ટ છે. એસાઇન્મેન્ટ સ્ટેટમેન્ટનું સ્વરૂપ નીચે મુજબ હોય છે :

< ચલ > = < પદાવલી >; એટલે કે <variable> = <expression>;

અહીં <expression> એ ડેટાકિંમતનો નિર્દેશ કે ગણતરીનો ઉલ્લેખ છે.

જ્યારે એસાઇન્મેન્ટ વિધાનનો અમલ થાય છે, ત્યારે સૌ પ્રથમ = ચિહ્નની જમણી બાજુની પદાવલીની કિંમત શોધવામાં આવે છે અને તે પરિણામને = ચિહ્નની ડાબી બાજુના ચલમાં મૂકવામાં આવે છે.

**ઉદાહરણ :** એક સાદું એસાઇન્મેન્ટ વિધાન લો : rate = 10.02f;

અહીં, એસાઇન્મેન્ટ વિધાનમાં ચલ 'rate' છે અને પદાવલી કે expression સંખ્યા '10.02f' છે. આ વિધાનનો અમલ થવાથી float પ્રકારના ચલ 'rate'ની કિંમત બદલાઈને 10.02 થશે.

હવે, બીજું એસાઇન્મેન્ટ વિધાન લો : balance = balance - cost;

અહીં, સૌ પ્રથમ ચલ balance અને costની મેમરીમાં રહેલી કિંમતોનો ઉપયોગ કરીને પદાવલી balance-costની કિંમત શોધવામાં આવે છે. તે પછી ચલ balanceની જૂની કિંમતને બદલીને પરિણામની કિંમત મૂકવામાં આવે છે.

જ્યારે પદાવલીની જમણી બાજુએ કોઈ ચલ વાપરવામાં આવે છે, ત્યારે તેનો અર્થ એ છે કે ચલમાં કિંમત સંગ્રહ કરેલી છે. જ્યારે એસાઇન્મેન્ટ સ્ટેટમેન્ટની ડાબી બાજુએ ચલ વાપરવામાં આવે છે ત્યારે તે કોઈ મેમરી સ્થાનાંકનો નિર્દેશ કરે છે કે જેને ચલનું નામ આપેલું હોય છે, જેમાં કિંમત મૂકવામાં આવે છે. આથી, પદાવલીની ડાબી બાજુના ચલને *lvalue* કહેવામાં આવે છે જે મેમરીના સ્થાનાંકનો નિર્દેશ કરે છે.

સામાન્ય રીતે, એસાઇન્મેન્ટ સ્ટેટમેન્ટની જમણી બાજુની પદાવલીનો પ્રકાર ડાબી બાજુના ચલના પ્રકાર જેવો હોવો જોઈએ. જો તે મેળ ખાતા ન હોય તો પદાવલીની કિંમત આપમેળે ચલના પ્રકાર સાથે મેળ ખાય (match) તે પ્રમાણે રૂપાંતરિત થાય છે. જો ડાબી બાજુના ચલના કરતા પદાવલીનો ડેટાપ્રકાર ઉચ્ચ હોય (larger) તો તે ચોકસાઈ સમસ્યા (precision problem)ને કારણે ભૂલ (error)માં પરિણમી શકે છે. ઉદાહરણ તરીકે, intમાંથી short કે doubleમાંથી floatમાં આપોઆપ રૂપાંતર હોઈ શકે નહીં.

### શૉર્ટહેન્ડ એસાઈન્મેન્ટ પ્રક્રિયક (Shorthand Assignment Operators)

જાવા પણ એસાઇન્મેન્ટની શૉર્ટહેન્ડ આવૃત્તિ પૂરી પાડે છે. તે ટાઇપ કરવાનો સમય બચાવે છે. તેનું સ્વરૂપ <variable> <operator> = <expression> હોય છે. તેની અસર <variable> = <variable> <operator> <expression> સમાન જ છે. અહીં પ્રક્રિયક બે સંકાર્ય વાપરતું દ્વિપદી (binary) પ્રક્રિયક હોવું જોઈએ.

શૉર્ટહેન્ડ એસાઇન્મેન્ટ પ્રક્રિયકનાં કેટલાંક ઉદાહરણો નીચે મુજબ છે : a += b અને q &&= p. અહીં a += b એ a = a + b સમાન છે, અને q &&= p એ q = q && p સમાન છે.

### ટાઇપકાસ્ટ (Type-cast)

કેટલાક કિસ્સાઓમાં આપણે ફરજિયાતથી રૂપાંતર ઇચ્છતા હોઈએ કે જે આપમેળે ન બને. આ માટે આપણે જે વાપરીએ છીએ, તેને ટાઇપ કાસ્ટ (type-cast) કહેવામાં આવે છે. આપણે જે કિંમતનું રૂપાંતર કરવા ઇચ્છતા હોય, તેની આગળ કૌંસમાં ટાઇપ નેઇમ (type-name) લખીને ટાઇપકાસ્ટ જણાવી શકીએ છીએ. તેનું સ્વરૂપ (<data-type>) <expression> હોય છે.

**ઉદાહરણ :** int a; short b;

a = 17; b = (short)a;

અહીં, ચલ aનું ટાઇપકાસ્ટ વાપરીને short પ્રકારની કિંમતથી ચોક્કસપણે (explicitly) રૂપાંતર કરેલું છે.

### જાવા પ્રક્રિયકોના પ્રિસિડન્સ અને એસોસિએટિવિટી (Precedence and Associativity of Java Operators) :

જ્યારે પદાવલીમાં કેટલાક પ્રક્રિયકો હોય ત્યારે પદાવલીના પ્રક્રિયકોની કયા ક્રમમાં ગણતરી કરવી તે જણાવતા સારી રીતે વ્યાખ્યાયિત કરેલા નિયમો જાવા ધરાવે છે. આ પ્રક્રિયકોની **અગ્રતા** (priority અથવા precedence) પ્રમાણે પદાવલીની ગણતરી કરવામાં આવે છે.

### અગ્રતાક્રમ કે પ્રિસિડન્સ ઑર્ડર (Precedence Order)

જ્યારે બે પ્રક્રિયકની અલગ અલગ અગ્રતા હોય, ત્યારે ઉચ્ચ અગ્રતા (higher precedence) ધરાવતા પ્રક્રિયક ઉપર પહેલાં પ્રક્રિયા કરવામાં આવે છે. ઉદાહરણ : પદાવલી a + b * cમાં ગુણાકારનો સરવાળા કરતાં વધારે પ્રિસિડન્સ હોવાથી પહેલાં b*cની ગણતરી કર્યા પછી તેનું પરિણામ aમાં ઉમેરવામાં આવે છે. આથી તે a + (b*c) લખવા સમાન છે. બાહ્ય કૌંસ દ્વારા અગ્રતાનિયમો (પ્રિસિડન્સ રૂલ્સ) બિનઅસરકારક બનાવી શકાય છે. આથી ગણતરી કરતાં સમયે ગૂંચવાડો ઊભો કરવાને બદલે છૂટથી કૌંસનો ઉપયોગ કરો.

### સહચારિતા કે એસોસિએટિવિટી (Associativity)

જ્યારે પદાવલીમાં એક સમાન પ્રિસિડન્સ ધરાવતાં બે પ્રક્રિયકો હોય, ત્યારે પદાવલીની ગણતરી તેના **એસોસિએટિવિટી** પ્રમાણે કરવામાં આવે છે. એસોસિએટિવિટી કઈ દિશામાં (ડાબી બાજુથી જમણી બાજુ અથવા જમણી બાજુથી ડાબી બાજુ) કાર્ય કરવાનું છે, તે નક્કી કરે છે. મોટા ભાગના કિસ્સાઓમાં એસોસિએટિવિટી ડાબી બાજુથી જમણી બાજુ હોય છે. એકપદી કાર્યો અને એસાઇન્મેન્ટમાં તે જમણી બાજુથી ડાબી બાજુ હોય છે.

આ પ્રકરણમાં ચર્ચા કરેલાં પ્રક્રિયકોની યાદી કોષ્ટક 7.4માં આપેલી છે, જેનો ક્રમ સૌથી વધારે પ્રિસિડન્સ (જેની ગણતરી સૌપ્રથમ થાય)થી ઓછામાં ઓછી પ્રિસિડન્સ (જેની ગણતરી સૌથી છેલ્લી થાય) પ્રમાણે છે :

| Operations | Operators | Associativity |
|---|---|---|
| Unary operations | ++, --, !, unary – and +, type-cast | Right-to-left |
| Multiplication, division, modulus | *, ?, % | Left-to-right |
| Addition and subtraction | +, - | Left-to-right |
| Relational operators | <, >, <=, >= | Left-to-right |
| Relational operators (Equality and inequality) | ==, != | Left-to-right |
| Logical AND | && | Left-to-right |
| Logical OR | \|\| | Left-to-right |
| Conditional operator | ? : | Right-to-left |
| Assignment operators | =, +=, -=, *=, /=, %= | Right-to-left |

**કોષ્ટક 7.4 : પ્રક્રિયકો અને તેનું પ્રિસિડન્સ**

ઉદાહરણ : x = y = z = 7ને x = (y = (z = 7)) તરીકે ગણીને ત્રણે ચલમાં કિંમત 7 મળશે. આ એસાઈન્મેન્ટ પ્રક્રિયકની જમણી બાજુથી ડાબી બાજુની એસોસિએટિવિટીને કારણે છે. તમને યાદ હશે કે એસાઈન્મેન્ટ એ એક પ્રક્રિયક છે, આથી એસાઈન્મેન્ટ સ્ટેટમેન્ટની કિંમત જમણી બાજુની પદાવલી બરાબર છે. આથી, x=y=z=7માં પહેલા z=7ની ગણતરી થાય છે. તે ચલ zને કિંમત 7 એસાઈન કરે છે અને પદાવલી z=7-ની કિંમત પણ 7 છે, આથી તે પદાવલી x=y=7 બનાવે છે.

બીજી બાજુએ, 72 / 2 / 3 એ (72 / 2) / 3 તરીકે ગણવામાં આવે છે કારણકે / પ્રક્રિયકની ડાબી બાજુથી જમણી બાજુની એસોસિએટિવિટી છે. અહીં એ નોંધવું જોઈએ કે જાવા ભાષાના વિગતવાર વર્ણનમાં કોઈ ચોક્કસ પ્રક્રિયક અગ્રતાક્રમ કોષ્ટક નથી અને વેબ પર તથા પાઠ્યપુસ્તકોમાં ઉપલબ્ધ કોષ્ટકો કેટલીક નાની બાબતોમાં સંમત નથી.

## કન્ટ્રોલ સ્ટ્રક્ચર (Control Structures)

સામાન્ય રીતે, વિધાનોનો અમલ એક પછી એક એમ ક્રમિક રીતે થાય છે. અમુક સમયે પ્રોગ્રામના તર્કને આ ક્રમનો પ્રવાહ બદલવાની જરૂર પડે છે. જે વિધાનો અમલના પ્રવાહને નિયંત્રણ કરવા સમર્થ બનાવે છે તેને નિયંત્રણમાળખાં કે કન્ટ્રોલ-સ્ટ્રક્ચર (control structure) ગણવામાં આવે છે.

કન્ટ્રોલ સ્ટ્રક્ચર બે પ્રકારનાં હોય છે : લૂપ્સ (loops) અને બ્રાન્ચિસ (branches). લૂપનો ઉપયોગ અમુક શરત સંતોષાય ત્યાં સુધી વિધાનોની શ્રેણીનું વારંવાર પુનરાવર્તન કરવાનું હોય ત્યારે થાય છે. બ્રાન્ચનો ઉપયોગ જ્યારે બે કે વધુ શક્ય કાર્ય-દિશામાંથી પસંદગી કરવાની હોય, ત્યારે થાય છે, આથી તેને સિલેક્ટિવ સ્ટ્રક્ચર (selective structure) પણ કહેવામાં આવે છે.

પ્રોગ્રામમાં સામાન્ય પ્રવાહના નિયંત્રણ માટે જાવામાં વપરાતા કન્ટ્રોલ-સ્ટ્રક્ચર નીચે મુજબ છે : if વિધાન, switch વિધાન, while લૂપ, do...while લૂપ અને for લૂપ. આ બધાં માળખાં એ એક વિધાન કે વિધાનોનો બ્લૉક હોઈ શકે છે.

## બ્લૉક (Block)

બ્લૉક-સ્ટેટમેન્ટ એ કૌંસની જોડી "{" અને "}" વચ્ચે લખાયેલાં વિધાનોનું જૂથ છે.

બ્લૉકનું સ્વરૂપ નીચે મુજબ હોય છે :

```
{
        <statements>
}
```

બ્લૉકનો ઉપયોગ નીચે જણાવેલા અનેક હેતુઓ માટે કરી શકાય છે :

- સામાન્ય રીતે કન્ટ્રોલ-સ્ટ્રક્ચરમાં વિધાનોની શ્રેણીના જૂથને એક એકમ બનાવી એક વિધાન તરીકે ગણવા માટે. (ટૂંક સમયમાં ચર્ચા કરીશું.)
- તાર્કિક રીતે સંબંધિત વિધાનોનું જૂથ બનાવવા માટે.
- બ્લૉકની અંદર વિધાનોના સ્થાનિક અવકાશ (local scope) સાથે નવા ચલ બનાવવા.

**ઉદાહરણ :**

```
{ // This block exchanges the values of x and y
        int temp;       // temporary variable for use in this block only
        temp = x;       // save a copy of x in temp
        x = y;          // copy y into x
        y = temp;       // copy temp into y
}
```

જ્યારે આપણે કોઈ બ્લોકની અંદર ચલ ઘોષિત કરીએ છીએ, ત્યારે તે ચલ તે બ્લોકમાં જ સ્થાનિક (local) રહે છે અને તે બ્લોકની બહાર તેનું અસ્તિત્વ રહેતું નથી. આપણે જે બ્લોકની અંદર ઘોષિત કરેલ હોય તેની બહાર તેનો ઉપયોગ કરી શકતા નથી.

આપણે જે બ્લોકની અંદર ચલ ઘોષિત કરેલા હોય તે બ્લોકની બહાર તે સંપૂર્ણપણે ન મેળવી શકાય તેવા (inaccessible) અને અદૃશ્ય (invisible) રહે છે. જ્યારે ચલ ઘોષિત કરવાના વિધાનનો અમલ કરવામાં આવે છે, ત્યારે ચલમાં કિંમત રાખવા માટે મેમરીની ફાળવણી કરવામાં આવે છે. જ્યારે બ્લોકનો અંત આવે છે, ત્યારે તે મેમરી છૂટી કરવામાં આવે છે અને તેનો ફરી ઉપયોગ કરવા માટે ઉપલબ્ધ બને છે. ચલ બ્લોક પૂરતો સ્થાનિક કહી શકાય.

એક સામાન્ય ખ્યાલ છે જેને ચલનો "સ્કોપ (scope)" (કાર્યક્ષેત્રની મર્યાદા) કહેવામાં આવે છે. ચલની કાર્યક્ષેત્રની મર્યાદા (scope) પ્રોગ્રામનો એ ભાગ છે જેમાં તે ચલ માન્ય છે. જે બ્લોકમાં ચલ વ્યાખ્યાયિત કરેલો હોય તે બ્લોક પૂરતો જ ચલનો સ્કોપ મર્યાદિત રહે છે.

જ્યારે આપણે કોઈ ચલના સ્કોપની અંદર બીજો ચલ તે જ નામનો ઘોષિત કરવા પ્રયત્ન કરીએ, ત્યારે ભૂલ (error) મેળવીએ છીએ. આકૃતિ 7.10માં આપેલું કોડલિસ્ટિંગ અને પરિણામ જુઓ. અહીં આપણે ફક્ત સારી સમજણ માટે labelled બ્લોકનો ઉપયોગ કર્યો છે. અહીં આપેલા ઉદાહરણમાં તે વાપરવું અનિવાર્ય નથી. આપણે labelled blockનો ઉપયોગ આ પ્રકરણમાં હવે પછી જોઈશું.

આકૃતિ 7.10માં આપેલા કોડલિસ્ટિંગમાં આપણે blk2 નામના બ્લોકમાં ચલ x ઘોષિત કરવા પ્રયત્ન કર્યો છે. આ બ્લોક blk2 એ main મેથડ બ્લોકની અંદર છે. main મેથડની અંદર ઘોષિત કરેલા ચલ xનો સ્કોપ blk2ની અંદર પણ છે. આથી, blk2 બ્લોકની અંદર ચલ x ઘોષિત કરવાથી તે main મેથડ બ્લોકમાંના ચલ x સાથે સંઘર્ષમય સ્થિતિ (conflicts) ઉદ્‌ભવશે અને કંપાઇલર ભૂલ દર્શાવશે.

Block.java - SciTE

File Edit Search View Tools Options Language Buffers Help

1 Block.java

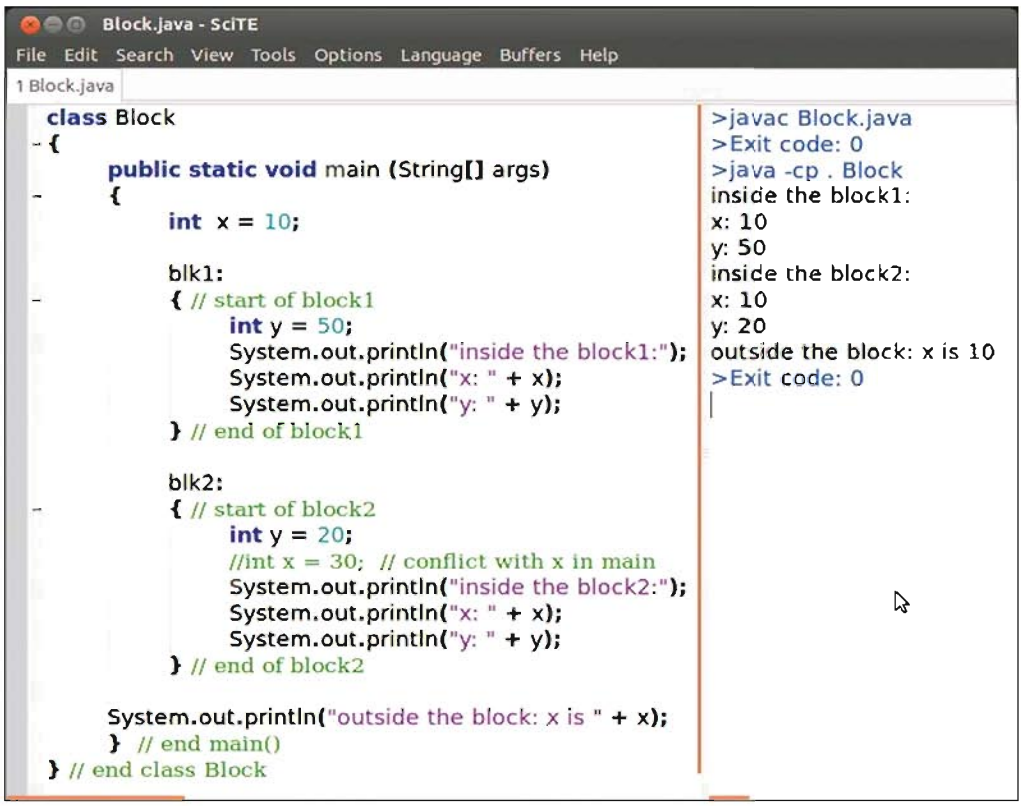
```
class Block
{
    public static void main (String[] args)
    {
        int x = 10;

        blk1:
        { // start of block1
            int y = 50;
            System.out.println("inside the block1:");
            System.out.println("x: " + x);
            System.out.println("y: " + y);
        } // end of block1

        blk2:
        { // start of block2
            int y = 20;
            int x = 30;  // conflict with x in main
            System.out.println("inside the block2:");
            System.out.println("x: " + x);
            System.out.println("y: " + y);
        } // end of block2

    System.out.println("outside the block: x is " + x);
    }  // end main()
} // end class Block
```

```
>javac Block.java
Block.java:18: x is already defined in main(java.lang.String[])
                int x = 30;  // conflict with x in main
                    ^
1 error
>Exit code: 1
```

**આકૃતિ 7.10 : એક જ કાર્યક્ષેત્રની મર્યાદા (સ્કોપ)માં સંઘર્ષમય સ્થિતિમાં ચલનાં નામો**

આકૃતિ 7.11માં કોડમાં ફેરફાર કરીને દર્શાવ્યો છે. અહીં, main મેથડના પૂર્ણ સ્કોપમાં ચલ x ઘોષિત કરેલ છે. આ જ બ્લોકમાં blk1 અને blk2 હોવાથી તેમાં પણ ચલ x ઉપલબ્ધ રહે છે. blk1 બ્લોકમાં ચલ y ઘોષિત કરેલ છે જે બ્લોકની બહાર અસ્તિત્વ ધરાવતો નથી. blk1ના અમલ પછી ચલ y કાઢી નાખવામાં આવે છે. જ્યારે blk2માં y ઘોષિત કરવામાં આવે છે, ત્યારે તે ઉપલબ્ધ મેમરીમાંથી કોઈ પણ મેમરીસ્થાન ઉપર કરે છે. જો આપણે blk2માં ચલ yને કોઈ કિંમત ન આપીએ.

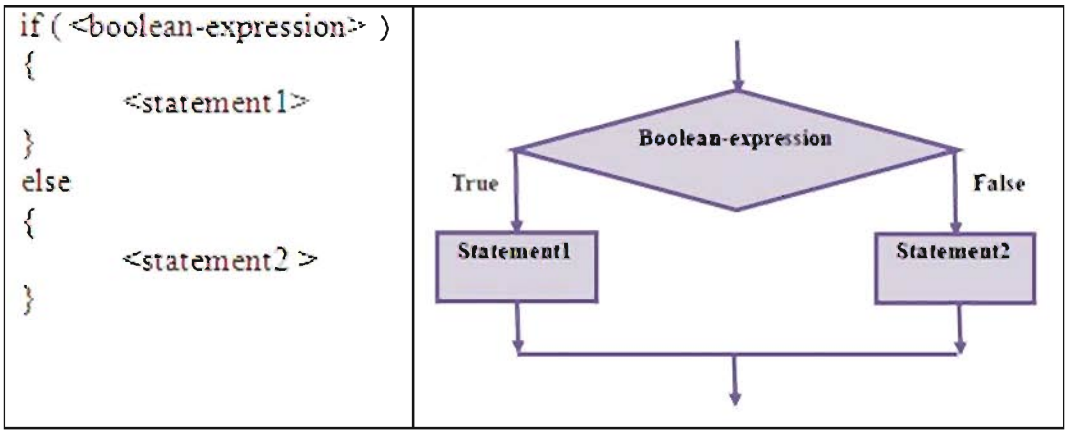
અને જ્યારે blk2માં તેનો નિર્દેશ કરવામાં આવે ત્યારે તે ભૂલ (error) પરત કરે છે. આનો અર્થ એ થાય કે blk1ના ચલ yને blk2ના ચલ y સાથે કોઈ સંબંધ નથી. હકીકતમાં blk1નો ચલ y એ blk2માં મેળવી શકાતો નથી.




આકૃતિ 7.11 : ચલનો સ્કોપ દર્શાવતો પ્રોગ્રામ

## if વિધાન (if Statement)

જ્યારે પ્રોગ્રામમાં if વિધાન વાપરવામાં આવે છે ત્યારે બુલિયન પદાવલીની કિંમત true છે કે false તેના પ્રમાણે બેમાંથી એક વિકલ્પ પ્રમાણે કાર્ય કરવા સમર્થ બનાવે છે. તે 'branching' અથવા 'decision' અથવા 'selective' કન્ટ્રોલ સ્ટ્રક્ચરનું ઉદાહરણ છે. તેનું સ્વરૂપ આકૃતિ 7.2માં દર્શાવ્યું છે.




આકૃતિ 7.12 : if વિધાનનું બંધારણ

જ્યારે if વિધાનનો અમલ થાય છે, ત્યારે સૌપ્રથમ બુલિયન પદાવલીની કિંમત શોધવામાં આવે છે. જો કિંમત true હોય, તો તે statement1નો અમલ કરશે; નહિ તો ચાવીરૂપ શબ્દ else પછી લખેલા વિધાનોના બ્લોકનો અમલ કરશે. if વિધાનમાં <statement> સામાન્ય રીતે બ્લોકસ્ટેટમેન્ટ હોય છે. તે કોઈ એક વિધાન પણ હોઈ શકે પણ બ્લોક વાપરવો સલાહભર્યું

છે, જેથી ભવિષ્યમાં જરૂર હોય, તો વધારાનાં વિધાન ઉમેરવાનું કાર્ય સરળ બને. નીચે આપેલો પ્રોગ્રામનો ખંડ જુઓ, જે પૂર્ણાંક સંખ્યા એકી છે કે બેકી તે નક્કી કરવા માટે if વિધાનનો ઉપયોગ કરે છે.

```
if ( x%2 == 0 ) // assume int x
  { // divisible by 2
        System.out.print(x);
        System.out.println (" is even");
  }
else
  { // not divisible by 2
System.out.println (x + " is odd");
  }
```

અહીં ચાવીરૂપ શબ્દ (keyword) else અને else પછી લખેલ બ્લોક વૈકલ્પિક છે. આથી, else વિના if વિધાનનું સ્વરૂપ નીચે પ્રમાણે હોઈ શકે :

```
if ( <boolean-expression> )
{
        <statement1>
}
```

જ્યારે એક if વિધાનની અંદર બીજું if વિધાન વાપરવામાં આવે, તો તેને nested-if વિધાન કહેવામાં આવે છે. હવે નીચે આપેલું ઉદાહરણ જુઓ, જે મેળવેલા ગુણ પ્રમાણે ગ્રેડ નક્કી કરે છે.

```
if ( marks >= 70 ) // int marks
  { grade = 'A';       // char grade
  }
else
  {
   if (marks >= 60)
        grade = 'B';
   else if (marks >= 50)
                grade = 'C';
        else
         grade = 'F';
 }
```

આપણે nested-if વિધાનનું એક વધારે ઉદાહરણ લઈએ.

```
if ( x > 0 )
        if (y > 0)
                System.out.println("both x and y are greater than zero");
else
        System.out.println("x <= 0");
```

અહીં, એવું જણાય છે કે else ભાગ "if (x > 0)" વિધાનને સંગત છે, પણ હકીકતમાં તે "if (y > 0)" સાથે જોડાયેલ છે, જે તેની નજીક છે. આ રીતે ફક્ત "if (x>0)"ના true ભાગનો અમલ થાય છે.

જો આપણે else ભાગને "if (y>0)" સાથે જોડવા ઇચ્છતા હોય, તો આપણે nested-ifને નીચે જણાવ્યા પ્રમાણે બ્લૉકમાં રાખવું જોઈએ.

```
if ( x > 0 )
{
        if (y > 0)
                System.out.println("both x and y are greater than zero");
}
else
System.out.println("x <= 0");
```

### સ્વિચ-સ્ટેટમેન્ટ (Switch Statement)

જ્યારે ચલ કે પદાવલીની કિંમત પ્રમાણે અલગ-અલગ ઘણાં કાર્યોના વિકલ્પ હોય, ત્યારે સ્વિચ-સ્ટેટમેન્ટ (switch statement) વાપરવામાં આવે છે. અહીં, તે પદાવલી ટેસ્ટ કરવાની છે તેનું પરિણામ એવા પ્રકારનું હોવું જોઈએ કે જે જુદી જુદી કિંમતો (discrete) હોય. સ્વિચ-સ્ટેટમેન્ટનું સ્વરૂપ નીચે પ્રમાણે હોય છે :

```
switch (<expression>)
{
        case <constant-1>:
                <statements-1>
                break;
        case <constant-2>:
                <statements-2>
                break;

        // more such cases

        case <constant-n>:
                <statements-n>
                break;

        default:
                <statements-n1>
}
```

સ્વિચ-સ્ટેટમેન્ટમાં ટેસ્ટ-પદાવલીનો પ્રકાર byte, char, short અથવા int હોવો જોઈએ. તે enum ડેટાપ્રકારની પણ હોઈ શકે (અહીં ચર્ચા કરેલી નથી.).

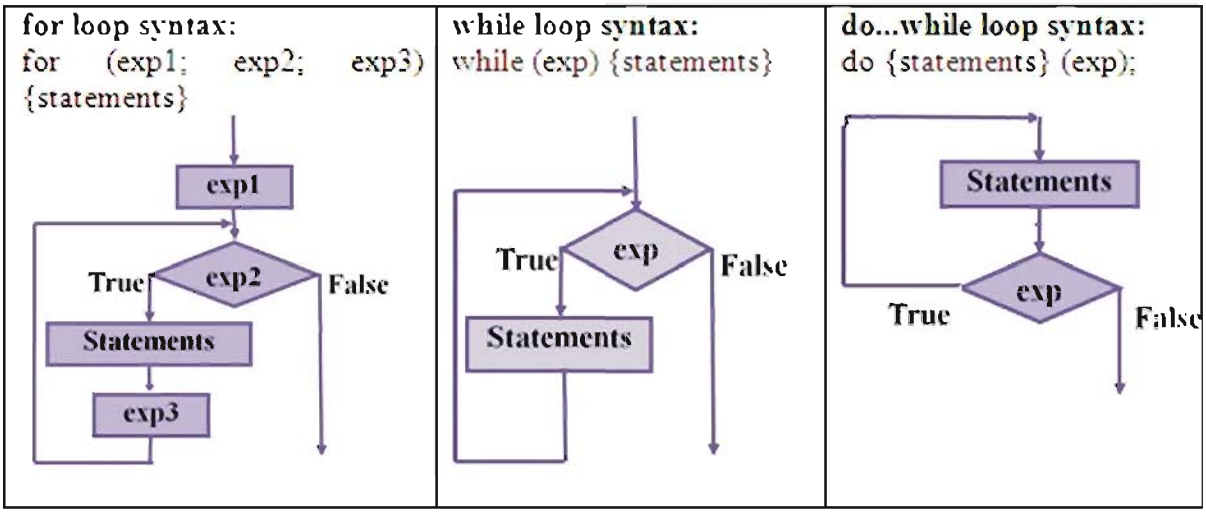
સ્વિચ-સ્ટેટમેન્ટનો અમલ કરતા સમયે case1 થી શરૂ કરી ટેસ્ટ-પદાવલીની કિંમત દરેક case કિંમત સાથે સરખાવવામાં આવે છે. જો બંને કિંમત સમાન મળે, તો તે caseનાં વિધાનો (બ્લૉક હોય તે જરૂરી નથી)નો અમલ થાય છે. જો સમાન

કિંમત ન મળે તો default વિધાનનો અમલ થાય છે. default વિધાન વૈકલ્પિક છે, આથી જો default ન આપેલું હોય અને કોઈ પણ case સાથેની કિંમત મૅચ ન થાય, તો કંઈ પણ કર્યા વિના સ્વિચ-સ્ટેટમેન્ટનો અંત આવે છે.

અહીં નોંધ કરો કે દરેક case વિધાન પછી break વિધાન ફરજિયાત નથી. break વિધાનનો ઉપયોગ સ્વિચ-સ્ટેટમેન્ટનો અંત લાવવા માટે થાય છે, એટલે કે, સ્વિચના અંત પછીના પહેલા વિધાન ઉપર જવા માટે થાય છે.

## રિપીટિટિવ કન્ટ્રોલ-સ્ટ્રક્ચર (Repetitive Control Structures)

જાવા ત્રણ પ્રકારનાં લૂપિંગની રચના પૂરી પાડે છે : for, while અને do...while. આકૃતિ 7.13માં આ બધાં લૂપની વાક્યરચના (syntax) અને માળખું (structure) દર્શાવેલું છે.




આકૃતિ 7.13 : રિપીટિટિવ કન્ટ્રોલ-સ્ટ્રક્ચર (પુનરાવર્તી નિયંત્રણમાળખા)

for અને while લૂપમાં સૌપ્રથમ ટેસ્ટપદાવલીની ગણતરી કરવામાં આવે છે અને જો શરત સાચી હોય, તો લૂપની અંદરનાં વિધાનોનો અમલ થાય છે. આ લૂપ પ્રવેશ-નિયંત્રિત (entry-controlled) અથવા પ્રી-ટેસ્ટ (pre-test) પ્રકારના લૂપ છે. અહીં, એ શક્ય છે કે લૂપની અંદરનાં વિધાનોનો બિલકુલ અમલ જ ન થાય.

જ્યારે પુનરાવૃત્તિ (iterations)ની સંખ્યા અગાઉથી નક્કી હોય, ત્યારે સામાન્ય રીતે for લૂપ વપરાય છે. આ લૂપની અંદર ત્રણે પદાવલીઓ વૈકલ્પિક હોય છે. પહેલી પદાવલી initializer (જે પ્રારંભિક કિંમત આપે છે.), બીજી પદાવલી condition (શરત) અને ત્રીજી પદાવલી iterator (પુનરાવૃત્તિ કરનાર) છે. આથી for(;;) એ માન્ય વિધાન છે, પણ લૂપમાંથી બહાર આવવા માટે કોઈ નિયંત્રણવિધાન (control-statement)ની જરૂર રહે છે. જો break વિધાનનો અમલ ન થાય, તો તે અનંત લૂપમાં પરિણમે છે.

do...while લૂપમાં પહેલાં પદાવલીઓનો અમલ થાય છે અને પછી પદાવલીની કિંમત ટેસ્ટ કરવામાં આવે છે. જો ટેસ્ટની શરતનું પરિણામ true મળે તો લૂપમાં પુનરાવર્તન થાય છે. આ રીતે, do...while લૂપ નિર્ગમન-નિયંત્રિત (exit-controlled) અથવા પોસ્ટ-ટેસ્ટ (post-test) પ્રકારનું લૂપ છે. અહીં, લૂપની અંદરનાં વિધાનો ઓછામાં ઓછા એક વખત અમલમાં આવે છે.

નીચેનાં ઉદાહરણો 0 થી 9ની પૂર્ણાંક સંખ્યાઓ ત્રણે પ્રકારનાં લૂપનો ઉપયોગ કરીને છાપે છે.

### for લૂપનો ઉપયોગ :

```
for (int i = 0; i < 10; i++)
{
        System.out.println(i);
}
```

### while લૂપનો ઉપયોગ

```
int i = 0;
while(i < 10)
{
        System.out.println(i++); //prints i before applying i++
}
```

### do...while લૂપનો ઉપયોગ

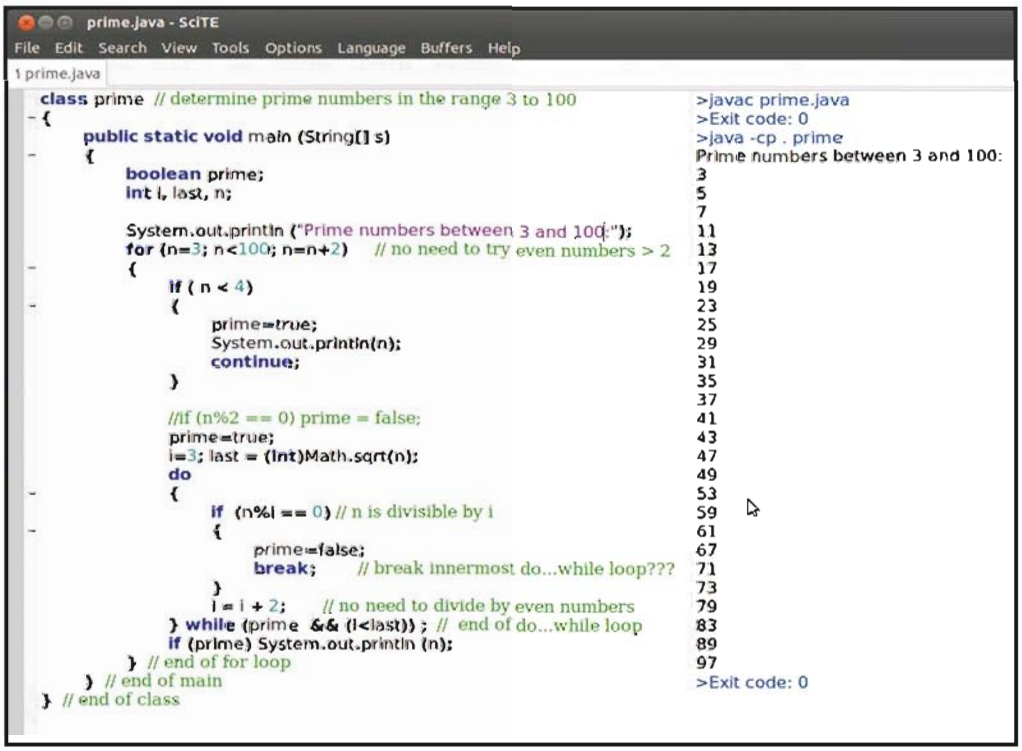
```
int i = 0;
do
{
System.out.println(i++);
} while(i < 10);
```

### break અને continue વિધાનનો ઉપયોગ (Use of Break and Continue Statement)

break વિધાનનો ઉપયોગ સ્વિચ કે લૂપ પ્રકારની રચનાની બહાર પ્રોગ્રામનું નિયંત્રણ ફેરવવા માટે થાય છે.
જ્યારે સ્વિચ (switch) સાથે break વાપરવામાં આવે છે, ત્યારે તે પછીનાં વિધાનોનો અમલ કર્યા વિના તેમને છોડી દઈને switch વિધાનના અંત પછીના પહેલા વિધાન ઉપર નિયંત્રણનું સ્થાનાંતર થાય છે.
લૂપ (loop)માં break વિધાન લૂપમાંથી નિર્ગમન (exit) માટે વપરાય છે. જ્યારે લૂપમાં break વિધાનનો અમલ થાય છે, ત્યારે લૂપમાં સમાયેલાં બધાં વિધાનો અમલ કર્યા વિના છોડી દેવામાં આવે છે અને વધારાનું કોઈ પુનરાવર્તન કરવામાં આવતું નથી. પ્રોગ્રામનું નિયંત્રણ લૂપના અંત પછીના પહેલા વિધાન ઉપર સ્થાનાંતર થાય છે. યાદ રાખો કે લૂપની બહાર અને સૌથી નજીકના આ વિધાન ઉપર break કૂદકો મારે છે.
continue વિધાનનો ઉપયોગ લૂપમાંનાં તેનાં પછીનાં વિધાનોને છોડી દેવા માટે અને તે પછીનાં પુનરાવર્તન માટે થાય છે. બંને વિધાનો break અને continueનો ઉપયોગ C ભાષાની રીતે જ થાય છે.

### નેસ્ટેડલૂપ (Nested-loops)

જાવામાં સમાન પ્રકારના કે અલગ પ્રકારના લૂપનું નેસ્ટિંગ (Nesting) કરી શકાય છે. આકૃતિ 7.4માં નેસ્ટેડ લૂપ, break અને continue વિધાનોનો ઉપયોગ દર્શાવ્યો છે.




આકૃતિ 7.14 : નેસ્ટેડ લૂપ, break અને continue વિધાનનો ઉપયોગ

આકૃતિ 7.14માં આપણે એક પ્રોગ્રામ લખેલો છે જે 3 થી 100 વચ્ચેની અવિભાજ્ય સંખ્યા (prime numbers) છાપે છે. આ ઉપરાંત તે sqrt વિધેયનો ઉપયોગ પણ દર્શાવે છે. જાવાની ક્લાસ-લાઇબ્રેરીમાં Math નામના ક્લાસનો સમાવેશ થાય છે. Math ક્લાસમાં ગાણિતિક કાર્યોનો સંપૂર્ણ સેટ વ્યાખ્યાયિત કરેલો છે. sqrt() વિધેય Math ક્લાસની એક સ્ટેટિક મેથડ સ્વરૂપ છે જે Math.sqrt() થી ઇન્વોક (invoke) કરી શકાય છે.

## લેબલ કરેલા લૂપ અને લેબલ કરેલા break (Labelled Loops and Labelled Break)

જ્યારે આપણે નેસ્ટેડ લૂપ વાપરીએ છીએ, ત્યારે તેમાં રહેલું break વિધાન કે જે સૌથી નજીકના લૂપમાં સમાયેલું છે તેને અટકાવી લૂપની બહાર નિયંત્રણ સ્થાનાંતર કરે છે. એ જ પ્રમાણે continue વિધાન તે જે લૂપમાં સમાયેલું છે તે ફરી શરૂ કરે છે.

જો આપણે લૂપ અટકાવવું હોય અથવા કયા લૂપનું પુનરાવર્તન કરવું તેનું નિયંત્રણ કરવા ઇચ્છતા હોઈએ તો આપણે લેબલ કરેલા લૂપનો ઉપયોગ કરી શકીએ છીએ. લેબલ કરેલું લૂપ વાપરવા માટે લૂપની શરૂઆત પહેલાં લેબલ (label) અને તેના પછી ગુરુવિરામ (:) લખવું પડે. જે લૂપમાં હોય તેની બહાર નિયંત્રણ લઈ જવું હોય ત્યારે break અથવા continue કીવર્ડ (keyword) પછી લેબલનું નામ લખવામાં આવે છે. આકૃતિ 7.15માંનો પ્રોગ્રામ લેબલ સાથેના લૂપનો ઉપયોગ દર્શાવે છે.

જ્યારે આકૃતિ 7.15ના કોડનો અમલ થાય છે, ત્યારે i*x ની કિંમત 350 થાય છે, જ્યારે i=5 અને x = 70 હોય. જ્યારે while લૂપની અંદર રહેલાં ifની શરતનું પરિણામ true મળે ત્યારે 'break out;' વિધાનનો અમલ થાય છે. તે બહારનું for લૂપ કે જે 'out' નામથી લેબલ કરેલું છે તેનો અંત લાવશે. જો ફક્ત break વિધાન જ વાપર્યું હોય તો તે જે while લૂપની અંદર છે તેમાંથી જ બહિર્ગમન કરતા અને for લૂપના પછીના પુનરાવર્તનથી ચાલુ રહેશે. અહીં, બહારનું for લૂપ i=6 અને તેનાથી વધારે કિંમત માટે અમલમાં આવશે નહીં.

LblBlock.java - SciTE

File Edit Search View Tools Options Language Buffers Help

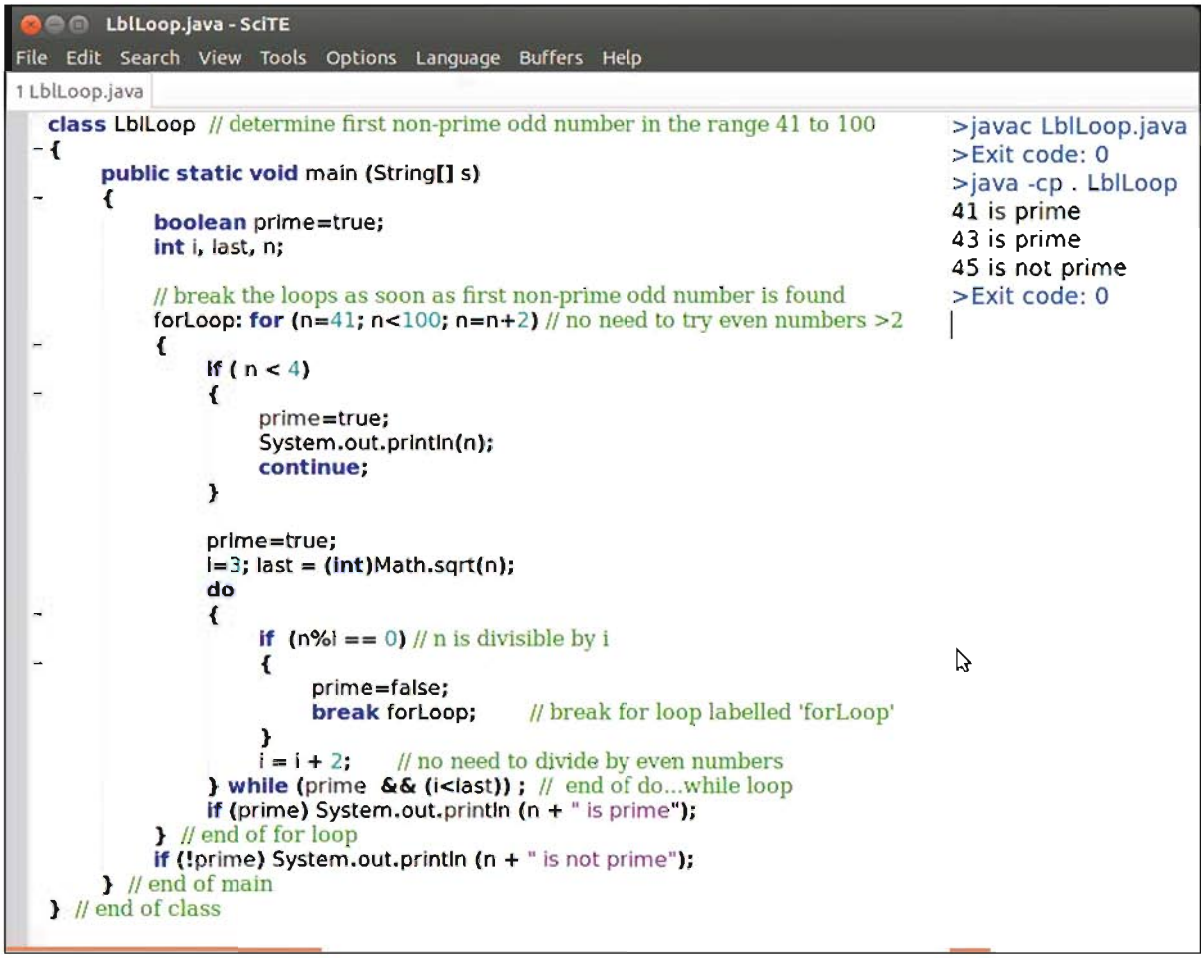
1 LblBlock.java

```
class LblBlock  // Use of labelled block to break the outer loop
{
    public static void main (String[] s)
    {
        int x=0;
        out: for (int i = 4; i <10; i++)
        {
            x=10;
            while (x < 100)
            {
                System.out.println ( "Inside while loop: i is "
                                    + i + ", x is " + x);
                if ( i * x == 350 )
                    break out;
                x = x+20;
            } // end while
            System.out.println ( "\nOutside while loop: i is "
                                + i + ", x is " + x + "\n");
        } // end out for loop
        System.out.println ( "\nOutside for loop: x is " + x);
    } // end of main
} // end of class
```

```
>javac LblBlock.java
>Exit code: 0
>java -cp . LblBlock
Inside while loop: i is 4, x is 10
Inside while loop: i is 4, x is 30
Inside while loop: i is 4, x is 50
Inside while loop: i is 4, x is 70
Inside while loop: i is 4, x is 90

Outside while loop: i is 4, x is 110

Inside while loop: i is 5, x is 10
Inside while loop: i is 5, x is 30
Inside while loop: i is 5, x is 50
Inside while loop: i is 5, x is 70

Outside for loop: x is 70
>Exit code: 0
```

આકૃતિ 7.15 : લેબલ કરેલા લૂપ અને breakનો ઉપયોગ

હવે આપણે પ્રોગ્રામમાં લેબલ કરેલા લૂપ અને breakનું બીજું ઉદાહરણ લઈએ જે 40 થી 100 વચ્ચેની પ્રથમ એકી વિભાજ્ય (non-prime) સંખ્યા આવે ત્યાં સુધીની અવિભાજ્ય સંખ્યાઓ (prime numbers) છાપે. જુઓ આકૃતિ 7.16.




**આકૃતિ 7.16 : 40થી 100 વચ્ચેની અભાજ્ય સંખ્યા છાપતો પ્રોગ્રામ**

## સારાંશ

આ પ્રકરણમાં આપણે જાવાની મૂળભૂત બાબતો વિશે ચર્ચા કરી. જાવામાં પ્રાથમિક ડેટા પ્રકારો આઠ પ્રકારના છે. જાવામાં character એ 2 બાઇટનો યુનિકોડ અક્ષર છે. જાવામાં અનેક પ્રકારનાં લિટરલ વપરાય છે. ન્યુમરિક લિટરલ પૂર્વનિર્ધારિત રીતે double પ્રકારનાં છે. C ભાષા જેવા જ કન્ટ્રોલ-સ્ટ્રક્ચર જેમકે if, switch, for લૂપ, while લૂપ અને do...while લૂપ ઉપલબ્ધ છે. છગડિયા કૌંસ { }ની વચ્ચે વિધાનો લખીને બ્લોકનો નિર્દેશ કરી શકાય છે. ચલના કાર્યક્ષેત્રની મર્યાદા (scope) તે બ્લોક પૂરતી જ છે જ્યાં તે વ્યાખ્યાયિત કરેલા છે.

## સ્વાધ્યાય

1. જાવામાં character ડેટાપ્રકારનું કદ કેટલું હોય છે ?
2. જાવામાં int અને long ડેટાપ્રકાર કેટલી બાઇટ રોકે છે તે જણાવો.
3. જાવામાં ઉપલબ્ધ if અને switch વિધાન સમજાવો.
4. જાવામાં ઉપલબ્ધ વિવિધ પુનરાવર્તિત માળખાં વિશે ચર્ચા કરો.
5. જાવામાં બ્લોક અથવા લૂપ માટે લેબલ કઈ રીતે વાપરી શકાય ?

**6.** નીચેનામાંથી ખરા વિકલ્પની પસંદગી કરો :

(1) જાવામાં કેટલી પ્રાથમિક (primitive) ડેટાટાઇપ હોય છે ?

(a) 2 (b) 4 (c) 8 (d) 16

(2) અપૂર્ણાંક લિટરલનો પૂર્વનિર્ધારિત ડેટાપ્રકાર ક્યો હોય છે ?

(a) int (b) long (c) float (d) double

(3) જાવામાં char ડેટાપ્રકાર માટે ક્યો કૅરેક્ટરસેટ (characterset) વપરાય છે ?

(a) Unicode (b) ASCII

(c) EBCIDIC (d) ઉપરના તમામ વિકલ્પ

(4) નીચેનામાંથી ક્યું વિધાન કોઈ પણ ભૂલ વિના કંપાઇલ થશે ?

(a) for(;;){int i=7}; (b) while (1){int i=7};

(c) while (True){int i=7}; (d) ઉપરના તમામ વિકલ્પ

(5) નીચેનામાંથી માન્ય ચલના નામનો પહેલો અક્ષર કોણ ન હોય શકે ?

(a) અન્ડરસ્કોર (_) (b) અંક (c) અક્ષર (d) ડૉલર ($)

(6) નીચેનામાંથી ક્યો પ્રાથમિક ડેટાપ્રકાર નથી ?

(a) char (b) long (c) byte (d) string

(7) બુલિયન ડેટાપ્રકારની પૂર્વનિર્ધારિત કિંમત શું હોય છે ?

(a) નલ (null) (b) true (c) false (d) 0

(8) અંકગણિતની પદાવલી 7/2નું શું પરિણામ મળશે ?

(a) 3 (b) 3.5 (c) 1 (d) 0

(9) અંકગણિતની પદાવલી -7%2નું શું પરિણામ મળશે ?

(a) -3 (b) -1 (c) 1 (d) -3.5

(10) અંકગણિતની પદાવલી -7.5%2નું શું પરિણામ મળશે ?

(a) -3 (b) -1.5 (c) 1.5 (d) ભૂલ (Error)

## પ્રાયોગિક સ્વાધ્યાય

નીચે જણાવેલાં કાર્ય માટે જાવાપ્રોગ્રામ લખો :

1. સ્ટોરમાં સેલ સમયે ₹ 5000ની ખરીદી ઉપર 10% વળતર આપવામાં આવે છે. એક પ્રોગ્રામ લખો, જે 'purchase' નામના ચલને કોઈ કિંમત ઍસાઇન કરે અને પછી વળતરબાદની કિંમતની ગણતરી કરે. ખરીદકિંમત અને આપેલ વળતર પ્રદર્શિત કરો.

2. એક પ્રોગ્રામ લખો, જે ગ્રાહકની ઉંમર અને ચલચિત્રના શોના સમય (મૅટિની કે અન્ય શો) પ્રમાણે ચલચિત્રની ટિકિટની કિંમત શોધે. વયસ્ક માટે નૉર્મલ શો અને મૅટિની-શોની ટિકિટ અનુક્રમે ₹ 100 અને ₹ 50 છે. વયસ્ક એ છે, જેની ઉંમર 13 વર્ષ કરતાં વધારે છે. બાળકો માટે નૉર્મલ શો અને મૅટિની-શોની ટિકિટ અનુક્રમે ₹ 60 અને ₹ 40 છે. ઉંમર (age) અને શોના સમય (show time) માટે યોગ્ય ડેટાપ્રકારના ચલ ઘોષિત કરો, આ ચલને કિંમત ઍસાઇન કરો અને ઉંમર, શો-ટાઇમ અને ટિકિટ-કિંમત છાપો.

3. switch વિધાનનો ઉપયોગ કરીને મેળવેલા ગુણ પ્રમાણે ગ્રેડ પ્રદર્શિત કરતો પ્રોગ્રામ લખો.

4. એક બૅન્ક ગ્રાહકોને વાર્ષિક 12%ના સાદા વ્યાજથી ધિરાણ આપે છે. પૂર્ણસમયનું વ્યાજ શરૂઆતમાં વસૂલવામાં આવે છે. માસિક હપ્તાની ગણતરી 36 માસના સમયગાળા અને ધિરાણ-રકમ ₹ 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000 અને 100000 માટે કરો.

5. એક પ્રોગ્રામ લખો, જે પૂર્ણાંક સંખ્યાઓ 5થી શરૂ કરી, જેનું વર્ગમૂળ 50 કે તેથી ઓછું હોય, તેવી સંખ્યાઓનું વર્ગમૂળ છાપે.

# જાવામાં ક્લાસ અને ઑબ્જેક્ટ 8

આપણે અગાઉ શીખી ગયા છીએ કે ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગમાં ઑબ્જેક્ટ (object) અને ક્લાસ (class) એ પાયાના એકમો છે. જાવા ઑબ્જેક્ટ આધારિત ભાષા છે. આ પ્રકરણ જાવા-પ્રોગ્રામિંગમાં ક્લાસ અને ઑબ્જેક્ટ કઈ રીતે બનાવવા તે સમજાવે છે. આ પ્રકરણના અભ્યાસ પછી જાવામાં ઑબ્જેક્ટ શું છે અને ક્લાસ શું છે, તેનું સ્પષ્ટ ચિત્ર મેળવવા તમે સમર્થ હશો.

## પરિચય (Introduction)

ક્લાસ (class) ડેટા (જેને ઍટ્રિબ્યૂટ કહે છે) અને પ્રોગ્રામનો કોડ (વિધેય, જેને મેથડ કહે છે) બન્ને ધરાવે છે.

અગાઉના પ્રકરણમાં આપણે ક્લાસનો ઉપયોગ કરીને 'main' નામની એક જ મેથડ ધરાવતો પ્રોગ્રામ લખ્યો. જ્યારે ક્લાસનો અમલ કરવામાં આવ્યો, ત્યારે main મેથડ વાપરીને એક સામાન્ય પ્રોગ્રામની જેમ વિનિયોગ ચાલ્યો. આપણે આ ઉદાહરણોમાં કોઈ પણ ડેટા મેમ્બર (data members)નો ઉપયોગ કર્યો ન હતો.

જાવા પ્રોજેક્ટમાં ફક્ત એક જ ક્લાસ વાપરવો શક્ય છે, પણ મોટા વિનિયોગ માટે તે રીત સારી નથી. સૉફ્ટવેર ડિઝાઇન કરતા સમયે સંપૂર્ણ વિનિયોગને સરળ ઘટકોમાં વિભાજિત કરી દેવા જોઈએ કે જે તાર્કિક રીતે સંબંધિત કાર્યો કરે. આ દરેક ઘટક કે ભાગનો આપણે એક ક્લાસ બનાવી શકીએ.

હવે, આપણે 'Room' નામના ઉદાહરણથી જાવા પ્રોગ્રામિંગમાં ક્લાસ અને ઑબ્જેક્ટના ખ્યાલ વિશે સમજીએ. ઘર, છાત્રાલય, હોટેલ અથવા નિશાળના ખંડ (રૂમ)-ની લાક્ષણિકતાઓ એકસમાન હોય છે. દરેક ખંડ તેની પ્રોપર્ટી (property) જેમ કે length, height, number of windows, number of doors અને directionથી અજોડ રીતે ઓળખાય છે. સરળતા માટે આપણે અહીં length, width, height અને number of windows લઈશું.

હવે આપણે કોડલિસ્ટિંગ 8.1માં દર્શાવ્યા પ્રમાણે જાવાપ્રોગ્રામ લખીશું, જે length, width, height અને nWindows ઍટ્રિબ્યૂટ સાથેનો 'Room' નામનો ક્લાસ અને ત્રણ મેથડ બનાવશે. પહેલી મેથડ ઍટ્રિબ્યૂટને કિંમત ઍસાઇન કરશે. બીજી મેથડ ક્ષેત્રફળની ગણતરી કરશે અને ત્રીજી મેથડ તેના ઍટ્રિબ્યૂટ પ્રદર્શિત કરશે. તે પછી આપણે આ ક્લાસનો ઉપયોગ કરવા માટે કોડ લખીશું.

```
/* Class Room */
class Room
{
        float length, width, height;
        byte nWindows;

        void setAttr (float l, float w, float h, byte n)
        {
                length = l; width = w; height = h;
                nWindows = n;
        }  // end setAttr () method

      double area ( )  // area =  length * width
```

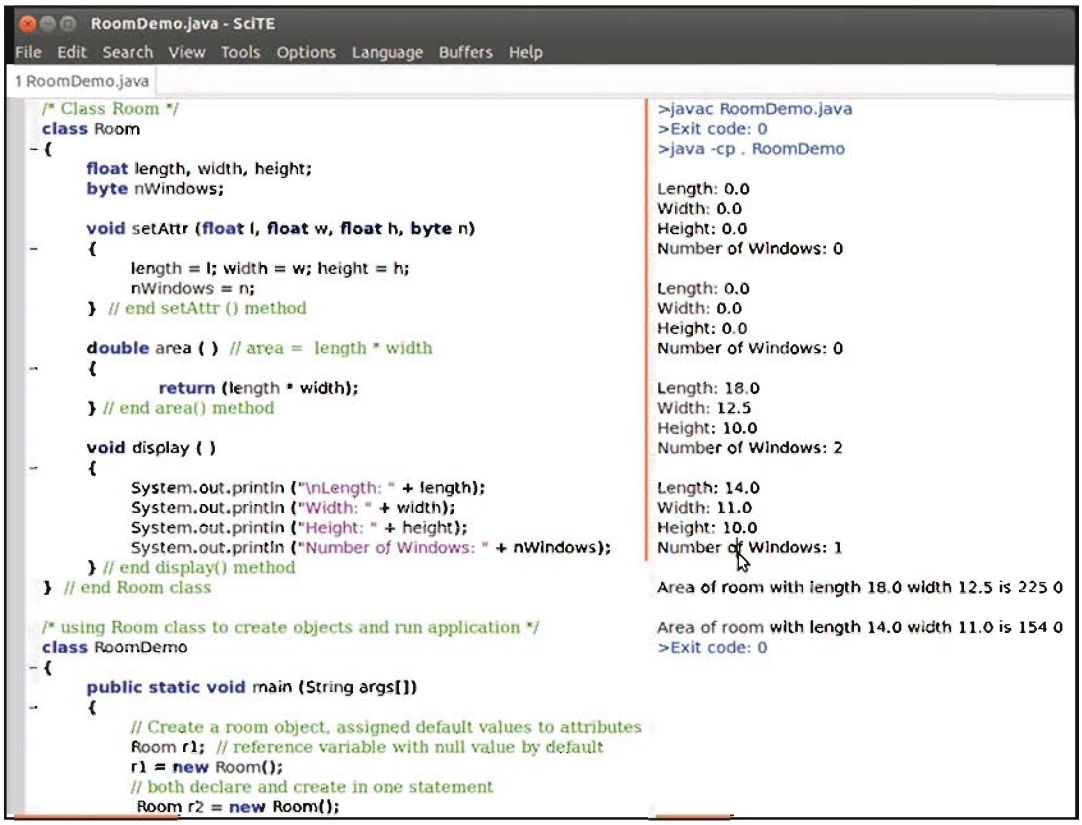
```
        {
                return (length * width);
        } // end area() method

        void display ( )
        {
                System.out.println ("\nLength: " + length);
                System.out.println ("Width: " + width);
                System.out.println ("Height: " + height);
                System.out.println ("Number of Windows: " + nWindows);
        } // end display() method
} // end Room class

/* using Room class to create objects and run application */
class RoomDemo
{
        public static void main (String args[])
        {
                // Create a room object, assigned default values to attributes
                Room r1;  // reference variable with null value by default
                r1 = new Room();
                // both declare and create in one statement
                 Room r2 = new Room();

            // Display two room objects with initial default values
                r1.display();
                r2.display();

                // Assign values of attributes of objects
                r1.setAttr (18, 12.5f, 10, (byte)2);
                r2.setAttr (14, 11, 10, (byte)1);

                // Display updated contents
                r1.display();
                r2.display();

                // Display area
                System.out.println ("\nArea of room with length " + r1.length
                   + " width " + r1.width +  " is " + r1.area());
                System.out.println ("\nArea of room with length " + r2.length
                   + " width " + r2.width +  " is " + r2.area());
        } // end main()
} // end RoomDemo
```

કોડલિસ્ટિંગ 8.1 : ક્લાસ અને ઑબ્જેક્ટ બનાવવા અને તેનો ઉપયોગ કરવો

અહીં, સૌપ્રથમ, 'Room' નામનો ક્લાસ બનાવવા માટે કોડ લખેલો છે. તે પછી 'Room' ક્લાસના ઑબ્જેક્ટ બનાવવા માટેનો કોડ લખેલો છે અને તેથી મેથડ (method) વાપરેલી છે. આ કાર્ય કરવા માટે main( ) મેથડ ધરાવતો એક બીજો ક્લાસ 'RoomDemo' બનાવેલો છે. સોર્સફાઇલમાં આ ક્લાસ કોઈ પણ ક્રમમાં હોઈ શકે.

અહીં, આપણે તાર્કિક રીતે સંબંધિત કાર્યો કરતા 'Room' ક્લાસ બનાવવા અને આ ક્લાસનો 'RoomDemo' નામના ક્લાસના વિનિયોગમાં ઉપયોગ એ બંને અલગ કરેલા છે.

જ્યારે એક પ્રોગ્રામમાં બે અથવા વધારે ક્લાસ હોય, ત્યારે ફક્ત એક જ ક્લાસ main( ) મેથડ ધરાવી શકે. આકૃતિ 8.1માં SciTE એડિટરમાં કોડનો અમલ દર્શાવ્યો છે.




**આકૃતિ 8.1 : એક કરતાં વધારે ક્લાસના ઉપયોગનું પ્રદર્શન કરવા માટે જાવાપ્રોગ્રામ**

આકૃતિ 8.1માં દર્શાવ્યા પ્રમાણે RoomDemo વિનિયોગના આપેલા ઉદાહરણમાં નીચે જણાવેલાં કાર્ય કરવામાં આવે છે.

- સૌપ્રથમ Room ક્લાસના બે ઑબ્જેક્ટ r1 અને r2 બનાવવામાં આવેલા છે. પૂર્વનિર્ધારિત કિંમત તેને આપવામાં આવે છે (initialized). (પૂર્વનિર્ધારિત રીતે આંકડાકીય કિંમત શૂન્ય છે.)
- display મેથડ વડે આ બંને ઑબ્જેક્ટની માહિતી પ્રદર્શિત કરવામાં આવે છે.
- ન્યુમરિક લિટરલ પ્રાચલો સાથે setAttr() મેથડનો અમલ (invoke) કરીને બંને ઑબ્જેક્ટ r1 અને r2 ના એટ્રિબ્યૂટમાં ફેરફાર થાય છે.
- બંને ઑબ્જેક્ટની નવી માહિતી પ્રદર્શિત કરવામાં આવે છે.

અંતમાં બંને Room ઑબ્જેક્ટનું ક્ષેત્રફળ પ્રદર્શિત કરવામાં આવે છે. અહીં, ક્ષેત્રફળ શોધવા માટે area( ) મેથડનો અમલ કરેલી છે.

## જાવામાં ક્લાસ (Class in Java)

આપણે અત્યાર સુધીમાં ક્લાસનો ઉપયોગ કરતો એક પ્રોગ્રામ લખેલો છે. હવે આપણે વાક્યરચના (syntax) અને ક્લાસ તથા ઑબ્જેક્ટ બનાવવા માટેની અન્ય માહિતી અને વિનિયોગમાં તેના ઉપયોગ વિશે શીખીએ.

ક્લાસ એ એકસમાન લાક્ષણિકતાઓ ધરાવતા અનેક ઑબ્જેક્ટનું એક માળખું (template) છે. ક્લાસ કોઈ ચોક્કસ ઑબ્જેક્ટના જૂથની લાક્ષણિકતાઓને સમાવે છે. ઉદાહરણ તરીકે, 'Room' એ બધા ખંડની સામાન્ય લાક્ષણિકતાઓ સાથેનું એક ટેમ્પ્લેટ છે.

જાવામાં class ચાવીરૂપ શબ્દ(keyword)નો ઉપયોગ કરીને ક્લાસને નીચે પ્રમાણે વ્યાખ્યાયિત કરવામાં આવે છે :

```
class <ClassName >
{
        <Variables>
        <Methods>
}
```

આપણે જાવામાં દરેક ક્લાસ બનાવીએ છીએ, તે સામાન્ય રીતે બે ઘટકોનો બનેલો હોય છે : ઍટ્રિબ્યૂટ (attribute) અને બિહેવ્યર (behaviour). ઍટ્રિબ્યૂટને ક્લાસમાં ચલથી વ્યાખ્યાયિત કરવામાં આવે છે. બિહેવ્યરને ક્લાસમાં મેથડથી વ્યાખ્યાયિત કરવામાં આવે છે. મેથડનો ઉપયોગ ઍટ્રિબ્યૂટને મેળવવામાં અને તેમાં ફેરફાર કરવા માટે થાય છે.

કોડલિસ્ટિંગ 8.1માં આપણે 'Room' નામનો ક્લાસ વ્યાખ્યાયિત કરેલો છે, જેના ઍટ્રિબ્યૂટ length, width, height અને nWindows નામના ચલથી વ્યાખ્યાયિત કરેલાં છે અને બિહેવ્યર setAttr(), display() અને area() નામની મેથડથી વ્યાખ્યાયિત કરેલા છે. 'Room' ક્લાસનો ઉપયોગ કરીને વિનિયોગ બનાવવા માટે એક અન્ય ક્લાસ બનાવ્યો છે.

## ઑબ્જેક્ટ બનાવવા (Creating Objects)

ક્લાસમાંથી ઑબ્જેક્ટ બનાવવા માટે નીચે જણાવેલાં પગલાંઓની જરૂર પડે છે :

- **Declaration :** <class name> <variable name> વાક્યરચના (syntax) સાથેનો ક્લાસ પ્રકારનો ચલ ઘોષિત કરવામાં આવે છે.
- **Instantiation :** (ઇન્સ્ટન્સિએશન - દૃષ્ટાંતીકરણ) મેમરી ફાળવીને ઑબ્જેક્ટ બનાવવા માટે **new** ચાવીરૂપ શબ્દ વપરાય છે.
- **Initialization :** નવા બનાવેલા ઑબ્જેક્ટને કિંમત આપવા માટે (initialize) કન્સ્ટ્રક્ટર (constructor) (એક વિશિષ્ટ પ્રકારની મેથડ) કૉલ કરવામાં આવે છે.

જાવામાં ક્લાસ એ int અને boolean જેવા સમાવિષ્ટ પ્રકારના જેવો જ એક પ્રકાર છે. આથી, ક્લાસના નામનો ઉપયોગ ઘોષિત વિધાનમાં ચલના પ્રકારનો ઉલ્લેખ કરવા, ઔપચારિક પ્રાચલનો પ્રકાર જણાવવા અને વિધેયની પરત કિંમતનો પ્રકાર જણાવવા થાય છે.

'Room' નામના ક્લાસનો ઑબ્જેક્ટ ઘોષિત કરવા માટે નીચેનું વિધાન વાપરો :

```
Room r1;
```

ચલને ઘોષિત કરવાથી ઑબ્જેક્ટ બનતો નથી. આ એક મહત્ત્વનો મુદ્દો યાદ રાખવો જોઈએ. જાવામાં કોઈ પણ ચલ ક્યારેય ઑબ્જેક્ટનો સંગ્રહ કરી શકતો નથી. ક્લાસ પ્રકારના ઉપયોગ વડે ઘોષિત કરેલો ચલ ફક્ત ઑબ્જેક્ટના નિર્દેશનો સંગ્રહ કરે છે. આથી, ક્લાસ પ્રકારના ચલને નિર્દેશિત ચલ (reference variable) પણ કહેવામાં આવે છે. અહીં, r1 એ નિર્દેશિત ચલ છે.

હવે પછીનું પગલું છે ઑબ્જેક્ટ બનાવવાનું. new ચાવીરૂપ શબ્દનો ઉપયોગ કરીને આપણે ઑબ્જેક્ટ બનાવી શકીએ. પ્રક્રિયક new ઑબ્જેક્ટ માટેની મેમરી ફાળવશે અને ભવિષ્યમાં તેનો ઉપયોગ કરી શકાય તે માટે તે ઑબ્જેક્ટનો સ્થાનાંક (address) પરત કરશે. રેફરન્સ (reference) એ મેમરીનો સ્થાનાંક છે, જ્યાં ઑબ્જેક્ટનો સંગ્રહ કરેલો છે. હકીકતમાં, મેમરીનો ખાસ ભાગ કે જેને હીપ (heap) કહેવામાં આવે છે, જ્યાં ઑબ્જેક્ટ રાખવામાં આવે છે.

જ્યારે ઑબ્જેક્ટ બનાવવામાં આવે છે ત્યારે, મેમરીની ફાળવણી ઉપરાંત 'constructor' નામની વિશિષ્ટ મેથડ શરૂઆતના કાર્ય કરવા માટે અમલમાં મૂકવામાં આવે છે. આપણે કન્સ્ટ્રક્ટર (constructor) વિશે આ પ્રકરણમાં હવે પછી શીખીશું.

હવે, આપણે Roomના પ્રકારનો એક ઑબ્જેક્ટ બનાવીએ અને તેનો સ્થાનાંક ચલ r1 ને સોંપીએ (ઍસાઇન કરીએ) :

```
r1 = new Room();
```

અહીં કૌંસ અગત્યના છે, તેને છોડી દેશો નહીં. ચલ (આર્ગ્યુમેન્ટ) વગરના ખાલી કૌંસ પૂર્વનિર્ધારિત constructorને કૉલ કરશે. તે પૂર્વનિર્ધારિત કિંમતો વડે ઑબ્જેક્ટના ઍટ્રિબ્યૂટની પ્રારંભિક કિંમત તેમાં મૂકશે (initializes). કૌંસમાં આર્ગ્યુમેન્ટની કિંમતો પણ હોઈ શકે કે જે ચલની પ્રારંભિક કિંમત નક્કી કરે. આ વપરાશકર્તા નિર્મિત કન્સ્ટ્રક્ટરનો ઉપયોગ કરીને શક્ય છે.

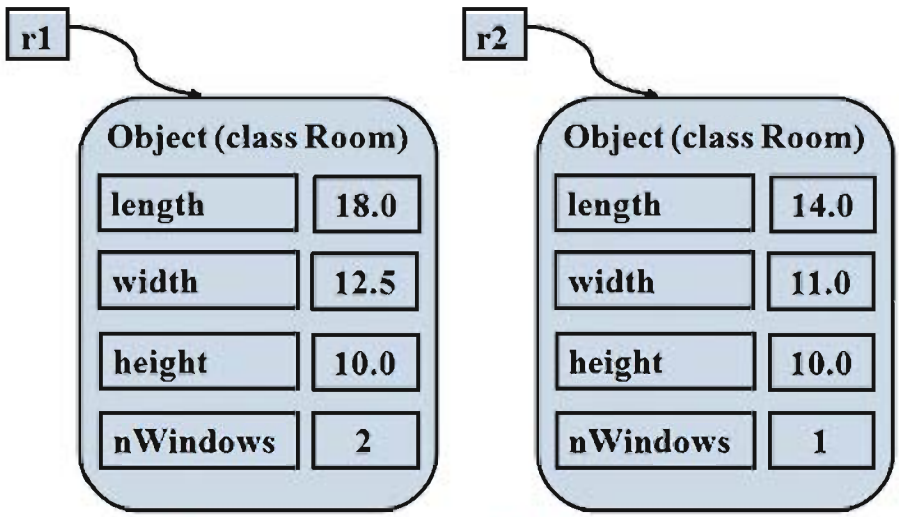
જ્યારે "r1=new Room();" વિધાનનો અમલ થાય છે, ત્યારે યાદ રાખો કે ઑબ્જેક્ટનો ચલ r1માં સંગ્રહ થતો નથી. ચલ r1 ઑબ્જેક્ટનો ફક્ત સ્થાનાંક (ઍડ્રેસ) જ ધરાવે છે.

ઑબ્જેક્ટ ઘોષિત કરવા અને બનાવવા માટે ઉપર જણાવેલાં બંને પગલાંને નીચે જણાવ્યા પ્રમાણે એક વિધાનમાં ભેગાં કરી શકાય છે :

```
Room r2 = new Room();
```

અહીં, ચલ r2 મેમરી સ્થાનાંક ધરાવે છે, જ્યાં નવો ઑબ્જેક્ટ બનાવેલો છે.

અહીં તે પણ નોંધવું જોઈએ કે ક્લાસ ફક્ત ચલનો પ્રકાર નક્કી કરે છે. ડેટા હકીકતમાં વ્યક્તિગત ઑબ્જેક્ટની અંદર હોય છે અને ક્લાસની અંદર નહીં. આ રીતે, દરેક ઑબ્જેક્ટને પોતાનો ડેટાનો સેટ હોય છે. કોડ લિસ્ટિંગ 8.1માં, આપણે new પ્રક્રિયકનો ઉપયોગ કરીને બે ઑબ્જેક્ટ r1 અને r2 બનાવેલા છે. આકૃતિ 8.2માં દર્શાવ્યા પ્રમાણે આ બંને ઑબ્જેક્ટને તેમની ડેટાની કિંમતને રાખવા માટે અલગ-અલગ મેમરીની ફાળવણી કરવામાં આવેલી છે.




**આકૃતિ 8.2 : Room ક્લાસના બે ઇન્સ્ટન્સ (instance)**

જાવામાં જ્યારે ઑબ્જેક્ટની પછી જરૂર રહેતી નથી, ત્યારે તે મેમરી ફરી વાપરવા માટે છૂટી કરવામાં આવે છે. જાવામાં ગાર્બેજ કલેક્ટર (garbage collector) હોય છે કે જે વણવપરાયેલા ઑબ્જેક્ટને શોધે છે અને તે આ ઑબ્જેક્ટ દ્વારા વપરાયેલી મેમરી પાછી મેળવે છે (એટલે કે છૂટી કરે છે). આપણે મેમરી મુક્ત કરવા માટે બાહ્ય રીતે કશું કરવાની જરૂર નથી.

ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગ (OOP) ભાષામાં ઑબ્જેક્ટ બનાવવાની ક્રિયાને **ઑબ્જેક્ટ ઇન્સ્ટન્શિએશન (object instantiation)** પણ કહેવામાં આવે છે. ઑબ્જેક્ટ માટે ડેટાનો સંગ્રહ કરવા માટે મેમરી ફાળવીને ઑબ્જેક્ટનો ઇન્સ્ટન્સ (instance) નિર્મિત કરવામાં આવે છે. કોઈ ક્લાસનો જે ઑબ્જેક્ટ ભાગ હોય, તેને ક્લાસનો **ઇન્સ્ટન્સ (instance)** કહેવામાં આવે છે.

આ રીતે, ક્લાસનો ઇન્સ્ટન્સ એ હકીકતમાં ઑબ્જેક્ટ માટેનો બીજો શબ્દ છે. ક્લાસ એ ઑબ્જેક્ટનું અમૂર્ત (abstract) સ્વરૂપ છે, જ્યારે ઇન્સ્ટન્સ એ મૂર્ત કે સાકાર સ્વરૂપ છે. હકીકતમાં, OOP ભાષામાં ઇન્સ્ટન્સ અને ઑબ્જેક્ટ શબ્દો ઘણી વખત એકબીજાની અદલાબદલીમાં વપરાય છે.

ક્લાસનો દરેક ઇન્સ્ટન્સ ક્લાસમાં ઘોષિત કરેલા ચલના ઍટ્રિબ્યૂટ માટે અલગ-અલગ કિંમત ધરાવી શકે છે. આ પ્રકારના ચલનો **ઇન્સ્ટન્સ વેરિયેબલ** (instance variable) તરીકે ઉલ્લેખ કરવામાં આવે છે. ઑબ્જેક્ટ બનાવતા સમયે ઇન્સ્ટન્સ વેરિયેબલનું નિર્માણ થાય છે અને ઑબ્જેક્ટના સંપૂર્ણ અસ્તિત્વ સુધી તે રહે છે.

**ઇન્સ્ટન્સ વેરિયેબલ** ઑબ્જેક્ટનાં ઍટ્રિબ્યૂટ વ્યાખ્યાયિત કરે છે. ક્લાસ ઍટ્રિબ્યૂટનો પ્રકાર વ્યાખ્યાયિત કરે છે. દરેક ઇન્સ્ટન્સ તે ઍટ્રિબ્યૂટની પોતાની કિંમતનો સંગ્રહ કરે છે.

ઑબ્જેક્ટના બિહેવ્યરને વ્યાખ્યાયિત કરવા માટે આપણે મેથડ બનાવીએ છીએ. જાવામાં મેથડ ફક્ત ક્લાસની અંદર જ વ્યાખ્યાયિત કરી શકાય છે. આ મેથડ ઇન્સ્ટન્સ વેરિયેબલની કિંમત મેળવવા અથવા બદલવા માટે ઑબ્જેક્ટનો ઉપયોગ કરીને ઇન્વોક (invoke) કરી શકાય છે. આવી મેથડ **ઇન્સ્ટન્સ મેથડ** તરીકે ઓળખાય છે.

ઇન્સ્ટન્સ મેથડ ઑબ્જેક્ટનું બિહેવ્યર વ્યાખ્યાયિત કરવા માટે વપરાય છે. મેથડ ઇન્વોક કરવી એ ઑબ્જેક્ટને કોઈ કાર્ય કરવા માટેનો હુકમ છે.

કોડલિસ્ટિંગ 8.1માં આપણે Room નામનો ક્લાસ length, width, height અને nWindows નામના ઇન્સ્ટન્સ વેરિયેબલ અને setAttr(), display() તથા area() નામની ઇન્સ્ટન્સ મેથડ સાથે વ્યાખ્યાયિત કરેલો છે.

સારાંશ રૂપે,

- new કી વર્ડથી ઑબ્જેક્ટ બનાવી શકાય છે.
- new કી વર્ડ ક્લાસના ઇન્સ્ટન્સને રજૂ કરતો ઑબ્જેક્ટનો નિર્દેશ પરત કરે છે.
- ક્લાસના બધા ઇન્સ્ટન્સની હીપ (heap) તરીકે ઓળખાતા ડેટા સ્ટ્રક્ચરમાં મેમરીની ફાળવણી થાય છે.
- દરેક ઑબ્જેક્ટ ઇન્સ્ટન્સનો પોતાનો ડેટા સેટ હોય છે.

### ઇન્સ્ટન્સ વેરિયેબલ ઍક્સેસ કરવા (કાર્ય કરવા માટે મેળવવા) અને ઇન્સ્ટન્સ મેથડ કોલ કરવી (Accessing instance variables and calling instance methods)

ઇન્સ્ટન્સ વેરિયેબલ અને ઇન્સ્ટન્સ મેથડ ઑબ્જેક્ટ દ્વારા મેળવવામાં (access) આવે છે. તેનો નિર્દેશ નીચે જણાવ્યા પ્રમાણે ડોટ પ્રક્રિયક (.) દ્વારા કરી શકાય છે :

<object refernce>.<instance variable or method>

ઉદાહરણ તરીકે, આપણે પ્રોગ્રામમાં Room r1ની lengthનો ઉલ્લેખ r1.length વાપરીને કરી શકીએ છીએ અને કોડલિસ્ટિંગ 8.1માં દર્શાવ્યા પ્રમાણે r1.display( ) મેથડને ઇન્વોક કરી Room r1ની ઍટ્રિબ્યૂટની કિંમતોને પ્રદર્શિત કરી શકીએ છીએ. અહીં એ નોંધ કરશો કે ડોટ (.) એ એક પ્રક્રિયક છે અને ડોટ પ્રક્રિયકની સહચારિતા (એસોસિએટિવિટી) ડાબી બાજુથી જમણી બાજુ હોય છે.

અહીં એ યાદ રાખવું જોઈએ કે પ્રોગ્રામમાં કોઈ પણ જગ્યાએથી ડેટા આ રીતે સીધો મેળવવાથી સુરક્ષિત હોવો જોઈએ. આ જાતનું રક્ષણ ઍક્સેસ મોડીફાયર (access modifier) વડે શક્ય છે, જેની ચર્ચા આપણે હવે પછી કરીશું.

જ્યારે આપણે એક જ ક્લાસની મેથડની અંદર ઇન્સ્ટન્સ વેરિયેબલ વાપરીએ છીએ, ત્યારે ડોટ (.) વાપરવાની કોઈ જરૂર નથી. ઉદાહરણ તરીકે, કોડલિસ્ટિંગ 8.1માં display મેથડમાં ડોટપ્રક્રિયકના ઉપયોગ વિના ઇન્સ્ટન્સ વેરિયેબલને ઍક્સેસ કરેલાં છે. અહીં એ શક્ય છે, કારણકે રેફરન્સ વેરિયેબલ r1નો ઉપયોગ કરીને મેથડ ઇન્વોક કરેલી છે અને આ રીતે r1 ઉપર સંગ્રહ કરેલા ઑબ્જેક્ટના અનુસંધાને ઉલ્લેખ કરેલાં ઇન્સ્ટન્સ વેરિયેબલ માનવામાં આવે છે.

આપણે અગાઉ શીખ્યા તે પ્રમાણે જ્યારે આપણે ક્લાસનો ઉપયોગ કરીને ફક્ત ઑબ્જેક્ટ ઘોષિત કરીએ છીએ, ત્યારે ઑબ્જેક્ટ

બનતો નથી. આ કિસ્સામાં રેફરન્સ વેરિયેબલ કોઈ ઑબ્જેક્ટનો નિર્દેશ કરતો નથી અને તેની પ્રારંભિક કિંમત પૂર્વનિર્ધારિત રીતે નલ (null) હોય છે. આ પ્રકારના નલ રેફરન્સ અથવા નલ પૉઇન્ટરનો ઉપયોગ અમાન્ય છે અને આથી નલ રેફરન્સ સાથેના ઇન્સ્ટન્સ વેરિયેબલ કે ઇન્વોકિંગ મેથડ ભૂલ (error) આપશે. નીચે આપેલો કોડ વાપરી જુઓ :

```
Room r1; // null value assigned to reference variable by default

System.out.println (r1.length); // illegal

r1.display();  // illegal
```

## ક્લાસ-વેરિયેબલ અને ક્લાસ-મેથડ (Class Variables and Class Methods)

આપણે અગાઉ ચર્ચા કરી તે પ્રમાણે જ્યારે new કી વર્ડનો ઉપયોગ કરીને ઑબ્જેક્ટ બનાવવામાં આવે છે ત્યારે તેના એટ્રિબ્યૂટની કિંમતોનો સંગ્રહ કરવા માટે હીપ (heap) વિસ્તારમાંથી મેમરીની ફાળવણી કરવામાં આવે છે. આ રીતે દરેક ઑબ્જેક્ટનાં પોતાના ઇન્સ્ટન્સ વેરિયેબલ હોય છે જે મેમરીમાં અલગ-અલગ જગ્યા રોકે છે.

હવે ધારો કે આપણે અત્યાર સુધીમાં બનાવેલા બધા Room ઑબ્જેક્ટની windowsની કુલ સંખ્યા આપણને જોઈએ છે. આ કિંમતનો સંગ્રહ કરવા માટે દરેક ક્લાસ દીઠ આપણે ફક્ત એક ચલની જરૂર પડશે. આ ચલને દરેક ઑબ્જેક્ટના એટ્રિબ્યૂટ તરીકે રાખવો એ અર્થહીન છે. હકીકતમાં તે ઑબ્જેક્ટનો ઍટ્રિબ્યૂટ નથી પણ તે ક્લાસનો ઍટ્રિબ્યૂટ છે.

આથી જ્યારે કેટલાક ચલની જરૂર હોય અને તે એક જ ક્લાસના બધા ઑબ્જેક્ટ ઇન્સ્ટન્સ વચ્ચે વાપરવાના હોય, તો દરેક ક્લાસ દીઠ ચલને મેમરી ફક્ત એક વાર જ ફાળવવી જોઈએ. આનો અર્થ એ થાય કે ચલ ક્લાસનો ભાગ છે અને ઑબ્જેક્ટનો નહીં. આવા ચલ ડેટાના પ્રકાર પહેલાં static કી વર્ડ વાપરી ને ક્લાસમાં ઘોષિત કરવામાં આવે છે અને આ સ્ટેટિક વેરિયેબલ (static variables) ક્લાસ વેરિયેબલ (class variable) કહેવાય છે.

ઇન્સ્ટન્સ વેરિયેબલની કિંમત ઇન્સ્ટન્સ (ઑબ્જેક્ટ માટે ફાળવવામાં આવેલી મેમરી)માં સંગ્રહ કરવામાં આવે છે, જ્યારે ક્લાસ-વેરિયેબલની કિંમત ક્લાસના પોતાનામાં જ સંગ્રહ કરે છે.

ઇન્સ્ટન્સ વેરિયેબલ સાથેના ક્લાસમાં નીચેનું વિધાન ઉમેરીને totWindows નામનો ક્લાસ-વેરિયેબલ ઘોષિત કરીએ :

```
static int totWindows;
```

સ્ટેટિક વેરિયેબલને ક્લાસના ઇન્સ્ટન્સ બનાવ્યા વિના મેળવી શકાય છે. (ઍક્સેસ કરી શકાય છે.) ઉદાહરણ તરીકે, 'Room' નામના ક્લાસમાં કોઈ પણ ઑબ્જેક્ટ બનાવ્યા વિના totwindowsની કિંમત પ્રદર્શિત કરવા આપણે પ્રયત્ન કરીશું, તો તે 0 પ્રદર્શિત કરશે.

ક્લાસ વેરિયેબલની ક્લાસમેથડ મેથડ ડેફિનેશન (method definition) પહેલાં static ચાવીરૂપ શબ્દ લખીને ઘોષિત કરી શકાય છે. કુલ બારીની સંખ્યા પ્રદર્શિત કરવા 'Room' ક્લાસની નીચેની મેથડ પ્રયત્ન કરી જુઓ.

```
static void displayTotalWindows()
{
    System.out.println("Total Windows: " +totWindows);
}
```

ક્લાસની બહાર <classname>.< class variable/method name>નો ઉપયોગ કરીને ક્લાસ વેરિયેબલ અને ક્લાસ મેથડ આપણે વાપરી શકીએ છીએ.

ઉદાહરણ તરીકે : Room.totWindows, Room.displayTotalWindows()

અહીં એ નોંધ કરશો કે આકૃતિ 8.3માં કોડલિસ્ટિંગમાં જણાવ્યા પ્રમાણે ક્લાસના નામ વિના એક જ ક્લાસની મેથડનો ઉપયોગ કરી ક્લાસના સભ્યો (વેરિયેબલ અને મેથડ)નો ઉલ્લેખ કરી શકાય છે.

RoomDemoStatic.java - SciTE

File Edit Search View Tools Options Language Buffers Help

1 RoomDemoStatic.java

```
/* Class Room */
class Room
{
        float length, width, height;
        byte nWindows;
        static int totWindows;  // class variable

        void setAttr (float l, float w, float h, byte n)
        {
                setWindows(n);
                length = l; width = w; height = h;
        }  // end setAttr () method

        void setWindows ( byte n)
        {
                totWindows = totWindows - nWindows + n;
                nWindows = n;
        }
        double area ( )  // area =  length * width
        {    return (length * width);      } // end area() method

        void display ( )
        {
                System.out.println ("\nLength: " + length + "\nWidth: " + width);
                System.out.println ("Height: " + height);
                System.out.println ("Number of Windows: " + nWindows);
        } // end display() method
}  // end Room class

/* using Room class to create objects and run application */
class RoomDemoStatic
{
        public static void main (String args[])
        {
                Room r1 = new Room();   Room r2 = new Room();

                r1.setAttr (18, 12.5f, 10, (byte)2);   r1.display();
                r2.setAttr (14, 11, 10, (byte)1);   r2.display();
                System.out.println("\nRoom2 Number of windows modified from 1 to 2");
                r2.setWindows((byte)2);
                System.out.println ("\nTotal number of Windows: " + Room.totWindows);
        }  // end main()
}  // end RoomDemo
```

**આકૃતિ 8.3 : ક્લાસ-વેરિયેબલ totWindowsનો ઉપયોગ**

આકૃતિ 8.3માં આપેલા કોડલિસ્ટિંગનો નિર્ગમ (આઉટપુટ) આકૃતિ 8.4માં આપેલો છે.

RoomDemoStatic.java - SciTE

File Edit Search View Tools Options Language Buffers Help

1 RoomDemoStatic.java

```
>javac RoomDemoStatic.java
>Exit code: 0
>java -cp . RoomDemoStatic

Length: 18.0
Width: 12.5
Height: 10.0
Number of Windows: 2

Length: 14.0
Width: 11.0
Height: 10.0
Number of Windows: 1

Room2 Number of windows modified from 1 to 2

Total number of Windows: 4
>Exit code: 0
```

**આકૃતિ 8.4 : આકૃતિ 8.3માં આપેલા કોડલિસ્ટિંગનો નિર્ગમ**

અહીં, આપણે એક વધારાની setWindows() નામની ઇન્સ્ટન્સ મેથડ લખી છે, જે બારીની કુલ સંખ્યામાં જૂની બારીની સંખ્યા બાદ કરીને અને નવી બારીની સંખ્યા ઉમેરીને સુધારો કરે છે. setWindows() મેથડ એ જ ક્લાસમાં વ્યાખ્યાયિત કરેલી છે. આથી ક્લાસના નામ વિના તે ક્લાસ વેરિયેબલને વાપરી શકે છે. main() મેથડમાં (જે RoomDemo સ્ટેટિક ક્લાસમાં વ્યાખ્યાયિત કરેલ છે) તે 'Room' ક્લાસની બહાર વ્યાખ્યાયિત કરેલી છે. આથી ક્લાસ વેરિયેબલ totWindowsને ક્લાસનું નામ વાપરીને Room.totWindowsથી ઍક્સેસ કરવું પડે.

ક્લાસ મેથડ એ જાતે ક્લાસથી વૈશ્વિક છે અને બીજા ક્લાસ કે ઑબ્જેક્ટને તે ઉપલબ્ધ છે. આથી, ક્લાસ મેથડ ક્લાસના ઇન્સ્ટન્સ અસ્તિત્વમાં છે કે નહીં તે ધ્યાનમાં લીધા વિના ગમે ત્યાં વાપરી શકાય છે.

હવે, પ્રશ્ન એ છે કે આપણે ક્લાસ મેથડ ક્યારે વાપરવી જોઈએ ? જે મેથડ કોઈ ચોક્કસ ઑબ્જેક્ટ ઉપર કાર્ય કરે અથવા ઑબ્જેક્ટને અસર કરે, તો તે ઇન્સ્ટન્સ મેથડ તરીકે વ્યાખ્યાયિત કરવી જોઈએ. એ મેથડ કે જે કેટલીક સામાન્ય યુટીલિટી પૂરી પાડે પણ ક્લાસના ઇન્સ્ટન્સ ઉપર સીધી અસર ન કરે, તેને ક્લાસ મેથડ તરીકે ઘોષિત કરવી વધુ સારી છે.

ઉદાહરણ તરીકે, આપેલી કોઈ સંખ્યા અવિભાજ્ય (prime) છે કે નહીં તે નક્કી કરતું વિધેય લો. આ વિધેયને ક્લાસ મેથડ તરીકે વ્યાખ્યાયિત કરવું જોઈએ. આકૃતિ 8.5માં કોડલિસ્ટિંગ અને પ્રોગ્રામનો આઉટપુટ દર્શાવ્યો છે.

primeClassMethod.java - SciTE

File Edit Search View Tools Options Language Buffers Help

1 primeClassMethod.java

```
// Static method (class method)
// isPrime (int) returns true if given integer is prime
class Prime
{
    static boolean isPrime (int n)
    {   //n>1 is prime if it is not divisible by any number except 1 and itself
        int i, last;

        if (n <= 1) return false;
        if (n < 4) return true;
        //if (n%2==0) return false;   // divisible by 2, so not prime

        last = (int) Math.sqrt(n);
        i=3;
        do
        {
            if (n%i == 0) return false; // n is divisible by i
            i = i + 2;      // no need to divide by even numbers
        } while (i<last) ;  // end of do...while loop

        return true;
    }  //end of method isPrime
} // end class primeFunc

class primeClassMethod
{
    public static void main (String[] s)
    {
        int i, n;

        System.out.println ("Prime numbers between 3 and 100:");
        for (n=3; n<100; n=n+2)
        {
            if (Prime.isPrime(n)) System.out.println(n);
        }
    } // end of main
} // end classs ClassMethodDemo
```

```
>javac primeClassMethod.java
>Exit code: 0
>java -cp . primeClassMethod
Prime numbers between 3 and 100:
3
5
7
11
13
17
19
23
25
29
31
35
37
41
43
47
49
53
59
61
67
71
73
79
83
89
97
>Exit code: 0
```

**આકૃતિ 8.5 : સ્ટેટિક મેથડનો ઉપયોગ કરીને આપેલી પૂર્ણાંક સંખ્યા પ્રાઇમ છે કે નહીં તે નક્કી કરવું.**

જાવાના રચયિતાઓએ આવી ક્લાસ મેથડ સાથેના પુષ્કળ પ્રમાણમાં સમાવિષ્ટ ક્લાસ પહેલેથી પૂરા પાડ્યા છે. પ્રકરણ 7માં આપણે Math ક્લાસનો કોઈ પણ ઑબ્જેક્ટ બનાવ્યા વિના sqrt() નામની સ્ટેટિક મેથડ વાપરેલી છે.

અહીં એ નોંધવું જોઈએ કે આપણે અત્યાર સુધી જે main() મેથડ વ્યાખ્યાયિત કરેલી છે તે એક ક્લાસ મેથડ પણ છે. આ કારણને લીધે જ આપણે main() મેથડ વ્યાખ્યાયિત કરતા સમયે static ચાવીરૂપ શબ્દનો ઉપયોગ કરીએ છીએ.

અહીં જોવા મળે છે તે પ્રમાણે ક્લાસના સ્થિર અને ચલિત (static and non-static) ભાગો અલગ-અલગ હેતુઓ પૂરા પાડે છે. સૉર્સ કોડમાંની static વ્યાખ્યાઓ એ વસ્તુઓ જણાવે છે, જે ક્લાસનો જાતે એક ભાગ છે, જ્યારે non-static વ્યાખ્યાઓ એ વસ્તુઓ જણાવે છે, જે ક્લાસના દરેક ઑબ્જેક્ટ ઇન્સ્ટન્સના ભાગ બનશે.

**યાદ રાખવા જેવા મુદ્દાઓ : (Points to Remember)**

- ક્લાસ-વેરિયેબલ અને ક્લાસ-મેથડને ક્લાસનેઇમ અથવા રેફરન્સ વેરિયેબલથી ઍક્સેસ કરી શકાય છે. તેની સુવાચ્યતા વધારવા માટે તેને ક્લાસનેઇમથી ઍક્સેસ કરવું વધારે સલાહભરેલું છે.
- ક્લાસ વેરિયેબલ અને ક્લાસ-મેથડને ઇન્સ્ટન્સ મેથડમાંથી પણ ઍક્સેસ કરી શકાય છે.
- ઇન્સ્ટન્સ વેરિયેબલ અને ઇન્સ્ટન્સ મેથડને ક્લાસ મેથડમાંથી ઍક્સેસ કરી ન શકાય કારણકે ક્લાસમેથડ કોઈ ઑબ્જેક્ટની હોતી નથી.

## ક્લાસમાં ઘોષિત કરેલા ચલ (વેરિયેબલ)નું વર્ગીકરણ (Classification of Variables Declared in a Class)

- **લોકલ વેરિયેબલ (Local Variables) :** જે વેરિયેબલ (ચલ) મેથડ કે બ્લૉકની અંદર વ્યાખ્યાયિત કરેલા હોય છે, તેને લોકલ વેરિયેબલ કહેવામાં આવે છે. મેથડના નિયમાનુસારના પ્રાચલો પણ લોકલ વેરિયેબલ છે. જ્યારે બ્લૉકની શરૂઆત થાય છે, ત્યારે તેનું નિર્માણ થાય છે અને જ્યારે મેથડ કે બ્લૉકનો અંત આવે છે, ત્યારે તેનો નાશ થાય છે. લોકલ વેરિયેબલની પૂર્વનિર્ધારિત કિંમતોથી શરૂઆત (initialized) થતી નથી.
- **ઇન્સ્ટન્સ વેરિયેબલ (Instance Variables) :** ઇન્સ્ટન્સ વેરિયેબલ એ કોઈ ક્લાસમાં પણ મેથડની બહાર વ્યાખ્યાયિત કરેલા ચલ (variable) છે. આ ચલ જ્યારે ઑબ્જેક્ટ નિર્મિત (instantiated) થાય છે, ત્યારે હીપ (heap) વિસ્તારમાંથી મેમરીની ફાળવણી થાય છે. ઇન્સ્ટન્સ વેરિયેબલને પૂર્વનિર્ધારિત કિંમતો આરંભમાં આપવામાં આવે છે (initialized).
- **ક્લાસ-વેરિયેબલ (Class Variables) :** ક્લાસ-વેરિયેબલને કોઈ ક્લાસની અંદર, મેથડની બહાર અને static ચાવીરૂપ શબ્દ સાથે વ્યાખ્યાયિત કરવામાં આવે છે. આ ચલને ક્લાસ દીઠ ફક્ત એક વાર મેમરીની ફાળવણી થાય છે અને તેના બધા ઑબ્જેક્ટ વડે તે વાપરી શકાય છે. ક્લાસ-વેરિયેબલને પૂર્વનિર્ધારિત કિંમતો આરંભમાં આપવામાં આવે છે (initizalized).

## પોલિમૉર્ફિઝમ (મેથડ ઓવરલોડિંગ) - Polymorphism (Method Overloading)

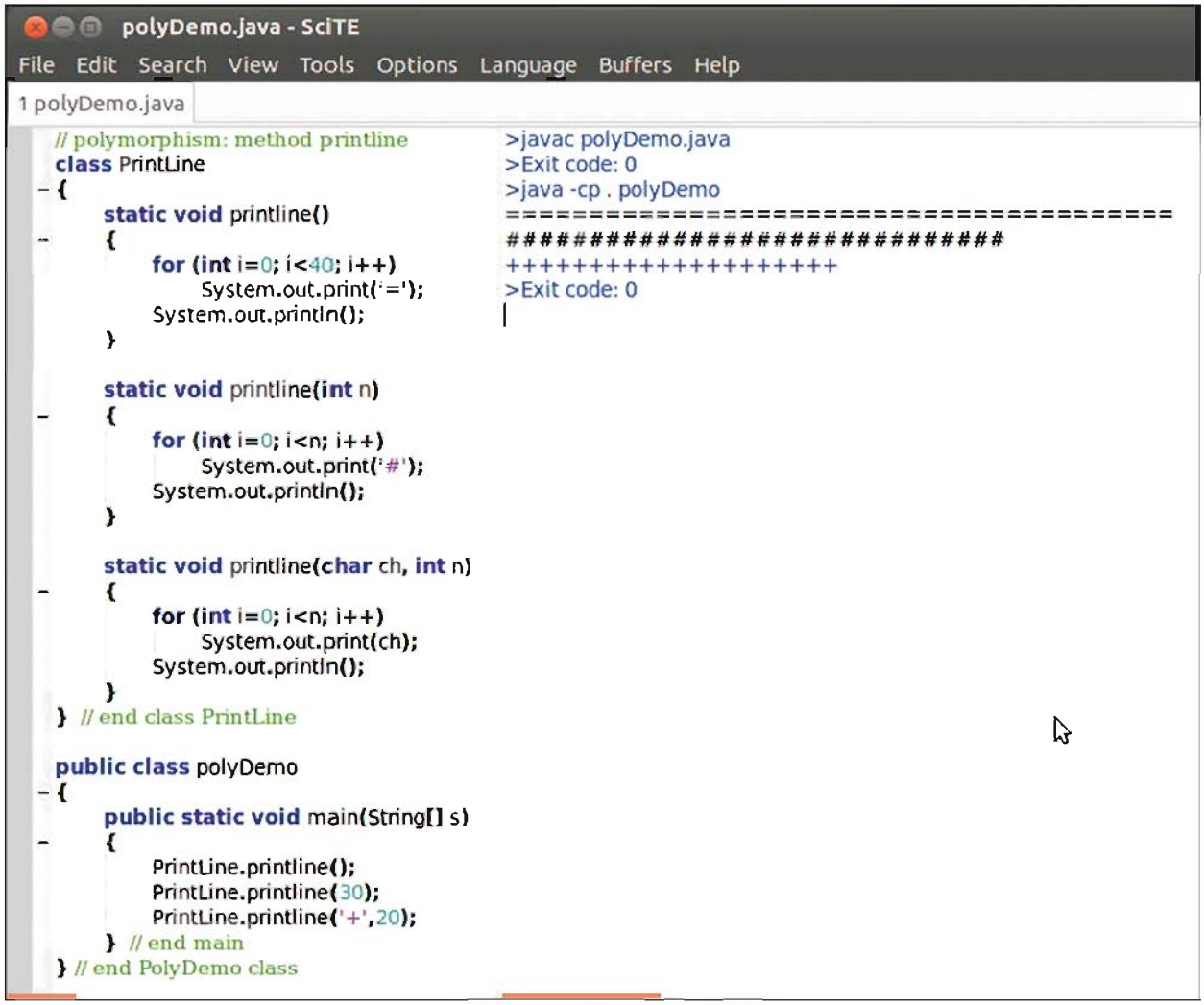
આપણે ઑબ્જેક્ટ બનાવવાનાં પ્રથમ બે પગલાં Declaration અને Instantiation વિશે અભ્યાસ કર્યો. ત્રીજું પગલું પ્રારંભિક કિંમત આપવાનું (Initialization) છે. આ માટે કન્સ્ટ્રક્ટરના ઉપયોગની જરૂર પડે છે. આપણે કન્સ્ટ્રક્ટર વિશે શીખીએ તે પહેલાં આપણે જાવામાં પોલિમૉર્ફિઝમનું અમલીકરણ કઈ રીતે કરી શકાય તે બાબત જોઈએ.

પોલિમૉર્ફિઝમ શબ્દનો અર્થ 'અનેક સ્વરૂપ' કે 'બહુરૂપતા' થાય છે; એટલે કે એક જ નામ સાથે જુદી-જુદી મેથડ. જાવામાં એક જ નામ ધરાવતી પણ અલગ-અલગ સિગ્નેચર (signature) સાથેની વિવિધ મેથડ હોઈ શકે. આને 'મેથડ ઓવરલોડિંગ (method overloading)' કહેવામાં આવે છે. મેથડની સિગ્નેચર એ મેથડનું નામ, પરત કિંમતનો પ્રકાર (ઑબ્જેક્ટ કે બેઇઝ પ્રકાર) અને પ્રાચલોની યાદીનો સમૂહ છે.

ઉદાહરણ તરીકે, બે પૂર્ણાંક સંખ્યામાંથી મહત્તમ, ત્રણ પૂર્ણાંક સંખ્યામાંથી મહત્તમ, ત્રણ ડબલ પ્રિઝિશન (double precision) ધરાવતી અપૂર્ણાંક સંખ્યામાંથી મહત્તમ કિંમત શોધવા માટે કોઈ વ્યક્તિને એકસરખા પ્રકારનું પણ જુદી-જુદી સંખ્યાઓ ઉપર કાર્ય કરવું પડે છે. આ પરિસ્થિતિમાં, જાવા એક સમાન નામ પણ અલગ અલગ પ્રાચલો સાથે મેથડ બનાવવાની સગવડ પૂરી પાડે છે. મહત્તમ સંખ્યા શોધવામાં કોઈ ઑબ્જેક્ટ બનાવવાની જરૂર નથી, આથી તેને સ્ટેટિક મેથડ તરીકે વ્યાખ્યાયિત કરી શકાય છે. ઉદાહરણ તરીકે :

```
static int max(int x, int y) {...}

static int max(int x, int y, int z) {...}

static double max(double x, double y, double z) {...}
```

તમે આકૃતિ 8.6માં આપેલા કોડલિસ્ટિંગના ઉદાહરણને સમજ્યા પછી ઉપર જણાવેલી રચના પ્રમાણેની max મેથડ વ્યાખ્યાયિત કરવાનો અને તેનો વિનિયોગમાં ઉપયોગ કરવાનો પ્રયત્ન કરી શકો. અહીં તે જણાવેલા પ્રાચલો પ્રમાણે લીટી છાપવા માટે પોલિમૉર્ફિઝમનો ઉપયોગ કરે છે. કોઈ પણ પ્રાચલ વિના printline() 40 વાર '=' અક્ષર, printline(int n) n વાર '#' અક્ષર જ્યારે printline(int n, char ch) n વાર જણાવેલો અક્ષર ch છાપે છે.




આકૃતિ 8.6 : મેથડ ઓવરલોડિંગ

## કન્સ્ટ્રક્ટર્સ (Constructors)

કન્સ્ટ્રક્ટર એ એક વિશિષ્ટ પ્રકારની મેથડ છે, જે નવા ઑબ્જેક્ટ નિર્મિત કરતાં સમયે ઇન્વોક કરવામાં આવે છે. કન્સ્ટ્રક્ટર કોઈ પણ કાર્ય કરી શકે પણ તે મુખ્યત્વે પ્રારંભિક કિંમતો આપવા માટે (initializing actions) રચવામાં આવે છે. આપણે અત્યાર સુધી વપરાશકર્તા વ્યાખ્યાયિત કન્સ્ટ્રક્ટર વિના ક્લાસનો ઉપયોગ કર્યો છે. દરેક ક્લાસના **રિફોલ્ટ કન્સ્ટ્રક્ટર** (default constructor) હોય છે; કોઈ વાર તેને ચલ વગરના કન્સ્ટ્રક્ટર તરીકે ઓળખવામાં આવે છે. રિફોલ્ટ કન્સ્ટ્રક્ટર કોઈ ચલ (argument) લેતા નથી. તે નવા બનાવેલા ઑબ્જેક્ટના ઍટ્રિબ્યૂટને તેના ડેટાપ્રકાર પ્રમાણે પૂર્વનિર્ધારિત કિંમતો આપે છે (initializes).

કન્સ્ટ્રક્ટર અન્ય સામાન્ય મેથડથી નીચે જણાવ્યા પ્રમાણે અલગ પડે છે :

- કન્સ્ટ્રક્ટરનું નામ ક્લાસના નામ સમાન જ હોવું જોઈએ.
- કન્સ્ટ્રક્ટરને પરતપ્રકાર (return type) હોતો નથી.

- જ્યારે નવો પ્રક્રિયક વાપરીને ઑબ્જેક્ટ બનાવવામાં આવે છે, ત્યારે જ કન્સ્ટ્રક્ટર અંતર્ગત રીતે (implicitly) ઇન્વોક કરવામાં આવે છે.
- કન્સ્ટ્રક્ટર પ્રોગ્રામમાં અન્ય કોઈ પણ જગ્યાએ સ્પષ્ટ રીતે (explicitly) ઇન્વોક કરવામાં આવતો નથી.

અન્ય મેથડની જેમ કન્સ્ટ્રક્ટરને પણ ઓવરલોડ (overload) કરી શકાય છે. તે અલગ-અલગ પ્રાચલોની યાદી સાથે શક્ય છે. ઉદાહરણ તરીકે, આપણે નીચે જણાવ્યા પ્રમાણે 'Room' ઑબ્જેક્ટના ઍટ્રિબ્યૂટને અલગ-અલગ પ્રાચલોથી પ્રારંભિક કિંમતો આપવા (initialize કરવા) માટે આપણા પોતાના કન્સ્ટ્રક્ટર લખીએ :

Room(float l, float w, float h, byte n) : lengthને l, width ને w, heightને h અને nWindowsને n કિંમત આપવા માટે (initialize કરવા).

Room(float l, float w) : lengthને l, widthને w, heightને 10 અને nWindowsને 1 કિંમત આપવા માટે.

આકૃતિ 8.7માં આપેલો પ્રોગ્રામ કન્સ્ટ્રક્ટરનો ઉપયોગ દર્શાવે છે.

RoomConstructorDemoA.java - SciTE

File Edit Search View Tools Options Language Buffers Help

1 RoomConstructorDemoA.java

```
/* Using Constructors */
class Room
{
    float length, width, height;
    byte nWindows;
    static int totWindows;  // class variable

    Room (float l, float w, float h, byte n)
    {
        length = l; width = w; height = h;
        nWindows = n;   totWindows+=n;
    }
    Room (float l, float w)
    {
        length = l; width = w; height = 10;
        nWindows = 1;     totWindows++;
    }

    double area ( )  // area =  length * width
    {    return (length * width);     } // end area() method
    void display ( )
    {
        System.out.println ("\nLength: " + length + "\nWidth: " + width);
        System.out.println ("Height: " + height);
        System.out.println ( "Windows: " + nWindows);
    } // end display() method
} // end Room class

class RoomConstructorDemoA  /* Application using Room */
{
    public static void main (String args[])
    {
        Room r1 = new Room(16, 12.5f);
        Room r2 = new Room(20, 14, 12, (byte)2);
        r1.display();  r2.display();
        System.out.println ("\nTotal number of Windows: " + Room.totWindows);
    } // end main()
} // end RoomConstructorDemoA
```

```
>javac RoomConstructorDemoA.java
>Exit code: 0
>java -cp . RoomConstructorDemoA

Length: 16.0
Width: 12.5
Height: 10.0
Windows: 1

Length: 20.0
Width: 14.0
Height: 12.0
Windows: 2

Total number of Windows: 3
>Exit code: 0
```

**આકૃતિ 8.7 : કન્સ્ટ્રક્ટરનો ઉપયોગ**

ક્લાસમાં વપરાશકર્તા વ્યાખ્યાયિત કન્સ્ટ્રક્ટરની ગેરહાજરીમાં ઑબ્જેક્ટ ચલ વગરના ડિફોલ્ટ કન્સ્ટ્રક્ટરનો ઉપયોગ કરીને બનાવવામાં આવે છે. પૂર્વનિર્ધારિત કિંમત ઍટ્રિબ્યૂટને આપવામાં આવે છે.

ક્લાસમાં વપરાશકર્તા વ્યાખ્યાયિત કન્સ્ટ્રક્ટરની હાજરીમાં ડિફોલ્ટ કન્સ્ટ્રક્ટર ઉપલબ્ધ હોતા નથી. ચલ વગરના કન્સ્ટ્રક્ટર સાથેનો ઑબ્જેક્ટ બનાવવાનો પ્રયત્ન કરવામાં આવે, તો કમ્પાઇલર એરર (error) પરત કરે છે. આ સમસ્યાનું નિરાકરણ લાવવા માટે આપણે નીચે જણાવ્યા પ્રમાણે વપરાશકર્તા દ્વારા વ્યાખ્યાયિત ચલ વગરનો કન્સ્ટ્રક્ટર પૂરો પાડવો જોઈએ : <classname> ( ) { };

આકૃતિ 8.7માં આપેલા કોડ લિસ્ટિંગ પ્રમાણે નીચે જણાવ્યા પ્રમાણે main મેથડમાં Room ઑબ્જેક્ટ બનાવવાનો પ્રયત્ન કરી જુઓ અને કમ્પાઇલેશન દરમિયાન દર્શાવાતી errorનું નિરીક્ષણ કરો : Room r3 = new Room();

આ error દૂર કરવા માટે વપરાશકર્તા વ્યાખ્યાયિત ચલ વગરનો કન્સ્ટ્રક્ટર 'Room () {};' ઉમેરો અને પછી તે પ્રોગ્રામનો અમલ કરો. હવે આકૃતિ 8.8માં આપેલા પ્રમાણે પ્રોગ્રામનો સફળ અમલ જુઓ.

RoomConstructorDemoB.java - SciTE

File Edit Search View Tools Options Language Buffers Help

1 RoomConstructorDemoB.java

```
/* Using Constructors */
class Room
{
    float length, width, height;
    byte nWindows;
    static int totWindows;  // class variable

    Room ( ) { }; // user-defined no-argument constructor
    Room (float l, float w, float h, byte n)
    {
        length = l; width = w; height = h;
        nWindows = n;   totWindows+=n;
    }
    Room (float l, float w)
    {
        length = l; width = w; height = 10;
        nWindows = 1;     totWindows++;
    }

    double area ( )  // area =  length * width
    {    return (length * width);     } // end area() method
    void display ( )
    {
        System.out.println ("\nLength: " + length + "\nWidth: " + width);
        System.out.println ("Height: " + height);
        System.out.println ( "Windows: " + nWindows);
    } // end display() method
} // end Room class

class RoomConstructorDemoB  /* Application using Room */
{
    public static void main (String args[])
    {
        Room r1 = new Room( );
        Room r2 = new Room(20, 14, 12, (byte)2);
        r1.display();  r2.display();
        System.out.println ("\nTotal number of Windows: " + Room.totWindows);
    } // end main()
} // end RoomConstructorDemoB
```

```
>javac RoomConstructorDemoB.java
>Exit code: 0
>java -cp . RoomConstructorDemoB

Length: 0.0
Width: 0.0
Height: 0.0
Windows: 0

Length: 20.0
Width: 14.0
Height: 12.0
Windows: 2

Total number of Windows: 2
>Exit code: 0
```

**આકૃતિ 8.8 : વપરાશકર્તા દ્વારા વ્યાખ્યાયિત ચલ વગરના કન્સ્ટ્રક્ટરનો ઉપયોગ**

## ઍક્સેસ કન્ટ્રોલ માટેનાં વિઝિબિલિટી મોડિફાયર (Visibility Modifiers for Access Control)

ઍક્સેસ કન્ટ્રોલ એ દૃશ્યતાની નિયંત્રણ બાબત છે. આથી, ઍક્સેસ મોડિફાયર **વિઝિબિલિટી મોડિફાયર** તરીકે પણ ઓળખાય છે. જો મેથડ અથવા વેરિયેબલ બીજા ક્લાસમાં દૃશ્યમાન (visible) હોય, તો જ અન્ય ક્લાસમાં તેનો ઉલ્લેખ કરી શકાય છે. આ પ્રકારના નિર્દેશથી મેથડ અથવા વેરિયેબલને સુરક્ષિત રાખવા માટે આપણે ચાર સ્તરનાં વિઝિબિલિટીનો ઉપયોગ જરૂરી સુરક્ષા પૂરી પાડવા માટે કરીએ છીએ.

સુરક્ષા કે પ્રોટેક્શન (protection)નાં ચાર p's પબ્લિક (public), પૅકેજ (package) (પૂર્વનિર્ધારિત સુરક્ષા), પ્રોટેક્ટેડ (protected) અને પ્રાઇવેટ (private) છે. પબ્લિક, પ્રોટેક્ટેડ અને પ્રાઇવેટ ઍક્સેસ મોડિફાયર ચલ કે મેથડના પ્રકાર પહેલાં વપરાય છે. જ્યારે કોઈ પણ મોડિફાયર વાપરેલો ન હોય, ત્યારે તે પૂર્વનિર્ધારિત પ્રકારની વિઝિબિલિટી પૅકેજ છે, જેનો ક્લાસમાં સમાવેશ થાય છે.

પૅકેજ (package) વિવિધ ક્લાસને વ્યવસ્થિત રૂપ આપવા માટે વપરાય છે. આ માટે સૉર્સફાઇલમાં પૅકેજ વિધાન સૌથી પ્રથમ કોમેન્ટ ન હોય અને બ્લૅન્ક લીટી ન હોય તે રીતે ઉમેરવામાં આવે છે. જ્યારે ફાઇલમાં પૅકેજવિધાન ન હોય ત્યારે ફાઇલમાં વ્યાખ્યાયિત ક્લાસને ડિફૉલ્ટ પૅકેજમાં રાખવામાં આવે છે. પૅકેજવિધાનની વાક્યરચના નીચે પ્રમાણે છે : package<packageName>;

આ પુસ્તકમાં આપણે ડિફોલ્ટ પૅકેજનો ઉપયોગ કરીશું. આપણા બધા પ્રોગ્રામમાં અત્યાર સુધી આપણે ઍક્સેસ મોડિફાયરનો ઉપયોગ કર્યો છે. કોષ્ટક 8.1માં ઍક્સેસ મોડિફાયર અને તેની વિઝિબિલિટી દર્શાવી છે.

| | | Type | | |
|---|---|---|---|---|
| **Access Modifier** | public | (default: package) | protected | private |
| **Visibility** | widest | → → → | → → | narrowest |

**કોષ્ટક 8.1 : ઍક્સેસ મોડિફાયરના પ્રકારો અને તેની વિઝિબિલિટી**

હવે આપણે ડિફોલ્ટ અને પ્રાઈવેટ મોડિફાયરનાં ઉદાહરણો જોઈશું.

### પબ્લિક (Public)

કોઈ પણ મેથડ કે ચલ જે ક્લાસમાં વ્યાખ્યાયિત કરેલા હોય તેમાં જ તે ઉપલબ્ધ (વિઝિબલ) હોય છે. જો આપણે એ ક્લાસની બહાર બધા ક્લાસમાં ઉપલબ્ધ બનાવવા ઇચ્છતા હોઈએ તો તે મેથડ અથવા ચલને પબ્લિક ઍક્સેસ માટે ઘોષિત કરો. આ સૌથી વધારે શક્ય ઍક્સેસ (access) છે. તે અન્ય પૅકેજમાં વ્યાખ્યાયિત ક્લાસને પણ વિઝિબિલિટી પૂરી પાડે છે. પબ્લિક ઍક્સેસ આપવા માટે ચલ કે મેથડના પ્રકાર પહેલાં public ઍક્સેસ મોડિફાયર વાપરો. ઉદાહરણ તરીકે,

```
public float length

public double area ( )
```

અહીં એ નોંધ કરશો કે public ચલ અને મેથડ બધી જ જગ્યાએ વિઝિબલ હોય છે અને આથી તે અન્ય સૉર્સફાઇલ અને પૅકેજમાંથી પણ ઍક્સેસ કરી શકાય છે.

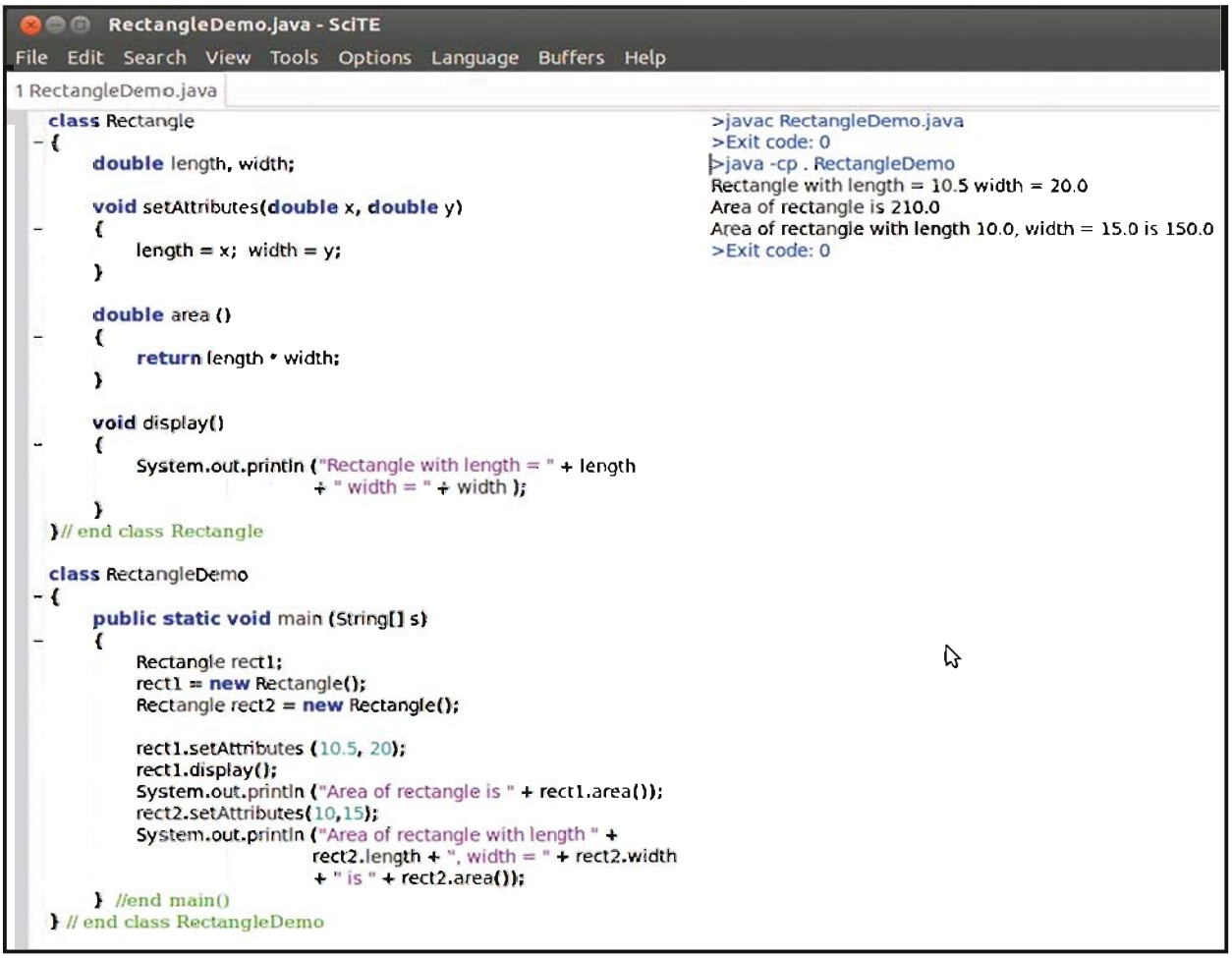
આપણે દરેક વ્યક્તિને ઉપલબ્ધ બનાવવા માટે main( ) મેથડ સાથે public ચાવીરૂપ શબ્દનો ઉપયોગ કર્યો છે.

```
public static void main(String[] args) { ... }
```

### પૅકેજ (કોઈ પણ મોડિફાયર વિના) Package (Without any Modifier)

આ બીજા સ્તરનો ઍક્સેસ (access) છે જેને કોઈ નિશ્ચિત નામ નથી. તે ઘોષિત કરવાના વિધાનમાં કોઈ પણ પ્રકારના ઍક્સેસ મોડિફાયરની ગેરહાજરી વડે જણાવવામાં આવે છે. આ ડિફોલ્ટ સ્તરનું પ્રોટેક્શન (રક્ષણ) છે. તેનો વિસ્તાર પબ્લિક ચલ કરતાં ઓછો છે. પૅકેજમાં બધી જગ્યાએથી ચલ કે મેથડને ઍક્સેસ કરી શકાય છે કે જ્યાં ક્લાસનો સમાવેશ થયેલો છે, પણ તે પૅકેજની બહાર નહીં. અહીં એ નોંધ કરશો કે જે સૉર્સફાઇલ package વિધાન વિનાની હશે, તે package સાથેની પૂર્વનિર્ધારિત ગણવામાં આવશે. આથી, અત્યાર સુધીના આપણા પ્રોગ્રામમાં તે public બરાબર જ છે.

આકૃતિ 8.9માં દર્શાવેલા પ્રોગ્રામનો સંદર્ભ લો. અહીં 'Rectangle' ક્લાસનાં બે ઍટ્રિબ્યૂટ છે : length અને width. તેને અનેક મેથડ પણ છે. આપણે કોઈ પણ મોડિફાયર ચાવીરૂપ શબ્દ વાપર્યો નથી, આથી તેનું પૂર્વનિર્ધારિત રીતે package પ્રોટેક્શન રહેશે. આ કારણને લીધે આ જ સૉર્સફાઇલમાં (પૂર્વનિર્ધારિત package) વ્યાખ્યાયિત અન્ય ક્લાસ 'Rectangle Demo'માં તે સીધેસીધા મેળવી શકાય છે (accessible).




**આકૃતિ 8.9 : પૂર્વનિર્ધારિત વિઝિબિલિટી મોડિફાયર, પૅકેજમાં સર્વત્ર ઉપલબ્ધ**

## પ્રોટેક્ટેડ (Protected)

આ સ્તરના પ્રોટેક્શનનો ઉપયોગ ફક્ત સબક્લાસને એક્સેસ કરવા માટે અથવા 'friend' તરીકે ઘોષિત કરેલી મેથડ સાથે સહિયારા ઉપયોગ માટે થાય છે. આથી, અગાઉ જણાવેલા બંને સ્તર કરતાં વિઝિબિલિટી ઓછી છે, પણ ચોથા સ્તરની 'private'માં પૂરી પાડવામાં આવેલી પૂર્ણ ગુપ્તતા (privacy) કરતાં વધુ છે.

પ્રોટેક્ટેડ પ્રોટેક્શનનો ઉપયોગ આપણે જ્યારે ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગનો ખ્યાલ ઇનહેરિટન્સ (inheritance) વાપરીશું, ત્યારે વધારે સુસંગત જણાશે.

## પ્રાઇવેટ (Private)

"private" કક્ષાનું પ્રોટેક્શન વાપરવાથી સૌથી ઉચ્ચ કક્ષાનું પ્રોટેક્શન મેળવી શકાય છે. તે સૌથી ઓછી દૃશ્યતા પૂરી પાડે છે. પ્રાઇવેટ મેથડ અને ચલને ક્લાસની અંદર જ વ્યાખ્યાયિત મેથડ દ્વારા સીધેસીધા એક્સેસ કરી શકાય છે. તે અન્ય કોઈ પણ ક્લાસ દ્વારા જોઈ શકાતા નથી.

આ બહુ જ પ્રતિબંધક જણાય, પણ હકીકતમાં તે સામાન્ય રીતે વપરાતા પ્રોટેક્શનનું સ્તર છે. તે ડેટા એનકૅપ્સ્યુલેશન (data encapsulation) પૂરું પાડે છે; દુનિયાની નજરમાંથી ડેટા છુપાવે છે અને તેની ખોટી રીતની ગણતરીને મર્યાદામાં રાખે છે. જે ઘટક સીધા કોઈ પણ સાથે તેના સબક્લાસના સમાવેશ સાથે સહિયારું ન હોય તે private છે.

ડેટા ઇનકૅપ્સ્યુલેશન પૂરો પાડવાનો સૌથી શ્રેષ્ઠ માર્ગ શક્ય એટલો વધારેમાં વધારે ડેટા private કરવાનો છે. તે તેની રચનાને તેના અમલીકરણની ક્રિયાથી અલગ કરે છે, અને તેનું કાર્ય કરવા માટે કોઈ ક્લાસને અન્ય ક્લાસની માહિતી જાણવાની જરૂરિયાતને ઓછામાં ઓછી કરે છે.

હવે, આપણે આકૃતિ 8.9માં આપેલા કોડલિસ્ટિંગમાં ફેરફાર કરીએ અને length તથા widthને private ઘોષિત કરીએ. private સભ્યો તે જ ક્લાસમાં સીધેસીધા ઉપલબ્ધ હોવાથી જ્યારે આપણે area() અને display() મેથડમાં તેનો ઉપયોગ કરીશું, ત્યારે તે કોઈ પણ પ્રકારની error નહીં બતાવે. જ્યારે તે RectangleDemo ક્લાસની main() મેથડમાં ઍક્સેસ કરવામાં આવશે, ત્યારે તે error દર્શાવશે.

સુધારેલો કોડ આકૃતિ 8.10માં દર્શાવ્યો છે. અહીં, આપણે private ઇન્સ્ટન્સ વેરિયેબલ ઉપરાંત કન્સ્ટ્રક્ટર વાપરેલા છે. નિર્ગમ વિભાગમાં error દર્શાવે છે તેની નોંધ કરો જે જણાવે છે કે 'length has private access in Rectangle'. આનો અર્થ એ થાય કે તે 'RectangleVisibility1' ક્લાસમાં ઍક્સેસ કરી ન શકાય.

RectangleVisibility1.java * SciTE

File Edit Search View Tools Options Language Buffers Help

1 RectangleVisibility1.java *

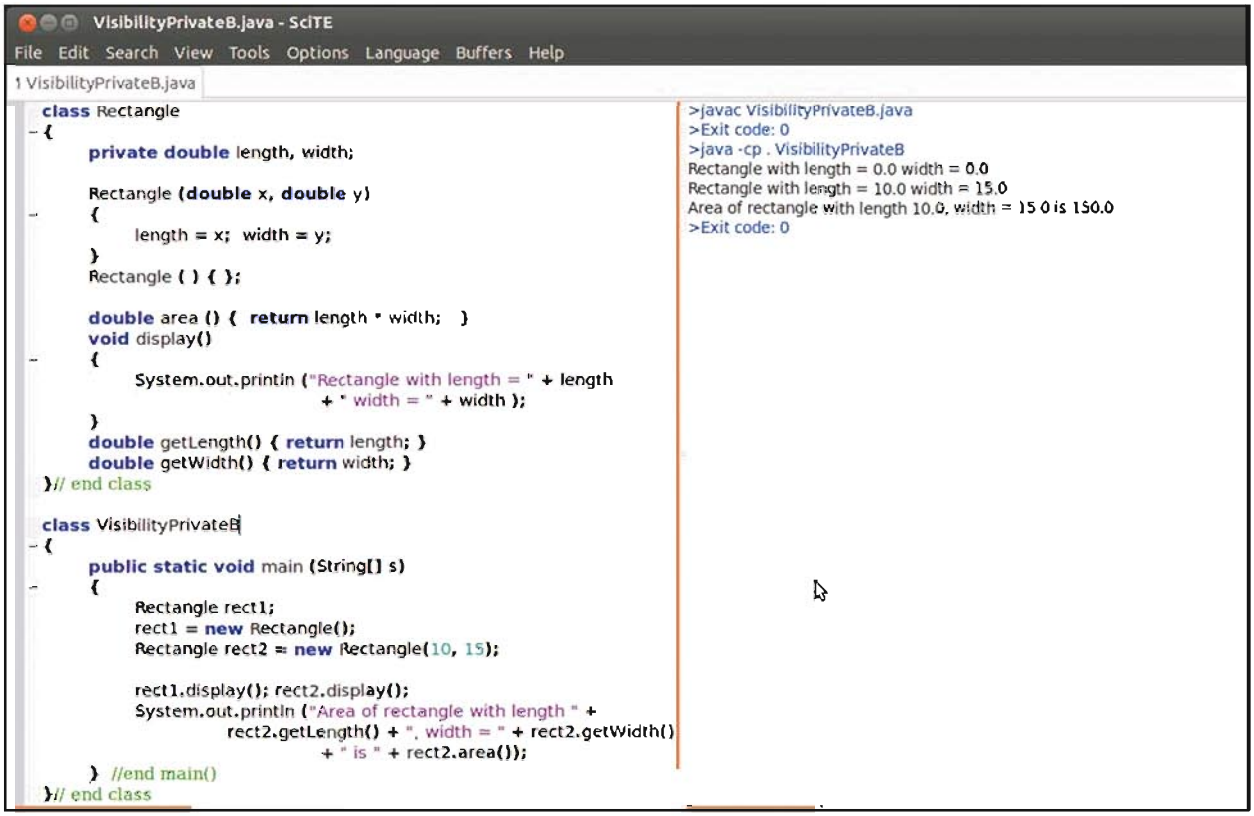
```
class Rectangle
{
    private double length, width;

    Rectangle (double x, double y)
    {
        length = x;  width = y;
    }

    Rectangle ( ) { };

    double area () {  return length * width; }

    void display()
    {
        System.out.println ("Rectangle with length = " + length
                            + " width = " + width );
    }
}// end class

class RectangleVisibility1
{
    public static void main (String[] s)
    {
        Rectangle rect1;
        rect1 = new Rectangle();
        Rectangle rect2 = new Rectangle(10, 15);

        rect1.display(); rect2.display();
        System.out.println ("Area of rectangle with length " +
                            rect2.length + ", width = " + rect2.width
                            + " is " + rect2.area());
    } //end main()
}// end class
```

```
>javac RectangleVisibility1.java
RectangleVisibility1.java:31: length has private access in Rectangle
                        rect2.length + ", width = " + rect2.width
                              ^
RectangleVisibility1.java:31: width has private access in Rectangle
                        rect2.length + ", width = " + rect2.width
                                                      ^
2 errors
>Exit code: 1
```

**આકૃતિ 8.10 : બીજા ક્લાસમાંથી private ઇન્સ્ટન્સ વેરિયેબલ ઍક્સેસ કરવાથી error મળે છે**

હવે સમસ્યા એ છે કે બીજા ક્લાસમાંથી private ચલ કઈ રીતે મેળવી શકાય ? અન્ય ક્લાસની મેથડ વડે ઍક્સેસિબલ (accessible) છે, તેના દ્વારા તે પરોક્ષ રીતે ઉપલબ્ધ બનાવી શકાય છે. આકૃતિ 8.11માં આપેલું કોડ લિસ્ટિંગ જુઓ.

અહીં આપણે બે મેથડ getLength() અને getWidth() ઉમેરેલી છે. અહીં કોઈ પણ મોડિફાયર વાપરેલો ન હોવાથી તેની package વિઝિબિલિટી છે અને આથી તે અન્ય ક્લાસ 'Visibility PrivateB' માં ઉપલબ્ધ છે. આ મેથડ દ્વારા આપણે private ડેટાફિલ્ડ (ઇન્સ્ટન્સ વેરિયેબલ) 'length' અને 'width'ની કિંમત મેળવી શકીએ છીએ. 'Visibility PrivateB' ક્લાસની main() મેથડના છેલ્લા નિર્ગમ વિધાનમાં getLength() અને getWidth() મેથડકૉલનો ઉપયોગ જુઓ.




**આકૃતિ 8.11 : public અને package મેથડ દ્વારા private વેરિયેબલને ઍક્સેસ કરવા**

## ઍક્સેસર અને મ્યુટેટર મેથડ (Accessor and Mutator Methods)

જ્યારે આપણે ડેટાને private ઘોષિત કરીને તેનો ઍક્સેસ મર્યાદિત કરીએ છીએ, ત્યારે આપણો હેતુ અન્ય ક્લાસની મેથડ વડે તે સીધેસીધા ઍક્સેસ કરવાથી અથવા તેમાં ફેરફાર કરવાથી તેનું રક્ષણ કરવાનો છે. જો આપણે આ ડેટાનો ઉપયોગ અન્ય (વ્યક્તિઓ) દ્વારા માન્ય રાખતા હોઈએ તો આપણે ઍક્સેસર (accessor) મેથડ લખીએ. જો આપણે આ ડેટામાં ફેરફાર અન્ય (વ્યક્તિઓ) દ્વારા માન્ય રાખતા હોઈએ તો આપણે મ્યુટેટર (mutator) મેથડ લખીએ.

માન્ય પ્રણાલિકા પ્રમાણે ઍક્સેસર અને મ્યુટેટર મેથડના નામ ચલના નામનો પ્રથમ અક્ષર કૅપિટલ બનાવીને તેના પૂર્વગ તરીકે અનુક્રમે get અને set લખવાથી બને છે. આ પ્રણાલીને કારણે ઍક્સેસર મેથડ "getter" તરીકે અને મ્યુટેટર મેથડ "setter" તરીકે પણ ઓળખાય છે.

આકૃતિ 8.11માં દર્શાવેલા કોડલિસ્ટિંગનું નિરીક્ષણ કરો, જેમાં આપણે "getter" મેથડ getLength() અને getWidth() મેથડનો ઉપયોગ કર્યો છે. જો આપણે ચલ 'length'નો પ્રકાર બદલીશું, તો તે અન્ય વપરાશકર્તાથી છૂપું રહેશે. તે ઍક્સેસર મેથડ 'getLength()'ના અમલીકરણમાં જ ફક્ત અસર કરશે.

જો આપણે બીજી મેથડને ફક્ત ડેટાકિંમત વાંચવાની જ પરવાનગી આપવા ઇચ્છતા હોય, તો આપણે "getter" મેથડ વાપરવી જોઈએ.

જો આપણે અન્ય મેથડને ડેટાની કિંમતમાં ફેરફાર કરવાની પરવાનગી આપવા ઇચ્છતા હોય, તો આપણે "setter" મેથડ વાપરવી જોઈએ. ઉદાહરણ તરીકે, setLength() મેથડને 'length' ઍટ્રિબ્યૂટની કિંમત નીચે જણાવ્યા પ્રમાણે ચલ પસાર કરીને સેટ કરવા માટે વ્યાખ્યાયિત કરી શકાય છે :

```
void setLength(float l) { length = l; }
```

ઍક્સેસર અને મ્યુટેટર મેથડનો ઉપયોગ અન્ય ક્લાસના વપરાશકર્તા દ્વારા ચલને સીધેસીધા ઍક્સેસ કરવા અને તેમાં ફેરફાર કરવાથી અટકાવે છે. આ બાબતની આદત કેળવવી જરા મુશ્કેલ જણાય છે, કારણકે આપણી જરૂરિયાત પ્રમાણે દરેક ઇન્સ્ટન્સ વેરિયેબલ માટે આપણે get અને set મેથડ લખવી પડે. આ નાની અગવડ આપણને સરળતાથી કોડને ફરી વાપરવાની અને તેની જાળવણીની ભેટ આપશે.

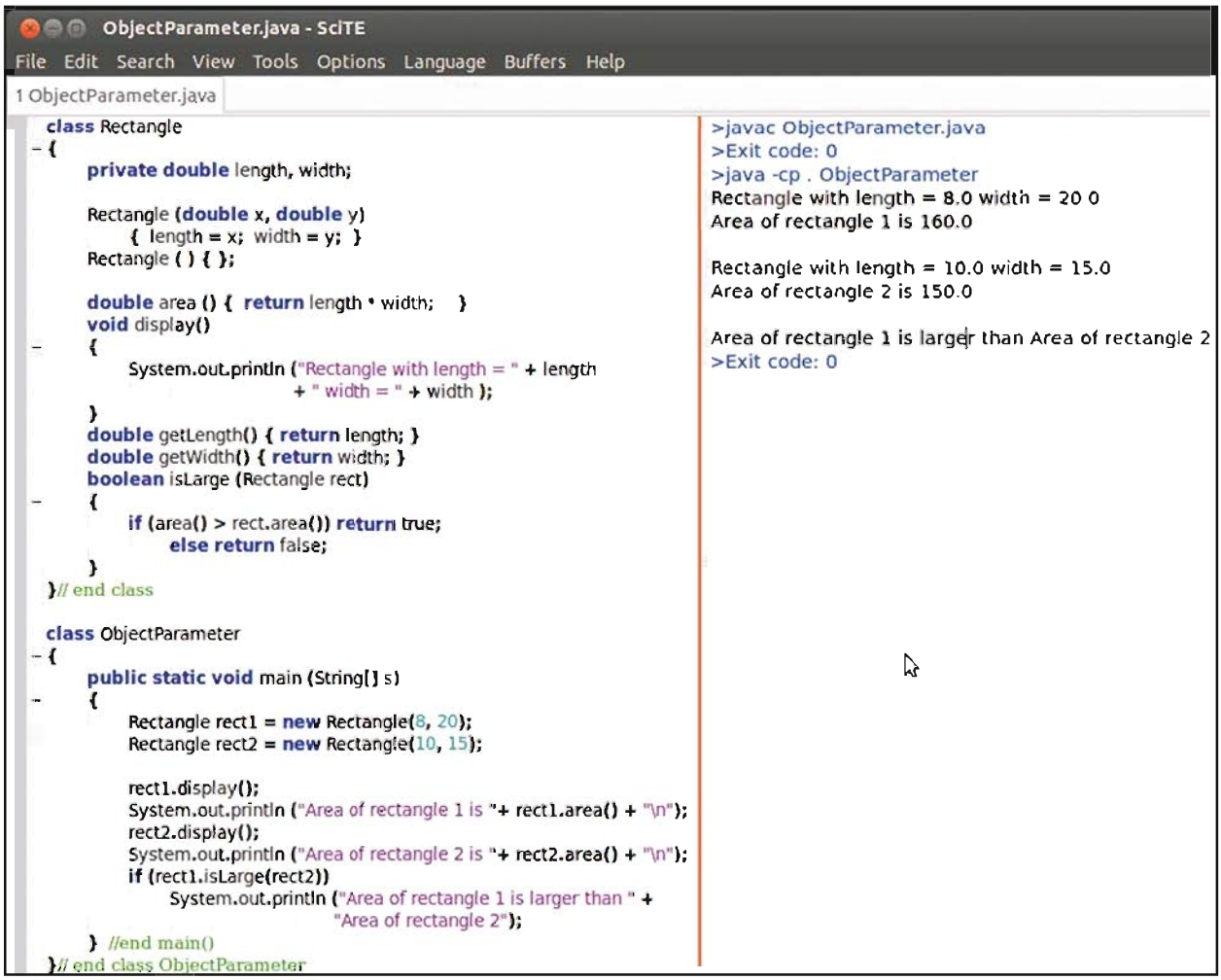
## મેથડમાં પ્રાચલ તરીકે ઑબ્જેક્ટ પસાર કરવો (Passing Object as a Parameter in a Method)

ચલના પ્રાથમિક ડેટાપ્રકાર અને ઉપલબ્ધ સમાવિષ્ટ (built-in) ડેટાપ્રકારોની જેમ જ ઑબ્જેક્ટ પણ મેથડમાં પ્રાચલ તરીકે પસાર કરી શકાય છે.

ઉદાહરણ તરીકે, આપણે rectangle ઑબ્જેક્ટ ઇન્વોક કરતી મેથડનું ક્ષેત્રફળ અન્ય લંબચોરસ કરતાં વધુ છે કે નહીં તે નક્કી કરવા ઇચ્છીએ છીએ. આ માટે આપણે એક મેથડ લખીએ, જેમાં અન્ય rectangle ઑબ્જેક્ટ એક ચલ કે પ્રાચલ તરીકે પસાર કરીએ. ચાલો, આપણે આકૃતિ 8.12માં દર્શાવ્યા પ્રમાણે (Rectangle) ક્લાસમાં 'isLarge' નામની મેથડ ઉમેરીએ અને તેને main() મેથડમાં વાપરીએ.

```
boolean isLarge (Rectangle rect)

{ if (area() > rect.area() ) return true; else return false; }
```

આકૃતિ 8.12માં દર્શાવ્યા પ્રમાણે main() મેથડમાં if વિધાનની અંદરનો મેથડ કૉલ જુઓ : rect1.isLarge(rect2). અહીં, 'rect1' એ કૉલિંગ અથવા ઇન્વોકિંગ ઑબ્જેક્ટ છે જ્યારે 'rect2' એ પ્રાચલ તરીકે પસાર કરેલો ઑબ્જેક્ટ છે. isLarge() મેથડમાં area() એ 'rect1' ઑબ્જેક્ટનું ક્ષેત્રફળ છે અને rect.area() ચલ તરીકે પસાર કરેલા ઑબ્જેક્ટના ક્ષેત્રફળનો ઉલ્લેખ કરે છે.




**આકૃતિ 8.12 : પ્રાચલ તરીકે પસાર કરેલ ઑબ્જેક્ટ**

અહીં એ યાદ રાખો કે પ્રાથમિક (primitive) પ્રકારના પ્રાચલો કિંમત તરીકે પસાર કરેલા છે. વાસ્તવિક (actual) પ્રાચલોની કિંમત નિયમાનુસાર (formal) પ્રાચલોમાં નકલ કરી પછી વિધેયનો અમલ થાય છે. નિયમાનુસાર પ્રાચલોમાં કરેલો ફેરફાર વાસ્તવિક પ્રાચલોને અસર કરતો નથી.

અહીં એ નોંધ કરશો કે ઑબ્જેક્ટના પ્રાચલો રેફરન્સ (Reference) થી પસાર કરેલા છે. આથી, મેથડની અંદરના ઑબ્જેક્ટમાં જે કંઈ ફેરફાર કરવામાં આવે છે, તેથી મૂળ ઑબ્જેક્ટમાં પણ અસર થાય છે. અહીં, વાસ્તવિક પ્રાચલના સ્થાનાંક (અને કિંમત નહીં)-ની નકલ નિયમાનુસાર પ્રાચલમાં કરવામાં આવે છે.

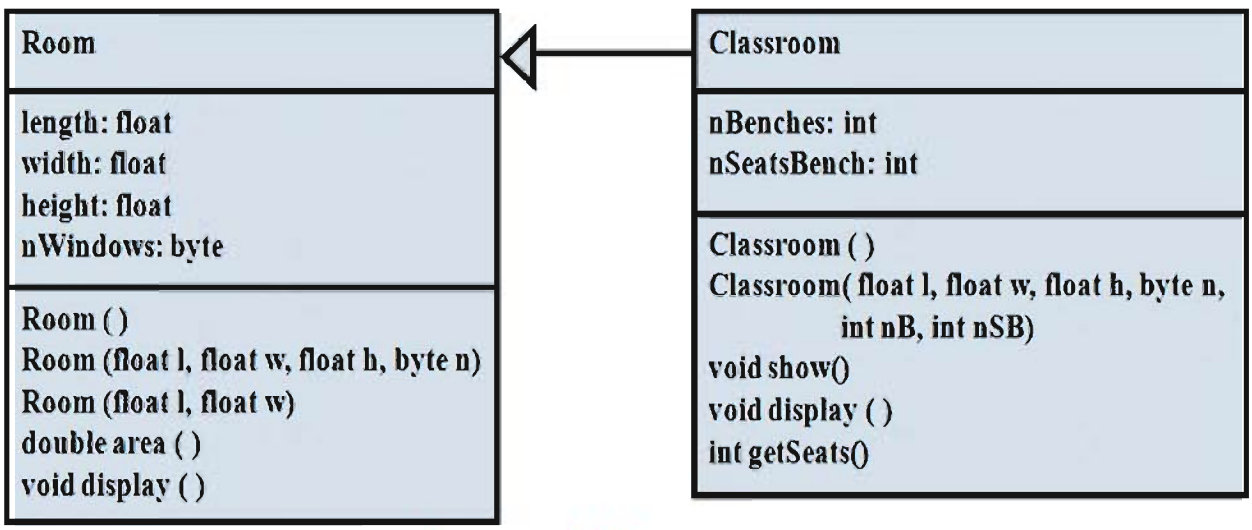
## ઇનહેરિટન્સ (Inheritance)

ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગ ભાષા ઇનહેરિટન્સના ઉપયોગ વડે ફરી વાપરવાની લાક્ષણિકતા (reusability feature) પૂરી પાડે છે. ઇનહેરિટન્સ આપણને હયાત ક્લાસને વિસ્તૃત કરીને વધારાના સામર્થ્ય સાથેનો નવો ક્લાસ બનાવવાની સગવડ આપે છે.

ઇનહેરિટન્સ બે ક્લાસ 'is-a' પ્રકારનો સંબંધ ધરાવતું મૉડેલ છે. ઉદાહરણ તરીકે classroom એ એક room છે અને student એ એક person છે. અહીં, room અને personને પેરન્ટક્લાસ (parentclass) કહેવામાં આવે છે અને classroom તથા studentને ચાઇલ્ડક્લાસ (child class) કહેવામાં આવે છે. પેરન્ટક્લાસને સુપર ક્લાસ (superclass) અથવા બેઇઝ ક્લાસ (base class) પણ કહેવામાં આવે છે. એ જ પ્રકારે ચાઇલ્ડક્લાસને સબક્લાસ (subclass) અથવા એક્સટેન્ડેડ ક્લાસ (extended class) પણ કહેવામાં આવે છે.

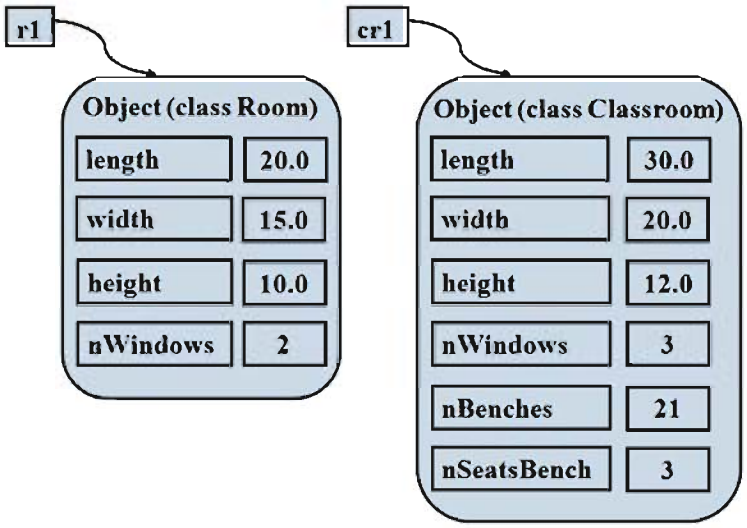
જ્યારે બે ક્લાસ વચ્ચે 'is-a' પ્રકારનો સંબંધ હોય છે, ત્યારે આપણે ઇનહેરિટન્સનો ઉપયોગ કરીએ છીએ. સામાન્ય ગુણધર્મો સુપર ક્લાસમાં રાખવામાં આવે છે. સબક્લાસ સુપર ક્લાસમાંથી બધા ઇન્સ્ટન્સ વેરિયેબલ અને મેથડ વારસામાં મેળવે છે અને તેના પોતાના વધારાના વેરિયેબલ અને મેથડ હોઈ શકે. અહીં નોંધ કરશો કે સબક્લાસમાં કન્સ્ટ્રક્ટર વારસામાં (inherit) મળતા નથી. ઉદાહરણ તરીકે, roomની જેમ classroomને પણ length, width, height અને number of windows ચલ હોય છે. આ ઉપરાંત તેમાં benchesની સંખ્યા અને દરેક benchની વિદ્યાર્થીઓ સમાવવાની ક્ષમતા પણ હોય છે. આ જ રીતે, સબક્લાસ સુપર ક્લાસની બધી જ મેથડ વારસામાં મેળવે છે અને તેની પોતાની વધારાની મેથડ પણ હોઈ શકે. અહીં, સબક્લાસ classroomની વધારાની મેથડ show(), display(), getSeats() તેના કન્સ્ટ્રક્ટરની મેથડ છે.

આકૃતિ 8.13માં ક્લાસ ડાયાગ્રામ દર્શાવ્યો છે, જેમાં 'classroom' નામનો ક્લાસ છે, જે તેના 'Room' નામના પેરન્ટક્લાસમાંથી આવેલો છે (derived). તીર સબક્લાસથી સુપરક્લાસ તરફ નિર્દેશ કરે છે તે જુઓ. સબક્લાસમાં ફક્ત વધારાના ઍટ્રિબ્યૂટ અને મેથડ જ દર્શાવવાની હોય છે. અહીં નોંધ કરશો કે સબક્લાસ એ સુપર ક્લાસનો સબસેટ નથી. હકીકતમાં, સબક્લાસ હંમેશાં તેના સુપર ક્લાસ કરતાં વધારે માહિતી અને મેથડ ધરાવે છે.




**આકૃતિ 8.13 : ઇનહેરિટન્સ ક્લાસ ડાયાગ્રામ**

જ્યારે સબક્લાસનો ઑબ્જેક્ટ ઇન્સ્ટન્સિયેટ (instantiate) થાય છે, ત્યારે ઇનહેરિટ થયેલાં સાથે તેના બધા ઍટ્રિબ્યૂટને મેમરી ફાળવવામાં આવે છે. આકૃતિ 8.14માં સુપર ક્લાસ Room અને સબક્લાસ classroomના ઇન્સ્ટન્સ દર્શાવ્યા છે.




આકૃતિ 8.14 : સુપર ક્લાસ અને સબક્લાસના ઈન્સ્ટન્સ

જાવામાં સબક્લાસ બનાવવા માટે ક્લાસની વ્યાખ્યામાં ચાવીરૂપ શબ્દ 'extends' વાપરવામાં આવે છે. હવે આપણે હયાત ક્લાસ 'Room'નો ઉપયોગ કરીને સબક્લાસ 'Classroom' બનાવીએ. હયાત ક્લાસ 'Room' કોડલિસ્ટિંગ 8.2માં દર્શાવ્યો છે.

```
class Room
{
        float length, width, height;
        byte nWindows;
        static int totWindows;  // class variable

        Room ( ) { }; // user-defined no-argument constructor
        Room (float l, float w, float h, byte n)
        {
                length = l; width = w; height = h;
                nWindows = n;    totWindows+=n;
        }
        Room (float l, float w)
        {
                length = l; width = w; height = 10;
                nWindows = 1;        totWindows++;
        }

    double area ( )  // area =  length * width
        {       return (length * width);        } // end area() method

          void display ( )
        {
                System.out.println ("\nLength: " + length + "\nWidth: " + width);
                System.out.println ("Height: " + height);
                System.out.println ( "Windows: " + nWindows);
        } // end display() method
}  // end Room class
```

કોડલિસ્ટિંગ 8.2 : ઈનહેરિટન્સના ઉપયોગનું ઉદાહરણ

હવે, આપણે કોડ લિસ્ટિંગ 8.3માં દર્શાવ્યા પ્રમાણે સુપર ક્લાસ 'Room'ને વિસ્તૃત કરવા કોડ ઉમેરી સબક્લાસ 'Classroom' બનાવીએ. સબક્લાસમાં બે વધારાના nBenches અને nSeatsBench ઇન્સ્ટન્સ વેરિયેબલ છે. અહીં nSeatsBench ચલ એક બેન્ચ ઉપર કેટલાં વિદ્યાર્થીઓ બેસી શકે તે સંખ્યાનો નિર્દેશ કરે છે. તેના પોતાના વધારાના કન્સ્ટ્રક્ટર અને ત્રણ મેથડ show(), display() અને getSeats() છે.

```
class Classroom extends Room
{
        int nBenches, nSeatsBench;
        Classroom( ) {};
        Classroom( float l, float w, float h, byte n, int nB, int nSB)
        {
                super (l,w,h,n);
                nBenches = nB; nSeatsBench = nSB;
        }
        void show()
        {
                super.display();
                System.out.println ("Benches: " + nBenches );
                System.out.println ("Seats per Bench: " + nSeatsBench );
                System.out.println ("Total Seats in a class: " + getSeats() );
        }
        void display()
        {
                System.out.println("\nClassroom with length " + length + " feet, width "
                    + width + " feet\nhas " + nBenches + " Benches, each to accomodate "
                    + nSeatsBench + " Students\nSo, Total seats in a class is "
                    + getSeats());
        }
        int getSeats() {return nBenches * nSeatsBench; }
}  // end class Classroom
```

**કોડલિસ્ટિંગ 8.3 : Classroom નામના સબક્લાસનો કોડ**

'Classroom' સબક્લાસમાં વપરાશકર્તા વ્યાખ્યાયિત કન્સ્ટ્રક્ટર અને મેથડ show()માં સુપરક્લાસ 'Room'માં લખાયેલા કોડનો ફરી ઉપયોગ થયેલો છે.

સુપર ક્લાસના કન્સ્ટ્રક્ટર સબક્લાસમાં વારસામાં આવતા ન હોવાથી સુપર ક્લાસના કન્સ્ટ્રક્ટરને કૉલ કરવા માટે સબક્લાસના કન્સ્ટ્રક્ટરમાં ચાવીરૂપ શબ્દ 'super' વાપરેલો છે. કન્સ્ટ્રક્ટરમાં આ કૉલ પહેલું વિધાન હોવું જોઈએ. જ્યારે સુપરક્લાસના કન્સ્ટ્રક્ટરનો સ્પષ્ટ રીતે (explicit) કૉલ ન હોય, ત્યારે પહેલા વિધાન તરીકે 'super()'નો કૉલ ગર્ભિત રીતે (implicitly) સુપરક્લાસના ચલ વગરના કન્સ્ટ્રક્ટરનો છે.

Classroom ઑબ્જેક્ટના એટ્રિબ્યૂટ પ્રદર્શિત કરવા માટે show() મેથડનો ઉપયોગ કર્યો છે. સુપરક્લાસ 'Room'માંથી ઇનહેરિટ (inherit) થયેલા પ્રથમ ચાર એટ્રિબ્યૂટર પ્રદર્શિત કરવા માટે આપણે સુપરક્લાસની display() મેથડના હયાત કોડનો ઉપયોગ કરવા ઇચ્છીએ છીએ. આપણે જાણીએ છીએ કે display() મેથડ સબક્લાસમાં પણ ઉપલબ્ધ છે.

show() મેથડની અંદર આપણો ઇરાદો સુપરક્લાસની display()ને કૉલ કરવાનો છે. આ માટે આપણે super.display()નો ઉપયોગ કર્યો છે.

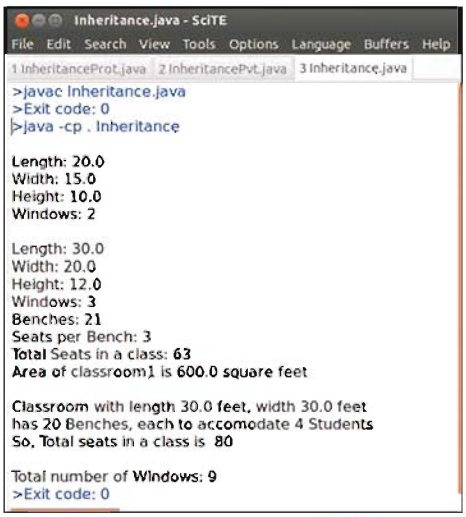
જ્યારે સુપર ક્લાસ અને સબક્લાસમાં એકસમાન સિગ્નેચર (signature) સાથેની મેથડ હોય, ત્યારે સુપર ક્લાસ મેથડની સબક્લાસમાં ઉપેક્ષા કરવામાં આવે છે. સબક્લાસની display() મેથડ સુપરક્લાસની display() મેથડની ઉપેક્ષા કરે છે. એનો અર્થ એ થાય કે આપણે સુપર ક્લાસની display() મેથડનો ફરી ઉપયોગ કર્યા વિના માહિતી જુદી રીતે પ્રદર્શિત કરવા ઇચ્છીએ છીએ. જ્યારે આપણે સુપર ક્લાસની આવી મેથડનો ઉલ્લેખ કરીએ, ત્યારે આપણે ચાવીરૂપ શબ્દ 'super' સાથે ડોટ પ્રક્રિયક અને મેથડનું નામ વાપરવું જોઈએ. અહીં આપણે સુપર ક્લાસની display() મેથડને ઇન્વોક કરવા માટે show() મેથડની અંદર super.display()નો ઉપયોગ કર્યો છે. getSeats() મેથડ ક્લાસરૂપની બેઠકની ક્ષમતાની ગણતરી કરવા માટે વાપરેલી છે.

કોડલિસ્ટિંગ 8.4માં દર્શાવ્યા પ્રમાણે કોડ ઉમેરીને ચાલો, આપણે આ ક્લાસનો વિનિયોગમાં ઉપયોગ કરીએ. અહીં આપણે સુપર ક્લાસ અને સબક્લાસ બંનેના ઑબ્જેક્ટ બનાવેલા છે.

```
class Inheritance  /* Application using Room, Classroom */
{
        public static void main (String args[])
        {
                Room r1 = new Room(20, 15, 10, (byte)2);
                r1.display();
                Classroom cr1 = new Classroom (30, 20, 12, (byte)3, 21, 3);
                cr1.show();
                System.out.println("Area of classroom1 is " + cr1.area() + " square feet");
                Classroom cr2 = new Classroom (30,30,10, (byte)4, 20, 4);
                cr2.display();
                System.out.println ("\nTotal number of Windows: " + Room.totWindows);
        }  // end main()
}  // end Inheritance
```

**કોડલિસ્ટિંગ 8.4 : Room અને Classroomના ઉપયોગ સાથેનો વિનિયોગ**

બધા જ ઇન્સ્ટન્સ વેરિયેબલ અને મેથડ સુપર ક્લાસમાંથી સબક્લાસમાં ઇનહેરિટ થાય છે. આથી, સુપર ક્લાસની area() મેથડને સબક્લાસના ઑબ્જેક્ટ દ્વારા પણ cr1.area()નો ઉપયોગ કરી ઇન્વોક કરી શકાય છે. જ્યારે સબક્લાસના ઑબ્જેક્ટનો ઉપયોગ કરી વિનિયોગમાં ઉપેક્ષા કરેલી મેથડનો ઉલ્લેખ કરવામાં આવે છે, ત્યારે તે સબક્લાસની મેથડ cr2.display()ને કૉલ કરે છે. કોડલિસ્ટિંગ 8.2, 8.3, 8.4ને ભેગાં કરીને બનાવેલા કોડનું આઉટપુટ આકૃતિ 8.15માં દર્શાવ્યું છે.




**આકૃતિ 8.15 : Classroom અને Room ક્લાસના ઉપયોગ બાદ ઇનહેરિટન્સનું આઉટપુટ**

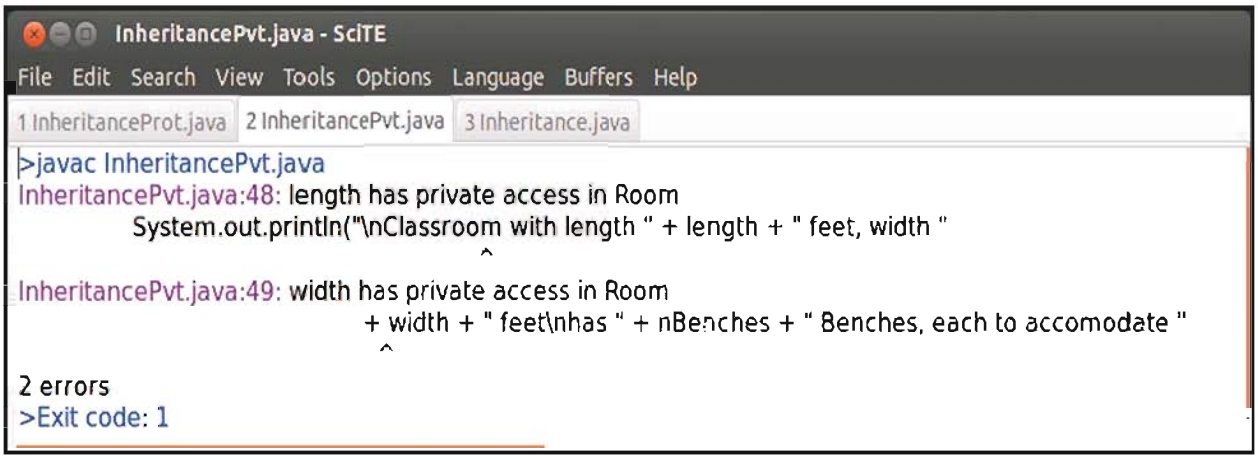
### સુપર ક્લાસના પ્રાઈવેટ મેમ્બર (Private Members of Superclass)

અગાઉના ઉદાહરણમાં બધા જ ઇન્સ્ટન્સની પૅકેજ વિઝિબિલિટી હતી. આથી, તે આખા પૅકેજમાં સીધેસીધા ઉપયોગ માટે ઉપલબ્ધ હતા. આથી, જો આપણે main() મેથડમાં સીધા r1.length અથવા cr1.width એક્સેસ કરવા પ્રયત્ન કરીએ તો તેમાં કોઈ ભૂલ નહીં હોય.

જો આપણે સુપર ક્લાસના ઇન્સ્ટન્સ વેરિયેબલની વિઝિબિલિટી બદલીને private કરીશું, તો આ વેરિયેબલ ક્લાસની બહાર સીધા ઉપલબ્ધ નહીં રહે. યાદ રાખો કે આ ઍટ્રિબ્યૂટ સબક્લાસના પણ છે, છતાં પ્રાઇવેટ ઇન્સ્ટન્સ વેરિયેબલ અથવા મેથડ તેના સબક્લાસમાં પણ વિઝિબલ નથી.

નીચે જણાવ્યા પ્રમાણે કોડલિસ્ટિંગ 8.2માં ઇન્સ્ટન્સ વેરિયેબલને ઘોષિત કરવાના વિધાનમાં તેને private બનાવો અને આકૃતિ 8.16માં દર્શાવ્યા પ્રમાણે તેના આઉટપુટમાં error આવી તેનું નિરીક્ષણ કરો.

```
private float length, width, height;

private byte nWindows;
```




**આકૃતિ 8.16 : ક્લાસના પ્રાઈવેટ મેમ્બર તે ક્લાસની બહાર ઍક્સેસિબલ નથી**

યાદ રાખો કે પ્રાઈવેટ મેમ્બર તે જે ક્લાસમાં વ્યાખ્યાયિત કરેલા હોય, તેમાં જ ફક્ત સીધા ઉપલબ્ધ હોય છે અને બીજે ક્યાંય પણ નહીં. તેને અન્ય જગ્યાએ ઉપલબ્ધ કરવા માટે પબ્લિક ઍક્સેસર અને મ્યુટેટર મેથડ 'getter' અને 'setter' મેથડનો ઉપયોગ કરો.

### સુપરક્લાસના પ્રોટેક્ટેડ મેમ્બર (Protected Members of Superclass)

અગાઉ 'Visibility Modifiers'ના વિષય ઉપર આપણે શીખ્યા કે ઇનહેરિટેડ સબક્લાસમાં 'protected' મેમ્બર 'private' મેમ્બર તરીકે ઉપલબ્ધ હોય છે. નીચે જણાવ્યા પ્રમાણે 'Room' ક્લાસના ઇન્સ્ટન્સ વેરિયેબલને બદલીને 'protected' કરો.

```
protected float length, width, height;

protected byte nWindows;
```

આ બદલાયેલો કોડ જ્યારે અમલમાં મૂકવામાં આવે છે, ત્યારે આકૃતિ 8.15માં દર્શાવ્યા પ્રમાણે આપણે એ જ પરિણામ મેળવીએ છીએ. અહીં એ નોંધવું જોઈએ કે જાવામાં બહુવિધ ઇનહેરિટન્સની સગવડ નથી. સબક્લાસ ફક્ત એક જ સુપરક્લાસમાંથી આવે છે.

### કૉમ્પોઝિશન અને એગ્રિગેશન (Composition and Aggregation)

કૉમ્પોઝિશન અને એગ્રિગેશન ક્લાસની રચના છે, જે અન્ય ઑબ્જેક્ટનો સમાવેશ કરે છે. તે અલગ-અલગ ક્લાસ વચ્ચે "has-a" પ્રકારનો સંબંધ રચે છે.

ઉદાહરણ તરીકે, જો આપણે 'Library' નામનો ક્લાસ વ્યાખ્યાયિત કરીએ, તો તેને એક reading room (વાંચનખંડ) હોય છે, અહીં reading room એ 'Room' નામના ક્લાસનો એક ઑબ્જેક્ટ છે. આ રીતે Libraryમાં Room હોય છે. જ્યારે કોઈ ક્લાસ અન્ય ક્લાસના ઑબ્જેક્ટનો સમાવેશ કરે, ત્યારે તે કન્ટેઇનર ક્લાસ (container class) તરીકે ઓળખાય છે.

**અન્ય ઉદાહરણ :**

- દરેક વ્યક્તિનું name અને address હોય છે. અહીં name ક્લાસ Nameમાં છે, જેને first name, middle name અને last name એમ ત્રણ ઍટ્રિબ્યૂટ છે. address ક્લાસ Addressમાં છે, જેના ઍટ્રિબ્યૂટ house number, apartment / society name, area, city, state, country અને pincode છે.
- કારને steering, wheels અને engine હોય છે. અહીં steering ક્લાસ Steeringમાં છે, wheel ક્લાસ Wheelમાં છે અને engine ક્લાસ Engineમાં છે. ક્લાસ Carના ત્રણે ઍટ્રિબ્યૂટ કારના ભાગ છે.

હવે આપણે નીચે જણાવેલા ઍટ્રિબ્યૂટ સાથેનો ક્લાસ 'Library' બનાવીએ :

- nBooks: int, પુસ્તકાલયમાં પુસ્તકની સંખ્યા
- nMagazines: int, પુસ્તકાલયમાં લવાજમ ભરેલાં સામયિકની સંખ્યા
- nNewspapers: int, પુસ્તકાલયમાં લવાજમ ભરેલાં વર્તમાનપત્રકની સંખ્યા
- readingRoom: Room

અહીં readingRoom એ બેઝિક ડેટાટાઇપનો નથી તે જુઓ. તેનો પ્રકાર 'Room' ક્લાસનો છે.

કોડ લિસ્ટિંગ 8.5માં Room અને Library નામના ક્લાસ બનાવવાનો કોડ અને તેનો 'Container.java' નામના વિનિયોગમાં ઉપયોગ દર્શાવ્યો છે.

```
/* Using objects as data members in a container class */
class Room
{
        protected float length, width, height;
        protected byte nWindows;
        static int totWindows;  // class variable

        Room ( ) { }; // user-defined no-argument constructor
        Room (float l, float w, float h, byte n)
        {
                length = l; width = w; height = h;
                nWindows = n;    totWindows+=n;
        }
        Room (float l, float w)
        {
                length = l; width = w; height = 10;
                nWindows = 1;        totWindows++;
        }
```

```
        double area ( )  // area =  length * width
    {       return (length * width);        } // end area() method
        void display ( )
    {
            System.out.println ("Length: " + length + "\nWidth: " + width);
            System.out.println ("Height: " + height);
            System.out.println ( "Windows: " + nWindows);
    } // end display() method
} // end Room class

class Library
{
    int nBooks, nMagazines, nNewspapers;
    Room readingRoom;
    Library( ) {};
    Library( int nB, int nM, int nN, Room r)
    {
            nBooks = nB; nMagazines=nM; nNewspapers=nN;
            readingRoom=r;
    }
    void display()
    {
            System.out.println ("\nLibrary Details:\nNumber of books: " + nBooks );
            System.out.println ("Number of subscribed magazines: " + nMagazines );
            System.out.println ("Number of subscribed newspapers : " + nNewspapers);
            System.out.println ("Reading Room:");
            readingRoom.display();
    }
} // end class Library

class Container  /* Application using Room, Library */
{
    public static void main (String args[])
    {
            Room r1 = new Room(20, 15, 10, (byte)2);
            r1.display();
            Library lib = new Library (300, 20, 5, r1);
            lib.display();
    } // end main()
} // end class Container
```

કોડલિસ્ટિંગ 8.5 : કોમ્પોઝિશન અને એગ્રિગેશનનું ઉદાહરણ

કોડલિસ્ટિંગ 8.5માં નીચેના મુદ્દાઓનું નિરીક્ષણ કરો :

- ક્લાસ 'container'ની main() મેથડમાં :
  - ક્લાસ 'Library'નો ઑબ્જેક્ટ lib ચાર ચલ (arguments) સાથે કન્સ્ટ્રક્ટર વાપરીને બનાવવામાં આવ્યો છે. Room ક્લાસનો r1 એ છેલ્લો ચલ છે.
  - ક્લાસ 'Library'ની મેથડ display() ઑબ્જેક્ટ libનો ઉપયોગ કરીને ઇન્વોક કરેલી છે.
- ક્લાસ 'Library'માં :
  - ઑબ્જેક્ટ readingRoom ક્લાસ 'Room'નો ઍટ્રિબ્યૂટ છે.
  - ચાર ચલ (arguments) સાથેના કન્ટ્રક્ટરમાં ક્લાસ 'Room'ની છેલ્લી ચલ 'r'નો ઉપયોગ કરે છે અને ઍસાઇન્મેન્ટ વિધાન 'readingRoom = r;' વાપરીને Roomના ઍટ્રિબ્યૂટને કિંમતો ઍસાઇન (assign) કરે છે.
  - display() મેથડ વ્યાખ્યાયિત કરેલી છે 'readingRoom.display();' દ્વારા 'Room' ક્લાસની display() મેથડ ઇન્વોક કરે છે. અહીં નોંધ કરશો કે display() મેથડની ઉપેક્ષા કરી નથી, આપણે ઇનહેરિટન્સનો ઉપયોગ કર્યો નથી. વાચક આ મેથડ માટે અન્ય નામ વાપરી શકે. જેમકે, 'lib.display()'ને બદલે 'Library' ક્લાસમાં show()નો ઉપયોગ main() મેથડમાં 'lib.show()' વડે.

આવા સંજોગોમાં વાચક એવું વિચારે કે - ઇનહેરિટન્સ શા માટે ન વાપરવું ? શા માટે 'Library' ક્લાસને 'Room' ક્લાસમાંથી ઇનહેરિટ ન કરવો અને તેમાં ત્રણ વધારાના ઍટ્રિબ્યૂટ ઉમેરવા ?

અહીં નોંધ કરશો કે 'Library' એ 'Room' પ્રકારનો નથી. 'Library' અને 'Room' વચ્ચે 'is-a' સંબંધ નથી, પણ 'has-a' સંબંધ છે. 'Library'માં 'Room' હોય, જેનો રીડિંગરૂમ તરીકે ઉપયોગ થાય છે. આથી, readingRoomને આપણે 'Room'ના પ્રકારનો જણાવ્યો છે અને તે 'Library' ક્લાસનો ઍટ્રિબ્યૂટ છે.

## સારાંશ

આપણે ક્લાસ અને ઑબ્જેક્ટ કઈ રીતે બનાવવા, ક્લાસના ઇન્સ્ટન્સ વેરિયેબલ કેવી રીતે ઍક્સેસ કરવા અને ઑબ્જેક્ટ વડે ઇન્સ્ટન્સ મેથડ ઇન્વોક કરવી તે બાબત અહીં શીખ્યા. કન્સ્ટ્રક્ટર એ ક્લાસના નામની જ, ચલ વગરની અને પરત પ્રકાર વગરની વિશિષ્ટ મેથડ છે. જ્યારે ઑબ્જેક્ટ નવા પ્રક્રિયક સાથે ઇન્સ્ટેન્શિયેટ કરવામાં આવે છે, ત્યારે ગર્ભિત રીતે કન્સ્ટ્રક્ટર કૉલ કરવામાં આવે છે, ઇન્સ્ટન્સ વેરિયેબલ દરેક ઑબ્જેક્ટના હોય છે. ક્લાસ વેરિયેબલ અને મેથડ 'static' ચાવીરૂપ શબ્દ વાપરીને વ્યાખ્યાયિત કરવામાં આવે છે. સ્ટેટિક મેમ્બરને ક્લાસ દીઠ ફક્ત એક જ વાર મેમરી ફાળવવામાં આવે છે અને ક્લાસના બધા ઑબ્જેક્ટ દ્વારા તેનો સહિયારો ઉપયોગ કરે છે, તે ક્લાસના હોય છે; ઑબ્જેક્ટના નહીં. આપણે ઍક્સેસ મોડિફાયર વિશે પણ અભ્યાસ કર્યો, જે ઇન્સ્ટન્સ મેમ્બરની વિઝિબિલિટી નક્કી કરે છે. પ્રાઇવેટ મેમ્બર ફક્ત ક્લાસની અંદર જ વિઝિબલ હોય છે કે જ્યાં તે વ્યાખ્યાયિત કરેલ હોય છે, પ્રોટેક્ટેડ મેમ્બર ફક્ત ઇનહેરિટ સબક્લાસમાં જ વિઝિબલ હોય છે, પૅકેજમેમ્બર પૅકેજમાં સર્વત્ર વિઝિબલ હોય છે અને પબ્લિક મેમ્બર બધી જ જગ્યાએ વિઝિબલ હોય છે. સુરક્ષાના હેતુ માટે પ્રાઇવેટ ઇન્સ્ટન્સ વેરિયેબલનો ઉપયોગ અને 'getters' અને 'setters' મેથડનો public તરીકે ઉપયોગ કરવો સલાહભર્યું છે. અંતમાં આપણે હયાત ક્લાસમાંથી નવો ક્લાસ ઇનહેરિટ કરવો, મેથડનો ફરી ઉપયોગ, તેને વિસ્તૃત કરવી અને તેની ઉપેક્ષા કરવી વિશે શીખ્યા. આપણે ક્લાસમાં ઇન્સ્ટન્સ વેરિયેબલ તરીકે ઑબ્જેક્ટનો ઉપયોગ પણ જોયો. તે ક્લાસને ઑબ્જેક્ટના કૉમ્પોઝિશન અને એગ્રિગેશન બનવાનું સામર્થ્ય પૂરું પાડે છે.

## સ્વાધ્યાય

1. ઑબ્જેક્ટનું ઇન્સ્ટેન્શિએશન એટલે શું ?
2. ક્લાસ-વેરિયેબલની જરૂરિયાત બાબત એક ઉદાહરણ આપો.
3. મેથડ અને કન્સ્ટ્રક્ટર વચ્ચેનો તફાવત જણાવો.

4. એક્સેસર અને મ્યુટેટર મેથડ વિશે જણાવો.

5. સબક્લાસમાંથી સુપરક્લાસનો કન્સ્ટ્રક્ટર કેવી રીતે ઇન્વોક કરી શકાય ?

6. સુપરક્લાસની ઉપેક્ષા કરેલી મેથડ સબક્લાસમાંથી કેવી રીતે ઇન્વોક કરી શકાય ?

7. 'એક્સેસ મોડિફાયર' વિશે ટૂંક નોંધ લખો.

8. ઇનહેરિટન્સનો ઉપયોગ અને અલગ-અલગ ક્લાસ વચ્ચે સંબંધના પ્રકાર પ્રમાણે કોમ્પોઝિશન અથવા એગ્રિગેશનનો ઉપયોગ સમજાવો.

9. મેથડ ઓવરલોડિંગ અને મેથડ ઓવરરાઇડિંગ વચ્ચેનો તફાવત સમજાવો.

10. નીચે આપેલા વિકલ્પોમાંથી સાચો વિકલ્પ પસંદ કરો :

(1) નીચેનામાંથી ઍટ્રિબ્યૂટ અને મેથડને કોણ વ્યાખ્યાયિત કરે છે ?

(a) ક્લાસ (b) ઑબ્જેક્ટ (c) ઇન્સ્ટન્સ (d) વેરિયેબલ

(2) નીચેનામાંથી ક્યો ચાવીરૂપ શબ્દ ક્લાસ વેરિયેબલ અને ક્લાસમેથડને ઘોષિત કરવા માટે વપરાય છે ?

(a) static (b) private (c) public (d) package

(3) નીચેનામાંથી ક્યો પ્રક્રિયક ઑબ્જેક્ટ બનાવે છે અને તેનો રેફરન્સ પરત કરે છે ?

(a) ડોટ (.) (b) new (c) કોલન (:) (d) ઍસાઇન્મેન્ટ (=)

(4) ક્લાસનો ઇન્સ્ટન્સ બનાવ્યા વિના નીચેનામાંથી કઈ મેથડ કૉલ કરી શકાય ?

(a) ઇન્સ્ટન્સ મેથડ (b) ક્લાસ મેથડ

(c) કન્સ્ટ્રક્ટર મેથડ (d) ઉપરના બધા વિકલ્પ

(5) નીચેનામાંથી એકસમાન નામ ધરાવતી પણ અલગ-અલગ પ્રાચલો સાથેની એક કરતાં વધારે મેથડનો ઉલ્લેખ કોણ કરે છે ?

(a) ઓવરલોડેડ મેથડ્સ (b) ઓવરરીડન મેથડ્સ

(c) ડુપ્લિકેટ મેથડ્સ (d) ઉપરના બધા વિકલ્પ

(6) નીચેનામાંથી કઈ મેથડ ઑબ્જેક્ટ બનાવતા સમયે આપોઆપ ઇન્વોક થાય છે ?

(a) ઇન્સ્ટન્સ મેથડ (b) કન્સ્ટ્રક્ટર

(c) ક્લાસ મેથડ (d) ઉપરના બધા વિકલ્પ

(7) નીચેનામાંથી ક્યો ચાવીરૂપ શબ્દ સબક્લાસ કન્સ્ટ્રક્ટરમાં સુપર ક્લાસ કન્સ્ટ્રક્ટરનો ઉલ્લેખ કરે છે ?

(a) extends (b) super

(c) સુપર ક્લાસનું નામ (d) new

(8) નીચેનામાંથી જાવામાં ઇન્સ્ટન્સ મેથડ ઇન્વોક કરવા માટે વપરાય શું છે ?

(a) ઑબ્જેક્ટનું નામ, કોલોન(:) અને મેથડનું નામ

(b) ઑબ્જેક્ટનું નામ, ડોટ(.) અને મેથડનું નામ

(c) ક્લાસનું નામ, કોલોન(:) અને મેથડનું નામ

(d) ક્લાસનું નામ, ડોટ(.) અને મેથડનું નામ

(9) નીચેનામાંથી ઇન્સ્ટન્સ મેથડ વડે શું ઍક્સેસિબલ છે ?

(a) ફક્ત ઇન્સ્ટન્સ વેરિયેબલ
(b) ફક્ત ક્લાસ વેરિયેબલ
(c) બંને ક્લાસ વેરિયેબલ અને ઇન્સ્ટન્સ વેરિયેબલ
(d) ઉપરના બધા વિકલ્પ

(10) જ્યારે સુપરક્લાસમાં મેથડનું નામ અને સિગ્નેચર સમાન હોય ત્યારે તેને શું કહેવામાં આવે છે ?

(a) ઓવરલોડેડ મેથડ્સ
(b) ઓવરરીડન મેથડ્સ
(c) ઇનહેરિટેડ મેથડ્સ
(d) ઉપરના બધા વિકલ્પ

## પ્રાયોગિક સ્વાધ્યાય

નીચેનાં કાર્ય માટે જાવાપ્રોગ્રામ લખો :

1. 'FixedDeposit' નામનો ક્લાસ બનાવો, જેના ત્રણ ઍટ્રિબ્યૂટ (મુદ્દલ રકમ, વાર્ષિક વ્યાજનો દર અને વર્ષમાં જમા રકમનો સમયગાળો) અને મેથડ હોય કે જે ચક્રવૃદ્ધિ વ્યાજની રીતે પાકતી મુદતની રકમ પરત કરે. main() મેથડ સાથેનો અન્ય 'FixedDepositDemo' નામનો ક્લાસ બનાવો. main() મેથડમાં બે ઑબ્જેક્ટ બનાવો, તેના ઍટ્રિબ્યૂટને કિંમતો ઍસાઈન કરો અને તેને પાકતી મુદતની રકમ સાથે પ્રદર્શિત કરો.

2. 'FixedDeposit' ક્લાસમાં નીચેના કન્સ્ટ્રક્ટર ઉમેરો અને તેનો ઑબ્જેક્ટ બનાવવામાં ઉપયોગ કરો.

   a. ચલ વગરના કન્સ્ટ્રક્ટર કે જે મુદ્દલ રકમને 1000, વાર્ષિક વ્યાજના દરને 5% અને જમા રકમના સમયગાળાને 3 વર્ષ પ્રારંભિક કિંમત (initialize) આપે.

   b. ત્રણ ઍટ્રિબ્યૂટને પ્રારંભિક કિંમત આપવા 3 ચલ (arguments) સાથેનો પ્રાચલોવાળો કન્સ્ટ્રક્ટર.

3. પ્રાયોગિક સ્વાધ્યાય-1માં 'FixedDeposit' ક્લાસના ઇન્સ્ટન્સ વેરિયેબલની વિઝિબિલિટી બદલીને 'private' કરો અને તેનો અમલ કરવા પ્રયત્ન કરો. તે error આપશે ? જો હા હોય તો, ઇન્સ્ટન્સ વેરિયેબલની કિંમત પ્રદર્શિત કરવા માટે ક્લાસમાં display() મેથડ ઉમેરો.

4. 'FixedDeposit' ક્લાસનાં ત્રણ ઍટ્રિબ્યૂટને get અને set કરવા માટે ઍક્સેસર અને મ્યુટેટર મેથડ ઉમેરો. આ મેથડનો ઉપયોગ કરીને main() મેથડના પ્રાઇવેટ ઇન્સ્ટન્સ વેરિયેબલની કિંમત પ્રદર્શિત કરો.

5. કુલ મુદ્દલ જમા રકમ ધરાવતા ક્લાસમાં 'totDeposit' ક્લાસ વેરિયેબલ ઉમેરો. કન્સ્ટ્રક્ટર અને સેટર મેથડમાં ફેરફાર કરો, જેથી કુલ જમા રકમ મેળવી શકાય. 'totDeposit' ચલની કિંમત પ્રદર્શિત કરવા મેથડ લખો અને તેનો ઉપયોગ બતાવો.

6. 'Rectangle' ક્લાસનો ઉપયોગ કરી 'Box' નામનો સબક્લાસ વધારાના 'height' ઍટ્રિબ્યૂટ અને 'volume' મેથડ સાથે ઉત્પન્ન કરો (નિર્માણ કરો - derive). (ઘનફળ = ઊંચાઈ x પહોળાઈ x લંબાઈ = ઊંચાઈ x ક્ષેત્રફળ)

# ઍરે અને સ્ટ્રિંગનો ઉપયોગ 9

પ્રકરણ 7માં આપણે જાવામાં ઉપલબ્ધ ડેટાના મૂળભૂત પ્રકારોનો અભ્યાસ કર્યો. મૂળભૂત પ્રકારના ચલ (variable)માં એક સમયે ફક્ત એક જ કિંમત રાખી શકાય છે. આ પ્રકારના ચલને અદિશ ચલ કે સ્કેલર વેરિયેબલ (scalar variable) કહેવામાં આવે છે. આ પ્રકરણમાં આપણે કેટલાક સંયોજિત કે કૉમ્પોઝિટ (composite) ડેટાપ્રકારો વિશે અભ્યાસ કરીશું જેનો ઉપયોગ એક કરતાં વધારે ડેટાકિંમતોના સમૂહનો સંગ્રહ કરવા માટે થાય છે, ઉદાહરણ તરીકે, ઍરે (array) અને સ્ટ્રિંગ (string).

## ઍરેનો પરિચય (Introduction to Array)

ઍરે (array) એ એક જ પ્રકારના ઘટકોના સંગ્રહને રજૂ કરતો ચલ (variable) છે. ઍરે સદિશ (vector), શ્રેણિક (matrix) અને અન્ય બહુ-પરિમાણીય (multi-dimensional) માહિતી રજૂ કરવા માટે ઉપયોગી છે. વેક્ટર (vector) એ એક-પરિમાણીય (1-D) ડેટા-સંરચના (data structure) છે, જે અક્ષરો, સંખ્યાઓ જેવી વસ્તુઓની યાદી સંગ્રહવા માટે વાપરી શકાય છે. રો (આડી હરોળ) અને કૉલમ (ઊભી હરોળ) વડે બનેલી કોષ્ટક જેવી દ્વિ-પરિમાણીય (2-D) ડેટા-સંરચના રજૂ કરવા માટે મૅટ્રિક્સ (matrix) વપરાય છે.

જ્યારે સમાન પ્રકારના અનેક ઘટકો ઉપર એકસમાન કાર્ય કરવાનું હોય, ત્યારે ઍરે ઉપયોગી બને છે. ઍરેના તમામ ઘટકો મેમરીમાં એકસાથે લગોલગ સંગ્રહાયેલા હોય છે. ઍરેના દરેક ઘટકને ઍરે વેરિયેબલ સાથે સંકળાયેલા સૂચક સ્થાનાંક (index position) વડે ઓળખવામાં આવે છે.

જાવામાં વસ્તુઓની યાદીનું સંચાલન કરવા માટે વપરાતો ઑબ્જેક્ટ એ ઍરે છે. ઍરે બનાવવા માટે બે પગલાંની ક્રિયા છે :

1. ઍરે-ઑબ્જેક્ટ ઘોષિત કરો.
2. ઍરે-ઑબ્જેક્ટની રચના કરો.

ઍરે-ઑબ્જેક્ટ બે રીતથી બનાવી શકાય છે :

1. new પ્રક્રિયક વાપરીને અને તેનું કદ જણાવીને.
2. સીધેસીધા ઍરેના ઘટકોને પ્રારંભિક કિંમત આપીને. (initializing)

## એક-પરિમાણીય ઍરે (1-D Array)

એક-પરિમાણવાળા ઍરે 1-D ઍરે તરીકે ઓળખાય છે. ઉદાહરણ તરીકે, વેક્ટર (vector) કે જેને એક અથવા વધારે અદિશ ચલ (scalar variables)ના સમૂહ તરીકે ગણી શકાય. marks1, marks2, marks3, marks4, marks5ની રીતે અલગ-અલગ ચલ ઘોષિત કરવાના બદલે આપણે ઍરે marks[5] ઘોષિત કરી શકીએ અને તેના દરેક ઘટકને marks[0], marks[1], marks[2], marks[3], marks[4] વાપરી આપણે તેને મેળવી શકીએ છીએ.

1-D ઍરે ઘોષિત કરવા માટે ચોરસ કૌંસની જોડી [ ] ઍરેના નામ પછી અથવા ડેટાના પ્રકાર પછી આપણે વાપરીએ છીએ. ઍરે ઘોષિત કરવાની વાક્યરચના નીચે પ્રમાણે છે :

<data type> <array name> []; અથવા <data type> [] <array name>;

ઉદાહરણ તરીકે, એક વિદ્યાર્થીના ગણિત વિષયની પાંચ કસોટીમાં મેળવેલા ગુણનો સંગ્રહ કરવા માટે આપણે પાંચ પૂર્ણાંક સંખ્યા ધરાવતા ઘટકોનો ઍરે વાપરી શકીએ. 'marks' નામનો પાંચ ઘટકો (elements) ધરાવતો ઍરે-ઑબ્જેક્ટ નીચે મુજબ બનાવી શકાય છે :

```
int marks[];                    // declare array object
marks = new int [5];            // create array object
```

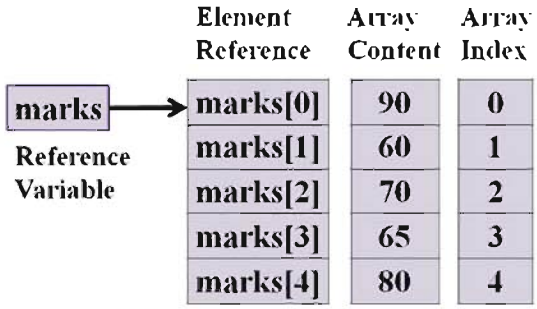
ઉપરનાં બંને વિધાનોને ભેગાં કરીને નીચે મુજબ એક જ વિધાનથી પણ આપણે ઍરે બનાવી શકીએ :

int marks[] = new int [5];  અથવા  int [] marks = new int [5];

અહીં ઍરેનું નામ 'marks' છે. પાંચ પૂર્ણાંક સંખ્યાઓને સંગ્રહ જ્યાં થશે, તે મેમરી સ્થાનાંકનો તે નિર્દેશ કરે છે. int ડેટા પ્રકારની પૂર્ણાંક સંખ્યાના સંગ્રહ માટે 4 બાઇટ વપરાય છે. આથી, 'marks' ઍરે માટે મેમરીમાં સળંગ 5 x 4 = 20 બાઇટની જરૂર પડે છે.

ઍરેના કોઈ ઘટકનો નિર્દેશ કરવા માટે આપણે આકૃતિ 9.1માં દર્શાવ્યા પ્રમાણે ઍરેના નામ પછી મોટા કૌંસ [ ]ની અંદર ઇન્ડેક્સ (index) અથવા સબસ્ક્રિપ્ટ (subscript)નો ઉપયોગ કરીએ છીએ. ઇન્ડેક્સ કે સબસ્ક્રિપ્ટ ઍરેમાં ઘટકનું સ્થાન દર્શાવે છે. ઉદાહરણ તરીકે, marks[2]. ઇન્ડેક્સની કિંમત શૂન્ય (0)થી શરૂ થાય છે.

```
int marks[] = {90, 60, 70, 65, 80};
```

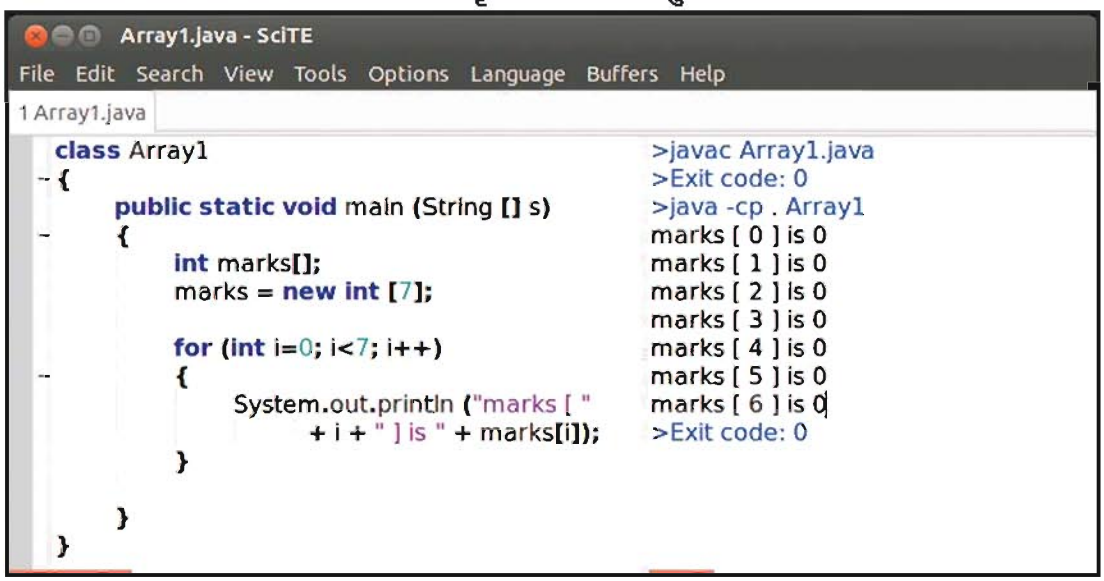



આકૃતિ 9.1 : એક-પરિમાણીય ઍરે

આકૃતિ 9.1માં ઍરે ચલ marksની ઇન્ડેક્સકિંમત 0થી 4 છે. અહીં, marks[0] ઍરેના પ્રથમ ઘટકનો નિર્દેશ કરે છે અને marks[4] એ છેલ્લા ઘટકનો નિર્દેશ કરે છે.

પ્રકરણ 8માં સમજૂતી આપ્યા પ્રમાણે ઑબ્જેક્ટ એક રેફરન્સ ચલ છે. ઍરે જાવામાં એક ઑબ્જેક્ટ હોવાથી ઍરેનું નામ પણ એક રેફરન્સ ચલ છે. તે મેમરીના એ સ્થાનાંકનો રેફરન્સ ધરાવે છે, જ્યાં ઍરેના ઘટકોનો સંગ્રહ થયેલો છે. આકૃતિ 9.1 જુઓ.

ઍરે એક ઑબ્જેક્ટ છે, આથી તેના ઘટકોને પૂર્વનિર્ધારિત (default) કિંમતો આપવામાં આવે છે (initialized). જાવા-પ્રોગ્રામમાં ઍરેનો ઉપયોગ કેવી રીતે કરવો તે આકૃતિ 9.2માં દર્શાવ્યું છે.



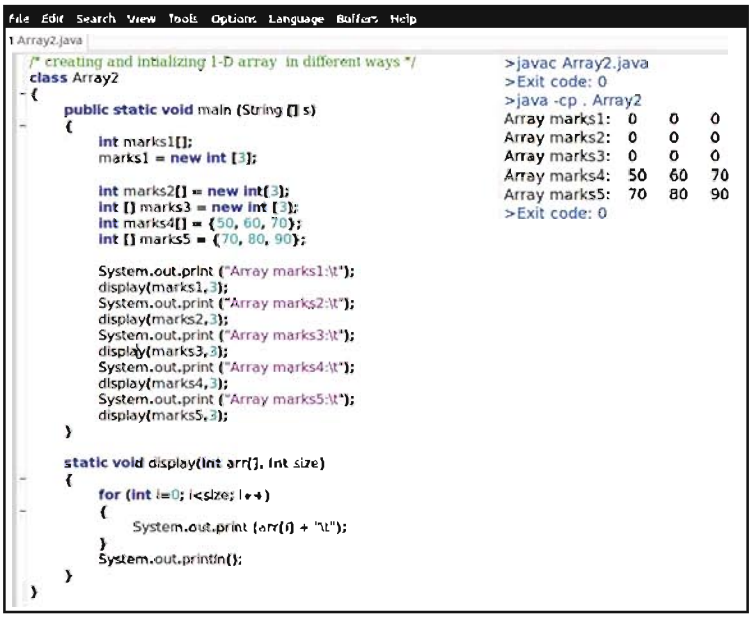

આકૃતિ 9.2 : ઍરે-ઑબ્જેક્ટના ઘટકોને પૂર્વનિર્ધારિત કિંમતો પ્રારંભિક કિંમત તરીકે આપવી

જ્યારે ઍરે ઘોષિત કરવામાં આવે છે, તે સમયે પણ તેના ડેટા ઘટકોની પ્રારંભિક કિંમત (initial values) જણાવી શકાય છે. 1-D ઍરેના ઘટકોની પ્રારંભિક કિંમતો છગડિયા કૌંસ { }માં અલ્પવિરામ વડે અલગ પાડેલી કિંમતો વડે આપી શકાય છે. ઉદાહરણ તરીકે,

int marks[ ] = {90, 70, 77};  અથવા  int [] marks = {90, 70, 77};

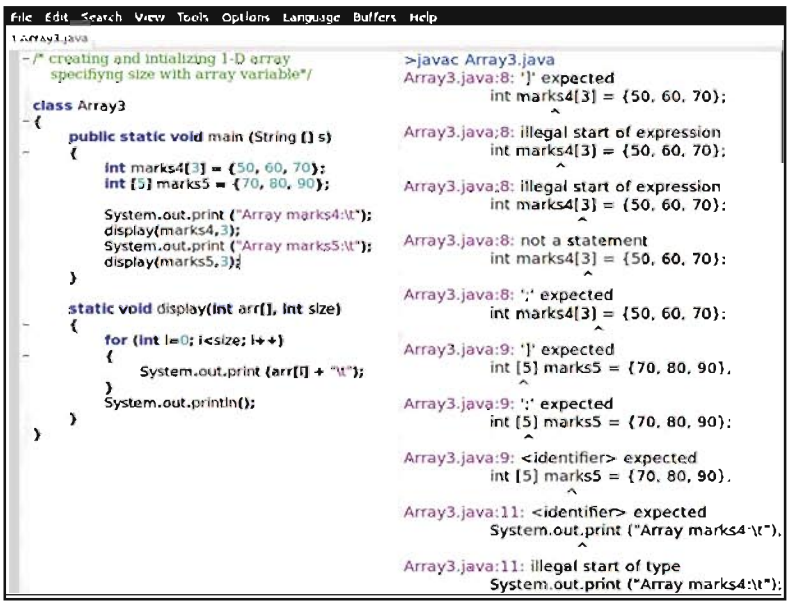
અહીં નોંધ કરો કે આપણે એરે બનાવતા સમયે જો તેના ઘટકોની પ્રારંભિક કિંમતો આપીએ, તો new પ્રક્રિયકના ઉપયોગની જરૂર રહેતી નથી. એરેનું કદ કૌંસમાં આપેલી કિંમતોની સંખ્યા બરાબર છે.

આકૃતિ 9.3માં આપેલો જાવાકોડ એરે બનાવવા માટે તેમજ તેના ઘટકોની પ્રારંભિક કિંમતો આપવા માટેની અલગ-અલગ રીતો દર્શાવે છે. એરેના ઘટકોની કિંમતો પ્રદર્શિત કરવા માટે વાપરેલી display() ક્લાસ મેથડની નોંધ લો. ક્લાસમેથડને static ઘોષિત કરાય છે અને તે ક્લાસના કોઈ પણ ઑબ્જેક્ટ વિના વાપરી શકાય છે.




આકૃતિ 9.3 : એરે-ઑબ્જેક્ટ બનાવવા તથા તેના ઘટકોને પ્રારંભિક કિંમતો આપવાની વિવિધ રીત

જાવામાં આપણે જ્યારે એરે ઘોષિત કરીએ, તે સમયે પરિમાણનું માપ (dimensions) અને તેના ઘટકોની પ્રારંભિક કિંમતો (initial values) બંને એકસાથે આપી શકતા નથી. આકૃતિ 9.4માં આ વિધાનને સમજાવતો કોડ દર્શાવ્યો છે. જો આપણે એરે ચલ પ્રારંભિક કિંમતો વગર (without initialization) ઘોષિત કરીએ, તો આકૃતિ 9.1માં દર્શાવ્યા પ્રમાણે સંદર્ભ રાખવા માટેનો માત્ર ચલ બને છે. એ સમયે એરે-ઑબ્જેક્ટ બનતો નથી, એટલે કે એરેના ઘટકો માટે મેમરી ફાળવવામાં આવતી નથી. જો આપણે એરેના ઘટકો વાપરવા માટે પ્રયત્ન કરીએ તો તે ભૂલમાં પરિણમે છે.




આકૃતિ 9.4 : એરે-ઑબ્જેક્ટના ઘટકોને પ્રારંભિક કિંમતો આપતા સમયે તેનું કદ જણાવવું અમાન્ય છે.

ઍરેનો દરેક ઘટક એક અલગ ચલ છે, જેનો ઇન્ડેક્સ (index) વાપરીને નિર્દેશ કરી શકાય છે. ઍરેના ઘટકની કિંમત બદલવા માટે અન્ય ચલની જેમ આપણે ઘટક ચલને ઍસાઇન્મેન્ટ વિધાનમાં ડાબી બાજુએ વાપરી શકીએ છીએ. ઉદાહરણ તરીકે, marks[3] = 56; ચાલો, આપણે 10 અપૂર્ણાંક સંખ્યાઓની સરેરાશ (મધ્યક) શોધવા માટેનો પ્રોગ્રામ લખીએ અને તેનો અમલ કરીએ. આકૃતિ 9.5માં આપેલું કોડલિસ્ટિંગ અને તેના આઉટપુટનો અભ્યાસ કરો.

ArrayAvg.java - SciTE

File Edit Search View Tools Options Language Buffers Help

1 ArrayAvg.java

```
/* compute average of 10 numbers */
class ArrayAvg
{
    public static void main (String [] s)
    {
        double numbers [] = { 10.5, 20.6, 30.8, 15.5,
                              17.3, 25.5, 27.2,
                              20, 30, 18.5};

        byte ctr;
        double sum=0, avg;

        System.out.println ("list of numbers is");
        for (ctr=0; ctr<10; ctr++)
        {
            System.out.println (numbers[ctr]);
            sum = sum + numbers[ctr];
        }
        avg = sum/10;
        System.out.println ("\nAverage of above numbers is "
                            + avg);
    } // main
} // class
```

```
>javac ArrayAvg.java
>Exit code: 0
>java -cp . ArrayAvg
list of numbers is
10.5
20.6
30.8
15.5
17.3
25.5
27.2
20.0
30.0
18.5

Average of above numbers is 21.59
>Exit code: 0
```

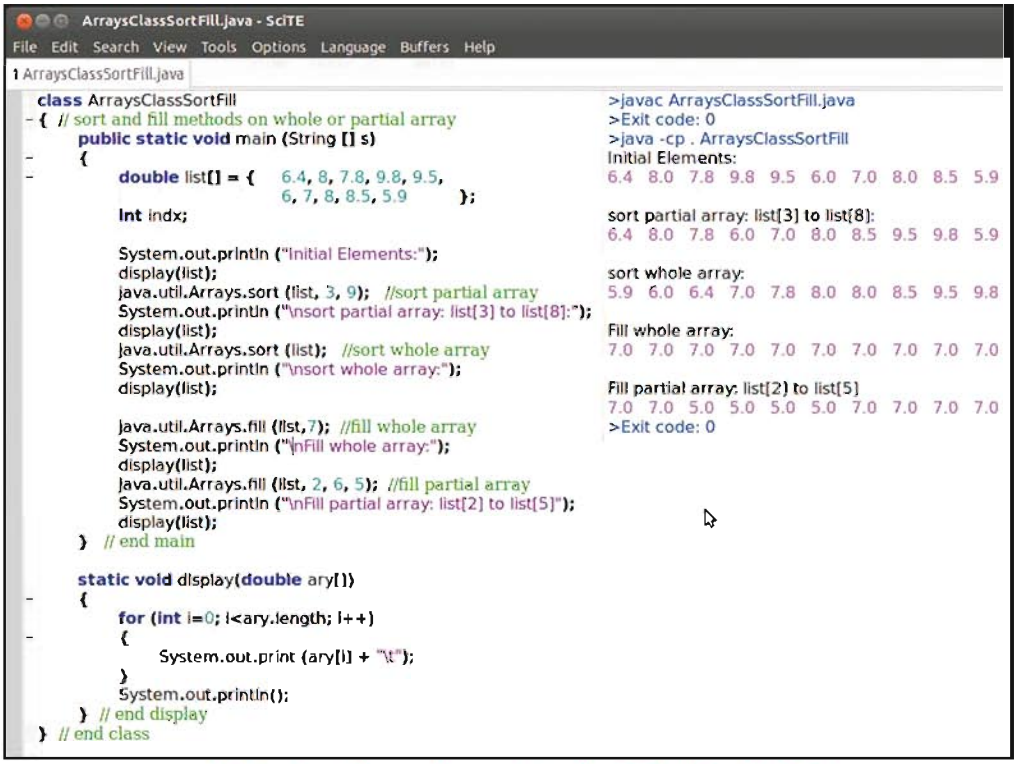
**આકૃતિ 9.5 : 1-D ઍરે અને લૂપનો ઉપયોગ કરીને 10 સંખ્યાની સરેરાશ શોધવાનો પ્રોગ્રામ**

અહીં, આપણને સમાન પ્રકારના અલગ-અલગ ઘટકો ઉપર એક જ જાતની ક્રિયા 'સંખ્યાને સરવાળામાં ઉમેરો', વારંવાર કરવાની જરૂર પડે છે. એકસાથે 10 ચલને ઘોષિત કરવા અને લૂપનો ઉપયોગ કરી ઍરેના જુદા-જુદા ઘટકો ઉપર એકસમાન પ્રક્રિયા કરવામાં ઍરેનો ઉપયોગ મદદરૂપ થાય છે.

ઍરે ઉપર આપણે અન્ય અનેક વિવિધ પ્રક્રિયાઓ કરી શકીએ છીએ. ઉદાહરણ તરીકે, બે ઍરેની સરખામણી કરવી, એક ઍરેના બધા ઘટકોની અન્ય ઍરેમાં નકલ કરવી, કોઈ ચોક્કસ ઘટક ઍરેમાં શોધવો, ઍરેના ઘટકોને ક્રમાનુસાર ગોઠવવા વગેરે. આ પ્રકારનાં બધાં કાર્યો માટે જેમ આપણે ઘટકોનું સરેરાશ શોધવા માટે લખી હતી તે પ્રમાણે પ્રોસીજર (procedures) લખી શકીએ. આ માટેનો કોડ આપણે જાતે લખવાને બદલે જાવા દ્વારા પૂરી પાડવામાં આવેલી java.util.Arrays ક્લાસની વિવિધ static methodsનો આપણે ઉપયોગ કરી શકીએ છીએ. જાવામાં ઍરેને Arrays classની મેથડ (methods)નો ઉપયોગ કરવાની સગવડ આપે છે. આકૃતિ 9.6માં દર્શાવેલું કોડલિસ્ટિંગ અને તેનો આઉટપુટ java.util.Arrays ક્લાસની sort() અને fill() મેથડનો ઉપયોગ કઈ રીતે કરી શકાય તે દર્શાવે છે.

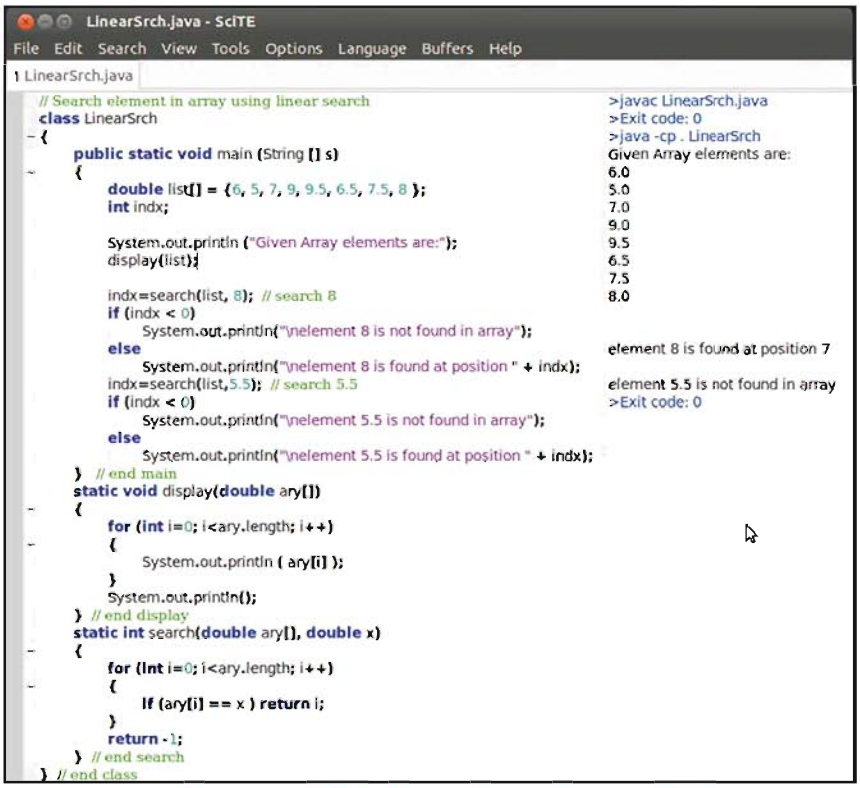
આપણે સમગ્ર ઍરે કે તેના અમુક ભાગને ક્રમાનુસાર ગોઠવવા માટે (શૉર્ટ કરવા) sort() મેથડ વાપરી શકીએ છીએ. આપણે જ્યારે મેથડના આર્ગ્યુમેન્ટ (argument) તરીકે ફક્ત ઍરે આપીએ, ત્યારે તે સમગ્ર ઍરેને (ઍરેના બધા ઘટકોને) સૉર્ટ કરે છે. જો આપણે ઍરે, આરંભસ્થાન, અંતિમસ્થાન એમ પણ આર્ગ્યુમેન્ટ આપીને મેથડ વાપરીએ, તો તે આરંભસ્થાનના ઘટકથી અંતિમસ્થાનથી આગળના ઘટક સુધીના આંશિક ઍરેને શૉટ (sort) કરશે. ઉદાહરણ તરીકે, java.util.Arrays.sort (list, 1, 5)નો ઉપયોગ કરવાથી તે list[1] થી list[5-1] સુધીના ઍરેના ઘટકોને શૉટ કરશે. જુઓ આકૃતિ 9.6.

સમગ્ર કે આંશિક ઍરેને કોઈ ચોક્કસ કિંમતથી ભરવા માટે 'fill' મેથડનો ઉપયોગ થાય છે. જ્યારે મેથડ ઍરે અને કિંમત એમ બે આર્ગ્યુમેન્ટ સાથે ઇન્વોક કરવામાં આવે, ત્યારે ઍરેના બધા ઘટકોમાં તે જણાવેલી કિંમત ભરવામાં આવે છે. જ્યારે આ મેથડ ઍરે, આરંભસ્થાન, અંતિમ સ્થાન અને કિંમત એમ ચાર આર્ગ્યુમેન્ટ સાથે ઇન્વોક કરવામાં આવે છે, ત્યારે તે આરંભસ્થાનથી શરૂ કરી અંતિમ સ્થાનથી આગળના ઘટક સુધી આપેલી કિંમત વડે ભરી દે છે. ઉદાહરણ તરીકે, fill (list, 7) ઍરેના બધા ઘટકો કિંમત 7 વડે ભરવામાં આવે છે, જ્યારે fill(list, 2, 6, 5) ઍરેના ઘટકો list[2] થી list[6-1]માં કિંમત 5 ભરવામાં આવે છે. જુઓ આકૃતિ 9.6.




**આકૃતિ 9.6 : એરેક્લાસની sort() અને fill() મેથડનો ઉપયોગ**

આપેલી કિંમતવાળો ઘટક એરેમાં શોધવા માટે એરે ક્લાસમાં binarySearch() મેથડ ઉપલબ્ધ છે. સુરેખશોધ (linear search) પદ્ધતિથી આપેલી કિંમતવાળો ઘટક એરેમાં શોધવા માટે આપણે આપણી પોતાની મેથડ લખીશું. સુરેખશોધ પદ્ધતિ એરેના એક પછી એક ઘટકોને આપેલી કિંમત સાથે ક્રમાનુસાર સરખાવે છે. આકૃતિ 9.7માં આપેલા કોડલિસ્ટિંગ અને તેના આઉટપુટનો અભ્યાસ કરો. આ પદ્ધતિ મુજબ જો એરેમાં આપેલી કિંમતવાળો ઘટક મળે, તો તેનું સૂચક સ્થાન (index position) પરત કરે છે, નહિ તો -1 કિંમત પરત કરે છે.




**આકૃતિ 9.7 : એરેમાં ચોક્કસ કિંમતવાળો ઘટક શોધવો**

## 2-D એરે (2-D Array)

દ્વિ-પરિમાણીય (2-D) એરે હાર અને સ્તંભ સ્વરૂપે કોષ્ટકીય માહિતીનો સંગ્રહ કરવા માટે વપરાય છે. ઉદાહરણ તરીકે, પાંચ વિદ્યાર્થીઓના ત્રણ કસોટીના ગુણ દર્શાવવા માટે આપણે આકૃતિ 9.8માં દર્શાવ્યા મુજબ પાંચ હાર અને ત્રણ સ્તંભવાળી કોષ્ટકીય ગોઠવણીનો ઉપયોગ કરી શકીએ. ગણિતમાં આપણે તેને પાંચ હાર અને ત્રણ સ્તંભવાળું શ્રેણિક (matrix) કહીએ છીએ, જેમાં દરેક ઘટકમાં ગુણ હોય છે.

| Students | Test1 | Test2 | Test3 |
|---|---|---|---|
| 1 | 50 | 60 | 70 |
| 2 | 35 | 30 | 50 |
| 3 | 70 | 75 | 80 |
| 4 | 80 | 85 | 90 |
| 5 | 40 | 50 | 55 |

આકૃતિ 9.8 : 2-D એરેની કોષ્ટકીય રજૂઆત

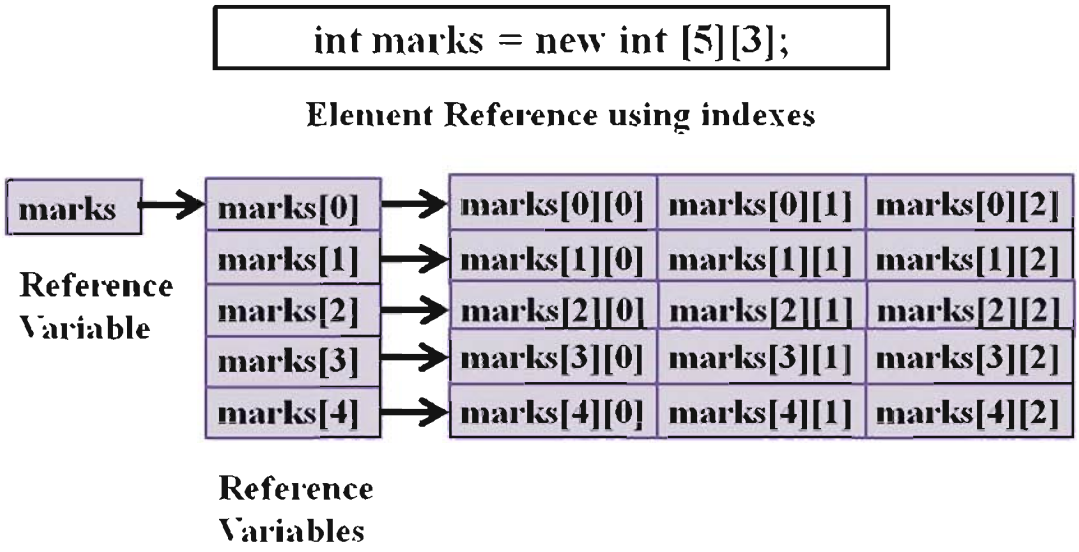
જાવામાં 2-D એરે ઘોષિત કરવા માટે એરેનું નામ અને મોટા કૌંસની બે જોડી [ ] [ ] કે જે અનુક્રમે પરિમાણો હાર (row) અને સ્તંભ (column)ના કદનો ઉલ્લેખ કરે છે, તેનો ઉપયોગ થાય છે. ઉદાહરણ તરીકે, સળંગ મેમરીમાં 5 x 3 = 15 પૂર્ણાંક કિંમતોનો સંગ્રહ કરવા માટે marks નામનો એરે ઘોષિત કરવા માટે તેમજ બનાવવા માટે નીચે આપેલું વિધાન વપરાય છે :

```
int marks [][] = new int [5][3];
```

અહીં, તાર્કિક દૃષ્ટિએ એરે એ 5 હાર અને 3 સ્તંભ સાથેનું એક કોષ્ટક છે. ભૌતિક દૃષ્ટિએ તે સળંગ મેમરીમાં સંગ્રહાયેલી 15 પૂર્ણાંક કિંમતો છે, જે 60 બાઇટ જગ્યા મેમરીમાં રોકે છે.

જાવા સીધી રીતે બહુ-પરિમાણીય એરે (multi-dimensional arrays)ની સગવડ પૂરી પાડતી નથી. 2-D એરે બનાવવા માટે આપણે એરેનો એરે (array of array) બનાવવો પડે. પરિમાણની સંખ્યા ઉપર કોઈ મર્યાદા નથી.

marks [5][3]માં એક 5 ઘટકોનો 1-D એરે-ઑબ્જેક્ટ બનાવે છે અને આકૃતિ 9.9માં દર્શાવ્યા પ્રમાણે આ દરેક ઘટક એ 3 પૂર્ણાંક સંખ્યાનો એક 1-D એરે-ઑબ્જેક્ટ છે.




આકૃતિ 9.9 : 2-D એરે એ એક 1-D એરેના એરે તરીકે

2-D ઍરેને ઘોષિત કરવાની અને પ્રારંભિક કિંમતો આપવાની રીત સંપૂર્ણપણે 1-D જેવી જ છે, સિવાય કે પરિમાણની સંખ્યા. 2-D ઍરેમાં દરેક હાર (row) એ 1-D ઍરે ઘટક તરીકે ગણાય છે. આથી, 1-D ઍરેની પ્રારંભિક કિંમતો આપવાની રીતે જ 2-D ઍરેની દરેક હારને પ્રારંભિક કિંમતો આપવા માટે છગડિયા કૌંસ { }ની અંદર અલ્પવિરામથી અલગ પાડેલી તેના ઘટકોની કિંમતો આપવામાં આવે છે. 2-D ઍરેને પ્રારંભિક કિંમતો આપવા માટે તેની દરેક પ્રારંભિક કિંમતો આપેલી હાર (initialized rows)ને અલ્પવિરામ ( , )થી અલગ પાડીને છગડિયા કૌંસ { }માં રાખવામાં આવે છે. 1-D ઍરેની જેમ જ 2-D ઍરે પણ આકૃતિ 9.10ના કોડલિસ્ટિંગમાં દર્શાવ્યા પ્રમાણે વિવિધ રીતોથી ઘોષિત કરી શકાય છે. આકૃતિ 9.11 આ કોડનો આઉટપુટ આપ્યો છે.

```
File  Edit  Search  View  Tools  Options  Language  Buffers  Help
1 Array2D.java *
/* creating and intializing 2-D array */
class Array2D
{
    public static void main (String [] s)
    {
        int marks1[] [ ];
        marks1 = new int [ 5] [3];
        int marks2[][] = new int [5][3];
        int [][] marks3 = new int [5][3];
        int marks4[] []= { {50, 60, 70}, {35, 30, 50},
                           {70, 75, 80}, {80, 85, 90},
                           {50, 50, 55}  };
        Int [] []marks5 = {{50, 60, 70}, {35, 30, 50},
                           {70, 75, 80}, {80, 85, 90},
                           {50, 50, 55}  };

        System.out.print ("2-D Array marks1\n:");
        display(marks1,5,3);
        System.out.print ("2-D Array marks2:\n");
        display(marks2,5,3);
        System.out.print ("2-D Array marks3:\n");
        display(marks3,5,3);
        System.out.print ("2-D Array marks4:\n");
        display(marks4,5,3);
        System.out.print ("2-D Array marks5:\n");
        display(marks5,5,3);
    } // main
    static void display(int arr[][], Int rows, Int cols)
    {
        for (int i=0; i<rows; i++)
        {
            for (int j=0; j<cols; j++)
            {
                System.out.print (arr[i] [j] + "\t");
            }
            System.out.println();
        }
    } // display
} // class
```

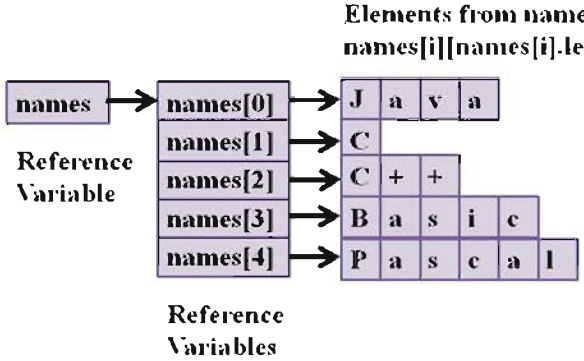
આકૃતિ 9.10 : 2-D ઍરેનું ઉદાહરણ

```
Array2D.java - SciTE
File  Edit  Search  View  Tools  Options  Language  Buffers  Help
1 Array2D.java
>javac Array2D.java
>Exit code: 0
>java -cp . Array2D
2-D Array marks1
:0    0    0
0     0    0
0     0    0
0     0    0
0     0    0
2-D Array marks2:
0     0    0
0     0    0
0     0    0
0     0    0
0     0    0
2-D Array marks3
0     0    0
0     0    0
0     0    0
0     0    0
0     0    0
2-D Array marks4:
50    60   70
35    30   50
70    75   80
80    85   90
50    50   55
2-D Array marks5.
50    60   70
35    30   50
70    75   80
80    85   90
50    50   55
>Exit code: 0
```

આકૃતિ 9.11 : આઉટપુટ

જાવામાં 2-D ઍરેને ઍરેનો ઍરે ગણવામાં આવે છે. આથી, 2-D ઍરેની હારમાં ગમે તેટલી સંખ્યામાં (પરિવર્તનશીલ - variable) સ્તંભ હોઈ શકે. આકૃતિ 9.12માં કમ્પ્યૂટર-પ્રોગ્રામિંગની પાંચ ભાષાઓનાં નામનો સંગ્રહ કરવા માટે અક્ષરોના 2-D ઍરેનો ઉપયોગ દર્શાવ્યો છે.

```
char names [ ][ ] = {   {'J', 'a', 'v', 'a'}, {'C'}, {'C', '+', '+'},
                        {'B', 'a', 's', 'i', 'c'}, {'P', 'a', 's', 'c', 'a', 'l'}   }
```




આકૃતિ 9.12 : અલગ-અલગ સંખ્યામાં સ્તંભ સાથેનો 2-D ઍરે

દરેક નામ અલગ-અલગ હારમાં સંગ્રહવામાં આવ્યાં છે અને દરેક નામમાં અક્ષરોની સંખ્યા અલગ-અલગ છે. આ રીતે દરેક હારનું કદ અલગ-અલગ છે. દરેક હારનું કદ 1-D એરેની 'length' નામની પ્રૉપર્ટી વડે જાણી શકાય છે.

આકૃતિ 9.13માં અલગ-અલગ માપના સ્તંભવાળા 2-D એરેનો ઉપયોગ કેવી રીતે કરી શકાય તે દર્શાવ્યું છે. અહીં આપણે 1-D એરેના ઘટકોની સંખ્યા જાણવા માટે length પ્રૉપર્ટીનો ઉપયોગ કરેલો છે. 2-D એરે માટે તે તેની હારની સંખ્યા પરત કરે છે. 1-D એરે માટે તે આપેલી હારમાં સ્તંભની સંખ્યા પરત કરે છે. અહીં યાદ રાખો કે 2-D એરેમાં ઘટકોની સંખ્યા એ તેમાં રહેલી હારની સંખ્યા છે અને દરેક હાર એ 1-D એરે છે. ટૂંકમાં, length પ્રૉપર્ટી જ્યારે ફક્ત એરેના નામ સાથે વાપરવામાં આવે, ત્યારે તે તેના પ્રથમ પરિમાણનું કદ પરત કરે છે.

Array2Dchar.java - SciTE

File Edit Search View Tools Options Language Buffers Help

1 Array2Dchar.java

```
/* creating and intializing 2-D array */
class Array2Dchar
{
    public static void main (String [] s)
    {
        char names[] []= {
                          {'J','a','v','a'},
                          {'C'},
                          {'C', '+', '+'},
                          {'B', 'a', 's', 'i', 'c'},
                          {'P','a','s','c', 'a', 'l'}
                         };

        System.out.println("Number of elements in 2-D array: "
                           + names.length + "\n");
        System.out.print
                ("Five names stored in 2-D Array of characters:\n");
        display(names,5);
    } // main
        static void display(char arr[][], int rows)
    {
        for (int i=0; i<rows; i++)
        {
            System.out.print ("Row " + i + " have " + arr[i].length +
                              " character elements: ");
            for (int j=0; j<arr[i].length; j++)
            {
                System.out.print ( arr[i] [j] );
            }
            System.out.println();
        }
    } // display
} // class
```

```
>javac Array2Dchar.java
>Exit code: 0
>java -cp . Array2Dchar
Number of elements in 2-D array: 5

Five names stored in 2-D Array of characters:
Row 0 have 4 character elements: Java
Row 1 have 1 character elements: C
Row 2 have 3 character elements: C++
Row 3 have 5 character elements: Basic
Row 4 have 6 character elements: Pascal
>Exit code: 0
```

**આકૃતિ 9.13 : અલગ-અલગ સંખ્યામાં સ્તંભ ધરાવતા 2-D એરેનો પ્રોગ્રામમાં ઉપયોગ**

હવે પછી આપણે જોઈશું કે અક્ષરોના 1-D એરેને જાવામાં ઉપલબ્ધ string ક્લાસના ઉપયોગથી વ્યાખ્યાયિત કરી શકાય છે. આકૃતિ 9.14માં આપણે નામનો સંગ્રહ કરવા માટે બાઇટનો 2-D એરે વાપર્યો છે. અહીં તે દર્શાવે છે કે char પ્રકારના ડેટાને બાઇટ પ્રકારમાં પરિવર્તિત કર્યા છે અને નામના અક્ષરોને તેની અનુરૂપ ASCII કિંમત આપવામાં આવી છે.

File Edit Search View Tools Options Language Buffers Help

1 Array2Dbyte.java

```
/* creating and intializing 2-D array */
class Array2Dbyte
{
    public static void main (String [] s)
    {
        byte names[] []= {
                          {'J','a','v','a'},
                          {67},
                          {'C', '+', '+'},
                          {'B', 'a', 's', 'i', 'c'},
                          {'P','a','s','c', 'a', 'l'}
                         };
        System.out.print ("Five names stored in 2-D Array of bytes:\n");
        display(names,5);
    } // main
        static void display( byte arr[][], int rows)
    {
        for (int i=0; i<rows; i++)
        {
            for (int j=0; j<arr[i].length; j++)
            {
                System.out.print ( arr[i] [j] + "\t" );
            }
            System.out.println();
        }
    } // display
} // class
```

```
>java -cp . Array2Dbyte
Five names stored in 2-D Array of bytes:
74   97   118  97
67
67   43   43
66   97   115  105  99
80   97   115  99   97   108
>Exit code: 0
```

**આકૃતિ 9.14 : 2-D એરે : અક્ષરોને અનુરૂપ પૂર્ણાંક સંખ્યાનો ઉપયોગ કરી બાઇટમાં સંગ્રહ**

**યાદ રાખવાના મુદ્દાઓ :**

- એરે એકસમાન પ્રકારના ડેટાનો સંગ્રહ છે.
- એરેના ઘટકોને તેના દરેક પરિમાણના ઇન્ડેક્સને કૌંસમાં [ ] આપીને વાપરી શકાય છે.
- ઇન્ડેક્સની કિંમત શૂન્યથી શરૂ થાય છે.
- બહુ-પરિમાણીય એરેમાં બીજા, ત્રીજા,...પરિમાણોનું કદ અલગ-અલગ હોઈ શકે છે.
- પરિમાણનું કદ જાણવા માટે 'length' ઍટ્રિબ્યૂટનો ઉપયોગ થાય છે.
- એરેને ઑબ્જેક્ટ તરીકે ગણવામાં આવે છે.
- એરેને તેની પ્રારંભિક કિંમતો આપ્યા વિના ઘોષિત કરવાથી એરે ઑબ્જેક્ટ બનતો નથી.

## સ્ટ્રિંગ (Strings)

સ્ટ્રિંગ એ અક્ષરોની એક શ્રેણી સિવાય બીજું કંઈ જ નથી. આથી અક્ષરોના 1-D એરેને સ્ટ્રિંગ તરીકે ગણી શકાય. આપણે અગાઉ string લિટરલ વાપર્યા છે અને તેનો અભ્યાસ પણ કર્યો છે, કે જેમાં અક્ષરોની શ્રેણીને બેવડા અવતરણ-ચિહ્ન (double quotes) વચ્ચે લખવામાં આવે છે.

સ્ટ્રિંગનો સંગ્રહ કરી શકે તેવા ચલ વાપરવા માટે જાવામાં બે પ્રકારની સ્ટ્રિંગ આપેલી છે, જે 'String' અને 'StringBuffer' નામના બે ક્લાસ વડે નિયંત્રિત થાય છે. આ પ્રકરણમાં આપણે પ્રથમ પ્રકારની સ્ટ્રિંગનો અભ્યાસ કરીશું.

સ્ટ્રિંગ ઑબ્જેક્ટ બનાવવા માટે વપરાતાં કેટલાંક કન્સ્ટ્રક્ટર (constructor) નીચે મુજબ છે :

- String () – ચલ રહિત String () એક પણ અક્ષર વગરનો સ્ટ્રિંગ ઑબ્જેક્ટ બનાવે છે.
- String (char ary[]) – ary ચલનો પ્રારંભિક કિંમત તરીકે ઉપયોગ કરીને સ્ટ્રિંગ ઑબ્જેક્ટ બનાવે છે.
- String (char ary[], int start, int len) – આપેલા પ્રથમ ચલ aryના start સ્થાનથી len જેટલા ઘટકોનો સ્ટ્રિંગ ઑબ્જેક્ટ બનાવે છે.
- String (String strObj) – ચલ તરીકે આપેલા ઑબ્જેક્ટ જેવો જ સ્ટ્રિંગ ઑબ્જેક્ટ બનાવે છે.
- String (string literal) – ચલ તરીકે આપેલા string literalને નિર્દેશ કરતો સ્ટ્રિંગ ઑબ્જેક્ટ બનાવે છે.

આકૃતિ 9.15માં વિવિધ પ્રકારના String ક્લાસના કન્સ્ટ્રક્ટરનો ઉપયોગ દર્શાવેલ છે. ઑબ્જેક્ટ બનાવતા સમયે new પ્રક્રિયકનો ઉપયોગ ભૂલતા નહીં.

String1.Java - SciTE

File Edit Search View Tools Options Language Buffers Help

1 String1.Java

```
/* creating String Objects */
class String1
{
    public static void main (String [] s)
    {
        char name [] = { 'J', 'a', 'v', 'a'};

        String str1 = new String();
        String str2 = new String(name);
        String str3 = new String(name, 1,2);
        String str4 = new String(str3);
        String str5 = new String("Java");

        System.out.println ("str1 is " + str1);
        System.out.println ("str2 is " + str2);
        System.out.println ("str3 is " + str3);
        System.out.println ("str4 is " + str4);
        System.out.println ("str5 is " + str5);
    } // main
} // class
```

```
>javac String1.java
>Exit code: 0
>java -cp . String1
str1 is
str2 is Java
str3 is av
str4 is av
str5 is Java
>Exit code: 0
```

**આકૃતિ 9.15 : વિવિધ કન્સ્ટ્રક્ટરના ઉપયોગ વડે string ક્લાસના ઑબ્જેક્ટ બનાવવા**

જાવામાં અક્ષરનો સંગ્રહ કરવા માટે અક્ષર દીઠ બે બાઇટ વપરાય છે. જગ્યા બચાવવા માટે જો ASCII અક્ષરો હોય, તો char પ્રકારના ડેટા-એરે વાપરવાને બદલે bytes પ્રકારના ડેટા-એરેનો ઉપયોગ કરવો જોઈએ. આ માટે આપણે કન્સ્ટ્રક્ટરમાં ચલ તરીકે byteનો એરે વાપરી શકીએ. આકૃતિ 9.15માં આપેલા પ્રોગ્રામના એરે char name[ ]ને બદલી byte name[ ] કરો અને પ્રોગ્રામનો અમલ કરવા કોશિશ કરો.

**નોંધ :** લિટરલ (literals)નો મેમરીમાં સંગ્રહ કરવામાં આવે છે. જ્યારે બે સ્ટ્રિંગ-ઑબ્જેક્ટ એકસરખા સ્ટ્રિંગ લિટરલ વાપરીને બનાવ્યા હોય, ત્યારે બીજા ઑબ્જેક્ટ માટે મેમરીમાં જગ્યા ફાળવાતી નથી. બંને ઑબ્જેક્ટ એક જ મેમરી સ્થાનનો ઉલ્લેખ કરે છે.

જ્યારે સ્ટ્રિંગ ઑબ્જેક્ટ new પ્રક્રિયક વાપરીને બનાવવામાં આવે છે, ત્યારે બંને સ્ટ્રિંગ એકસમાન હોય, તોપણ મેમરીમાં અલગ જગ્યા ફાળવવામાં આવે છે. આકૃતિ 9.16માં આપેલું કોડલિસ્ટિંગ આ પ્રકારનું ઉદાહરણ છે.

String2.java - SciTE

File Edit Search View Tools Options Language Buffers Help

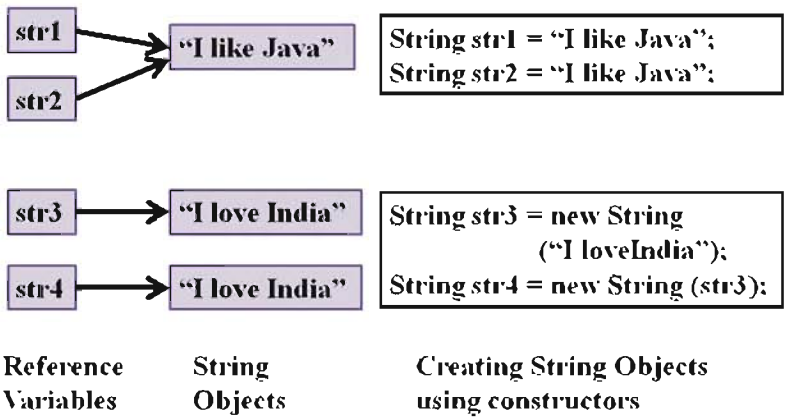
1 String2.java

```
/* String Objects */
class String2
{
      public static void main (String [] s)
      {
            String str1 = "I like Java";
            String str2 = "I like Java";
            String str3 = new String("I love India");
            String str4 = new String(str3);

            System.out.println ("str1==str2: "
                              + (str1==str2));
            System.out.println ("str3==str4: "
                              + (str3==str4));

            System.out.println ("str1: " + str1);
            System.out.println ("str2: " + str2);
            System.out.println ("str3: " + str3);
            System.out.println ("str4: " + str4);
      } // main
} // class
```

```
>javac String2.java
>Exit code: 0
>java -cp . String2
str1==str2: true
str3==str4: false
str1: I like Java
str2: I like Java
str3: I love India
str4: I love India
>Exit code: 0
```

**આકૃતિ 9.16 : સ્ટ્રિંગ લિટરલ અને new પ્રક્રિયક વાપરીને એકસરખા સ્ટ્રિંગ ઑબ્જેક્ટ બનાવવા**

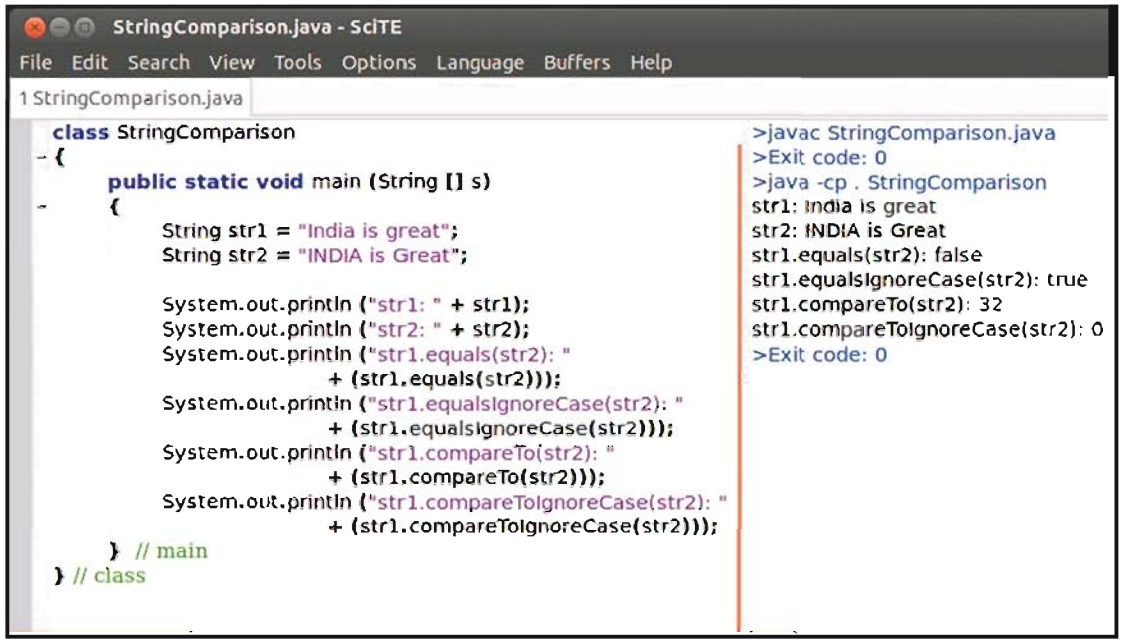
આકૃતિ 9.16માં આપેલા કોડલિસ્ટિંગમાં "str1 == str2" એ str1 અને str2 એમ બે રેફરન્સ ચલમાં સંગ્રહ કરેલી વિગતની તુલના કરે છે અને નહિ કે str1 અને str2 વડે નિર્દેશ કરેલા ઑબ્જેક્ટની. આ બાબતની નોંધ કરો. અહીં str1 અને str2 ઑબ્જેક્ટ new પ્રક્રિયક વિના અને એકસરખા સ્ટ્રિંગ લિટરલ વાપરીને બનાવ્યા છે. આથી બંને ચલ str1 એ બનાવેલા ઇન્સ્ટન્સનો જ નિર્દેશ કરે છે, આકૃતિ 9.17માં દર્શાવ્યા પ્રમાણે str2 ઑબ્જેક્ટ માટે અલગ મેમરીની ફાળવણી કરવામાં આવી નથી. આ જ પ્રોગ્રામમાં str3 અને str4 સ્ટ્રિંગ ઑબ્જેક્ટ new પ્રક્રિયકનો ઉપયોગ કરીને બનાવ્યા છે. બંને ઑબ્જેક્ટમાંની માહિતી સરખી છે, પરંતુ બંને અલગ-અલગ મેમરીસ્થાનનો સંદર્ભ ધરાવે છે. આ બંને ઑબ્જેક્ટ માટે અલગ-અલગ મેમરીની ફાળવણી કરવામાં આવી છે, જુઓ આકૃતિ 9.17.




**આકૃતિ 9.17 : એકસમાન સ્ટ્રિંગ**

## સ્ટ્રિંગ ક્લાસમેથડ (String Class Methods)

બે સ્ટ્રિંગની તુલના કરવી, સ્ટ્રિંગની લંબાઈ મેળવવી, બે સ્ટ્રિંગને જોડવી, સ્ટ્રિંગમાંથી આંશિક સ્ટ્રિંગ (સબસ્ટ્રિંગ) મેળવવી, સ્ટ્રિંગને પરાવર્તિત કરવી, સ્ટ્રિંગનું વિભાજન કરવું, અક્ષર અથવા અક્ષરસમૂહને સ્ટ્રિંગમાં શોધવા વગેરે કાર્યો માટે સ્ટ્રિંગ-ક્લાસ કેટલીક મેથડ પૂરી પાડે છે. આપણે પ્રોગ્રામ દ્વારા કેટલીક મેથડનો ઉપયોગ કરવાના ઉદાહરણ જોઈશું. આકૃતિ 9.18માં લખેલો પ્રોગ્રામ સ્ટ્રિંગ અથવા સબસ્ટ્રિંગની સરખામણી કરવા માટેની વિવિધ રીત દર્શાવે છે.




આકૃતિ 9.18 : સ્ટ્રિંગ ક્લાસની મેથડનો ઉપયોગ કરી સ્ટ્રિંગની તુલના

કોષ્ટક 9.1માં સ્ટ્રિંગની તુલના કરવા માટેની વિવિધ મેથડનું વર્ણન અને વાક્યરચના દર્શાવી છે.

| મેથડ | વર્ણન |
|---|---|
| boolean equals (String str) | મેથડકૉલ કરતી સ્ટ્રિંગ અને પ્રાચલ str (ઑબ્જેક્ટ) સરખાં હોય, તો true પરત કરે છે. |
| boolean equalsIgnoreCase (String str) | મેથડકૉલ કરતી સ્ટ્રિંગ અને પ્રાચલ str ઑબ્જેક્ટના અક્ષરોના કેસ (case) (કૅપિટલ અને બીજી એબીસીડી)ની અવગણના કરીને તુલના કરે છે. જો બંને સરખાં હોય, તો true પરત કરે છે. (case insensitive) |
| int compareTo (String str) | જો મેથડકૉલ કરનાર સ્ટ્રિંગ ઑબ્જેક્ટ પ્રાચલ strની તુલનાએ સમાન હોય, મોટી હોય અથવા નાની હોય, તો અનુક્રમે 0, >0, <0 પૂર્ણાંક સંખ્યા પરત કરે છે. |
| int compareToIgnoreCase (String str) | CompareTo મેથડ પ્રમાણે જ પરંતુ case પ્રત્યે અસંવેદનશીલ |

કોષ્ટક 9.1 : સ્ટ્રિંગની તુલના માટે સ્ટ્રિંગ ક્લાસની વિવિધ મેથડ

સ્ટ્રિંગ ક્લાસ નીચે જણાવેલાં અન્ય કાર્યો કરવા માટે અનેક મેથડ પૂરી પાડે છે. આમાંની કેટલીક કોષ્ટક 9.2માં આપેલી છે.

- સ્ટ્રિંગનો અમુક ભાગ અલગ કરવો.
- અક્ષરો કે શબ્દસમૂહ (સબસ્ટ્રિંગ) બદલવા.
- સ્ટ્રિંગનું સબસ્ટ્રિંગમાં વિભાજન કરવું.
- અક્ષરોની સંખ્યા મેળવવી.
- આપેલા ઇન્ડેક્સસ્થાન પરનો અક્ષર મેળવવો.

- સ્ટ્રિંગને bytes પ્રકારના એરેમાં પરિવર્તિત કરવી.
- સ્ટ્રિંગના અક્ષરોને સ્મૉલ અથવા કૅપિટલ કરવા.
- સ્ટ્રિંગના અંતમાં બીજી સ્ટ્રિંગ ઉમેરવી.
- સ્ટ્રિંગ કે તેના ભાગની નકલ કરવી.

| મેથડ | વર્ણન |
|---|---|
| int length() | મેથડકૉલ કરતી સ્ટ્રિંગ ઑબ્જેક્ટમાં રહેલા અક્ષરોની સંખ્યા પરત કરે છે. |
| char indexAt (int index) | મેથડકૉલ કરતી સ્ટ્રિંગ ઑબ્જેક્ટમાં પ્રાચલ ઇન્ડેક્સ સ્થાને આવેલા અક્ષરને પરત કરે છે. ઇન્ડેક્સ શૂન્યથી ગણાય છે. |
| byte [] getBytes() | મેથડકૉલ કરતી સ્ટ્રિંગના અક્ષરોને byte એરેમાં પરત કરે છે. |
| void getChars (int fromIndx, int toIndx, char target [], int targetIndx) | મેથડકૉલ કરતી સ્ટ્રિંગ ઑબ્જેક્ટના fromIndx સ્થાનથી toIndx-1 સ્થાન સુધીના અક્ષરોની લક્ષ્ય એરેમાં targetIndx સ્થાન પછી નકલ કરે છે. |
| String concat(String str) | મેથડકૉલ કરતી સ્ટ્રિંગના અંતમાં પ્રાચલ strના અક્ષરોને ઉમેરીને સ્ટ્રિંગ ઑબ્જેક્ટ પરત કરે છે. |
| String toLowerCase() | મેથડકૉલ કરતી સ્ટ્રિંગના બધા જ અક્ષરોને સ્મૉલ અક્ષરોમાં બનાવીને સ્ટ્રિંગ પરત કરે છે. |
| String toUpperCase() | મેથડકૉલ કરતી સ્ટ્રિંગના બધા જ અક્ષરોને કૅપિટલ અક્ષરમાં પરિવર્તિત કરીને સ્ટ્રિંગ પરત કરે છે. |

કોષ્ટક 9.2 : સ્ટ્રિંગક્લાસની અન્ય મેથડ

આકૃતિ 9.19માં આપેલાં કોડલિસ્ટિંગમાં આમાંની કેટલીક મેથડનો ઉપયોગ દર્શાવ્યો છે.

StringConversion.java - SciTE

File Edit Search View Tools Options Language Buffers Help

1 StringConversion.java

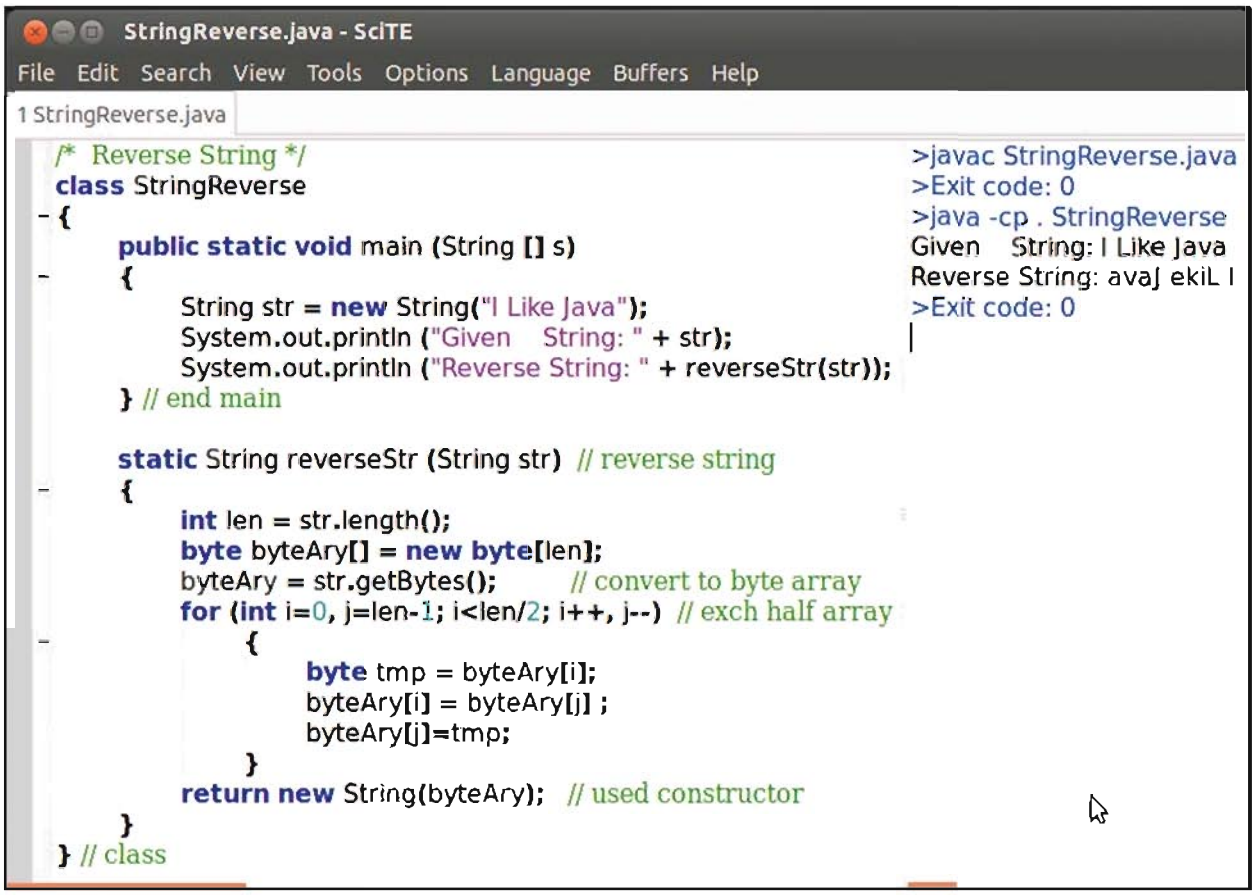
```
/* String conversions */
class StringConversion
{
    public static void main (String [] s)
    {
        String str1 = "I like Java";
        String strAry[] = new String[5];
        byte byteAry[] = new byte[20];

        System.out.println ("Given string is \"" + str1 + "\"");
        System.out.println ("String in lowercase: \""
                            + str1.toLowerCase()+ "\"");
        System.out.println ("String in uppercase: \""
                            + str1.toUpperCase()+ "\"");

        System.out.println("\nString converted to array of bytes");
        byteAry = str1.getBytes();
        for (int i=0; i<byteAry.length; i++)
            System.out.println ( byteAry[i] ) ;
    } // main
} // class
```

```
>javac StringConversion.java
>Exit code: 0
>java -cp . StringConversion
Given string is "I like Java"
String in lowercase:  "i like java"
String in uppercase:  "I LIKE JAVA"

String converted to array of bytes
73
32
108
105
107
101
32
74
97
118
97
>Exit code: 0
```

આકૃતિ 9.19 : સ્ટ્રિંગને પરિવર્તિત કરતી મેથડનો ઉપયોગ

**નોંધ :** એરે ચલ માટે length એ ઍટ્રિબ્યૂટ કે પ્રોપર્ટી છે, જ્યારે સ્ટ્રિંગ ઑબ્જેક્ટ માટે length એ મેથડ છે. આથી સ્ટ્રિંગ ઑબ્જેક્ટ સાથે length મેથડ વાપરતા સમયે () કૌંસનો ઉપયોગ કરવો જરૂરી છે.

જાવામાં સ્ટ્રિંગ ક્લાસમાં સ્ટ્રિંગને ઉલટાવવા માટે કોઈ મેથડ ઉપલબ્ધ નથી. આથી, આપણે સ્ટ્રિંગને ઉલટાવવા માટે પ્રોગ્રામ લખીએ. આકૃતિ 9.20માં આપેલા કોડલિસ્ટિંગ અને તેના આઉટપુટનો અભ્યાસ કરો.



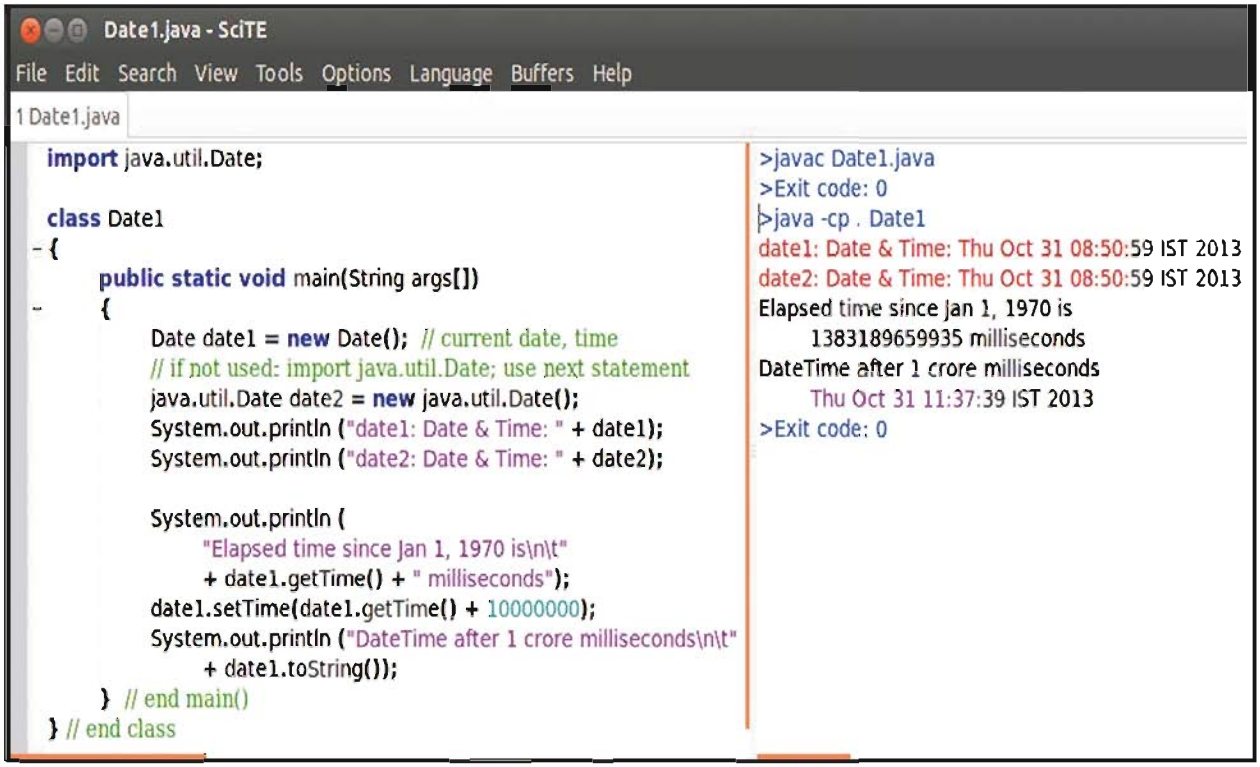

**આકૃતિ 9.20 : સ્ટ્રિંગને ઉલટાવવી**

અહીં, આપણે વપરાશકર્તા વ્યાખ્યાયિત reverseStr નામની મેથડ લખી છે, જે નીચે જણાવેલાં પગલાં પ્રમાણે સ્ટ્રિંગને ઉલટાવે છે :

1. સ્ટ્રિંગક્લાસની length() મેથડ વાપરીને સ્ટ્રિંગમાં રહેલા અક્ષરોની સંખ્યા (સ્ટ્રિંગની લંબાઈ) નક્કી કરો.
2. સ્ટ્રિંગક્લાસની getBytes() મેથડના ઉપયોગથી સ્ટ્રિંગને byte એરેમાં પરિવર્તિત કરો.
3. byte એરેમાં ડાબી બાજુના અડધા ઘટકોની (ડાબી બાજુથી જમણી બાજુ તરફ) જમણી બાજુના અડધા ઘટકો (છેલ્લે જમણી બાજુથી ડાબી બાજુ તરફ) સાથે અદલાબદલી કરો.
4. કન્સ્ટ્રક્ટર વાપરીને byte એરેમાંથી સ્ટ્રિંગ ઑબ્જેક્ટ બનાવો અને તેને પરત કરો.

## Date ક્લાસ (Date Class)

આપણે જાવામાં ઉપલબ્ધ String ક્લાસ અને Arrays ક્લાસનો અભ્યાસ કર્યો. જાવાલાઇબ્રેરીમાં java.util નામના પૅકેજમાં Date ક્લાસ પૂરો પાડે છે. Date ક્લાસ તારીખ અને સમય બંનેનો સમાવેશ કરે છે અને મિલિસેકન્ડ સુધીની ચોકસાઈ સહિત કિંમત જણાવે છે. આકૃતિ 9.21માં કોડલિસ્ટિંગ અને તેના આઉટપુટ સાથે Date ક્લાસનો ઉપયોગ દર્શાવ્યો છે.




**આકૃતિ 9.21 : Date ક્લાસનો ઉપયોગ**

કોષ્ટક 9.3માં Date ક્લાસની કેટલીક મેથડની યાદી આપેલી છે.

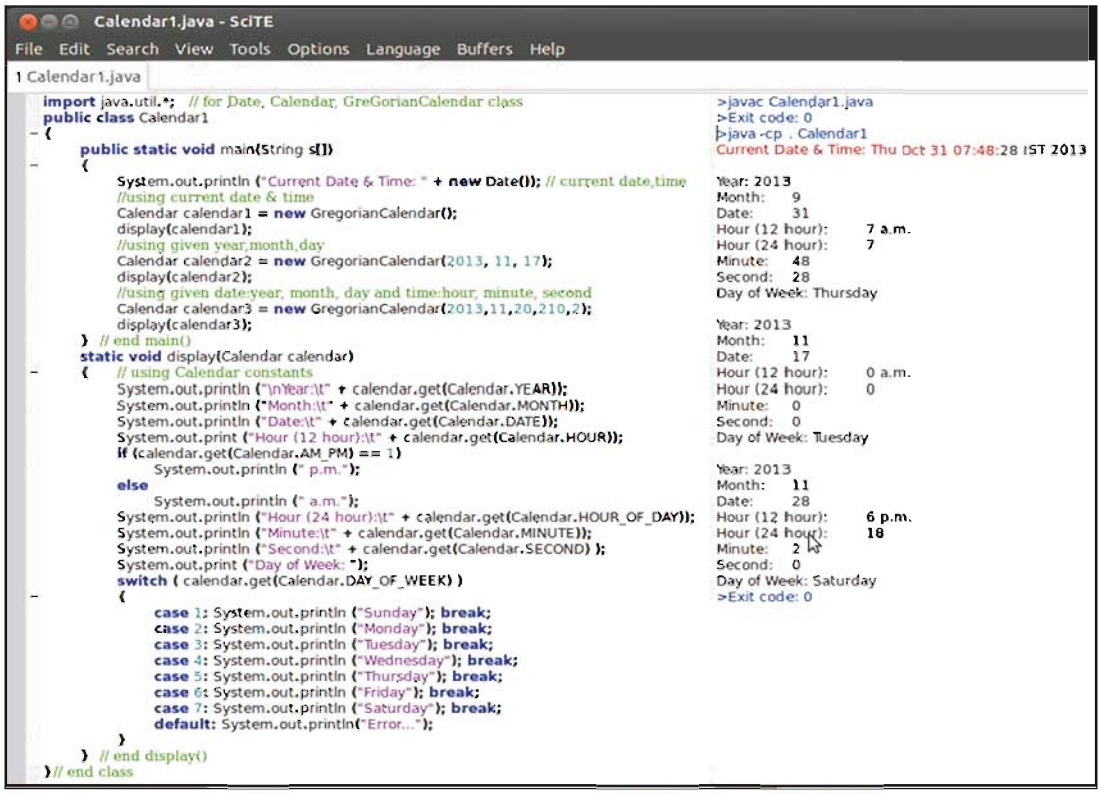
| મેથડ | વર્ણન |
|---|---|
| Date() | સિસ્ટમના તત્કાલીન સમયનો ઉપયોગ કરીને Date ઑબ્જેક્ટ બનાવે છે. |
| Date (long elapsedTime) | જાન્યુઆરી, 1, 1970 GMTથી પ્રાચલમાં આપેલા સમય વચ્ચેનો સમયગાળો મિલિસેકન્ડમાં ગણીને Date ઑબ્જેક્ટ બનાવે છે. |
| String toString() | Date ઑબ્જેક્ટના તારીખ અને સમયને સ્ટ્રિંગમાં રજૂ કરી તે સ્ટ્રિંગ પરત કરે છે. |
| long getTime() | જાન્યુઆરી 1, 1970 GMTથી અત્યાર સુધીની મિલિસેકન્ડ પરત કરે છે. |
| void setTime (long elapsedTime) | પ્રાચલમાં આપેલા વિતેલા સમયનો ઉપયોગ કરી નવી તારીખ અને સમય સેટ કરે છે. |

**કોષ્ટક 9.3 : Date ક્લાસની વિવિધ મેથડ**

## Calendar ક્લાસ (Calendar Class)

Date ક્લાસની જેમ Calendar ક્લાસ પણ java.util પૅકેજમાં આપવામાં આવ્યો છે. આ ક્લાસ વર્ષ, માસ, તારીખ, કલાક, મિનિટ અને સેકન્ડ જેવી કૅલેન્ડરની વિગતવાર માહિતી મેળવવા માટે વાપરી શકાય છે. અહીં, આપણે calendar ક્લાસના સબક્લાસ GregorianCalendarના ઉપયોગ બાબત જોઈશું.

આકૃતિ 9.22માં ઉદાહરણ રૂપે Calendar ક્લાસનો ઉપયોગ કરતો પ્રોગ્રામ અને તેનું આઉટપુટ દર્શાવ્યું છે. અહીં તારીખ અને સમયના વિવિધ ઘટક પ્રદર્શિત કરવા માટે વપરાશકર્તાએ વ્યાખ્યાયિત display() ક્લાસમેથડનો ઉપયોગ કર્યો છે, તેની નોંધ કરો.




**આકૃતિ 9.22 : Calendar ક્લાસ અને તેના અચલનો ઉપયોગ**

get મેથડની જેમ set મેથડ વાપરીને Calendar ક્લાસના ક્ષેત્ર અચલ (field constants)ની કિંમત આપણે સેટ કરી શકીએ છીએ. ઉદાહરણ તરીકે, જો Calendar ક્લાસનો calendar ઑબ્જેક્ટ હોય, તો 'calendar.set(Calendar.DATE, 20)' મેથડકૉલનો અમલ કરતાં મહિનાની તારીખ 20 સેટ થશે.

કોષ્ટક 9.4માં Calendar ક્લાસમાં વ્યાખ્યાયિત પૂર્ણાંક સંખ્યાવાળા અચલોની યાદી આપેલી છે. આ અચલોને અર્થપૂર્ણ અને સ્વયંસ્પષ્ટ નામો આપવામાં આવેલાં છે :

| અચલ | વર્ણન |
|---|---|
| YEAR | કૅલેન્ડરનું વર્ષ |
| MONTH | કૅલેન્ડરનો મહિનો (જાન્યુઆરી માટે 0 અને ડિસેમ્બર માટે 11) |
| DATE | મહિનાનો દિવસ |
| DAY_OF_MONTH | મહિનાનો દિવસ (DATE સમાન) |
| HOUR | 12 કલાકની સંકેતલિપિ પ્રમાણે કલાક |
| HOUR_OF_DAY | 24 કલાકની સંકેતલિપિ પ્રમાણે કલાક |
| MINUTE | મિનિટ |
| SECOND | સેકન્ડ |
| AM_PM | AM માટે 0 અને PM માટે 1 |
| DAY_OF_WEEK | અઠવાડિયામાં દિવસનો ક્રમ (રવિવાર માટે 1 અને શનિવાર માટે 7) |
| WEEK_OF_MONTH | મહિનામાં અઠવાડિયાનો ક્રમ |
| WEEK_OF_YEAR | વર્ષમાં અઠવાડિયાનો ક્રમ |
| DAY_OF_YEAR | વર્ષમાં દિવસનો ક્રમ (પ્રથમ દિવસ માટે 1) |

**કોષ્ટક 9.4 : Calendar ક્લાસમાં વ્યાખ્યાયિત અચલ**

## સારાંશ

આ પ્રકરણ ઍરે, સ્ટ્રિંગ અને ડેઇટનો ઉપયોગ કેવી રીતે કરવો, તે સમજાવે છે. એકસમાન પ્રકારના ચલોના સમૂહ માટે એરેનો ઉપયોગ થાય છે. જાવા મૂળભૂત રીતે ફક્ત 1-D એરે પૂરા પાડે છે. 1-D એરેનો ઍરે વાપરીને આપણે વ્યાવહારિક રીતે બહુ-પરિમાણીય ઍરે મેળવી શકીએ છીએ. ઍરેને Arrays ક્લાસના ઑબ્જેક્ટ તરીકે ગણવામાં આવે છે, તેથી એરેનું નામ એ સંદર્ભચલ (reference variable) છે, જે તેના ઘટકો જે જગ્યાએ સંગૃહીત કરવામાં આવ્યા છે તે મેમરી સ્થાનાંકનો ઉલ્લેખ કરે છે. ઍરે-ઑબ્જેક્ટ સાથે Arrays ક્લાસની તમામ મેથડ વાપરી શકાય છે. આવી મેથડનાં ઉદાહરણો છે : sort, fill. જાવામાં આપેલા String ક્લાસ અક્ષરોની શ્રેણી સાથે કામ કરવાનું સામર્થ્ય પૂરું પાડે છે. સ્ટ્રિંગ ઉપર કરી શકાતાં કાર્યોનાં ઉદાહરણો : સ્ટ્રિંગની સરખામણી, સ્ટ્રિંગમાં રહેલા અક્ષરોની સંખ્યા શોધવી, સ્ટ્રિંગમાં રહેલા અક્ષરોના કેસને રૂપાંતરિત કરવા. તારીખ અને સમય સાથે કામ કરવા માટે Date અને Calendar ક્લાસ આપવામાં આવ્યા છે. Calendar ક્લાસની મેથડ વાપરીને આપણે વર્ષ, માસ, તારીખ, કલાક, મિનિટ અને સેકન્ડ જેવા ઘટકોની કિંમત મેળવી શકીએ છીએ, તેમજ સેટ કરી શકીએ છીએ.

## સ્વાધ્યાય

**1.** ઍરે વિશે ઉદાહરણ આપી સમજાવો.

**2.** 1-D અને 2-D ઍરે વચ્ચેનો તફાવત જણાવો.

**3.** નીચે જણાવેલા ક્લાસનો ઉપયોગ સમજાવો :

a. String b. Date c. Calendar

**4.** **નીચે આપેલા વિકલ્પમાંથી સાચો વિકલ્પ જણાવો :**

(1) નીચેનામાંથી શું ઍરેની પ્રારંભિક ઇન્ડેક્સ કિંમતનો ઉલ્લેખ કરે છે ?

(a) 0 (b) 1 (c) null (d) ઉપરના તમામ વિકલ્પ

(2) sales[5][12] ઍરેમાં દ્વિતીય પરિમાણનું કદ કેટલું છે ?

(a) 5 (b) 12 (c) 60 (d) 10

(3) sales[5][12] ઍરે માટે sales.length પદાવલી શું પરત કરશે ?

(a) 5 (b) 12 (c) 60 (d) 120

(4) જો પ્રારંભિક કિંમતો આપ્યા વિના sales [5][12] ઍરે વ્યાખ્યાયિત કરવામાં આવે, તો sales[0][0]ની શરૂઆતની કિંમત શું હશે ?

(a) 0 (b) પૂર્વનિર્ધારિત કિંમત (c) કમ્પાઇલેશન એરર (d) 60

(5) String ક્લાસના ઑબ્જેક્ટમાં 'length' શું નિર્દેશ કરે છે ?

(a) ઍટ્રિબ્યૂટ (b) મેથડ (c) ક્લાસ વેરિયેબલ (d) ક્લાસનું નામ

(6) જો 'str' એ String ક્લાસનો ઑબ્જેક્ટ હોય અને તેમાં "Thank GOD" માહિતી હોય, તો str.length()ની કિંમત કેટલી હશે ?

(a) 9 (b) 10 (c) 8 (d) 11

(7) જો આપણે Calendar ક્લાસની get મેથડમાં DAY_OF_WEEK પ્રાચલ તરીકે વાપરીએ, તો કયા પ્રકારની કિંમત પરત થશે ?

(a) int (b) char (c) String (d) boolean

## પ્રાયોગિક સ્વાધ્યાય

1. એક પ્રોગ્રામ લખો, જે દિવસના 6 am થી 6 pm સુધીના દરેક ક્લાકના તાપમાનની 12 કિંમતોનો સંગ્રહ કરવા માટે ઍરેનો ઉપયોગ કરે. આ કિંમતો આપવા માટે પ્રારંભિક કિંમતો અથવા ઍસાઇન્મેન્ટ વિધાનોનો ઉપયોગ કરો. દિવસનું મહત્તમ અને ન્યૂનતમ તાપમાન છાપો.

2. એક પ્રોગ્રામ લખો, જે પાંચ વેચનાર વ્યક્તિઓના અઠવાડિક વેચાણની માહિતીનો સંગ્રહ કરે. વેચનાર વ્યક્તિને તેના અઠવાડિક વેચાણના સરેરાશના પ્રમાણે અઠવાડિક પ્રોત્સાહન રકમ આપવામાં આવે છે. જો સરેરાશ અઠવાડિક વેચાણ ₹ 10,000 સુધી હોય, તો 10 %, સરેરાશ અઠવાડિક વેચાણ ₹ 10000થી ₹ 30,000 સુધીમાં હોય, તો 15 % અને ₹ 30,000થી વધુ હોય, તો 20 % પ્રોત્સાહન રકમ છે. બધી વેચનાર વ્યક્તિઓનું દરરોજનું કુલ વેચાણ શોધો. આ ઉપરાંત વેચનાર વ્યક્તિની અઠવાડિક પ્રોત્સાહન રકમની ગણતરી કરો.

3. એક પ્રોગ્રામ લખો, જે આપેલી સ્ટ્રિંગ palindrome છે કે નહીં તે જણાવે. (સ્ટ્રિંગ palindrome કહેવાય, જો તેને ઉલટાવવાથી પણ સરખી રહે, જેમકે "1771" અને "madam".)

4. આજની તારીખને "DD-MMM-YYYY" સ્વરૂપમાં છાપવાનો પ્રોગ્રામ લખો. ઉદાહરણ તારીખ, 17th November 2013ને 17-Nov-2013 તરીકે છાપો. આ ઉપરાંત અઠવાડિયાનો દિવસ શબ્દમાં છાપો.

5. તમારી જન્મતારીખ અને સમય પ્રમાણે તારીખ અને સમય સેટ કરવાનો પ્રોગ્રામ લખો. તમારા જન્મદિવસે અઠવાડિયાનો કયો દિવસ હતો તે છાપો.

# જાવામાં અપવાદરૂપ પરિસ્થિતિનું વ્યવસ્થાપન 10

સામાન્ય રીતે આપણે એવું માનીએ છીએ કે, કંપાઈલ કરેલો પ્રોગ્રામ એ ક્ષતિરહિત હોય છે અને તેથી સફળતાપૂર્વક તેનો અમલ થઈ શકે છે. પરંતુ એકાદ કિસ્સામાં એવું પણ બને કે આવો ક્ષતિરહિત પ્રોગ્રામ પણ અમલ દરમિયાન અધવચ્ચે અટકી પડે. ઉદાહરણ તરીકે, જો આપણે એવો પ્રોગ્રામ લખ્યો હોય કે જે કોઈ ચોક્કસ વેબસાઈટ સાથે જોડાણ કરીને વેબપેજને ડાઉનલોડ કરે, તો સામાન્ય સંજોગોમાં પ્રોગ્રામ અપેક્ષા મુજબ ચાલશે, પરંતુ ધારો કે, આ પ્રોગ્રામ એવા કમ્પ્યૂટર પર ચલાવવામાં આવે કે જેને ઈન્ટરનેટ સાથે જોડાણ ન હોય, તો પ્રોગ્રામ અનઅપેક્ષિત પરિણામો આપે તેવું બને. જ્યારે કોઈ પ્રોગ્રામમાં આપણે ફાઈલ સાથે કામ કરતાં હોઈએ, ત્યારે પણ આવું જ બનશે. ઉદાહરણ તરીકે, ધારો કે કોઈ પ્રોગ્રામ માત્ર વાંચી શકાય તેવી (Read Only) ફાઈલમાં સુધારા-વધારા કરવા પ્રયત્ન કરતો હોય અથવા એવી ફાઈલને ખોલવાનો પ્રયત્ન કરતો હોય કે જે કમ્પ્યૂટરમાં ઉપલબ્ધ જ ન હોય. આ પ્રકારના કિસ્સાઓને જાવામાં "અપવાદરૂપ પરિસ્થિતિ" (Exceptions) તરીકે ઓળખવામાં આવે છે. અપવાદરૂપ પરિસ્થિતિઓને પરિણામે પ્રોગ્રામ અપેક્ષિત ધોરણો અનુસાર ચાલતો નથી અને કદાચ પ્રોગ્રામ અધવચ્ચેથી અટકી પણ શકે છે.

અપવાદ એ પ્રોગ્રામના અમલ દરમિયાન ઉદ્‌ભવતી સમસ્યાનો સંકેત છે, જે મોટે ભાગે ભૂલસંદેશ દર્શાવે છે. જોકે, અપવાદ જવલ્લે જ ઉદ્‌ભવતા હોય છે, તેમ છતાં, પ્રોગ્રામ લખતી વખતે આવા અપવાદરૂપ પરિસ્થિતિઓનું નિયમન થઈ શકે તે માટે જરૂરી કાળજી લેવી જ જોઈએ. અપવાદનું નિયમન એ એવી આગોતરી વ્યવસ્થા છે કે જે, જાણે કે કોઈ સમસ્યા સર્જાઈ જ નથી તેમ માની પ્રોગ્રામનો અમલ ચાલુ જ રાખવા દે છે અથવા પ્રોગ્રામનો અનિયંત્રિત રીતે અણધાર્યો અંત લાવતા પહેલાં તે ઉપયોગકર્તાને તેની જાણ કરે છે.

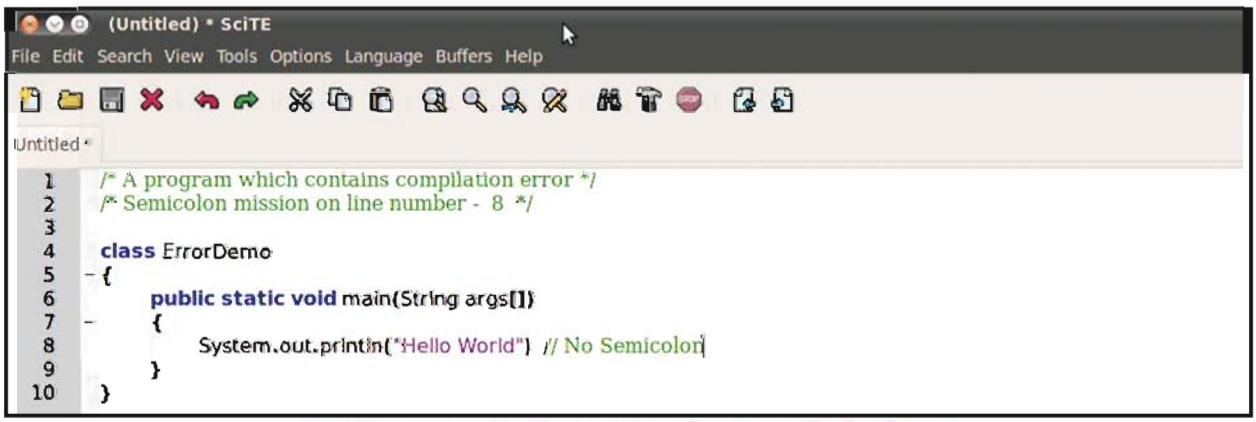
આ પ્રકરણમાં આપણે આપણા પ્રોગ્રામમાં ઉદ્‌ભવતા અપવાદરૂપ પરિસ્થિતિઓનું નિયમન કરવા માટેની યુક્તિઓ શીખીશું, તદ્‌ઉપરાંત જાવામાં ઉપલબ્ધ કેટલાક પ્રમાણભૂત અપવાદરૂપ પરિસ્થિતિઓ, તેમજ આપણા પ્રોગ્રામમાં અપવાદરૂપ પરિસ્થિતિ સર્જાય તેમ છતાં પ્રોગ્રામના ચોક્કસ ભાગનો હંમેશા અમલ થાય જ તે માટેની યુક્તિઓ પણ શીખીશું. અંતમાં, આપણે આપણા પોતાના અપવાદના પ્રકારોને વર્ણવીને તેનો ઉપયોગ કરવાની યુક્તિ વિશે જોઈશું.

## અપવાદના પ્રકારો

જાવામાં, તમામ પ્રકારની ક્ષતિવાળી પરિસ્થિતિને અપવાદ તરીકે ઓળખવામાં આવે છે. પ્રોગ્રામમાં ઉદ્‌ભવતી ભૂલોને મુખ્યત્વે "કંપાઈલ કરતી વખતે ઉદ્‌ભવતી ભૂલો" (compile-time errors) અને "અમલ દરમિયાન ઉદ્‌ભવતી ભૂલો" (run-time errors) એમ બે પ્રકારોમાં વિભાજિત કરી શકાય.

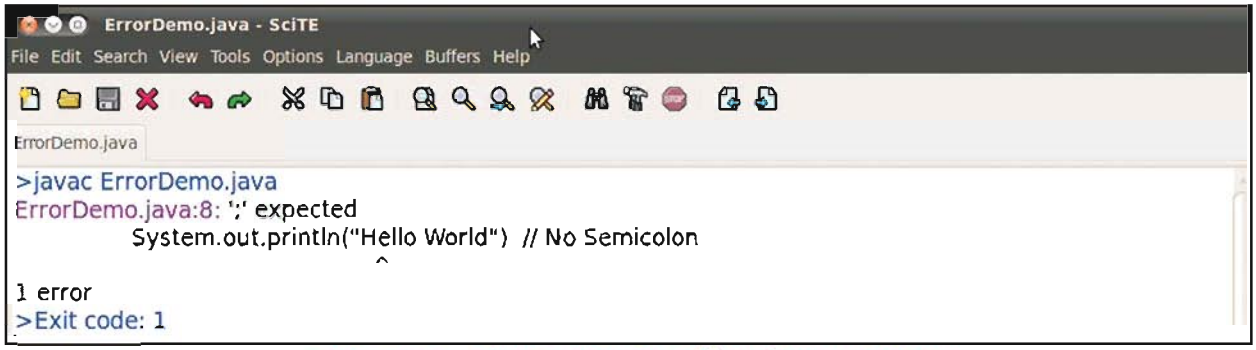
## કંપાઈલ કરતી વખતે ઉદ્‌ભવતી ભૂલો

અગાઉના પ્રકરણમાં કોઈ પણ જાવા પ્રોગ્રામને કેવી રીતે કંપાઈલ કરવો તે આપણે શીખી ગયા છીએ. કોઈ પણ પ્રોગ્રામિંગ ભાષાની મૂળ સૂચનાઓ (sourcecode)ને કમ્પ્યૂટર દ્વારા અમલમાં મૂકી શકાય તેવી સૂચનાઓ (objectcode)માં રૂપાંતરિત કરવા માટે કંપાઈલરનો ઉપયોગ કરવામાં આવે છે. જો પ્રોગ્રામમાં કોઈ જોડણીની ભૂલ (syntax error) હશે, તો આપણને કંપાઈલેશન કરતી વખતે જ ભૂલ દર્શાવવામાં આવશે અને તેને કારણે આપણે ".class" ફાઈલ બનાવી શકીશું નહીં. કેટલીક સામાન્ય જોડણીની ભૂલોમાં અલ્પવિરામ " , " ચિહ્ન ન હોવું, અવ્યાખ્યાયિત ચલનો ઉપયોગ, કોઈ ચાવીરૂપ શબ્દની ખોટી જોડણી અને ઊઘડતા કૌંસની સંખ્યાના પ્રમાણમાં બંધ થતા કૌંસની સંખ્યાનો વધારો કે ઘટાડો વગેરેનો સમાવેશ થાય છે. આકૃતિ 10.1માં દર્શાવેલ જાવાપ્રોગ્રામમાં અર્ધવિરામ ચિહ્નની કમી છે.




આકૃતિ 10.1 : કંપાઈલ કરતી વખતે ભૂલ દર્શાવતો પ્રોગ્રામ

ઉપરનો પ્રોગ્રામ જ્યારે કંપાઇલ કરવામાં આવશે ત્યારે કંપાઇલ કરતી વખતે ભૂલ દર્શાવાશે. આવું એટલા માટે થશે, કારણ કે આપણે સૂચનાની લીટી નંબર-8ના અંતે અર્ધવિરામ (semicolon) ';' મૂકવાનું ભૂલી ગયા છીએ. આકૃતિ 10.2માં દર્શાવ્યા મુજબ જાવા કંપાઇલર જ્યારે કંપાઇલેશનનું પરિણામ દર્શાવે છે, ત્યારે તે જ્યાં ભૂલ જણાઈ હોય તે લીટીના ક્રમ (line number) સહિત ભૂલનો પ્રકાર પણ સૂચવે છે.



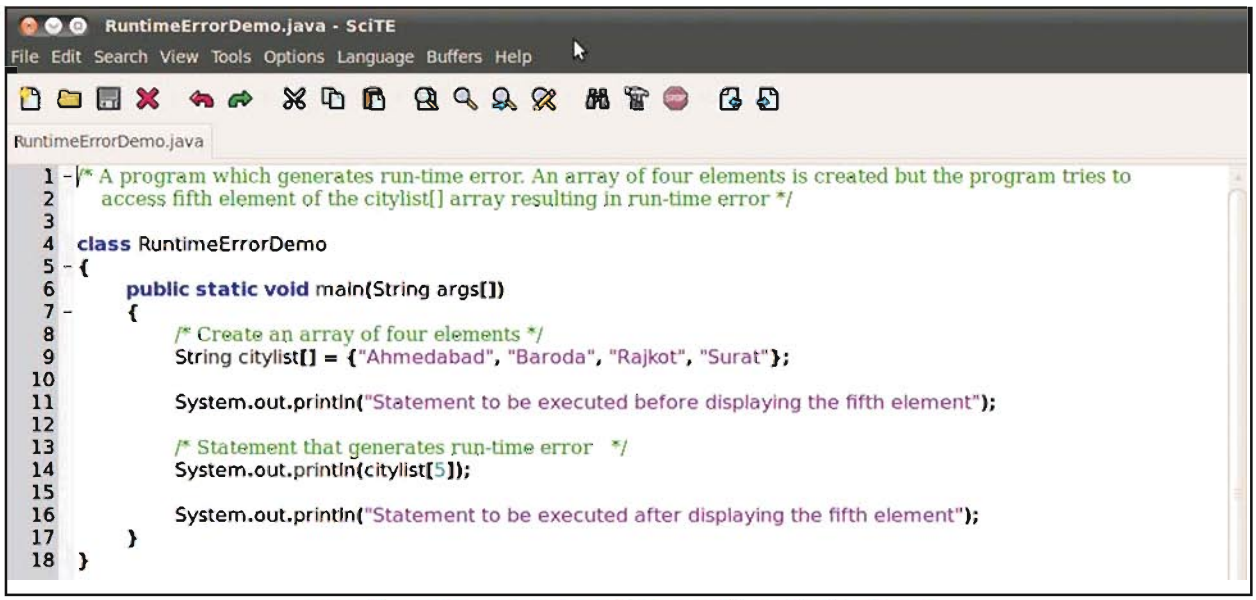

**આકૃતિ 10.2 : આકૃતિ 10.1માં દર્શાવેલ પ્રોગ્રામનું પરિણામ**

પ્રોગ્રામને કંપાઇલ કરતી વખતે ઉદ્ભવતી ક્ષતિઓ મોટે ભાગે પ્રોગ્રામરની ભૂલો હોય છે અને તેને જ્યાં સુધી સુધારવામાં ન આવે ત્યાં સુધી તે પ્રોગ્રામને કંપાઇલ થવા દેતી નથી.

**નોંધ :** કમ્પ્યૂટરવિજ્ઞાનના ક્ષેત્રે, કોઈ પણ આદેશ કે પ્રોગ્રામનો સફળતાપૂર્વક અમલ થયો છે કે નહીં તેનો નિર્દેશ "Exit code" કે "Exit status" કરે છે. સંજ્ઞા 0 (શૂન્ય), આદેશ સફળતાપૂર્વક અમલમાં મુકાયાનો નિર્દેશ કરે છે, જ્યારે સંજ્ઞા 1 એવું દર્શાવે છે કે, કમાન્ડનો અમલ કરતી વખતે કોઈક સમસ્યા ઉદ્ભવી છે. આકૃતિ 10.2માં છેલ્લી લીટી Exit code : 1નો નિર્દેશ કરે છે. તેનો અર્થ એ થાય કે, ErrorDemo.java નામના કંપાઇલેશન પ્રોગ્રામનું કાર્ય સફળ રહ્યું નથી.

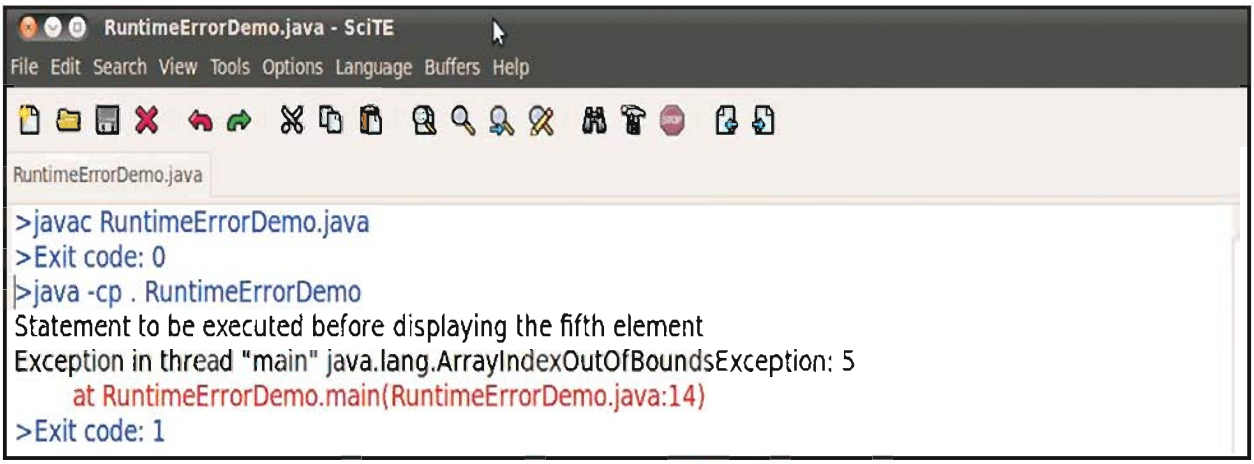
### પ્રોગ્રામના અમલ દરમિયાન ઉદ્ભવતી ભૂલો

જો પ્રોગ્રામની સૂચનાઓમાં જોડણીની કોઈ ભૂલ નહીં હોય, તો પ્રોગ્રામ સફળતાપૂર્વક કંપાઇલ થશે અને આપણને ".class" ફાઇલ મળશે. જો કે, આમ છતાં પ્રોગ્રામ આપણી અપેક્ષા પ્રમાણે જ ચાલશે તેવી કોઈ ખાતરી મળતી નથી. તો ચાલો, આકૃતિ 10.3માં દર્શાવેલ અન્ય ઉદાહરણ જોઈએ, જે કંપાઇલ થઈ જશે, પણ અમલ કરતી વખતે અસામાન્ય સંજોગોમાં અટકી જશે.




**આકૃતિ 10.3 : અમલ દરમિયાન ભૂલો દર્શાવતો પ્રોગ્રામ**

આકૃતિ 10.3માં દર્શાવેલ પ્રોગ્રામમાં જોડણીની કોઈ ભૂલ ન હોવાથી તે કંપાઇલ થઈ જશે. પ્રોગ્રામમાં "citylist[]" નામનો એરે બનાવ્યો છે, જે ચાર જુદાં-જુદાં શહેરોનાં નામ સાચવશે. 14મી સૂચનામાં આપણે citylist નામના એરેના એવા ઘટકની વિગતો દર્શાવવાનો પ્રયત્ન કરીએ છીએ, જે હયાત જ નથી. અહીં આ કિસ્સો અપવાદરૂપ પરિસ્થિતિ છે. પ્રોગ્રામના અમલ દરમિયાન જ અપવાદ ઉદ્ભવે છે. પ્રોગ્રામના અમલનું પરિણામ આકૃતિ 10.4માં દર્શાવાયેલ છે.




**આકૃતિ 10.4 : આકૃતિ 10.3માં દર્શાવેલ પ્રોગ્રામનું પરિણામ**

આકૃતિ 10.4માં દર્શાવેલ પરિણામમાંથી આપણે એ નોંધી શકીએ છીએ કે પ્રોગ્રામની સૂચના ક્રમ 14 પરથી પ્રોગ્રામનો અમલ અણધારી રીતે અંત પામ્યો. પરિણામમાં એક કારણદર્શક સંદેશ "ArrayIndexOutOfBoundsException" દર્શાવાયો છે, જે પ્રોગ્રામના અમલ દરમિયાન ઉદ્‌ભવેલ અપવાદનો નિર્દેશ કરે છે.

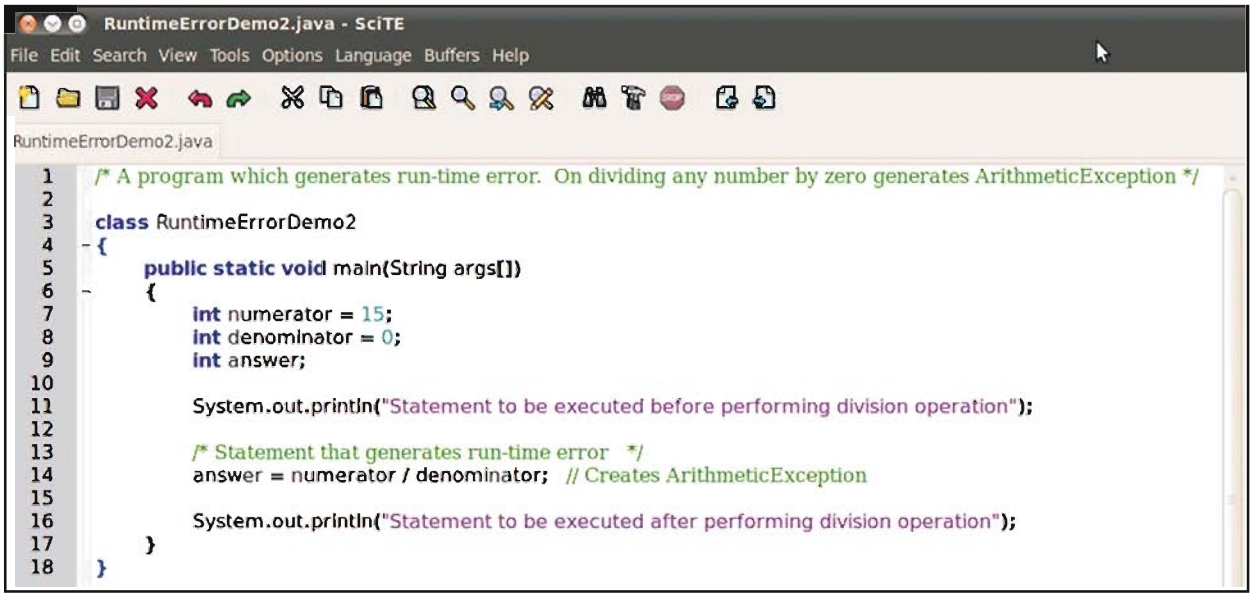
હવે આપણે આવા કેટલાક અન્ય કિસ્સા જોઈએ કે જે અપવાદ સર્જે છે. જાવામાં દરેક પ્રકારના અપવાદ માટે, સંબંધિત એક્સેપ્શન ક્લાસ (Exception class) ઉપલબ્ધ છે.

Java.lang અને java.io પૅકેજ વિવિધ અપવાદો સાથે કામ કરતા વર્ગોનો પદાનુક્રમ ધરાવે છે. બહોળા પ્રમાણમાં જોવા મળતા કેટલાક અપવાદોની યાદી કોષ્ટક 10.1માં આપવામાં આવેલ છે.

| એક્સેપ્શન ક્લાસ (Exception Class) | અપવાદ માટેનું કારણ (Condition resulting in Exception) | ઉદાહરણ (Example) |
|---|---|---|
| ArrayIndexOutOfBoundsException | ઍરેના એલિમેન્ટ તરીકે ઍરેની મર્યાદાની બહારની હોય તેવી કિંમતનો ઉપયોગ. | int a[] = new int[4];<br>a[13] = 99; |
| ArithmeticException | કોઈ પણ સંખ્યાને શૂન્ય વડે ભાગવાનો પ્રયત્ન. | int a = 50 / 0; |
| FileNotFoundException | અસ્તિત્વમાં ન હોય તેવી ફાઇલનો ઉપયોગ કરવાનો પ્રયત્ન. | |
| NullPointerException | જ્યાં કોઈ ઑબ્જેક્ટ આવશ્યક હોય તેવા કિસ્સામાં ખાલી કિંમત (null)નો ઉપયોગ કરવાનો પ્રયત્ન. | String s = null;<br>System.out.println(s.length()); |
| NumberFormatException | કોઈ શબ્દમાળા (string)ને સાંખ્યિક રૂપમાં ફેરવવા માટેનો પ્રયત્ન. | String s = "xyz";<br>int i = Integer.parseInt(s); |
| PrinterIOException | મુદ્રણ સમયે થતી ઉદ્‌ભવેલ ઇનપુટ/આઉટપુટ ભૂલ (I/O error) | |

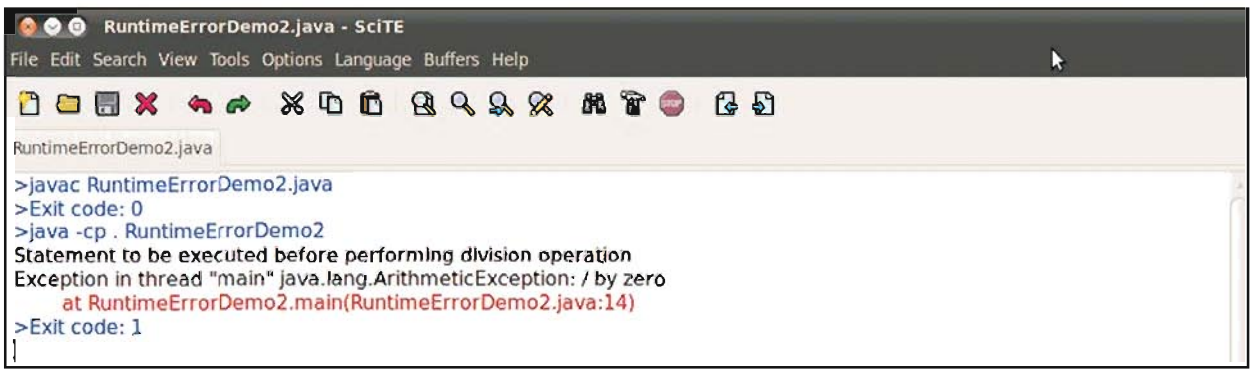
**કોષ્ટક 10.1 : બહોળા પ્રમાણમાં જોવા મળતા કેટલાક અપવાદોની યાદી**

આકૃતિ 10.5 અપવાદ સર્જતો એક વધુ પ્રોગ્રામ દર્શાવે છે.




**આકૃતિ 10.5 : ગાણિતિક અપવાદનું દૃષ્ટાંત આપતો પ્રોગ્રામ**

આકૃતિ 10.5માં દર્શાવેલ પ્રોગ્રામમાં આપણે એક પૂર્ણાંક સંખ્યાને શૂન્ય વડે ભાગવાનો પ્રયત્ન કરીએ છીએ. કોઈ પણ સંખ્યાને જ્યારે શૂન્ય વડે ભાગવામાં આવે છે, ત્યારે પરિણામ અનંતતામાં પરિણમે છે. અહીં પ્રોગ્રામ સફળતાપૂર્વક કંપાઇલ થશે, પરંતુ ગાણિતિક અપવાદને કારણે અનપેક્ષિત પરિણામ આપશે. પ્રોગ્રામના અમલનું પરિણામ આકૃતિ 10.6માં દર્શાવેલ છે.




**આકૃતિ 10.6 : આકૃતિ 10.5માં દર્શાવેલ પ્રોગ્રામનું પરિણામ**

આપણે શૂન્ય વડે ભાગાકાર કરતા હોવાથી, જ્યારે ભાગાકારની ક્રિયા કરતું વિધાન અમલમાં આવશે, ત્યારે JVM તરત જ પ્રોગ્રામનો અંત લાવશે.

### અપવાદરૂપ પરિસ્થિતિનું વ્યવસ્થાપન

અપવાદ એ ભૂલની સ્થિતિ છે. અપવાદરૂપ પરિસ્થિતિનું વ્યવસ્થાપન એ ક્ષતિઓનું વ્યવસ્થાપન કરવાની વસ્તુલક્ષી (object-oriented) યુક્તિ છે. અપવાદરૂપ પરિસ્થિતિઓનું વ્યવસ્થાપન કરતી વખતે એ બાબતની ખાતરી રાખવાનો પ્રયત્ન કરવામાં આવે છે કે, અણધારી રીતે પ્રોગ્રામનો અંત ન આવી જાય કે તે અનપેક્ષિત પરિણામ ન આપે. આ વિભાગમાં આપણે અપવાદરૂપ પરિસ્થિતિઓનું વ્યવસ્થાપન કેવી રીતે કરવું તે શીખીશું.

અપવાદરૂપ પરિસ્થિતિના વ્યવસ્થાપન માટેનો પ્રોગ્રામ (Exception handler) લખવા માટે જાવા try, catch અને finally જેવા કી વર્ડનો ઉપયોગ કરે છે. try, catch અને finally જેવા ચાવીરૂપ શબ્દો અપવાદ ઉપસ્થિત થાય ત્યારે ઉપયોગમાં લેવાય છે. આ કી વર્ડ વિધાનોના સમૂહને રજૂ કરે છે.

- tryના વિભાગ (Block)માં એવી સૂચનાઓનો સમાવેશ થાય છે કે જે, એક કે વધુ અપવાદોનો ઉદ્ભવ થવા દે છે.

- catch વિભાગમાં એવી સૂચનાઓ હોય છે કે જે, સંબંધિત try વિભાગમાં સર્જાય તેવા ચોક્કસ પ્રકારના અપવાદોની સંભાળ લેવા માટે લખવામાં આવી હોય છે.
- finally વિભાગનો અમલ હંમેશાં પ્રોગ્રામનો અંત આવે તે પહેલાં કરવામાં આવે છે. પછી ભલે, try વિભાગમાં કોઈ અપવાદો સર્જાયા હોય કે ન સર્જાયા હોય.

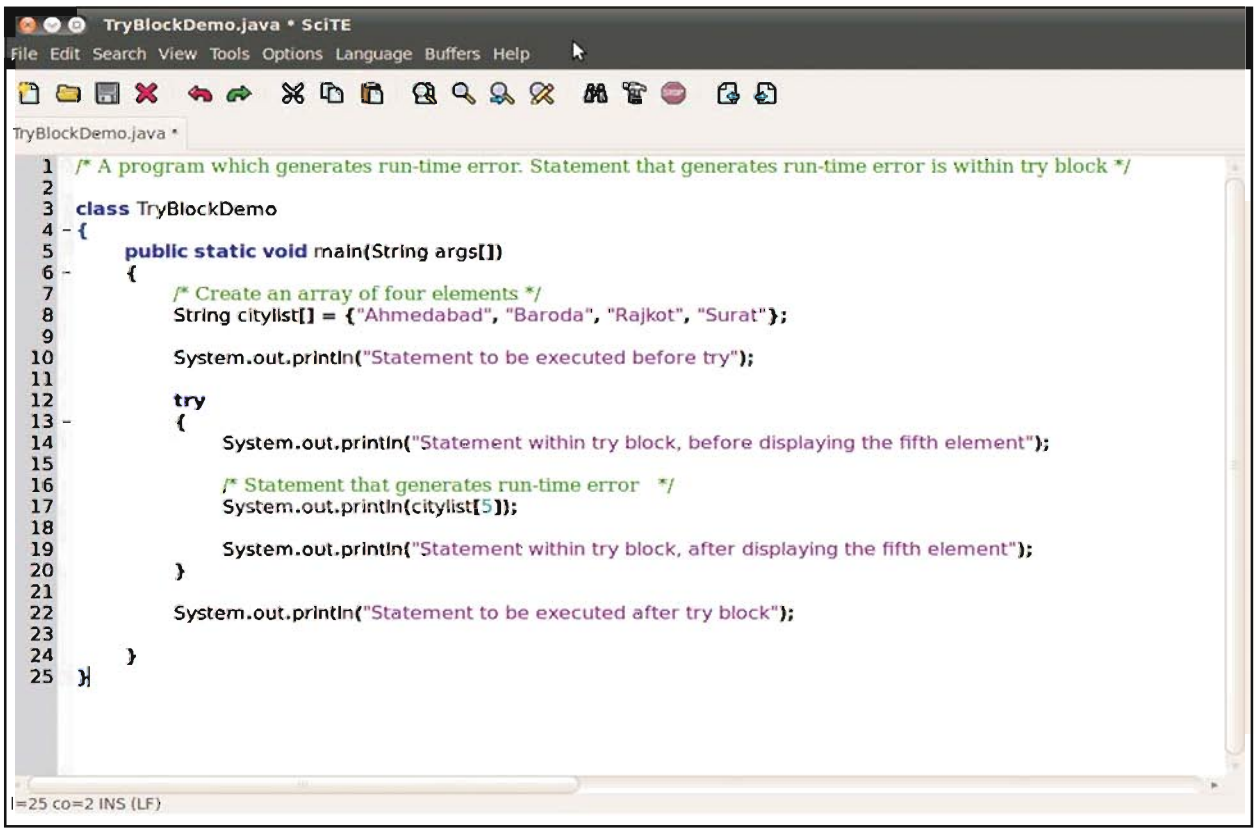
હવે આ ત્રણેય વિભાગોને એક પછી એક વિગતવાર સમજીએ.

## Try વિભાગ

Try વિધાન કૌંસમાં સમાવિષ્ટ વિધાનોનો વિભાગ છે. આ વિધાનો આપણે જે અપવાદની દેખરેખ રાખવા ઇચ્છીએ છીએ, તે માટેની જરૂરી સૂચનાઓ છે. જો તેના અમલ દરમિયાન કોઈ સમસ્યા ઉદ્‌ભવે, તો એક અપવાદ ઊભો કરવામાં આવશે. જાવામાં દરેક પ્રકારનો અપવાદ એક ઑબ્જેક્ટને અનુરૂપ હોય છે. try વિભાગ એક કે તેથી વધુ અપવાદો ઊભા કરી શકે છે. try વિભાગની વાક્યરચના નીચે દર્શાવ્યા મુજબ છે :

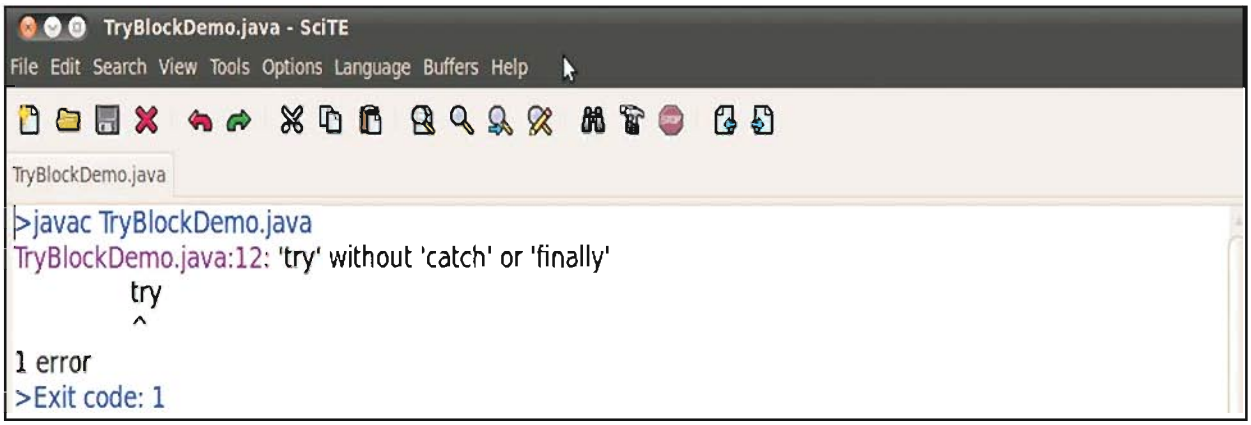
```
try
{
   // set of statements that may generate one or more exceptions.
}
```

હવે try વિભાગની અંદર લખવામાં આવતાં વિધાનો જોઈએ. તે એક અપવાદ સર્જે છે કે જે આપણે અગાઉ જોઈ ગયા છીએ. આકૃતિ 10.7માં દર્શાવેલ સૂચનાઓનો જ્યારે અમલ કરવામાં આવશે, ત્યારે તેમાં અપવાદના વ્યવસ્થાપન (Exception handler)ની વ્યવસ્થા ન હોવાને કારણે પ્રોગ્રામનો અંત લાવશે. પ્રોગ્રામના અમલ દરમિયાન ભૂલો સર્જાવાની શક્યતા ધરાવતી સૂચનાઓના ભાગને try વિભાગમાં જ લખવો જોઈએ.




આકૃતિ 10.7 : try વિભાગ સમજાવતો પ્રોગ્રામ

આકૃતિ 10.7માં દર્શાવેલ પ્રોગ્રામને કંપાઇલ કરવાનો પ્રયત્ન કરવામાં આવશે તો તે કંપાઇલેશન વખતે ભૂલ દર્શાવશે, કારણ કે, try બ્લોક પછી catch વિભાગ અથવા finally વિભાગ હોવો જરૂરી છે. કંપાઇલેશન દરમિયાન સર્જાતી ભૂલ આકૃતિ 10.8માં દર્શાવેલ છે.



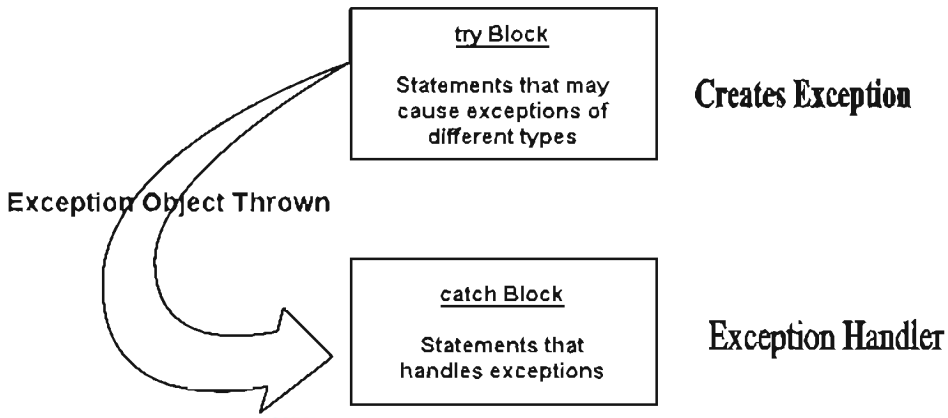

આકૃતિ 10.8 : આકૃતિ 10.7માં દર્શાવેલ પ્રોગ્રામની કંપાઈલેશન દરમિયાન સર્જાયેલ ભૂલ

## Catch વિભાગ

Catch વિભાગ હંમેશા try વિભાગની તરત પાછળ હોવો જોઈએ. વ્યવસ્થાપન કરવા માટે જરૂરી સૂચનાઓ આ વિભાગમાં આવે છે. Catch વિભાગ અપવાદનું વ્યવસ્થાપન કરનારો વિભાગ હોવાથી જાવામાં તેને "એક્સેપ્શન હેન્ડલર" (Exception Handler) કહે છે. કોઈ એક try વિભાગ માટે એક કે તેથી વધુ catch વિભાગ હોઈ શકે છે. Catch વિભાગની વાક્યરચના નીચે દર્શાવ્યા મુજબ છે :

```
try
{
      // Set of statements that may generate one or more exceptions
 }
catch(Exception_Type Exception_object)
{
       // Code to handle the exception
 }
```

Catch વિભાગમાં કી વર્ડ તરીકે catch લખી તેની પાછળ એક પેરામીટર આપવામાં આવે છે. અપવાદના વ્યવસ્થાપન માટેની સૂચનાઓ કૌંસની વચ્ચે લખવાની હોય છે. આ વિભાગે કયા પ્રકારના અપવાદ સાથે કામ પાર પાડવાનું છે તે પેરામીટર દ્વારા દર્શાવવામાં આવે છે. જાવા વિવિધ પ્રકારના અપવાદો માટે સહાય આપે છે. આ મુદ્દાની ચર્ચાના પાછળના ભાગે આપણે અનેક catch વિભાગનો ઉપયોગ કરી કેવી રીતે વિવિધ પ્રકારના અપવાદોને પાર પાડી શકાય તે જોઈશું. આકૃતિ 10.9 try-catch વિભાગની રચના સમજાવે છે.




આકૃતિ 10.9 : અપવાદ-વ્યવસ્થાપન રચના

હવે યોગ્ય અપવાદ-વ્યવસ્થાપન ગોઠવતા પ્રોગ્રામનું ઉદાહરણ જોઈએ. આ સમજવા માટે હવે આપણે catch વિભાગમાં ઑબ્જેક્ટને ઝીલીને ArrayIndexOutOfBounds અપવાદનું વ્યવસ્થાપન કરીશું. પ્રકરણના પાછળના ભાગે આપણે પ્રોગ્રામનો અંત લાવ્યા વિના પ્રોગ્રામનો અમલ ચાલુ રાખવા માટેની સૂચનાઓને કેવી રીતે લખવી તે જોઈશું.

TryCatchDemo.java - SciTE

File Edit Search View Tools Options Language Buffers Help

TryCatchDemo.java

```
1  /* A program which generates run-time error. Statement that generates run-time error is within
2     try block, there is a corresponding catch block immediately below the try block */
3
4  class TryCatchDemo
5  {
6      public static void main(String args[])
7      {
8          /* Create an array of four elements */
9          String citylist[] = {"Ahmedabad", "Baroda", "Rajkot", "Surat"};
10         System.out.println("Statement to be executed before try");
11         try
12         {
13             System.out.println("Statement within try block, before displaying the fifth element");
14
15             /* Statement that generates run-time error  */
16             System.out.println(citylist[5]);
17
18             System.out.println("Statement within try block, after displaying the fifth element");
19         }
20         catch(ArrayIndexOutOfBoundsException eobj)
21         {
22             System.out.println("Within Catch Block");
23             System.out.println("Caught Exception object of type : " + eobj);
24         }
25         System.out.println("Statement to be executed after try...catch");
26
27     }
28 }
```

l=28 co=2 INS (LF)

**આકૃતિ 10.10 : try...catch વિભાગનો ઉપયોગ સમજાવતો પ્રોગ્રામ**

આકૃતિ 10.10માં દર્શાવેલ સૂચનાઓ સફળતાપૂર્વક કંપાઇલ થશે અને તેનો અમલ પણ થશે. આકૃતિ 10.10માં દર્શાવેલ પ્રોગ્રામમાં સૂચનાક્રમ 16 પરના વિધાનથી અપવાદ સર્જાશે. કારણકે, 16મી લીટી પર આપણે citylist[ ] ઍરેના પાંચમા ઘટકનો ઉપયોગ કરવાનો પ્રયત્ન કરીએ છીએ, જોકે ખરેખર ઍરેમાં તો માત્ર ચાર ઘટક જ છે. આપણો પ્રોગ્રામ એવી ઘટકકિંમત આપીને ઍરે ઘટકનો ઉપયોગ કરવા પ્રયત્ન કરે છે કે જે ઍરેની મર્યાદાની બહાર છે, જે સ્વાભાવિક રીતે જ પ્રોગ્રામને એક અપવાદ તરફ દોરી જાય છે. જ્યારે અપવાદ ઉદ્ભવે છે, ત્યારે ArrayIndexOutOfBoundsException પ્રકારનો ઑબ્જેક્ટ બનાવીને છોડવામાં આવશે. એ પછી તેને સંબંધિત catch વિભાગ આ અપવાદનું વ્યવસ્થાપન કરે છે અને અનપેક્ષિત રીતે પ્રોગ્રામનો અંત આવવા દેતો નથી. catch વિભાગ "eobj" ઑબ્જેક્ટનો નિર્દેશ ધરાવે છે, જેને try વિભાગ દ્વારા સર્જન કરીને મોકલવામાં આવેલ છે. આકૃતિ 10.11 પ્રોગ્રામનું પરિણામ દર્શાવે છે.

TryCatchDemo.java - SciTE

File Edit Search View Tools Options Language Buffers Help

TryCatchDemo.java

```
>javac TryCatchDemo.java
>Exit code: 0
>java -cp . TryCatchDemo
Statement to be executed before try
Statement within try block, before displaying the fifth element
Within Catch Block
Caught Exception object of type : java.lang.ArrayIndexOutOfBoundsException: 5
Statement to be executed after try...catch
>Exit code: 0
```

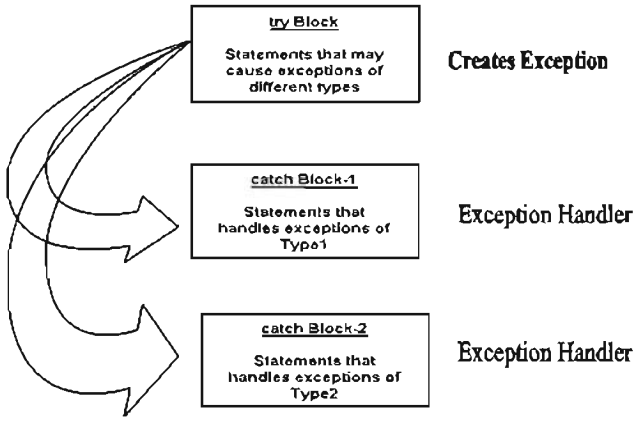
**આકૃતિ 10.11 : આકૃતિ 10.10માં દર્શાવેલ પ્રોગ્રામનું પરિણામ**

આકૃતિ 10.11 પરથી આપણે અવલોકન કરી શકીએ છીએ કે અપવાદની ઉપસ્થિતિમાં પ્રોગ્રામનો અંત આવ્યો નહીં. "after try...catch" લખાણ ધરાવતા વિધાનનો અમલ કરી દર્શાવવામાં આવશે.

### એક કરતાં વધુ catch વિભાગ

એક જ પ્રોગ્રામમાં અનેક અપવાદરૂપ પરિસ્થિતિ સર્જાઈ શકે છે. ઉદાહરણ તરીકે, ધારો કે આપણે એક ચોક્કસ ફાઇલને દૂરસ્થિત કમ્પ્યૂટર પર અપલોડ કરવી છે, તો તે બે સ્પષ્ટ અપવાદો તરફ દોરી જશે. એક, જો ફાઇલ આપણા કમ્પ્યૂટર પર ઉપલબ્ધ નહીં હોય, બીજો જો આપણું કમ્પ્યૂટર ઇન્ટરનેટ સાથે જોડાણ ધરાવતું ન હોય. એક કરતાં વધુ અપવાદોને સંભાળવા માટે જાવામાં જોગવાઈ છે. અગાઉ ચર્ચા કર્યા મુજબ, અપવાદ સર્જાવાની શક્યતા ધરાવતી સૂચનાઓ try વિભાગમાં જ લખાવી જોઈએ. એ સિવાય દરેક પ્રકારના અપવાદને અલગ રીતે સંભાળવા માટે એક કરતાં વધુ catch વિભાગ હોઈ શકે.

try વિભાગ જો વિવિધ પ્રકારના અનેક અપવાદો થ્રો (Throw) કરે, તો આપણે જુદા-જુદા અપવાદોનું વ્યવસ્થાપન કરી શકે તેવા અનેક catch વિભાગ લખી શકીએ. આ વ્યવસ્થા પ્રોગ્રામરને દરેક પ્રકારના અપવાદ માટે અલગ તર્ક (logic) લખવા મદદરૂપ થાય છે. આકૃતિ 10.12 અનેક catch વિભાગનો ઉપયોગ દર્શાવે છે.




**10.12 : અનેક catch વિભાગ સાથે અપવાદ-વ્યવસ્થાપનની રચના**

આકૃતિ 10.13 એવો પ્રોગ્રામ દર્શાવે છે, જેમાં એક કરતાં વધુ catch વિભાગનો ઉપયોગ કરવામાં આવેલ છે. અહીં try વિભાગની અંદર બે જુદા જ પ્રકારના અપવાદ ઊભા થયા. ArrayIndexOutOfBoundsException પ્રકારનો અપવાદ પ્રથમ catch વિભાગ દ્વારા પકડી લેવામાં આવશે અને ArithmaticExpression પ્રકારનો અપવાદ બીજા catch વિભાગ દ્વારા પકડી પાડવામાં આવશે.

MultipleCatchDemo.java - SciTE

File Edit Search View Tools Options Language Buffers Help

MultipleCatchDemo.java

```
1  /* A program which generates multiple run-time error. There are multiple catch blocks to handle various
2     particular Exceptions */
3
4  class MultipleCatchDemo
5  {
6      public static void main(String args[])
7      {
8          String citylist[] = {"Ahmedabad", "Baroda", "Rajkot", "Surat"};
9          int numerator = 15, denominator = 0, answer;
10         System.out.println("Statement to be executed before try block");
11         try
12         {
13             System.out.println("Begining of try block...");
14             System.out.println(citylist[5]); // Generates ArrayIndexOutOfBoundsException
15             answer = numerator / denominator ; // Generates ArithmeticException
16             System.out.println("End of try block...");
17         }
18         catch(ArrayIndexOutOfBoundsException eobj) {
19             System.out.println("Within first catch block, exception caught : " + eobj);
20         }
21         catch(ArithmeticException eobj) {
22             System.out.println("Within second catch block, exception caught : " + eobj);
23         }
24         catch(Exception eobj) {
25             System.out.println("Within last catch block, exception caught : " + eobj); //Generic block
26         }
27         System.out.println("End of Program...");
28     }
29 }
```

**આકૃતિ 10.13 : એકથી વધુ catch વિભાગનો ઉપયોગ સમજાવતો પ્રોગ્રામ**

છેલ્લો catch વિભાગ કોઈ પણ પ્રકારના અપવાદનું વ્યવસ્થાપન કરી શકે છે. તે એક ડીફોલ્ટ પ્રકારનો catch વિભાગ છે અને જ્યારે અનેક catch વિભાગ હોય, ત્યારે આ વિભાગને છેલ્લા વિભાગ તરીકે રાખવો પડે. પ્રોગ્રામ લખતી વખતે કોઈ ચોક્કસ catch બ્લોકનો ક્રમ કયો રાખ્યો છે તે અસર કરતો નથી પરંતુ ડિફોલ્ટ વિભાગ તો બધા catch વિભાગ પછી છેલ્લે રાખવો પડે છે. ઉદાહરણ તરીકે, આકૃતિ 10.13માં દર્શાવેલ પ્રોગ્રામમાં આપણે 18મી લીટી પરથી શરૂ થતા catch વિભાગને 21મી લીટી પરથી શરૂ થતા catch વિભાગ વડે તેમના ક્રમમાં અદલબદલ કરી શકીએ. જોકે, 24મી લીટી પરથી શરૂ થતા catch વિભાગને છેલ્લે જ રાખવો પડે.

એક કરતાં વધુ try વિભાગને એકની અંદર બીજો, બીજાની અંદર ત્રીજો એમ નેસ્ટિંગ કરીને લખી શકાય, પરંતુ દરેક try વિભાગના સંબંધિત catch વિભાગ લખવાની કાળજી લેવી જરૂરી છે.

### Finally વિભાગ

સામાન્ય રીતે, Finally વિભાગ try વિભાગનો અમલ કર્યા પછી અંતે clean up કરવા માટે વપરાય છે. સંબંધિત try વિભાગની અંદરથી કયા અપવાદો થ્રો કરવામાં (thrown) આવશે તેની પરવા કર્યા વગર જ્યારે આપણે એ ખાત્રી કરવા ઇચ્છીએ છીએ કે, કોઈ ચોક્કસ સૂચનાઓનો અમલ કરવાનો છે, ત્યારે આપણે finally વિભાગનો ઉપયોગ કરીએ છીએ. સંબંધિત try વિભાગ દ્વારા અપવાદ થ્રો કરવામાં આવ્યા છે કે નહીં તેની પરવા કર્યા વિના finally વિભાગનો અમલ ચોક્કસપણે કરવામાં આવે જ છે. પ્રોગ્રામની પૂર્ણતાના સમયે જો ફાઇલ બંધ કરવી જરૂરી હોય અથવા કોઈ અગત્યનું સંસાધન મુક્ત કરવાનું હોય ત્યારે finally વિભાગ બહોળા પ્રમાણમાં ઉપયોગમાં લેવાય છે. finally વિભાગની વાક્યરચના નીચે મુજબ છે :

```
finally
 {
    // clean-up code to be executed last
 // statements within this block always get executed even though if run-time errors
       terminate the program abruptly
 }
```

દરેક try વિભાગની પાછળ ઓછામાં ઓછો એક વિભાગ તો હોવો જ જોઈએ, જે catch વિભાગ હોય અથવા finally વિભાગ હોય. આકૃતિ 10.14 finally વિભાગનો ઉપયોગ કરવાનું ઉદાહરણ દર્શાવે છે.

FinallyDemo.java * SciTE

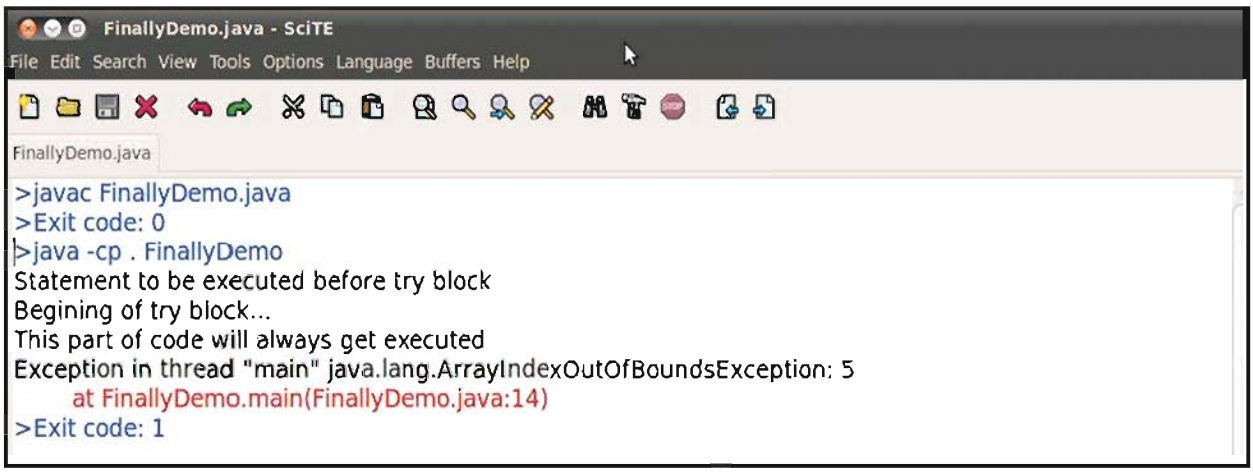
File Edit Search View Tools Options Language Buffers Help

FinallyDemo.java *

```
1  /* A program which generates multiple run-time error. There is a finally block following try block.
2     There are no catch blocks in this program */
3
4  class FinallyDemo
5  {
6      public static void main(String args[])
7      {
8          String citylist[] = {"Ahmedabad", "Baroda", "Rajkot", "Surat"};
9          int numerator = 15, denominator = 0, answer;
10         System.out.println("Statement to be executed before try block");
11         try
12         {
13             System.out.println("Begining of try block...");
14             System.out.println(citylist[5]);  // Generates ArrayIndexOutOfBoundsException
15             answer = numerator / denominator ; // Generates ArithmeticException
16             System.out.println("End of try block...");
17         }
18         finally
19         {
20             System.out.println("This part of code will always get executed");
21         }
22
23         System.out.println("End of Program...");
24     }
25 }
```

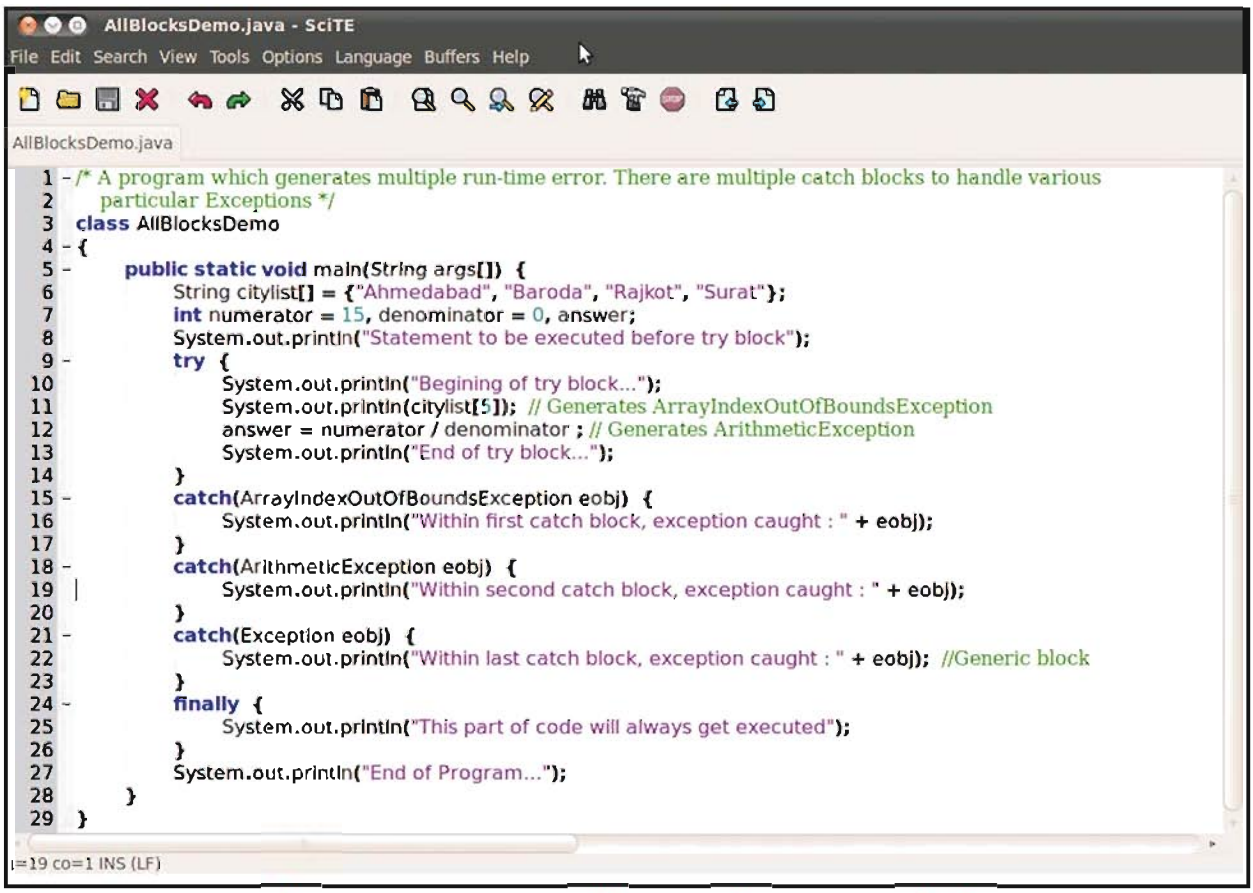
**આકૃતિ 10.14 : catch વિભાગ સિવાય finally વિભાગ સમજાવતો પ્રોગ્રામ**

આકૃતિ 10.14માં દર્શાવેલ પ્રોગ્રામમાં એક પણ catch વિભાગ ન હોવાને લીધે, 14મી લીટી પર ઉદ્ભવતા અપવાદને કારણે અધવચ્ચેથી પ્રોગ્રામનો અંત આવી જાય છે; 15 અને 16મી લીટી પર આવેલાં વિધાનોનો અમલ થશે નહીં. જોકે, finally વિભાગની ઉપસ્થિતિને લીધે, પ્રોગ્રામનો અંત આવે તે પહેલાં પ્રોગ્રામ finally વિભાગમાં પડેલાં વિધાનોનો અમલ કરે છે. પ્રોગ્રામનું પરિણામ આકૃતિ 10.15માં દર્શાવેલ છે.

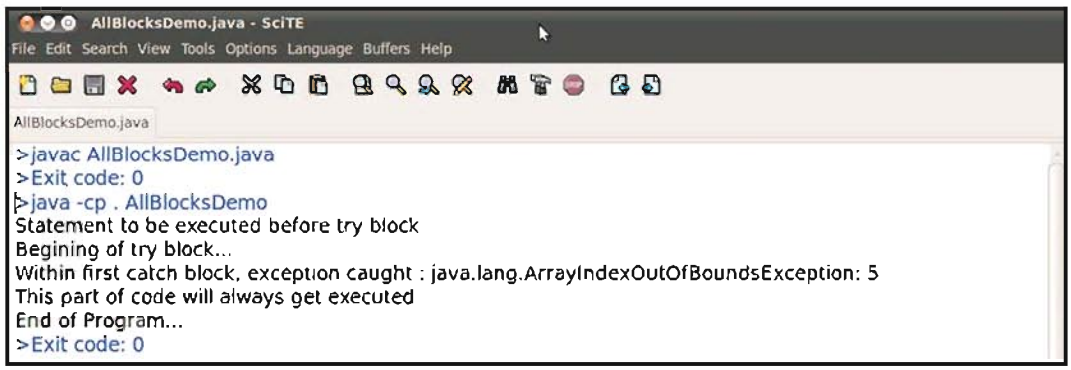



**આકૃતિ 10.15 : આકૃતિ 10.14માં દર્શાવેલ પ્રોગ્રામનું પરિણામ**

હવે ઘણા બધા catch વિભાગ અને એક finally વિભાગ એમ બધાનો એકસાથે ઉપયોગ કરાયો હોય, તેવું એક ઉદાહરણ જોઈએ. આકૃતિ 10.16માં દર્શાવેલ પ્રોગ્રામ આ બધા વિભાગ ધરાવે છે. એ રીતે, try વિભાગમાંથી ઉદ્ભવતા દરેક અપવાદ માટેના સંબંધિત અનેક catch વિભાગ સાથેનો આ એક સંપૂર્ણ પ્રોગ્રામ છે. પ્રોગ્રામનું પરિણામ આકૃતિ 10.17માં દર્શાવેલ છે.




**આકૃતિ 10.16 : try, catch અને finally વિભાગ સમજાવતો પ્રોગ્રામ**




**આકૃતિ 10.17 : આકૃતિ 10.16માં દર્શાવેલ પ્રોગ્રામનું પરિણામ**

પરિણામ પરથી એ સ્પષ્ટ થાય છે કે, 11મી લીટીથી પ્રોગ્રામના અમલનો અંકુશ પ્રથમ catch વિભાગ પર બદલાય છે, પછીથી તે finally વિભાગ પર બદલાય છે. છેલ્લા બે catch વિભાગનો અમલ થયો નહીં. આખો પ્રોગ્રામ ચાલ્યો નથી તેમ છતાં તેનો યોગ્ય રીતે અંત આવ્યો છે.

Finally વિભાગ કોઈ એક ચોક્કસ try વિભાગ સાથે જોડાયેલ હોય છે અને તે સંબંધિત try વિભાગ માટેના કોઈ પણ catch વિભાગ પછી તરત જ આવેલો હોવો જોઈએ. જો પ્રોગ્રામમાં કદાચ catch વિભાગ ન હોય તો finally વિભાગ try વિભાગની તરત પછી મૂકી શકાય. જો finally અથવા catch વિભાગ સચોટ જગ્યાએ મૂકવામાં નહીં આવ્યા હોય, તો પ્રોગ્રામ કંપાઈલ થઈ શકશે નહીં.

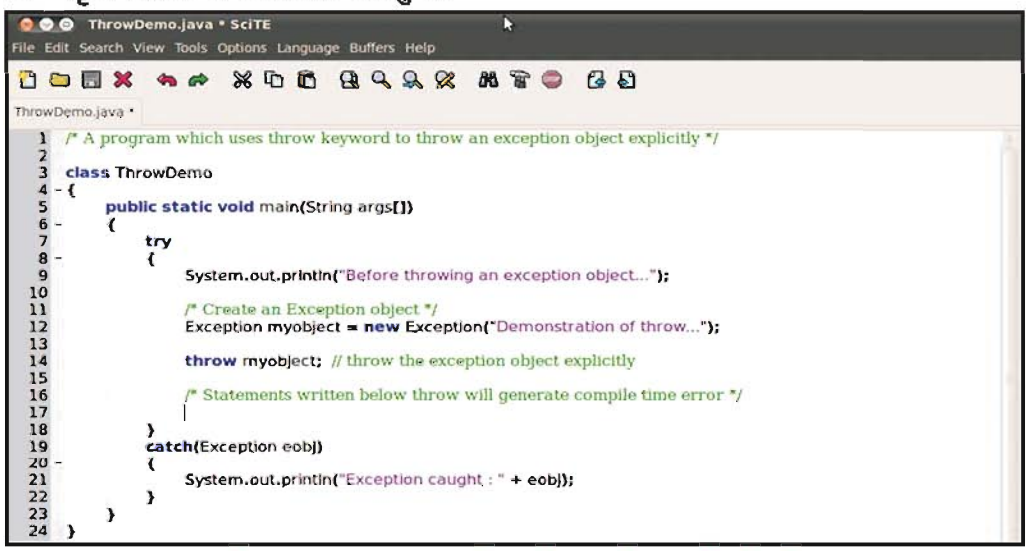
## Throw વિધાન

અપવાદના ઑબ્જેક્ટને નિશ્ચિત રૂપે થ્રો કરવા (throw) કી વર્ડનો ઉપયોગ કરવામાં આવે છે. અત્યાર સુધી આપણે જોયેલા ઉદાહરણરૂપ પ્રોગ્રામોમાં JVM અપવાદરૂપ ઑબ્જેક્ટ તૈયાર કરીને આપોઆપ થ્રો કરતું હતું. ઉદાહરણ તરીકે, જ્યારે આપણે શૂન્ય વડે ભાગાકાર કરવાની ક્રિયા કરવાનો પ્રયત્ન કર્યો, ત્યારે ArithmeticExceptionનો ઑબ્જેક્ટ બનાવવામાં આવ્યો અને JVM દ્વારા આપોઆપ તેને થ્રો કરવામાં આવ્યો.

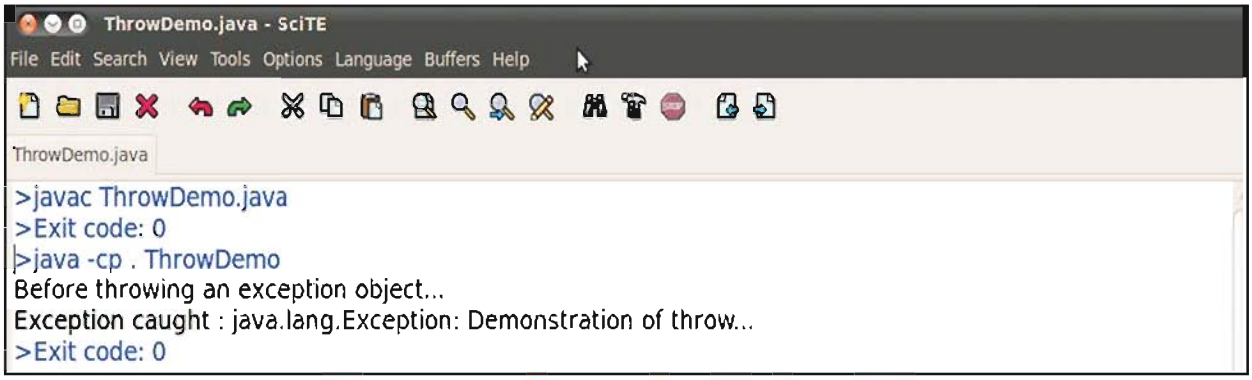
અપવાદનો ઑબ્જેક્ટ બનાવીને નિશ્ચિત રૂપે થ્રો કરવા જાવા વ્યવસ્થા પૂરી પાડે છે. આપણે જે ઑબ્જેક્ટને થ્રો કરીએ, તે java.lang.Throwable (Throwable ક્લાસ અથવા તેના કોઈ પણ સબ-ક્લાસનો ઑબ્જેક્ટ) પ્રકારનો હોવો જ જોઈએ, અન્યથા કંપાઈલ કરતી વખતે ભૂલ ઉદ્ભવશે. અપવાદ ઑબ્જેક્ટને થ્રો કરવા માટેની વાક્યરચના નીચે મુજબ છે :

```
throw exception_object;
```

જ્યારે throw વિધાન મળશે ત્યારે, સંબંધિત catch વિભાગની શોધ શરૂ થઈ જશે. try કે catch વિભાગ પછીના કોઈ પણ વિધાનનો અમલ કરવામાં આવશે નહીં. આકૃતિ 10.18માં લખેલ સૂચનાઓ throw વિધાનનો ઉપયોગ દર્શાવે છે અને એનું પરિણામ આકૃતિ 10.19માં આપવામાં આવ્યું છે.




**આકૃતિ 10.18 : throw વિધાનનો ઉપયોગ દર્શાવતો પ્રોગ્રામ**




**આકૃતિ 10.19 : આકૃતિ 10.18માં દર્શાવેલ પ્રોગ્રામનું પરિણામ**

આકૃતિ 10.18માં દર્શાવેલ પ્રોગ્રામમાં આપણે Exception ક્લાસનો eobj નામનો એક ઑબ્જેક્ટ બનાવ્યો. એ જ ઑબ્જેક્ટને throw વિધાનનો ઉપયોગ કરીને આપણે થ્રો કર્યો. નિશ્ચિતપણે થ્રો કરવામાં આવતા અપવાદ-ઑબ્જેક્ટને સંભાળવા માટે catch વિભાગ હોવો જ જોઈએ.

### throws ઉપવાક્ય

આપણે try-catch-finally વિભાગને જોયા. અત્યાર સુધી આપણે ચર્ચેલા પ્રોગ્રામ એ સાદા પ્રોગ્રામ હતા, જેમાં પદ્ધતિ (method)ના ઉપયોગનો સમાવેશ કરવામાં આવ્યો ન હતો. અહીં એવો વિચાર આવે કે, જો method કે constructor દ્વારા અપવાદ ઉદ્ભવે તો શું થાય ? એવા સંજોગોમાં try-catch વિભાગને આપણે ક્યાં મૂકીશું ? method દ્વારા ઉદ્ભવતા અપવાદનું વ્યવસ્થાપન કરવા માટે બે વૈકલ્પિક રસ્તા છે :

- method કે constructorમાં જ try-catch વિભાગ લખો કે જે કદાચ અપવાદનું ઉદ્ભવકેન્દ્ર છે.
- try વિભાગની અંદર જ method કે constructorને ઇન્વોક કરીને (કે જે કદાચ અપવાદ સર્જી શકે.)

constructor કે methodની અંદરની સૂચના અપવાદ થ્રો કરી શકે તેમ છે, તે જણાવવા method કે constructorને વ્યાખ્યાયિત કરતી વખતે throws ઉપવાક્યનો ઉપયોગ કરી શકાય. તે એવો પણ સંકેત કરે છે કે, methodમાં અપવાદનું વ્યવસ્થાપન કરી શકે તેવો catch વિભાગ નથી. જ્યારે આપણે constructor કે method લખીએ છીએ કે જે તેને બોલાવનાર તરફ અપવાદ મોકલી શકે, તો તે હકીકતનું દસ્તાવેજીકરણ કરવું ઉપયોગી છે. throws ચાવીરૂપ શબ્દ methodની ઘોષણા કરતી વખતે જ ઉપયોગ કરાય છે.

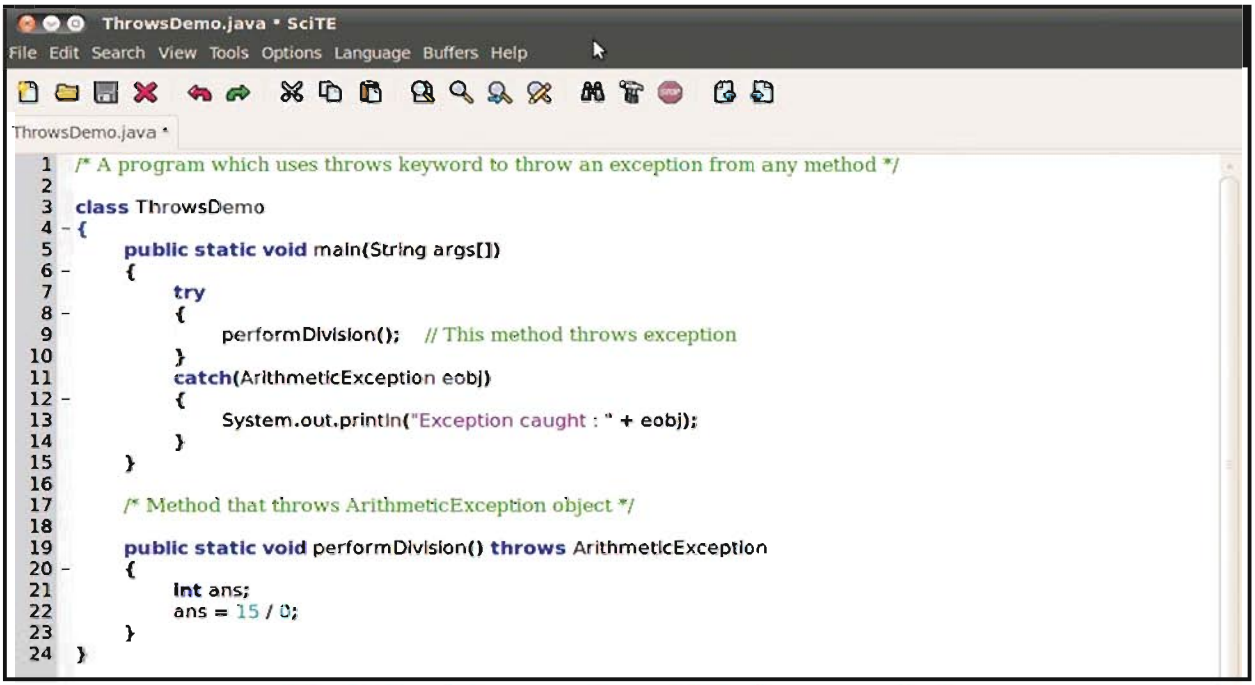
methodની ઘોષણા કરતી વખતે throws ઉપવાક્યનો નીચે મુજબ ઉપયોગ કરી શકાય છે :

```
method_Modifiers return_type method_Name(parameters) throws Exception list... {
.....
// body of the method
.....
}
```

એક method અનેક અપવાદો થ્રો કરી શકે છે. method દ્વારા થ્રો કરી શકાતા દરેક પ્રકારના અપવાદને methodના headerમાં દર્શાવવો જરૂરી છે. ઉદાહરણ તરીકે methodનું header નીચે મુજબ હોઈ શકે.

```
performDivision() throws ArithmeticException, ArrayIndexOutOfBoundsException
{
.....
// body of the method
.....
}
```

આકૃતિ 10.20માં દર્શાવેલ પ્રોગ્રામ ઉપયોગકર્તા-નિર્મિત methodની ઉપસ્થિતિમાં throws ઉપવાક્યના ઉપયોગનું નિદર્શન કરે છે. એ પછીની સૂચનાઓમાં એ નોંધવું ખાસ જરૂરી છે કે, જો method અપવાદ ઑબ્જેક્ટને બહાર પાડે, તો તેને અનુરૂપ catchhandler હોવું જોઈએ. જોકે, તેમ છતાં, જો આપણે methodની અંદર જ અપવાદનો પ્રકાર ઝીલતા હોઈએ, તો તેને થ્રો કરવાની કોઈ જરૂર રહેતી નથી.




**આકૃતિ 10.20 ચાવીરૂપ શબ્દ throwsનો ઉપયોગ સમજાવતો પ્રોગ્રામ**

આકૃતિ 10.20માં દર્શાવેલ પ્રોગ્રામમાં આપણે PerformDivision() નામની એક મેથડ (method) લખી છે, આ મેથડમાં ArithmeticExceptionનો ઑબ્જેક્ટ ઉદ્ભવશે. આપણા પ્રોગ્રામની main() મેથડમાં performDivision() નામની મેથડને કોલ કરવામાં આવે છે. અત્રે એ તદ્દન સ્વાભાવિક છે કે, PerformDivision() મેથડમાં અપવાદ-વ્યવસ્થાપનતંત્ર ન હોવાને લીધે બોલાવનાર મેથડ દ્વારા જ અપવાદ ઝીલવામાં આવશે અને તેથી બોલાવનાર મેથડમાં અપવાદ-વ્યવસ્થાપન માટે Exception handler હોવું જોઈએ. એ જ રીતે, જાવાક્લાસના Constructorsમાં પણ અપવાદો હોઈ શકે છે.

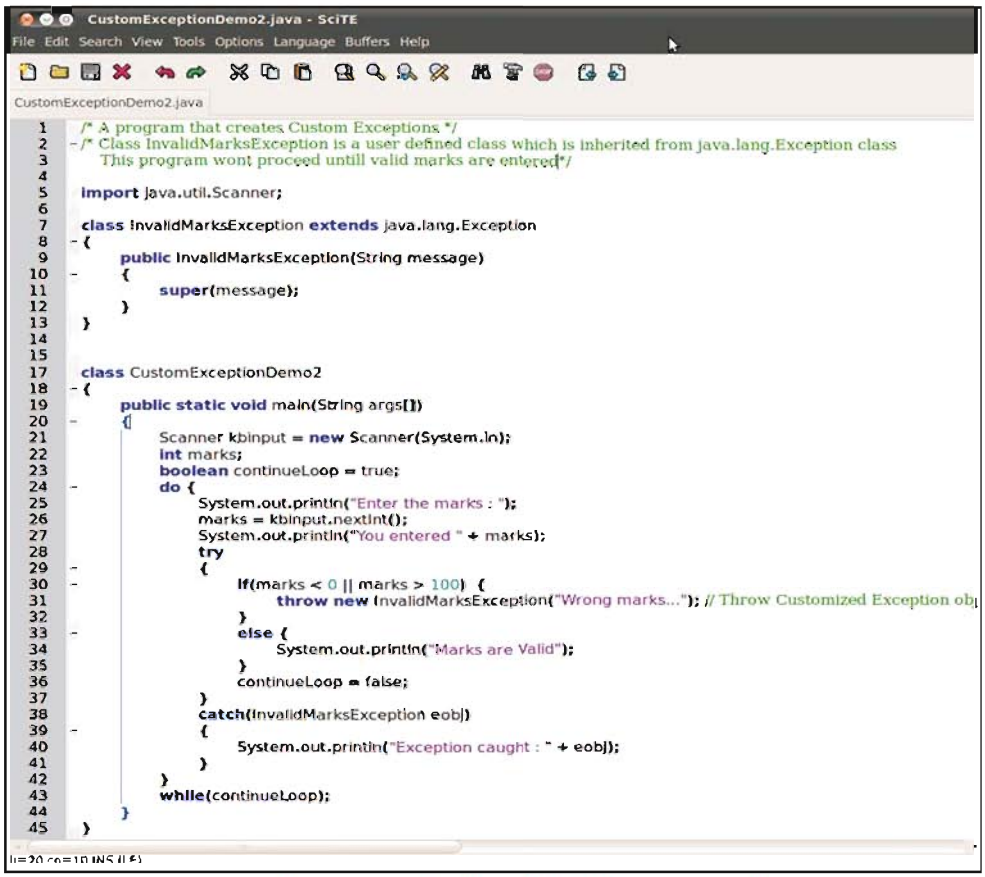
## જરૂરિયાત મુજબના અપવાદોનું સર્જન

જાવા જે-તે વિનિયોગની ખાસ સમસ્યાઓ અનુસાર આપણા પોતાના અપવાદોનું સર્જન કરવા દે છે. ઉદાહરણ તરીકે, ધારોકે, આપણે ગુણપત્રક તૈયાર કરવાનો પ્રોગ્રામ લખતા હોઈએ, જેમાં ઉપયોગકર્તાને વિવિધ વિષયના ગુણ દાખલ કરવા માટે પૂછવામાં આવતું હોય. દરેક વિષયના ગુણ 0થી 100ની વચ્ચે જ હોવા જોઈએ, ધારોકે, ઉપયોગકર્તા કોઈ વિષયના ગુણ તરીકે ઋણ સંખ્યા અથવા 100થી ઉપરની સંખ્યા દાખલ કરે, તો પ્રોગ્રામે તરત જ અપવાદ ઊભો કરવો જોઈએ. આવા પ્રકારના અપવાદ જે-તે વિનિયોગ પૂરતા ખાસ હોય છે. જાવા આવા વિનિયોગ માટે ખાસ એવા અપવાદો માટે આંતરપ્રસ્થાપિત અપવાદ-વર્ગ (Built-in Exception Classes) આપતા નથી.

એક્સેપ્શન ક્લાસનો સબક્લાસ બનાવીને આપણે ઉપયોગકર્તા-નિર્મિત અપવાદો તૈયાર કરી શકીએ. આ અપવાદોને throws વિધાનની મદદથી બાહ્ય રીતે થ્રો કરી શકાય. જોકે, આ અપવાદ ઝીલીને તેને યોગ્ય રીતે પાર પાડવા જરૂરી છે. એક પ્રોગ્રામ જોઈએ, જે ગુણની યથાર્થતા ચકાસવા જરૂરિયાત મુજબના અપવાદનું સર્જન કરતો હોય.

આકૃતિ 10.21માં દર્શાવેલ પ્રોગ્રામ ઉપયોગકર્તા તરફથી ઇનપુટ સ્વીકારે છે. 21મી લીટી પર, કી-બોર્ડથી ઇનપુટ સ્વીકારવા આપણે java.util.Scanner ક્લાસનો ઉપયોગ કર્યો છે. scanner ક્લાસની nextInt() પદ્ધતિ કૉન્સોલ (console) પરથી પૂર્ણાંક સંખ્યા વાંચવામાં મદદરૂપ બને છે. scanner ક્લાસની કાર્યપદ્ધતિ વિશેની ચર્ચા આપણે હવે પછીના પ્રકરણમાં કરીશું, જેમાં ફાઇલ અને I/Oની ચર્ચા કરવામાં આવેલ છે.

અહીં આપણે બે વર્ગ પ્રસ્થાપિત કર્યા. જરૂરિયાત અનુસારના અપવાદ (Custom Exception) તૈયાર કરવા એક વધારાના ક્લાસની જરૂરિયાત છે. "InvalidMarksException" વર્ગ java.lang પૅકેજના અપવાદવર્ગ (Exception Class)ને વિસ્તારે છે, તે સિંગલ પેરામીટર કન્સ્ટ્રક્ટર (Single Parameter Constructor) ધરાવે છે, જે ભૂલના પ્રકારને વર્ણવવા માટે શબ્દમાળાને (String) સ્વીકારે છે.




**આકૃતિ 10.21 : ઉપયોગકર્તા નિર્મિત અપવાદવર્ગ**

મુખ્ય પદ્ધતિમાં, (વિનિયોગ માટે ખાસ) વ્યાવસાયિક તર્ક મુજબ સૂચના લખવામાં આવી છે, જે ગુણ નિર્ધારિત મર્યાદાની અંદર જ છે કે કેમ તેની ખાત્રી કરે છે. જો દાખલ કરેલ ગુણ નિર્ધારિત મર્યાદાની અંદર નહીં આવતા હોય તો આપણે "InvalidMarksException" પ્રકારનો ઑબ્જેક્ટ તૈયાર કરીશું અને તેને થ્રો કરીશું. આ અપવાદને પાર પાડવા માટે એક catch વિભાગ હોવો જરૂરી છે. જ્યાં સુધી ગુણની યોગ્ય સંખ્યા દાખલ નહીં કરાય, ત્યાં સુધી આ પ્રોગ્રામ આગળ વધશે નહીં. જો ઉપયોગકર્તા ગુણની અયોગ્ય સંખ્યા દાખલ કરે તેવા કિસ્સામાં જ્યાં સુધી યોગ્ય સંખ્યા દાખલ નહીં થાય, ત્યાં સુધી ઉપયોગકર્તાને યોગ્ય સંખ્યા દાખલ કરવા કહ્યા કરવામાં આવશે. અત્રે એ નોંધવું જોઈએ કે, try-catch વિભાગને do-while લૂપની અંદર મૂકવામાં આવેલ છે. પ્રોગ્રામનું પરિણામ આકૃતિ 10.22માં દર્શાવવામાં આવેલ છે.


**આકૃતિ 10.22 : આકૃતિ 10.21માં દર્શાવેલ પ્રોગ્રામનું પરિણામ**

**નોંધ :** SciTE એ પ્રોગ્રામ ટાઇપ કરવા માટેનું એડિટર છે. જો પ્રોગ્રામ કી-બોર્ડ પાસેથી ડેટા સ્વીકારતો હોય તો પ્રોગ્રામને કમાન્ડપ્રોમ્પ્ટ પર અમલમાં મૂકવો સલાહભર્યું છે. જોકે, એવો સાદો પ્રોગ્રામ કે જેને ઉપયોગકર્તા સાથે સંવાદની બહુ જરૂર ન હોય તેવા પ્રોગ્રામને SciTE એડિટરની અંદર જ અમલમાં મૂકી શકાય છે.

### અપવાદરૂપ પરિસ્થિતિના વ્યવસ્થાપનના ફાયદા

આખા પ્રકરણમાં, આપણે જાવાપ્રોગ્રામમાં અપવાદરૂપ પરિસ્થિતિના વ્યવસ્થાપનના ઉપયોગનું સમર્થન કર્યું. હવે એ સ્પષ્ટ થવું જોઈએ કે, એક સારા પ્રોગ્રામનો અધવચ્ચે અચાનક અંત આવી જાય તેના કરતાં હંમેશાં અપેક્ષિત અપવાદોનું વ્યવસ્થાપન થાય તેમ કરવું જોઈએ. આપણા જાવાપ્રોગ્રામમાં અપવાદ-વ્યવસ્થાપનનો ઉપયોગ કરવાના ફાયદા ટૂંકમાં જોઈએ. જાવાપ્રોગ્રામમાં અપવાદ-વ્યવસ્થાપનનો ઉપયોગ કરવાના કેટલાક ફાયદા નીચે મુજબ છે :

- તે આપણને પ્રોગ્રામના સામાન્ય પ્રવાહને જાળવવામાં મદદરૂપ નીવડે છે. અપવાદ-વ્યવસ્થાપનની ગેરહાજરીમાં પ્રોગ્રામનો પ્રવાહ ક્યારેક અવરોધાય છે.
- તે સામાન્ય સૂચનાઓને બદલે અપવાદ-વ્યવસ્થાપન માટેની અલગ સૂચના લખવાની છૂટ આપે છે.
- ભૂલના પ્રકારોને એકજૂથમાં લાવી શકાય છે અને પ્રોગ્રામમાં અલગ તારવી શકાય છે.
- ક્લાયન્ટને પ્રોગ્રામ પૂરા પાડતાં પહેલાં, પ્રોગ્રામ ડીબગ થયેલ છે તેની ખાતરી આપી શકાય છે.
- પ્રોગ્રામનો અમલ કરતી વખતે અમલ દરમિયાન ઉદ્ભવતી વિવિધ ભૂલોને પકડી પાડવા સરળ તંત્ર-વ્યવસ્થા પૂરી પાડે છે.

### સારાંશ

આ પ્રકરણમાં આપણે એ શીખ્યા કે, ઉદ્ભવેલી ભૂલને દર્શાવવા પ્રોગ્રામ અપવાદોનો ઉપયોગ કરી શકે છે. પ્રોગ્રામ try, catch અને finally વિભાગનાં સંયોજનનો ઉપયોગ કરીને અપવાદને શોધી શકે છે.

અપવાદને બહાર પાડવા માટે આપણે throw વિધાનનો ઉપયોગ કરીએ છીએ અને તેને અપવાદના ઑબ્જેક્ટ (java.lang.throwable નો સબક્લાસ) પણ પ્રદાન કરીએ છીએ. એ પદ્ધતિ કે જે ન શોધી શકાયેલ અપવાદ મોકલે છે, તેની ઘોષણામાં throws ઉપવાક્યનો સમાવેશ હોવો જ જોઈએ. try વિધાનમાં ઓછામાં ઓછો એક catch વિભાગ અથવા finally વિભાગ અને એક કરતાં વધુ catch વિભાગ હોઈ શકે. અમુક ઘટના ક્યારેય ન બનવી જોઈએ, તે ચકાસવા નિશ્ચિતપણે ઉપયોગ કરાય છે. તે પ્રોગ્રામર દ્વારા સુધારા-વધારા કરવા માટે ઉપયોગમાં લેવાય છે.

### સ્વાધ્યાય

**1.** કંપાઇલેશન વખતની ભૂલ અને અમલ દરમિયાનની ભૂલ વચ્ચે શું તફાવત છે ?

**2.** અપવાદ એટલે શું ? જાવામાં મળતા કેટલાક અપવાદનાં ઉદાહરણ આપો.

**3.** જાવામાં અપવાદોનું વ્યવસ્થાપન કેવી રીતે કરવામાં આવે છે ?

**4.** try, catch અને finally વિભાગની ઉપયોગિતા શું છે ?

**5.** throw અને throws કી વર્ડની ઉપયોગિતા શું છે ?

**6.** જરૂરિયાત અનુસારનો અપવાદ તમે કેવી રીતે બનાવશો ?

**7.** નીચે આપેલા પ્રશ્નો માટે દરેકની સાથે આપેલા ચાર વિકલ્પોમાંથી યોગ્ય વિકલ્પ પસંદ કરી જવાબ આપો :

(1) ઑબ્જેક્ટ આધારિત પ્રોગ્રામિંગ પરિભાષામાં નીચેના પૈકી કઈ ભૂલની સ્થિતિ (error condition) ગણાય છે ?

(a) વિસંગતતા (anomaly)
(b) ટૂંકાક્ષર (abbreviation)
(c) અપવાદ (exception)
(d) ચલન (deviation)

(2) તમામ જાવા-અપવાદો માટે નીચેનામાંથી કયો શબ્દ સાચો છે ?

(a) ભૂલ (Errors)
(b) અમલ દરમિયાનના અપવાદ (Run-time exceptions)
(c) બહાર પાડી શકાય તેવા (Throwables)
(d) છોડી દઈ શકાય તેવા (Omissions)

(3) નીચેનામાંથી કયું વિધાન સાચું છે ?

(a) અપવાદો એ ભૂલ કરતાં વધુ ગંભીર છે.
(b) ભૂલ એ અપવાદો કરતાં વધુ ગંભીર છે.
(c) અપવાદો અને ભૂલો બન્ને એકસરખા ગંભીર છે.
(d) અપવાદો અને ભૂલ એક જ બાબત છે.

(4) નીચેનામાંથી કયું તત્ત્વ try વિભાગમાં સમાવવામાં આવતું નથી ?

(a) કી વર્ડ try
(b) કી વર્ડ catch
(c) છગડિયો કૌંસ
(d) ક્યારેક અપવાદ સર્જતાં વિધાનો

(5) અપવાદ સર્જાય ત્યારે નીચેનામાંથી કયો વિભાગ વ્યવસ્થાપન કરે છે કે યોગ્ય ક્રિયા હાથ ધરે છે ?

(a) try
(b) catch
(c) throws
(d) handles

(6) નીચેનામાંથી કયું catch વિભાગની અંદર હોવું જોઈએ ?

(a) finally વિભાગ
(b) અપવાદનું વ્યવસ્થાપન કરતું એક વિધાન
(c) અપવાદનું વ્યવસ્થાપન કરવા જરૂરી બધાં વિધાન
(d) throws કી વર્ડ

(7) જ્યારે try વિભાગ કોઈ અપવાદ નહીં સર્જે અને તમે અનેક catch વિભાગ બનાવ્યા હશે, ત્યારે શું થશે ?

(a) તમામનો અમલ થશે.
(b) માત્ર બંધબેસતું પ્રથમ અમલમાં આવશે.
(c) એક પણ catch વિભાગ અમલમાં આવશે નહીં.
(d) માત્ર પ્રથમ catch વિભાગ અમલમાં આવશે.

(8) નીચેનામાંથી try...catch વિભાગનો ઉપયોગ કરવાનો ફાયદો કયો છે ?

(a) અપવાદરૂપ ઘટના દૂર કરવામાં આવે છે.
(b) અપવાદરૂપ ઘટના ઓછી કરવામાં આવે છે.
(c) અપવાદરૂપ ઘટનાઓને નિયમિત ઘટનાઓ સાથે સાંકળવામાં આવે છે.
(d) અપવાદરૂપ ઘટનાઓને, નિયમિત ઘટનાઓથી અલિપ્ત કરવામાં આવે છે.

(9) નીચેના પૈકી કઈ પદ્ધતિ (method) અપવાદને થ્રો કરી શકે ?

(a) throws ઉપવિધાન સાથેની પદ્ધતિ
(b) catch વિભાગ સાથેની પદ્ધતિ
(c) try વિભાગ સાથેની પદ્ધતિ
(d) finally વિભાગ સાથેની પદ્ધતિ

(10) કોઈ મેથડનો પૂર્ણ રીતે ઉપયોગ શક્ય છે કે કેમ તે જાણવા માટે નીચેના પૈકી કયું ઓછું અગત્યનું છે ?

(a) મેથડના પરિણામનો પ્રકાર

(b) મેથડને જરૂરી આર્ગ્યુમેન્ટનો પ્રકાર

(c) મેથડમાં રહેલ વિધાનોની સંખ્યા

(d) મેથડ દ્વારા થ્રો (throw) કરવામાં આવતા અપવાદોનો પ્રકાર

## પ્રાયોગિક સ્વાધ્યાય

1. એક એવો જાવાપ્રોગ્રામ લખો કે જે એરેમાં એલિમેન્ટ ઉમેરવા "add()", નામની પદ્ધતિ (method)નો ઉપયોગ કરતો હોય, પદ્ધતિમાં એક try વિભાગ પણ ઉમેરો, જેથી કરીને પદ્ધતિ ArrayIndexOutOfBoundsExceptionને સંભાળી શકે.

2. ઉપર્યુક્ત ઉદાહરણમાં, try...catch વિભાગને પદ્ધતિમાંથી કાઢી નાંખો. "add()" પદ્ધતિને શરૂ કરનારી પદ્ધતિ throws ચાવીરૂપ શબ્દનો ઉપયોગ કરીને અપવાદનું વ્યવસ્થાપન કરી શકવી જોઈએ.

3. જ્યારે તમે કોઈ ઋણ સંખ્યાનું વર્ગમૂળ (square root) મેળવવા પ્રયત્ન કરતા હો, ત્યારે એક ગાણિતિક અપવાદ (ArithmeticException) થ્રો કરે અને ઝીલે (catch કરે) એવો જાવાપ્રોગ્રામ લખો. ઉપયોગકર્તાને કોઈ સંખ્યા દાખલ કરવા જણાવો અને એના ઉપર Math.sqrt() પદ્ધતિ (method)નો ઉપયોગ કરવાનો પ્રયત્ન કરો. આ વિનિયોગ વર્ગમૂળ દર્શાવશે અથવા અપવાદને ઝીલીને યોગ્ય સંદેશ દર્શાવશે. આ પ્રોગ્રામને SqrtException.java નામ આપી સાચવી દો.

4. એક એવો જાવાપ્રોગ્રામ લખો કે જે, જન્મતારીખની યથાર્થતા ચકાસે. આ માટે જરૂરિયાત પ્રમાણેનો અપવાદ "InvalidBirthDateException" બનાવો. ઉપયોગકર્તાને કોઈ પણ તારીખ દાખલ કરવા જણાવો. જો જન્મતારીખ હાલની તારીખ પછીની કોઈ તારીખ હોય તો "InvalidBirthDateException" નામનો અપવાદ બહાર પાડો (throw કરો). આ અપવાદને પાર પાડવા માટે યોગ્ય સૂચનાઓ (code) લખો.

5. એક એવો જાવાપ્રોગ્રામ લખો કે જે, બૅંક્વ્યવહારોને ગોઠવે. balanceAmount અને withdrawAmount નામના બે ચલ લો. જો withdrawAmount એ balanceAmount કરતાં ઓછી હોય, તો પ્રોગ્રામ તરત જ ધ્યાન દોરે તેવું થવું જોઈએ.

6. એક એવો જાવાપ્રોગ્રામ લખો કે જે, બૅંક્વ્યવહારોને ગોઠવે. balanceAmount અને withdrawAmount નામના બે ચલ લો. આ માટે ખાસ જરૂરિયાત માટેનો "InvalidTransaction" નામનો અપવાદ બનાવો. જો withdrawAmount એ balanceAmount કરતાં ઓછી હોય, તો તમારો પ્રોગ્રામ તરત જ "InvalidTransaction" નામનો અપવાદ થ્રો કરવો જોઈએ. આ અપવાદને પાર પાડવા માટે યોગ્ય સૂચનાઓ (Exception handlers) લખો.

7. એક એવો જાવાપ્રોગ્રામ લખો કે જે, જન્મતારીખની યથાર્થતા ચકાસે. આ માટે ખાસ જરૂરિયાત અનુસારના "InvalidDateException", "InvalidMonthException" અને "InvalidYearException" નામના ત્રણ અપવાદ બનાવો. જો તારીખ ઋણ હોય અથવા 30 કરતાં વધુ હોય, તો "InvalidDateException" અપવાદ થ્રો કરો. જો મહિનો ઋણ હોય અથવા 12 કરતાં વધુ હોય, તો "InvalidMonthException" અપવાદ થ્રો કરો. જો વર્ષ 1950 અથવા હાલના વર્ષ પછીનું વર્ષ હોય, તો "InvalidYearException" અપવાદ થ્રો કરો.

# ફાઇલ-વ્યવસ્થાપન 11

આ પ્રકરણમાં આપણે જાવાપ્રોગ્રામ દ્વારા હાર્ડડિસ્ક પરની જુદી-જુદી ફાઇલ અને ડિરેક્ટરીને કેવી રીતે ઉપયોગમાં લેવી, ઓળખવી અને તેનું સંવર્ધન કરવું તે જોઈશું. સમગ્ર પ્રકરણ દરમિયાન આપણે ઇનપુટ/આઉટપુટ પ્રક્રિયા અને ફાઇલ-વ્યવસ્થાપન પર ધ્યાન કેન્દ્રિત કરીશું. આપણે java.io પૅકેજમાં ઉપલબ્ધ સૌથી સામાન્ય ક્લાસ (classes) અને java.util પૅકેજના થોડા ક્લાસનો ઉપયોગ કરીશું. તો હવે, ફાઇલ અને ડિરેક્ટરીનાં વિહંગાવલોકન (overview) સાથે તેની ચર્ચા શરૂ કરીએ.

### કમ્પ્યૂટર ફાઇલને સમજીએ :

કમ્પ્યૂટર સિસ્ટમનાં સંગ્રહ-સાધનોને નાશવંત સંગ્રહ (Volatile Storage) અને અવિનાશી સંગ્રહ (Non-volatile Storage) એમ બે વિભાગમાં વહેંચવામાં આવે છે.

નાશવંત સંગ્રહ એ કામચલાઉ છે. જ્યારે કમ્પ્યૂટર બંધ થાય છે, ત્યારે ચલમાં સાચવેલી માહિતી નાશ પામે છે. જાવા-પ્રોગ્રામ કે જે, ચલમાં કિંમતો સાચવે છે, તે રેન્ડમ ઍક્સેસ મેમરી (RAM)નો ઉપયોગ કરે છે. સામાન્ય રીતે, ચલ, ઑબ્જેક્ટ અને તેના અનુસંધાન RAM માં સાચવવામાં આવે છે. એક વાર પ્રોગ્રામનો અંત આવે અથવા કમ્પ્યૂટર બંધ થાય ત્યારે ડેટા નાશ પામે છે.

અવિનાશી સંગ્રહ એ કાયમી સંગ્રહ છે; જ્યારે કમ્પ્યૂટરને મળતો વીજપ્રવાહ બંધ થાય, ત્યારે પણ ડેટા નાશ પામતો નથી. જાવાપ્રોગ્રામ જ્યારે ડિસ્ક ઉપર સાચવવામાં આવે છે, ત્યારે આપણે કાયમી સંગ્રહનો ઉપયોગ કરીએ છીએ. કમ્પ્યૂટર ફાઇલ એ અવિનાશી સાધન પર સંગ્રહવામાં આવતો ડેટાસમૂહ છે. ફાઇલો હાર્ડડિસ્ક, USB ડ્રાઇવ, ઑપ્ટિકલ ડિસ્ક અને કૉમ્પેક્ટ ડિસ્ક જેવાં કાયમી સંગ્રહસાધનો પર અસ્તિત્વ ધરાવે છે. ફાઇલમાં સંગ્રહવામાં આવતો ડેટા ઘણી વાર સતત ડેટા (persistent data) તરીકે પણ ઓળખાય છે.

ફાઇલોને વળી પાછી ટેક્સ્ટફાઇલ અને દ્વિઅંકી (Binary) ફાઇલ એમ મુખ્યત્વે બે ભાગમાં વહેંચી શકાય.

ટેક્સ્ટફાઇલ એવો ડેટા ધરાવે છે કે જે કોઈ ટેક્સ્ટ એડિટર દ્વારા વાંચી શકાય છે, કારણકે, ડેટા ASCII અથવા Unicode જેવી પદ્ધતિના ઉપયોગ દ્વારા સંજ્ઞાંકિત કરાયેલ હોય છે. ટેક્સ્ટફાઇલ ડેટાફાઇલ હોઈ શકે છે, જે હકીકતો ધરાવતી હોય, જેવી કે, પે-રોલ ફાઇલ જે કર્મચારી ક્રમ, નામ અને પગાર સમાવે છે; અથવા કેટલીક ટેક્સ્ટફાઇલ એ પ્રોગ્રામફાઇલ અથવા વિનિયોગ-ફાઇલ પણ હોઈ શકે, જે સૉફ્ટવેર માટેની સૂચનાઓ સાચવી શકે છે. gedit, vi, pico જેવા એડિટર દ્વારા તૈયાર કરાયેલી ફાઇલો એ ટેક્સ્ટફાઇલનાં ઉદાહરણ છે. તેમનાં અનુલંબન તરીકે txt, java અથવા c હોઈ શકે છે.

દ્વિઅંકી ફાઇલ એવો ડેટા ધરાવે છે, જે લખાણ તરીકે સંજ્ઞાંકિત (Encoded) કરાયો નથી. તેનું લખાણ દ્વિઅંકી સ્વરૂપે હોય છે, જેનો અર્થ એ છે કે, ડેટાને બાઇટ સ્વરૂપે ઉપયોગમાં લેવાય છે. દ્વિઅંકી ફાઇલનાં કેટલાંક ઉદાહરણરૂપ અનુલંબનો .jpeg, .mp3 અને .class છે.

જાવા ભાષા વિવિધ ક્રિયાઓને સમર્થન આપે છે, જે ફાઇલ અથવા ડિરેક્ટરી પર કરી શકાય છે. જાવાપ્રોગ્રામના ઉપયોગ દ્વારા ફાઇલ પર કરી શકાતી કેટલીક ક્રિયાઓની યાદી નીચે મુજબ છે :

- ફાઇલ કે ડિરેક્ટરીનો પથ નક્કી કરવો.
- ફાઇલ ખોલવી.
- ફાઇલમાં લખવું.
- ફાઇલમાંથી વાંચવું.

- ફાઇલને બંધ કરવી.
- ફાઇલને કાઢી નાંખવી.
- ફાઇલના ગુણધર્મો (attributes)ની પૃચ્છા કરવી.

જાવા એવા તૈયાર આંતરપ્રસ્થાપિત ક્લાસ (inbuilt classes) આપે છે, જેમાં આવાં બધાં કામો પાર પાડવા આપણને મદદરૂપ થવા માટેની પદ્ધતિ (method) આપેલી જ હોય છે. આ ક્લાસ java.io પૅકેજમાં ઉપલબ્ધ હોય છે. જાવાસ્ટ્રીમ (streams)ના ખ્યાલનો ઉપયોગ કરે છે અને તે બાઇટ્સ અને અક્ષરો પર I/O ક્રિયાઓને પાર પાડવા બે જુદી-જુદી કક્ષાના જાવા- ક્લાસ પૂરા પાડે છે. ફાઇલમાંથી માહિતી વાંચવી અને ફાઇલમાં માહિતી લખવાની વિસ્તૃત ચર્ચા આગળના વિભાગમાં કરવામાં આવશે.

## જાવામાં ફાઇલ-ક્લાસ (File Class in Java)

java.io.File ક્લાસમાં ફાઇલ કે ડિરેક્ટરીની લાક્ષણિકતા વિશેની માહિતીનો સમાવેશ કરવામાં આવે છે. ફાઇલ કે ડિરેક્ટરીઓના ગુણધર્મો (ઍટ્રિબ્યૂટ્સ) મેળવવા ફાઇલ-ક્લાસનો ઉપયોગ કરી શકાય. આપણે ફાઇલ કે ડિરેક્ટરીને create/rename/delete કરી શકીએ છીએ. આપણે ફાઇલના વિવિધ ગુણધર્મો પણ જાણી શકીએ. જેવા કે, ફાઇલના ઉપયોગની પરવાનગી, ફાઇલનું કદ અથવા ફાઇલમાં છેલ્લે ક્યારે સુધારા-વધારા કરવામાં આવ્યા હતા તે સમય વગેરે. ફાઇલ-ક્લાસ સાથે સંબંધ ધરાવતો હોય તેવા ફાઇલ-ઑબ્જેક્ટનું સર્જન કરવાનો ગર્ભિત અર્થ એ થતો નથી કે, તે ફાઇલ કે ડિરેક્ટરી અસ્તિત્વ ધરાવે છે. એક ફાઇલ-ઑબ્જેક્ટમાં હાર્ડડિસ્ક પરની ફાઇલ કે ડિરેક્ટરીનું પથનામ અથવા સરનામું મૂકવામાં આવે છે.

ફાઇલ અથવા ડિરેક્ટરી પર વિવિધ પ્રક્રિયાઓ કરવા માટે ઉપયોગમાં લઈ શકાય, તેવી ફાઇલ-ક્લાસની આશરે 30 પદ્ધતિઓ છે. જો કે તેમ છતાં, ફાઇલમાંથી વાંચવા માટે કે ફાઇલમાં લખવા માટે ફાઇલ-ક્લાસ કોઈ પદ્ધતિ પૂરી પાડતા નથી. આવી ક્રિયાઓ પાર પાડવા માટે ઘણાં બધા 'સ્ટ્રીમ-ક્લાસ' (પ્રવાહક્લાસ) ઉપલબ્ધ છે. તો ચાલો, આપણે ફાઇલ-ક્લાસના વધુ ઉપયોગમાં લેવાતા સર્જકો (constructors) અને પદ્ધતિઓ (methods) વિશે શીખવાની શરૂઆત કરીએ.

## ફાઇલ-ક્લાસના સર્જકો (Constructors of File Class)

ફાઇલ-ક્લાસના ઉપયોગ દ્વારા આપણે, કોઈ પણ ફાઇલને તેનો સંબંધિત પથ આપીને અથવા શબ્દમાળા સ્વરૂપે તેનો સંપૂર્ણ પથ આપીને અનુસંધાન આપી શકીએ. કોઈ ફાઇલ અથવા ડિરેક્ટરીનો ઉલ્લેખ કરવા ફાઇલ-ક્લાસ નીચે મુજબના સર્જકો પૂરા પાડે છે :

```
File(String path)
File(String directory_path, String file_name)
File(File directory, String file_name)
```

હવે, ઉપર્યુક્ત સર્જકો માટે એક ઉદાહરણ લઈએ. લિનક્સમાં "/etc" ડિરેક્ટરીમાં ઉપલબ્ધ "passwd" ફાઇલ, સિસ્ટમમાં હયાત ઉપયોગકર્તાની માહિતી સાચવે છે. ધારો કે, આપણે તેના ગુણધર્મો (attributes) દર્શાવવા હોય, તો નીચે દર્શાવેલ ત્રણ રીતના ઉપયોગ દ્વારા તેના જાવાફાઇલ ઑબ્જેક્ટ બનાવી શકાય :

- File fileobj = new File("/etc/passwd"); એવો પથ દર્શાવીને.
- ડિરેક્ટરી અને ફાઇલનામને બે જુદા-જુદા આર્ગ્યુમેન્ટ દર્શાવીને

```
File fileobj = new File("/etc", "passwd");
```

- dirobj ઑબ્જેક્ટમાં મૂકેલી ડિરેક્ટરીના અનુસંધાન વાપરીને

```
File dirobj = new File("/etc");
File fileobj = new File(dirobj, "passwd");
```

## ફાઇલ-ક્લાસની પદ્ધતિઓ

કોષ્ટક 11.1 ફાઇલ-ક્લાસની વધુ ઉપયોગમાં લેવાતી પદ્ધતિઓનો સારાંશ રજૂ કરે છે.

| પદ્ધતિ (Method) | વર્ણન (Description) |
|---|---|
| boolean exists() | જો ફાઇલ કે ડિરેક્ટરી હયાત હશે, તો true કિંમત પરત કરશે અન્યથા false કિંમત પરત કરે છે. |
| boolean isFile() | જો ફાઇલ હયાત હશે, તો true કિંમત પરત કરશે અન્યથા false કિંમત પરત કરે છે. |
| boolean isDirectory() | જો ડિરેક્ટરી હયાત હશે, તો true કિંમત પરત કરશે અન્યથા false કિંમત પરત કરે છે. |
| boolean isHidden() | જો ફાઇલ કે ડિરેક્ટરી છુપાયેલી હશે, તો true કિંમત પરત કરે છે. |
| String getAbsolutePath() | ફાઇલ કે ડિરેક્ટરીનો સંપૂર્ણ પથ (absolute path) પરત કરે છે. |
| String getName() | ઑબ્જેક્ટ દ્વારા સંબોધાયેલ ફાઇલ કે ડિરેક્ટરીનું નામ પરત કરે છે. |
| String getPath() | ફાઇલ કે ડિરેક્ટરીનો પથ (path) પરત કરે છે. |
| long length() | તે ફાઇલમાં રહેલા બાઇટ્સની સંખ્યા પરત કરે છે. |
| String[] list() | ડિરેક્ટરીમાં ઉપલબ્ધ ફાઇલો અને ડિરેક્ટરીઓનાં નામ પરત કરે છે. |
| File[] listFiles() | ડિરેક્ટરીમાં ફાઇલ સૂચવતા પથનામના સારાંશ (abstract pathnames)નો ઍરે પરત કરે છે. |

**કોષ્ટક 11.1 : ફાઇલ-ક્લાસની વધુ ઉપયોગમાં આવતી પદ્ધતિઓ**

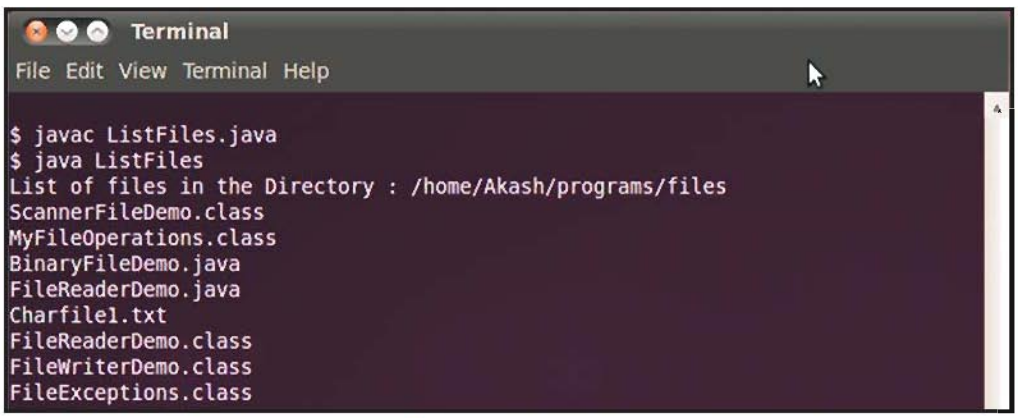
હવે કોઈ આપેલી ડિરેક્ટરીમાં ઉપલબ્ધ ફાઇલોનાં નામની યાદી આપતા પ્રોગ્રામને સમજવાનો પ્રયત્ન કરીએ. કોઈ ચોક્કસ ડિરેક્ટરીનો ઉલ્લેખ કરતા ફાઇલ-ક્લાસનો ઑબ્જેક્ટ બનાવ્યા પછી, આપણે તે ડિરેક્ટરીમાં ઉપલબ્ધ તમામ ફાઇલોની યાદી મેળવવા list() પદ્ધતિનો ઉપયોગ કરી શકીએ. કોડલિસ્ટિંગ 11.1માં આપેલ પ્રોગ્રામ "/home/Akash/programs/files" ડિરેક્ટરીમાં ઉપલબ્ધ ફાઇલોની યાદી મેળવવાનું નિદર્શન કરે છે. કોઈ ચોક્કસ ડિરેક્ટરી માટે ફાઇલો અને ડિરેક્ટરીઓની સંખ્યા ગણવા માટે પ્રોગ્રામમાં આપણે એક ચલનો ઉપયોગ કરેલ છે તેમજ ડિરેક્ટરીમાં ઉપલબ્ધ ફાઇલ-ઑબ્જેક્ટને સાચવવા ફાઇલ-ક્લાસનો listOfFiles નામનો ઍરેનો ઉપયોગ કરાયો છે.

```
//List the contents of a Directory
import java.io.File;
public class ListFiles
{
    public static void main(String args[])
    {
        //Provide a Directory Path
        String path = "/home/Akash/programs/files";
        String files;
        int countOfFiles;
        try
        {
```

```
                File folder = new File(path);
                //Store the list of files in an array of File[] objects
                File[] listOfFiles = folder.listFiles();
                //count the number of files in the folder
                countOfFiles = listOfFiles.length;

                System.out.print("List of files in the Directory : ");
                System.out.println(folder.getAbsolutePath());

                //Iterate to display the name of each file
                for(int i=0; i<countOfFiles; i++)
                {
                        if(listOfFiles[i].isFile())
                        {
                                files = listOfFiles[i].getName();
                                System.out.println(files);
                        }
                }
            }
            catch(Exception eobj)
            {
                System.out.println(eobj);
            }
        }
    }
```

કોડલિસ્ટિંગ 11.1 : આપેલ ડિરેક્ટરીમાં ઉપલબ્ધ ફાઇલોની યાદી મેળવવાનો પ્રોગ્રામ

કોડલિસ્ટિંગ 11.1નું પરિણામ આકૃતિ 11.1માં દર્શાવેલ છે.




11.1 : કોડલિસ્ટિંગ 11.1નું પરિણામ

**નોંધ :**

ઉપર્યુક્ત પ્રોગ્રામ "/home/Akash/programs/files" રિરેક્ટરીમાં હયાત ફાઇલ અને રિરેક્ટરીઓની યાદી રજૂ કરે છે. પરિણામ નિહાળવા માટે આપણે યોગ્ય પથ અને ફાઇલના નામનો ઉપયોગ કરી શકીએ.

## સ્ટ્રીમ (Stream)નો પરિચય

ફાઇલને કેવી રીતે સુધારવી કે ફાઇલના લખાણને કેવી રીતે દર્શાવવું તે અત્યાર સુધી આપણે શીખ્યા નથી. આવી ક્રિયાઓ કરવા માટે આપણે સ્ટ્રીમ (stream)ના ખ્યાલને સમજવો પડશે. ફાઇલમાં લખવા કે ફાઇલમાંથી વાંચવાની ક્રિયા પાર પાડવા માટે જાવા સ્ટ્રીમ-ક્લાસનો ઉપયોગ કરે છે.

ઇનપુટ અને આઉટપુટ માટે વપરાતાં સાધનોના પ્રકાર વિશે આપણે અગાઉ શીખી જ ગયા છીએ. ઉદાહરણ તરીકે કી-બોર્ડ એ એક ઇનપુટ માટેનું સાધન છે, જ્યારે મૉનિટર એ આઉટપુટ માટેનું સાધન છે. હાર્ડડિસ્કમાં સાચવેલ ફાઇલમાંથી આપણે ડેટા વાંચી પણ શકીએ છીએ અને તેમાં લખી પણ શકીએ છીએ, તે કારણથી હાર્ડડિસ્કને ઇનપુટ તેમજ આઉટપુટ એમ બંને માટેનાં સાધન તરીકે વર્ગીકૃત કરી શકાય. વિવિધ ક્ષમતા અને વિવિધ ઉત્પાદકો તરફથી બજારમાં અનેક સાધનો ઉપલબ્ધ છે. ઉદાહરણ તરીકે, હાર્ડડિસ્ક અનેક ઉત્પાદકો દ્વારા બને છે અને 500GB અથવા 1 TB એમ જુદી-જુદી સંગ્રહક્ષમતાવાળી હોય છે. આ ઉપરાંત હાર્ડડિસ્કને USB અથવા SATA જેવા જુદાં-જુદાં કેબલ વડે જોડી શકાય છે. તેમ છતાં, ફાઇલોમાં ડેટા લખવા કે વાંચવાની ક્રિયા કરવાનો પ્રોગ્રામ લખતી વખતે જાવાના પ્રોગ્રામરે હાર્ડડિસ્કનો પ્રકાર કે તેની સંગ્રહક્ષમતા જેવી બાબતો વિશે ચિંતા કરવાની જરૂર નથી. આવું એટલા માટે શક્ય બને છે, કારણકે જાવા સ્ટ્રીમની કાર્યપદ્ધતિ પૂરી પાડે છે.

સ્ટ્રીમ એ ડેટા માટે મૂળ કે ગંતવ્ય તરીકે ઉપયોગમાં લેવાતા ઇનપુટ કે આઉટપુટ સાધનની ટૂંકી રજૂઆત છે. સ્ટ્રીમને આપણે પ્રોગ્રામમાં આવતા કે પ્રોગ્રામમાંથી જતા બાઇટની હારમાળા તરીકે કલ્પી શકીએ. આપણે સ્ટ્રીમનો ઉપયોગ કરીને ડેટાને લખી શકીએ કે વાંચી શકીએ.

જ્યારે આપણે સ્ટ્રીમમાં ડેટા લખતા હોઈએ, ત્યારે તે સ્ટ્રીમને આઉટપુટ-સ્ટ્રીમ (Output stream) કહે છે. આઉટપુટ-સ્ટ્રીમ પ્રોગ્રામમાંથી હાર્ડડિસ્ક પર ફાઇલમાં અથવા મૉનિટર પર અથવા નેટવર્કના કોઈ એક કમ્પ્યૂટર પર ડેટાને તબદીલ કરી શકે છે. પ્રોગ્રામ માટે બાહ્ય સાધન પરથી ડેટા વાંચવા માટે ઇનપુટ-સ્ટ્રીમ (Input stream) ઉપયોગમાં લેવાય છે. તે કી-બોર્ડ પરથી અથવા હાર્ડડિસ્ક પરની ફાઇલમાંથી ડેટાને પ્રોગ્રામમાં તબદીલ કરી શકે છે.

ઇનપુટ કે આઉટપુટ ક્રિયાઓ માટે સ્ટ્રીમનો ઉપયોગ કરવા પાછળનું મુખ્ય કારણ, આપણા પ્રોગ્રામને ઉપયોગમાં લેવાનારાં સાધનોથી સ્વતંત્ર બનાવવાનો છે. સ્ટ્રીમના બે મુખ્ય લાભ નીચે મુજબ છે :

- સાધનોની ટેક્નિકલ વિગતો વિશે જાણવાની પ્રોગ્રામરને ચિંતા કરવાની જરૂર રહેતી નથી.
- પ્રોગ્રામની મૂળ સૂચનાઓ (સૉર્સકોર્ડ)માં કોઈ પણ સુધારા કર્યા વિના પ્રોગ્રામ વિવિધ પ્રકારનાં ઇનપુટ/આઉટપુટ સાધનો માટે કામ કરી શકે છે.

## જાવામાં સ્ટ્રીમ-ક્લાસ

બાઇટ-સ્ટ્રીમ (દ્વિઅંકી પ્રવાહ) / અને કૅરેક્ટર-સ્ટ્રીમ (શબ્દપ્રવાહ)ને સમજવા માટે ચાલો સૌપ્રથમ આપણે બાઇટ અને અક્ષરની રજૂઆત વચ્ચેનો ભેદ સમજવાનો પ્રયત્ન કરીએ. ચાલો, આપણે સંખ્યા "5"નું ઉદાહરણ લઈએ; કોષ્ટક 11.2માં દર્શાવ્યા મુજબ આપણે આ સંખ્યાને બે જુદી-જુદી રીતે રજૂ કરી શકીએ :

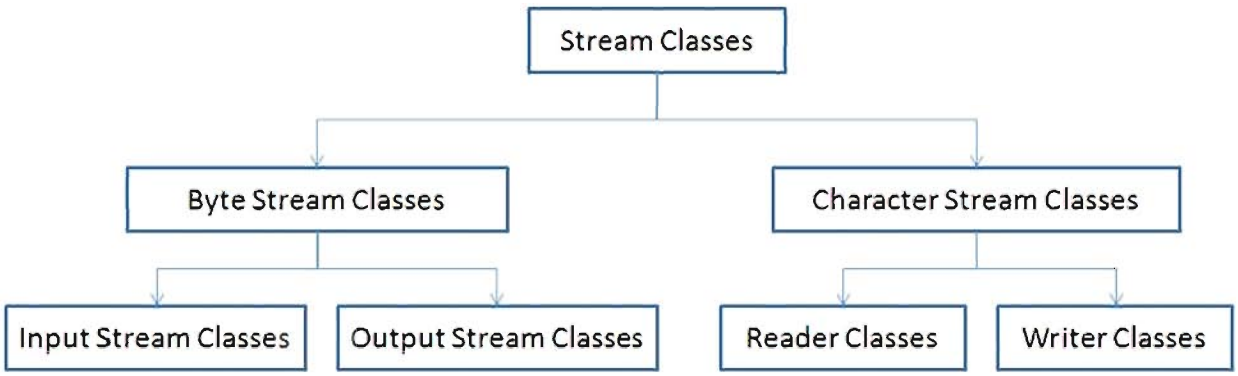
| રજૂઆત | વિગત | બાઇનરી રજૂઆત |
|---|---|---|
| અક્ષર 5 | ASCII કિંમત : 53 | 00110101 |
| બાઇનરી અંક 5 | બાઇનરી કિંમત : 5 | 00000101 |

**કોષ્ટક 11.2 : અક્ષર અને બાઇનરી રજૂઆત**

કોઈ પણ અક્ષરને સામાન્ય રીતે ASCII અથવા યુનિકોડ (Unicode) સ્વરૂપે સાચવવામાં આવે છે. પરંતુ, જ્યારે તેને ગણતરીના ઉદ્દેશ્યથી વાપરવામાં આવે છે, ત્યારે તેની બાઇનરી કિંમત અર્થપૂર્ણ બને છે. એક વધુ ઉદાહરણ લઈએ, int i = 32 વિધાન 'i' નામના ચલને પૂર્ણાંક પ્રકારના ચલ તરીકે ઘોષિત કરે છે, જેમાં 32 કિંમત સચવાય છે. જોકે 32ને '3' અને '2' એમ બે જુદા-જુદા અક્ષરો તરીકે રજૂ કરી શકાય. જ્યારે આપણે "Human beings have 32 teeth" એવું વાક્ય લખતા હોઈએ, ત્યારે શાબ્દિક રજૂઆત સલાહભરી છે, જ્યાં આપણે સંખ્યા ઉપર કોઈ પણ પ્રકારની ક્રિયાઓ કરતાં નથી. પરંતુ, જ્યારે આપણે ગાણિતિક ક્રિયાઓ કરવી હોય, ત્યારે આપણે int, float અથવા double જેવા ડેટાપ્રકાર (datatype)નો ઉપયોગ કરીએ છીએ, જે આપણને સંખ્યા દ્વિઅંકી સ્વરૂપે સાચવવા દે છે.

જાવા બે પ્રકારની સ્ટ્રીમને માન્યતા આપે છે, બાઇટ-સ્ટ્રીમ (byte stream) અને કૅરેક્ટર-સ્ટ્રીમ (character stream). એવી સ્ટ્રીમ કે જે, ડેટાને ફાઇલમાં કે કોઈ સાધન તરફ બાઇટ સ્વરૂપે તબદીલ કરે તેને બાઇટ-સ્ટ્રીમ અથવા બાઇનરી સ્ટ્રીમ કહે છે. બાઇટ-સ્ટ્રીમના ઉપયોગથી તૈયાર થતી ફાઇલોને બાઇનરી ફાઇલ તરીકે ઓળખવામાં આવે છે. ધારો કે, જો આપણે ફાઇલમાં પૂર્ણાંક, ડબલ (double) અથવા બુલિયન (boolean) જેવા ચલ સાચવવા ઇચ્છતા હોઈએ, તો આપણે બાઇનરી ફાઇલ જ ઉપયોગમાં લેવી પડે. એરે કે ઑબ્જેક્ટ સાચવવા માટે પણ બાઇનરી ફાઇલનો ઉપયોગ કરી શકાય છે. એ જ રીતે, ટેક્સ્ટફાઇલો અને પ્રોગ્રામકોડ, બનાવવા કૅરેક્ટર-સ્ટ્રીમનો ઉપયોગ કરાય છે. vi અથવા SciTE જેવા ટેક્સ્ટ એડિટરમાં તેને ખોલી શકાય છે.

જાવાક્લાસના બે ગણ પૂરા પાડે છે, કૅરેક્ટર-સ્ટ્રીમ-ક્લાસ અને બાઇનરી સ્ટ્રીમ-ક્લાસ. java.io પૅકેજમાં ઉપલબ્ધ કૅરેક્ટર-સ્ટ્રીમ-ક્લાસ કૅરેક્ટર/ડેટા સાથે કામ કરે છે, જ્યારે java.io પૅકેજમાં રહેલ બાઇટ-સ્ટ્રીમ-ક્લાસ બાઇનરી ડેટા સાથે કામ કરે છે. આકૃતિ 11.2 સ્ટ્રીમ-ક્લાસનું વર્ગીકરણ દર્શાવે છે.



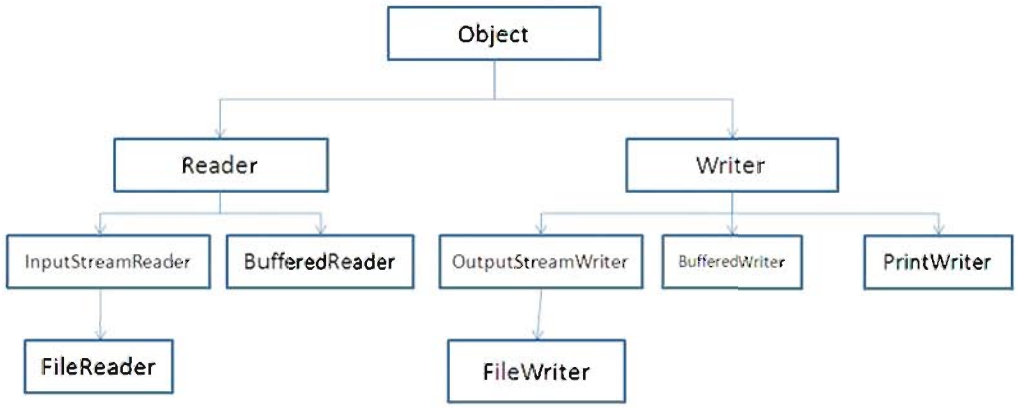

**11.2 : સ્ટ્રીમ-ક્લાસનું વર્ગીકરણ**

જાવા-સ્ટ્રીમને ઇનપુટ-સ્ટ્રીમ અને આઉટપુટ-સ્ટ્રીમ નામના બે મૂળ પ્રકારમાં વર્ગીકૃત કરી શકાય. ઇનપુટ-સ્ટ્રીમ ઉદ્ગમ (ફાઇલ, કી-બોર્ડ)માંથી ડેટા વાંચે છે, જ્યારે આઉટપુટ-સ્ટ્રીમ ગંતવ્યસ્થાન (ફાઇલ, આઉટપુટ સાધનો) પર ડેટા લખે છે.

java.io પૅકેજ સ્ટ્રીમ-ક્લાસનો એવો સમૂહ ધરાવે છે, જે ફાઇલમાં લખવાની અને વાંચવાની ક્રિયાને શક્ય બનાવે છે. આ ક્લાસનો ઉપયોગ કરવા માટે પ્રોગ્રામ દ્વારા java.io પૅકેજને આયાત (import) કરવું જરૂરી છે. આમ તો, કૅરેક્ટર અને બાઇટ પર ક્રિયા કરવા માટે ઘણા ક્લાસ છે. જોકે આપણે અહીં સૌથી વધુ ઉપયોગમાં આવતા ક્લાસની જ ચર્ચા કરીશું. કારણ કે java.io પૅકેજમાં આવતા દરેક ક્લાસની ચર્ચા કરવી એ આ પાઠ્યપુસ્તકની મર્યાદાબહાર છે.

### કૅરેક્ટર-સ્ટ્રીમ-ક્લાસ

કૅરેક્ટર-સ્ટ્રીમ-ક્લાસ એ java.io પૅકેજમાં ઉપલબ્ધ ક્લાસનો સમૂહ છે. તેને 16-bit યુનિકોડ અક્ષરો વાંચવા અને લખવા માટે ઉપયોગમાં લઈ શકાય છે. કૅરેક્ટર-સ્ટ્રીમ-ક્લાસને વળી પાછા રીડર (Reader) અને રાઇટર (Writer) ક્લાસ એમ બે ભાગમાં વર્ગીકૃત કરી શકાય છે. રીડર-ક્લાસ (Reader class) એ ફાઇલમાંથી અક્ષરો વાંચવા માટે તૈયાર કરેલા ક્લાસનો સમૂહ છે. જ્યારે, રાઇટર-ક્લાસ (Writer class) એ ફાઇલમાં અક્ષરો લખવા માટે તૈયાર કરેલા ક્લાસનો સમૂહ છે. કૅરેક્ટર-સ્ટ્રીમ-ક્લાસનો પદાનુક્રમ આકૃતિ 11.3માં દર્શાવેલ છે.




**11.3 : કૅરેક્ટર-સ્ટ્રીમ-ક્લાસનો પદાનુક્રમ**

આકૃતિ 11.3માં દર્શાવ્યા મુજબ, java.io.Reader ક્લાસ અને java.io.Writer ક્લાસ, ઑબ્જેક્ટ ક્લાસમાંથી ઉદ્ભવ્યા છે. તે ઍબ્સ્ટ્રેક્ટ ક્લાસ છે (એવા ક્લાસ કે જે કોઈ ઑબ્જેક્ટ બનાવવા ઉપયોગમાં લઈ શકાતા નથી) અને તેના સબ-ક્લાસ દ્વારા અમલમાં મૂકવાની પદ્ધતિઓના ગણ સાથે મળે છે. InputStreamReader અને BufferedReader એ Reader Classના સબ-ક્લાસ (પેટા ક્લાસ) છે. FileReader ક્લાસ InputStreamReader ક્લાસનો સબ-ક્લાસ છે. એ જ રીતે, OutputStreamWriter, BufferedWriter અને PrintWriter એ Writer ક્લાસના સબ-ક્લાસ છે. FileWriter ક્લાસ એ OutputStreamWriter ક્લાસનો સબ-ક્લાસ છે.

હવે, આપણે ઉપર દર્શાવેલ કેટલાક ક્લાસના કન્સ્ટ્રક્ટર અને પદ્ધતિઓ સમજીએ. પદ્ધતિઓ અને કન્સ્ટ્રક્ટરનું વિગતવાર વર્ણન ઓનલાઇન http://docs.oracle.com/javase/6/docs/api/ પરથી મેળવી શકાય છે.

### Writer ક્લાસ

કૅરેક્ટર-સ્ટ્રીમ લખવા માટે રાઇટર ક્લાસ એ મૂળ આધારરૂપ ક્લાસ છે. ઍબ્સ્ટ્રેક્ટ રાઇટર-ક્લાસ ચોક્કસ કાર્યપ્રણાલી ઘોષિત કરે છે, જે તમામ કૅરેક્ટર આઉટપુટ-સ્ટ્રીમ માટે ઉપલબ્ધ બને છે. આપણે ફાઇલમાં લખવા માટેની ક્રિયા કરવા આપણા પ્રોગ્રામમાં FileWriter ક્લાસનો ઉપયોગ કરીશું.

રાઇટર-ક્લાસની પદ્ધતિ IOException થ્રો કરે. જ્યારે કોઈ I/O ક્રિયા નિષ્ફળ જાય, ત્યારે IOException બને છે. IOException એ ચકાસેલ અપવાદ છે, માટે આપણે તેની કાળજી લેવી જ પડે, અન્યથા કંપાઇલિંગ ભૂલ દર્શાવશે. કોષ્ટક 11.3 રાઇટર-ક્લાસની કેટલીક પદ્ધતિઓની યાદી રજૂ કરે છે. આ પદ્ધતિઓ તેના સબ-ક્લાસ દ્વારા ઉપયોગમાં લેવાય છે.

| પદ્ધતિ (Method) | વર્ણન (Desctiption) |
|---|---|
| void close() | સ્ટ્રીમ (stream)ને બંધ કરે છે. |
| void write(int c) | સ્ટ્રીમમાં 'c'ના પાછળના 16 બીટ લખે છે. |
| void write(String s) | સ્ટ્રીમમાં શબ્દમાળા તરીકે 's' લખે છે. |

**કોષ્ટક 11.3 : FileWriter ક્લાસની કેટલીક પદ્ધતિઓ**

OutputStreamWriter ક્લાસ રાઇટર ક્લાસને વિસ્તારે છે. તે અક્ષરોની સ્ટ્રીમને બાઇટની સ્ટ્રીમમાં પરિવર્તિત કરે છે. FileWriter ક્લાસ, OutputStreamWriterને વિસ્તારે છે અને ફાઇલમાં અક્ષરો પરિણામ રૂપે આપે છે. તેના કન્સ્ટ્રક્ટરોમાંના કેટલાક નીચે મુજબ છે :

FileWriter(String filepath) throws IOException

FileWriter(File fileobj) throws IOException

FileWriter(String filepath, boolean append) throws IOException

ઉપર્યુક્ત કન્સ્ટ્રક્ટરમાં filepath પેરામીટર એ ફાઇલનું પૂરું પથનામ છે અને fileobj એ ફાઇલક્લાસ ઑબ્જેક્ટ છે, જે ફાઇલને વર્ણવે છે. છેલ્લા કન્સ્ટ્રક્ટરમાં જો appendની કિંમત true મળશે, તો ફાઇલના છેડે અક્ષરો જોડવામાં આવશે, અન્યથા લખાણ ફાઇલના હાલના લખાણની ઉપર લખાઈ જશે. ઉદાહરણ તરીકે, FileWriterનો ઑબ્જેક્ટ આપણે નીચે મુજબ બનાવી શકીએ :

FileWriter fwobject = new FileWriter("/java/files/Charfile1.txt");

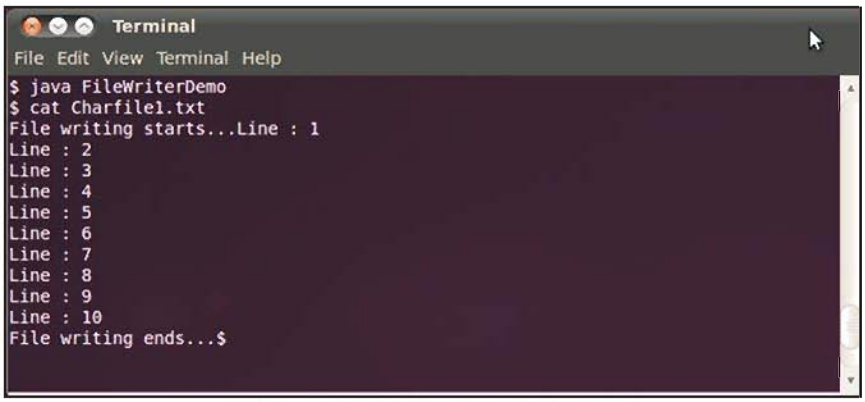
એક એવું ઉદાહરણ લઈએ જે ફાઇલમાં કેવી રીતે લખવું તે વર્ણવે. આપણે એવું અનુમાન કરીશું કે, ફાઇલો હાલની ડિરેક્ટરીમાં સચવાય છે. કોડલિસ્ટિંગ 11.2 એ દર્શાવે છે કે, "Charfile1.txt" ફાઇલ કેવી રીતે બનાવવી જે હયાત જ નથી. તદ્ઉપરાંત, આપણે write() પદ્ધતિનો ઉપયોગ કરીને તે ફાઇલમાં કેટલીક માહિતી લખીશું. આ પ્રોગ્રામમાં ઉપયોગમાં લેવાતી write() અને close() જેવી પદ્ધતિ રાઇટર ક્લાસમાંથી વારસામાં મેળવાયેલ છે. આ કોડલિસ્ટિંગનું પરિણામ આકૃતિ 11.4માં દર્શાવેલ છે. પ્રોગ્રામનો અમલ કર્યા બાદ, આપણે નવી બનાવેલી "Charfile1.txt" ફાઇલની અંદર પડેલ લખાણ દર્શાવવા cat કમાન્ડનો ઉપયોગ કરીશું.

```
// Write to a file using character stream
import java.io.*;
class FileWriterDemo
{
        public static void main(String args[])
        {
                FileWriter fwobject = null;
                try  {

                        // Create an object of FileWriter
                        fwobject = new FileWriter("Charfile1.txt");

                        //Write strings to the file
                        fwobject.write("File writing starts...");

                        for(int i = 1; i < 11 ; i++)
                                fwobject.write("Line : " + i + "\n");

                        fwobject.write("File writing ends...");
                }
                catch(Exception eobj)
                {
                        System.out.println(eobj);
```

```
                }
                finally
                {
                        try {
                        // Close the filewriter
                        fwobject.close();
                        }
                        catch(Exception eobj)
                        {
                                System.out.println(eobj);
                        }
                }
        }
}
```

**કોડલિસ્ટિંગ 11.2 : ફાઇલ લખવાની ક્રિયા વર્ણવતો પ્રોગ્રામ**

ફાઇલમાં લખવાની પ્રક્રિયા પૂર્ણ થયા પછી સ્ટ્રીમ ઓબ્જેક્ટ બંધ કરાય તે અગત્યનું છે. ખુલ્લી રહેલી ફાઇલ કમ્પ્યૂટર સિસ્ટમનાં સંસાધનો વાપરે છે અને ફાઇલની રીત મુજબ અન્ય પ્રોગ્રામ તેનો ઉપયોગ પણ ન કરી શકે તેવું બને. જેવી ક્રિયા પૂરી થાય કે તરત જ ફાઇલ બંધ કરવી જરૂરી છે.




**11.4 : કોડલિસ્ટિંગ 11.2નું પરિણામ**

## રીડર ક્લાસ (Reader Classes)

કૅરેક્ટર-સ્ટ્રીમ વાંચવા માટે રીડર ક્લાસ એ આધારરૂપ ક્લાસ છે. એબસ્ટ્રેક્ટ રીડર ક્લાસ ચોક્કસ કાર્યપ્રણાલી ઘોષિત કરે છે, જે તમામ કૅરેક્ટર ઇનપુટ-સ્ટ્રીમ માટે ઉપલબ્ધ બને છે. આપણે ફાઇલમાંથી વાંચવાની ક્રિયા કરવા માટે આપણા પ્રોગ્રામમાં FileReader ક્લાસનો ઉપયોગ કરીશું. કોષ્ટક 11.4 રીડર-ક્લાસની કેટલીક પદ્ધતિઓની યાદી રજૂ કરે છે, જે પદ્ધતિઓ તેના સબ-ક્લાસ દ્વારા ઉપયોગમાં લેવાય છે.

| પદ્ધતિ (Method) | વર્ણન (Description) |
|---|---|
| void close() | streamને બંધ કરે છે. |
| int read() | સ્ટ્રીમમાંથી પછીનો ઉપલબ્ધ અક્ષર વાંચે છે. સ્ટ્રીમનો અંત દર્શાવવા તે "-1" પરત કરે છે. |

**કોષ્ટક 11.4 : FileReader ક્લાસની થોડીક પદ્ધતિઓ**

InputStreamReader ક્લાસ રીડર ક્લાસને વિસ્તારે છે. તે બાઇટની સ્ટ્રીમને અક્ષરોની સ્ટ્રીમમાં પરિવર્તિત કરે છે. FileReaderClass, InputStreamReaderને વિસ્તારે છે અને ફાઇલમાંથી અક્ષરો વાંચે છે. તેના કન્સ્ટ્રક્ટરોમાંનાં કેટલાંક નીચે મુજબ છે :

FileReader(String filepath) throws FileNotFoundException

FileReader(File fileobj) throws FileNotFoundException

FileReaderનો એક ઑબ્જેક્ટ આપણે નીચે મુજબ બનાવી શકીએ :

FileReader frobject = new FileReader("/java/files/Charfile1.txt");

કોડલિસ્ટિંગ 11.2માં દર્શાવેલ પ્રોગ્રામમાં આપણે "Charfile1.txt" ફાઇલમાં લખવા પ્રયત્ન કર્યો. હવે ચાલો આપણે એક એવો પ્રોગ્રામ લખીએ, જે આપણે બનાવેલી ફાઇલમાંથી માહિતી વાંચતો હોય. કોડલિસ્ટિંગ 11.3માં દર્શાવેલ પ્રોગ્રામ FileReader ક્લાસની read() પદ્ધતિનો ઉપયોગ કરી ફાઇલમાંથી ડેટા વાંચે છે. આ read પદ્ધતિ એ Reader ક્લાસમાંથી વારસામાં મેળવાયેલ છે.

```
// Reading from a file using character stream
import java.io.*;
class FileReaderDemo
{
        public static void main(String args[])
        {
                FileReader frobject = null;
                try  {

                        // Create an object of FileReader
                        frobject = new FileReader("Charfile1.txt");
                        int i;
                        char ch;
                        while ( ( i = frobject.read()) != -1)
                        {
                                ch = (char) i ;
                                System.out.print(ch);
                        }
                }
                catch(Exception eobj)
                {
                        System.out.println(eobj);
                }
                finally
                {
```

```
                    try {
                    // Close the filewriter
                    frobject.close();
                }
                catch(Exception eobj)
                {
                    System.out.println(eobj);
                }
            }
        }
    }
```

**કોડલિસ્ટિંગ 11.3 : ફાઇલમાંથી વાંચવાની ક્રિયા વર્ણવતો પ્રોગ્રામ**

કોડલિસ્ટિંગ 11.3નું પરિણામ આકૃતિ 11.5માં દર્શાવેલ છે. અહીં એ જણાય છે કે, પ્રોગ્રામનો અમલ સ્ક્રીન ઉપર પરિણામ દર્શાવે છે. તે "Charfile1.txt" ફાઇલના તમામ લખાણને દર્શાવે છે.


**11.5 : કોડલિસ્ટિંગ 11.3નું પરિણામ**

ફાઇલનો અંત (End of File - EOF) દર્શાવવા એક ખાસ ચિહ્ન છે. ફાઇલમાંથી વાંચતી વખતે, ફાઇલનો અંત આવી ગયો છે, એવી પ્રોગ્રામને ખબર પડવી જ જોઈએ. જોકે, પ્રોગ્રામ ઇનપુટ-સ્ટ્રીમમાંથી વાંચે છે, માટે java read() પદ્ધતિ સ્ટ્રીમમાં ડેટાનો અંત દર્શાવવા "-1" પરત કરે છે.

### બાઇટ-સ્ટ્રીમ-ક્લાસ (Byte Stream Classes)

અત્યાર સુધી આપણે એ જોયું કે, java.io પૅકેજનો ઉપયોગ કરીને અક્ષરો પર કેવી રીતે પ્રક્રિયા કરી શકાય. મોટા ભાગના જીવંત ઉપયોગોમાં આંકડાકીય ગણતરીઓની જરૂર પડે છે. જેમકે, ફાઇલમાં માલસામાન (ઇન્વેન્ટરી)ની વિગતો સાચવવી અને પડતરકિંમતની ગણતરી કરવી અથવા ફાઇલમાં કર્મચારીઓની માહિતી સાચવવી અને તેમના પગારની ગણતરી કરવી. આવા પ્રકારની ક્રિયાઓ માટે જાવા બાઇનરી સ્ટ્રીમ (binary stream) અને તેમના સહયોગી ક્લાસ આપે છે.

java.io પૅકેજમાં FileInputStream અને FileOutputStream ક્લાસ, આપણને ડિસ્કમાં કોઈ પણ ફાઇલમાં બાઇટ લખવા માટે કે ફાઇલમાંથી બાઇટ વાંચવા માટે સુવિધા આપે છે. આ ક્લાસ InputStream અને OutputSream ક્લાસના સબ-ક્લાસ છે.

## FileOutputStream

FileOutputStream એ OutputStreamનો સબ-ક્લાસ અને કોઈ આઉટપુટ-સ્ટ્રીમ અથવા ફાઇલમાં બાઇટ લખવા માટે ઉપયોગમાં લેવાય છે. આ ક્લાસનો ઉપયોગ કરવા માટે સૌપ્રથમ આપણે ફાઇલ ઑબ્જેક્ટ બનાવવો પડે એ પછી, ફાઇલમાં બાઇટ લખવા OutputStream એબ્સ્ટ્રેક્ટ ક્લાસમાંથી લાવેલ write પદ્ધતિનો ઉપયોગ કરી શકીએ. FileOutputStream ક્લાસની વધુ ઉપયોગમાં લેવાતી કેટલીક પદ્ધતિઓ કોષ્ટક 11.5માં દર્શાવેલ છે.

| પદ્ધતિ (Method) | વર્ણન (Description) |
|---|---|
| void close() | ફાઇલ આઉટપુટ-સ્ટ્રીમને બંધ કરે છે અને સ્ટ્રીમ સાથે સંકળાયેલ કોઈ પણ સિસ્ટમ સંસાધનોને મુક્ત કરે છે. |
| void write(int b) | આ આઉટપુટ-સ્ટ્રીમમાં નિર્દિષ્ટ બાઇટ લખે છે. |
| void write(byte[] b) | આ ફાઇલ આઉટપુટ-સ્ટ્રીમમાં, નિર્દિષ્ટ બાઇટ એરેમાંથી b.length બાઇટ લખે છે. |

**કોષ્ટક 11.5 : FileOutputStream ક્લાસની થોડીક પદ્ધતિઓ**

FileOutputStreamનું કન્સ્ટ્રક્ટર ફાઇલના સ્થાનનો પથ દર્શાવતી શબ્દમાળા (string) અથવા ફાઇલ ક્લાસનો ઑબ્જેક્ટ જ સ્વીકારી શકે છે. ચાલો, સામાન્ય રીતે ઉપયોગમાં લેવાતાં આવા કેટલાક કન્સ્ટ્રક્ટર જોઈએ :

FileOutputStream(String name) throws FileNotFoundException

અથવા

FileOutputStream(File file) throws FileNotFoundException

FileOutputStream નાં દૃષ્ટાંત બનાવવાનાં ઉદાહરણ નીચે દર્શાવ્યા મુજબ છે :

```
FileOutputStream fosobject = new FileOutputStream("/home/Akash/myfile.txt");
```

અન્ય રીતે પણ આપણે ઉપયોગ કરી શકીએ :

```
File fobj = new File("/home/Akash/myfile.txt");
FileOutputStream fosobject = new FileOutputStream(fobj);
```

જો ફાઇલનું નામ એ કોઈ ફાઇલને બદલે કોઈ ડિરેક્ટરીનો નિર્દેશ કરતું હોય કે ફાઇલ હયાત જ ન હોય, કે કોઈ કારણસર ફાઇલ ખોલી શકાય તેમ ન હોય, તો ઉપર દર્શાવેલ કન્સ્ટ્રક્ટર FileNotFoundException થ્રો કરશે.

## FileInputStream

FileInputStream એ InputStreamનો સબ-ક્લાસ છે અને ફાઇલમાંથી બાઇટ વાંચવા માટે ઉપયોગમાં લેવાય છે. કોષ્ટક 11.6માં દર્શાવ્યા પ્રમાણે તે ફાઇલોમાંથી વાંચવાની ક્રિયા કરવા માટે કેટલીક પદ્ધતિના ગણ પૂરા પાડે છે.

| પદ્ધતિ (Method) | વર્ણન (Description) |
|---|---|
| void close() | ફાઇલ ઇનપુટ-સ્ટ્રીમને બંધ કરે છે અને સ્ટ્રીમ સાથે સંકળાયેલ કોઈ પણ સિસ્ટમ સંસાધનોને મુક્ત કરે છે. |
| int read() | આ ઇનપુટ-સ્ટ્રીમમાંથી ડેટાબાઇટ વાંચે છે. |
| int read(byte[] b) | આ ફાઇલ ઇનપુટ-સ્ટ્રીમમાં, નિર્દિષ્ટ બાઇટ એરેમાંથી b.length વાંચે છે. |

**કોષ્ટક 11.6 : FileInputStream ક્લાસની થોડીક પદ્ધતિઓ**

આપણે એક એવો પ્રોગ્રામ જોઈએ કે જે કોડલિસ્ટિંગ 11.4માં દર્શાવ્યા પ્રમાણે ફાઇલમાં લખવા અને વાંચવાની ક્રિયા કરવા માટે ઉપર દર્શાવેલ ક્લાસનો ઉપયોગ કરતો હોય.

```
//Program to read and write bytes to binary file
import java.io.*;
class BinaryFileDemo {
        public static void main(String args[])
        {
                FileOutputStream outobject = null;
                FileInputStream inobject = null;
                String cities = " Rajkot \n Ahmedabad \n Vadodara \n Vapi \n";

                //Convert cities into byte array
                byte citiesarray[] = cities.getBytes();

                try {
                        // Create object of Binary output stream
                        outobject = new FileOutputStream("Binaryfile.dat");
                        //Write the array of bytes into file
                        outobject.write(citiesarray);
                        outobject.close();

                        //Create object of Binary input stream
                        inobject = new FileInputStream("Binaryfile.dat");
                        //Variable to read each byte
                        int i;
                        //Read each byte from the file and display
                        while((i = inobject.read())!=-1)
                        {
                                System.out.print((char)i);
                        }
                        inobject.close();

                }
                catch(Exception eobj)
                {
                        System.out.println(eobj);
                }
        }
}
```

કોડલિસ્ટિંગ 11.4 : બાઇનરી ફાઇલમાં બાઇટ લખવા અને વાંચવા માટેનો પ્રોગ્રામ

કોડલિસ્ટિંગ 11.4માં દર્શાવેલ પ્રોગ્રામ "Binaryfile.dat" ફાઇલમાં એક શબ્દમાળા (સ્ટ્રીંગ) લખે છે, પછીથી તે એ જ ફાઇલમાંથી ડેટા વાંચવા અને સ્ક્રીન પર દર્શાવવા FileInputStreamનો એક ઓબ્જેક્ટ બનાવે છે. આપણે એ નોંધવું જ જોઈએ કે, વાંચવાની ક્રિયા પૂર્ણ થયા પછી પૂર્ણાંક સંખ્યાને અક્ષરમાં પરિવર્તિત કરવા typecastingનો અમલ કરવામાં આવ્યો છે.

**નોંધ :** ટેક્સ્ટફાઇલ અને બાઇનરી ફાઇલ વચ્ચેનો તફાવત નિહાળવા માટે, કોડલિસ્ટિંગ 11.2 અને 11.4માં આપેલ પ્રોગ્રામના ઉપયોગ દ્વારા તૈયાર કરેલ ફાઇલો ખોલો.

## કી-બોર્ડ પરથી મળતા ઇનપુટની પ્રક્રિયા

આ વિભાગમાં, આપણે જાવાપ્રોગ્રામને કી-બોર્ડ દ્વારા ડેટા ઇનપુટ કરવા માટેના વિવિધ રસ્તા જાણીશું. આપણે જાણીએ છીએ તે પ્રમાણે, કોઈ પણ પ્રોગ્રામ, કી-બોર્ડ/GUI મારફત જીવંત સંપર્ક દ્વારા અથવા 'કમાન્ડલાઇન આર્ગ્યુમેન્ટ' (કમ્પ્યૂટરને અપાતા આદેશની સાથે જ ટાઇપ કરાતો ડેટા) અથવા ફાઇલમાંથી જરૂરી ડેટા ઇનપુટ મેળવી શકે. જોકે, એવા ઘણા ક્લાસ છે, જે કી-બોર્ડ પરથી ઇનપુટ મેળવવાની સવલત કરી આપી શકે. અહીં આપણે સૌથી વધુ ઉપયોગમાં લેવાતી ટેક્નિક પૈકીની બે ટેક્નિકની ચર્ચા કરીશું, જેમાંની એક છે, java.util પૅકેજનો સ્કૅનર ક્લાસ (scanner class) અને બીજો છે java.io પૅકેજનો કૉન્સોલ ક્લાસ (console class).

## સ્કૅનરક્લાસ

સ્કૅનરક્લાસ, java.util પૅકેજનો એક ભાગ છે; તે કી-બોર્ડ કે ફાઇલ પરથી ઇનપુટ મેળવવા માટે વિવિધ પદ્ધતિઓ પ્રદાન કરે છે. આ ક્લાસની ખાસ વિશેષતા એ છે કે, તે એક ચિહ્ન (delimiter)નો ઉપયોગ કરીને તે ઇનપુટ કરાતી શબ્દમાળાને ટોકન (શબ્દો)માં વિભાજિત કરે છે. (ખાલી જગ્યા (white space) એ સામાન્ય રીતે ઉપયોગમાં લેવાતું ચિહ્ન છે.) દરેક ટોકન જુદા-જુદા પ્રકારનાં હોઈ શકે છે. ઉદાહરણ તરીકે, "India-1947" જેવી શબ્દમાળા "String-int" કિંમત તરીકે વાંચી શકાય. હવે આપણે કન્સ્ટ્રક્ટર અને સ્કૅનરક્લાસની કેટલીક પદ્ધતિઓ વિશે જાણીએ :

```
Scanner(String str)

Scanner(InputStream isobject)

Scanner(File fobject) throws FileNotFoundException
```

સ્કૅનર-ઓબ્જેક્ટ એક શબ્દમાળા, ફાઇલ-ઓબ્જેક્ટ અથવા InputStream ઓબ્જેક્ટમાંથી બનાવી શકાય છે, ઉદાહરણ તરીકે ફાઇલ અને કી-બોર્ડ પરથી વાંચવા માટે આપણે નીચેની રીતે કન્સ્ટ્રક્ટરનો ઉપયોગ કરી શકીએ.

```
Scanner fileinput = new Scanner(new File("Students.dat"));

Scanner kbinput = new Scanner(System.in);
```

સ્કૅનરક્લાસ સાથે ઉપલબ્ધ કેટલીક અગત્યની પદ્ધતિઓની યાદી કોષ્ટક 11.7માં દર્શાવેલ છે.

| પદ્ધતિ (Method) | વર્ણન (Descrption) |
|---|---|
| void close() | સ્કૅનરને બંધ કરે છે. |
| String next() | પછીનો ટોકન પરત કરે છે. |
| boolean hasNext() | જો ઇનપુટમાં ટોકન હશે, તો true કિંમત પરત કરે છે. |
| int nextInt() | ઇનપુટની પછીના ટોકનને Int તરીકે સ્કૅન કરે છે. |
| float nextFloat() | ઇનપુટની પછીના ટોકનને Float તરીકે સ્કૅન કરે છે. |
| String nextLine() | ઇનપુટની પછીના ટોકનને Line તરીકે સ્કૅન કરે છે. |

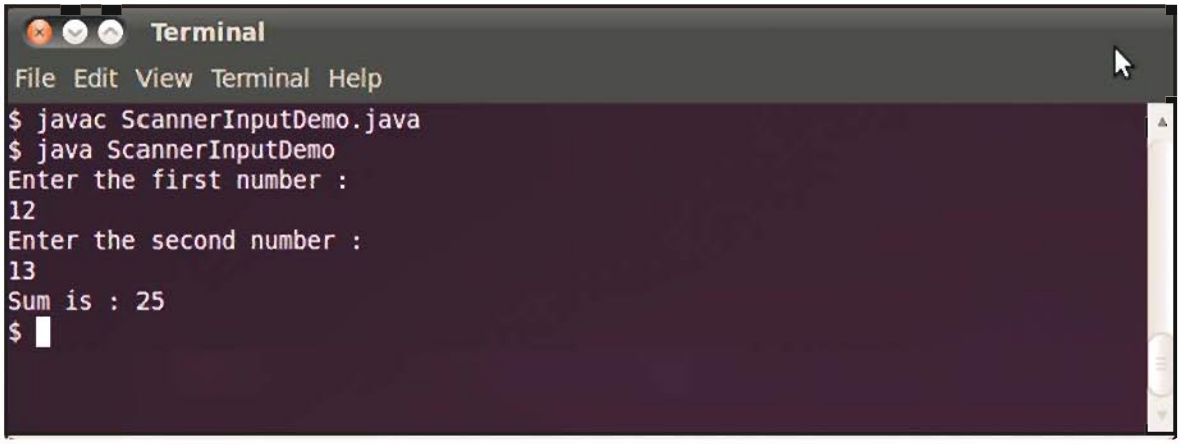
**કોષ્ટક 11.7 : Scanner ક્લાસની પદ્ધતિઓ**

આપણે હવે એક એવો પ્રોગ્રામ જોઈએ કે જે ઉપયોગકર્તા પાસેથી પરસ્પર સંવાદ દ્વારા બે સંખ્યા વાંચે અને તે બે સંખ્યાનો સરવાળો દર્શાવે. કોડલિસ્ટિંગ 11.5 આ પ્રોગ્રામ દર્શાવે છે.

```
//Accepts input at command prompt
import java.io.*;
import java.util.*;
class ScannerInputDemo
{
	public static void main(String args[])
	{
		Scanner kbinput = null;
		int number1;
		int number2;
		int sum=0;
		try
		{
			// Create an object of Scanner class
			// that reads from Standard Input
			kbinput = new Scanner(System.in);
			System.out.println("Enter the first number : ");
			//Read the integer number from console
			number1 = kbinput.nextInt();
			System.out.println("Enter the second number : ");
			//Read the integer number from console
			number2 = kbinput.nextInt();
			sum = number1 + number2;
			System.out.println("Sum is : " + sum);
		}
		catch(Exception eobj)
		{
			System.out.println(eobj);
		}
		finally {
			try {
				kbinput.close();
			}
			catch(Exception eobj)
			{
				System.out.println(eobj);
			}
		}

	}
}
```

કોડલિસ્ટિંગ 11.5 : બે સંખ્યા ઉમેરવાનો પ્રોગ્રામ

કોડલિસ્ટિંગ 11.5માં આપણે સ્કેનરક્લાસનો એક ઑબ્જેક્ટ બનાવ્યો. સ્કેનર ક્લાસનો કન્સ્ટ્રક્ટર આર્ગ્યુમેન્ટ તરીકે "System.in" સ્વીકારે છે, જેથી કરીને તે સ્ટાન્ડર્ડ ઇનપુટ (કી-બોર્ડ) પરથી વાંચી શકે. આ બંને સંખ્યાઓ, સ્કેનરક્લાસની nextInt() પદ્ધતિનો ઉપયોગ કરીને પૂર્ણાંક સંખ્યા તરીકે લેવાશે. આકૃતિ 11.6 પ્રોગ્રામનું પરિણામ દર્શાવે છે.




**11.6 : કોડલિસ્ટિંગ 11.5નું પરિણામ**

ફાઇલમાંથી વાંચવા માટે પણ સ્કેનરક્લાસનો ઉપયોગ થઈ શકે છે. આપણે એક એવું ઉદાહરણ જોઈએ, જેમાં થોડા વિદ્યાર્થીઓ વિશે માહિતી ધરાવતી ફાઇલમાંથી ડેટા વાંચવા માટે આપણે સ્કેનરક્લાસનો ઉપયોગ કરતા હોઈએ. એવું ધારી લઈએ છીએ કે "Students.dat" ફાઇલ હયાત છે જ. તેમાં 5 ફિલ્ડ છે : student_no, student_name, ત્રણ વિષયના ગુણની માહિતી ધરાવે છે. Students.dat ફાઇલનું માળખું અને ડેટા નીચે દર્શાવ્યા મુજબ છે :

1 Akash 45 65 55

2 Badal 10 20 30

3 Zakir 45 40 60

4 David 65 50 75

દરેક વિદ્યાર્થીનો ડેટા વાંચવા અને કુલ ગુણની ગણતરી કરીને તેને પરિણામ રૂપે રજૂ કરવા કોડલિસ્ટિંગ 11.6માં દર્શાવ્યા મુજબ આપણે એક પ્રોગ્રામ લખીશું.

```
//Accepts input from a file "Students.dat" and calculate the total marks of each student
import java.io.*;
import java.util.*;
class ScannerFileDemo
{
        public static void main(String args[])
        {
                Scanner fileinput = null;
                int rollno, mark1, mark2, mark3, totalmarks;
                String name = null;
                File fobject;
        try
        {
```

```
                    // Specify the file from where data is to be read
                    fobject = new File("Students.dat");
                    // Create an object of Scanner class that reads from File
                    fileinput = new Scanner(fobject);
                    //Display the default Delimiter to separate fields within a file
                    System.out.println("Default delimeter is : " + fileinput.delimiter() + "\n");
                    // Iterate to read the values of each record
                    while(fileinput.hasNext()) {
                    rollno = fileinput.nextInt();
                    name = fileinput.next();
                    mark1=fileinput.nextInt();
                    mark2=fileinput.nextInt();
                    mark3=fileinput.nextInt();
                    totalmarks = mark1 + mark2 + mark3;
                    System.out.println("Total marks of  Rollno "  + rollno + " ," +name+ " are
                                                                    : "  +  totalmarks);
                    }
                    fileinput.close();
            }
            catch(Exception eobj)
            {
                    System.out.println(eobj);
            }
    }
    }
```

કોડલિસ્ટિંગ 11.6 : પ્રત્યેક વિદ્યાર્થીના કુલ ગુણની ગણતરી કરવાનો પ્રોગ્રામ

પ્રોગ્રામનું પરિણામ આકૃતિ 11.7માં દર્શાવ્યા મુજબ મોનિટર પર દર્શાવાશે.


11.7 : કોડલિસ્ટિંગ 11.6નું પરિણામ

### કૉન્સોલ ક્લાસ

અગાઉના ઉદાહરણમાં આપણે ઉપયોગકર્તાના ઇનપુટને વાંચવા સ્કેનર ક્લાસનો ઉપયોગ કર્યો હતો. સ્કેનર ક્લાસ ઉપરાંત java.io.Console નામનો એક અન્ય ક્લાસ પણ છે, જે કી-બોર્ડ પરથી ઇનપુટ મેળવવા ઉપયોગમાં લેવાય છે. આ ક્લાસનો ઉપયોગ ખાસ કરીને જ્યારે ઇનપુટને છૂપા સ્વરૂપે (ટાઇપ કરાતા અક્ષરો સ્ક્રીન પર ન દર્શાવાય તે રીતે) મેળવવાનું હોય, ત્યારે કરીએ છીએ.

કૉન્સોલ ક્લાસ ગુપ્ત સંકેત (પાસવર્ડ) વાંચવા માટેની એક પદ્ધતિ પૂરી પાડે છે. જ્યારે પાસવર્ડ વાંચવામાં આવે છે, ત્યારે ઉપયોગકર્તા દ્વારા મૉનિટર (સ્ક્રીન) પર ટાઇપ કરવામાં આવતા ઇનપુટને ગુપ્ત રખાય છે અથવા કૉન્સોલના સ્ક્રીન પર દર્શાવાતા નથી અને તે પરત મૂલ્ય તરીકે અક્ષરોનો એરે પરત કરશે. કૉન્સોલ ક્લાસ java.io પૅકેજનો એક ભાગ છે. બહોળા પ્રમાણમાં ઉપયોગમાં લેવાતી કૉન્સોલ ક્લાસની કેટલીક પદ્ધતિ કોષ્ટક 11.8માં દર્શાવેલ છે :

| પદ્ધતિ (Methods) | વર્ણન (Description) |
|---|---|
| String readLine() | આ પદ્ધતિ કૉન્સોલ પરથી લખાણની એક જ લાઇન વાંચે છે. |
| char[] readPassword() | આ પદ્ધતિ કૉન્સોલ પરથી ઇકો (echoing)ની અસરમુક્ત અવસ્થામાં પાસવર્ડ અથવા પાસફ્રેઝ (passphrase) વાંચે છે. |
| Console printf(String format, Object args) | આ પદ્ધતિ કૉન્સોલના પરિણામની સ્ટ્રીમમાં ચોક્કસ સ્વરૂપ સાથેની શબ્દમાળા લખવા માટે ઉપયોગમાં લેવાય છે. |

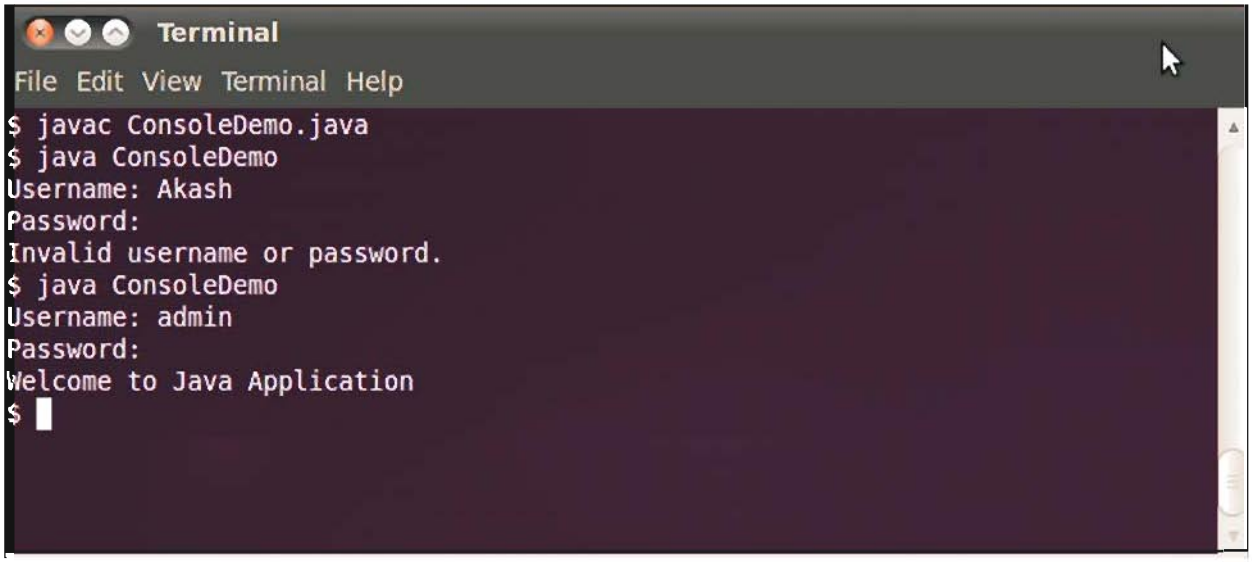
કોષ્ટક 11.8 : Console ક્લાસની પદ્ધતિઓ

કોડલિસ્ટિંગ 11.7માં દર્શાવેલ પ્રોગ્રામ, યૂઝરનેમ (username) અને પાસવર્ડ (password) વાંચવા કૉન્સોલક્લાસનો ઉપયોગ રજૂ કરે છે. વધુમાં, તે એ બાબતની ખાતરી પણ કરી લે છે કે, યૂઝરનેમ અને પાસવર્ડ સાચો છે કે નહીં.

```
// Program to reading passwords
import java.io.Console;
import java.util.Arrays;
public class ConsoleDemo {

    public static void main(String[] args) {
     Console console = System.console();
    String username = console.readLine("Username: ");
     char[] password = console.readPassword("Password: ");

   if (username.equals("admin") &&  String.valueOf(password).equals("secret")) {
      console.printf("Welcome to Java Application \n");
     } else {
     console.printf("Invalid username or password.\n");
     }
   }
}
```

કોડલિસ્ટિંગ 11.7 : યૂઝરનેમ અને પાસવર્ડ વાંચતો પ્રોગ્રામ

આકૃતિ 11.8માં પ્રોગ્રામનું પરિણામ દર્શાવાયું છે. યૂઝરનેમ અને પાસવર્ડ ઈનપુટ કરવા આપણે બે પ્રયત્નોનો ઉપયોગ કર્યો છે. પ્રથમ પ્રયત્નમાં યૂઝરનેમ ખોટું હતું અને તેથી આપણે invalid username or password સંદેશનો ઉપયોગ કર્યો. બીજા પ્રયત્નમાં, આપણે સાચો યૂઝરનેમ અને પાસવર્ડ દાખલ કર્યો.




**11.8 : કોડલિસ્ટિંગ 11.7નું પરિણામ**

અનેક ઉપયોગકર્તા પદ્ધતિના કિસ્સામાં ઉપયોગકર્તાની યાદી વાંચવા અને ફાઇલમાંથી તેમના પાસવર્ડ સરખાવવા માટે પ્રોગ્રામને વિસ્તારી શકાય.

આ પ્રકરણમાં ચર્ચેલા વિવિધ ક્લાસ ઉપરાંત, જાવાફાઇલનો ઉપયોગ કરી ઓબ્જેક્ટને સાચવવા અને પુનઃપ્રાપ્ત (retrieve) કરવા માટેના ક્લાસ પૂરા પાડે છે. તે ફાઇલનો યાદચ્છિક રીતે ઉપયોગ કરવા માટેના ક્લાસ અને પદ્ધતિઓ પણ પૂરી પાડે છે. અત્યાર સુધી, આપણે એવી પદ્ધતિઓ જોઈ કે જે ક્રમિક (sequential) ક્રિયાઓ પાર પાડી શકે. જોકે, તેમ છતાં એવા ક્લાસ પણ છે જે ક્રમિક રીતે ઉપયોગ કરવાને બદલે આપણને સીધા n-ક્રમ પરના રેકર્ડ પર જવાની સુવિધા આપે. આ બધા ક્લાસની ચર્ચા કરવી આ પાઠ્યપુસ્તકની મર્યાદાબહારનું હોઈ java.io પૅકેજની અન્ય વિવિધ રસપ્રદ લાક્ષણિકતાઓનો ખ્યાલ મેળવવાનું અમે વિદ્યાર્થીઓ પર છોડીએ છીએ.

## સારાંશ

આ પ્રકરણમાં આપણે ફાઇલ સંબંધી ક્રિયાઓ વિશે શીખ્યા. ફાઇલ-ક્રિયાઓ કરવા માટે java.io.File ક્લાસનો કેવી રીતે ઉપયોગ કરવો તે આપણે જોયું. આપણે સ્ટ્રીમનો ખ્યાલ મેળવ્યો અને વિવિધ પ્રકારની ઈનપુટ અને આઉટપુટ-સ્ટ્રીમનો કેવી રીતે ઉપયોગ કરવો તે આપણે શીખ્યા. ફાઇલ કે કી-બોર્ડ પરથી ડેટાનો ઉપયોગ કરવા માટે સ્કૅનર-ક્લાસનો ઉપયોગ કેવી રીતે કરવો તે પણ આપણે શીખ્યા. અંતમાં, કી-બોર્ડ પરથી ડેટા ઈનપુટ કરવા માટે કૉન્સોલ-ક્લાસનો ઉપયોગ કેવી રીતે કરવો તે આપણે શીખ્યા.

## સ્વાધ્યાય

1. જાવાપ્રોગ્રામિંગમાં ફાઇલ શા માટે અગત્યની છે ? કેવા સંજોગો હેઠળ તમે ડેટાને ફાઇલમાં સાચવશો ?
2. ફાઇલ અને ડિરેક્ટરી પર કરી શકાતી વિવિધ ક્રિયાઓ જણાવો.
3. જાવામાં શા માટે સ્ટ્રીમ (Stream)નો ખ્યાલ રજૂ કરવામાં આવ્યો છે ? સ્ટ્રીમનો ઉપયોગ કરવાના ફાયદા જણાવો.

**4. નીચેના દરેક પ્રશ્નો માટે આપેલા વિકલ્પો પૈકી યોગ્ય વિકલ્પ પસંદ કરી ઉત્તર આપો :**

(1) નીચેનાંમાંથી ક્યું વિધાન સાચું છે ?

(a) નાશવંત પ્રકારની સંગ્રહવ્યવસ્થા થોડીક સેકન્ડમાં નાશ પામે છે.

(b) નાશવંત પ્રકારનો સંગ્રહ જ્યારે કમ્પ્યૂટર બંધ કરવામાં આવે, ત્યારે નાશ પામે છે.

(c) કમ્પ્યૂટરડિસ્ક એ નાશવંત પ્રકારનું સંગ્રહસાધન છે.

(d) ઉપરનાં તમામ

(2) નીચેનામાંથી ક્યું કમ્પ્યૂટરસિસ્ટમમાં નાશવંત સાધન પર સાચવેલ ડેટાના સમૂહનો નિર્દેશ કરે છે ?

(a) ફાઇલ (b) વિનિયોગ (c) નાશવંત ડેટા (d) હાર્ડડિસ્ક

(3) નાનાથી મોટા ડેટા મુજબ ડેટા પદાનુક્રમ નીચેનામાંથી ક્યા ક્રમમાં આવે ?

(a) file:character:field:record (b) file:character:record:field

(c) character:field:file:record (d) character:field:record:file

(4) સ્ટ્રીમ બાબતે નીચેનામાંથી ક્યું વિધાન સાચું છે ?

(a) સ્ટ્રીમ હંમેશાં બે દિશામાં વહે છે.

(b) સ્ટ્રીમ એ માર્ગ (channel) છે, જેમાંથી ડેટા વહે છે.

(c) પ્રોગ્રામમાં કોઈ પણ એક સમયે ફક્ત એક જ સ્ટ્રીમ (પ્રવાહ) ખુલ્લી હોઈ શકે.

(d) ઉપરનાં તમામ

(5) રેકોર્ડના ફિલ્ડની વચ્ચે નીચેનામાંથી ક્યું ચિહ્ન બે ફિલ્ડને જુદા પાડવા ઉપયોગમાં લેવાય છે ?

(a) પથ (b) રેલિમીટર (c) ચલ (d) ખાલી જગ્યા

(6) નીચેનામાંથી કઈ ક્રિયાઓ પાર પાડવા માટે સ્કેનરક્લાસનો ઉપયોગ કરી શકાય ?

(a) કી-બોર્ડ પરથી ઇનપુટ સ્વીકારવા (b) ફાઇલમાંથી વાંચવા

(c) રેલિમીટર વડે જુદા પાડતી શબ્દમાળા વર્ણવવી (d) ઉપરનાં તમામ

## પ્રાયોગિક સ્વાધ્યાય

**1.** આપેલી ડિરેક્ટરીમાં પડેલી ".txt" અનુલંબન ધરાવતી તમામ ફાઇલોની યાદી દર્શાવવા જાવાપ્રોગ્રામ લખો.

**2.** એક એવો જાવાપ્રોગ્રામ લખો, જે ફાઇલનું નામ સ્વીકારે અને ચકાસણી કરે કે આપેલ ફાઇલ અસ્તિત્વમાં છે કે નહીં ? જો તે ફાઇલ હયાત હોય, તો તે ફાઇલની લાક્ષણિકતાઓ દર્શાવો.

**3.** "friends.dat" નામની ફાઇલ બનાવવા માટે જાવાપ્રોગ્રામ લખો; writer ક્લાસનો ઉપયોગ કરી આ ફાઇલમાં તમારા મિત્રોનાં નામ લખો.

4. એક એવો જાવાપ્રોગ્રામ લખો, જેમાં ઉપયોગકર્તા ફાઇલનું નામ દાખલ કરે, એ પછી, એ ફાઇલની અન્ય ફાઇલમાં નકલ કરે.

5. /etc/passwd ની ફાઇલને તમારી ડિરેક્ટરીમાં mypasswd.dat નામ વડે નકલ કરવા જાવાપ્રોગ્રામ લખો.

6. આપેલ ફાઇલમાં અક્ષરોની સંખ્યા ગણવા માટેનો જાવાપ્રોગ્રામ લખો.

7. ફાઇલનું કદ બાઇટના માપથી દર્શાવવા જાવાપ્રોગ્રામ લખો.

8. જે ઉપયોગકર્તા પાસેથી મુદ્દલની રકમ, વ્યાજનો દર અને મુદતના વર્ષની વિગત ઇનપુટ મેળવીને સાદું વ્યાજ ગણી આપતો જાવાપ્રોગ્રામ લખો.

9. એક એવો પારસ્પરિક સંવાદ કરતો (interactive) જાવાપ્રોગ્રામ લખો. જે કૉન્સોલ પર ડેટા દાખલ કરી આપેલા ઇંચની કિંમતને સેન્ટિમીટરમાં અને સેન્ટિમીટરની કિંમતને ઇંચમાં બદલી કરી આપે.

# 12 લેટેક્સની મદદથી દસ્તાવેજનું પ્રકાશન

OpenOffice.orgના રાઇટરનો ઉપયોગ કરીને દસ્તાવેજ બનાવતા આપણે શીખી ગયા. આ પ્રકરણમાં આપણે ટેક્સ અને લેટેક્સ ટાઇપસેટિંગ સૉફ્ટવેર વિશે ચર્ચા કરીશું તથા લેટેક્સના ફાયદાઓની પણ ચર્ચા કરીશું. ત્યાર બાદ ટેક્સલાઇવનો ઉપયોગ કરીને લેટેક્સ કઈ રીતે વપરાય તે ભણીશું, આ એક ટેક્સ અને લેટેક્સ સૉફ્ટવેરનું મિશ્રણ ધરાવતું પૅકેજ છે. તેમાં SciTE એડિટરનો પણ સમાવેશ થાય છે.

## લેટેક્સનો ઉપયોગ (Use of LaTex)

લેટેક્સ વાપરવા માટે લેટેક્સ વિતરણ સૉફ્ટવેરની જરૂર પડે જેમાં ટેક્સ અને બીજા વધારાનાં સૉફ્ટવેરનો સમાવેશ થાય છે. ટેક્સલાઇવ એ પ્રમાણભૂત ઉબુન્ટુ રિપોઝિટરીમાં ઉપલબ્ધ ખૂબ જ લોકપ્રિય લેટેક્સ વિતરણ સૉફ્ટવેર છે. આઉટપુટ જોવા માટે બીજા સાદા એડિટર અને સૉફ્ટવેરની પણ જરૂર પડે, કારણકે લેટેક્સ જુદા-જુદા ઘણાંબધાં પ્રકારના ફાઇલમાળખાંમાં આઉટપુટ આપે છે. આ માટે ફાઇલમાળખાને આધારે આઉટપુટ જોવા માટે જુદાં-જુદાં સંબંધિત સૉફ્ટવેરની જરૂર પડે છે.

કોઈ પણ સાદા ટૅક્સ્ટ-એડિટરનો ઉપયોગ કરીને લેટેક્સનો દસ્તાવેજ બનાવી શકાય (જેવા કે gedit અથવા SciTE). દસ્તાવેજ બનાવવા માટે તેના જુદા-જુદા ભાગોને લેટેક્સના કમાન્ડ વડે ચિહ્નિત કરવામાં આવે છે. ઉદાહરણ તરીકે, દસ્તાવેજનું શીર્ષક આપવા માટે **\title** કમાન્ડનો ઉપયોગ થાય છે. દસ્તાવેજના લેખકનું નામ લખવા **\author** તેમજ દસ્તાવેજના સર્જનની તારીખ આપવા માટે **\date** કમાન્ડનો ઉપયોગ થાય છે. આ જ રીતે, **\chapter, \section, \subsection, \paragraph** વગેરે કમાન્ડ દસ્તાવેજનું લૉજિકલ માળખું બનાવવા માટે વપરાય છે.

આ દસ્તાવેજને વ્યાવસાયિક શૈલીમાં ગોઠવવા માટેના આંતરિક રસ્તાઓ લેટેક્સમાં ઉપલબ્ધ છે. જ્યારે આપણે દસ્તાવેજનું લખાણ લખીએ છીએ, ત્યારે આપણને કમાન્ડ સાથેનું ગોઠવણી વગરનું લખાણ જ દેખાય છે. ત્યાર પછી લેટેક્સ દસ્તાવેજ ઉપર પ્રક્રિયા કરીને એક આઉટપુટ ફાઇલ બનાવે છે. સાથે-સાથે કેટલીક વધારાની ફાઇલો પણ બનાવે છે. ઘણા બધા કિસ્સામાં આ વધારાની ફાઇલોને કોઈ પણ માહિતી ગુમાવ્યા વગર સુરક્ષિત રીતે (રિલિટ) દૂર કરી શકાય છે.

જ્યારે આપણે યોગ્ય સૉફ્ટવેરની મદદથી આઉટપુટ ફાઇલ જોઈશું અથવા પ્રિન્ટર વડે પ્રિન્ટ કાઢીશું, ત્યારે આપણને એક વ્યવસ્થિત માળખામાં ગોઠવેલ દસ્તાવેજ જોવા મળશે. જો આપણે દસ્તાવેજના આ દેખાવથી સંતુષ્ટ ન હોઈએ તો, આપણે આંતરિક શૈલી અથવા આપણી પોતાની શૈલીનો ઉપયોગ કરીને દસ્તાવેજમાં ફેરફાર કરી શકીએ.

જ્યારે-જ્યારે દસ્તાવેજમાં આવો ફેરફાર કરીશું, ત્યારે ફેરફારની અસર આઉટપુટમાં જોવા માટે દસ્તાવેજને પુનઃ-કમ્પાઇલ કરવો પડશે. ટેક્સ અને લેટેક્સ બંને .tex ફાઇલ અનુલંબનનો ઉપયોગ કરે છે. લેટેક્સમાં હવે pdflatex કમાન્ડ ઉપલબ્ધ છે, જે PDF (Portable Document Format) માળખામાં આઉટપુટ આપે છે. PDF ફાઇલને સ્ક્રીન ઉપર જોઈ શકાય છે, તેમજ પ્રિન્ટરની મદદથી પ્રિન્ટ પણ કાઢી શકાય છે. જે પ્રિન્ટ મૉનિટર ઉપર દેખાય છે તેવી જ આબેહૂબ હશે. વેબસાઇટ ઉપર દસ્તાવેજનું વિતરણ (શેરિંગ) કરવા માટે PDF ફાઇલ ખૂબ જ પ્રચલિત છે. ઉબુન્ટુના evince ડૉક્યુમેન્ટ વ્યુવર સૉફ્ટવેરમાં PDF દસ્તાવેજને જોઈ શકાય છે. જ્યારે દસ્તાવેજને સંપાદિત-કમ્પાઇલ અને જોવાનો ક્રમ આ મુજબ છે :

- Gedit જેવા કોઈ સાદા ટૅક્સ્ટ-એડિટરની મદદથી દસ્તાવેજ સંપાદિત કરો.
- જ્યાં ટેક્સફાઇલ સેવ કરી હોય તે ડિરેક્ટરીમાં ***pdflatex filename*** કમાન્ડ વડે પ્રૉમ્પ્ટ ઉપરથી દસ્તાવેજને કમ્પાઇલ કરો.

- GUIની મદદ વડે અથવા કમાન્ડ પ્રૉમ્પ્ટ ઉપરથી ***evince pdffilename*** કમાન્ડ આપીને બનેલી PDF ફાઇલને જુઓ. (જ્યારે તમે PDF ફાઇલ બંધ કરશો, ત્યાર પછી જ ટર્મિનલ તમને પ્રૉમ્પ્ટ દેખાડશે.)

લેટેક્સ ફાઇલમાં સુધારાવધારા કરવા માટે આપણે SciTE (સાઇટ) એડિટરનો ઉપયોગ કરી શકીએ છીએ. gedit અને SciTE બંનેમાં (સિન્ટેક્સને) વાક્યરચનાને અલગ રીતે પ્રદર્શિત કરવામાં આવે છે. (વાંચવામાં અને ઓળખવામાં સરળતા માટે બીજી ભાષાના ઘટકને અલગ રંગમાં દર્શાવવામાં આવે છે.), gedit એડિટર કરતા SciTEમાં એક ફાયદો એ છે કે તેમાં જ દસ્તાવેજને કમ્પાઇલ કરીને જોઈ શકાય છે. તમે બધા જ, SciTE એડિટરથી પરિચિત છો. આપણે લેટેક્સ ભણવા માટે અહીં SciTEનો જ ઉપયોગ કરીશું. પીડીએફ લેટેક્સ સાથે SciTEનો ઉપયોગ કરવા માટે પ્રકરણના અંતમાં આપેલ માહિતી મુજબ કન્ફિગ્યુરેશન ફાઇલમાં ફેરફાર કરવો પડે. જરૂરી સૉફ્ટવેર પ્રસ્થાપિત કર્યા પછી ઉપયોગકર્તાએ ફક્ત એક જ વખત આ કાર્ય કરવું પડે છે.

## લેટેક્સ ભાષા (The LaTex Language)

લેટેક્સ એ મૂળભૂત એક માર્ક-અપ ભાષા છે. લેટેક્સના પ્રોગ્રામમાં સાદું લખાણ તેમજ માર્ક-અપ લખાણ હોય છે, જેને કમાન્ડ તરીકે ઓળખવામાં આવે છે. કેટલાક કમાન્ડ સ્વતંત્ર હોય છે, તે લખાણના કોઈ પણ ભાગને નિશાન કરતા નથી. આવા કમાન્ડ જ્યારે લેટેક્સ દસ્તાવેજની પ્રક્રિયા કરે છે ત્યારે ઘણાંબધાં મહત્ત્વનાં કાર્ય કરતા હોય છે. જેમકે લખાણ અથવા દસ્તાવેજ વિશેની માહિતી પૂરી પાડે છે, દસ્તાવેજના માળખામાં નિશાની કરેલ લખાણની ભૂમિકા દર્શાવે છે. (અને આના કારણે લેટેક્સ લખાણને ચોક્કસ માળખામાં ગોઠવે છે.) તે સીધેસીધું ગોઠવણીનું માળખું આપે છે અથવા લેટેક્સને દસ્તાવેજની ચોક્કસ પ્રક્રિયા કરવાની સૂચના આપે છે. (ઉદાહરણ તરીકે, નિશ્ચિત પાનાંના કદનો ઉપયોગ કરવો, એકી નંબરના પેજ ઉપરથી જ નવું પ્રકરણ શરૂ કરવું વગેરે....)

લેટેક્સના કમાન્ડની શરૂઆત \બૅકસ્લેશ અક્ષર પછી કમાન્ડનું નામ એ રીતે થાય છે. કમાન્ડનું નામ ફક્ત મૂળાક્ષરો અથવા મૂળાક્ષરો સિવાયના એક અક્ષરનું બનેલું હોય છે. લેટેક્સના કમાન્ડ કેસ-સૅન્સિટિવ છે. (કૅપિટલ (મુખ્ય) અને નાના અક્ષરને અલગ અલગ ગણે છે.) કેટલાક કમાન્ડ વધારાની માહિતી પણ સ્વીકારે છે. (ઉદાહરણ તરીકે \textcolor કમાન્ડમાં લખાણ કયા રંગમાં દર્શાવવું છે, તે રંગ પણ આપવો પડે છે.) આ વધારાની માહિતીને આર્ગ્યુમેન્ટ તરીકે ઓળખવામાં આવે છે.

આર્ગ્યુમેન્ટ મુખ્ય બે પ્રકારની હોય છે. વૈકલ્પિક આર્ગ્યુમેન્ટ, નામ પ્રમાણે આવી આર્ગ્યુમેન્ટ ફરજિયાત હોતી નથી. આવી આર્ગ્યુમેન્ટ આપણે આપી શકીએ અથવા ન આપીએ તોપણ ચાલે. જો આપણે આવી એક અથવા એક કરતા વધારે વૈકલ્પિક આર્ગ્યુમેન્ટ આપવી હોય, તો તેને કમાન્ડના નામ પછી [ ] (ચોરસ કૌંસમાં) અલ્પવિરામ કરીને આપી શકાય. આ આર્ગ્યુમેન્ટ પછી { } (છગડિયા કૌંસ)માં આપેલ ફરજિયાત આર્ગ્યુમેન્ટ (જો હોય તો) આપવામાં આવે છે. ઉદાહરણ તરીકે, \documentclass[12pt]{article}; કમાન્ડમાં documentclass એ કમાન્ડનું નામ છે. 12pt એ વૈકલ્પિક આર્ગ્યુમેન્ટ છે જ્યારે article એ ફરજિયાત આર્ગ્યુમેન્ટ છે.

લેટેક્સમાં બધા જ વ્હાઇટ સ્પેસ અક્ષરો (સ્પેસ, ટેબ અને નવી લાઇન) જેમ છે તેમજ રાખવામાં આવે છે. એક કરતાં વધારે વ્હાઇટ સ્પેસ અક્ષરોને એક અક્ષરમાં પરિવર્તિત કરવામાં આવે છે. લીટીની શરૂઆતમાં આપવામાં આવેલ વ્હાઇટ સ્પેસ અક્ષરને અવગણવામાં આવે છે અને એક કરતાં વધારે ખાલી લીટીને નવા ફકરાની શરૂઆત ગણવામાં આવે છે. એટલે આનો મતલબ એ થયો કે આપણે લખાણને એક કરતાં વધારે લીટીમાં લખીશું, તો પણ તે લખાણને જ્યાં સુધી ખાલી લીટી નહિ આવે, ત્યાં સુધી એ આઉટપુટમાં સતત લખાણ તરીકે જ દર્શાવશે. જો લખાણમાં નવી લીટી ઉમેરવી હોય, તો \\ (લાઇન બ્રેક કમાન્ડ) લીટીના અંતમાં આપવો પડે છે. પરંતુ ફકરાની છેલ્લી લાઇનમાં આ કમાન્ડ આપવાની જરૂર નથી. આકૃતિ 12.1માં આવું સતત લખાણ તથા લખાણમાં લાઇનબ્રેક ઉમેરતું ઉદાહરણ આપવામાં આવેલું છે. (\textsf કમાન્ડની ચર્ચા પછી કરીશું.) આકૃતિ 12.2 લેટેક્સ ફાઇલનું આઉટપુટ દર્શાવે છે.

```
\documentclass[12pt]{article}
\title{Line handling in \LaTeX}
\date{May 2013}
\begin{document}
 \maketitle
 \section{Continuous Text}\textsf{
 We have no wings,  we cannot fly
 But we have legs to sail and climb
 By slow degrees and by and by
 The cloudy summits of our time}
 \section{Text with Seperate Lines}\textsf{
 Heights by great men reached and kept \\
 Were not attained by a sudden flight \\
 But they, while their companions slept, \\
 Were toiling upwards in the night}
\end{document}
```

**આકૃતિ 12.1 : લીટીના સંચાલનનું ઉદાહરણ દર્શાવતી ફાઈલ**

## 1 Continuous Text

We have no wings, we cannot fly But we have legs to sail and climb By slow degrees and by and by The cloudy summits of our time

## 2 Text with Seperate Lines

Heights by great men reached and kept
Were not attained by a sudden flight
But they, while their companions slept,
Were toiling upwards in the night

**આકૃતિ 12.2 : લીટી-સંચાલનના ઉદાહરણનું આઉટપુટ**

નીચેના અક્ષરો એ લેટેક્સના અનામત (Reserved) અક્ષરો છે.

# $ % & _ (અન્ડરસ્કોર) { } ^ ~ \

લેટેક્સમાં આ અક્ષરોનો ખાસ અર્થ છે. આ અક્ષરોનો સીધેસીધો ઉપયોગ લેટેક્સના લખાણમાં કરી શકાતો નથી. જો આ અક્ષરો આપણે લખાણમાં વાપરવા હોય, તો તે નીચે મુજબ લખીને વાપરી શકાય છે.

\# \$ \% \& _ \{ \} \^{} \~{} \textbackslash{}

છેલ્લા ત્રણ અક્ષરોની ખાસ નોંધ લો. ચિહ્ન < અને > જુદી જ રીતે છપાય છે. (મેથ મોડ સિવાય) તેને લખવા માટે **\textless** અને **\textgreater** લખવા પડે છે. ` (ગ્રેવ એસેન્ટ અથવા બેકકોટ) અને ' (એપોસ્ટ્રોફ અથવા સ્ટ્રેઇટ કોટ)નો ઉપયોગ લખાણને એક અવતરણચિહ્નમાં લખવા માટે થાય છે. દા.ત., `Book Code'. બે અવતરણચિહ્ન કરવા માટે આ ચિહ્ન બે વખત લખવાં પડે છે. દા.ત. ``Book Code''. (આ બે વખત લખાયેલા સ્ટ્રેઇટ કોટ છે, એક વખત લખાયેલ ડબલ કોટ નથી.) તે ફાઇલની અંદર વિચિત્ર રીતે દેખાય છે પરંતુ આઉટપુટમાં સરખી રીતે દેખાય છે.

લેટેક્સ લખાણના ભાગને ચિહ્નિત કરવા માટે જૂથનો ઉપયોગ કરે છે. આવાં જૂથ છગડિયા કૌંસ { અને }માં લખવામાં આવે છે. આવા જૂથમાં આપવામાં આવેલા કોઈ પણ કમાન્ડ ફક્ત જૂથમાં રહેલ કમાન્ડ પછીના લખાણને જ અસર કરે છે. અને કેટલાક કમાન્ડ જૂથમાં હોય છે જે આખા જૂથને લાગુ પડે છે. જૂથ એ ટૂંકું લખાણ જેવું કે લીટીનો ભાગ, થોડી લીટીઓ કે ફકરાને જૂજ – થોડા કમાન્ડ એકસાથે લાગુ પાડવા માટે ઉપયોગી છે.

જ્યારે મોટી સંખ્યામાં કમાન્ડ લાગુ પાડવા હોય (ઉદાહરણ તરીકે ટેબલની ગોઠવણી કરવી હોય અથવા ગાણિતિક સમીકરણ લખવું હોય) અથવા લખાણના કોઈ મોટા ભાગને ઘણા બધા કમાન્ડની અસર આપવી હોય (જેમ કે ઘણા બધા ફકરાઓ, આખો વિભાગ) ત્યારે લેટેક્સ એક અલગ સગવડ પૂરી પાડે છે. આ સગવડને એન્વાયર્નમેન્ટ તરીકે ઓળખવામાં આવે છે. આ એન્વાયર્નમેન્ટ \begin{environment-name} કમાન્ડથી શરૂ થાય છે અને \end{environment-name} કમાન્ડથી પૂરું થાય છે.

એન્વાયર્નમેન્ટમાં લખેલ બધા જ લખાણ ઉપર એન્વાયર્નમેન્ટની બધી જ (ફૉર્મેટિંગ) માળખાકીય લાક્ષણિકતા લાગુ પડે છે. એન્વાયર્નમેન્ટ નેસ્ટેડ પણ હોઈ શકે; એક એન્વાયર્નમેન્ટમાં બીજું એન્વાયર્નમેન્ટ હોય તેને નેસ્ટેડ એન્વાયર્નમેન્ટ કહેવાય. આવા કેટલાક વિષયવસ્તુના પ્રકાર પ્રમાણે જુદા જુદા પ્રમાણભૂત એન્વાયર્નમેન્ટ છે. જેવા કે સમીકરણ (equation), અવતરણ (quotation), ટેબલ અને યાદી (list). આ બધા જ એન્વાયર્નમેન્ટ તેના વિષયવસ્તુ પ્રમાણે તૈયાર ફૉર્મેટિંગ કમાન્ડ સાથે આવે છે.

લેટેક્સના દસ્તાવેજને છાપી શકાય છે તેમજ પ્રદર્શિત કરી શકાય છે. લેટેક્સમાં જુદાં-જુદાં ઘણાંબધાં ફિચર (લાક્ષણિકતા) છે. જેમકે પ્રોગ્રામિંગ કરવું, દસ્તાવેજનો કોઈક ભાગ સ્વયંસંચાલિત બનાવવો અથવા એક કરતા વધુ દસ્તાવેજો બનાવવા (મેઇલમર્જ) તેમજ અન્ય વ્યક્તિ અથવા ટીમ દ્વારા બનાવેલ ટેમ્પલેટ અથવા પૅકેજનો ઉપયોગ કરવો. કેટલીક વખત આ વ્યક્તિને પણ લેટેક્સનો કોડ ખેંચવાની જરૂર પડે છે.

આવા કિસ્સામાં જટિલ ભાગનું વિગતવાર વિશ્લેષણ આપવાથી આવા કોડને બીજી વ્યક્તિ સરળતાથી સમજી શકે છે. આવું વિગતવાર વિશ્લેષણ કૉમેન્ટના ભાગ રૂપે આપી શકાય. લેટેક્સમાં % ચિહ્નથી કૉમેન્ટ શરૂ કરી શકાય છે. % પછીનું તમામ લખાણ, લીટીના અંત સુધીનું કૉમેન્ટ તરીકે ગણાય છે. કૉમેન્ટ એ એડિટરમાં રહેલ કોડને વાંચવા માટે, સમજવા માટે અને જરૂર જણાય ત્યાં સુધારાવધારા કરવા માટે ઉપયોગકર્તાને ઉપયોગી છે. કંપાઇલેશનમાં કૉમેન્ટને અવગણવામાં આવે છે તેમજ આઉટપુટમાં પણ કૉમેન્ટનું કોઈ સ્થાન હોતુ નથી.

## લેટેક્સ દસ્તાવેજનું માળખું (The Structure of a LaTex Document)

લેટેક્સ દસ્તાવેજના બે ભાગ છે : પ્રસ્તાવના અને વિષયવસ્તુ. પ્રસ્તાવનામાં મેટાડેટા (માહિતીની માહિતી) હોય છે. મેટાડેટા એ દસ્તાવેજ વિશેની માહિતી છે. (ઉદાહરણ તરીકે, આ કેવા પ્રકારનો દસ્તાવેજ છે, દસ્તાવેજના લેખક કોણ છે, દસ્તાવેજ ક્યારે બનાવેલ છે) અને લેટેક્સ દસ્તાવેજની પ્રક્રિયા કઈ રીતે કરશે, તેની સૂચના હોય છે. મૂળ વિષયવસ્તુ હંમેશાં એન્વાયર્નમેન્ટ દસ્તાવેજમાં હોય છે, જે \begin{document} અને \end{document} કમાન્ડની વચ્ચે લખાયેલ હોય છે.

## પ્રાસ્તાવિક ભાગ (The Preamble)

લેટેક્સમાં વિવિધતાસભર ઘણાબધા જુદી-જુદી લાક્ષણિકતા અને માળખાવાળા દસ્તાવેજ બનાવી શકાય છે. લેટેક્સે એ જાણવું જરૂરી છે કે મૂળ ફાઇલ તરીકે ક્યા પ્રકારનો દસ્તાવેજ ઉપયોગમાં આવે છે. પ્રસ્તાવનાનું સૌથી પહેલું તત્ત્વ (એલિમેન્ટ) **\documentclass**{document-class-name} હોવું જ જોઈએ. જે દસ્તાવેજનો પ્રકાર જણાવે છે. કેટલાક સામાન્ય દસ્તાવેજના ક્લાસ કોષ્ટક 12.1માં આપેલ છે. કેટલાક દસ્તાવેજ-ક્લાસને વિકલ્પો પણ હોય છે. કોષ્ટક 12.2માં આવા સામાન્ય વિકલ્પ આપેલ છે.

| દસ્તાવેજ-ક્લાસ | ઉદ્દેશ (ઉપયોગ) |
|---|---|
| article | સ્વતંત્ર લેખ લખવા માટે |
| book | આખું પુસ્તક લખવા માટે |
| slides | પ્રદર્શન માટેની સ્લાઇડ બનાવવા માટે, આમાં ફૉન્ટનું કદ આપોઆપ મોટું થઈ જાય છે |
| letter | પત્ર લખવા માટે |
| beamer | બીમર પૅકેજનો ઉપયોગ કરીને ઑફિસસ્યુટ જેવું જ પ્રદર્શન બનાવવા માટે |

**કોષ્ટક 12.1 : કેટલાક સામાન્ય દસ્તાવેજ-ક્લાસ**

| વિકલ્પ | કાર્ય |
|---|---|
| 10pt, 11pt, 12pt | દસ્તાવેજના મુખ્ય ફૉન્ટનું કદ 10 પૉઇન્ટ (પૂર્વનિર્ધારિત), 11 પૉઇન્ટ કે 12 પૉઇન્ટ રાખવા માટે. |
| a4paper, letterpaper, legalpaper | પાનાંનું કદ નક્કી કરવા માટે. આ બધા જુદાં-જુદાં આંતરરાષ્ટ્રીય પાનાંનાં કદ છે. આમાંનું ઑફિસના ઉપયોગ માટેનું A4, લેટર અને લીગલ એ સામાન્ય કદ છે. |
| fleqn | સમીકરણો અને રચનાઓને વચ્ચે રાખવાને બદલે ડાબી બાજુ (લેફ્ટ અલાઇન) દર્શાવવા માટે |
| landscape | દસ્તાવેજને લૅન્ડસ્કેપમાં છાપવા માટે (આડા પાનાંમાં છાપવા માટે) |

કોષ્ટક 12.2 : કેટલાક દસ્તાવેજ-ક્લાસના કેટલાક સામાન્ય વિકલ્પ

વૈકલ્પિક પૅકેજની જાહેરાત પછી દસ્તાવેજના ક્લાસને જાહેર કરવામાં આવે છે. લેટેક્સ પદ્ધતિ પોતે જ કેટલીક સામાન્ય ટાઇપસેટિંગની જરૂરિયાત પૂરી પાડે છે, પરંતુ ઉપયોગકર્તાને જોઈતું જરૂરી બધું જ પૂરું પાડતું નથી. આથી વધારાની સગવડ માટે લેટેક્સ ઉપયોગકર્તાને પોતાનું પૅકેજ બનાવવાની સવલત આપે છે. લેટેક્સના ઉપયોગકર્તાનો મોટો સમૂહ છે, જે આવા નવાં પૅકેજ બનાવે છે અથવા અસ્તિત્વમાં હોય એવા પૅકેજમાં પોતાની જરૂરિયાત મુજબ ફેરફારો કરીને તેને Comprehensive Tex Archive Network (CTAN) ઉપર વહેંચે છે.

લેટેક્સમાં આવાં ઘણાંબધાં પૅકેજ પહેલેથી જ પ્રસ્થાપિત કરેલાં હોય છે. આપણા દસ્તાવેજમાં આવાં એક અથવા એકથી વધારે પૅકેજનો ઉપયોગ કરવો હોય, તો તેને પ્રાસ્તાવિક ભાગ (પ્રસ્તાવનામાં) \usepackage{package-name} કમાન્ડ વડે જાહેર (ડિક્લેર) કરવાં પડે છે. કેટલાંક પૅકેજને તેનું વર્તન નક્કી કરવા માટે વિકલ્પ પણ હોય છે. જો આમાંના કોઈ પણ વિકલ્પનો ઉપયોગ આપણે ન કરવો હોય, તો એક જ લાઇનમાં એક જ \usepackage કમાન્ડ વડે એક કરતાં વધારે પૅકેજને અલ્પવિરામ આપીને જાહેર (ડિક્લેર) કરી શકાય. કોષ્ટક 12.3માં કેટલાંક સામાન્ય ઉપયોગમાં આવતાં પૅકેજ દર્શાવવામાં આવ્યા છે.

| પૅકેજ | વર્ણન |
|---|---|
| amsmath | આ પૅકેજમાં લેટેક્સમાં ગણિત માટેનાં વિસ્તરણ હોય છે. મૂળભૂત રીતે અમેરિકન ગણિત સોસાયટી માટે બનાવવામાં આવેલ. |
| color | લખાણના કલર ઉમેરવા માટે |
| easylist | એક કરતાં વધુ સ્તરની યાદી ઉમેરવા માટે |
| geometry | પાનાંની રચના માટે, જેમકે પાનાનું કદ નક્કી કરવું, સ્થાપન (ઓરિએન્ટેશન) નક્કી કરવું માર્જિન નક્કી કરવું વગેરે |
| listings | દસ્તાવેજમાં પ્રોગ્રામિંગ કોડ ઉમેરવા માટે |
| setspace | લીટી વચ્ચેની જગ્યા બદલવા માટે (લાઇનસ્પેસિંગ) |

કોષ્ટક 12.3 : સામાન્ય રીતે ઉપયોગમાં આવતાં પૅકેજ

વધુમાં માહિતીના ત્રણ ઘટકો પૂરા પાડવામાં આવે છે, જે નીચે દર્શાવેલ છે :

\title{the-title-of-the-document} – દસ્તાવેજનું શીર્ષક

\author{author(s) of the document} – દસ્તાવેજના લેખક

\date{date of creation / last update of the document, in any format} – દસ્તાવેજ બનાવ્યાની તારીખ

જો લેટેક્સ તમારા માટે સ્વયંસંચાલિત રીતે શીર્ષકનું સર્જન કરે તેમ તમે ઇચ્છતા હો, તો તમારે શીર્ષક અને લેખકની માહિતી પૂરી પાડવી જરૂરી છે, પરંતુ તારીખ દર્શાવવી મરજિયાત છે. જો આ માહિતી પૂરી પાડવામાં નહિ આવે તો, કમ્પાઇલેશનની તારીખને શીર્ષક તરીકે લેવામાં આવશે. આ માહિતી પ્રસ્તાવનામાં અથવા તો દસ્તાવેજ એન્વાયર્નમેન્ટમાં પ્રથમ વસ્તુ તરીકે આપવામાં આવે છે.

## દસ્તાવેજ એન્વાયર્નમેન્ટ (The Document Environment)

લેખ અને સ્લાઇડ માટેના દસ્તાવેજ એન્વાયર્નમેન્ટને ફક્ત દસ્તાવેજના મુખ્ય વિષયવસ્તુને અનુસરતું શીર્ષક હશે. આ શીર્ષક લેટેક્સ દ્વારા સ્વયંસંચાલિત રીતે અપાશે. જ્યારે તે \maketitle કમાન્ડ મેળવશે, અલબત્ત \title, \author અને \date કમાન્ડ્સ \maketitle પહેલાં અપાઈ જવા જોઈએ. કારણકે તે માહિતી શીર્ષકના સર્જનમાં ઉપયોગી બને છે.

એક પુસ્તકમાં ઘણા વિસ્તૃત વિભાગ હોઈ શકે અલબત્ત તેમાંના મોટા ભાગના સ્વૈચ્છિક હોઈ શકે. પુસ્તકના દસ્તાવેજ એન્વાયર્નમેન્ટને મુખ્ય ત્રણ ભાગમાં વિભાજિત કરી શકાય - આગળની વિગત, મુખ્ય વિગત અને પાછળની વિગત. આ માટે અનુક્રમે \frontmatter, \mainmatter અને \backmatter કમાન્ડ આપવામાં આવે છે. આ વિવિધ વિભાગોમાં અલગ-અલગ વસ્તુઓ હોઈ શકે, જેમકે આગળની વિગતનું શીર્ષક, વિષયવસ્તુનું કોષ્ટક અને પ્રસ્તાવના તેમજ પાછળની વિગતની ગ્રંથસૂચિ, અનુક્રમણિકા અને સંદર્ભ સૂચિ અને મુખ્ય વિગતમાં પ્રકરણોના સ્વરૂપમાં પ્રાથમિક વિષય વસ્તુ, વિભાગો અને પેટા વિભાગો.

પુસ્તકના મુખ્ય વિષયવસ્તુને સ્તરીકરણ માળખુ હોય છે. જ્યાં પુસ્તકનું વિભાગોમાં, વિભાગોનું પ્રકરણમાં, પ્રકરણોનું મુદ્દાઓમાં, મુદ્દાઓનું પેટા મુદ્દાઓમાં, પેટા મુદ્દાઓનું પેટા-પેટા મુદ્દાઓમાં, પેટા-પેટા મુદ્દાઓનું ફકરાઓમાં, ફકરાઓનું પેટા ફકરાઓમાં વિભાજન કરવામાં આવે છે. આ માટે અનુક્રમે \part, \chapter, \section, \subsection, \subsubsection, \paragraph અને \subparagraph કમાન્ડ આપવામાં આવે છે.

દરેક કમાન્ડમાં ફરજિયાત એક આર્ગ્યુમેન્ટ આપવામાં આવે છે. શીર્ષક તથા એક વૈકલ્પિક આર્ગ્યુમેન્ટ વિષયવસ્તુના કોષ્ટકનું શીર્ષક, જે નવો વિભાગ, પ્રકરણ કે મુદ્દો શરૂ કરે છે. મુખ્ય લખાણમાં દર્શાવવામાં આવતું શીર્ષક લાંબું હોઈ શકે તેમજ તેને આંતરિક ફૉર્મેટિંગ પણ હોઈ શકે (જેમકે અક્ષરોને ઘાટા કરવા (બોલ્ડ) અથવા અક્ષરોને ત્રાંસા કરવા (ઇટાલિક)) જ્યારે વિષયવસ્તુના કોષ્ટકને આપવામાં આવેલ શીર્ષક ટૂંકું હોય તેમજ લખાણની સાતત્યતા જાળવવા માટે તેને ખાસ પ્રકારનું કાંઈ ફૉર્મેટિંગ આપવામાં આવેલું હોતું નથી. આ 7 ઘટકો એકબીજાની અંદર આવેલા હોય છે તેમજ દરેક સ્તરને પૂર્ણાંકનંબર આપવામાં આવેલ હોય છે, જેમકે વિભાગને સ્તર 1, પ્રકરણને સ્તર 0, મુદ્દાને સ્તર 1 વગેરે.

લેટેક્સ દ્વારા વિભાગ, પ્રકરણ અને મુદ્દાઓને સ્વયંસંચાલિત અનુક્રમનંબર આપવામાં આવે છે. આ માટે લેખકે ચિંતા કરવાની હોતી નથી. લેખક પ્રકરણ, મુદ્દા-પેટા મુદ્દાઓના ક્રમને ગમે ત્યારે ફેરવી શકે છે. આ માટે ફરી વખત અનુક્રમનંબર આપવાની તસ્દી લેવાની જરૂર રહેતી નથી. વિભાગોના અનુક્રમ રોમનમાં અપાય છે (I, II, III, વગેરે), જ્યારે પ્રકરણ, મુદ્દાઓ અને પેટા મુદ્દાઓ વગેરેના અનુક્રમ અરેબિકમાં અપાય છે (1, 2, 3, વગેરે).

\appendix કમાન્ડ પછીનાં (જે એક જ વખત આપી શકાય છે) બધાં જ પ્રકરણો પરિશિષ્ટો તરીકે સમજવામાં આવે છે અને તેમને બધાને અનુક્રમનંબર મોટા મૂળાક્ષર વડે આપવામાં આવે છે (A, B, C વગેરે). આગળના મુદ્રણનાં પાનાંઓને રોમનમાં અનુક્રમનંબર આપવામાં આવે છે, જ્યારે મુખ્ય મુદ્રણ અને પાછળના મુદ્રણમાં અરેબિકમાં અનુક્રમનંબર આપવામાં આવે છે, જે 1થી શરૂ થશે. આગળના મુદ્રણ અને પાછળના મુદ્રણ બંનેમાં પ્રકરણ હોય છે (જેવા કે પ્રસ્તાવના, કબૂલાતનામું, ગ્રંથસૂચિ). આવાં પ્રકરણોને સામાન્ય રીતે મુદ્દાઓ કે પેટાઘટકો હોતા નથી. દસ્તાવેજના જુદા ઘટકને દર્શાવતા કમાન્ડમાં નંબરની જગ્યાએ તારો (*) પણ હોઈ શકે. ઉદાહરણ તરીકે \section* કમાન્ડ અનુક્રમનંબર વગરનો મુદ્દો બનાવવા માટે વાપરી શકાય.

મૂળભૂત રીતે અનુક્રમનંબર 2 સ્તર સુધી આપી શકાય છે. એટલે કે પેટા મુદ્દા સુધી આપી શકાય છે. પેટા-પેટા મુદ્દાઓ અને ત્યાર પછીના ભાગને અનુક્રમનંબર આપવામાં આવતા નથી. આને બદલવા માટે લેટેક્સની પ્રસ્તાવનાના આંતરિક ગણકમાં ફેરફાર કરીને બદલી શકાય. ઉદાહરણ તરીકે \setcounter{secnumdepth}{3} કમાન્ડ ઘટકને સ્તર 3 (પેટા-પેટા મુદ્દા) સુધી અનુક્રમનંબર આપે છે. પૂર્ણવિરામ (.) અને ઘટકના નંબર વડે મુખ્ય ઘટકના ઘટકોને અનુક્રમનંબર આપી શકાય. આમાં પ્રકરણ એક અપવાદ છે. તેને આપેલ અનુક્રમનંબરને કોઈ વિભાગનંબર કે પૂર્ણવિરામ હોતું નથી. પ્રકરણને સાદો અરેબિક નંબર આપેલો હોય છે. જો એક પુસ્તકના વિભાગ IIમાં પ્રકરણ 5 અને તેમાં મુદ્દા 4ના પેટા મુદ્દા 1 હોય, તો આ પેટા મુદ્દાને 5.4.1 અનુક્રમનંબર આપવામાં આવે છે.

\tableofcontents કમાન્ડ આપવામાં આવે છે, ત્યારે એક સુવ્યવસ્થિત ગોઠવેલ વિષયવસ્તુનું કોષ્ટક (TOC) શીર્ષક ઘટક દ્વારા લેટેક્સમાં આપમેળે બને છે. ફરી વખત, વિષયવસ્તુના કોષ્ટકનું સ્તર 2 સુધી હોય છે (પેટા મુદ્દા સુધી). પરંતુ આને tocdepth આંતરિક ગણક વડે બદલી શકાય છે.

અહીં એક નોંધવા જેવો મહત્ત્વનો મુદ્દો એ છે કે લેટેક્સ સૉર્સફાઇલની શરૂથી અંત સુધી ક્રમશઃ એક જ વખતમાં પ્રક્રિયા કરે છે અને ક્રમશઃ આઉટપુટ ફાઇલ પણ બનાવે છે. તે ફાઇલમાં આગળ અને પાછળ હરતું-ફરતું નથી. આ એક સમસ્યા છે. દસ્તાવેજમાં વિષયવસ્તુનું કોષ્ટક પહેલાં અને ત્યાર પછી પાનાંનંબર સાથે પ્રકરણ અને મુદ્દાઓ આઉટપુટમાં જોવામાં મળે છે. જોકે આ તબક્કે લેટેક્સને પ્રકરણ કે મુદ્દાઓ વિશે માહિતી હોતી નથી. આવી પરિસ્થિતિમાં લેટેક્સને ઘણી વખત રન કરવું પડે છે.

પ્રથમ રનમાં લેટેક્સ દસ્તાવેજના માળખા વિશેની માહિતી ભેગી કરે છે અને એક પૂરક ફાઇલમાં સંગ્રહ કરે છે. આ તબક્કે આઉટપુટ ફાઇલમાં TOC ખાલી હોય છે અથવા જૂની પૂરક ફાઇલની માહિતી દર્શાવે છે. બીજા રનમાં તે પૂરક ફાઇલમાંથી સાચી માહિતી એકઠી કરીને શરૂઆતમાં સાચું TOC (વિષયવસ્તુનું કોષ્ટક) બનાવે છે.

TOCની જેમજ લેટેક્સ આકૃતિની યાદી, કોષ્ટકની યાદી, અન્યોન્ય સંબંધ (એક પુસ્તકમાં બીજા ફકરાનો નિર્દેશ), ગ્રંથ-સૂચિ, પારિભાષિક શબ્દકોશ અથવા અનુક્રમણિકા રાખે છે. આ બધી જ વસ્તુ લેખકનો ભાર હળવો કરે છે અને આના જ કારણે લેટેક્સ પ્રખ્યાત છે.

### ઉદાહરણ

લેટેક્સ પુસ્તકને કઈ રીતે ગોઠવે છે, તે જોવા માટે આપણે એક સૉર્સફાઇલ બનાવીએ અને તેનું આઉટપુટ જોઈએ. આ ઉદાહરણમાં આપણે લેટેક્સ સાથે SciTE એડિટરનો ઉપયોગ કરીશું.

- SciTE એડિટરમાં **File → New** મેનુવિકલ્પ પસંદ કરી નવી ફાઇલ બનાવો.
- લિસ્ટિંગ 12.1માં આપેલ માહિતી SciTE એડિટરમાં (ટાઇપ કરો) લખો.

```
\documentclass[12pt]{book}
\usepackage{amsmath}
\title{\huge Mathematics \\[3\baselineskip]
\Large Standard 12}
\author{Gujarat State Board of School Textbooks}
\date{2013}
\setcounter{secnumdepth}{2}
\setcounter{tocdepth}{1}
 \begin{document}
 \frontmatter
  \maketitle
  \chapter{\MakeUppercase{Fundamental Duties}}
  \tableofcontents
  \chapter{\MakeUppercase{About This Textbook...}}
\mainmatter
 \part{Semester I}
 \chapter{Set Operations}
 \section{Introduction}
 \section*{Exercise 1.1}
 \section{Properties of the Union Operation}
 \subsection{Union is a Binary Operation}
 \section{Properties of the Intersection Operation}
 \subsection{Intersection is a Binary Operation}
 \subsection{Associative Law}
 \chapter{Number Systems}
 \section{Introduction}
 \section*{Exercise 2.1}
 \section{Irrational Numbers}
 \chapter{Polynomials}
 \chapter{Coordinate Geometry}
 \chapter{Some Primary Concepts in Geometry : 1}
 \chapter*{Answers}
 \markboth{\MakeUppercase{Answers}}{}
 \addcontentsline{toc}{chapter}{Answers}
 \part{Semester II}
 \chapter{Quadrilaterals}
```

```
\section{Introduction}
\section{Plane Quadrilateral}
\chapter{Areas of Parallelograms and Triangles}
\section{Introduction}
\section{Interior of a Triangle}
\chapter{Circle}
\chapter{Surface Area and Volume}
\chapter*{Answers}
\markboth{\MakeUppercase{Answers}}{}
\addcontentsline{toc}{chapter}{Answers}
\appendix
\chapter{Terminology}
\backmatter
\end{document}
```

**લિસ્ટિંગ 12.1 : લેટેક્સમાં પુસ્તકનો એક નમૂનો**

- **File → Save** મેનુવિકલ્પ પસંદ કરી ફાઇલને સેવ કરો. અહીં એ નોંધી લેશો કે ફાઇલનું અનુલંબન .tex હોવું જોઈએ.
- લેટેક્સ ફાઇલને કમ્પાઇલ કરવા માટે **Tools → Build** વિકલ્પ પસંદ કરો. (શૉર્ટકટ કી : F7) આઉટપુટ વિન્ડોમાં ઘણાબધા સંદેશાઓ દેખાશે. જો આ સંદેશાઓમાં છેલ્લી લીટી (વાદળી કલરમાં) Exit code: 0 હોય તો આપણું કમ્પાઇલેશન સફળ થયું છે. નહિ તો આપણને ભૂલસંદેશ મળશે. પરંતુ ઘણી વાર તેનું અર્થઘટન કરવું મુશ્કેલ છે.
- જો કમ્પાઇલેશન સફળ થયું હોય તો, ડૉક્યુમેન્ટ વ્યૂઅરમાં ફાઇલ જોવા માટે **Tools → Go** મેનુવિકલ્પ પસંદ કરો. (શૉર્ટકટ કી : F5) SciTEમાં પરત ફરતા પૂર્વે ડૉક્યુમેન્ટ વ્યૂઅર બંધ કરવાનું ભૂલશો નહિ.

અહીં એ નોંધી લેશો કે આપણે લિસ્ટિંગ 12.1માં મૂળભૂત બુકશૈલીમાં કેટલાક ફેરફારો કર્યા છે, જેમાંની કેટલીક લાક્ષણિકતાઓની ચર્ચા અહીં કરી નથી. લેટેક્સ ટાઇપસેટિંગ ભારે વૈવિધ્યપૂર્ણ છે.

### લખાણ ફૉર્મેટિંગ (Text Formatting)

આપણે લેટેક્સ દસ્તાવેજમાં આખેઆખો ફકરો એન્ટર કી દબાવ્યા વગર લખી શકીએ છીએ. લેટેક્સ ત્યાર પછી આ લખાણને ગોઠવે છે. લેટેક્સ પાનાની પહોળાઈ, ફૉન્ટનું કદ અને ગોઠવણીના વિકલ્પના આધારે નક્કી કરે છે કે કેટલું લખાણ પહેલી લીટીમાં, કેટલું લખાણ બીજી લીટીમાં અને કેટલું લખાણ ત્યાર પછી આવશે. લેટેક્સ એક શબ્દને બે ભાગમાં વિભાજિત કરવાનું ટાળે છે. જો ફરજિયાત રીતે એક શબ્દને બે ભાગમાં વિભાજિત કરવો પડે, તો તે બંને ભાગોને જોડવા માટે (-) સંયોગ ચિહ્નનો ઉપયોગ કરે છે. ઉદાહરણ તરીકે formatting નામનો શબ્દ એક લીટીમાં ન સમાઈ શકતો હોય, તો પહેલી લીટીમાં formatt– અને ત્યાર પછીની લીટીમાં ingથી શરૂ થાય છે.

જ્યારે બીજી તરફ, કેટલીક પરિસ્થિતિ કે લખાણમાં કેટલાક તકનિકી બહુવિધ શબ્દોને અલગ-અલગ લીટીમાં વિભાજિત કરી શકાતા હોતા નથી. ઉદાહરણ તરીકે up to બે શબ્દો છે, પરંતુ એક લીટીના અંતમાં up અને બીજી લીટીની શરૂઆતમાં to લખવું અપેક્ષિત નથી. કારણ કે બંને શબ્દો સ્વતંત્ર છે અને બંનેનો અર્થ અલગ છે. જો આમ કરવામાં આવે, તો વાચક બીજી લીટીની શરૂઆતમાં to શબ્દ જોઈને આશ્ચર્ય પામશે.

આ એક સમતલ વાંચનમાં વિઘ્નરૂપ અનુભવ છે. આ રીતને અવગણવા માટે up અને to બંને શબ્દો કોઈ એક લીટીમાં જ હોવા જોઈએ. કાં તો પહેલી લીટીમાં અથવા બીજી લીટીમાં લેટેક્સમાં આવું કરવા માટે બે શબ્દો વચ્ચે મૂકી ન શકાય તેવી જગ્યા ~ (ટાઇલ્ડ) અક્ષર વડે દર્શાવવામાં આવે છે.

લેટેક્સમાં ફૉન્ટને ત્રણ વર્ગમાં વિભાજિત કરવામાં આવે છે. રોમન (સેરિફ પણ કહી શકાય), સેન્સ સેરિફ અને મોનોસ્પેસ. રોમન ફૉન્ટને રેખાના અંતમાં ટૂંકી આડી લીટી હોય છે. સેન્સ સેરિફને આવી ટૂંકી આડી લીટી હોતી નથી જ્યારે મોનોસ્પેસ ફૉન્ટ બધા જ અક્ષરો માટે સમાન પહોળાઈ ધરાવે છે. સામાન્ય રીતે મોનોસ્પેસ ફૉન્ટનો ઉપયોગ કમ્પ્યૂટરમાં કોડ લખવા માટે થાય છે. જ્યારે રોમન ફૉન્ટ એ મૂળભૂત ફૉન્ટ છે. આ ત્રણેય પ્રકારના ફૉન્ટનો ઉપયોગ કરવા માટે \textrm{text}, \textsf{text} અને \texttt{text} કમાન્ડ વાપરી શકાય. આકૃતિ 12.2માં તમે સેરિફ અને સેન્સ સેરિફ ફૉન્ટ વચ્ચેનો તફાવત જોઈ શકશો, જેમાં શીર્ષકનો વિભાગ મૂળભૂત સેરિફ ફૉન્ટમાં અને બોડીનો વિભાગ સેન્સ સેરિફ ફૉન્ટમાં છે.

\tiny, \scriptsize, \footnotesize, \small, \normalsize, \large, \LARGE, \huge અને \Huge કમાન્ડ વડે ફૉન્ટના કદમાં ફેરફાર કરી શકાય. અહીં એ નોંધી લેશો કે કમાન્ડ કેસ-સેન્સિટિવ છે. \textbf, \textit અને \emph કમાન્ડનો ઉપયોગ લખાણમાં ઘાટું (બોલ્ડ), ત્રાંસા (ઇટાલિક) કે ભાર દર્શાવતી (સામાન્ય રીતે ત્રાંસા જેવી જ) અસર આપવા માટે થાય છે. \textsc કમાન્ડનો ઉપયોગ નાના કદમાં મોટા અક્ષરો (કૅપિટલ અક્ષર) દર્શાવવા થાય છે જ્યારે fixltx2e પૅકેજના \textsuperscript અને \textsubscript કમાન્ડ વડે લખાણને સુપરસ્ક્રિપ્ટ (એક ચિહ્ન ઉપર કરેલું બીજું ચિહ્ન) અને સબસ્ક્રિપ્ટ (એક ચિહ્નની નીચે કરેલું બીજું ચિહ્ન)માં લખી શકાય છે.

## ફકરાની ગોઠવણી (Paragraph Formatting)

લેટેક્સમાં બે લીટીની વચ્ચે અંતર રાખવા માટે setspace પૅકેજમાં singlespace, onehalfspace, doublespace અને spacing{amount-of-spacing} એન્વાયર્નમેન્ટ ઉપલબ્ધ છે. લેટેક્સમાં મૂળભૂત રીતે બૉડીનું લખાણ સંપૂર્ણ ઉચિત ગોઠવાયેલ (જસ્ટિફાઇ) હોય છે. જો લખાણ ડાબી તરફ, જમણી તરફ અથવા વચ્ચે સીધી લીટીમાં ગોઠવવું હોય, તો flushleft, flushright અને center એન્વાયર્નમેન્ટનો ઉપયોગ કરવો પડે. મથાળાની નીચે આપેલ ફકરાને બાદ કરતાં બાકીના ફકરાની પહેલી લીટી આરંભમાં જગ્યા છોડીને લખેલ હોય છે.

ફકરાની શરૂઆતમાં બાહ્ય રીતે \indent અને \noindent કમાન્ડનો ઉપયોગ કરીને, પહેલી લીટીના આરંભમાં જગ્યા છોડવી કે ન છોડવી તે નક્કી કરી શકાય છે. verbatim એન્વાયર્નમેન્ટ પ્રક્રિયા કર્યા વગર આઉટપુટ જેમ છે તેમ (ખાસ અક્ષર, જગ્યા, નવી લાઇન અને લેટેક્સ કમાન્ડ સહિત) જ આપે છે. moreverb પૅકેજ-પ્રોગ્રામ કોડ સુધીમાં લીટીને નંબર આપવા માટે એક ફરજિયાત આર્ગ્યુમેન્ટ-પહેલી લીટીનો અનુક્રમનંબર-સાથે એક સૂચિકરણ (લિસ્ટિંગ) એન્વાયર્નમેન્ટ પૂરું પાડે છે.

## પૃષ્ઠ લે-આઉટ (Page Layout)

લેટેક્સમાં geometry પૅકેજનો ઉપયોગ કરીને પાનાંને લેઆઉટ આપી શકાય છે. \usepackage કમાન્ડમાં પાનાંનું કદ અને માર્જિન વૈકલ્પિક આર્ગ્યુમેન્ટ તરીકે પાસ કરી શકાય છે. દા.ત., નીચે આપેલો કમાન્ડ પાનાનું કદ A4 ઉપરનું માર્જિન 1 ઇંચ, નીચેનું માર્જિન 2 ઇંચ, ડાબી બાજુનું માર્જિન 1.5 ઇંચ અને જમણી બાજુનું માર્જિન 1 ઇંચ ગોઠવે છે.

\usepackage[a4paper, top=1in, bottom=2in, left=1.5in, right=1in]{geometry}

પાનાંનું કદ આંતરરાષ્ટ્રીય પ્રમાણિત હોય છે. સામાન્ય રીતે નિયમિત વપરાતા પ્રિન્ટરમાં પાનાનું કદ A4, લેટર અથવા લીગલ હોય છે. તેમને અનુક્રમે a4paper, letterpaper અને legalpaper વડે દર્શાવી શકાય છે. portrait (મૂળભૂત) અને landscape વિકલ્પ વડે પાનાનું ઓરિએન્ટેશન ગોઠવી શકાય છે.

દસ્તાવેજો એક બાજુ અથવા બે બાજુ છપાયેલા હોય છે. લેખ મૂળભૂત એક બાજુ, જ્યારે પુસ્તકો બંને બાજુ છપાયેલ હોય છે. બંને બાજુ છપાયેલ દસ્તાવેજને ડાબા પાનાં (બેકી) તથા જમણા પાના (એકી)માં અલગ અલગ માર્જિન છોડીને

જુદા પાડી શકાય છે. બાઇન્ડિંગ (ચોપડીનું વેષ્ટન) કરવા માટેની ખાલી જગ્યા પણ છોડવામાં આવે છે તેમજ એક એવો નિયમ પણ છે કે પ્રકરણની શરૂઆત એકી પાના ઉપરથી થવી જોઈએ.

## લેટેક્સમાં ગણિતશાસ્ત્રની સામગ્રીનું ટાઇપસેટિંગ (Typesetting Mathematical Content in LaTeX)

લેટેક્સમાં જટિલ ગાણિતિક સામગ્રીને આપોઆપ ગોઠવી આપવાની ક્ષમતા છે. અમેરિકન મેથેમેટિકલ સોસાયટીએ બનાવેલ amsmath, amssymb અને amsfonts પૅકેજનો ઉપયોગ કરીને ગાણિતિક સામગ્રીને ગોઠવવાનો એક સર્વસામાન્ય રસ્તો લેટેક્સમાં છે.

amsmath પૅકેજ ગાણિતિક સામગ્રી માટેના ઘણાંબધાં એન્વાયર્નમેન્ટ વ્યાખ્યાયિત કરે છે. ટાઇપસેટિંગ માટેના બે રસ્તાઓ છે. સૂત્ર અને સમીકરણ. જેને આપણે એક લીટીના ભાગમાં અથવા તો સ્વતંત્ર લીટીમાં છાપવાના હોય છે. math એન્વાયર્નમેન્ટનો ઉપયોગ કરીને પહેલા આ સ્વરૂપ મેળવી શકાય છે, ત્યાર પછીથી displaymath એન્વાયર્નમેન્ટનો ઉપયોગ કરવો પડે છે.

સમીકરણ (equation) એન્વાયર્નમેન્ટ એ એક display એન્વાયર્નમેન્ટ છે. જે સમીકરણના ક્રમ આપોઆપ દર્શાવે છે. લખાણમાં math એન્વાયર્નમેન્ટનો ઉપયોગ કરવાનો સરળ રસ્તો એ છે કે ગાણિતિક લખાણ સામગ્રીને $...$ની વચ્ચે લખવી. ગાણિતિક એન્વાયર્નમેન્ટમાં દરેક શબ્દને ગાણિતિક ચલ તરીકે ગણવામાં આવે છે. જે ટેક્સ્ટ (લખાણનું) એન્વાયર્નમેન્ટ કરતાં જુદુ છે. આ એન્વાયર્નમેન્ટને સરખી રીતે સમજવા માટે લિસ્ટિંગ 12.2માં આપેલ કોડની એક .tex ફાઇલ બનાવો.

- SciTEમાં **File → New** મેનુવિકલ્પની મદદ વડે નવી ફાઇલ બનાવો.
- લિસ્ટિંગ 12.2માં આપેલ લખાણને SciTE એડિટરમાં લખો.

```
\documentclass[12pt]{article}
\usepackage{amsmath}
\title{Introduction to \LaTeX}
\date{May 2013}
\begin{document}
\section*{math environment}
The quadratic equation, in its general form, is
\begin{math}
ax^2 + bx + c = 0
\end{math}.
You learnt about them in class X.

The quadratic equation, in its general form, is $ax^2 + bx + c = 0$. You learnt about
them in class X.

\section*{displaymath environment}
The quadratic equation, in its general form, is
\begin{displaymath}
ax^2 + bx + c = 0
\end{displaymath}. You learnt about them in class X.

\end{document}
```

**લિસ્ટિંગ 12.2 : Math એન્વાયર્નમેન્ટનું પ્રદર્શન**

- **File → Save** મેનુવિકલ્પનો ઉપયોગ કરીને ફાઇલને સેવ કરો.
- હવે લેટેક્સ ફાઇલને કમ્પાઇલ કરવા માટે **Tools → Build** મેનુવિકલ્પ પસંદ કરો. (શૉર્ટકટ કી : F7)
- જો કમ્પાઇલેશન સફળતાપૂર્વક થયું હોય તો, મૂળભૂત ડૉક્યુમેન્ટવ્યૂઅરમાં ફાઇલ જોવા માટે **Tools → Go** મેનુવિકલ્પ પસંદ કરો. (શૉર્ટકટ કી : F5). આકૃતિ 12.3માં આ ફાઇલનું આઉટપુટ દર્શાવેલ છે.

## math environment

The quadratic equation, in its general form, is $ax^2 + bx + c = 0$. You learnt about them in class X.

The quadratic equation, in its general form, is $ax^2 + bx + c = 0$. You learnt about them in class X.

## displaymath environment

The quadratic equation, in its general form, is

$$ax^2 + bx + c = 0$$

. You learnt about them in class X.

**આકૃતિ 12.3 : લિસ્ટિંગ 12.2માં આપેલ કોડના આઉટપુટનો એક ભાગ**

### ગાણિતિક ચિહ્નો (સંજ્ઞા)નો ઉપયોગ (Using Mathematical Symbols)

ગણિતશાસ્ત્ર ઘણાંબધાં ચિહ્નોનો ઉપયોગ કરે છે. અહીં આપણે લેટેક્સમાં ઉપયોગમાં આવતાં આવાં ગાણિતિક ચિહ્નોની ચર્ચા કરીશું. ગ્રીક મૂળાક્ષરના અક્ષરને તેના કમાન્ડ છે, જેવા કે \alpha, \beta, \gamma, \pi વગેરે નાના અક્ષર બનાવે છે, જ્યારે આ જ કમાન્ડનો પહેલો અક્ષર મોટો અક્ષર હોય, તો ઉદાહરણ તરીકે \Alphaનો તે ગ્રીક ભાષાનો મોટો અક્ષર બનાવે છે.

આ ઉપરાંત ગાણિતિક ચિહ્નો માટેના પણ કેટલાક કમાન્ડ છે. AMS પૅકેજનો ઉપયોગ કરીને આકૃતિ 12.4 અને આકૃતિ 12.5માં આવા કેટલાક લેટેક્સના કમાન્ડ અને તેનાં ગાણિતિક ચિહ્નો દર્શાવવામાં આવ્યા છે. ત્રિકોણમિતીનાં વિધેયો sin, cos અને બીજા બધા માટે કમાન્ડ દર્શાવવાનું કારણ એ છે કે તેને સ્વતંત્ર ચલ કરતાં વિધેય તરીકે ઓળખી શકાય એ રીતે ગોઠવવાની જરૂર હોય છે.

| | | | |
|---|---|---|---|
| < | $<$ | > | $>$ |
| = | $=$ | \leq | $\leq$ |
| \geq | $\geq$ | \neq | $\neq$ |
| \times | $\times$ | \div | $\div$ |
| \pm | $\pm$ | \mp | $\mp$ |
| \in | $\in$ | \notin | $\notin$ |
| \supset | $\supset$ | \subset | $\subset$ |
| \supseteq | $\supseteq$ | \subseteq | $\subseteq$ |
| \cup | $\cup$ | \cap | $\cap$ |

**આકૃતિ 12.4 : લેટેક્સમાં વપરાતી કેટલીક ગાણિતિક સંજ્ઞાઓ**

| | | | |
|---|---|---|---|
| \cong | $\cong$ | \propto | $\propto$ |
| \rightarrow | $\rightarrow$ | \parallel | $\parallel$ |
| \leftrightarrow | $\leftrightarrow$ | \leftarrow | $\leftarrow$ |
| \angle | $\angle$ | \bigodot | $\bigodot$ |
| \triangle | $\triangle$ | \overleftrightarrow{AB} | $\overleftrightarrow{AB}$ |
| \stackrel\frown{AB} | $\stackrel{\frown}{AB}$ | \overrightarrow{AB} | $\overrightarrow{AB}$ |
| \overline{AB} | $\overline{AB}$ | \perp | $\perp$ |
| 45^{\circ} | $45^{\circ}$ | \implies | $\implies$ |
| \iff | $\iff$ | \therefore | $\therefore$ |
| \because | $\because$ | \sin | $\sin$ |
| \cos | $\cos$ | \tan | $\tan$ |
| \sec | $\sec$ | \csc | $\csc$ |
| \cot | $\cot$ | \theta | $\theta$ |

આકૃતિ 12.5 : લેટેક્સમાં વપરાતી વધુ કેટલીક ગાણિતિક સંજ્ઞાઓ

### ગાણિતિક પ્રક્રિયકોનો ઉપયોગ (Using Mathematical Operators)

લેટેક્સમાં ઘણા બધા ગાણિતિક પ્રક્રિયકોનો ઉપયોગ થાય છે. પરંતુ પાવર શોધવા માટેનો પ્રક્રિયક (x2) અને ઇન્ડેક્સનો પ્રક્રિયક (x1) અલગથી આપવામાં આવતો નથી. આ માટે અનુક્રમે સામાન્ય સુપરસ્ક્રિપ્ટ પ્રક્રિયક ^ (કેરેટ અક્ષર) અને સામાન્ય સબસ્ક્રિપ્ટ પ્રક્રિયક _ (અન્ડરસ્કોર અક્ષર)નો ઉપયોગ થાય છે. સુપરસ્ક્રિપ્ટ પ્રક્રિયક લખાણને ઉપર તરફ અને સબસ્ક્રિપ્ટ લખાણને નીચે તરફ દર્શાવે છે. બંને પ્રક્રિયક લખાણના કદને ઘટાડે છે. સંપૂર્ણ કિંમત દર્શાવવા માટે પદાવલીને બે ઊભી લીટી | વચ્ચે લખવામાં આવે છે.

વિધેય બનાવવા માટે \frac{numerator}{denominator} કમાન્ડનો ઉપયોગ થાય છે, જ્યારે x સંખ્યાનું વર્ગમૂળ શોધવા માટે \sqrt{x} કમાન્ડનો ઉપયોગ થાય છે. આ પ્રક્રિયકો નેસ્ટેડ હોઈ શકે. એટલે કે વર્ગમૂળમાં ભાજકનો એક અપૂર્ણાંક બીજા અપૂર્ણાંકમાં અને ઉપરનો અંક બીજા અપૂર્ણાંકમાં હોઈ શકે વગેરે. આ માટે કોઈ મર્યાદા નથી. આ માટે ઘટકના કદ અને સ્થાનની કાળજી લેટેક્સ પોતે જ રાખે છે. આ માટે સામાન્ય રીતે કૌંસનો ઉપયોગ થાય છે. લિસ્ટિંગ 12.3માં ગાણિતિક પ્રક્રિયકોનો ઉપયોગ દર્શાવતું ઉદાહરણ આપેલ છે.

- **File → New** મેનુવિકલ્પનો ઉપયોગ કરી SciTEમાં નવી ફાઇલ બનાવો.
- લિસ્ટિંગ 12.3માં આપેલ લખાણ SciTE એડિટરમાં લખો.

```
\documentclass[12pt]{article}
\usepackage{amsmath}
\setlength{\parindent}{0pt}
\title{Introduction to \LaTeX}
\date{May 2013}
\begin{document}
\begin{math}
x^2 \\[6pt]
x_1 \\[6pt]
x_i \\[6pt]
x_i^2 \\[6pt]
x^y \\[6pt]
a^bc \\[6pt]
{a^b}^c \\[6pt]
a^{b^c} \\[6pt]
\frac{x}{y} \\[6pt]
\sqrt{b} \\[6pt]
\frac{\frac{1}{3}+\frac{3}{2}}{\frac{2}{3}+\frac{1}{2}} \\[6pt]
\frac{-b\pm\sqrt{b^2-4ac}}{2a} \\[6pt]
\frac{-b\pm\sqrt{\Delta}}{2a} \\[6pt]
\sqrt{1+\frac{1}{\sqrt{1+\frac{1}{\sqrt{1+\frac{1}{2}}}}}} \\[6pt]
\sqrt{(x_1-x_2)^2+(y_1-y_2)^2} \\[6pt]
|x-y| \\[6pt]
(\frac{x_1+x_2}{2},\frac{y_1+y_2}{2})
\end{math}
\end{document}
```

**લિસ્ટિંગ 12.3 : ગાણિતિક પ્રક્રિયકોનું ઉદાહરણ**

- **File → Save** મેનુવિકલ્પ વડે ફાઇલને સેવ કરો.
- હવે **Tools → Build** મેનુવિકલ્પ (શૉર્ટકટ કી : F7) પસંદ કરી તમારી લેટેક્સ ફાઇલને કમ્પાઇલ કરો.
- જો સફળતાપૂર્વક કમ્પાઇલેશન થયું હોય, તો **Tools → Go** મેનુ વિકલ્પ (શૉર્ટકટ કી : F5) વડે ડૉક્યુમેન્ટ વ્યૂઅરમાં ફાઇલ જુઓ. આકૃતિ 12.6 ડૉક્યુમેન્ટ વ્યૂઅરમાં ફાઇલનું આઉટપુટ દર્શાવે છે.

$$x^2 \qquad \sqrt{b}$$

$$x_1 \qquad \frac{\frac{1}{3}+\frac{3}{2}}{\frac{2}{3}+\frac{1}{2}}$$

$$x_i \qquad \frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

$$x_i^2 \qquad \frac{-b\pm\sqrt{\Delta}}{2a}$$

$$x^y \qquad \sqrt{1+\frac{1}{\sqrt{1+\frac{1}{\sqrt{1+\frac{1}{2}}}}}}$$

$$a^bc \qquad \sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$$

$$a^{b^c} \qquad |x-y|$$

$$a^{b^c} \qquad \left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right)$$

$$\frac{x}{y}$$

આકૃતિ 12.6 : લિસ્ટિંગ 12.3નું બે વિભાગમાં વિભાજિત આઉટપુટ

## સમીકરણોનો ઉપયોગ (Using Equations)

લેટેક્સ સમીકરણો માટે એક ખાસ સમીકરણ (equation) એન્વાયર્નમેન્ટ પૂરું પાડે છે. દરેક સમીકરણને સમીકરણ (equation) એન્વાયર્નમેન્ટમાં લખવામાં આવે છે, તેને ગણિત (Math) એન્વાયર્નમેન્ટમાં જોડવામાં આવતાં નથી. સમીકરણોને આપમેળે અનુક્રમ નંબર આપવામાં આવે છે અને કેન્દ્રમાં ગોઠવવામાં આવે છે (center-aligned). લિસ્ટિંગ 12.4માં સમીકરણ (equation) એન્વાયર્નમેન્ટનું ઉદાહરણ દર્શાવવામાં આવ્યું છે. તેમજ આકૃતિ 12.7માં આઉટપુટનો ભાગ દર્શાવવામાં આવ્યો છે.

```
\documentclass[12pt]{article}
\setlength{\parindent}{0pt}
\usepackage{amsmath}
\title{Introduction to \LaTeX}
\date{May 2013}
\begin{document}
\begin{equation}
\sin^2\theta + \cos^2\theta=1
\end{equation}
\begin{equation}
```

```
\sec^2\theta - \tan^2\theta=1
\end{equation}
\begin{equation}
\csc^2\theta - \cot^2\theta=1
\end{equation}
\end{document}
```

**લિસ્ટિંગ 12.4 : સમીકરણ (Equation) એન્વાયર્નમેન્ટનો ઉપયોગ**

$$\sin^2\theta + \cos^2\theta = 1 \qquad (1)$$

$$\sec^2\theta - \tan^2\theta = 1 \qquad (2)$$

$$\csc^2\theta - \cot^2\theta = 1 \qquad (3)$$

**આકૃતિ 12.7 : લિસ્ટિંગ 12.4નું આઉટપુટ**

## લેટેક્સના ફાયદાઓ (Advantages of LaTeX)

લેટેક્સના ઘણાબધા લાભ છે. ટેક્સને લેટેક્સમાં વિસ્તૃત કરવામાં આવેલ છે. લેટેક્સને પણ વિસ્તૃત કરી શકાય. લેટેક્સમાં નવાં લક્ષણો ઉમેરવા માટે અથવા વૈકલ્પિક અમલીકરણ માટે, લેટેક્સની સુવિધાઓ વધારવા માટે કોઈ પણ વ્યક્તિ વધારાના પૅકેજ બનાવી શકે છે. સમગ્ર વિશ્વમાં લેટેક્સ વપરાશકર્તાઓ દ્વારા વિવિધ જરૂરિયાતો પૂરી પાડવા માટે આવાં હજારો પૅકેજ બનાવવામાં આવ્યાં છે. આમાંનાં ઘણાંબધાં પૅકેજ નિઃશુલ્ક છે. તેને CTAN (The Comprehensive TeX Archive Network)ની વેબસાઇટ website at www.ctan.org ઉપર હોસ્ટ કરવામાં આવ્યાં છે.

લેટેક્સ જટિલ ગાણિતિક સૂત્રોને સરસ અને યોગ્ય રીતે ગોઠવવા માટે ખૂબ જ સારું છે. આ કારણે જ તે લેખકો, ગણિતશાસ્ત્રના પ્રકાશકો, ઇજનેરી, કમ્પ્યૂટરવિજ્ઞાન અને બીજા તકનીકી વિસ્તારમાં ખુબજ લોકપ્રિય છે. બીજું કારણ એ છે કે તે એક ખૂબ જ વ્યવસ્થિત ઑપનસોર્સ છે અને જ્ઞાનવર્ધિનીની શૈક્ષણિક ભાવના અને સહયોગથી વિકસાવવામાં આવ્યું છે, તેથી તે શિક્ષણશાસ્ત્રીઓ અને વિદ્વાનોમાં પણ લોકપ્રિય છે. આ ક્ષેત્રના લોકો લેટેક્સનો ઉપયોગ કરે છે, પોતાનાં મંતવ્યો અને અનુભવો વર્ધિનીને છે, એકબીજાને મદદ કરે છે અને લેટેક્સના વધુ વિકાસ માટે નવાં પૅકેજ બનાવે છે અને વહન કરે છે.

લેટેક્સમાં આપમેળે અનુક્રમનંબર આપવાની અને સંદર્ભ આપવાની તેમજ આપમેળે વિષયવસ્તુનું કોષ્ટક, અનુક્રમણિકા અને બીજી આવી જરૂરિયાત પૂરી પાડવાની સગવડ છે, જે લેખકનો માનસિક બોજો હળવો કરે છે.

### સારાંશ

આ પ્રકરણમાં આપણે લેટેક્સનો ઉપયોગ કરીને તકનિકી અને ગાણિતિક દસ્તાવેજ કઈ રીતે ગોઠવાય તે શીખ્યા. લેટેક્સનાં આવશ્યક તત્ત્વો ભણ્યાં. આપણે અહીં લેટેક્સ દસ્તાવેજના માળખા અને વાક્યરચનાની ચર્ચા કરી. ત્યાર પછી લેટેક્સમાં મૂળભૂત દસ્તાવેજ કઈ રીતે બનાવાય તેમજ કઈ રીતે ગોઠવાય, તેની ચર્ચા કરી અને છેલ્લે લેટેક્સની મુખ્ય તાકાત - ગાણિતિક સામગ્રી કઈ રીતે ગોઠવાય તે જોઈ ગયા. તે શબ્દપ્રક્રિયકથી કઈ રીતે જુદું પડે છે, તેની પણ ચર્ચા કરી અને અંતમાં લેટેક્સના ફાયદાઓની ચર્ચા કરી.

## SciTE અને લેટેક્સ કન્ફિગ્યુર કરવું (ફક્ત શિક્ષકો માટે)

- જો SciTE પહેલેથી ખૂલેલું હોય તો બંધ કરો.
- ટર્મિનલ ઓપન કરો.
- નીચેનો કમાન્ડ રન કરો :

  sudo gedit /usr/share/scite/tex.properties&

  આ કમાન્ડ આપવાથી gedit એડિટરમાં ફાઇલ ખૂલશે. તેમાં નીચે પ્રમાણે ફેરફાર કરો :
- નીચેની લીટીઓ કાઢી નાખો :

  file.patterns.tex=*.tex;*.sty

  file.patterns.context=*.tex;*.tui;*.tuo;*.sty

  ખાતરી કરો કે તમે નીચે આપેલી લીટી કાઢી નથી :

  file.patterns.latex=*.tex;*.sty;*.aux;*.toc;*.idx
- લીટી બદલો :

  **command.go.$(file.patterns.latex)=gv $(FileName).pdf**

  ના સ્થાને

  command.go.$(file.patterns.latex)=evince $(FileName).pdf લીટી લખો
- ફાઇલને સેવ કરી gedit બંધ કરો.
- SciTE શરૂ કરો. યોગ્ય લેટેક્સ ફાઇલ ખોલી F7 અને F5 દબાવો. લેટેક્સ ફાઇલ કમ્પાઇલ થઈને એક પીડીએફ ફાઇલ ખૂલશે. SciTEમાં પરત ફરતા પૂર્વે પીડીએફ ફાઇલ અવશ્ય બંધ કરો.

### સ્વાધ્યાય

1. શબ્દપ્રક્રિયક સાથે લેટેક્સની સરખામણી કરી બંનેનાં સારાં અને નબળાં પાસાંની યાદી તૈયાર કરો.
2. લેટેક્સની લોકપ્રિયતાનાં મુખ્ય કારણોની યાદી તૈયાર કરો.
3. લેટેક્સમાં સીધાં ઉપયોગ ન કરી શકાય તેવા અક્ષરો અને અનામત અક્ષરોની યાદી તૈયાર કરો.
4. લેટેક્સ-દસ્તાવેજનું માળખું સમજાવો.
5. \frac કમાન્ડ અને નેસ્ટેડ \frac કમાન્ડ ઉદાહરણ સહિત સમજાવો.
6. **નીચે આપેલા વિકલ્પોમાંથી યોગ્ય વિકલ્પ પસંદ કરો :**

   (1) આધુનિક શબ્દપ્રક્રિયા સૉફ્ટવેર નીચેનામાંથી કઈ પદ્ધતિ પ્રમાણે ચાલે છે ?

   (a) WIGIWIS (b) WISYWIG (c) WYSIWYG (d) WISYWYG

   (2) નીચેનામાંથી કયો અનામત અક્ષર લેટેક્સમાં નથી ?

   (a) @ (b) % (c) $ (d) ^

(3) નીચેનામાંથી કયું \begin{name}થી શરૂ થાય છે અને \end{name}થી પૂરું થાય છે ?

(a) જૂથ (b) મુદ્દો (c) એન્વાયર્નમેન્ટ (d) પ્રસ્તાવના

(4) નીચેનામાંથી કયો અક્ષર લેટેક્સમાં કૉમેન્ટ (ટિપ્પણી) માટે વપરાય છે ?

(a) $ (b) % (c) # (d) &

(5) નીચેનામાંથી કયો ભાગ લેટેક્સમાં મેટાડેટા ધરાવે છે ?

(a) પ્રાસ્તાવિક ભાગ (b) TOC (c) પ્રસ્તાવના (d) એન્વાયર્નમેન્ટ

(6) નીચેનામાંથી કઈ વેબસાઇટ લેટેક્સનાં પૅકેજ ધરાવે છે ?

(a) CTAN (b) CLAN (c) CTEN (d) CLEN

(7) નીચેનામાંથી કયું આપમેળે અનુક્રમનંબર આપતું નથી ?

(a) \section (b) \subsection (c) \chapter* (d) \part

(8) નીચેનામાંથી કયું એન્વાયર્નમેન્ટ લખાણની લીટીમાં ગાણિતિક વિષયવસ્તુ દર્શાવે છે ?

(a) displaymath (b) math (c) equation (d) text

(9) નીચેનામાંથી કયો કમાન્ડ સમૂહ-યુનિયનનું ચિહ્ન બનાવે છે ?

(a) \cup (b) \setunion (c) \cap (d) \union

(10) અનુક્રમણિકા અથવા સબસ્ક્રિપ્ટ દર્શાવવા માટે નીચેનામાંથી કયો પ્રક્રિયક વપરાય છે ?

(a) _ (અન્ડરસ્કોર) (b) ^ (કૅરેટ) (c) – (બાદબાકી) (d) < (ની કરતાં ઓછું)

(11) નીચેનામાંથી કયું પ્રસ્તાવનામાં પ્રથમ લીટી તરીકે વપરાય છે ?

(a) \usepackage (b) \title (c) \maketitle (d) \documentclass

## પ્રાયોગિક સ્વાધ્યાય

1. લેટેક્સમાં આ પુસ્તકના માળખાની રચના કરો. તેમાં પ્રસ્તાવના, વિષયવસ્તુનું કોષ્ટક, પ્રકરણ, મુદ્દાઓ, પેટા મુદ્દાઓ વગેરેનો સમાવેશ કરો. દરેક વિભાગમાં થોડાક શબ્દો લખો.
2. લખાણ-ગોઠવણ (ફૉર્મેટિંગ) દર્શાવતો લેટેક્સ દસ્તાવેજ બનાવો.
3. ફકરાની ગોઠવણ (ફૉર્મેટિંગ) દર્શાવતો એક લેટેક્સ-દસ્તાવેજ તૈયાર કરો.
4. એક લેટેક્સ-દસ્તાવેજ બનાવી પૃષ્ઠ લેઆઉટ સાથે તેનો પ્રયોગ કરો.
5. નીચે આપેલ વિષયવસ્તુનો એક લેટેક્સ-દસ્તાવેજ તૈયાર કરો :

(a) If A = {x | x ∈ N, x ≤ 4}, B = {-1, 0, 1, 2, 3} and C = {0, 1, 2}, then (A ∪ B) ∩ (A ∪ C) = {0, 1, 2, 3, 4}.

(b) If A = {x | x ∈ N, x ≤ 7} and B = {2, 4, 6} then B ⊂ A.

(c) $\frac{(81)^{\frac{1}{4}}}{(625)^{\frac{1}{4}}} + \frac{(216)^{\frac{1}{3}}}{(8)^{\frac{1}{3}}} - (729)^{\frac{1}{6}}$

(d) X ⊂ Y and Y ⊂ X ⇒ X = Y

(e) $\overrightarrow{PQ} \cap \overrightarrow{PR} = P$

(f) $BE \cap EG = \{E\}$

(g) $\sqrt{s(s-a)(s-b)(s-c)}$

(h) $Areaof \odot(O,r) = \pi r^2$

(i) $\sqrt{a+2\sqrt{b}} = \sqrt{x} + \sqrt{y}$

(j) $x = \frac{a+\sqrt{a^2-4b}}{2}, y = \frac{a-\sqrt{a^2-4b}}{2}$

(k) $\frac{h(\tan\alpha - \tan\beta)}{\tan\alpha\tan\beta}$

# અન્ય ઉપયોગી નિઃશુલ્ક ટૂલ્સ અને સેવાઓ 13

આ પ્રકરણમાં આપણે કેટલાંક નિઃશુલ્ક ટૂલ્સ અને સેવાઓની ચર્ચા કરીશું. અહીં આપણે ઝડપથી માહિતી-સંકોચન તકનિકનો પરિચય મેળવીશું. આ તકનિક વડે આપણે માહિતી સંગ્રહ કરવા માટે સ્મૃતિસંચય (Memory) અને તક્તિ (Disk) ઉપરની જગ્યામાં ઘટાડો કરી શકીએ છીએ. અહીં આપણે ફાઇલ અને ડિરેક્ટરી ઉપર આ તકનિકનો ઉપયોગ કરવા માટેના આર્ચિવ મૅનેજર ટૂલ વિશે ચર્ચા કરીશું. એક પ્રતિષ્ઠિત મલ્ટિમીડિયા ટૂલ, VLC મીડિયા પ્લેયરની ટૂંકમાં ચર્ચા કરીશું. અહીં આપણે ગુગલમૅપની સેવાની પણ ચર્ચા કરીશું, જે આપણને જુદી જુદી જગ્યાના નકશાઓ પૂરી પાડતી ઑનલાઇન સેવા છે. આપણે બે નાના પણ ઉપયોગી વિનિયોગ (Application) જોઈશું. કૅરેક્ટરમૅપ, જે લખાણમાં આપણને જુદી-જુદી ભાષાનાં ચિહ્નો અને અક્ષરો ઉમેરવા આપે છે. ત્યાર પછી આપણે આંકડાકીય ગણતરી માટેનું 'આર' (R) એન્વાયરમેન્ટનું સામાન્ય નિરીક્ષણ કરીશું, જેમાં આપણે સામાન્ય ગાણિતિક પ્રક્રિયાઓ જેવી કે મધ્યક અને મધ્યસ્થ શોધવા વિશે ભણીશું. તેમજ Rમાં બારચાર્ટ અને હિસ્ટોગ્રામ દોરતા શીખીશું. છેલ્લે આપણે બે ટૂલ્સ, રેશનલ પ્લાન અને સ્કાઇપ જોઈશું. આમાંના કેટલાક વિનિયોગ (Application) ઉબન્ટુ 10.04 LTSમાં પહેલેથી જ પ્રસ્થાપિત કરેલ હોય છે અને કેટલાક ઉબુન્ટુના ભંડાર (Repositories)માંથી પ્રસ્થાપિત કરવાની જરૂર પડે છે.

## માહિતી-સંકોચન (Data Compression)

અદ્યતન કમ્પ્યૂટર સિસ્ટમમાં લાખો ફાઇલો હોય છે. ઘણી વખત એક કમ્પ્યૂટરમાંથી બીજા કમ્પ્યૂટર અથવા સંગ્રહ ઉપકરણમાં ઘણી બધી ફાઇલો અથવા આખી ડિરેક્ટરી પરિવહન કરવાની જરૂર પડે છે. જ્યારે આપણને આવી કમ્પ્યૂટરફાઇલ (કે જેમાં માહિતી, પ્રોગ્રામ અથવા બંને હોય) એક કમ્પ્યૂટરમાંથી બીજા કમ્પ્યૂટર અથવા સંગ્રહ-ઉપકરણમાં સ્થાનાંતરિત કરવાની જરૂર પડે, ત્યારે સ્થાનાંતરિત અથવા સંગ્રહ થતો માહિતીનો જથ્થો ચિંતાનો વિષય બની શકે છે. કમ્પ્યૂટર નેટવર્ક અને બાહ્ય સંગ્રહ ઉપકરણો બંને સામાન્ય રીતે કમ્પ્યૂટરના આંતરિક ઘટકો જેટલા ઝડપી નથી. આમ, મોટા જથ્થામાં માહિતીનું પરિવહન વધુ સમય લે છે. જો પરિવહન માટે ઇન્ટરનેટનો ઉપયોગ કરવામાં આવે, તો ઇન્ટરનેટની ધીમી ગતિના કારણે એ પરિવહન માટે ઘણો વધુ સમય લે છે. આ પરિવહન ઇન્ટરનેટના જોડાણ ઉપર પણ ભાર મૂકે છે. જો કોઈ વપરાશકર્તા અથવા સંસ્થા અમર્યાદિત ઇન્ટરનેટ યોજના ન ધરાવતા હોય, તો મોટા જથ્થાની માહિતીનું પરિવહન ઊંચી કિંમતમાં પરિણમે છે.

એવી જ રીતે ફાઇલ અને ડિરેક્ટરીઓનું સંગ્રહ-ઉપકરણમાં પરિવહન કરવામાં આવે, તો સંગ્રહ-ઉપકરણની મર્યાદિત ક્ષમતા અને વિવિધ ઉપયોગના કારણે પરિવહન કરવાનો માહિતીજથ્થો ફરીથી એક સમસ્યા બની જાય છે. આ સમસ્યાને ધ્યાનમાં લઈને શક્ય હોય ત્યાં કમ્પ્યૂટર ફાઇલ અને આખી ડિરેક્ટરી દ્વારા રોકવામાં આવતી જગ્યાને ઘટાડવાની જરૂર પડે છે. અનુકૂળતાની દૃષ્ટિએ, ઘણા બધા કિસ્સામાં ફાઇલના વિશાળ જથ્થા અથવા જટિલ ડિરેક્ટરી બંધારણ કરતાં એક અલગ ફાઇલને નિયંત્રિત કરવી ઇચ્છનીય છે.

કમ્પ્યૂટર વૈજ્ઞાનિકોએ આખી ડિરેક્ટરીને એક ફાઇલમાં મૂકવા માટેની તકનિક વિકસાવી છે. આવી ફાઇલને 'આર્ચિવ' ફાઇલ તરીકે ઓળખવામાં આવે છે. તેઓએ કમ્પ્યૂટર ફાઇલ અને ડિરેક્ટરી માટેની સંગ્રહસ્થાનની જરૂરિયાતો ઘટાડવા માટે ઘણી બધી તકનિકો વિકસાવી છે. આ તકનિકોને માહિતી-સંકોચન કહેવામાં આવે છે.

સામાન્ય રીતે માહિતી-સંકોચન, માહિતીના સંકેતલેખન દ્વારા માહિતીના પુનરાવર્તનને ઓળખવા અને આવા પુનરાવર્તનને ઘટાડવા અથવા દૂર કરવાનું કામ કરે છે. આવી કેટલીક તકનિકો, જગ્યાસંરક્ષણ માટે ઓછી મહત્ત્વની માહિતીને ઓળખે છે અને તેને દૂર કરે છે. ઉદાહરણ તરીકે આકૃતિ 13.1માં બતાવ્યા પ્રમાણે બાળકોની કવિતા ધ્યાને લો. તેમાં ચોક્કસપણે

શબ્દોનાં ઘણાં પુનરાવર્તન છે. એમાં આ પ્રક્રિયાનો લાભ લેવા માટે દરેક શબ્દને એક અંક અથવા અક્ષર દ્વારા રજૂ કરી શકાય. સાંકેતિક લિપિમાં લખાયેલી ફાઇલની શરૂઆતમાં ટેબલની અંદર દરેક અંક / અક્ષર ક્યા શબ્દને રજૂ કરે છે, તે માહિતી દર્શાવવામાં આવે છે. (આ માહિતી ફાઇલને તેના મૂળ સ્વરૂપમાં ફેરવવા માટે જરૂરી છે.) ત્યાર પછી એક શબ્દ માટે એક અંક / અક્ષરનો ઉપયોગ કરવામાં આવે છે. આ રીતે, લાંબા શબ્દો ફક્ત એક વખત અને ત્યાર પછી તે શબ્દોને એક અક્ષર વડે દર્શાવવામાં આવે છે અને ચિહ્નો જેમ છે તેમજ છોડી દેવામાં આવે છે. અહીં ટેબલની શરૂઆતમાં ^ (કેરેટ) ચિહ્ન અને અંતમાં $ ચિહ્નનું નિશાન કરવામાં આવેલ છે. આકૃતિ 13.1ના બીજા ભાગમાં સાંકેતિક લિપિમાં લખેલ ફાઇલ પણ દર્શાવવામાં આવેલ છે.

One little, two little, three little Indians
Four little, five little, six little Indians
Seven little, eight little, nine little Indians
Ten little Indian boys.

Ten little, nine little, eight little Indians
Seven little, six little, five little Indians
Four little, three little, two little Indians
One little Indian boy.

Actual Contents

---

^0:One 1:little 2:two 3:three 4:Indians 5:Four
6:five 7:six 8:Seven 9:eight A:nine B:Ten
C:Indian D:boys E:boyS0 1, 2 1, 3 1 4
5 1, 6 1, 7 1 4
8 1, 9 1, A 1 4
B 1 C D.

B 1, A 1, 9 1 4
8 1, 7 1, 6 1 4
5 1, 3 1, 2 1 4
0 1 C E.

Encoded Contents

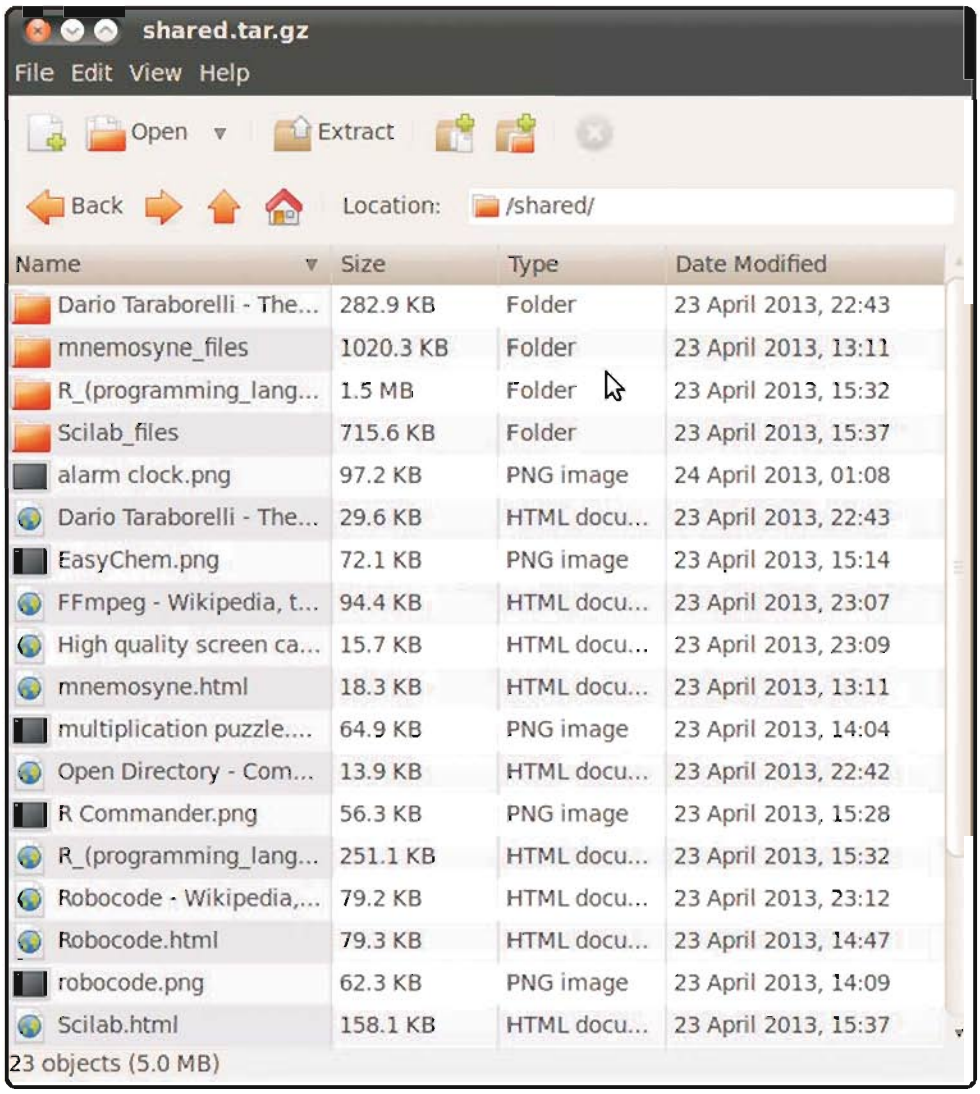
**આકૃતિ 13.1 : માહિતી-સંકોચનનું ઉદાહરણ**

મૂળ ફાઇલનું કદ 324 બાઇટ હતું અને સાંકેતિક લિપિમાં ફેરવેલ ફાઇલનું કદ ફક્ત 226 બાઇટ થયું. આમ થવાનું કારણ દરેક શબ્દ પૂરેપૂરો એક જ વખત વાપરવામાં આવ્યો અને ત્યાર પછી તે શબ્દને ફક્ત એક અક્ષર વડે દર્શાવવામાં આવ્યો. ઊલટી પ્રક્રિયા વડે સાંકેતિક લિપિમાં લખેલ ફાઇલને ગમે ત્યારે તેના મૂળ રૂપમાં ફેરવી શકાય છે.

અહીં આપેલી પદ્ધતિ એકદમ સરળ છે. મૂળ માહિતી-સંકોચનનો પ્રોગ્રામ વ્યવહારુ-અલ્ગોરિધમનો ઉપયોગ કરે છે, જે ફાઇલના કદમાં મોટા પ્રમાણમાં ઘટાડો કરે છે અને તે માહિતીના સિદ્ધાંત ઉપર આધારિત છે. સમૃદ્ધ લિનક્સ આર્ચિવના સંચાલન માટે તૈયાર નિઃશુલ્ક અને ઓપનસોર્સ સૉફ્ટવેર પૂરાં પાડે છે, જેને આર્ચિવ મૅનેજર તરીકે ઓળખવામાં આવે છે.

## આર્ચિવ મૅનેજર (Archive Manager)

આકૃતિ 13.2માં દર્શાવ્યા પ્રમાણે આર્ચિવ મૅનેજર એક કરતાં વધુ ફાઇલ અથવા આખી ડિરેક્ટરીને એક ફાઇલમાં જોડી દે છે, જે આર્ચિવ (ફાઇલોનો ભંડાર) તરીકે ઓળખાય છે. લિનક્સમાં TAR (ટેપ આર્ચિવર) એ સર્વસામાન્ય આર્ચિવ-માળખું છે. તે આર્ચિવને સંકોચનની સગવડ પણ પૂરી પાડે છે, જે તેની ફાઇલનું કદ ઘટાડે છે. તે એક કરતાં વધુ સંકોચન અલ્ગોરિધમ અને સંકોચિત ફાઇલમાળખાને સમર્થન આપે છે. ઝીપ (zip) ફાઇલમાળખું અને tar.gz ફાઇલમાળખું એ સર્વસામાન્ય સંકુચિત ફાઇલમાળખું છે. જ્યારે ઝિપ ફાઇલમાળખું એક કરતાં વધુ ફાઇલને એક ઝીપ ફાઇલમાં સંગ્રહ કરે છે. યુનિક્સ / લિનક્સ પદ્ધતિમાં સામાન્ય રીતે એક કરતાં વધુ ફાઇલોને એક ટાર (tar) માળખાની સંકોચનરહિત ફાઇલમાં ભેગી કરવામાં આવે છે અને ત્યાર બાદ gzip પ્રોગ્રામ (GNU ઝીપ, એક ઓપનસૉર્સ આર્ચિવ પ્રોગ્રામ છે.)નો ઉપયોગ કરીને ઝીપમાળખાની ફાઇલમાં સંકોચન કરવામાં આવે છે. આવી ફાઇલને સામાન્ય રીતે tar.gz અનુલંબન હોય છે અને તેને ટારબૉલ (Tar Ball) તરીકે ઓળખવામાં આવે છે. લિનક્સમાં ફાઇલોના જથ્થાને અથવા સૉફ્ટવેરને વિતરણ કરવા માટે 'ટારબૉલ' એ ખૂબ જ સામાન્ય માળખું છે. ઝીપમાળખું પણ જુદાં-જુદાં નામ હેઠળ જુદા-જુદા વિનિયોગ માટે ઉપયોગમાં આવે છે. ઉદાહરણ તરીકે જાવા (JAVA)માં JAR ફાઇલ અને ઑફિસમાં OpenOffice.org ફાઇલમાળખું ઝીપસંકોચનનો ઉપયોગ કરે છે. નવી આર્ચિવ બનાવવા માટે અને આર્ચિવ ખોલવા તેમજ ફાઇલમાળખાં પસંદ કરવા માટે આર્ચિવ મૅનેજર ફાઇલ નામનો અનુલંબન ભાગ ફાઇલમાળખાને ઓળખવા માટે ઉપયોગમાં લે છે.




આકૃતિ 13.2 : આર્ચિવ મૅનેજર

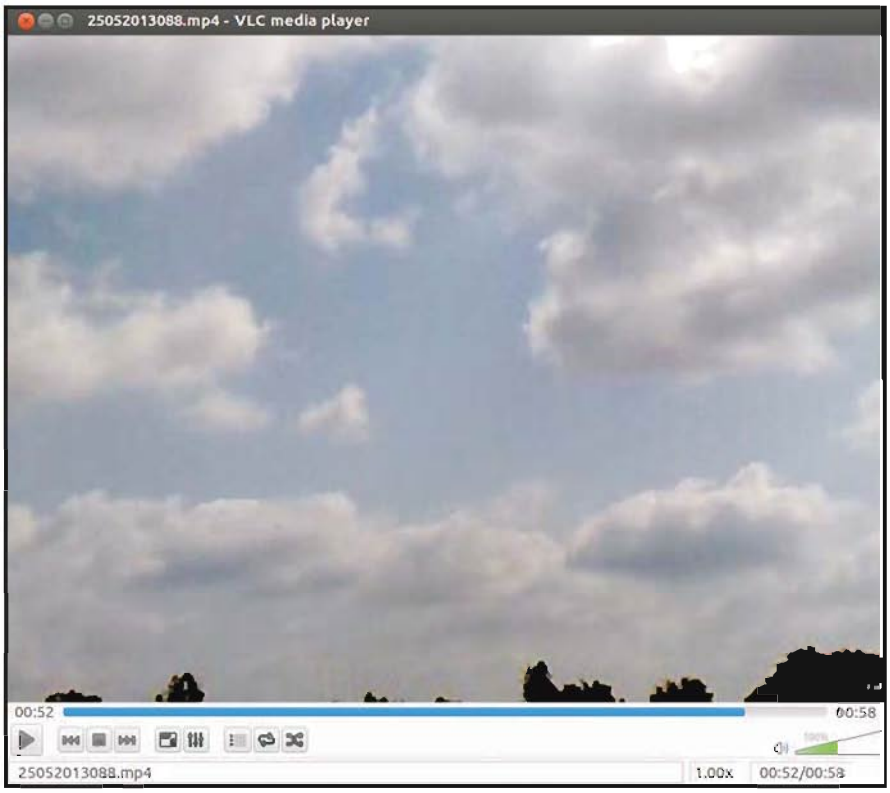
આર્ચિવ ફાઇલની વિષયવસ્તુના અન્વેષણ માટે, આર્ચિવમાંથી ફાઇલોને બહાર કાઢવા માટે, આર્ચિવમાં નવી ફાઇલોને ઉમેરવા માટે, આર્ચિવમાંથી ફાઇલોને દૂર કરવા માટે અને બીજાં કેટલાંક કાર્ય કરવા માટે પણ આર્ચિવ મૅનેજરનો ઉપયોગ થાય છે. સામાન્ય રીતે આર્ચિવ મૅનેજર શરૂ કરવા માટે કોઈ મેનુવિકલ્પ નથી. ફાઇલબ્રાઉઝરમાં આર્ચિવ ફાઇલ ઉપર ડબલ ક્લિક કરવાથી આર્ચિવ મૅનેજર શરૂ થાય છે. ફાઇલ અથવા ડિરેક્ટરી ઉપર જમણી (રાઇટ) ક્લિક કરી મેનુમાંથી Compress વિકલ્પ પસંદ કરીને નવી આર્ચિવ ફાઇલ બનાવી શકાય છે. જે આ ફાઇલ અથવા ડિરેક્ટરીના વિષયવસ્તુની આર્ચિવ બનાવે છે. આર્ચિવ ફાઇલ માટે આ વિકલ્પ ઉપલબ્ધ હોતો નથી. જો આપણે ઇચ્છીએ તો Application મેનુના પેટામેનુ Accessoriesમાં આર્ચિવ મૅનેજરનો ઉમેરો કરી શકીએ. આવું કરવા માટે ઉપરના ડાબા ખૂણાના ઉબુન્ટુ આઇકન ઉપર જમણી (રાઇટ) ક્લિક કરો. લિસ્ટમાંથી Edit મેનુ પસંદ કરો અને ડાયલોગબૉક્સમાં એસેસરીઝ ઉપર ક્લિક કરો. આર્ચિવ મૅનેજર માટેના ચેકબૉક્સ પસંદ કરીને ડાયલોગબૉક્સ બંધ કરો.

જ્યારે આર્ચિવ મૅનેજરમાં આર્ચિવ ખૂલેલી હોય ત્યારે આકૃતિ 13.2માં દર્શાવ્યા મુજબ ફાઇલબ્રાઉઝરમાં ખોલેલી ફાઇલ સમાન દેખાય છે. તેમનાં કાર્ય અને કી-બોર્ડના ટૂંકા કમાન્ડ (Shortcuts) પણ સરખા જ છે. ડિરેક્ટરી ઉપર ડબલક્લિક કરવાથી તેજ ડિરેક્ટરી ખૂલે છે અને તેમાં રહેલી વિષયવસ્તુ દૃશ્યમાન થાય છે. ટૂલબારમાં રહેલ Up અને Back બટન દ્વારા પૅરેન્ટ ડિરેક્ટરી (જો આર્ચિવમાં ઉપલબ્ધ હોય તો)માં અને પહેલાંની ડિરેક્ટરીમાં (જો હોય તો) જઈ શકાય છે. આમાંનું જ્યારે લાગુ પડે, ત્યારે જ શક્ય બને છે.

આ રીતે, આર્ચિવ મૅનેજર આપણને આર્ચિવ ફાઇલનો ઉપયોગ કરીને બૅકઅપ લેવા, બાહ્યસંગ્રહ ઉપકરણ વડે સ્થાનાંતરિત કરવા નેટવર્ક વડે સ્થાનાંતરિત કરવા, તક્તી (Disk) ઉપરની જગ્યા બચાવવા વગેરે માટે ઉપયોગી છે.

## VLC મીડિયાપ્લેયર (The VLC Media Player)

આપણે આગળ ચર્ચા કર્યા મુજબ ઉબન્ટુમાં મલ્ટિમીડિયા ઑડિયો, વીડિયો વગેરે (શ્રાવ્ય અને દૃશ્ય) વગાડવા માટેનું ટૂલ આપે છે. તેમ છતાં બીજું મહત્ત્વનું ઓપનસોર્સ ટૂલ VLC (મૂળ સ્વરૂપે તે VideoLAN Clientનું ટૂંકાક્ષરી) મીડિયાપ્લેયર આકૃતિ 13.3માં દર્શાવ્યું છે. તે તેની લાક્ષણિકતા અને સાર્વત્રિકતાને કારણે ખૂબ જ પ્રખ્યાત છે.




આકૃતિ 13.3 : VLC મીડિયાપ્લેયર

પૅરિસમાં વિશ્વવિદ્યાલયના સંગીતપ્રિય વિદ્યાર્થીઓએ અભ્યાસલક્ષી પ્રોજેક્ટ (યોજના) માટે શરૂ કરેલ તે હાલમાં જાહેરજનતા માટે ઘણી બધી ઑપરેટિંગ સિસ્ટમમાં ઉપલબ્ધ છે અને કરોડો વખત એ ડાઉનલોડ થયેલ છે. મલ્ટિમીડિયાના વિષયવસ્તુની એક સમસ્યા એ છે કે હાર્ડવેર ઉપકરણ (audio/video streams) પરથી આવતી મલ્ટિમીડિયા માહિતીને કમ્પ્યૂટરની માહિતીમાં ફેરવવા અને કમ્પ્યૂટર ઉપરની માહિતીને હાર્ડવેર ઉપકરણ ઉપર વગાડવા, ફરીથી ઑડિયો/વીડિયોમાં ફેરવવાના ઘણા બધા માર્ગ ઉપલબ્ધ છે.

આ પરિવર્તન (સાંકેતીકરણ) અને ઊલટું પરિવર્તન (અસાંકેતીકરણ) એક સૉફ્ટવેર વડે કરવામાં આવે છે. જેને કોડેક (કોડર-ડીકોડર) તરીકે ઓળખવામાં આવે છે. દરેક મલ્ટિમીડિયા માહિતીના માળખાને પોતાનું કોડેક જરૂરી હોય છે. VLCનો ફાયદો એ છે કે તે દરેક પ્રખ્યાત કોડેક અને દરેક પ્રખ્યાત માળખાને સમર્થન આપે છે. તે મોટા ભાગનાં દરેક ઉપકરણો જેવાં કે વેબકૅમેરા, એચડી મૉનિટર, દરેક પ્રકારનાં સ્પીકર, માઇક્રોફોન, હેડફોન વગેરેને સમર્થન આપે છે. ઑડિયો અને વીડિયો વગાડવા માટે VLC ઘણા બધા વિકલ્પો આપે છે. તે મલ્ટિમીડિયા ફાઇલને એક માળખામાંથી બીજા માળખામાં પરિવર્તિત પણ કરી શકે છે. તે નેટવર્કમાં રહેલ બીજા કમ્પ્યૂટરમાંથી ઑડિયો અને વીડિયો મેળવી પણ શકે છે અને ઑડિયો અને વીડિયો મોકલી પણ શકે છે.

**Applications → Sound & Video** મેનુ વડે VLC શરૂ કર્યા પછી, **Media → Open File...** મેનુવિકલ્પ વડે આપણે એક અથવા વધારે ફાઇલને ખોલી શકીએ છીએ. **Media → Open Directory...** વિકલ્પ વડે આપણે એકસાથે આખી ડિરેક્ટરી ખોલી શકીએ છીએ. જો આપણે એક ડિરેક્ટરી ખોલીએ તો, હકીકતમાં VLC તે ડિરેક્ટરીમાં રહેલી વગાડી શકાય તેવી તમામ મીડિયા ફાઇલને ખોલે છે. જ્યારે એક અથવા એક કરતાં વધારે ફાઇલ ખૂલેલી હોય ત્યારે, તે ફાઇલ પ્લેલિસ્ટમાં ઉમેરાય છે. પ્લેલિસ્ટ એ મીડિયા ફાઇલ વગાડવાની યાદી છે. પ્લેલિસ્ટમાં રહેલ મીડિયાને આપણે અનુક્રમ પ્રમાણે અથવા આડા-અવળા ક્રમમાં વગાડી શકીએ છીએ. આપણે આગળ અથવા પાછળના માર્ગ (TRACK) પર જઈ શકીએ છીએ. બીજી વખત આજ ટ્રેક વગાડવા માટે આપણે પ્લેલિસ્ટને સેવ (SAVE) પણ કરી શકીએ છીએ. **View → Playlist** વિકલ્પની મદદ વડે આપણે પ્લેલિસ્ટને ખોલી શકીએ છીએ અને **Media → Save Playlist to File** વિકલ્પ વડે પ્લેલિસ્ટ સેવ કરી શકીએ છીએ. VLC પ્લેલિસ્ટના ઘણાં બધાં માળખાંને સમર્થન આપે છે પણ M3U માળખું એ સર્વસામાન્ય માળખું છે.

VLC મીડિયાપ્લેયર વિન્ડોના નીચેના ભાગમાં એક પ્રગતિ લીટી (પ્રોગ્રેસ બાર) દર્શાવવામાં આવે છે, જે ચાલુ ટ્રેકનો કુલ સમય અને ટ્રેક કેટલો વાગી ચૂક્યો છે તે સમય દર્શાવે છે. બાર ઉપર રહેલા નાના એવા સ્લાઇડરને સરકાવવામાં આવે, તો ચાલુ ટ્રેકમાં આપણે આગળ અને પાછળ હરીફરી શકીએ છીએ. તેની નીચે પ્લેબટન હોય છે. (જમણી તરફ દર્શાવેલું ત્રિકોણ જેવું નિશાન) જ્યારે ટ્રેક વાગતો હોય, ત્યારે આ બટન પોઝબટન (અટકવા માટેનું બટન) બની જાય છે. (બે સમાંતર ઊભી લીટી જેવું નિશાન). જ્યારે આપણે વગાડવું અટકાવી દઈએ, ત્યારે પાછું આ બટન પ્લેમાં ફેરવાય જાય છે. ત્યાર પછીનાં ત્રણ બટનમાંનું વચ્ચેનું બટન એ ટ્રેકને વગાડાતો સંપૂર્ણ બંધ કરવા માટેનું છે. જ્યારે ડાબી અને જમણી તરફના ડબલ તીરવાળાં બટન આપણને આગળ અને પાછળના ટ્રેક ઉપર લઈ જાય છે. ત્યાર પછીનું બટન વીડિયોને વિન્ડોમાં જોવા અથવા આખી સ્ક્રીનમાં જોવાના વિકલ્પમાં ફેરવવા માટેનું કાર્ય કરે છે. આનો મતલબ એ થયો કે જ્યારે આપણે બટન ઉપર ક્લિક કરીએ, ત્યારે તે આપણને એક સ્થિતિમાંથી બીજી સ્થિતિમાં ફેરવે છે. આખી સ્ક્રીનમાં વીડિયો જોતી વખતે ચાલુ કરવા, અટકાવવા કે બંધ કરવા માટેનાં નિયમન અદશ્ય હોય છે. વીડિયો ઉપર માઉસકર્સરને ફેરવવાથી કામચલાઉ રીતે અસ્થાયી વિન્ડોમાં તેને જોઈ શકાય છે. ત્યાર પછીનું બટન એ પ્લેલિસ્ટ દર્શાવવા માટેનું છે. આ બધા જ વિકલ્પ મેનુમાં પણ ઉપલબ્ધ છે.

VLCમાં બીજી ઘણી બધી કાર્યપ્રણાલી છે, અહીં આપણે મલ્ટિમીડિયા ફાઇલને એક માળખામાંથી બીજા માળખામાં પરિવર્તિત કરવા માટે VLC નો કઈ રીતે ઉપયોગ થાય, તેની ચર્ચા કરીશું. આ માટે મેનુમાંથી **Media → Convert / Save** વિકલ્પ પસંદ કરો. એડ (Add) બટનનો ઉપયોગ કરી, જે ફાઇલને પરિવર્તિત કરવાની છે, તેને પસંદ કરો અને ઉમેરો. **Convert / Save** બટનની બાજુની યાદીમાંથી પરિવર્તિત (Convert) વિકલ્પ પસંદ કરો. આથી એક અન્ય ડાયલોગબૉક્સ ખુલશે. અહીં, આપણે નિર્ધારિત ફાઇલનું નામ પૂરું પાડવું પડશે. (જેમાં પરિવર્તિત ટ્રેકને સેવ કરવાનો છે તે). આપણે, આપેલ ડ્રૉપડાઉનમાંથી ઇચ્છિત ફાઇલમાળખું (ઑડિયો અથવા વીડિયો) પસંદ કરવું જરૂરી રહેશે. સ્ટાર્ટ (શરૂ) બટન ઉપર ક્લિક કરવાથી પ્રક્રિયા શરૂ થશે અને પ્રગતિ દર્શાવવામાં આવશે. ક્યારેક આપણને પરિવર્તનની પ્રક્રિયામાં ભૂલનો સંદેશ પણ મળે છે.

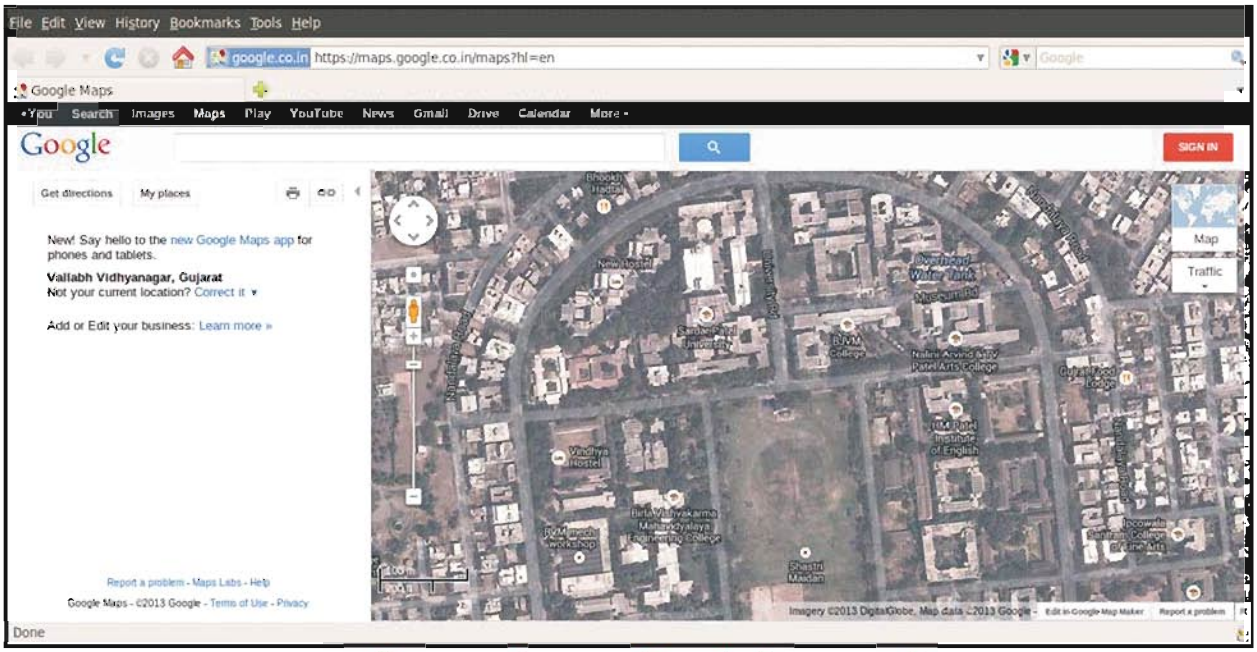
## ગુગલ નકશો (Google Map)

આકૃતિ 13.4 માં દર્શાવેલ ગુગલ નકશો એ ગુગલ ઇનકોર્પોરેશનની ઇન્ટરનેટ આધારિત નિઃશુલ્ક સેવા છે. ગુગલે ઉપગ્રહની મદદ વડે આકૃતિ મેળવીને, કાર ઉપર કેમેરા બાંધીને, બીજી સંસ્થાઓ પાસેથી માહિતી ખરીદીને અને દુનિયાના હજારો વ્યક્તિગત ઉપયોગકર્તાઓ પાસેથી માહિતી મેળવીને વર્ષોના સમયગાળા પછી આખી પૃથ્વીના વિશાળ નકશાની માહિતી ભેગી કરી છે. ગુગલ નકશો કોઈ પણ વ્યક્તિને નકશામાં ફેરફાર કરવા અને જમીનની નિશાની, ઇમારત વગેરે ઓળખવા માટે પરવાનગી આપે છે. તે કોઈ પણ જગ્યાનો ફોટોગ્રાફ અપલોડ કરવા અને કોઈ પણ જગ્યાની પરીક્ષણ-માહિતી આપવાની પરવાનગી પણ આપે છે. આ સેવાનો ઉપયોગ હરતાંફરતાં ઉપકરણો જેવા કે સ્માર્ટફોન અને ટેબ્લેટમાં ખૂબ જ થાય છે.


આકૃતિ 13.4 : ગુગલ નકશાની સેવા

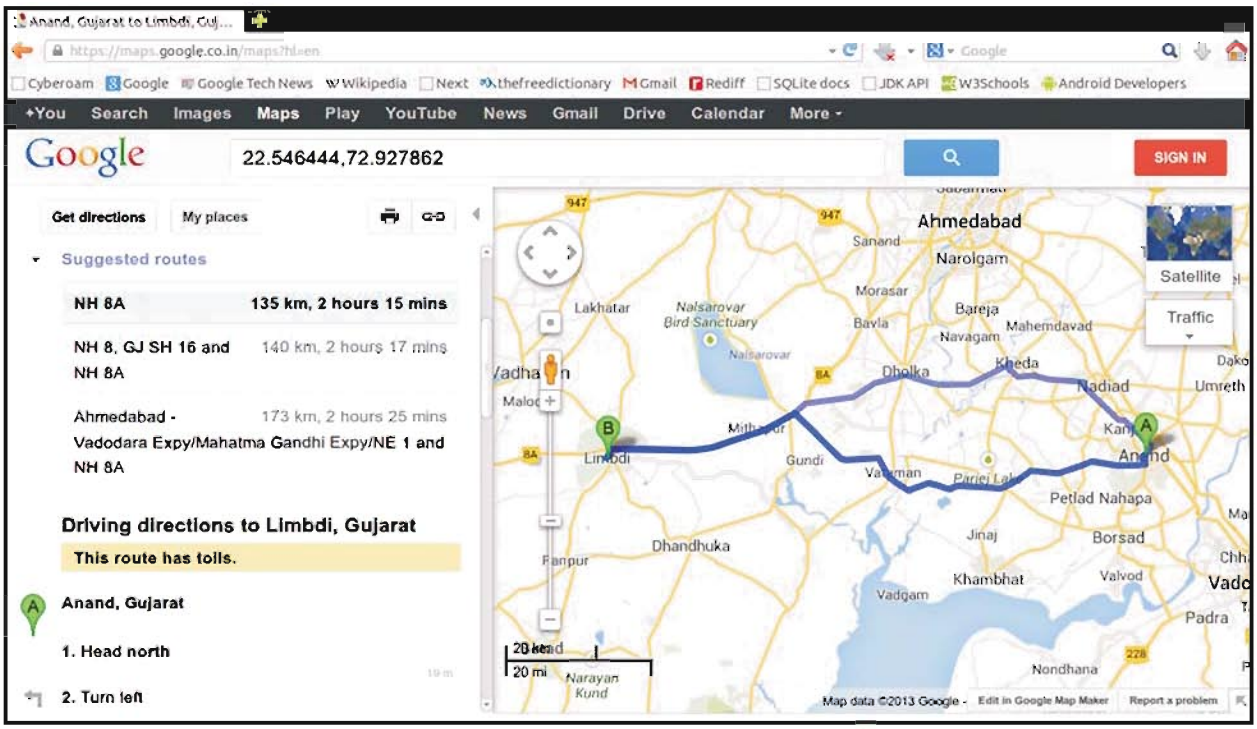
આ સેવા વેબબ્રાઉઝરમાં http://www.google.co.in વેબસાઇટ ખોલીને ઉપરના મેનુમાંથી મૅપ (નકશાઓ) વિકલ્પ પસંદ કરીને મેળવી શકાય છે અથવા સીધેસીધું http://maps.google.co.in. આપીને પણ મેળવી શકાય છે. મોબાઇલ ફોનમાં આ સેવા મોબાઇલ ફોનના વેબબ્રાઉઝર વડે તદ્ઉપરાંત ગુગલ નકશાના વિનિયોગ (Application) વડે મેળવી શકાય છે. જ્યારે આ સેવા આપણે ચાલુ કરીએ છીએ, ત્યારે તે પ્રથમ આપણા હાલના વિસ્તારને જાણવાનો પ્રયત્ન કરે છે. જ્યારે કમ્પ્યૂટર ઉપર તે ઉપયોગકર્તાના વિસ્તારનો ખ્યાલ તે ઉપયોગકર્તાના ઇન્ટરનેટના જોડાણ ઉપરથી મેળવે છે. મોબાઇલ ફોન ઉપર, GPS (Global Positioning System) સગવડનો ઉપયોગ હોય, તો ઉપયોગકર્તાના વિસ્તારનો ખ્યાલ ખૂબ જ ચોકસાઈથી મળે છે. આ પદ્ધતિ ઉપગ્રહોના તરંગોનો ઉપયોગ કરીને અંદાજિત થોડા મીટરની ભૂલથી ઉપયોગકર્તાનો વિસ્તાર શોધી કાઢે છે. ગુગલ નકશો જો તે અચોક્કસ હોય, તો વિસ્તારને ચોક્કસ કરવાની / બરાબર કરવાની પરવાનગી આપે છે અને પછી તે હાલના વિસ્તારનો નકશો દર્શાવે છે. આપણે નકશાને જુદી દિશામાં ઘસડીને અથવા ડાબી બાજુના વર્તુળમાં રહેલ તીરના નિશાન ઉપર ક્લિક કરીને આમતેમ ફેરવી શકીશું. ડાબી બાજુનું ઊભું સ્લાઇડરબાર આપણને નકશાને દૂર લઈ જવા અને નજીક લાવવાની પરવાનગી આપે છે.

ગુગલ નકશાઓ કોઈ એક જગ્યાની માહિતી જુદાં જુદાં સ્વરૂપે આપી શકે છે. આકૃતિ 13.4માં દર્શાવ્યા પ્રમાણે તેનો સામાન્ય દેખાવ એ નકશો (Map View) છે. સેટેલાઇટ (ઉપગ્રહ) બટન ઉપર ક્લિક કરવાથી આકૃતિ 13.5માં દર્શાવ્યા પ્રમાણે આપણા આ દેખાવને ઉપગ્રહ ચિત્રના દેખાવમાં ફેરવી શકીએ છીએ. આ દેખાવમાં આકાશના ઉપગ્રહ દ્વારા લેવાયેલ તસવીર દર્શાવવામાં આવી છે. આ તસવીર પરિચિત ઇમારતો અને રસ્તાઓ ઓળખવા માટે પર્યાપ્ત અને સ્પષ્ટ હોય છે. આ સેવા એક જગ્યા Aથી બીજી જગ્યા B સુધી પહોંચવા માટેનો દિશાનિર્દેશ પણ કરે છે. આ બે જગ્યાઓ બે જુદાં-જુદાં શહેર અથવા એક શહેરનાં જુદાં-જુદાં સ્થળ પણ હોઈ શકે. આ સેવા જ્યાં શક્ય હોય, ત્યાં રસ્તાઓની પસંદગી પણ પૂરી પાડે છે. આકૃતિ 13.6 ઉદાહરણ પૂરું પાડે છે. જો આપણે GPS રિસીવરવાળું મોબાઇલ ઉપકરણ વાપરીએ તો મુસાફરી દરમિયાન આ સેવા આપણને (એક વળાંકથી બીજા વળાંક) એક તરફથી બીજી તરફ જવાનું માર્ગદર્શન પણ પૂરું પાડે છે. રસ્તા ઉપર અથવા ગીચ શહેરી વિસ્તારમાં ચાલતાં અથવા વાહન ચલાવતા આ સેવા હાલની જગ્યાએથી નિશ્ચિત જગ્યા સુધી પહોંચવાનો રસ્તો નક્કી કરી આપે છે અને આપણને સ્ક્રીન ઉપર અથવા અવાજ વડે (બોલીને સૂચના આપીને) માર્ગદર્શન આપે છે.




આકૃતિ 13.5 : ઉપગ્રહ તસવીર દર્શાવતો ગુગલ નકશો

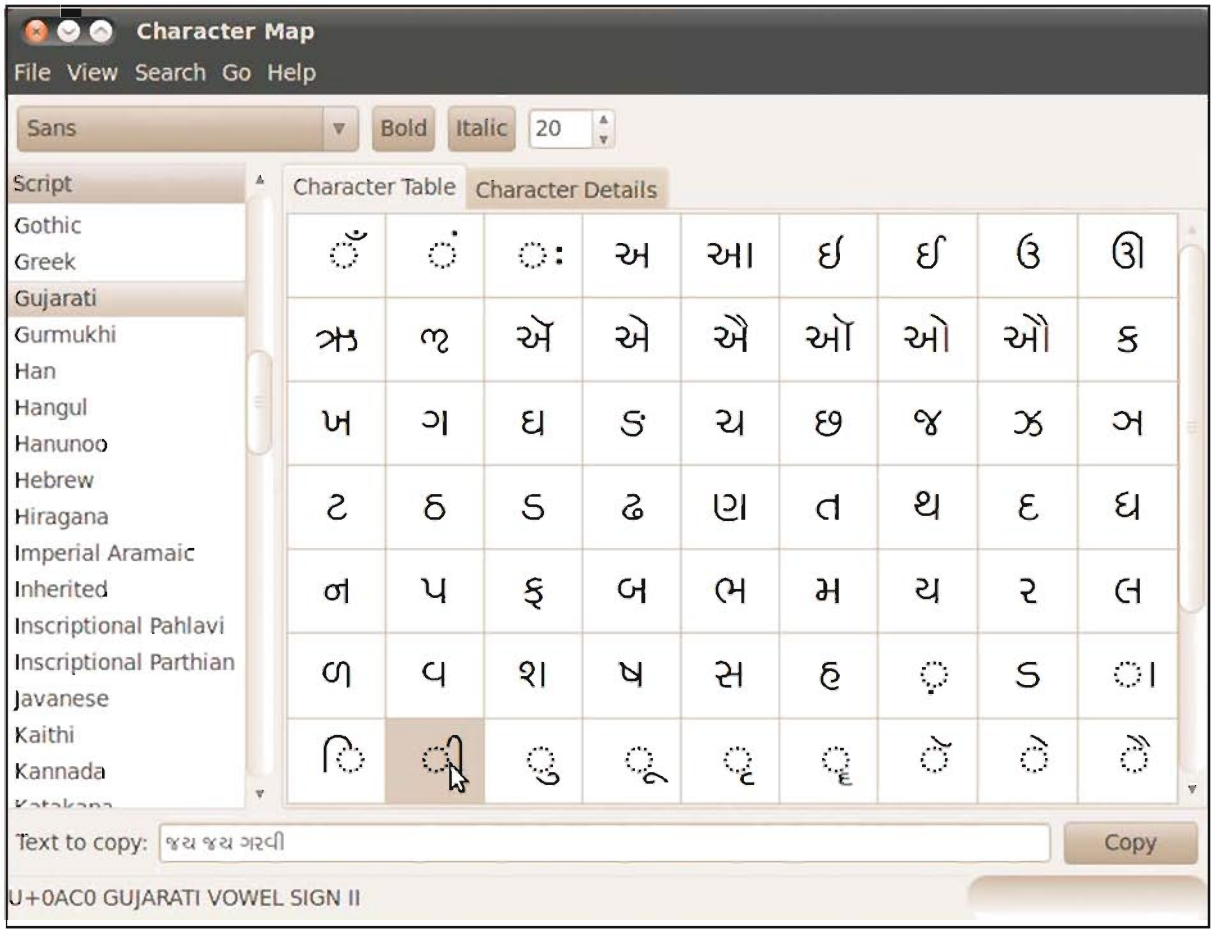
આ સેવાના જુદા-જુદા ઘણા ઉપયોગ છે. આ સેવાનો ઉપયોગ સ્થળની ચોક્કસ જગ્યા શોધવા માટે પણ થાય છે. આ સેવાનો ઉપયોગ કરી આપણે જે શહેરથી પરિચિત ન હોઈએ, તેવા શહેર અથવા અજાણી જગ્યા પર પહોંચવા માટેનો દિશાનિર્દેશ પણ મેળવી શકીએ છીએ. સરકારી સંસ્થાઓ અને વેપારીઓ, લોકોને પોતાની ઑફિસે કઈ રીતે પહોંચવું તેની માહિતી પણ આ સેવા વડે પૂરી પાડે છે. આ સેવાનો ઉપયોગ કરીને કોઈ પણ વ્યક્તિ કોઈ પણ જગ્યાનો નકશો વેબસાઇટમાં દર્શાવી શકે છે. ઘણી બધી સંસ્થાઓ આવા નકશાઓ પોતાની વેબસાઇટમાં દર્શાવે છે. પ્રવાસીની માહિતી દર્શાવતી વેબસાઇટ અને સરકારનો પ્રવાસન વિભાગ પણ આ સેવાનો ઉપયોગ કરે છે. બસનો માર્ગ દર્શાવવા, ચાલુ ટ્રેનનો હાલનો વિસ્તાર દર્શાવવા વગેરે કાર્યો માટે પણ આનો ઉપયોગ થાય છે. આ સેવા નજીકનું ATM, બૅન્ક, ભોજનાલય, બસસ્ટૉપ અથવા જરૂરી કોઈ સ્થળ શોધવાની સગવડતા પણ આપે છે.




આકૃતિ 13.6 : ગુગલ નકશાનો ઉપયોગ કરીને દિશાનિર્દેશ શોધવો

## અક્ષરનકશો (Character Map)

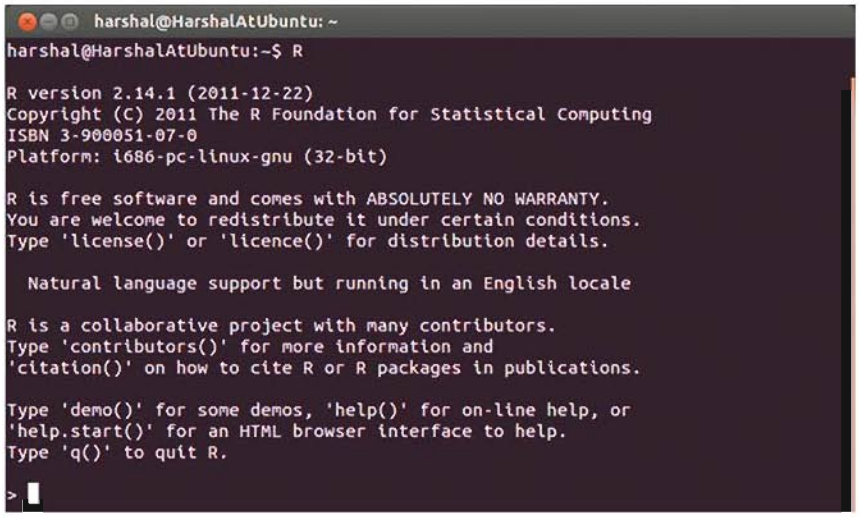
આકૃતિ 13.7માં દર્શાવેલ (અક્ષરનકશો) કૅરેક્ટરમૅપ પ્રોગ્રામનો ઉપયોગ કોઈ પણ વિનિયોગમાં યુનિકોડ (Unicode) અક્ષર દાખલ કરવા માટે થાય છે. પ્રથમ ડાબી તકતી (પેન)માંથી લિપિ પસંદ કરવાની અને ત્યાર પછી જે અક્ષરને લખાણ અથવા નકલના વિસ્તારમાં ઉમેરવાનો હોય, તેના ઉપર ડબલ ક્લિક કરવાની. ક્લિક કર્યા પછી ખાલી જગ્યા અને ચિહ્નોને કી-બોર્ડ વડે સીધા જ દાખલ કરી શકાય છે. અક્ષરો ભેગા કરવા, જેમકે ગુજરાતીમાં 'દીર્ઘ ઈ' ઉમેરવા માટે અનુરૂપ યોગ્ય અચલને ભેગા કરવા પડે છે. નીચે સ્ટેટસ લાઈનમાં એક ક્લિક વડે પસંદ કરેલ અક્ષર અથવા ડબલ ક્લિક વડે દાખલ કરેલ અક્ષરની ટૂંકી માહિતી દર્શાવે છે. જ્યારે અક્ષરો વિશેની વધુ માહિતી Character Details નામના ટેબમાં આપેલ છે. આવશ્યક જથ્થામાં વિષયવસ્તુ મેળવ્યા પછી, આપણે આ વિષયવસ્તુની નકલ (Copy) કરી અને કોઈ વિનિયોગમાં પેસ્ટ (Paste) કરી શકીએ છીએ. બીજી કોઈ લિપિમાં પ્રાસંગિક ધોરણે તમારે થોડા અક્ષરો જ ટાઇપ કરવા હોય, તો આ માટે કૅરેક્ટરમૅપ એ એક સારો ઉકેલ છે.




આકૃતિ 13.7 : કૅરેક્ટરમૅપ પ્રોગ્રામ

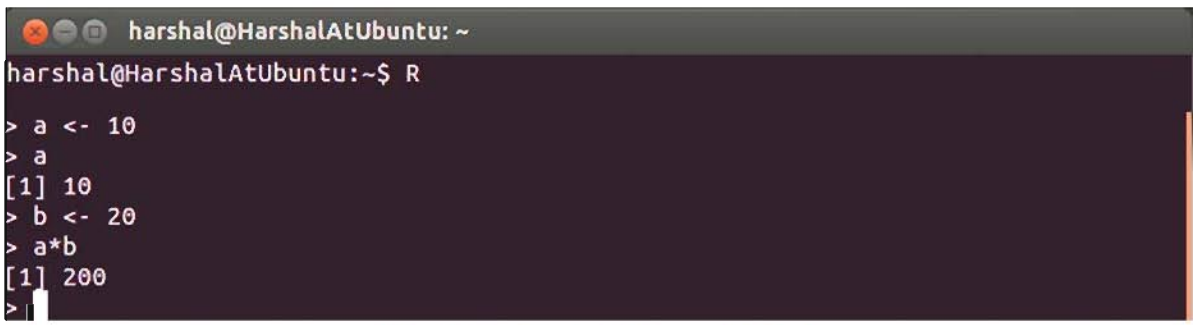
## 'આર' સૉફ્ટવેર (The 'R' Software)

'આર' એ એક આંકડાકીય ગણતરી માટેનું નિઃશુલ્ક સૉફ્ટવેર છે. તે એક GNU યોજના છે. તેનો આંકડાકીય વિશ્લેષણ માટે બહોળા પ્રમાણમાં ઉપયોગ થાય છે. તેને પોતાની સ્ક્રિપ્ટિંગ ભાષા છે. તે એક કેસ સંવેદનશીલ (કેસ સેન્સિટિવ) ભાષા છે. 'આર'ને કમાન્ડલાઈન અને ગ્રાફિક્સ નામના બે કાર્યપ્રદેશ છે. જો આપણે GUI પ્રદેશનો ઉપયોગ કરવો હોય, તો આપણે ઉબુન્ટુ સૉફ્ટવેર કેન્દ્રમાંથી આર કમાન્ડર અથવા આર સ્ટુડિયો નામનું ગ્રાફિકલ એડિટર પ્રસ્થાપિત કરવું પડે. 'આર'ને પ્રસ્થાપિત કરવા માટેના કમાન્ડ આ પ્રકરણના અંતમાં આપેલ છે. ટર્મિનલ વિન્ડોમાંથી આર લિપિને બોલાવવા માટે, ટર્મિનલ ખોલીને કમાન્ડપ્રોમ્પ્ટ ઉપર આર (R) લખો અને એન્ટર કી દબાવો. આકૃતિ 13.8માં દર્શાવ્યા પ્રમાણે તમને આર પ્રોમ્પ્ટની સાથે સ્વાગતસંદેશ મળશે.




આકૃતિ 13.8 : આર કમાન્ડ પ્રોમ્પ્ટ

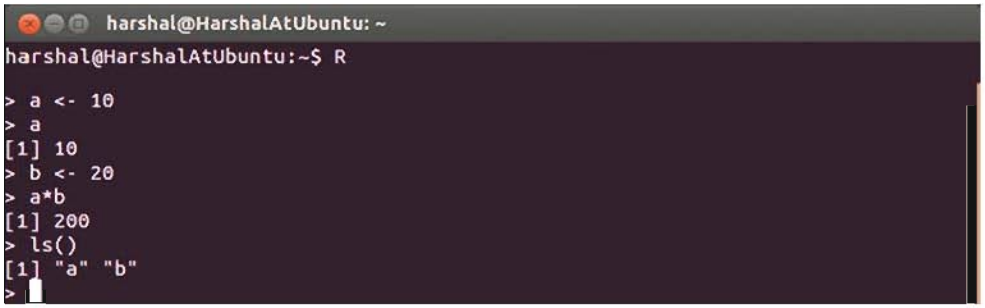
લિનક્સ શેલની જેમ જ R કોમેન્ટની નિશાની તરીકે # નો ઉપયોગ કરે છે. # પછીની કોઈ પણ લખાણ લાઈનના અંત સુધી કોમેન્ટ (ટિપ્પણી) તરીકે ગણવામાં આવે છે. નંબર અને સ્ટ્રિંગ બે મૂળભૂત ડેટા ટાઈપ છે. (માહિતીના પ્રકાર છે.) સ્ટ્રિંગને એક અવતરણ અથવા ડબલ (બે) અવતરણ ચિહ્નોમાં બાંધવામાં આવે છે. બીજી પ્રોગ્રામિંગ ભાષાની જેમ જ સામાન્ય પ્રક્રિયકો અને વિધેયો પણ ઉપલબ્ધ છે. વસ્તુઓની ક્રમિક યાદી (એરે અથવા લિસ્ટ) સામાન્યપણે ઉપયોગમાં આવે છે અને તેનો વેક્ટર તરીકે પણ ઉલ્લેખ થાય છે. યાદી (લિસ્ટ) બનાવવા માટે C વિધેયનો ઉપયોગ થાય છે, જે જુદાં-જુદાં નંબરને એક યાદીમાં ભેગા કરે છે. પ્રક્રિયકો જેવા કે +, -, *, / અને બીજા બધા એક નંબર તેમજ યાદી ઉપર એકસરખી રીતે જ કાર્ય કરે છે. તેમ છતાં, જ્યારે દ્વિઅંકી (બાઈનરી) પ્રક્રિયકનાં બંને સંકાર્ય (Operands) યાદી (લિસ્ટ) હોય ત્યારે, તેની લંબાઈ (તેની અંદરનાં તત્ત્વોની સંખ્યા) સરખી હોવી જોઈએ, અચલનું નામ દાખલ કરી અને એન્ટર દબાવવાથી તેનું મૂલ્ય દેખાશે. આ વિભાવનાનું ઉદાહરણ આકૃતિ 13.9માં દર્શાવેલ છે.




આકૃતિ 13.9 : R 'આર'માં અચલનો ઉપયોગ

અહીં > ની નિશાની આરનો પ્રોમ્પ્ટ દર્શાવે છે. વિધાન a ← 10 'a' નામનો અચલ બનાવે છે અને તેને કિંમત 10 આપે છે. વિધાન > a પછી એન્ટર કી દબાવતા અચલ aની કિંમત દેખાશે. આ જ રીતે વિધાન b ← 20 'b' નામનો અચલ બનાવે છે અને તેને 20 કિંમત આપે છે. વિધાન > a*b પછી એન્ટર કી દબાવતા અચલ 'a' અને 'b' ની કિંમતનો ગુણાકાર કરી, તેનું પરિણામ પ્રોમ્પ્ટ ઉપર દર્શાવે છે.

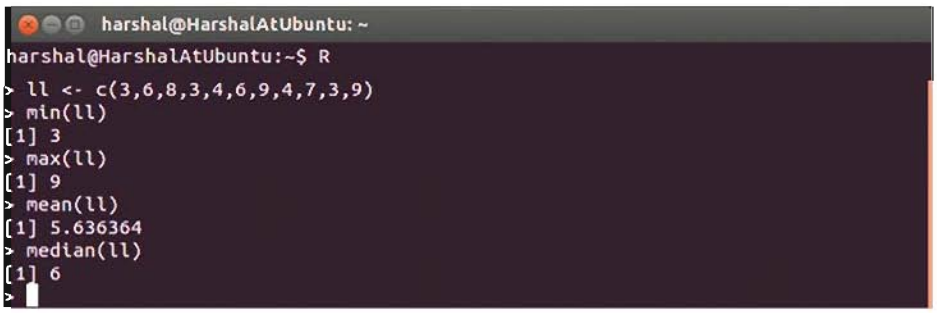
કેટલાક સામાન્ય R કમાન્ડ q() આરમાંથી બહાર નીકળવા માટે, help() ઑનલાઈન મદદ મેળવવા માટે, demo() કંઈક પ્રદર્શન જોવા માટે અને help.start() બ્રાઉઝરમાં ઑનલાઈન મદદ ખોલવા માટે વપરાય છે. જો કોઈ નિશ્ચિત વિધેય માટે મદદ જોઈતી હોય, તો આપણે help(function name) વાક્યરચનાનો ઉપયોગ કરવાની જરૂર પડે છે. જ્યારે આપણે Rની બહાર નીકળીએ ત્યારે આપણને પૂછવામાં આવે છે કે આપણે માહિતીને (અચલની કિંમતને) સેવ કરવી છે ? જો આપણે 'હા' કહીએ અથવા "y" કહીએ ત્યારે, વર્તમાન રિરેક્ટરીની એક ફાઈલમાં આ માહિતી સેવ કરવામાં આવે છે અને જ્યારે આ જ રિરેક્ટરીમાં ફરી વખત આપણે R શરૂ કરીએ, ત્યારે આ માહિતીને ઉપલબ્ધ કરે છે. આકૃતિ 13.10માં દર્શાવ્યા મુજબ વિધેય ls() બનાવેલા તમામ અચલની યાદી પ્રસિદ્ધ કરે છે.




આકૃતિ 13.10 : 'આર'માં ls()નો ઉપયોગ

ક્રમાંકની હારમાળા બનાવવા માટે 'શરૂનો નંબર : અંતનો નંબર' વાક્યરચનાનો ઉપયોગ થાય છે. ઉદાહરણ તરીકે 1:5 એ c(1, 2, 3, 4, 5)ની બરાબર છે.

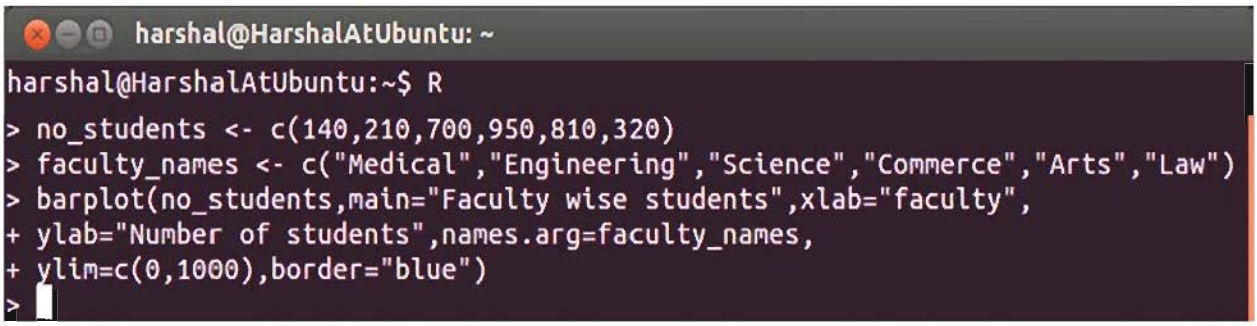
'આર' આંકડાશાસ્ત્રનાં કાર્યોને તદ્દન નિખાલસ સ્વરૂપે પૂરા પાડે છે. ઉદાહરણ તરીકે આકૃતિ 13.11માં દર્શાવ્યા પ્રમાણે યાદીમાંથી ન્યૂનતમ કિંમત, મહત્તમ કિંમત, મધ્યક અને મધ્યસ્થ શોધવા માટે આપણે min(list), max(list), mean(list) અને median(list) વિધેયનો ઉપયોગ કરી શકશે.




આકૃતિ 13.11 : Rમાં ગાણિતિક વિધેયો

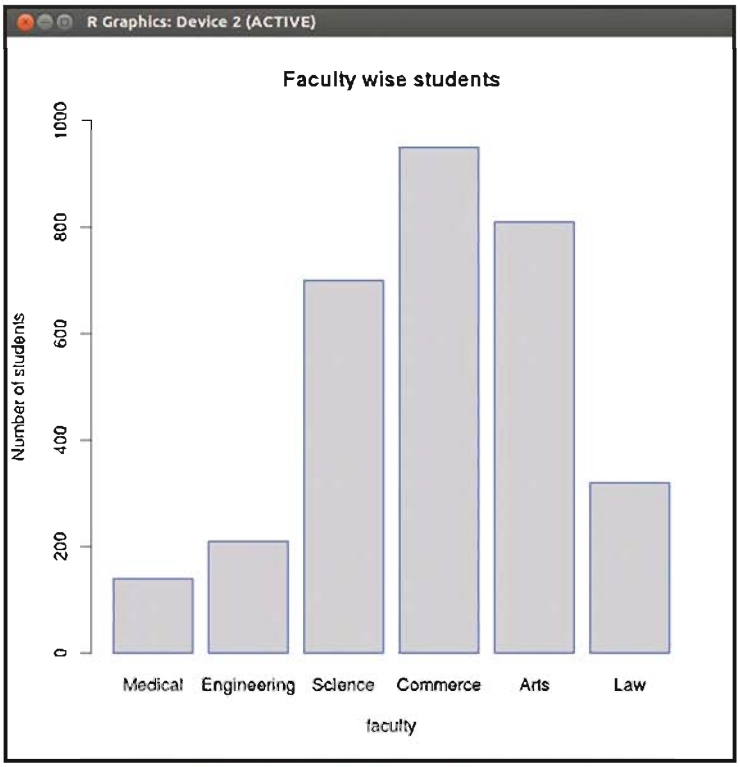
### 'આર'માં આલેખ દોરવા

ધારો કે આપણે એક બારચાર્ટ દોરવો છે, જે યુનિવર્સિટીમાં વિદ્યાશાખા પ્રમાણે વિદ્યાર્થીઓની સંખ્યા દર્શાવે છે. જે માટે આપણે વિદ્યાર્થીઓની સંખ્યાની વિભાગ અનુસાર એક યાદી અને એક વિદ્યાશાખાની યાદી બનાવીશું. ત્યાર પછી આપણે આકૃતિ 13.12 માં બતાવ્યા પ્રમાણે barplot() વિધેયનો ઉપયોગ કરીને તેમાં જુદી-જુદી આર્ગ્યુમેન્ટ પાસ કરીને આલેખ દોરીશું.



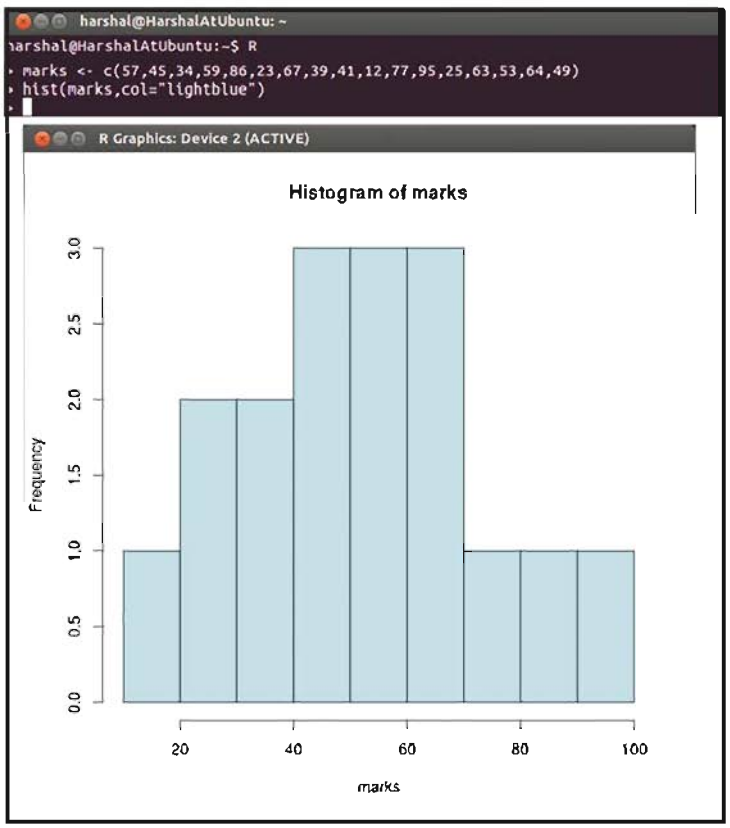

આકૃતિ 13.12 : Rમાં આલેખ દોરવા

પહેલી આર્ગ્યુમેન્ટ માહિતીની યાદી છે અને બાકીની તમામ આર્ગ્યુમેન્ટનું સ્વરૂપ નામ=કિંમત છે. આપણે આર્ગ્યુમેન્ટ તરીકે main, xlab, ylab, names.arg, ylim અને border નો ઉપયોગ કર્યો છે, જે મુખ્ય મથાળું (Title), એક્સ ધરીનું લેબલ, વાય ધરીનું લેબલ, બાર માટેની કિંમત, વાય ધરી ઉપરની કિંમતની રેન્જ અને બોર્ડરનો કલર દર્શાવે છે. નિરીક્ષણ કરો કે જ્યારે કમાન્ડ લાંબો હોય અને કમાન્ડ પૂરો થયા પહેલા આપણે એન્ટર કી દબાવીએ, તો જ્યાં સુધી કમાન્ડ પૂરો ન થાય ત્યાં સુધી R આપણને + પ્રોમ્પ્ટ આપે છે. આકૃતિ 13.13માં દર્શાવ્યા પ્રમાણે સામાન્યતઃ આલેખને ગ્રીડલાઇન હોતી નથી.




આકૃતિ 13.13 : Rમાં બનાવેલ બારચાર્ટ

હિસ્ટોગ્રામ બનાવવો પણ કઠિન નથી. ધારો કે આપણી પાસે જુદા-જુદા વિદ્યાર્થીઓના કોઈ એક વિષયના 100માંથી મેળવેલા ગુણ છે. આકૃતિ 13.14માં આપેલ કોડનો ઉપયોગ કરીને હિસ્ટોગ્રામ દોરી શકાય, જે આ જ આકૃતિમાં દર્શાવ્યા પ્રમાણેનો દેખાશે.




આકૃતિ 13.14 : Rમાં બનાવેલ હિસ્ટોગ્રામ

## રેશનલ પ્લાન (Rational Plan)

જુદા જુદા કાર્યક્ષેત્રમાં કામ કરતા લોકો જેવા કે બાંધકામ, ઇજનેરી સેવા અને સલાહકાર વેપાર ધંધા અથવા સોફ્ટવેર બનાવવા વગેરેને યોજના (Project) ઉપર કામ કરવાનું હોય છે અને આવી યોજનામાં સમય નાણાં અને સાધનોની મુશ્કેલી હોય છે. કોઈ પણ કારણોસર આવી યોજના મોડી પૂરી થાય તો યોજનાનો ખર્ચ વધી જાય છે. આ તબક્કે આવી યોજનાના સંચાલન માટે એક સુવ્યવસ્થિત ગોઠવણી કરવી પડે છે. જે ગોઠવણ નિર્ધારિત સમયમાં અને નાણાંમાં યોજના પૂરી કરવામાં મદદ કરે.

રેશનલ પ્લાન એ એક આવો જ ઓપન સોર્સ સોફ્ટવેર છે જે યોજનાના સંચાલકને આવો પ્લાન બનાવવામાં, નિર્વાહ કરવામાં મદદ કરે છે. તે યોજનાના જીવનચક્ર દરમ્યાન યોજનાના સંચાલકને મદદ કરે છે. આમાં ત્રણ ડેસ્કટોપ ઉત્પાદનનો સમાવેશ થાય છે. રેશનલ પ્લાન સિંગલ, મલ્ટિ અને વ્યૂઅર.

રેશનલ પ્લાન સિંગલ એ સ્વતંત્ર યોજનાના સંચાલન માટે ઉપયોગી છે. તે બીજી યોજના સાથે જોડાયેલ કે કોમન સાધનોનો વપરાશ કરતી અન્ય યોજના સાથે જોડાયેલ માટે હોતો નથી. તે સંચાલકને સામાન્ય યોજના માટેની માહિતી જેવી કે નામ, નોંધ, લીક, ધારણાઓ, અવરોધ અને જોખમો વગેરે માહિતી યોજના સાથે જોડવા આપે છે. તે વિગતવાર નોંધપત્રક બનાવવા, તેમાં સુધારાવધારા કરવા તથા તેને દૂર કરવાની સગવડ આપે છે. એક વખત સાધન સામગ્રી બની ગયા પછી તેને આપણે કાર્યની ફાળવણી કરી શકીએ છીએ. જે આપણને યોજનાનું ટ્રેકિંગ ટૂલ પૂરું પાડે છે. જેવા કે નિર્ણાયક રસ્તો (Critical Path), કાર્ય માટેની લક્ષ્ય પૂર્ણતાની કિંમત, માહિતીના સમયસર તબક્કાનું કામ અને કિંમત વગેરે. આપણે છાપી શકાય તેવો અહેવાલ બનાવી શકીએ છીએ અને બીજી યોજના સંચાલનના ટૂલમાંથી માહિતી આયાત કરી શકીએ છીએ અને બીજા માળખામાં માહિતી નિકાસ પણ કરી શકીએ છીએ.

રેશનલ પ્લાન મલ્ટિ એ એવી યોજનાનું સંચાલન કરવામાં મદદ કરે છે કે જેમાં એક કંપનીના સાધનો જુદી-જુદી યોજનામાં વહેંચાયેલાં હોય. તે એકબીજા ઉપર આધારિત યોજનાનું સંચાલન પણ કરે છે. તેમાં રેશનલ પ્લાન સિંગલમાં આપેલ તમામ લક્ષણ (ફિચર્સ)નો સમાવેશ થાય છે. વધારેમાં તે, દરેક યોજનામાં રોકાયેલ સાધનોની માહિતી (કામ, કિંમત અને વધારે ફાળવણી)ની ગણતરી પણ કરી આપે છે. તે જુદી જુદી યોજનાઓ સાથે સંબંધિત કાર્યનું જોડાણ, યોજનાની માહિતીનું વિશ્લેષણ અને યોજનાનો પૉર્ટફોલિયો બનાવવાની પરવાનગી આપે છે.

રેશનલ પ્લાન વ્યૂઅર એ એક વધારાનું ટૂલ છે જે મૂળ ફાઇલના માળખાં (.xrp)માં રહેલ યોજનાને વહેંચવા (શેર કરવા) માટે બનાવેલ છે. પણ તેનો ઉપયોગ માઇક્રોસૉફ્ટ યોજનાની ફાઇલ ખોલવા માટે પણ થાય છે. સાધનોને ફાળવેલી કામગીરી જોવા માટે તેમજ ગોઠવેલ કાર્યપદ્ધતિમાં કોઈ પણ પ્રકારનો ફેરફાર કર્યા વગર યોજનાનો વિકાસ જોવા માટે તે ખૂબજ ઉપયોગી છે.

ઉબુન્ટુ 10.04માં રેશનલ પ્લાન આવતું નથી અને સમર્થન પણ કરતું નથી તે ફક્ત ઉબુન્ટુ 12.04 અને ત્યાર પછીના વર્ઝનમાં જ ઉપલબ્ધ છે.

## સ્કાઇપ (Skype)

તમે કેટલીય સંદેશા માટેની જુદી જુદી સેવાઓ જેવી કે યાહુ મૅસેન્જર, ગુગલટોક અથવા રેડિફબોલનો ઉપયોગ કર્યો હશે, જેનો ઉપયોગ આપણે વાસ્તવિક સમયમાં ગપસપ (ચેટિંગ) માટે કરીએ છીએ અને તેમાં લખાણ, દૃશ્ય અને શ્રાવ્યમાહિતીનો ઉપયોગ પણ કરીએ છીએ. સ્કાઇપ આવું જ એક સૉફ્ટવેર છે, જે આપણને કમ્પ્યૂટરમાં ઇન્ટરનેટનો ઉપયોગ કરીને કૉલ કરવા માટેની સગવડ આપે છે.

સ્કાઇપ ઉપયોગકર્તાને અવાજ, લખાણ અને વીડિયોનો ઉપયોગ કરીને બીજા સાથે સંપર્ક કરવાની પરવાનગી આપે છે. આપણે આપણા ટેલીફોન નેટવર્કમાં ફોન કરવા માટે પણ સ્કાઇપનો ઉપયોગ કરી શકીએ. ટેલીફોન અથવા મોબાઇલ ફોન ઉપર કરેલા ફોનનો ખર્ચ ઉપયોગકર્તાના ખાતામાં ઉધાર થાય છે અને સ્કાઇપમાં કરેલ ફોન કોલ નિઃશુલ્ક હોય છે. તે ફાઇલને એક જગ્યાએથી બીજી જગ્યાએ ફેરવવાની તેમજ વીડિયો-કૉન્ફરન્સની વધારાની સગવડ પણ પૂરી પાડે છે.

સ્કાઇપ સૉફ્ટવેરને નિઃશુલ્ક ડાઉનલોડ કરી શકાય છે, પરંતુ તેનો સૉર્સકોડ માલિકીનો છે, તેમાં કોઈ પણ પ્રકારના સુધારાવધારા કરી શકાતા નથી. સ્કાઇપનો ઉપયોગ કરવો હોય, તો અવાજ(સાઉન્ડ)નું ઇનપુટ અને આઉટપુટ કાર્યરત હોવું જોઈએ.

આધુનિક કમ્પ્યૂટરમાં આ સગવડ હોય જ છે. લેપટોપમાં તો અંદર જ આપેલ હોય છે જ્યારે ડેસ્કટોપ કમ્પ્યૂટરમાં આપણે હેડફોન, સ્પીકર અને વેબકૅમેરાને બહારથી જોડવાં પડે છે. આપણે સ્કાઇપને સૉફ્ટવેર કેન્દ્રમાંથી પ્રસ્થાપિત કરી શકીએ છીએ.

સ્કાઇપ શરૂ કરવા માટે, **Applications → Internet → Skype** વિકલ્પ પસંદ કરો. જો તે તમે પ્રથમ વખત જ ખોલતા હશો, તો એક ઉપયોગકર્તા માટે પરવાના માટેનાં કરારની વિન્ડો ખોલશે. તેનું લખાણ વાંચી અને "I agree" બટન ઉપર ક્લિક કરો. હવે આપણે આકૃતિ 13.15માં આપેલ વિન્ડો જોઈ શકીશું.


**આકૃતિ 13.15 : સ્કાઇપની શરૂઆતની વિન્ડો**

સ્કાઇપસેવાનો ઉપયોગ કરવા માટે ઉપયોગકર્તાનું ખાતું હોવું જોઈએ. જો ન હોય તો, "Don't have a Skype Name yet?" ઉપર ક્લિક કરો અને સ્કાઇપ નામ બનાવવા માટેનાં પગલાંને અનુસરો. એક વખત તમારી પાસે સ્કાઇપ નામ અને પાસવર્ડ આવી ગયા પછી આકૃતિ 13.15માં બતાવેલ ટેક્સ્ટબૉક્સમાં માહિતી ભરો અને "Sign in" બટન ક્લિક કરો. જો બધું જ બરાબર હશે તમે બીજા સ્કાઇપના ઉપયોગકર્તા સાથે સંવાદ કરી શકશો અથવા સ્કાઇપનો ઉપયોગ કરીને ફોનકૉલ કરી શકશો.

## સારાંશ

આ પ્રકરણમાં આપણે કેટલાંક ઉપયોગી નિ:શુલ્ક ટૂલ્સ અને સેવાઓની ચર્ચા કરી. આપણે માહિતી-સંકોચનની તકનિક વિશે ભણ્યા જે માહિતીસંગ્રહ કરવા માટેની જગ્યામાં ઘટાડો કઈ રીતે કરી શકાય તે શીખવે છે. આ તકનિકનો ઉપયોગ ફાઇલ અને ડિરેક્ટરી ઉપર કરવા માટેનું ટૂલ આર્ચિવ (આર્કાઇવ્ઝ) મૅનેજર પણ જોયું. આપણે મલ્ટિમીડિયા પ્લેયર VLCની પણ ચર્ચા કરી. આપણે ગુગલ નકશા સેવાની પણ ચર્ચા કરી, જે આપણને જુદા-જુદા વિસ્તારના નકશાઓ ઑનલાઇન પૂરા પાડે છે. લખાણમાં જુદી-જુદી ભાષાનાં અક્ષરો અને ચિહ્નોને ઉમેરવા માટેના અક્ષરનકશો (કૅરેક્ટર મૅપ) વિશે પણ અભ્યાસ કર્યો. ગાણિતિક ગણતરી માટેના 'આર' એન્વાયરમેન્ટ ઉપર પણ ચર્ચા કરી, જેમાં આપણે સામાન્ય ગણતરી જેવી કે મધ્યસ્થ અને મધ્યક શોધવા વિશે ચર્ચા કરી અને ''આર''માં બારચાર્ટ અને હિસ્ટોગ્રામ કેવી રીતે બનાવાય તે વિશે પણ ભણ્યા અને છેલ્લે આપણે રેશનલ પ્લાન અને સ્કાઇપનો ઉપયોગ પણ જોઈ ગયા.

## શિક્ષક માટે સૂચના

આ પ્રકરણમાં આપેલ ટૂલ્સ ફક્ત પ્રદર્શન માટે જ છે. શિક્ષકે વિદ્યાર્થીને આ ટૂલ્સ ડેમો સ્વરૂપે દર્શાવવાના છે. પ્રાયોગિક સ્વાધ્યાયમાં આપેલા ઉદાહરણનો ઉપયોગ કરીને શિક્ષકે વિદ્યાર્થીઓને આ ટૂલ્સનો ઉપયોગ દેખાડવાનો છે. શિક્ષકે કમ્પ્યૂટરમાં 'આર' સૉફ્ટવેર પ્રસ્થાપિત કરવું પડશે. R પ્રસ્થાપિત કરવા માટે નીચે પ્રમાણેનો કમાન્ડ આપો :

*sudo apt-get install r-base r-base-dev*

### સ્વાધ્યાય

**1.** માહિતી-સંકોચન વિશે ટૂંકનોંધ લખો.

**2.** આપણને માહિતી-સંકોચન ટૂલની જરૂરત શા માટે છે ?

**3.** આર્ચિવ (આર્કાઇવ્ઝ) મૅનેજરનાં મુખ્ય લક્ષણોની યાદી બનાવો.

**4.** VLC મીડિયાપ્લેયરનાં મુખ્ય લક્ષણો જણાવો.

**5.** ગુગલ નકશાના વિનિયોગની યાદી બનાવો.

**6.** કૅરેક્ટરમૅપ (અક્ષરનકશો) પ્રોગ્રામનો ઉપયોગ સમજાવો.

**7.** **આપેલા વિકલ્પમાંથી સાચો વિકલ્પ પસંદ કરો :**

(1) નીચેનામાંથી કઈ ફાઇલ તેમાં રહેલ આખા ડિરેક્ટરી-માળખાનો ઉલ્લેખ કરે છે ?

(a) અપાચે (b) આર્ચી (c) આર્ચિ (d) આર્ચિવ (આર્કાઇવ્ઝ)

(2) ટાર (TAR)નું આખું નામ શું છે ?

(a) ટેપ આર્ચિવર (b) ટેક આર્ચિવર (c) ટેસ્ટ આર્ચિવર (d) ટાઇટ આર્ચિવર

(3) કયા આર્ચિવ પ્રકારમાં પાસવર્ડ વિકલ્પ ઉપલબ્ધ હોય છે ?

(a) ઝીપ (b) ટાર (c) tar.gz (d) ઝીપ અને tar.gz બંને

(4) નીચેનામાંથી કયું મીડિયાપ્લેયર લાક્ષણિકતાની દૃષ્ટિએ સમૃદ્ધ છે ?

(a) VAC (b) VEC (c) VLC (d) VNC

(5) VLCનું આખું નામ શું છે ?

(a) વીડિયોલેન ક્લાયન્ટ (b) વીડિયો લાઇનકોડર

(c) વીડિયો લેન્થકોડર (d) વીડિયો લિસ્ટ ક્રિએટર

(6) કઈ તકનિક આપણા સ્થાનનો ચોક્કસ ખ્યાલ આપે છે ?

(a) GRS (b) GPRS (c) GRPS (d) GPS

(7) કોઈ પણ વિનિયોગમાં અક્ષરો દાખલ કરવા કયા પ્રોગ્રામનો ઉપયોગ થાય છે ?

(a) કૅરેક્ટર રિસ્પ્લે (b) કૅરેક્ટર ઇન્સર્ટ

(c) કૅરેક્ટર મૅપ (અક્ષરનકશો) (d) કૅરેક્ટર સિલેક્ટ

(8) Rમાંથી બહાર નીકળવા માટે કયો કમાન્ડ વપરાય છે ?

(a) quit() (b) q() (c) exit() (d) close()

(9) Rમાં બારગ્રાફ બનાવવા માટે કયા વિધેયનો ઉપયોગ થાય છે ?

(a) bar() (b) plot() (c) bargraph() (d) barplot()

(10) નીચેનામાંથી રેશનલ પ્લાનનાં જુદાં-જુદાં સ્વરૂપ કયા છે ?

(a) સિંગલ, મલ્ટિ, વ્યૂઅર (b) સિંગ્યુલર, મલ્ટિપલ

(c) વ્યૂ, પ્રિવ્યૂ (d) સર્વર, ક્લાયન્ટ

## પ્રાયોગિક સ્વાધ્યાય

1. ઝીપફાઇલ બનાવી તેમાં આખી ડિરેક્ટરીનો સંગ્રહ કરો.
2. tar.gz ફાઇલ બનાવી, તેમાં આખી ડિરેક્ટરીનો સંગ્રહ કરો.
3. સબડિરેક્ટરી ફાઇલ1 (files1)માં ઝીપ આર્ચિવમાંથી બધી જ ફાઇલો બહાર કાઢો.
4. સબડિરેક્ટરી ફાઇલ2 (files2)માં tar.gz આર્ચિવમાંથી બધી જ ફાઇલો બહાર કાઢો.
5. ગુગલ નકશામાં તમારી શાળા શોધો. સેટેલાઇટ તસવીરમાં તમે તમારી ઇમારત ઓળખી શકો છો ?
6. ગુગલ નકશામાં તમારું ઘર શોધો. તમારા ઘરેથી શાળા સુધી પહોંચવા માટેનો માર્ગ શોધો. આ રસ્તો એ જ છે, જેનો તમે દરરોજ ઉપયોગ કરો છો ?
7. કૅરેક્ટરમૅપ (અક્ષરનકશો)નો ઉપયોગ કરીને geditમાં सत्यमेवजयते લખો.
8. કૅરેક્ટરમૅપ (અક્ષરનકશો)નો ઉપયોગ કરીને geditમાં $\sin(\alpha - \beta) = \sin\alpha \cos\beta - \cos\alpha \sin\beta$ લખો.
9. વર્ષના મહિનાના દરેક દિવસોનો બારગ્રાફ બનાવો.
10. પાઠ્યપુસ્તકના પાનાંનો બારગ્રાફ બનાવો.
11. અઠવાડિયાનાં દિવસોનાં નામમાં (રવિવાર, સોમવાર..... શનિવાર) વારંવાર આવતા અક્ષરો માટેનો હિસ્ટોગ્રામ દોરો. ફક્ત એવા અક્ષરોનો જ સમાવેશ કરો જે ઓછામાં ઓછા એક વખત આવતા હોય.
12. તમારું પોતાનું સ્કાઇપ ઉપયોગકર્તાખાતું બનાવો અને સ્કાઇપસેવા દ્વારા આપવામાં આવતી સગવડનું નિરીક્ષણ કરો.