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 પ્રૉપર્ટી (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
```

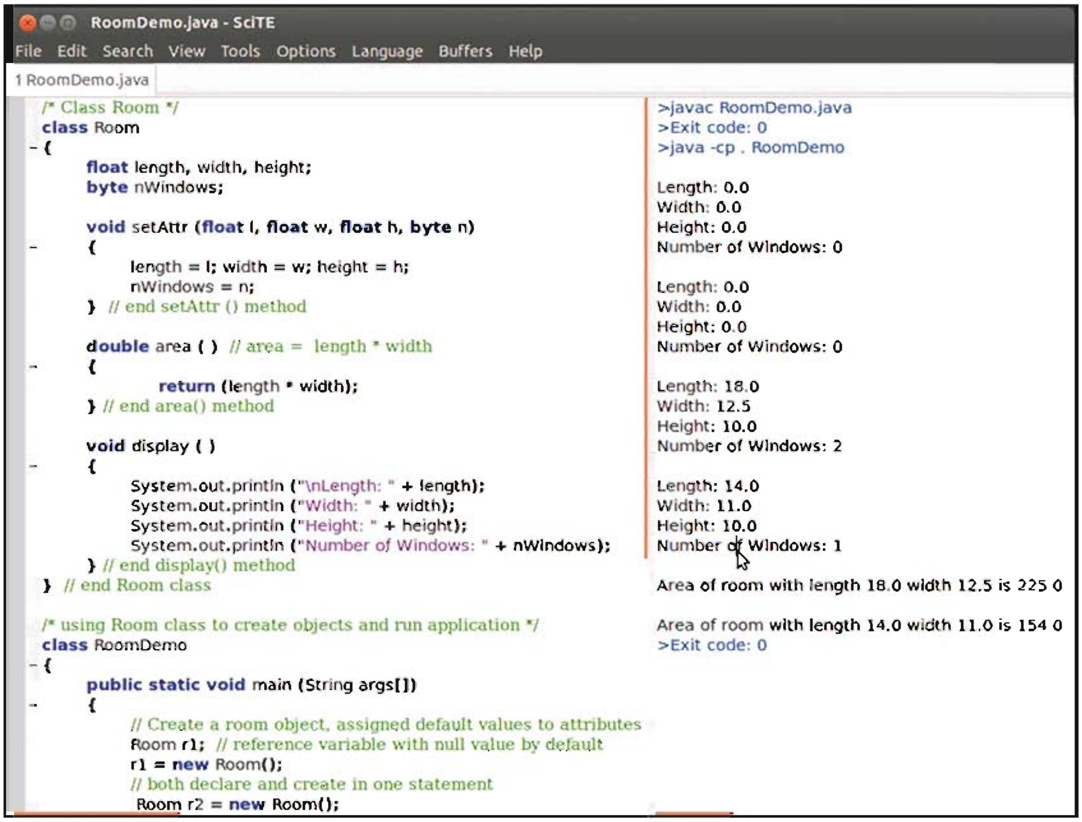
```
        {
                return (length * width);
        } // end area() method

      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. સૉર્સફાઇલમાં 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.

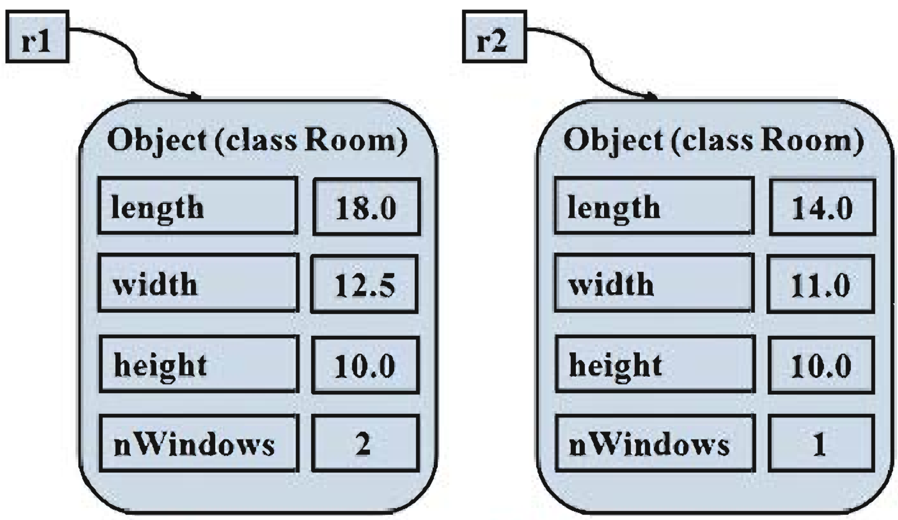
જ્યારે "r1=new Room();" વિધાનનો અમલ થાય છે, ત્યારે યાદ રાખો કે ઑબ્જેક્ટનો ચલ r1માં સંગ્રહ થતો નથી. ચલ r1 ઑબ્જેક્ટનો ફક્ત સ્થાનાંક (ઍડ્રેસ) જ ધરાવે છે.

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

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

ઈન્સ્ટન્સ વેરિયેબલ ઑબ્જેક્ટનાં ઍટ્રિબ્યૂટ વ્યાખ્યાયિત કરે છે. ક્લાસ ઍટ્રિબ્યૂટનો પ્રકાર વ્યાખ્યાયિત કરે છે. દરેક ઈન્સ્ટન્સ 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 ઈન્સ્ટન્સ વેરિયેબલની કિંમત મેળવવા અથવા બદલવા માટે ઑબ્જેક્ટનો ઉપયોગ કરીને ઈન્વૉક (invoke) કરી શકાય છે. આવી મેથડ **ઈન્સ્ટન્સ મેથડ** તરીકે ઓળખાય છે.

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

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

સારાંશ રૂપે,

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

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

ઈન્સ્ટન્સ વેરિયેબલ અને ઈન્સ્ટન્સ 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 ઈન્વૉક 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 મોડિફાયર (access modifier) વડે શક્ય છે, જેની ચર્ચા આપણે હવે પછી કરીશું.

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

```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-મૅથડને 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 એક્સેસ કરી શકાય છે.
- ઇન્સ્ટન્સ વેરિયેબલ અને ઇન્સ્ટન્સ મેથડને ક્લાસ મૅથડમાંથી ઍક્સેસ કરી ન શકાય કારણકે ક્લાસમૅથડ 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. મૅથડના નિયમાનુસારના પ્રાચલો પણ લોકલ વેરિયેબલ છે. જ્યારે બ્લૉકની શરૂઆત થાય છે, ત્યારે તેનું નિર્માણ થાય છે અને જ્યારે મૅથડ 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, મૅથડની 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 (મૅથડ ઓવરલોડિંગ) - Polymorphism (Method Overloading)

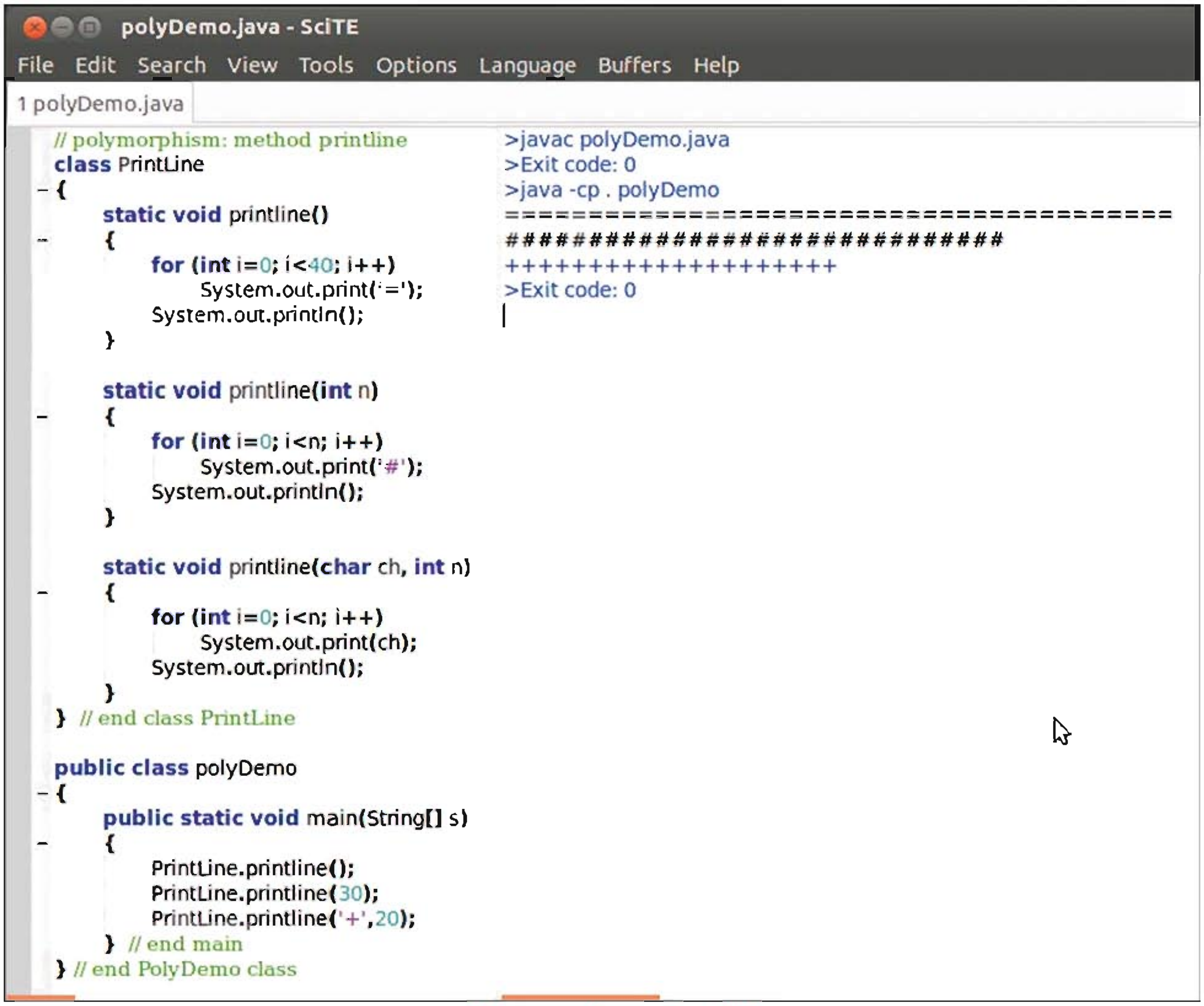
આપણે ઑબ્જેક્ટ બનાવવાનાં પ્રથમ બે પગલાં Declaration અને Instantiation વિશે અભ્યાસ કર્યો. ત્રીજું પગલું પ્રારંભિક કિંમત આપવાનું (Initialization) છે. આ માટે કન્સ્ટ્રક્ટરના ઉપયોગની જરૂર પડે છે. આપણે કન્સ્ટ્રક્ટર વિશે શીખીએ તે પહેલાં આપણે જાવામાં પોલિમૉર્ફિઝમનું અમલીકરણ કઈ રીતે કરી શકાય તે બાબત જોઈએ.

પોલિમોર્ફિઝમ 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 મૅથડનું 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 મૅથડ 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 println() 40 વાર '=' અક્ષર, println(int n) n વાર '#' અક્ષર જ્યારે println(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 **ડિફોલ્ટ કન્સ્ટ્રક્ટર** (default constructor) હોય છે; કોઈ વાર તેને ચલ વગરના કન્સ્ટ્રક્ટર તરીકે ઓળખવામાં આવે છે. ડિફોલ્ટ 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) ઇન્વૉક કરવામાં આવે છે.
- કન્સ્ટ્રક્ટર પ્રોગ્રામમાં અન્ય કોઈ પણ જગ્યાએ સ્પષ્ટ રીતે (explicitly) ઇન્વૉક 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 ડિફૉલ્ટ કન્સ્ટ્રક્ટરનો ઉપયોગ કરીને બનાવવામાં આવે છે. પૂર્વનિર્ધારિત કિંમત ઍટ્રિબ્યૂટને આપવામાં આવે છે.

ક્લાસમાં વપરાશકર્તા વ્યાખ્યાયિત કન્સ્ટ્રક્ટરની હાજરીમાં ડિફૉલ્ટ 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

એક્સેસ કન્ટ્રોલ માટેનાં વિઝિબિલિટી મોડિફાયર (Visibility Modifiers for Access Control)

એક્સેસ કન્ટ્રોલ એ દૃશ્યતાની નિયંત્રણ બાબત છે. આથી, એક્સેસ 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 એક્સેસ 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 સોર્સફાઇલ અને પૅકેજમાંથી પણ ઍક્સેસ કરી શકાય છે.

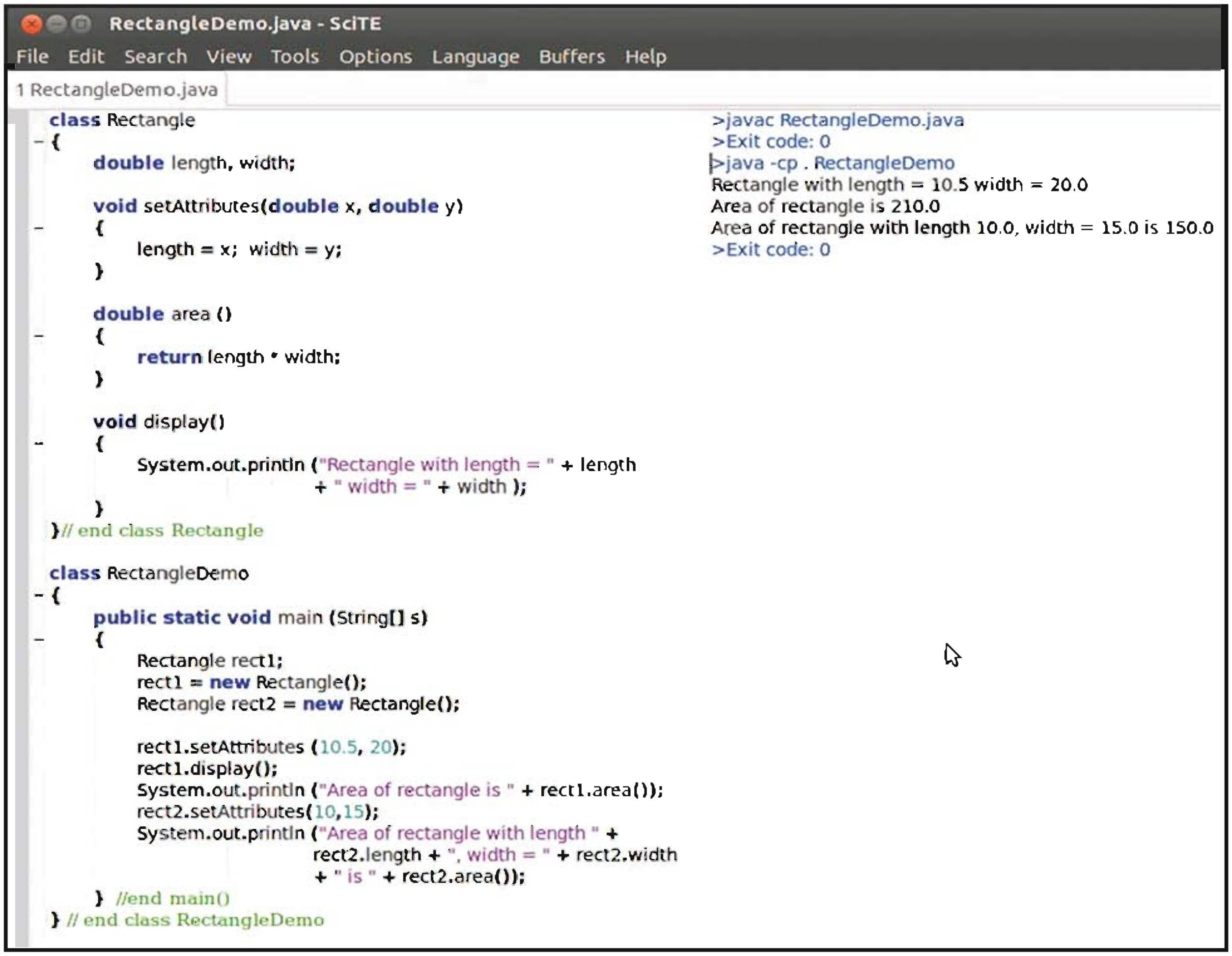
આપણે દરેક વ્યક્તિને ઉપલબ્ધ બનાવવા માટે main( ) મેથડ સાથે public ચાવીરૂપ શબ્દનો ઉપયોગ કર્યો છે.

```
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 સોર્સફાઇલ 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 સોર્સફાઇલમાં (પૂર્વનિર્ધારિત package) વ્યાખ્યાયિત અન્ય ક્લાસ 'Rectangle Demo'માં તે સીધેસીધા મેળવી શકાય છે (accessible).




```
class Rectangle
{
    double length, width;

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

    double area ()
    {
        return length * width;
    }

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

class RectangleDemo
{
    public static void main (String[] s)
    {
        Rectangle rect1;
        rect1 = new Rectangle();
        Rectangle rect2 = new Rectangle();

        rect1.setAttributes (10.5, 20);
        rect1.display();
        System.out.println ("Area of rectangle is " + rect1.area());
        rect2.setAttributes(10,15);
        System.out.println ("Area of rectangle with length " +
                        rect2.length + ", width = " + rect2.width
                        + " is " + rect2.area());
    } //end main()
} // end class RectangleDemo
```

```
>javac RectangleDemo.java
>Exit code: 0
>java -cp . RectangleDemo
Rectangle with length = 10.5 width = 20.0
Area of rectangle is 210.0
Area of rectangle with length 10.0, width = 15.0 is 150.0
>Exit code: 0
```

આકૃતિ 8.9 : પૂર્વનિર્ધારિત વિઝિબિલિટી મોડિફાયર, પૅકેજમાં સર્વત્ર ઉપલબ્ધ

પ્રોટેક્ટેડ (Protected)

આ સ્તરના પ્રોટેક્શનનો ઉપયોગ ફક્ત સબક્લાસને ઍક્સેસ 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 ઍક્સેસ 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 *

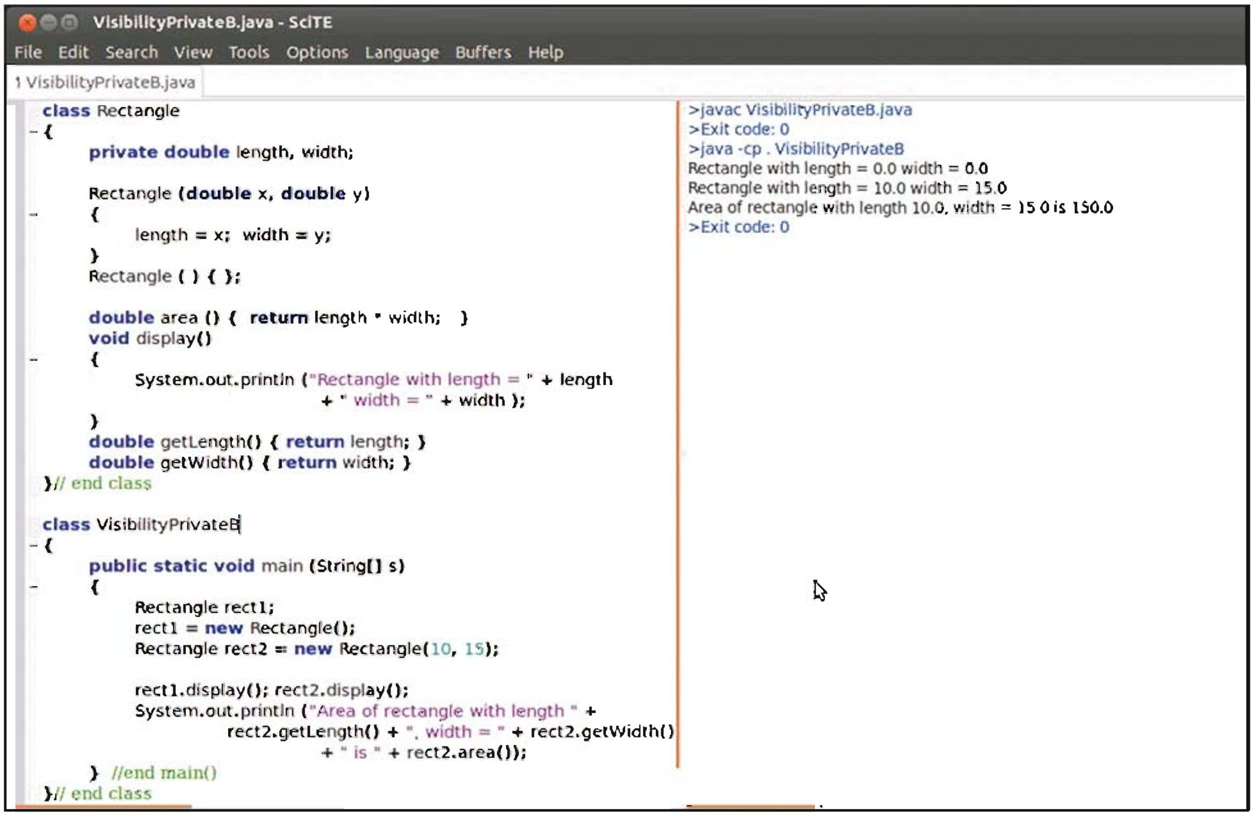
```
class Rectangle
{
    private double length, width;

    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.




```
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 );
      }
      double getLength() { return length; }
      double getWidth() { return width; }
}// end class

class VisibilityPrivateB
{
      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.getLength() + ", width = " + rect2.getWidth()
                        + " is " + rect2.area());
      } //end main()
}// end class
```

```
>javac VisibilityPrivateB.java
>Exit code: 0
>java -cp . VisibilityPrivateB
Rectangle with length = 0.0 width = 0.0
Rectangle with length = 10.0 width = 15.0
Area of rectangle with length 10.0, width = 15.0 is 150.0
>Exit code: 0
```

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.

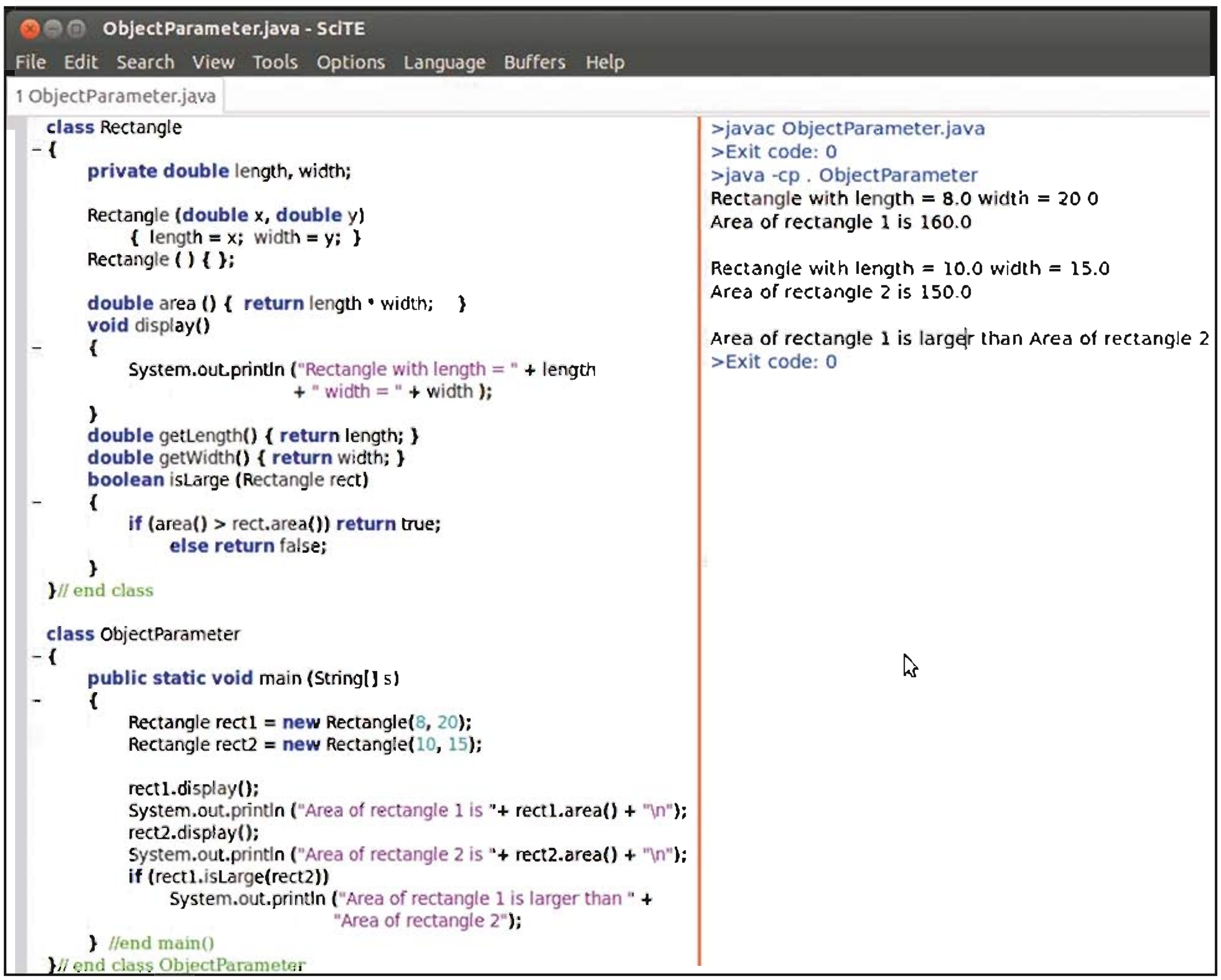
મેથડમાં પ્રાચલ તરીકે ઑબ્જેક્ટ પસાર કરવો (Passing Object as a Parameter in a Method)

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 ઇન્વૉક 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 કોલ જુઓ : rect1.isLarge(rect2). અહીં, 'rect1' એ કોલિંગ અથવા ઇન્વૉકિંગ 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.

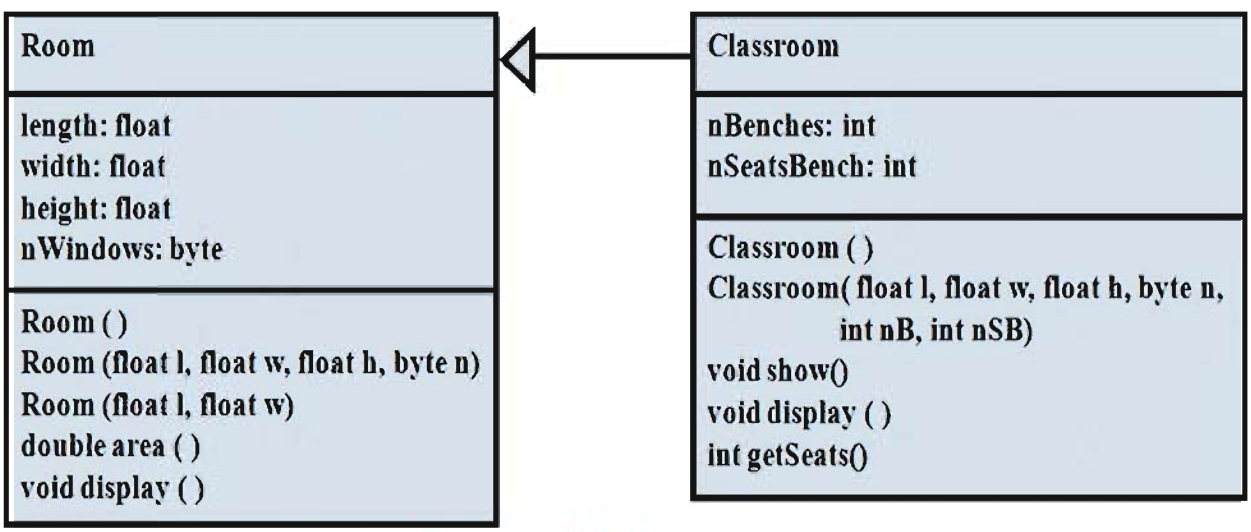
ઈનહેરિટન્સ (Inheritance)

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

ઈનહેરિટન્સ 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 ચાઈલ્ડક્લાસ (child class) કહેવામાં આવે છે. પેરન્ટક્લાસને સુપર ક્લાસ (superclass) અથવા બેઈઝ ક્લાસ (base class) પણ કહેવામાં આવે છે. એ જ પ્રકારે ચાઈલ્ડક્લાસને સબક્લાસ (subclass) અથવા ઍક્સટેન્ડેડ ક્લાસ (extended class) પણ કહેવામાં આવે છે.

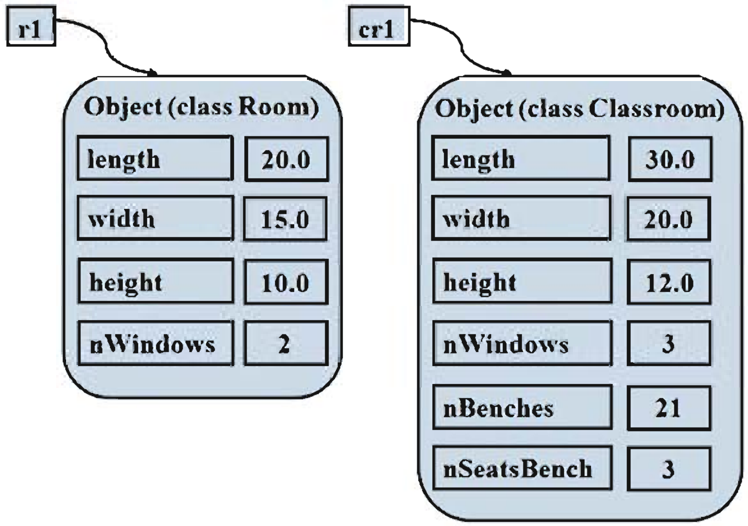
જ્યારે બે ક્લાસ વચ્ચે 'is-a' પ્રકારનો સંબંધ હોય છે, ત્યારે આપણે ઈનહેરિટન્સનો ઉપયોગ કરીએ છીએ. સામાન્ય ગુણધર્મો સુપર ક્લાસમાં રાખવામાં આવે છે. સબક્લાસ સુપર ક્લાસમાંથી બધા ઈન્સ્ટન્સ વેરિયેબલ અને મેથડ વારસામાં મેળવે છે અને તેના પોતાના વધારાના વેરિયેબલ અને મેથડ હોઈ શકે. અહીં નોંધ કરશો કે સબક્લાસમાં કન્સ્ટ્રક્ટર 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 : ઈનહેરિટન્સ ક્લાસ ડાયાગ્રામ

જ્યારે સબક્લાસનો ઑબ્જેક્ટ ઈન્સ્ટન્સિયેટ (instantiate) થાય છે, ત્યારે ઈનહેરિટ થયેલાં સાથે તેના બધા ઍટ્રિબ્યૂટને મેમરી ફાળવવામાં આવે છે. આકૃતિ 8.14માં સુપર ક્લાસ Room અને સબક્લાસ classroomના ઈન્સ્ટન્સ 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 ઍટ્રિબ્યૂટ પ્રદર્શિત કરવા માટે show() મેથડનો ઉપયોગ કર્યો છે. સુપરક્લાસ 'Room'માંથી ઇનહેરિટ (inherit) થયેલા પ્રથમ ચાર ઍટ્રિબ્યૂટર 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 કોલ કરવાનો છે. આ માટે આપણે super.display()નો ઉપયોગ કર્યો છે.

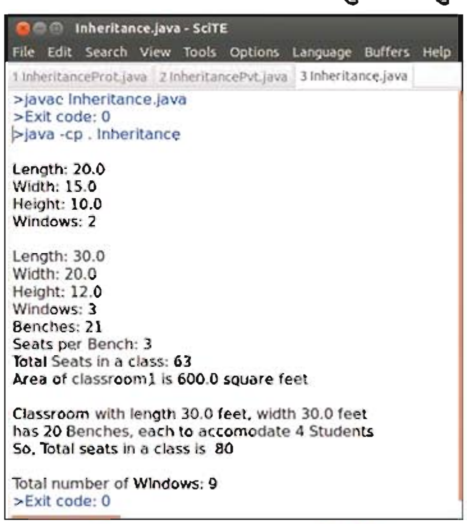
જ્યારે સુપર ક્લાસ અને સબક્લાસમાં એકસમાન સિગ્નેચર (signature) સાથેની મેથડ હોય, ત્યારે સુપર ક્લાસ મેથડની સબક્લાસમાં ઉપેક્ષા કરવામાં આવે છે. સબક્લાસની display() મેથડ સુપરક્લાસની display() મેથડની ઉપેક્ષા કરે છે. એનો અર્થ એ થાય કે આપણે સુપર ક્લાસની display() મેથડનો ફરી ઉપયોગ કર્યા વિના માહિતી જુદી રીતે પ્રદર્શિત 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 ઇન્વૉક 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 ઇન્વૉક કરી શકાય છે. જ્યારે સબક્લાસના ઑબ્જેક્ટનો ઉપયોગ કરી વિનિયોગમાં ઉપેક્ષા કરેલી મેથડનો ઉલ્લેખ કરવામાં આવે છે, ત્યારે તે સબક્લાસની મેથડ cr2.display()ને કોલ 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

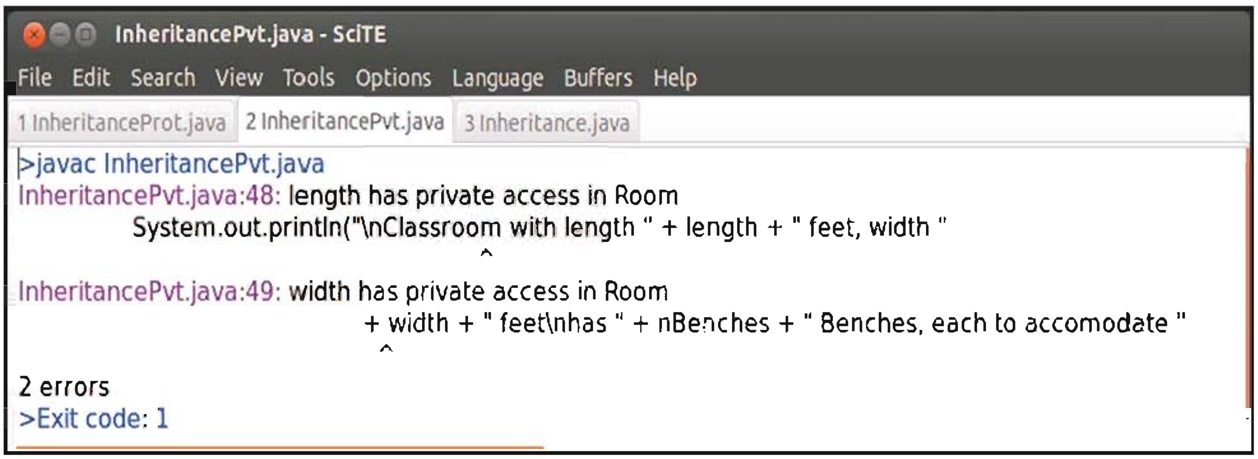
સુપર ક્લાસના પ્રાઈવેટ મેમ્બર (Private Members of Superclass)

અગાઉના ઉદાહરણમાં બધા જ ઇન્સ્ટન્સની પેકેજ વિઝિબિલિટી હતી. આથી, તે આખા પેકેજમાં 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 મૅથડ 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 પ્રાઇવેટ 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' મૅથડનો 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 વર્તમાનપત્રની 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 : કૉમ્પોઝિશન 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 ઇન્વૉક 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 એસાઇન્મેન્ટ વિધાન 'readingRoom = r;' વાપરીને Roomના ઍટ્રિબ્યૂટને કિંમતો એસાઇન (assign) કરે છે.
    - display() મેથડ વ્યાખ્યાયિત કરેલી છે 'readingRoom.display();' દ્વારા 'Room' ક્લાસની display() મેથડ ઇન્વૉક 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 ઇન્વૉક કરવી તે બાબત અહીં શીખ્યા. કન્સ્ટ્રક્ટર એ ક્લાસના નામની જ, ચલ વગરની અને પરત પ્રકાર વગરની વિશિષ્ટ મેથડ છે. જ્યારે ઑબ્જેક્ટ નવા પ્રક્રિયક સાથે ઇન્સ્ટેન્સિયેટ 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 એક્સેસ 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 ઇન્સ્ટેન્સિએશન 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 એટ્રિબ્યૂટ 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 ઍસાઇન 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)