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 વેબસાઇટ 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 ઇન્ટરનેટ સાથે જોડાણ ન હોય, તો પ્રોગ્રામ અનઅપેક્ષિત પરિણામો આપે તેવું બને. જ્યારે કોઈ પ્રોગ્રામમાં આપણે ફાઇલ સાથે કામ કરતાં હોઈએ, ત્યારે પણ આવું જ બનશે. ઉદાહરણ તરીકે, ધારો કે કોઈ પ્રોગ્રામ માત્ર વાંચી શકાય તેવી (Read Only) ફાઇલમાં સુધારા-વધારા કરવા પ્રયત્ન કરતો હોય અથવા એવી ફાઇલને 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 ''કંપાઇલ 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 કંપાઇલ 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 કંપાઇલરનો 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" ફાઇલ 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.

(Untitled) * SciTE

File Edit Search View Tools Options Language Buffers Help

Untitled *

```
/* A program which contains compilation error */
/* Semicolon mission on line number -  8  */

class ErrorDemo
{
        public static void main(String args[])
        {
                System.out.println("Hello World")  // No Semicolon
        }
}
```

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.

```
ErrorDemo.java - SciTE
File Edit Search View Tools Options Language Buffers Help
ErrorDemo.java

>javac ErrorDemo.java
ErrorDemo.java:8: ';' expected
        System.out.println("Hello World")  // No Semicolon
                                         ^
1 error
>Exit code: 1
```

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.

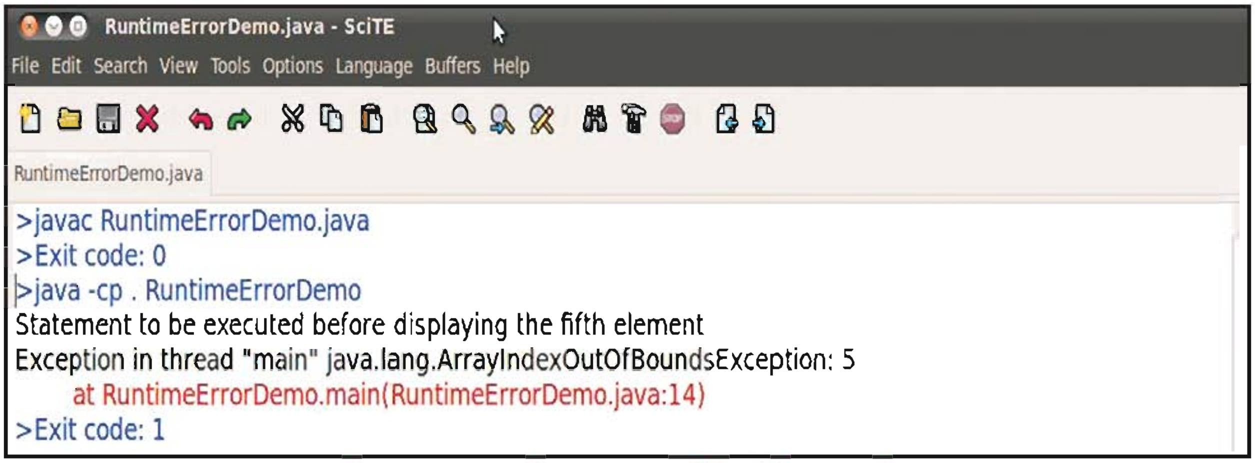
પ્રોગ્રામના અમલ દરમિયાન ઉદ્ભવતી ભૂલો

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.

```
RuntimeErrorDemo.java - SciTE
File Edit Search View Tools Options Language Buffers Help
RuntimeErrorDemo.java

1  /* A program which generates run-time error. An array of four elements is created but the program tries to
2     access fifth element of the citylist[] array resulting in run-time error */
3
4  class RuntimeErrorDemo
5  {
6      public static void main(String args[])
7      {
8          /* Create an array of four elements */
9          String citylist[] = {"Ahmedabad", "Baroda", "Rajkot", "Surat"};
10
11         System.out.println("Statement to be executed before displaying the fifth element");
12
13         /* Statement that generates run-time error   */
14         System.out.println(citylist[5]);
15
16         System.out.println("Statement to be executed after displaying the fifth element");
17     }
18 }
```

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 ઍરે બનાવ્યો છે, જે ચાર જુદાં-જુદાં શહેરોનાં નામ સાચવશે. 14મી સૂચનામાં આપણે citylist નામના ઍરેના 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.

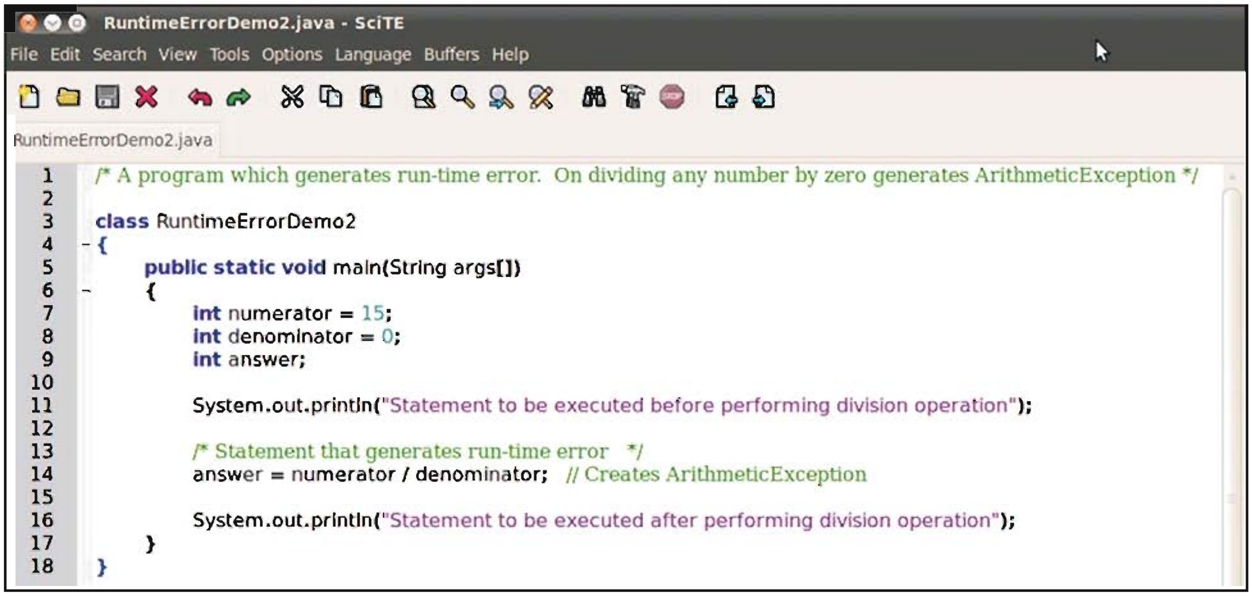
હવે આપણે આવા કેટલાક અન્ય કિસ્સા જોઈએ કે જે અપવાદ સર્જે છે. જાવામાં દરેક પ્રકારના અપવાદ માટે, સંબંધિત એક્સેપ્શન ક્લાસ (Exception class) ઉપલબ્ધ છે.

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]; 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; System.out.println(s.length());
NumberFormatException	કોઈ શબ્દમાળા (string)ને સાંખ્યિક રૂપમાં ફેરવવા માટેનો પ્રયત્ન.	String s = "xyz"; 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 દ્રષ્ટાંત 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.

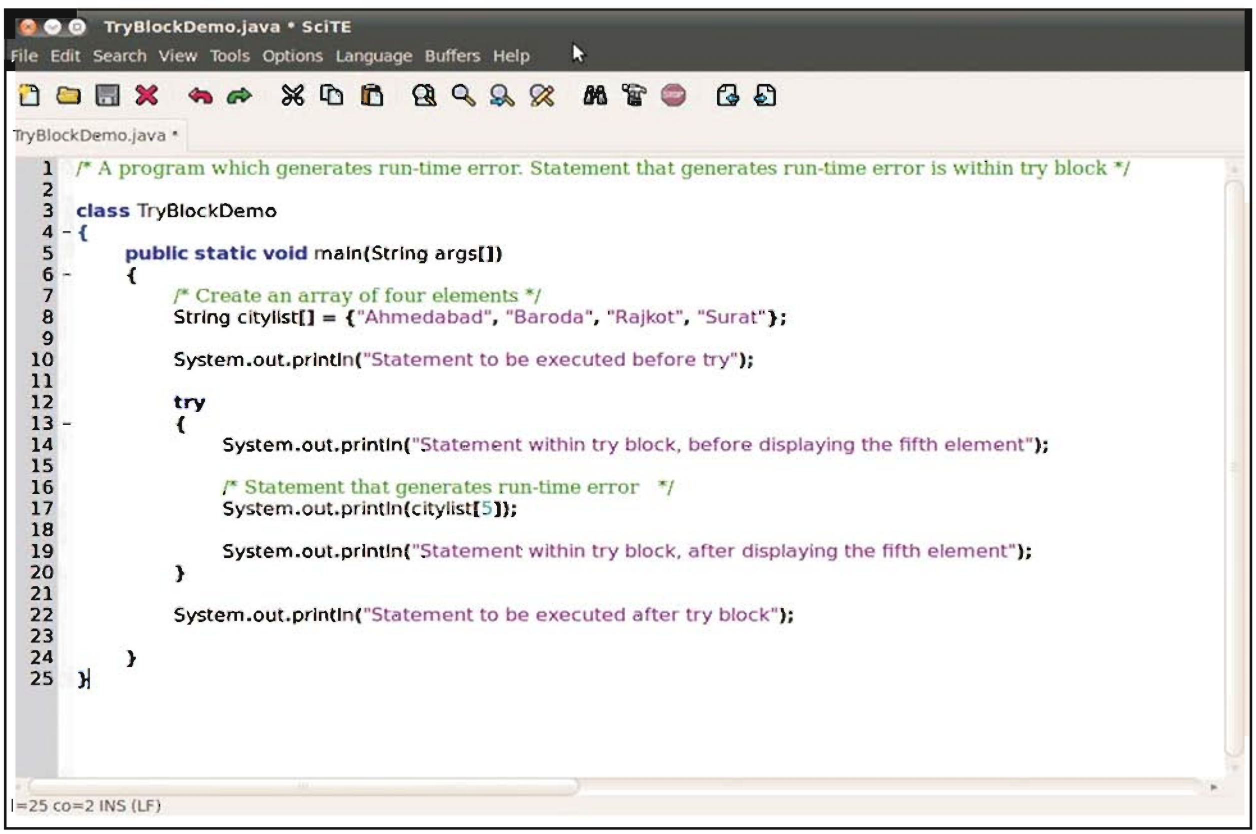
હવે આ ત્રણેય વિભાગોને એક પછી એક વિગતવાર સમજીએ.

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 :

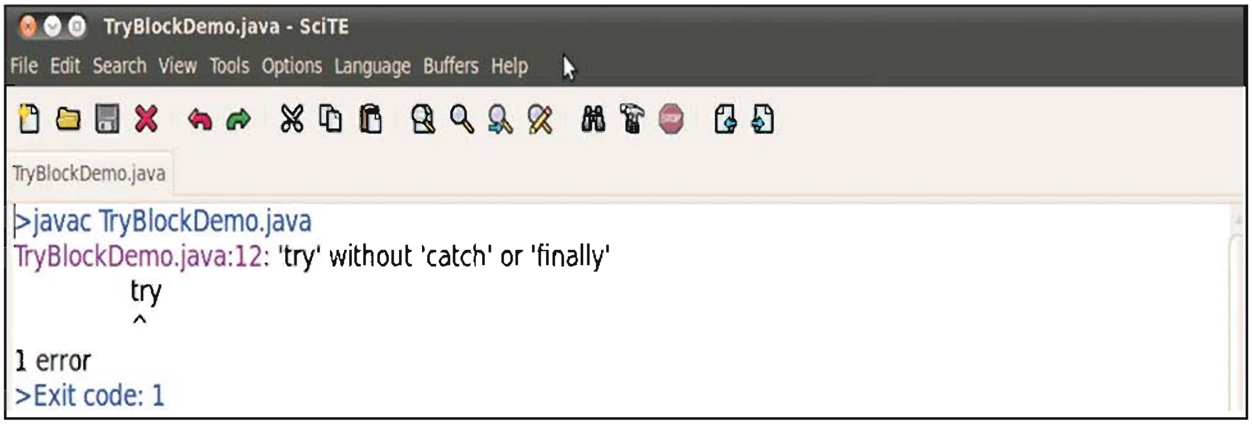
```
try
{
   // set of statements that may generate one or more exceptions.
}
```

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 બ્લૉક પછી catch વિભાગ અથવા finally વિભાગ હોવો જરૂરી છે. કંપાઇલેશન દરમિયાન સર્જાતી ભૂલ આકૃતિ 10.8માં દર્શાવેલ છે.



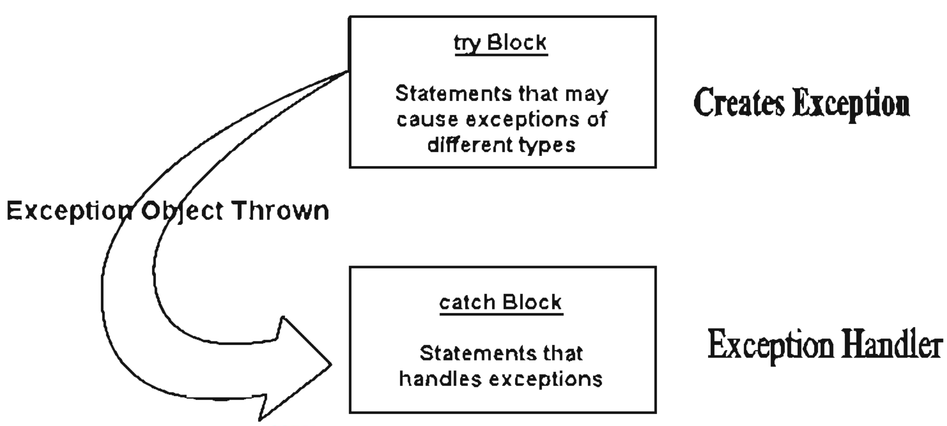
આકૃતિ 10.8 : આકૃતિ 10.7માં દર્શાવેલ પ્રોગ્રામની કંપાઈલેશન દરમિયાન સર્જાયેલ ભૂલ

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

```
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 }
```

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

```
>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
```

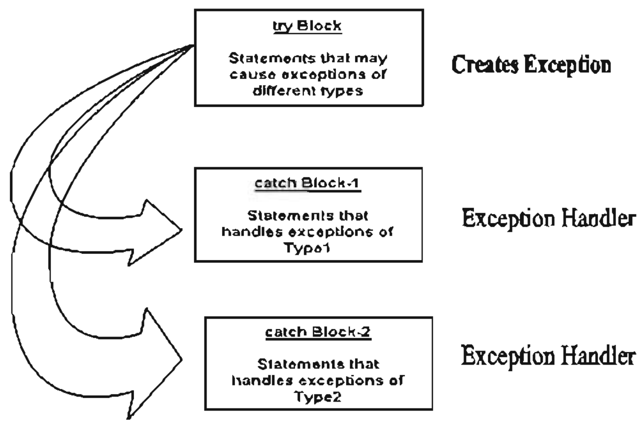
આકૃતિ 10.11 : આકૃતિ 10.10માં દર્શાવેલ પ્રોગ્રામનું પરિણામ

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

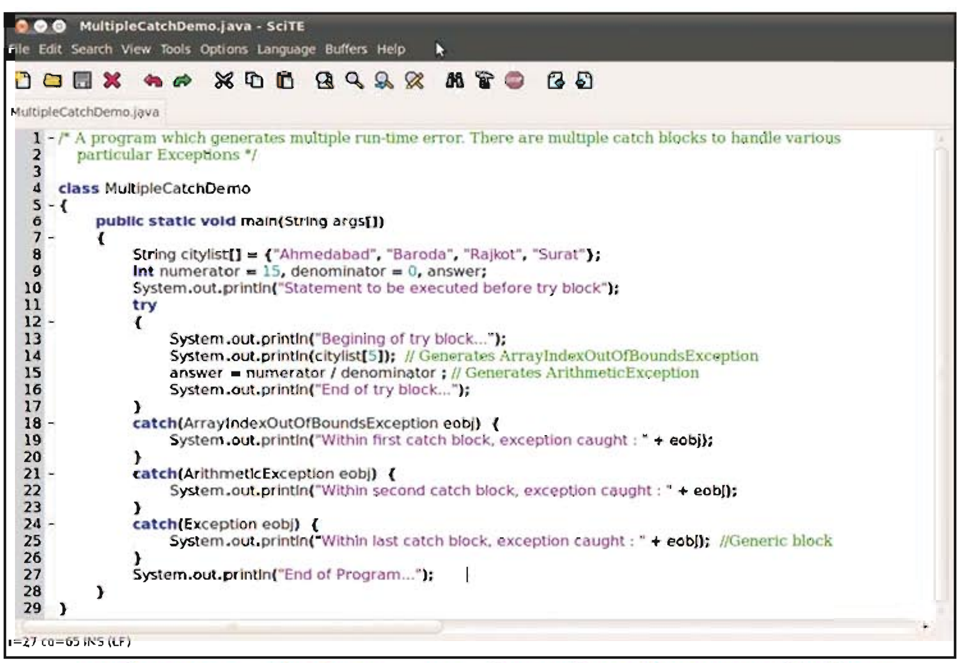
એક જ પ્રોગ્રામમાં અનેક અપવાદરૂપ પરિસ્થિતિ સર્જાઈ શકે છે. ઉદાહરણ તરીકે, ધારો કે આપણે એક ચોક્કસ ફાઈલને દૂરસ્થિત કમ્પ્યૂટર પર અપલોડ કરવી છે, તો તે બે સ્પષ્ટ અપવાદો તરફ દોરી જશે. એક, જો ફાઈલ આપણા કમ્પ્યૂટર પર ઉપલબ્ધ નહીં હોય, બીજો જો આપણું કમ્પ્યૂટર ઈન્ટરનેટ સાથે જોડાણ ધરાવતું ન હોય. એક કરતાં વધુ અપવાદોને 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.




```
/* A program which generates multiple run-time error. There are multiple catch blocks to handle various
   particular Exceptions */

class MultipleCatchDemo
{
    public static void main(String args[])
    {
        String citylist[] = {"Ahmedabad", "Baroda", "Rajkot", "Surat"};
        int numerator = 15, denominator = 0, answer;
        System.out.println("Statement to be executed before try block");
        try
        {
            System.out.println("Begining of try block...");
            System.out.println(citylist[5]);  // Generates ArrayIndexOutOfBoundsException
            answer = numerator / denominator ; // Generates ArithmeticException
            System.out.println("End of try block...");
        }
        catch(ArrayIndexOutOfBoundsException eobj) {
            System.out.println("Within first catch block, exception caught : " + eobj);
        }
        catch(ArithmeticException eobj) {
            System.out.println("Within second catch block, exception caught : " + eobj);
        }
        catch(Exception eobj) {
            System.out.println("Within last catch block, exception caught : " + eobj);  //Generic block
        }
        System.out.println("End of Program...");
    }
}
```

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 બ્લૉકનો 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 *

```java
/* A program which generates multiple run-time error. There is a finally block following try block.
   There are no catch blocks in this program */

class FinallyDemo
{
	public static void main(String args[])
	{
		String citylist[] = {"Ahmedabad", "Baroda", "Rajkot", "Surat"};
		int numerator = 15, denominator = 0, answer;
		System.out.println("Statement to be executed before try block");
		try
		{
			System.out.println("Begining of try block...");
			System.out.println(citylist[5]);  // Generates ArrayIndexOutOfBoundsException
			answer = numerator / denominator ; // Generates ArithmeticException
			System.out.println("End of try block...");
		}
		finally
		{
			System.out.println("This part of code will always get executed");
		}

		System.out.println("End of Program...");
	}
}
```

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.

FinallyDemo.java - SciTE

```
>javac FinallyDemo.java
>Exit code: 0
>java -cp . FinallyDemo
Statement to be executed before try block
Begining of try block...
This part of code will always get executed
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
	at FinallyDemo.main(FinallyDemo.java:14)
>Exit code: 1
```

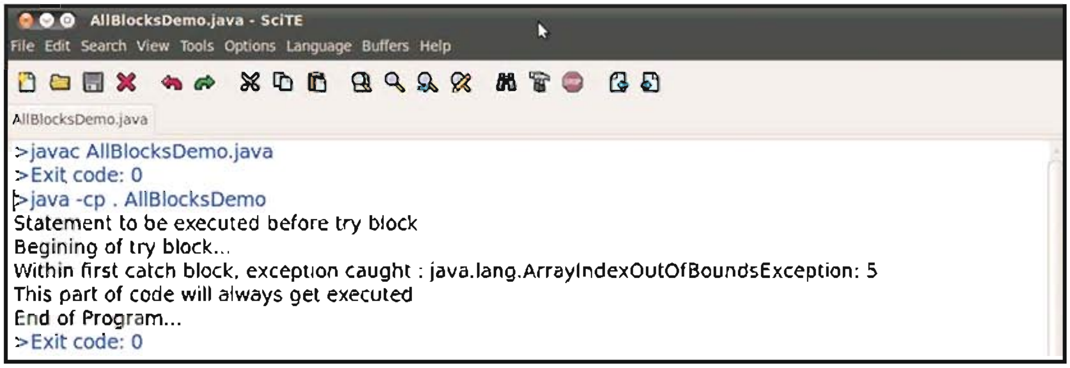
આકૃતિ 10.15 : આકૃતિ 10.14માં દર્શાવેલ પ્રોગ્રામનું પરિણામ

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.

AllBlocksDemo.java - SciTE

```java
/* A program which generates multiple run-time error. There are multiple catch blocks to handle various
   particular Exceptions */
class AllBlocksDemo
{
    public static void main(String args[]) {
        String citylist[] = {"Ahmedabad", "Baroda", "Rajkot", "Surat"};
        int numerator = 15, denominator = 0, answer;
        System.out.println("Statement to be executed before try block");
        try {
            System.out.println("Begining of try block...");
            System.out.println(citylist[5]);  // Generates ArrayIndexOutOfBoundsException
            answer = numerator / denominator ; // Generates ArithmeticException
            System.out.println("End of try block...");
        }
        catch(ArrayIndexOutOfBoundsException eobj) {
            System.out.println("Within first catch block, exception caught : " + eobj);
        }
        catch(ArithmeticException eobj) {
            System.out.println("Within second catch block, exception caught : " + eobj);
        }
        catch(Exception eobj) {
            System.out.println("Within last catch block, exception caught : " + eobj);  //Generic block
        }
        finally {
            System.out.println("This part of code will always get executed");
        }
        System.out.println("End of Program...");
    }
}
```

આકૃતિ 10.16 : try, catch અને finally વિભાગ સમજાવતો પ્રોગ્રામ



```
AllBlocksDemo.java - SciTE
File Edit Search View Tools Options Language Buffers Help
AllBlocksDemo.java
>javac AllBlocksDemo.java
>Exit code: 0
>java -cp . AllBlocksDemo
Statement to be executed before try block
Begining of try block...
Within first catch block, exception caught : java.lang.ArrayIndexOutOfBoundsException: 5
This part of code will always get executed
End of Program...
>Exit code: 0
```

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 કંપાઇલ થઈ શકશે નહીં.

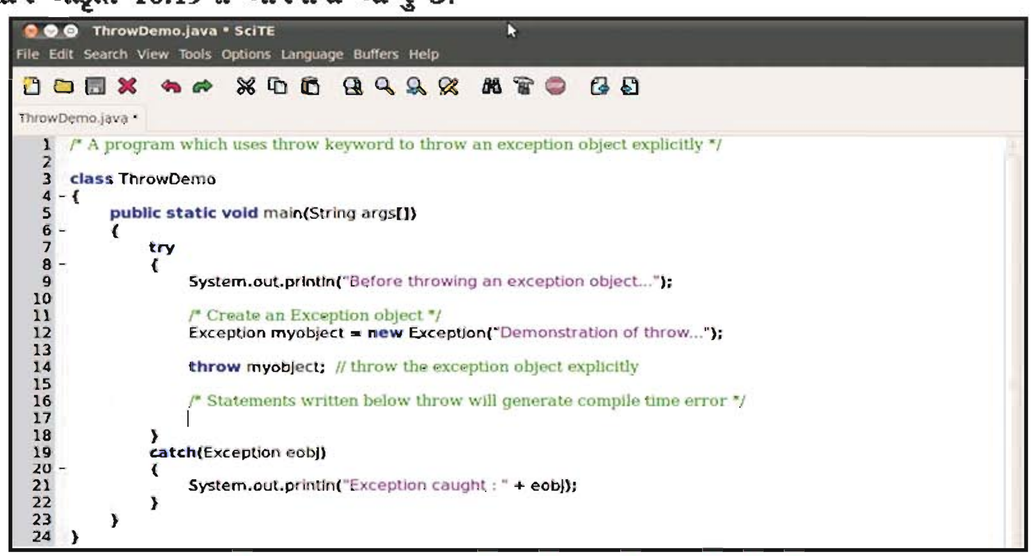
Throw વિધાન

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 કંપાઇલ કરતી વખતે ભૂલ ઉદ્દ્ભવશે. અપવાદ ઑબ્જેક્ટને થ્રો કરવા માટેની વાક્યરચના નીચે મુજબ છે :

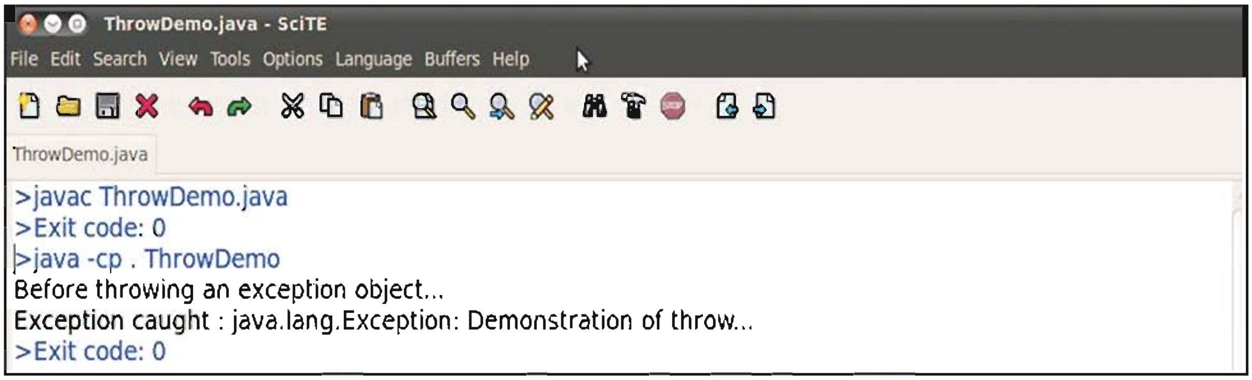
```
throw exception_object;
```

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.



```
ThrowDemo.java * SciTE
File Edit Search View Tools Options Language Buffers Help
ThrowDemo.java *
/* A program which uses throw keyword to throw an exception object explicitly */

class ThrowDemo
{
	public static void main(String args[])
	{
		try
		{
			System.out.println("Before throwing an exception object...");

			/* Create an Exception object */
			Exception myobject = new Exception("Demonstration of throw...");

			throw myobject; // throw the exception object explicitly

			/* Statements written below throw will generate compile time error */

		}
		catch(Exception eobj)
		{
			System.out.println("Exception caught : " + eobj);
		}
	}
}
```

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.

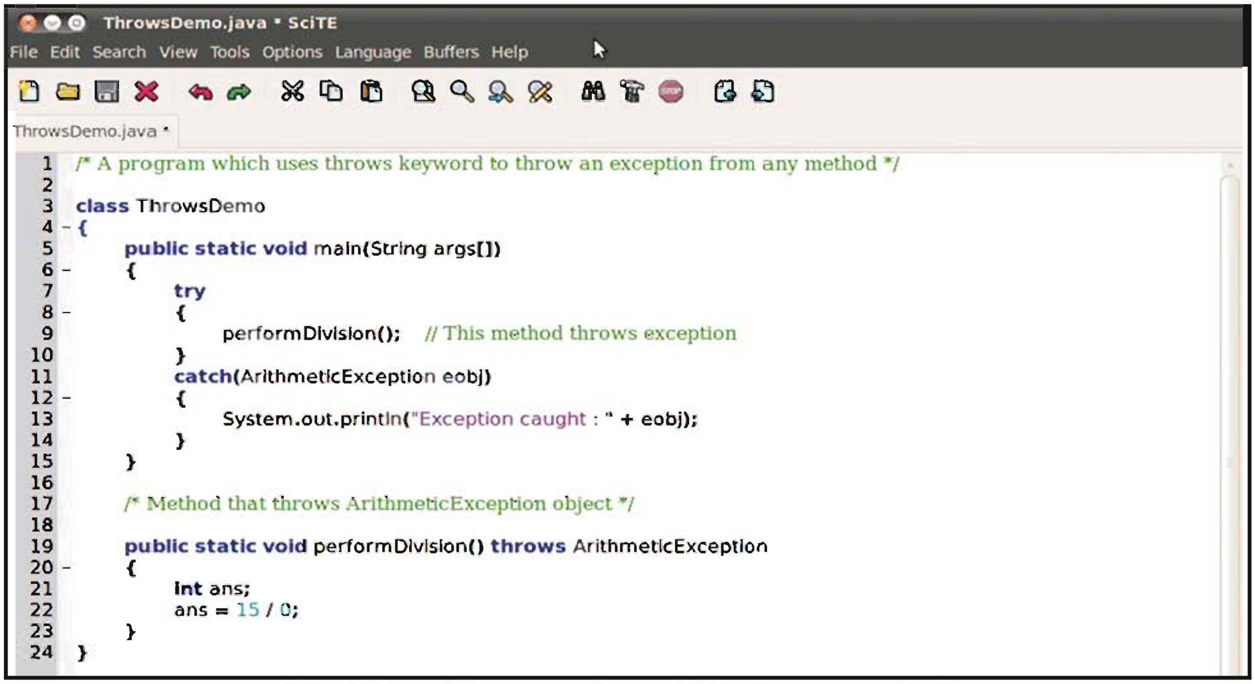
methodની ઘોષણા કરતી વખતે throws ઉપવાક્યનો નીચે મુજબ ઉપયોગ કરી શકાય છે :

```
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 ફાઈલ 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.

```
CustomExceptionDemo2.java - SciTE
File Edit Search View Tools Options Language Buffers Help
CustomExceptionDemo2.java

/* A program that creates Custom Exceptions */
/* Class InvalidMarksException is a user defined class which is inherited from java.lang.Exception class
   This program wont proceed untill valid marks are entered*/

import java.util.Scanner;

class InvalidMarksException extends java.lang.Exception
{
	public InvalidMarksException(String message)
	{
		super(message);
	}
}


class CustomExceptionDemo2
{
	public static void main(String args[])
	{
		Scanner kbinput = new Scanner(System.in);
		int marks;
		boolean continueLoop = true;
		do {
			System.out.println("Enter the marks : ");
			marks = kbinput.nextInt();
			System.out.println("You entered " + marks);
			try
			{
				if(marks < 0 || marks > 100)  {
					throw new InvalidMarksException("Wrong marks..."); // Throw Customized Exception obj
				}
				else {
					System.out.println("Marks are Valid");
				}
				continueLoop = false;
			}
			catch(InvalidMarksException eobj)
			{
				System.out.println("Exception caught : " + eobj);
			}
		}
		while(continueLoop);
	}
}
```

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.

```
Terminal
File Edit View Terminal Help

$ javac CustomExceptionDemo2.java
$ java CustomExceptionDemo2
Enter the marks :
145
You entered 145
Exception caught : InvalidMarksException: Wrong marks...
Enter the marks :
-47
You entered -47
Exception caught : InvalidMarksException: Wrong marks...
Enter the marks :
78
You entered 78
Marks are Valid
$
```

આકૃતિ 10.22 : આકૃતિ 10.21માં દર્શાવેલ પ્રોગ્રામનું પરિણામ

નોંધ : SciTE એ પ્રોગ્રામ ટાઈપ 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. કંપાઈલેશન 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) ટૂંકાણ (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 ઍરેમાં 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, બૅંકવ્યવહારોને 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, બૅંકવ્યવહારોને 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.