Website

https://www.curiousinfotechsolutions.com

Managing Errors and Exceptions

Java Programming

Day-19

Email id:
info@curiousinfotechsolutions.com

Syllabus Day-19

- **Brief**
  - ✓**Why multiple inheritance is not supported by Java?**

Errors..Introduction

- Error is common to make mistak *while developing as well as typing a program*.
- A mistake might **lead to an error** causing the program to produce unexpected results.
- Errors are the **wrongs** that can *make a program go wrong*.
- An error may produce an **incorrect output** or may **terminate the execution of the program abruptly** or **even may cause the system to crash**.
- It is therefore important to **detect and manage properly** all the possible error conditions in the program so that ***program will not terminate or crash during execution.***

Types of Errors

- Errors may broadly be ***classified into two categories***:
  - Compile-time Error
  - Run-time Error
- Compile-time Error
  - All syntax errors will be detected and displayed by the java Complier and therefore these errors are known as compile-time errors.
  - Whenever the compile displays an error, it will not create the .class file.
  - It is therefore necessary that we fix all the errors before we can successfully compile and run the program.
- Run-time Error
  - A runtime error occurs when a program is syntactically correct but contains an issue that is only detected during program execution.
  - These issues cannot be caught at compile-time by the Java compiler and are only detected by the Java Virtual Machine (JVM) when the application is running.

Illustration of Compile-time errors


```
/* This program contains an error */
1.   class Error
2.   {
3.   public static void main(String args[])
4.   {
5.   System.out.println("Hello Java")  //Missing ;
6.   }
7.   }
```

```
Compilation failed due to following
error(s).Main.java:4: error: ';' expected
                        System.out.println("Hello World")
                                                        ^
1 error
```

The most common problems are:

- Missing semicolons
- Missing (or Mismatch of) brackets in classes and methods
- Misspelling of identifiers and keywords
- Missing double quotes in strings
- Use of undeclared variables
- Incompatible types in assignments / initialization
- Bad references to objects
- Use of = in place of == operator

Run-time Errors

- Dividing an integer by Zero.
- Accessing an element that is out of the bounds of an array.
- Trying to store a value into an array of an incompatible class or type.
- Trying to cast an instant of a class to one of it subclasses.
- Passing a parameter that is not in a valid range or value for a method.
- Trying to illegally change the state of a thread.
- Attempting to use a negative size for an array.
- Converting invalid string to a number.
- Accessing a character that is out of bounds of a string.

Illustration of Run-time errors


```
/* This program contains an error */
1.   class Main
2.   {
3.     public static void main(String args[])
4.    {
5.      int a=10, b=5, c=5;
6.      int x = a / (b-c); //Division by zero
7.      System.out.println(x) ;
8.      int y = a / (b+c);
9.      System.out.println(x) ;
10.  }
11. }
```

```
Exception in thread "main"
java.lang.ArithmeticException: / by zeroat
Main.main(Main.java:6)
```

Exceptions - Introduction

- An exception is a condition that is ***caused by a run-time error in the program.***
- When the Java interpreter encounters an error such as dividing an integer by zero.
- It creates an exception object and throws it (i.e. ***informs us that an error has occurred***).
- The purpose of exception handling mechanism is to ***provide a means to detect and report*** an "Exceptional circumstance" so that appropriate action can be taken.

Exceptions performs the following tasks:

- Find the problem
  - ***Means Hit the exception***
- Inform that an error has occurred
  - ***Means throw the exception***
- Receive the error information
  - ***Means catch the exception***
- Take corrective actions
  - ***Means handle the exception***

Common Java Exceptions

Exception Type	Cause of Exception
ArithmeticeException	Caused by math errors such as division by zero
ArrayIndexOutOfBoundsException	Caused by bad array indexes
ArrayStoreException	Caused when a program tries to store the wrong type of data in an array
FileNotFoundException	Caused by an attempt to access a non-existent file
IOException	Caused by general I/O failures, such as inability to read from a file
NullPointerException	Caused by referencing a null object.
NumberFormatException	Caused when a conversion between strings and number fails
OutOfMemoryException	Caused when there's not enough memory to allocate a new object
SecurityOverflowException	Caused when the system runs out of stack space
SringIndexOutOfBoundsException	Caused when a program attempts to access a non-existent character position in a string

Syntax of Exception Handling Code

Exception object creator

Try Block

Statement that causes an exception

Throws exception object

Exception handler

catch Block

Statement that handles the exception

Use of simple try and catch statement

Syntax

……

……

try

{

Statement

}

Catch (exception-type e)

{

Statement

}

……

……

```
class Main
{
   public static void main(String args[])
   {
      int a=10, b=5, c=5;
      try {
      int x = a / (b-c); //Division by zero
      System.out.println(x);
      } catch (ArithmeticException e) {
         System.out.println("Division by zero");
      }
      int y = a / (b+c);
      System.out.println(y) ;
   }
}
```

Catching invalid command line arguments


```
class Main
{  public static void main(String args[])
   {  int invalid = 0, valid=0, number;
      for (int i = 0; i < args.length; i++)
      { try{ number = Integer.parseInt(args[i]); }
         catch(NumberFormatException e){
         invalid = invalid + 1;
         System.out.println("invalid number: "+ args[i]);
         continue;
         }
         valid = valid + 1;
      }
      System.out.println("Total valid Number are:"+valid);
      System.out.println("Total invalid Number are:"+invalid);
   }
}
```

```
F:\Skill_Updates\JavaProg>javac Main.java

F:\Skill_Updates\JavaProg>java Main 15 25.75 40 Java 10.5 65
invalid number: 25.75
invalid number: Java
invalid number: 10.5
Total valid Number are:3
Total invalid Number are:3

F:\Skill_Updates\JavaProg>
```

Multiple catch statement

- try
- {
  - Statement
- }
- catch (Exception-type-1 e)
- {
  - Statement
- }
- catch (Exception-type-2 e)
- {
  - Statement
- }

```
class Main {
  public static void main(String args[]){
    int a[] = {5,10}, b = 5;
      try {
        int x = a[2] / b - a[1];
      } catch(ArithmeticException e) {
        System.out.println("Division by zero");
      } catch(ArrayIndexOutOfBoundsException e){
        System.out.println("Array index error");
      } catch(ArrayStoreException e){
        System.out.println("Wrong Data Type");
      }
      int y = a[1] / a[0]; System.out.println("y="+y);
  }
}
```

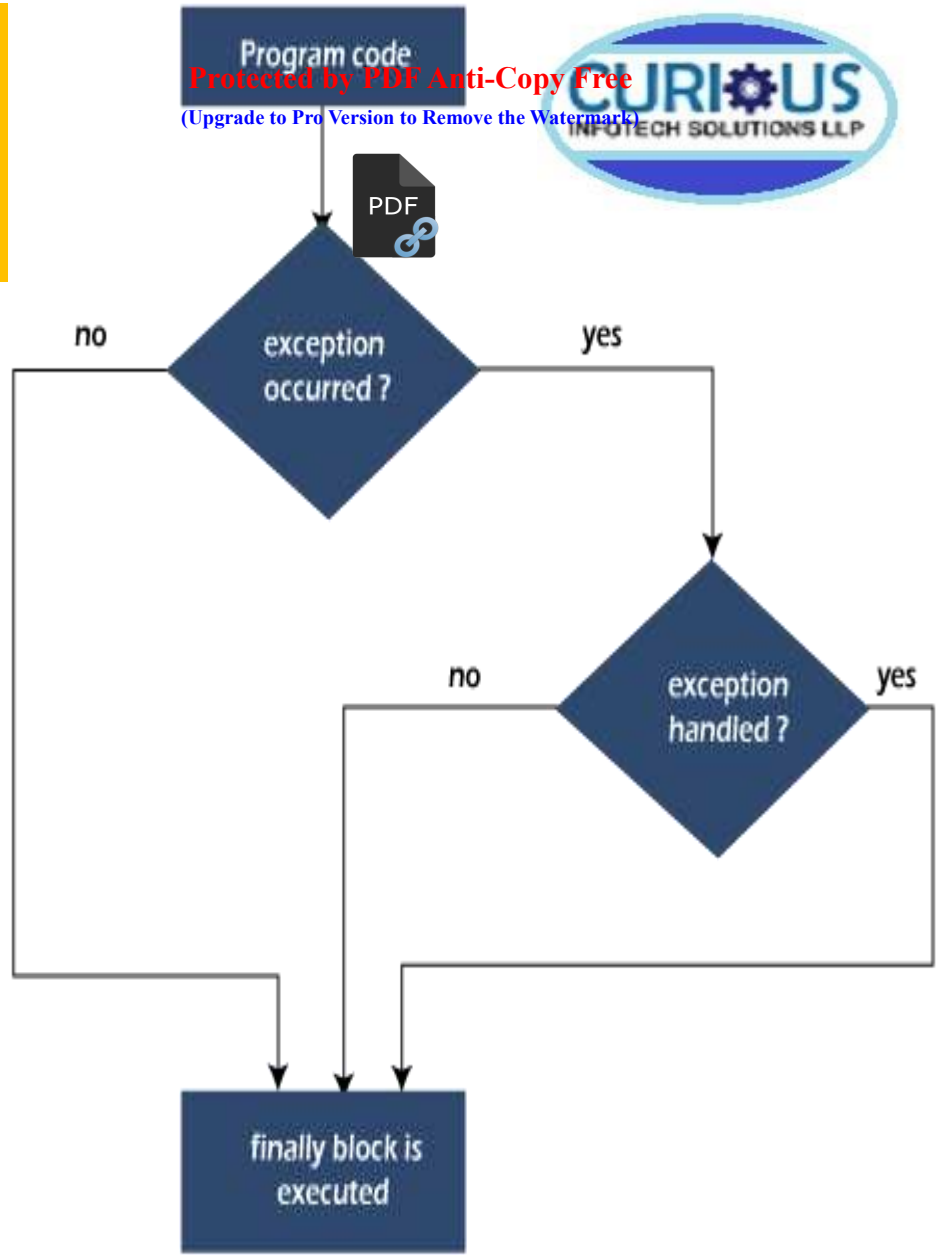
Array index error
y=2

Using finally Statement

- Java supports another statement know as finally statement that can be used to handle an exception that is not caught by any of the previous catch statements.
- finally block can be used to handle any exception generated within a try block.
- It may be added immediately after the try block or after the last catch block.

Contd..*finally*

- try
- {
  - .....
  - .....
- }
- finally
- {
  - .....
  - .....
- }



- try
- {
  - .....
- }
- catch
- {
  - .....
- }
- catch
- {
  - .....
- }
- finally
- {
  - .....
- }

```
class Main {
  public static void main(String args[]){
    try{
        int data=25/5;
        System.out.println(data);
    }
    catch(ArithmeticException e){
        System.out.println(e);
    }
    finally {
        System.out.println("finally block is always executed");
    }
    System.out.println("End of Program...");
  }
}
```



```
class Main {
  public static void main(String args[]){
    try{
        int data=25/0;
        System.out.println(data);
    }
    catch(ArithmeticException e){
        System.out.println(e);
    }
    finally {
        System.out.println("finally block is always executed");
    }
    System.out.println("End of Program...");
  }
}
```

5
finally block is always executed
End of Program...

java.lang.ArithmeticException: / by zero
finally block is always executed
End of Program...

Contd..*finally*


```
class Main {
  public static void main(String args[]){
     try{
          int data=25/0;
          System.out.println(data);
     }
     finally {
         System.out.println("finally block is always executed");
     }
     System.out.println("End of Program...");
  }
}
```

```
finally block is always executed
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Main.main(Main.java:4)
```

Throwing our own exception

- There may be times when we would like to throw our own exceptions.
- We can do this by using the keyword throw as:
  - throw new Throwable_subclass;
- Example:
  - throw new ArithmetricException();
  - throw new NumberFormatException();

Example..*throwing our own exception*


```
import java.lang.Exception;
class MyException extends Exception
{
    MyException(String msg)
    {
        super(msg);
    }
}
```

```
class Main
{
    public static void main (String[] args) {
        int x=5, y=1000;
        try {
            float z = (float) x / (float) y;
            if (z < 0.01)
            {
                throw new MyException("Number is too small");
            }
        } catch(MyException e) {
            System.out.println("Caught my exception");
            System.out.println(e.getMessage());
        } finally {
            System.out.println("I am always here");
        }
    }
```

Output

Caught my exception
Number is too small
I am always here