INHERITANCE:

Deriving new classes from existing classes such that the new classes acquire all the features of existing classes is called inheritance. The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class). **extends** is the keyword used to inherit the properties from one class to another class.

Syntax:

```
class SuperClass
{
   ..................
   ..................
}
class SubClass extends SuperClass
{
   ..................
   ..................
}
```

Example: Write a JAVA Program to derive classes from one class to another class.

```
class SuperClass
{
      void add(int a,int b)
      {
            int c=a+b;
            System.out.println("Addition is "+c);
      }
}
class SubClass extends SuperClass
{
      void sub(int a,int b)
      {
            int c=a-b;
            System.out.println("Subtraction is "+c);
      }
}
class Demo{
 public static void main( String args[] )
 {
       SuperClass s1=new SuperClass();
       s1.add(5,6);
       SubClass s2=new SubClass();
       s2.sub(15,10);
       s2.add(15,27);
 }
}
```

Output:

```
Addition is 11
Subtraction is 5
Addition is 42
```

When an object to SubClass is created, it contains a copy of SuperClass within it. This means there is a relation between the SuperClass and SubClass objects. This is the reason why SuperClass members are available to SubClass. Note that we do not create A SuperClass object, but still a copy of it is available to SubClass object.

Q: *What is the advantage of Inheritance?*
Ans: *In inheritance, a programmer reuses the super class code without rewriting it, in creation of sub classes. So, developing the classes becomes very easy. Hence, the programmer's productivity is increased.*

The 'super' Keyword:

If we create an object to super class, we can access only the super class members, but not the sub class members. But if we create sub class object, all the members of both super and sub classes are available to it. This is the reason; we always create an object to sub class in inheritance. Sometimes, the super class members and sub class members may have same names. In that case, by default only sub class members are accessible. This is shown in the following example program.

Example:

```
class One
{
      int i=10;
      void show()
      {
            System.out.println("Super Class Method i: "+i);
      }
}
class Two extends One
{
      int i=20;
      void show()
      {
            System.out.println("Sub Class Method i: "+i);
      }
}
class Demo{
 public static void main( String args[] )
 {
       Two t=new Two();
       t.show();
 }
}
```

Output:

Sub Class Method i: 20

Please observe that sub class method `t.show();` calls and executes only sub class method. And hence the instance variable i value 20 is displayed. To access super class members and methods by using „super" keyword.

- super can be used to refer super class variables, as:
  `super.variableName`
- super can be used to refer super class methods, as:
  `super.methodName()`
- super can be used to refer super class constructor.
  We need not to call default constructor of the super class, as it is by default available to sub class. To call parameterized constructor, we can write:
  `super(values)`

Example: Write a program to access the super class method and instance variable by using super keyword from sub class.

```
class One
{
      int i=10;
      void show()
      {
            System.out.println("Super Class Method i: "+i);
      }
}
class Two extends One
{
      int i=20;
      void show()
      {
            System.out.println("Sub Class Method i: "+i);
            super.show();
            System.out.println("Super Class Variable i: "+super.i);
      }
}
class Demo{
 public static void main( String args[] )
 {
       Two t=new Two();
       t.show();
 }
}
```

Output:

```
Sub Class Method i: 20
Super Class Method i: 10
Super Class Variable i: 10
```

Example: Write a program to access parameterized constructor of the super class can be called from sub class using super keyword.

```
class One
{
      int i;
      One(int i)
      {
            this.i=i;
            this.i=this.i+2;
      }
}
class Two extends One
{
      int i;
      Two(int i)
      {
            super(i);
            this.i=i;
            System.out.println("Sub Class Variable i: "+i);
            System.out.println("Super Class Variable i: "+super.i);
      }
}
class Demo{
 public static void main( String args[] )
 {
       Two t=new Two(5);
 }
}
```

Output:

```
Sub Class Variable i: 5
Super Class Variable i: 7
```

final keyword:

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many contexts. Final can be:

a) variable
b) method
c) class

- The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable.
- It can be initialized in the constructor only.
- The blank final variable can be static also which will be initialized in the static block only.

a) Java `final` variable:

Example:

```
class Bike9{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike9 obj=new  Bike9();
 obj.run();
 }
}
```

Output:

Compile Time Error

b) Java `final` method:

Example:

```
class Bike{
  final void run()
  {
      System.out.println("running");
  }
}
class Honda extends Bike
{
   void run()
   {
     System.out.println("running safely with 100kmph");
   }
   public static void main(String args[])
   {
      Honda honda= new Honda();
      honda.run();
   }
}
```

Output:

Compile time Error.

c) java `final` class

```
final class Bike
{
}
class Honda1 extends Bike
{
  void run()
  {
      System.out.println("running safely with 100kmph");
```

```
    }
    public static void main(String args[]){
    Honda1 honda= new Honda1();
    honda.run();
    }
}
```

Output:

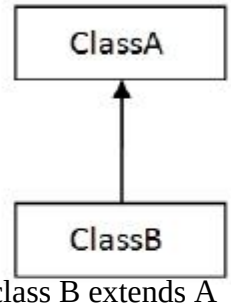
Compile time error

Types of Inheritance:

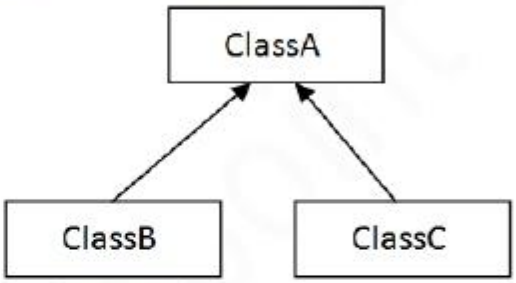
There are three types of inheritance in java.

1. Single Inheritance:

Producing sub classes from one super class is called single inheritance.



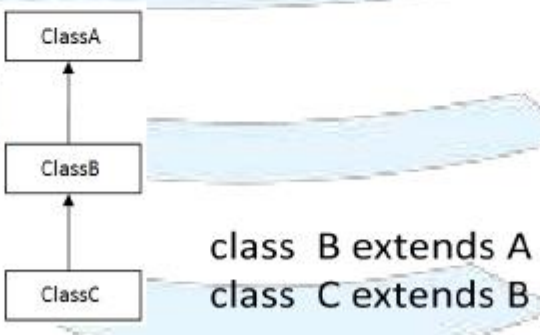
class B extends A



class B extends A
class C extends A

2. Multilevel Inheritance:

When a class extends a class, which extends anther class then this is called multilevel inheritance.



Example: Write a JAVA Program to illustrate multilevel inheritance.

```
class A
{
      void show()
      {
            System.out.println("Method A");
      }
}
```

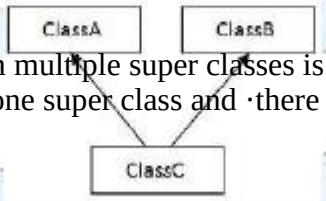
JAVA-3

```
class B extends A
{
      void show()
      {
            super.show();
            System.out.println("Method B");
      }
}
class C extends B
{
      void show()
      {
            super.show();
            System.out.println("Method C");
      }
}
class MultiLevel
{
      public static void main(String[] args)
      {
            C c1=new C();
            c1.show();
      }
}
```

Output:

Method A
Method B
Method C

3. Multiple Inheritance:



Producing sub classes from multiple super classes is called multiple inheritance. In this case, there will be more than one super class and ·there can be one or more sub classes.

- Only single inheritance is available in java. There is no multiple inheritance in java. Multiple inheritance leads to confusion for the programmer. For example, class A has got a member x and class B has also got a member x. When another class C .extends both the classes, then there is a confusion regarding which copy of x is available in C.
- To overcome this problem JavaSoft people provide interface concept, expecting the programmers to achieve multiple inheritance by multiple interfaces.

INTERFACE:

An interface contains only abstract methods which are all incomplete methods. So it is not possible to create an object to an interface. In this case, we can create separate classes where we can implement all the methods of the interface. These classes are called implementation classes. Since, implementation classes will have all the methods with body; it is possible to create objects to the implementation classes. The flexibility lies in the fact that every implementation class can have its own implementation of the abstract methods of the interface.

Syntax:

```
interface MyInter{
        constants
        abstract methods
}
```

Example: Write a Java Program to illustrate multiple inheritance using multiple interfaces.

```
interface Father
{
      double HT=6.2;
      void height();
}
interface Mother
{
      double HT=5.8;
      void color();
}
class Child implements Father, Mother
{
      public void height()
      {
            double ht=(Father.HT+Mother.HT)/2;
            System.out.println("Child's Height= "+ht);
      }
      public void color()
      {
            System.out.println("Child Color= brown");
      }
      public static void main(String[] args)
      {
            Child c=new Child();
            c.height();
            c.color();
      }
}
```

Output:

```
Child's Height= 6.0
Child Color= brown
```

Relationship between classes and Interfaces:

- One class can “implements” one or more interfaces.
- On interface can “extends” one interface.
- One class can “extends” another class.
- One interface cannot “extends” or “implements” another class.

Abstract Method:

An abstract method is a method without method body. An abstract method is written when the same method has to perform different tasks depending on the object calling it.

Abstract Class:

An abstract class is a class that contains 0 or more abstract methods.

- It is not possible to create objects to abstract classes because it is not possible to estimate the total memory required to create the objects.
- So, JVM cannot create objects to an abstract class. We should create sub classes and all the abstract methods should be implemented in the sub classes.
- If any abstract method is not implemented, then that sub class should be declared as „abstract“. In this case, we cannot create an object to the sub class. We should create another sub class to this sub class and implement the remaining abstract method there.

Example:

```
abstract class MyClass
{
	abstract void calculate(double r);
}
class Square extends MyClass
{
	void calculate(double r)
	{
		System.out.println("Square = "+(r*r));
	}
}
class SquareRoot extends MyClass
{
	void calculate(double r)
	{
		System.out.println("Square Root = "+Math.sqrt(r));
	}
}
class Demo2
{
	public static void main(String[] args)
	{
		Square s1=new Square();
		s1.calculate(4);
		SquareRoot s2=new SquareRoot();
		s2.calculate(4);
	}
}
```

Output:

Square = 16.0
Square Root = 2.0

Example: write a JAVA Program for abstract class to find areas of different shapes.

```
abstract class  Shape
{
      abstract  void findCircle(double r);
      abstract  void findTriangle(double b, double h);
      abstract  void findRectangle(double w, double h);

}
class AreaShape extends Shape
{
      void findCircle(double r)
      {
            double a=3.14*r*r;
            System.out.println("Area of Circle = "+a);
      }
      void findTriangle(double b, double h)
      {
            double a=0.5*b*h;
            System.out.println("Area of Triangle = "+a);
      }
      void findRectangle(double w, double h)
      {
            double a=w*h;
            System.out.println("Area of Rectangle = "+a);
      }
      public static void main(String[] args)
      {
            AreaShape as=new AreaShape();
            as.findCircle(4.3);
            as.findTriangle(6.1,4.5);
            as.findRectangle(5.5,7.2);
      }
}
```

Output:

Area of Circle = 58.0586
Area of Triangle = 13.725
Area of Rectangle = 39.6

Difference Between abstract classes and interfaces:

Abstract Clas	Interface
An abstract class is written when there are some common features shared by all the objects.	An interface is written when all the features are implemented differently in different objects.
When an abstract class is written, it is the duty of the programmer to provide subclasses to it.	An interface is written when the programmer wants to leave implementation to third party vendors.
An abstract class contains some abstract methods and also concrete methods.	An interface contains only abstract methods.
An abstract class can contain instance variables also.	An interface can contain only constants doesn‟t contain instance variables.
All the abstract methods should be implemented in its subclasses.	All the interface methods should be implemented classes.
Abstract class can declare by using “abstract” keyword.	Interfaces declared by using “interface” keyword.

Example:

```
abstract class Car
{
	String regdno;
	Car(String r)
	{
		regdno=r;
	}
	void openTank()
	{
		System.out.println("Fill the Tank");
	}
	abstract void steering();
	abstract void breaks();
}
class Maruthi extends Car
{
	Maruthi(String r)
	{
		super(r);
	}
	void steering()
	{
		System.out.println("Normal Steering");
	}
	void breaks()
	{
		System.out.println("Normal breaks");
	}
}
class Hyundai extends Car
{
	Hyundai(String r)
	{

		super(r);
```

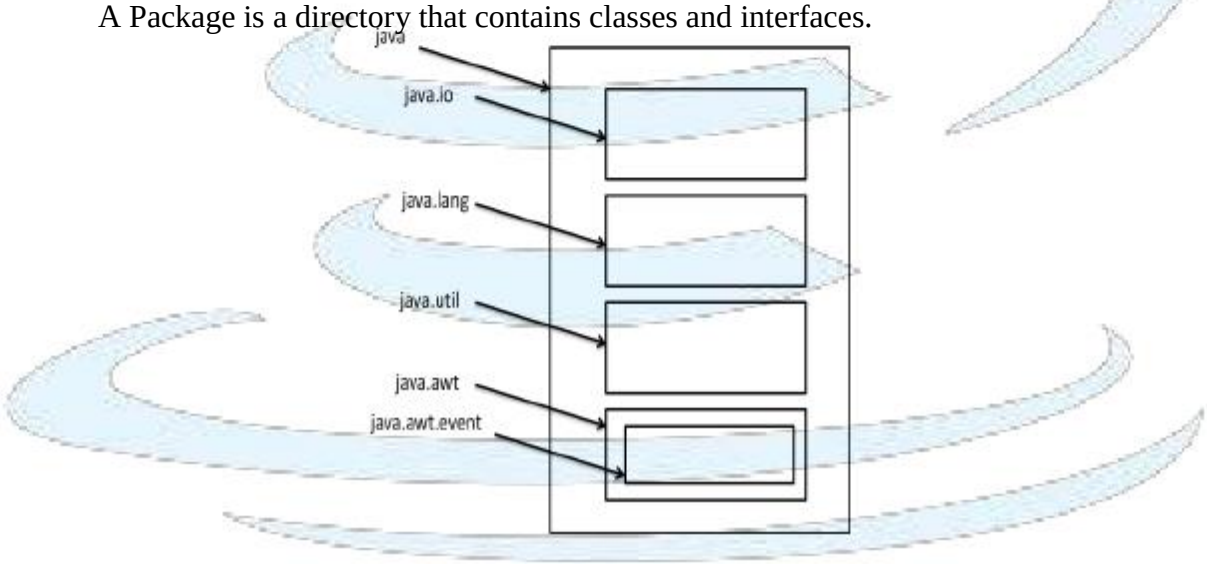
```
	}
	void steering()
	{
		System.out.println("Power Steering");
	}
	void breaks()
	{
		System.out.println("Hydraulic breaks");
	}
}
class DemoCar
{
	public static void main(String[] args)
	{
		Maruthi m=new Maruthi("AP16ED5506");
		m.openTank();
		m.steering();
		m.breaks();
		Hyundai h=new Hyundai("AP16ED3835");
		h.openTank();
		h.steering();
		h.breaks();
	}
}
```

Output:

Fill the Tank
Normal Steering
Normal breaks
Fill the Tank
Power Steering
Hydraulic breaks

Packages:

A Package is a directory that contains classes and interfaces.



Advantages of Packages:

- Packages are useful to arrange group of classes and interfaces into a group. This puts together all the classes and interfaces performing the same task in the same package. For example, java.io package contains all the classes and interfaces performing input and output operations.
- Packages hide the classes and interfaces into a sub directory. So, that accidental deletion of classes and interfaces will not take place.
- The classes and interfaces of a package are isolated from the classes and interfaces in another package. This means that you can use same name of a class of two different packages.
- A group of packages is called a library. The classes and interfaces of a package are like books in a library and can be reused several times.

Types of packages:

There are two types of packages.

1. Built-in packages
2. User-defined packages

1. Built-in packages

- **java.lang:** lang stands for language. This package got primary classes and interfaces essential for developing a basic Java program.
- **Java.util:** util stands for utility. This package contains useful classes and interfaces like Stack, LinkedList, Hashtable, Vector, Arrays, etc. These classes are called collections. There are also classes for handling date and time operations.
- **java.io:** io stands for input and output. This package contains streams. A stream represents flow of data from one place to another place.
- **java.awt:** awt stands for abstract window toolkit. This package helps to develop GUI (Graphics User Interface) where programs with colorful screens, paintings and images etc., can be developed.
- **javax.swing:** This package helps to develop GUI like java.awt. The 'x' in javax represents that it is an extended package which means it is a package developed from another package by adding new features to it. In fact, javax.swing is an extended package of java.awt.
- **java.net:** net stands for network. Client-Server programming can be done by using this package.
- **java.applet:** Applets are programs which come from a server into a client and get executed on the client machine on a network. Applet class of this package is useful to create and use applets.
- **java.text:** This package .has two important Classes, DateFormat to format dates and times, and NumberFormat which is useful to format numeric values.
- **java.sql:** sql stands for structured query language. This package helps to connect to databases like Oracle or Sybase, retrieve the data from them and use it in a Java program.

2. User-defined packages

Just like the Built-in packages shown earlier, the users of the Java language can also create their own packages. They are called user-defined packages. User-defined packages can also be imported into other classes and used exactly in the same way as the Built-in packages. To create a package the keyword "package" is used as:

package packname;
package packname.subpackname;

- Write a program to create package with name pack and create class Addition.

```
package pack;
public class Addition
{
	public void add(int a,int b)
	{
		int c=a+b;
		System.out.println("The sum is "+c);
	}
}
```

- To compile the program by using following command,

```
javac -d . Addition.java
```

- The -d option tells the Java compiler to create a separate sub directory and place the class file there. The dot (.) after -d indicates that the package should be created in the current directory.
- From the above statement our package with "Addition.class" is ready. The next step is we have use the `add()` method in the program. For this purpose, we have to create another class and import that package.
- Write a java program to how to use Addition class of a package pack.

```
import pack.Addition;
class DemoPack
{
	public static void main(String[] args)
	{
		Addition a1=new Addition();
		a1.add(15,27);
	}
}
```

Output: The Sum is 42

Example: Write a program to create package with name pack and create class Calculator and use them another class.

```
package pack;
public class Calculator
{
	public void add(int a,int b)
	{
		int c=a+b;
		System.out.println("The Sum is "+c);
	}
```

```
	public void sub(int a,int b)
	{
		int c=a-b;
		System.out.println("The Subtraction is "+c);
	}
	public void mul(int a,int b)
	{
		int c=a*b;
		System.out.println("The Multiplication is "+c);
	}
	public void div(int a,int b)
	{
		int c=a/b;
		System.out.println("The Division is "+c);
	}
}
```

Output: javac –d . Calculator.java

```
import pack.Calculator;
class DemoPack2
{
	public static void main(String[] args)
	{
		Calculator c=new Calculator();
		c.add(15,27);
		c.sub(35,5);
		c.mul(15,22);
		c.div(45,5);
	}
}
```

Output: javac DemoPack2.java

java DemoPack2

The Sum is 42
The Subtraction is 30
The Multiplication is 330
The Division is 9

If the package pack is not available in the current directory, then what happens? Suppose our program is running in D:\ and the package available in C:\sub. In this case, the compiler should be given information regarding the package location by mentioning the directory of the package in class path.

Setting CLASSPATH:

The CLASSPATH is an environment variable that tells the Java compiler where to look for class files to import. CLASSPATH is generally set to a directory or a JAR (Java Archive) file.

- To see what is there in currently in CLASSPATH variable in your system. You can type the command in windows.

```
echo %CLASSPATH%
```

- Suppose, preceding command has disp1ayed class path as:

```
C:\rnr;.
```

- This means the current class path is set to rnr directory in C: \ and also to the current directory represented by dot (**.**). Our package pack does not exist in either rnr or current directory. Our package exists in D:\sub, as:

```
set CLASSPATH=D:\sub;.;%CLASSPATH%
```

Access Specifiers in Java:

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	YES	NO	NO	NO
Default	YES	YES	NO	NO
Protected	YES	YES	YES	NO
Public	YES	YES	YES	YES

Exception Handling

A software engineer may also commit several errors while designing the project or developing the code. These errors are also called „bugs" and the removing them is called „debugging".

Errors:

There are basically three types of errors in java.

a) Compile-time errors
b) Logical errors
c) Runtime errors

a) Compile-time errors

These are syntactical errors found in the code, due to which a program fails to compile. For example, forgetting a colon in the statements like if, while, for, def, etc. will result in compile-time error. Such errors are detected by java compiler and the line number along with error description is displayed by the java compiler.

b) Logical errors

These errors depict flaws in the logic of the program. The programmer might be using a wrong formula of the design of the program itself is wrong. Logical errors are not detected either by java compiler or JVM. The programme is solely responsible for them.

c) Runtime errors

These errors represent inefficiency of the computer system to execute a particular statement. For example, insufficient memory to store something or inability of the microprocessor to execute some statement come under run-time errors.

Example:

```
class Division
{
	public static void main(String[] args)
	{
		System.out.println("WELCOME");
	}
}
```

Output:

```
Exception in thread "main" java.lang.NoSuchMethodError: main
```

Note: Runtime errors are not detected by the java compiler. They are detected by the JVM, only at runtime.

Exceptions:

- An exception is a runtime error. All exceptions that are occur only by runtime but some exceptions are detected at compile time and some others at runtime.
- The exceptions that are checked at compilation time by the java compiler are called „checked exceptions".
- The exceptions that are checked by the JVM are called „unchecked exceptions".
- Unchecked exceptions and errors are considered as unrecoverable and the programmer cannot do anything when they occur. The programmer can write a java program with unchecked exceptions and errors and can compile the program. He can see their effect only when he runs the program.
- So, java compiler allows him to write a java program without handling the unchecked exceptions and errors.
- Checked exceptions should either handle them or throw them without handling them. He cannot simply ignore them, as java compiler will remind him of them.

Example:

```
import java.io.*;
import java.lang.*;
class Test {
    public static void main(String[] args)
    {
		InputStreamReader isr = new InputStreamReader(System.in);
		BufferedReader br = new BufferedReader(isr);
		System.out.print("Enter a single Character: ");
		char ch = (char)br.read();
		System.out.print("\n The Character is "+ch);
    }
}
```

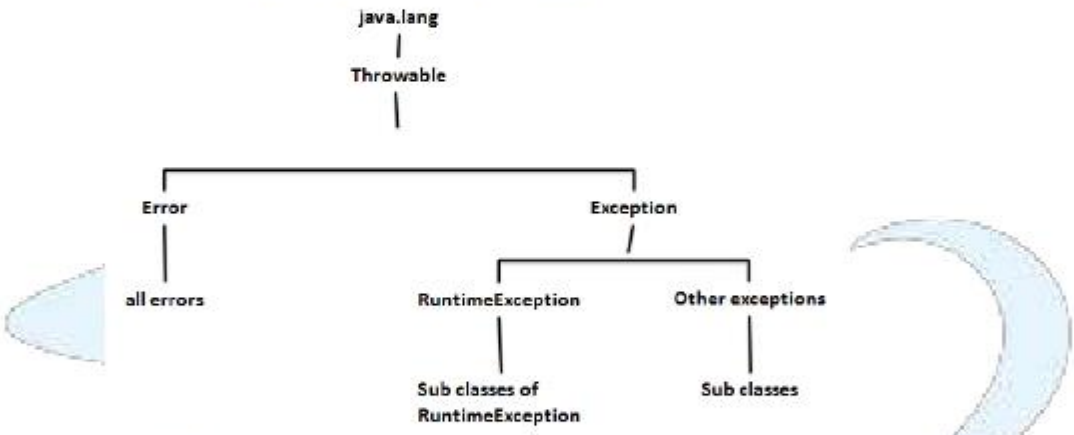
Output:

```
javac Test.java
Test.java:9: unreported exception java.io.IOException; must be caught or declared to
be thrown
        char ch = (char)br.read();
                          ^
1 error
```

- Here, at compilation-time Java Compiler shows an IOException i.e., Checked Exception. It must be caught or declared to be thrown. So, we throw it out of **`main()`** method without handling it. This is done by throws clause written after **`main()`** method in the above program, as:

```
public static void main(string[] args)throws IOException
```

- All Exceptions are declared as classes in java. Even the errors are also represented by classes. All these classes are descended from a super class called **`Throwable`**. And this class belongs to **`java.lang`** package.
- An exception is an error which can be handled. It means when an exception happens, the programmer can do something to avoid any harm. But an error is an error which cannot be handled, it happens and the programmer cannot do anything.



Exception Handling:

When there is an exception, the user data may be corrupted. This should be tackled by the programmer by carefully designing the program. For this, he should perform the following 3 steps:

Step 1: The programmer should observer the statements in his program where there may be a possibility of exceptions. Such statements should be written inside a **`try`** block. A **`try`** block looks like as follows:

```
try{
        statements;
}
```

The greatness of try block is that even if some exception arises inside it, the program will not be terminated. When JVM understands that there is an exception, it stores the exception details in an exception stack and then jumps into a catch block.

Step 2: The programmer should write the Catch block where he should display the exception details to the user. This helps the user to understand that there is some error in the program. The programmer should also display a message regarding what can be done to avoid this error. `catch` block looks like as follows:

```
catch(ExceptionClass ref)
{
      Statements;
}
```

Step 3: Lastly, the programmer should perform clean up operations like closing the files a terminating the threads. The programmer should write this code in the `finally` block. final block looks like as follows:

```
finally{
      Statements;
}
```

Example: Write a program by using try, catch and finally block.

```
class Division
{
      public static void main(String[] args)
      {
            try{
                  System.out.println("WELCOME");
                  int a=5;
                  int b=0;
                  int c=a/b;
                  System.out.println("The Division is "+c);
            }
            catch(ArithmeticException ae)
            {
              System.out.println("Division with zero is not
              possible");
            }
            finally{
                  System.out.println("LOGOUT");
            }
      }
}
```

Output:

```
WELCOME
Division with zero is not possible
LOGOUT
```

Handing Multiple Exceptions:

Most of the times there is possibility of more than one exception present in the program. In this case, the programmer should write multiple catch blocks to handle each one of them.

Example: write a java program to handle multiple exceptions.

```
import java.util.*;
class Division
{
	public static void main(String[] args)
	{
		try{
			System.out.println("WELCOME");
			Scanner sc=new Scanner(System.in);
			System.out.print("Enter a value: ");
			int a=sc.nextInt();
			System.out.print("Enter b value: ");
			int b=sc.nextInt();
			int c=a/b;
			System.out.println("The Division is "+c);
		}
		catch(InputMismatchException ae)
		{
			System.out.println("Wrong Input");
		}
		catch(ArithmeticException ae)
		{
		  System.out.println("Division with zero is not
		  possible");
		}
		finally{
			System.out.println("LOGOUT");
		}
	}
}
```

Output-1:

```
WELCOME
Enter a value: 5
Enter b value: k
Wrong Input
LOGOUT
```

Output-2:

```
WELCOME
Enter a value: 5
Enter b value: k
Division with zero is not possible
LOGOUT
```

throws clause:

Even if the programme is not handling runtime exceptions, the Java compiler will not give any error related to runtime exceptions. But the rule is that the programmer should handle checked exceptions. In case the programmer does not want to handle the checked exceptions, he should throw them out using throws clause. Otherwise, there will be an error flagged by Java compiler.

Example: Write a program that shows the use of `throws` clause.

```
import java.io.*;
import java.lang.*;
class Test {
    public static void main(String[] args)throws IOException
    {
          InputStreamReader isr = new InputStreamReader(System.in);
          BufferedReader br = new BufferedReader(isr);
          System.out.print("Enter the String: ");
          String str = br.readLine();
          System.out.print("\n The String is "+str);
    }
}
```

throw clause:

There is also a throw statement available in Java to throw an exception explicitly and catch it. Let us see how it can be used. In the following program, we are creating an object of NullPointerException class and throwing it out of try block, as shown here:

Example: Write a program that shows the use of `throw` clause.

```
import java.util.*;
class ThrowException
{
      public static void main(String[] args)
      {
            try{
                  System.out.println("WELCOME");
                  throw new NullPointerException("Exception Data");
            }
            catch(NullPointerException ne)
            {
                  System.out.println(ne);
            }
      }
}
```

Output:

```
WELCOME
java.lang.NullPointerException: Exception Data
```

Types of Exceptions:

There are two types of Exceptions.

a) Built-in Exceptions

b) User-defined Exceptions

a) Built-in Exceptions

Exception	Description
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
ClassNotFoundException	Class not found.
InterruptedException	One thread has been interrupted by another thread.
NoSuchMethodException	A requested method does not exist.

b) User-defined Exceptions

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, like the built-in exceptions; the user also creates own exceptions which are called 'user-defined exceptions'. The following steps are followed in creation of user-defined exceptions.

- The user should create an exception class as a subclass to Exception class. Since all exceptions are subclasses of Exception class, the user should also make his class a subclass to it. This is done as:

```
class MyException extends Exception
```

- The user can write a default constructor in his own exception class. He can use it, in case he does not want to store any exception details.

```
MyException() { }
```

- The user can create a parameterized constructor with a string as a parameter. He can use to store exception details. He can call super class (Exception) constructor from this and send the string there.

```
MyException(String str)
{
      super(str);
}
```

- When the user wants to raise his own exception, he should create an object to his exception class and throw it using throw clause, as:

```
MyException me=new MyException("Exception Details");
throw me;
```

Example: Write a program to define user-defined Exception.

```
import java.util.Scanner;
class MyException extends Exception
{
      MyException(String str)
      {
            super(str);
      }
}
class ExceptionDemo
{
      public static void main(String[] args)
      {
            try{
            System.out.println("WELCOME");
            System.out.print("Enter amount to withdraw: ");
            Scanner sc=new Scanner(System.in);
            double bal=sc.nextDouble();
            if(bal>25000)
            {
              MyException me=new MyException("Balance is very
              high");
              throw me;
            }
            System.out.println("Balance is withdarwn");
            }
            catch(MyException me)
            {
                  System.out.println(me);
            }
      }
}
```

Output-1:

```
WELCOME
Enter amount to withdraw: 26000
MyException: Balance is very high
```

Output-2:

```
WELCOME
Enter amount to withdraw: 15000
Balance is withdrawn
```

Assertion:

- Assertion is a statement in java. It can be used to test your assumptions about the program.
- While executing assertion, it is believed to be true. If it fails, JVM will throw an error named AssertionError. It is mainly used for testing purpose.

Advantage of Assertion:

- It provides an effective way to detect and correct programming errors.

Syntax:

assert expression;

or

assert expression1:expression2;

Example:

```
import java.util.Scanner;
class AssertionExample
{
 public static void main( String args[] )
 {
  Scanner scanner = new Scanner( System.in );
  System.out.print("Enter ur age ");
  int value = scanner.nextInt();
  assert value>=18:" Not valid";
  System.out.println("value is "+value);
 }
}
```

If you use assertion, it will not run simply because assertion is disabled by default. To enable the assertion, -ea or -enableassertions switch of java must be used.

Compile it by: javac AssertionExample.java

Run it by: java -ea AssertionExample

Output: Enter ur age 11

Exception in thread "main" java.lang.AssertionError: Not valid