UNIT – III

- **INHERITANCE**
  - **METHOD OVERLAODING**
  - **SUPER KEYWORD**
  - **FINAL KEYWORD**
  - **ABSTRACT CLASS**
- **INTERFACES**
- **PACKAGES**
  - **CREATING PACKAGES**
  - **USING PACKAGES**
  - **ACCESS PROTECTION**
  - **JAVA.LANG.PACKAGE**
- **EXCEPTIONS**
  - **EXCEPTION HANDLING TECHNIQUES**
  - **TRY, CATCH, FINALLY, THROW, THROWS**
  - **USER DEFINED EXCEPTION**
  - **EXCEPTION ENRICHMENT**
- **ASSERTIONS**

INHERITANCE IN JAVA

Inheritance in java is a mechanism in which one object acquires all the properties and behaviours of parent object.

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.

Inheritance represents the **IS-A relationship**, also known as ***parent-child*** relationship.

Syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class.

In the terminology of Java, a class that is inherited is called a super class. The new class is called a subclass.

Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through **interface** only.

When a class extends multiple classes i.e. known as multiple inheritance.

Q) Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Consider a scenario where A, B and C are three classes. The C class inherits A and B classes. If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.

Since compile time errors are better than runtime errors, java renders compile time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error now.

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B{//suppose if it were

 Public Static void main(String args[]){
   C obj=new C();
   obj.msg();//Now which msg() method would be invoked?
}
}
```

Test it Now

```
Compile Time Error
```

Method Overloading in Java

If a class have multiple methods by same name but different parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Advantage of method overloading?

Method overloading **increases the readability of the program**.

Different ways to overload the method:

There are two ways to overload the method in java:

1. By changing number of arguments
2. By changing the data type

In java, Method Overloading is not possible by changing the return type of the method.

SUPER KEYWORD IN JAVA

The **super** keyword in java is a reference variable that is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.

Usage of java super Keyword

1. super is used to refer immediate parent class instance variable.
2. super() is used to invoke immediate parent class constructor.
3. super is used to invoke immediate parent class method.

FINAL KEYWORD IN JAVA

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

1. variable
2. method
3. 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.

1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

2) Java final method

If you make any method as final, you cannot override it.

3) Java final class

If you make any class as final, you cannot extend it.

ABSTRACT CLASS IN JAVA

A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).

A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.

Example abstract class

```
abstract class A {}
```

Abstract method

A method that is declared as abstract and does not have implementation is known as abstract method.

Example abstract method

```
abstract void printStatus();//no body and abstract
```

Example of abstract class that has abstract method:

In this example, Bike the abstract class that contains only one abstract method run. It implementation is provided by the Honda class.

```
abstract class Bike
{
  abstract void run();
}
class Honda4 extends Bike{
void run()
{
System.out.println("running safely..");
}
public static void main(String args[])
{
 Bike obj = new Honda4();
 obj.run();
}
}
```

Test it Now

```
running safely..
```

Another example of abstract class in java:

File: TestBank.java

```
 abstract class Bank
{
abstract int getRateOfInterest();
}
class SBI extends Bank
{
int getRateOfInterest(){return 7;}
}
class PNB extends Bank{
int getRateOfInterest(){return 7;}
}

class TestBank{
public static void main(String args[]){
Bank b=new SBI();//if object is PNB, method of PNB will be invoked
int interest=b.getRateOfInterest();
System.out.println("Rate of Interest is: "+interest+" %");
}
}
```

Test it Now

```
Rate of Interest is: 7 %
```

INTERFACE IN JAVA

An **interface in java** is a blueprint of a class. It has static constants and abstract methods only.

The interface in java is **a mechanism to achieve fully abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritances in Java.

Java Interface also **represents IS-A relationship**.

It cannot be instantiated just like abstract class.

Why use Java interface?

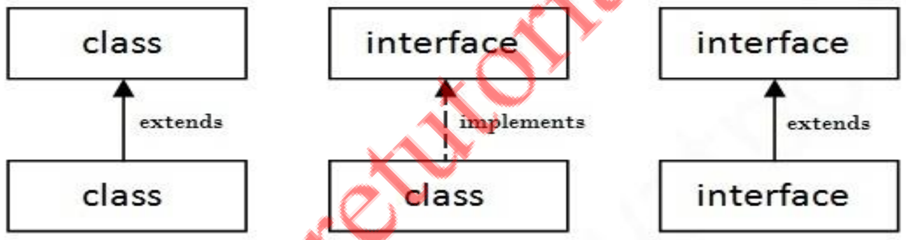
There are mainly three reasons to use interface. They are given below.

- It is used to achieve fully abstraction.
- By interface, we can support the functionality of multiple inheritance.

NOTE: Interface fields are public, static and final by default, and methods are public and abstract.

Understanding relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface but a **class implements an interface**.



Simple example of Java interface

In this example, Printable interface have only one method, its implementation is provided in the A class.

```
interface printable  {
void print();
}
class A6 implements printable  {
public void print()
{
```

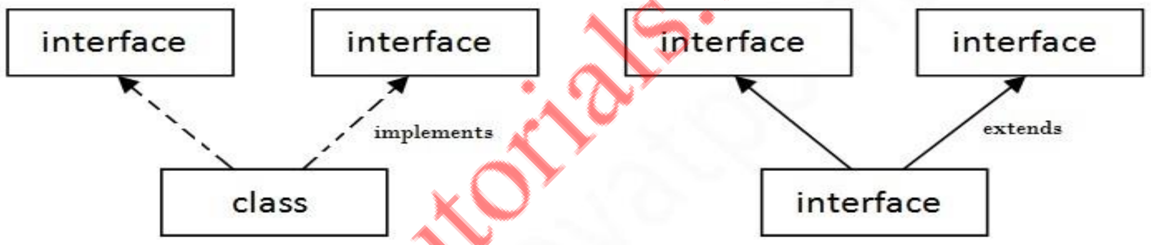
```
System.out.println("Hello");
}
public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

Test it Now

```
Output:Hello
```

Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.



Multiple Inheritance in Java

```
interface Printable{
void print();
}

interface Showable{
void show();
}

class A7 implements Printable,Showable{

public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
A7 obj = new A7();
obj.print();
```

```
        obj.show();
         }
        }
```

Test it Now

```
Output:  Hello
         Welcome
```

Q) What is marker or tagged interface?

An interface that have no member is known as marker or tagged interface. For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

```
//How Serializable interface is written?
public interface Serializable
{

}
```

Nested Interface in Java

An interface can have another interface i.e. known as nested interface. We will learn it in detail in the nested classes chapter.

For example:

```
interface printable{
 void print();
 interface MessagePrintable
  {
   void msg();
  }
}
```

JAVA PACKAGE

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.

Simple example of java package

The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package");
   }
}
```

How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

```
javac -d directory javafilename
```

For **example**

```
javac -d . Simple.java
```

The -d switch specifies the destination where to put the generated class file.

How to run java package program

You need to use fully qualified name e.g. mypack.Simple etc to run the class.

To Compile: javac -d . Simple.java

To Run: java mypack.Simple

```
Output:Welcome to package
```

The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The . Represents the current folder.

How to access package from another package?

There are three ways to access the package from outside the package.

1. import package.*;
2. import package.classname;
3. fully qualified name.

1) Using packagename.*

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

*Example of package that import the packagename.**

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}

//save by B.java
 package mypack;
import pack.*;
class B{
  public static void main(String args[]){
       A obj = new A();
       obj.msg();
      }
     }
```

```
Output: Hello
```

2) Using packagename.classname

If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
//save by A.java

package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java

package mypack;
import pack.A;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

```
Output: Hello
```

3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

```
//save by A.java

package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java

package mypack;
class B{
  public static void main(String args[]){
   pack.A obj = new pack.A();//using fully qualified name
   obj.msg();
  }
}
```

```
Output:Hello
```

EXCEPTION HANDLING IN JAVA

The **exception handling in java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

What is exception

Dictionary Meaning: Exception is an abnormal condition.

In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

What is exception handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

Advantage of Exception Handling

The core advantage of exception handling is to maintain the normal flow of the application**. Exception normally disrupts the normal flow of the application that is why we use exception handling.**

Let's take a scenario:

```
1.      statement 1;
2.      statement 2;
3.      statement 3;
4.      statement 4;
5.      statement 5;//exception occurs
6.      statement 6;
7.      statement 7;
8.      statement 8;
9.      statement 9;
10.     statement 10;
```

Suppose there is 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the exception will be executed. That is why we use exception handling in java.

Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

Difference between checked and unchecked exceptions

1) Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Common scenarios where exceptions may occur

There are given some scenarios where unchecked exceptions can occur. They are as follows:

1) Scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.

```
int a=50/0;//ArithmeticException
```

2) Scenario where NullPointerException occurs

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

```
String s=null;
System.out.println(s.length());//NullPointerException
```

3) Scenario where NumberFormatException occurs

The wrong formatting of any value, may occur NumberFormatException. Suppose I have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

```
String s="abc";
int i=Integer.parseInt(s);//NumberFormatException
```

4) Scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

```
int a[]=new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException
```

Java Exception Handling Keywords

There are 5 keywords used in java exception handling.

1. try
2. catch
3. finally
4. throw
5. throws

Java try block

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

Syntax of java try-catch

```
try{
//code that may throw exception
}
catch(Exception_class_Name ref){}
```

Syntax of try-finally block

```
try{
//code that may throw exception
}
finally{}
```

Java catch block

Java catch block is used to handle the Exception. It must be used after the try block only.

You can use multiple catch block with a single try.

EXAMPLE:

```
public class Testtrycatch2{
  public static void main(String args[]){
   try{
      int data=50/0;
   }catch(ArithmeticException e){System.out.println(e);}
   System.out.println("rest of the code...");
}
}
```

Test it Now

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Now, as displayed in the above example, rest of the code is executed i.e. rest of the code... statement is printed.

Java finally block

Java finally block is a block that is used *to execute important code* such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java finally block must be followed by try or catch block.

EXAMPLE:

```
public class TestFinallyBlock2{
  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("rest of the code...");
  }
}
```

Test it Now

```
Output:Exception in thread main java.lang.ArithmeticException:/ by zero
       finally block is always executed
       rest of the code...
```

Java throw keyword

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception.

The syntax of java throw keyword is given below.

```
throw exception;
```

java throw keyword example

In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

```
public class TestThrow1{
   static void validate(int age){
     if(age<18)
      throw new ArithmeticException("not valid");
     else
      System.out.println("welcome to vote");
   }
   public static void main(String args[]){
      validate(13);
      System.out.println("rest of the code...");
  }
}
```

Test it Now

Output:

```
Exception in thread main java.lang.ArithmeticException:not valid
```

Java throws keyword

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Syntax of java throws

```
return_type method_name() throws exception_class_name{
 ...
}
```

Java throws example

Let's see the example of java throws clause which describes that checked exceptions can be propagated by throws keyword.

```
import java.io.IOException;
class Testthrows1{
  void m()throws IOException{
    throw new IOException("device error");//checked exception
  }
  void n()throws IOException{
    m();
  }
  void p(){
   try{
    n();
   }catch(Exception e){System.out.println("exception handled");}
  }
  public static void main(String args[]){
   Testthrows1 obj=new Testthrows1();
   obj.p();
   System.out.println("normal flow...");
  }
}
```

Test it Now

Output:

```
exception handled
normal flow...
```

EXCEPTION ENRICHMENT IN JAVA

Exception enrichment is an alternative to **exception wrapping**

In exception enrichment you do not wrap exceptions. Instead you add contextual information to the original exception and rethrow it. Rethrowing an exception does not reset the stack trace embedded in the exception.

Here is an example:

```
public void method2() throws EnrichableException{

  try{

    method1();

  } catch(EnrichableException e){

    e.addInfo("An error occurred when trying to ...");

    throw e;

  }

}

public void method1() throws EnrichableException {

  if(...) throw new EnrichableException(
```

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 of using Assertion:

There are two ways to use assertion. First way is:

```
assert expression;
```

and second way is:

```
assert expression1 : expression2;
```

Simple Example of Assertion in java:

```
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);
 }
}
```