UNIT-3

1) Explain Different Types of Inheritance?

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

The idea behind inheritance is that we can create new classes that are built upon existing classes. When we inherit from an existing class, we reuse methods and fields, and add new methods and fields to adapt our new class to new situations.

Inheritance represents the **IS-A relationship**.

Syntax of Inheritance

```
class Subclass extends Superclass

{

      //methods and fields
```

The keyword extends indicates that we are making a new class that derives from an existing class. In the terminology of Java, a class that is inherited is called a superclass. The new class is called a subclass.

Example of inheritance

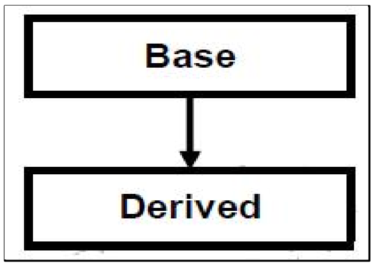
```
Class A
{
   public void methodA()
  {
    System.out.println("Base class method");
  }
}
Class B extends A
{
  public void methodB()
  {
    System.out.println("Child class method");
  }
  public static void main(String args[])
  {
    B obj = new B();
    obj.methodA();
    obj.methodB();
  }
}
```

Types of Inheritance:

i) Single Inheritance
ii) Hierarchical Inheritance
iii) Multi Level Inheritance
iv) Hybrid Inheritance
v) Multiple Inheritance

i. Single Inheritance

when a single derived class is created from a single base class then the inheritance is called as single inheritance.



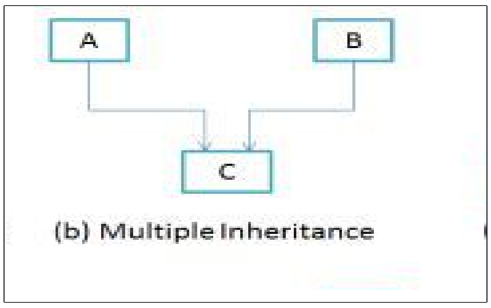
(a) Single Inheritance

Example :

```
Class A
{
  public void methodA()
  {
    System.out.println("Base class method");
  }
}
Class B extends A
{
  public void methodB()
  {
    System.out.println("Child class method");
  }
  public static void main(String args[])
  {
    B obj = new B();
    obj.methodA();
    obj.methodB();
  }
}
```

ii) Multiple Inheritance

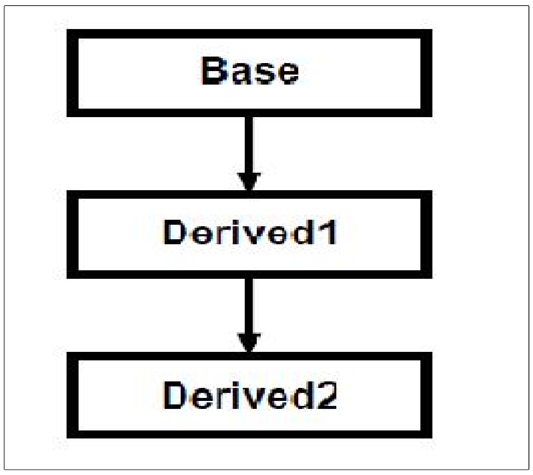
“**Multiple Inheritance**” refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with “multiple inheritance” is that the derived class will have to manage the dependency on two base classes.



Note 1: Multiple Inheritance is very rarely used in software projects. Using Multiple inheritance often leads to problems in the hierarchy. This results in unwanted complexity when further extending the class.

iii. Multi Level Inheritance

when a derived class is created from another derived class, then that inheritance is called as multi level inheritance.



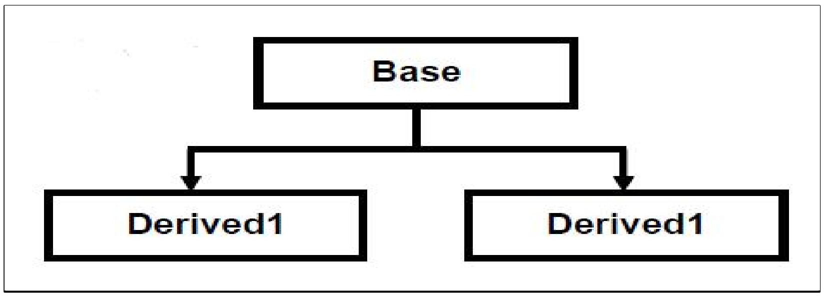
(c) Multilevel Inheritance

Example :

```
        Class X
        {
          public void methodX()
          {
            System.out.println("Class X method");
          }
        }
        Class Y extends X
        {
           public void methodY()
          {
             System.out.println("class Y method");
          }
       }
     Class Z extends Y
     {
        public void methodZ()
        {
          System.out.println("class Z method");
        }
        public static void main(String args[])
        {
            Z obj = new Z();
            obj.methodX();
            obj.methodY();
            obj.methodZ();
       }
     }
```

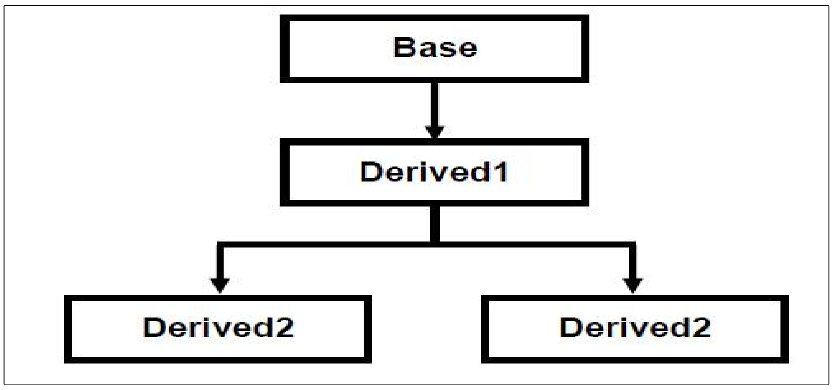
iv. Hierarchical Inheritance

when more than one derived classes are created from a single base class, then that inheritance is called as hierarchical inheritance.



(d) Hierarchical Inheritance

v. Hybrid Inheritance
Any combination of single, hierarchical and multi level inheritances is called as hybrid inheritance.

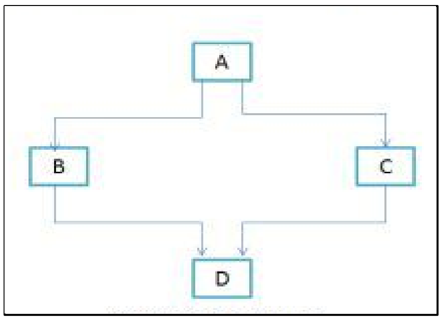


(e) Hybrid Inheritance

2) Does Java support multiple Inheritance? Justify your answer.

Multiple inheritance is not supported in case of class because of ambiguity. But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class.

Consider the below diagram which shows multiple inheritance as Class D extends both Class B & C. Now let's assume we have a method in class A and class B & C overrides that method in their own way. Wait!! here the problem comes - Because D is extending both B & C so if D wants to use the same method which method would be called (the overridden method of B or the overridden method of C). Ambiguity. That's the main reason why Java doesn't support multiple inheritance.



We can achieve multiple inheritance in Java by using interfaces.

Example:

```
interface X
{
  public void Display();
}
interface Y
{
  public void Display();
}
class Demo implements X, Y
{
  public void Display()
  {
     System.out.println(" Multiple inheritance example using interfaces");
  }
}
```

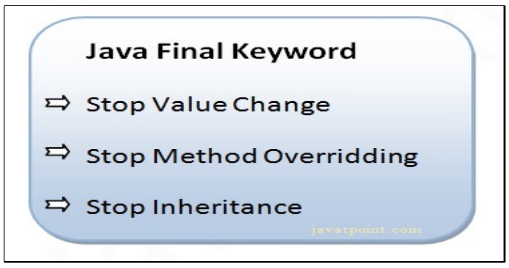
3) With suitable code segments illustrate various uses of 'final' keyword?

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

i. variable

ii. method
iii. 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. We will have detailed learning of these. Let's first learn the basics of final keyword.



i) final variable

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

Example of final variable

There is a final variable speed limit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Bike
{
 final int speed=90;
  void run()
   {
     speed=400;
    System.out.println(speed);
    }
 public static void main(String args[])
{
 Bike obj=new  Bike();
 obj.run();
  }
}

Output:Compile Time Error
```

ii) final method

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

Example of final method

```
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

iii) final class

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

Example of final class

```
final class Bike
{
}
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

4) Describe about 'super' keyword in java.

super keyword:

The **super** 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 super Keyword

i. super is used to refer immediate parent class instance variable.
ii. super() is used to invoke immediate parent class constructor.
iii. super is used to invoke immediate parent class method.

i) Super is used to refer immediate parent class instance variable.

```
class Vehicle
{
   int speed=50;
}
class Bike extends Vehicle
{
      int speed=100;
       void display()
     {
         System.out.println (super.speed);
     }
    public static void main(String args[])
    {
        Bike b=new Bike();
        b.display();
    }
}
```

Output:50

ii) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor as given below:

```
class Vehicle
{
  Vehicle(){System.out.println("Vehicle is created");}
}
class Bike extends Vehicle
{
   Bike()
   {
     super();
```

```
              System.out.println("Bike is created");
             }
            public static void main(String args[])
            {
               Bike b=new Bike();
             }
         }
```

```
Output:  Vehicle is created
         Bike is created
```

iii) super can be used to invoke parent class method.

The super keyword can also be used to invoke parent class method. It should be used in case subclass contains the same method as parent class as in the example given below:

```
class Person
{
     void message()
     {
       System.out.println("welcome");
      }
}
class Student extends Person
{
      void message()
     {
        System.out.println("welcome to java");
      }
       void display()
     {
        message();
        super.message();
     }
public static void main(String args[])
{
   Student s=new Student();
   s.display();
  }
}
```

```
Output: welcome to java
     welcome
```

In the above example Student and Person both classes have message () method if we call message() method from Student class, it will call the message() method of Student class not of Person class because priority is given to local.

5) Explain about Abstract class in JAVA

A class that is declared with abstract keyword is known as abstract class. **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only important things to the user and hides the internal details. For example sending sms, we just type the text and send the message. We don't know the internal processing about the message delivery.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

i. Abstract class (0 to 100%)
ii. Interface (100%)

Abstract class

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**.

Syntax to declare the abstract class

```
abstract class <class_name>
{

}
```

abstract method

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

Syntax to define the abstract method

```
abstract return_type <method_name>();//no braces{}
```

Example of abstract class that have abstract method

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

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

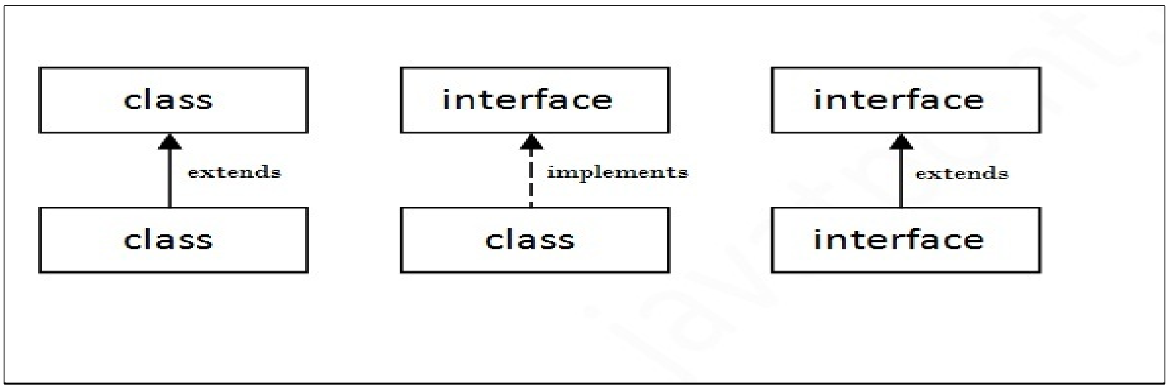
```
Output:running safely..
```

6) Explain about Interface concept in JAVA.

An **interface in java** is a blueprint of a class. It has static constants and abstract methods. The interface in java is **a mechanism to achieve abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.

Java Interface also **represents IS-A relationship**. It cannot be instantiated just like abstract class.

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.



Example:

```
 interface X
{
  public void Display();
}
interface Y
{
  public void Display();
}
class Demo implements X, Y
{
  public void Display()
  {
     System.out.println(" Multiple inheritance example using interfaces");
  }
   Public static void main(String args[])
    {
       Demo d=new Demo();
       d.Display();
    }
}
```

7) Explain Method Overriding concept in JAVA?

Method Overriding

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding**.

Advantage of Java Method Overriding

- Method Overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method Overriding is used for Runtime Polymorphism

Rules for Method Overriding

i) method must have same name as in the parent class
ii) method must have same parameter as in the parent class.
iii) must be IS-A relationship (inheritance).

Example of method overriding

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method is same and there is IS-A relationship between the classes, so there is method overriding.

```
class Vehicle
```

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

8) What are the various types of exceptions available in Java? Also discuss on how they are handled?

Exception

- 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.

Exception Handling in Java

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

In this page, we will know about exception, its type and the difference between checked and unchecked exceptions.

Exception Handling

Exception Handling is a mechanism to handle runtime errors.

Advantage of Exception Handling

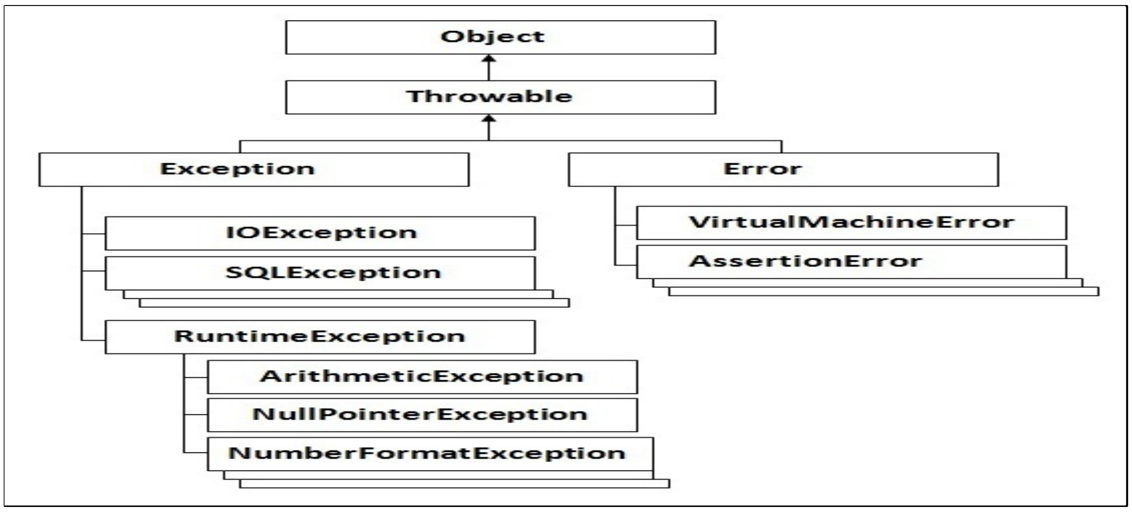
The core advantage of exception handling is that normal flow of the application is maintained. Exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

```
statement 1;
statement 2;
statement 3;
```

statement 4;
statement 5;
statement 6;
statement 7;
statement 8;
statement 9;
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.

Hierarchy of Exception classes:



Types of Exception:

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

i) Checked Exception
ii) Unchecked Exception
iii) Error

i) 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 by Compiler.

ii) 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.

iii) Error

Error is irrecoverable

e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Five keywords used in Exception handling:

i) try
ii) catch
iii) throws
iv) throw
v) finally

i) try block

Enclose the code that might throw an exception in try block. It must be used within the method and must be followed by either catch or finally block.

Syntax of try with catch block

```
try
{
   .......
}
 catch(Exception_class_Name reference)
 {
 }
```

Syntax of try with finally block

```
try
{
   ...........
}
finally
{
}
```

ii) catch block

Catch block is used to handle the Exception. It must be used after the try block.

```
class Simple
{
  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...");
 }
}
```

iii) Throws: It can be used to identify the exception but it can't handle the exception.

```
class Simple
{
  public static void main(String args[]) throws ArithmeticException
 {
      int data=10/0;
   }
System.out.println("rest of the code...");
 }
}
```

9) Give the differences between throw and throws.

Difference between throw and throws:

throw keyword	throws keyword
1.throw is used to explicitly throw an exception.	1. throws is used to declare an exception.
2. checked exception cannot be propagated without throws.	2. checked exception can be propagated with throws.
3. throw is followed by an instance.	3. throws is followed by class.

4. throw is used within the method.	4. throws is used with the method signature.
5.we cannot throw multiple exception	5. we can declare multiple exception e.g. public void method()throws IOException,SQLException.

10) Explain about packages in java.

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.

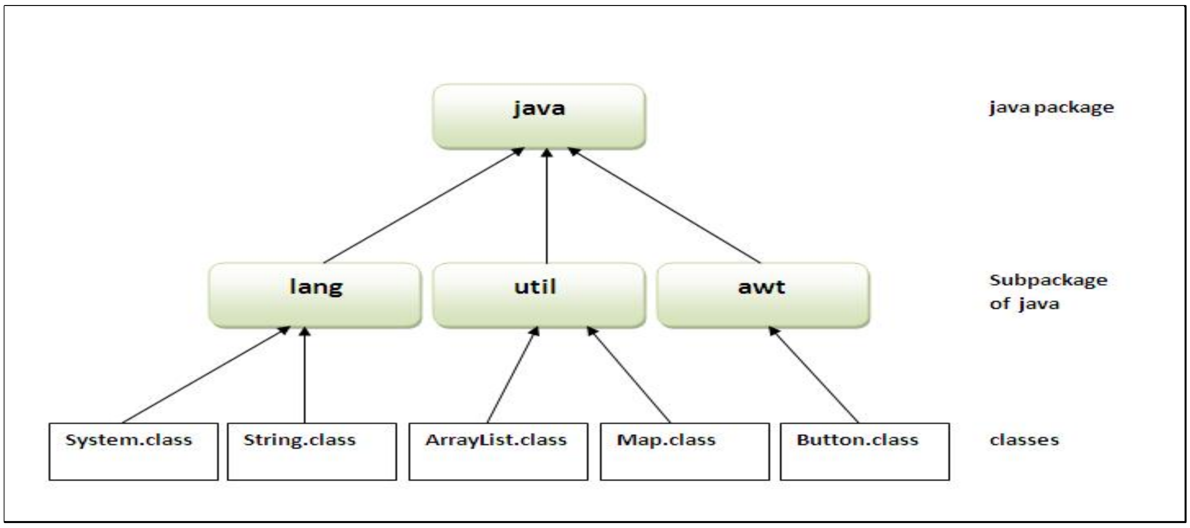
Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package:

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

ii) Java package provides access protection.

iii) Java package removes naming collision.



Example:

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

How to compile java package

```
javac -d directory javafilename
```

Example

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

11) How to access package from another package?

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

i) import package.*;
ii) import package.classname;
iii) fully qualified name.

i) Using packagename.*

If we use package.* then all the classes and interfaces of this package will be accessible but not sub packages.

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

```
//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();
      }
    }
```

ii) Using packagename.classname

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

```
//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();
       }
     }
```

iii) Using fully qualified name

If we use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But we 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.

```
//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();
      obj.msg();
   }
}
```

12) Explain java.lang package in detail.

Java.lang package in Java

Provides classes that are fundamental to the design of the Java programming language. The most important classes are Object, which is the root of the class hierarchy, and Class, instances of which represent classes at run time.

Following are the Important Classes in Java.lang package:

- Boolean: The Boolean class wraps a value of the primitive type boolean in an object.
- Byte: The Byte class wraps a value of primitive type byte in an object.
- Character: The Character class wraps a value of the primitive type char in an object.
- Character.Subset: Instances of this class represent particular subsets of the Unicode character set.
- Character.UnicodeBlock: A family of character subsets representing the character blocks in the Unicode specification.
- Class –: Instances of the class Class represent classes and interfaces in a running Java application.
- ClassLoader: A class loader is an object that is responsible for loading classes.
- ClassValue: Lazily associate a computed value with (potentially) every type.
- Compiler: The Compiler class is provided to support Java-to-native-code compilers and related services.
- Double: The Double class wraps a value of the primitive type double in an object.
- Enum: This is the common base class of all Java language enumeration types.
- Float: The Float class wraps a value of primitive type float in an object.
- InheritableThreadLocal: This class extends ThreadLocal to provide inheritance of values from parent thread to child thread: when a child thread is created, the child receives initial values for all inheritable thread-local variables for which the parent has values.
- Integer :The Integer class wraps a value of the primitive type int in an object.

- Long: The Long class wraps a value of the primitive type long in an object.
- Math: The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.
- Number: The abstract class Number is the superclass of classes BigDecimal, BigInteger, Byte, Double, Float, Integer, Long, and Short.
- Object: Class Object is the root of the class hierarchy.
- Package: Package objects contain version information about the implementation and specification of a Java package.
- Process: The ProcessBuilder.start() and Runtime.exec methods create a native process and return an instance of a subclass of Process that can be used to control the process and obtain information about it.
- ProcessBuilder: This class is used to create operating system processes.
- ProcessBuilder.Redirect: Represents a source of subprocess input or a destination of subprocess output.
- Runtime: Every Java application has a single instance of class Runtime that allows the application to interface with the environment in which the application is running.
- RuntimePermission: This class is for runtime permissions.
- SecurityManager: The security manager is a class that allows applications to implement a security policy.
- Short: The Short class wraps a value of primitive type short in an object.
- StackTraceElement: An element in a stack trace, as returned by Throwable.getStackTrace().
- StrictMath: The class StrictMath contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.
- String: The String class represents character strings.
- StringBuffer: A thread-safe, mutable sequence of characters.
- StringBuilder: A mutable sequence of characters.
- System: The System class contains several useful class fields and methods.
- Thread: A thread is a thread of execution in a program.
- ThreadGroup: A thread group represents a set of threads.
- ThreadLocal: This class provides thread-local variables.
- Throwable: The Throwable class is the superclass of all errors and exceptions in the Java language.
- Void: The Void class is an uninstantiable placeholder class to hold a reference to the Class object representing the Java keyword void.