UNIT-IV

Inheritance, Packages & Interfaces

Abstract

- Introduction of Package
- Create package
- Import keyword
- Accessing rules for package
- Interface
- Abstract class
- Final class

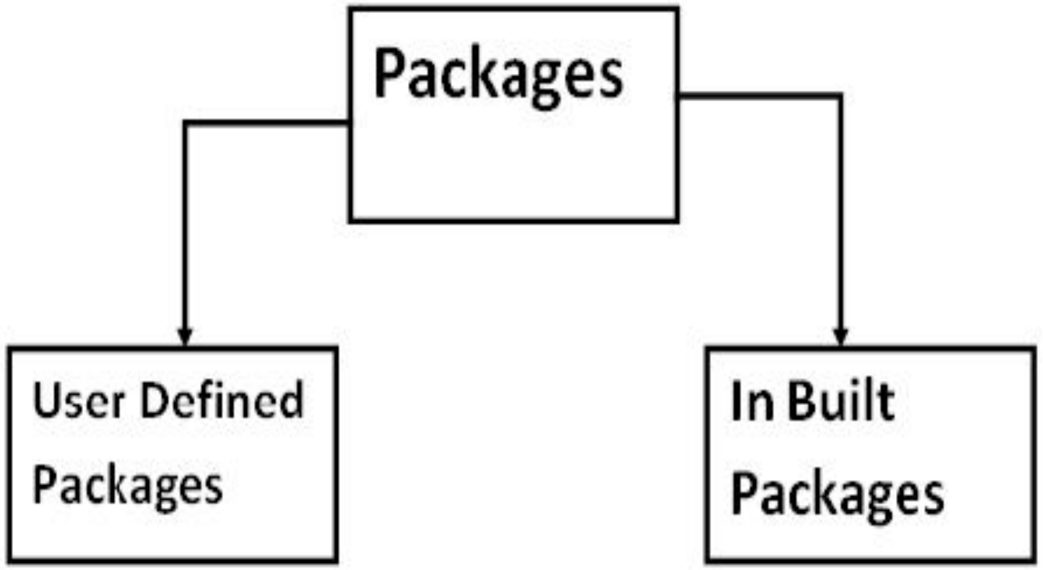
Package

- Packages are used in Java in order to **prevent naming conflicts**, to **control access**, to make **searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc.**

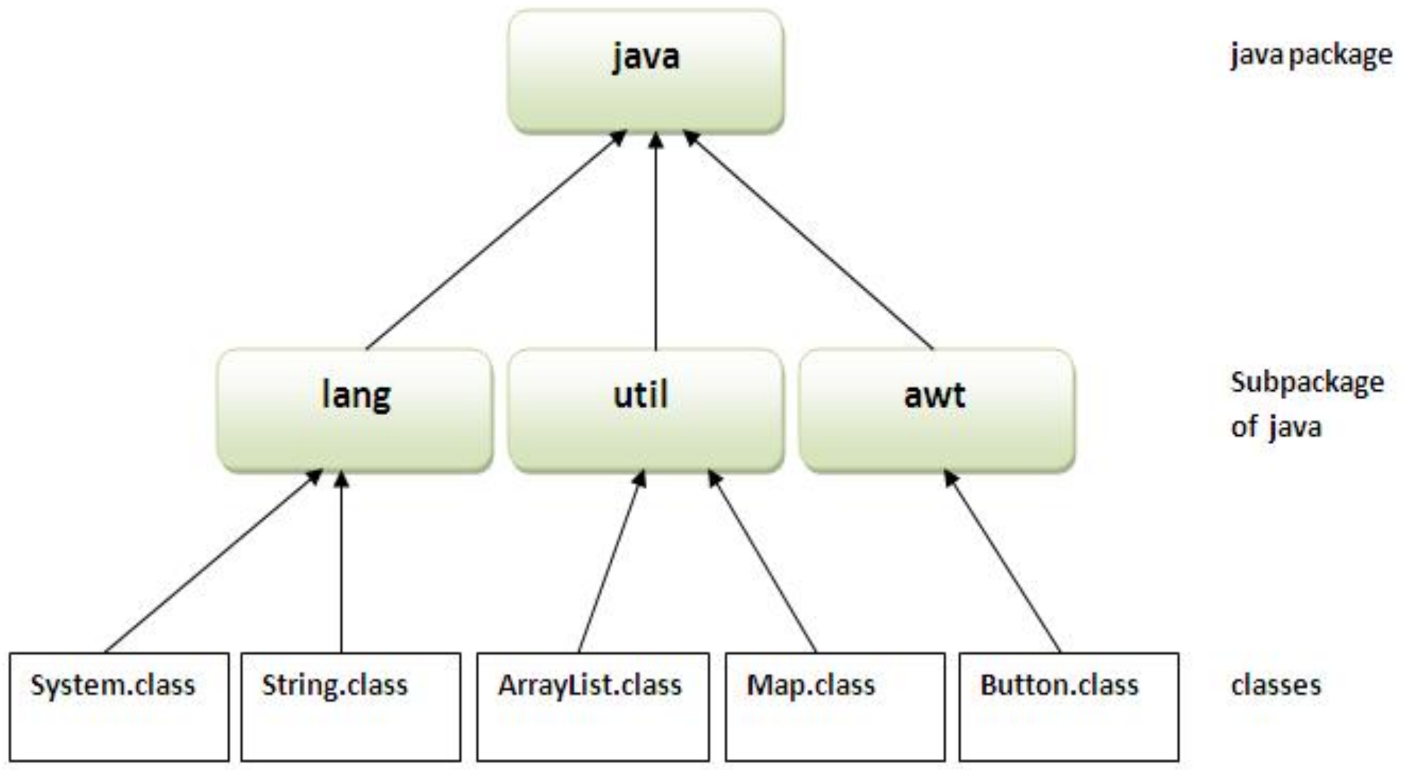
- A **Package** can be defined as a **grouping of related types (classes, interfaces, enumerations and annotations) providing access protection** and **namespace management.**

Package

- Basically, there are **2 types** of packages in JAVA.



In Built Package



Package

- **Advantage :**
  - Java package is used to **categorize** the classes and interfaces so that they can be easily maintained.
  - Java package **provides access protection.**
  - Java package **removes naming collision.**

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

Package

- **Creating package Syntax:**

```
package package_name;
```

- **Example:**

```
package employee;
```

Package

```
package first;

public class Simple // simple.java

{

      public static void main(String args[])

      {

    System.out.println(“Package is created");

   }

}
```

Package

- **Compile java package:**

```
javac -d Destination_folder file_name.java
```

- **After compile package,**
  - The folder with the given package name is created in the specified destination,
  - the compiled class files will be placed in that folder.

Package

- **Compile :** javac -d . Simple.java
- **Run :** java first.Simple
- **Output:** Package is created

- **Note:**
  - The -d is a switch that tells the compiler where to put the class file. It represents destination.
  - The . represents the current folder.

‘import’ keyword

- If a class wants to use another class in the same package, then package name need not be used.

- **Classes in the same package find each other without any special syntax.**

- There are three ways to access the package from outside the package.
  - **import packagename.*;**
  - **import packagename.classname;**
  - **fully qualified name.**

‘import’ keyword

- **Using packagename.* :**
  - If you use **packagename.*** then all the **classes and interfaces** of this package will be accessible but **not subpackages**.
  - **Syntax:** import packagename.*****;

- **Using packagename.classname :**
  - If you use **packagename.classname** then only declared **class** of this package will be accessible.
  - **Syntax:** import packagename.**classname**;

‘import’ keyword

- **Using fully qualified name :**
  - If you use **fully qualified name** then only **declared class** of this package will be accessible. So, no need to import the package.
  - 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.** **Ex:** java.util and java.sql packages contain **Date** class.

Access rules for packages

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Interface

- **An *interface* in java is a blueprint of a class. It has static constants and abstract methods.**
- The interface in Java is *a* ***mechanism used 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.

Interface

- **Why do we use interface ?**
- It is used to achieve total **abstraction**.
- Since java does not support **multiple inheritance** in case of class, but by using interface it can achieve multiple inheritance .
- It is also used to achieve **loose coupling**.

- **Note:**
  - **It cannot be instantiated just like the abstract class.**

Declaration of Interface

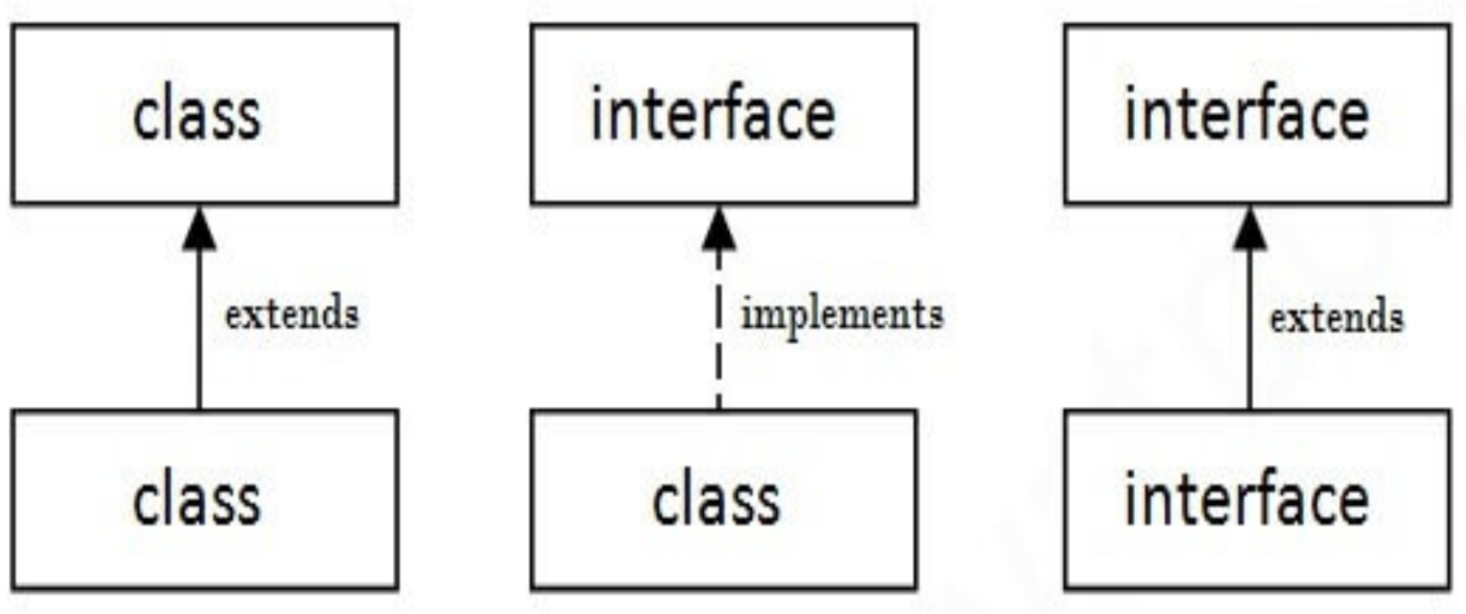
- An interface is declared by using the ***interface*** keyword.
- It provides total abstraction;
- To implement interface use ***implements*** keyword.
- All the **methods** in an interface **are declared with the empty body**, and all the **fields are public, static and final by default**.
- **A class that implements an interface must implement all the methods declared in the interface.**

Declaration of Interface

- **syntax:**

```
interface <interface_name>
{
    // declare constant fields
    // declare methods that abstract
    // by default.
}
```

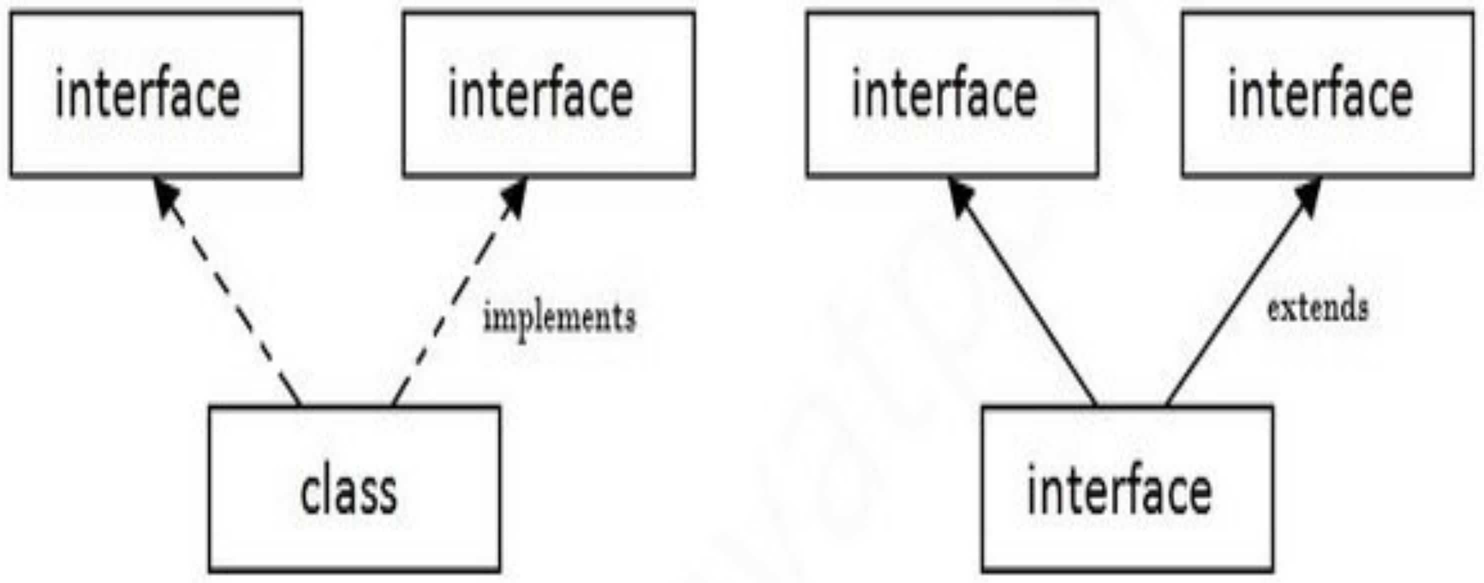
Class and Interface



Declaration of Interface

```
interface printable  //interface declaration
{
    void print();
}
class Sample implements printable  // implement interface
{
    public void print()
    {
        System.out.println("Hello");
    }
public static void main(String args[])
{
    Sample obj = new Sample();
    obj.print();
}
}
```

Multiple Inheritance using Interface



Multiple Inheritance in Java

Multiple Inheritance using Interface

```
interface Printable
{
	void print();
}
interface Showable
{
	void show();
}
class A7 implements Printable,Showable  //multiple inheritance
{
	public void print()
	{	System.out.println("Hello");
	}
	public void show()
	{	System.out.println("Welcome");
}
```

Multiple Inheritance using Interface

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

Interface Inheritance

- A class implements an interface, but one interface extends another interface.
- **Example:**

```
interface Printable
{
        void print();
}
interface Showable extends Printable  //interface inheritance
{
        void show();
}
```

Interface Inheritance

```
class TestInterface4 implements Showable
{
    public void print()
    {   System.out.println("Hello");
    }
    public void show()
    {   System.out.println("Welcome");
    }

public static void main(String args[])
{
    TestInterface4 obj = new TestInterface4();
    obj.print();
    obj.show();
 }
}
```

Abstract class

- In C++, if a class has at least one pure virtual function, then the class becomes abstract.
- Unlike C++, in Java, **a separate keyword *abstract* is used to make a class abstract.**
- **A class which is declared with the *abstract* keyword is known as an abstract class in Java. It can have abstract and non-abstract methods (method with the body).**

Abstract class

- **Note:**
  - An abstract class must be declared with an ***abstract*** keyword.
  - It can have abstract and non-abstract methods.
  - **It cannot be instantiated.**
  - It can have **constructors and static methods** also.
  - It can have final methods which will force the subclass not to change the body of the method.

Abstract class

- **Example of abstract class**

abstract class A{}

- **Example of abstract method**

abstract void printStatus();

Abstract class

```
abstract class Shape  //abstract class
{
     abstract void draw();  //abstract method
}

class Rectangle extends Shape
{
     void draw()
     {    System.out.println("drawing rectangle");
     }
}
```

Abstract class

```
class Circle1 extends Shape
{
    void draw()
    {   System.out.println("drawing circle");
    }
}
class TestAbstraction1
{
public static void main(String args[])
{
    Shape s=new Circle1();  //upcasting
    s.draw();
}
}
```

Final class

- The main purpose of using a class being declared as final is **to prevent the class from being subclasses**.
- If a class is marked as final then **no class can inherit** any feature from the final class.
- **We cannot extend a final class.**

Final class

```
final class XYZ  //final class
{
}
class ABC extends XYZ
{
    void demo()
    {    System.out.println("My Method");
    }
    public static void main(String args[])
    {
        ABC obj= new ABC();
        obj.demo();
    }
}
```

class V/s Interface

CLASS	**INTERFACE**
Supports only multilevel and hierarchical inheritances but not multiple inheritance	Supports all types of inheritance – multilevel, hierarchical and multiple
"extends" keyword should be used to inherit	"implements" keyword should be used to inherit
Should contain only concrete methods (methods with body)	Should contain only abstract methods (methods without body)
The methods can be of any access specifier (all the four types)	The access specifier must be public only
Methods can be final and static	Methods should not be final and static
Variables can be private	Variables should be public only
Can have constructors	Cannot have constructors
Can have main() method	Cannot have main() method as main() is a concrete method

Abstract class V/s Interface

ABSTRACT CLASS	**INTERFACE**
Abstract class can **have abstract and non-abstract**methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.

Abstract class V/s Interface

ABSTRACT CLASS	INTERFACE
An **abstract class**can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
An **abstract class**can be extended using keyword "extends".	An **interface class**can be implemented using keyword "implements".
A Java**abstract class**can have class members like private, protected, etc.	Members of a Java interface are public by default.

Example:
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}