❖ Interface:

- An interface is similar to the class.
- Interface is a collection of abstract methods and final static variables.
- Interface is used to achieve the multiple inheritance concepts in Java.
- All the abstract methods of the interface need to be defined in its sub class.
- An Interface does not contain any constructors.
- We cannot create the objects of an interface.
- All of the methods of the Interface are by default abstract.
- All of the variables of the Interface are by default static, final.
- An interface is not extends the properties of the class but it is implemented by a class.
- One interface can extend the multiple interfaces.

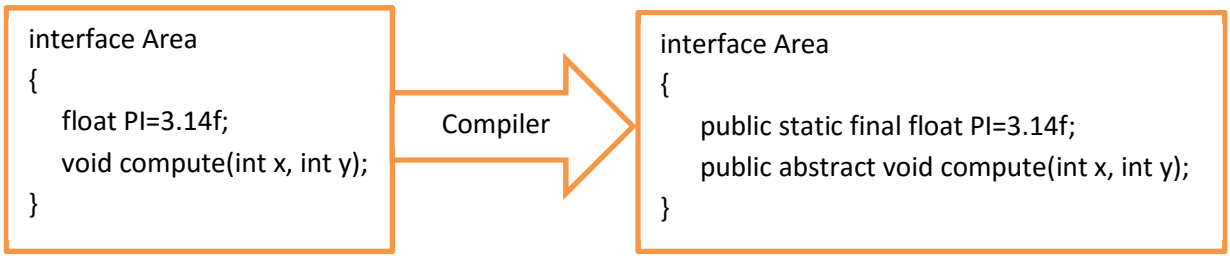
Syntax:

```
interface Interface_Name
{
        datatype final_variable_name1=value1; //Any number of final,static variables
        ---
        ---
        returntype method_name(parameter_list); //Any number of abstract method declarations
        ---
        ---
}
```

Example:

```
interface Area
{
        float PI=3.14f;
        void compute(int x,int y);
}
```

The java compiler adds public and abstract keywords before the interface method. It adds public, static and final keywords before data members.



Implementing interface:

- Class implements an interface.
- The class uses the **implements** keyword for implementing the interface.
- Class is responsible for implementing the abstract methods of the interface.

Syntax:

```
class class_name implements interface_name
{
    //body of class
}
```

Example:

```
interface Area
{
       float  PI=3.14f;
       void compute(int  r);
}

class Circle implements Area
{
      public void compute(int r)
      {
            System.out.println("Area of Circle="+(PI*r*r));
      }
      public static void main(String args[])
      {
           Circle c1=new Circle();
           c1.compute(10);
      }
}
```

Output:

```
Area of Circle=314.0
```

Extending interface:

- One interface can extend another interface same way that a class can extend another class.
- The **extends** keyword is used to extend an interface.
- In this case sub interface inherits the properties of super interface, but it will not define the methods of the super interface.
- Java class does not extend more than one class because java does not support to the multiple inheritance.
- But interface is not a class, it can extend two or more interfaces, they are separated by the commas.

Syntax:

```
interface sub_interface extends super_interface
{
       //body of sub interface
}
```

Example:

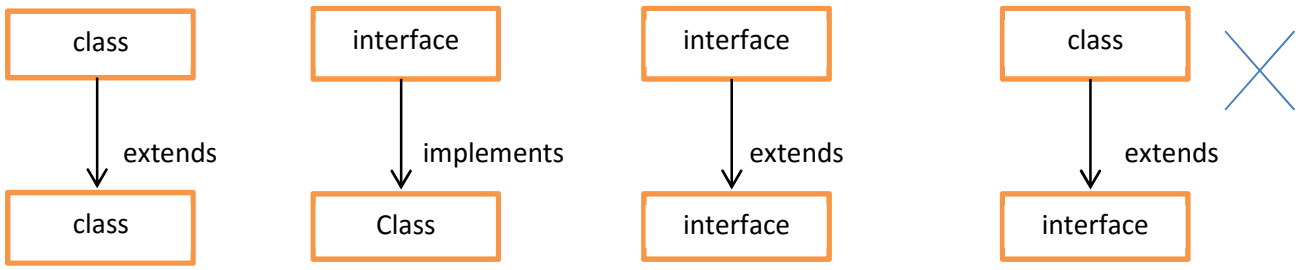
```
interface Abc
{
       void display();
}

interface Xyz extends Abc
{
      void show();

}
class Mnp implements Xyz
{
       public void display()
       {
                System.out.println("I am from Abc interface:");
       }
       public void show()
       {
                System.out.println("I am from Xyz interface:");
       }
       public static void main(String args[])
       {
                Mnp m1=new Mnp();
                m1.display();
                m1.show();
       }
}
```

Output:

```
I am from Abc interface:
I am from Xyz interface:
```

❖ Understand the relationship between classes and interfaces:

class extends class class implements interface interface extends interface interface extends class



❖ Multiple inheritance in Java by using interface:

- To inherit the properties of more than one base class into sub class is known as multiple inheritances.
- In multiple inheritance we can combine the features of more than one existing classes into new class.
- Below diagram shows the multiple inheritance concepts:

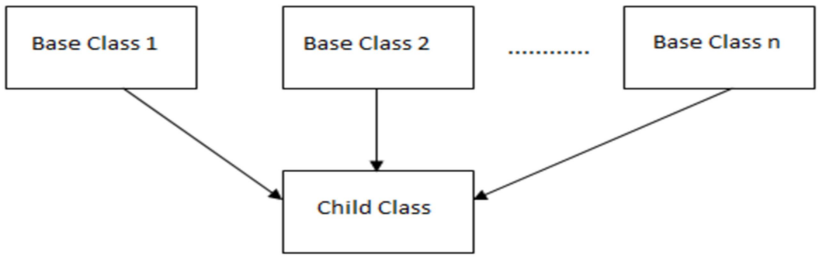


Fig: Multiple Inheritance

- Java classes cannot have more than one super class. But in most of the real time application multiple inheritances is required. So java provides an alternative approach is known as interface.
- Interface is a collection of static final variables and abstract methods. It is used to achieve the multiple inheritance in Java.
- If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance in Java.
- Below diagram shows, How to achieve the multiple inheritance in Java language:

Multiple Inheritance in Java

- In first diagram, class implementing more than one interface and in second diagram interface extending more than one interfaces to achieve the multiple inheritance in Java.

Example:

```
interface Abc
{
       void display();
}

interface Xyz
{
      void show();

}
class Mnp implements Abc,Xyz
{
       public void display()
        {
                 System.out.println("I am from Abc interface:");
        }
       public void show()
        {
                 System.out.println("I am from Xyz interface:");
        }
       public static void main(String args[])
       {
                 Mnp m1=new Mnp();
                 m1.display();
                 m1.show();
       }
}
```

Output:

```
I am from Abc interface:
I am from Xyz interface:
```

❖ Difference between Abstract class and Interface:

Abstract class	Interface
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods.
2) Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
3) Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
4) Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
5) The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
6) Abstract class can extend another Java class and implements multiple interfaces.	An interface can extend another Java interface only.
7) Abstract class achieves partial abstraction.	Interface achieves fully abstraction.

8) Example:
abstract class Abc
{
abstract void draw();
}
	Example:
	interface Xyz
	{
	void draw();
	}

❖ Difference between class and Interface:

Class	Interface
1) We can create objects from class.	We cannot create the objects from interface.
2) The members of a class can be private, public or protected.	The members of an interface are always public.
3) Class contains abstract or non-abstract methods.	Interface has only abstract methods.
4) Class can contain methods definitions.	Interface can contain only declaration of methods without body.
5) The class keyword is used to declare class.	The interface keyword is used to declare interface.
6) Class can implement any number of interfaces and can extend only one class.	An interface can extend multiple interfaces but cannot implement any class.
7) Constructor present in class	Constructor not present in class

8) Example:
class Abc
{
void draw();
}
	Example:
	interface Xyz
	{
	void draw();
	}

❖ Java Package:

- Putting classes and interfaces together is known as Package.
- Package is a collection of similar types of classes and interfaces.
- Packages are acts as container for the classes and interfaces.
- We can achieve the reusability features of Java by using Inheritance and packages.
- Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces.
- A package is a collection of related classes. It helps Organize your classes into a folder structure and make it easy to locate and use them. More importantly, it helps improve re-usability.
- Each package in Java has its unique name and organizes its classes and interfaces into a separate namespace, or name group.

Advantage of Java Package:

- 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 classes present in the packages of other programs can be reused.
- Package provides the good ways of hide the classes.
- In package, classes can be unique i.e. two classes in two different packages can have the same name.

There are two types of packages in Java:

1. Java API packages/Built-in packages.
2. User Defined packages.

❖ Java API Packages/Built-in Packages:

- Java API provides a number of classes grouped into different packages according to their functionality.
- The already defined package like java.io.*, java.lang.* etc. are known as built-in packages.
- Below diagram shows Java API packages:

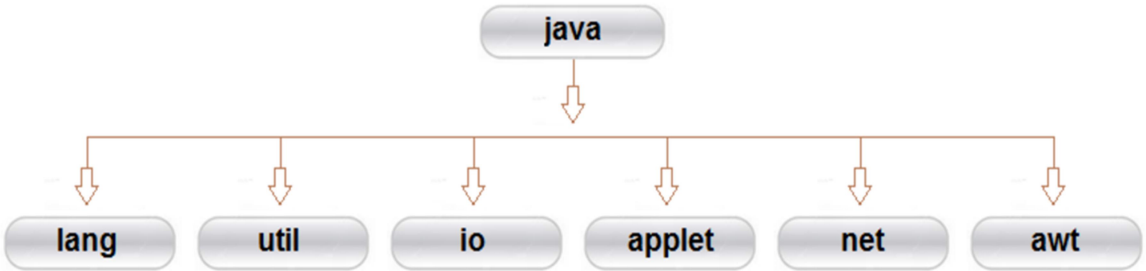


Fig. Java API Packages

- There are six main packages are present in Java programming language.
- Following table gives the information about the Java system packages and their classes

Package Name	Contents of the package
java.lang	Language support classes. They include classes for primitive types, string, math functions, thread and exceptions.
java.util	Language utility classes such as vectors, hash tables, random numbers, data, etc.
java.io	Input/output support classes. They provide facilities for the input and output of data.
java.applet	Classes for creating and implementing applets.
java.net	Classes for networking. They include classes for communicating with local computers as well as with internet servers.
java.awt	Set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.

❖ User defined Packages:

- Creating a package in java is quite easy.
- A package is always defined in a separate folder having the same name as a package name.
- A package is a collection of related classes. It helps Organize your classes into a folder structure and make it easy to locate and use them. More importantly, it helps improve re-usability.
- Each package in Java has its unique name and organizes its classes and interfaces into a separate namespace, or name group.
- The package must contain one or more classes or interfaces. This implies that a package cannot be empty.
- The classes or interfaces that are in the package must have their source files in the same directory structure as the name of the package.
- A package is always defined in a separate folder having the same name as a package name.
- Define all classes in that package folder.
- All classes of the package which we wish to access outside the package must be declared public.
- All classes within the package must have the package statement as its first line.
- All classes of the package must be compiled before use (So that its error free)

Creation of packages includes following steps:

1. Declare the package at the beginning of the java source code file using below syntax.

Syntax:

```
package package_name;
```

Above statement should be used in the beginning of the program to include that program in that particular package.

2. Define a class which is to be put in the package and declare it public.

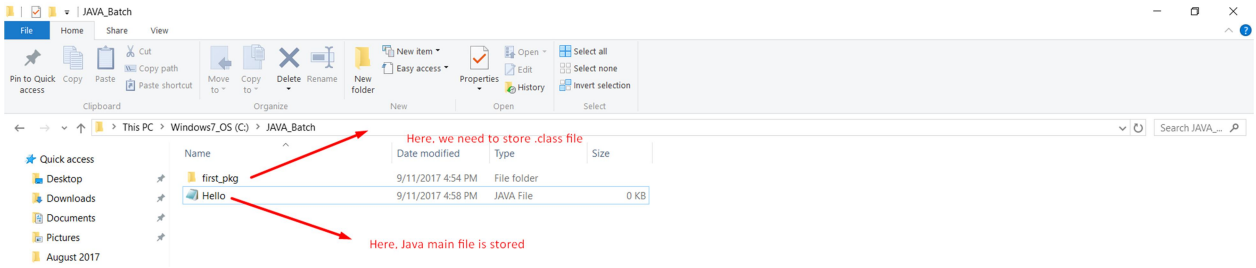
Example:

```
package first_pkg;

public class Hello
{
        public void display()
        {
                System.out.println("I am from Hello class:");
        }
}
```

In above example **first_pkg** is the package name and class Hello is added into it.

3. Create the subdirectory which has given the same name of package and stored it into main directory where the source code is present.

4. Compile the source file, after that .class file stored into the subdirectory which is created in step 3.



❖ How to access package from another package/import statement:

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.* statement then all the classes and interfaces of this package will be accessible.
- import is a predefined keyword.

Syntax:

```
import Package_Name.*;
```

Example:

```
//Save by Abc.java

package first_pkg;

public class Abc
{
	public void display()
	{
		System.out.println("I am from Abc class:");
	}
}
```

```
//Save by Xyz.java

package second_pkg;
import first_pkg.*;

class Xyz
{
	public static void main(String args[])
	{
		Abc a1=new Abc();
		a1.display();
	}
}
```

Output:

```
I am from Abc class
```

2) import packagename.classname;

- If you use package.classname statement then only declared class of that package will be accessible.
- import is a predefined keyword.

Syntax:

```
import Package_Name.classname;
```

Example:

```
//Save by Abc.java

package first_pkg;

public class Abc
{
	public void display()
	{
		System.out.println("I am from Abc class:");
	}
}
```

```
//Save by Xyz.java

package second_pkg;
import first_pkg.Abc;

class Xyz
{
	public static void main(String args[])
	{
		Abc a1=new Abc();
		a1.display();
	}
}
```

Output:

```
I am from Abc class
```

3) Using fully qualified name;

- If you use fully qualified name then only declared class of this package will be accessible.
- No need to use the import keyword.
- 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:

```
//Save by Abc.java

package first_pkg;

public class Abc
{
	public void display()
	{
		System.out.println("I am from Abc class:");
	}
}
```

```
//Save by Xyz.java

package second_pkg;

class Xyz
{
	public static void main(String args[])
	{
		first_pkg.Abc a1=new first_pkg.Abc ();  //using fully qualified name
		a1.display();
	}
}
```

Output:

```
I am from Abc class
```

❖ Java Static Import

- The static import feature of Java is used to access any static member of a class directly.
- There is no need to use the class name while accessing the static members.
- The static import statement can be used to import static members from classes and use them without using class name.
- The difference between import and static import: - The import allows the java programmer to access classes of a package without using package name whereas the static import feature allows accessing the static members of a class without using the class name.
- The import provides accessibility to classes and interface whereas static import provides accessibility to static members of the class.

Syntax:

```
import static Package_Name.*;
          or
import static Package_Name.classname;
```

Example:

```
import static java.lang.System.*;

class StaticImportDemo
{
         public static void main(String args[])
         {

                  out.println("Hello"); //Now no need of System.out
                  out.println("Java");
         }
}
```

Output:

```
Hello
Java
```

Advantage:

- Less coding is required if you have access any static member of a class.

Disadvantage:

- If you overuse the static import feature, it makes the program unreadable and unmaintainable.