JAVA INTERFACE AND PACKAGES

INTRODUCTION TO 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 to achieve abstraction**.
- There can be only abstract methods in the java interface not method body.
- It cannot be instantiated just like abstract class.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

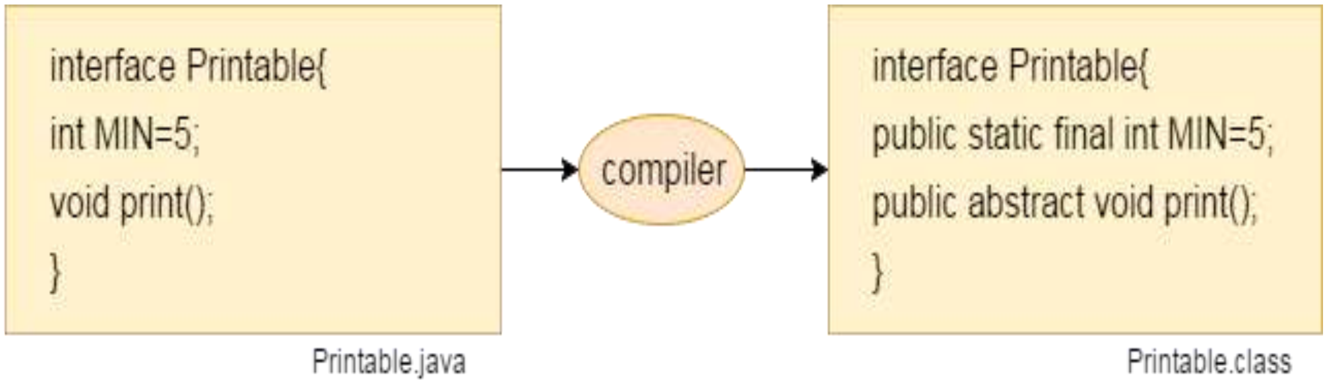
DECLARING INTERFACE

- The **interface** keyword is used to declare an interface.
- Syntax:

```
public interface NameOfInterface
{
   // Any number of final, static fields
   // Any number of abstract method declarations
}
```

interface

- An interface is implicitly abstract.
- So not need to use the **abstract** keyword while declaring an interface.
- Each Fields in an interface is also implicitly **static** and **final**, so the static and final keyword is not needed (Refer below diagram).
- Methods in an interface are also implicitly public.


```
interface Printable{
int MIN=5;
void print();
}
```

```
interface Printable{
public static final int MIN=5;
public abstract void print();
}
```

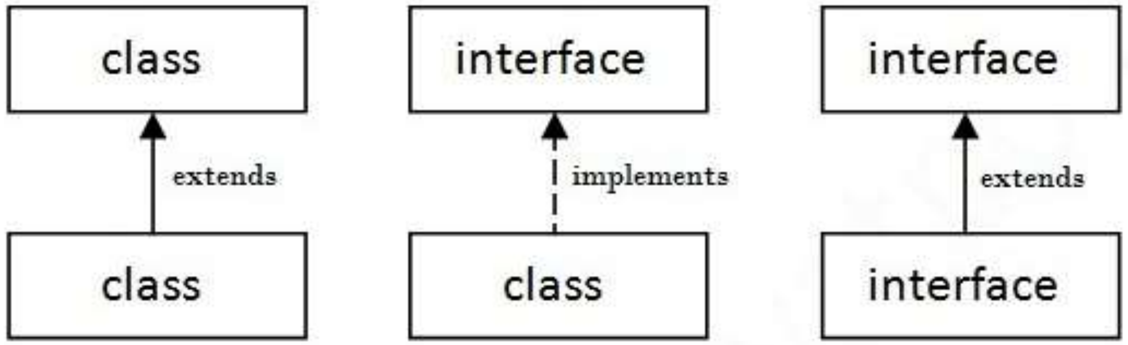
Interface example

Example:

```
 Interface ItemConstants                          //interface declared
 {
      int code = 1001;               // Variable declared in interface
      string name = “Fan”;
      void display();                // Method declared in interface
}
```

Extending interface

- An interface can extend another interface like class.
- The **extends** keyword is used to extend an interface.
- A **class implements an interface**.



```
interface Name2 extends Name1
{
          Body of Name2
}
```

implementing interface

- Interfaces are used as “Super classes” whose properties are inherited by classes.
- syntax

```
Class className implements interfacename
{
        Body of classname
}
```

implementing interface

Example:

```
interface Drawable             // Interface declared
{
void draw();
}
class Rectangle implements Drawable   //implementing
{
public void draw()
{
System.out.println("drawing rectangle");
}}
```

OUTPUT: drawing rectangle

implementing interface

```
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();
 } }
```

OUTPUT:

```
Hello
Welcome
```

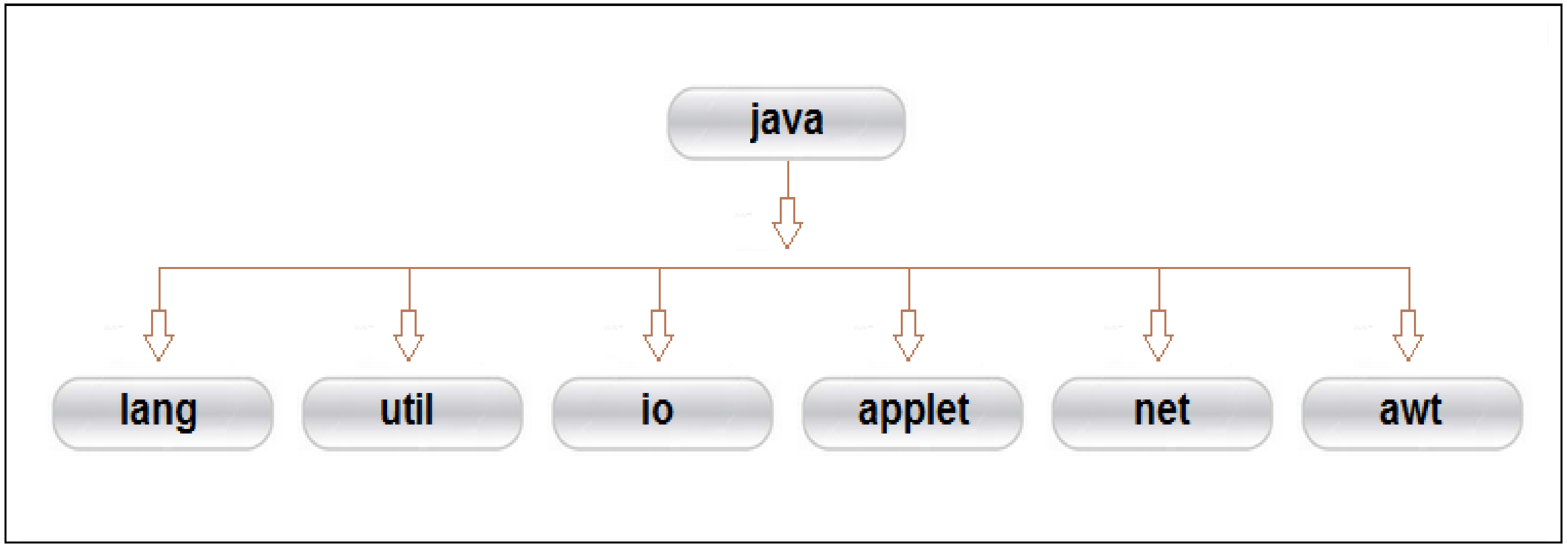
JAVA PACKAGES

JAVA PACKAGES

- A **java package** is a group of similar types of classes, interfaces and sub-packages.
- Package in java can be categorized in two types,
  1. Java API package (Built-in package)
  2. User-defined package. (Defined by user)
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.
- Java package provides access protection.

1. JAVA API PACKAGES

- Java **API(Application Program Interface)** provides a large numbers of classes grouped into different packages according to functionality.
- Most of the time we use the packages available with the Java API.



1. JAVA API PACKAGES

PACKAGE	CONTENTS
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.

USING API PACKAGES

- The import statements are used at the top of the file, before any class declarations.
- The first statement allows the specified class in the specified package to be imported.
- For example,

Import java.awt.color;

- The above statement imports class color and therefore the class name can now be directly used in the program.
- The below statement imports every class contained in the specified package.

Import java.awt.*;

- The above statement will bring all classes of java.awt package.

USER DEFINED PACKAGES CREATE A PACKAGE

- To create a package, a name should be selected for the package.
- Include a **package** statement along with that name at the top of every source file that contains the classes, interfaces.
- The package statement should be the first line in the source file.
- There can be only one package statement in each source file, and it applies to all types in the file.

CREATING A PACKAGE

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

CREATING A PACKAGE

To compile and run the package,

To Compile: javac -d . Simple.java

To Run: java mypack.Simple

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

OUTPUT: Welcome to package

ACCESSING A PACKAGE

ACCESSING A PACKAGE:

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

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

ACCESSING A PACKAGE

1. USING IMPORT PACKAGE.*;

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

Output: Hello

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

ACCESSING A PACKAGE

2. USING PACKAGE.CLASSNAME

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

Output: Hello

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

ACCESSING A PACKAGE

2. USING FULLY QUALIFIED NAME

- Using fully qualified name can declared a 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.

ACCESSING A PACKAGE

2. USING 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(args[])
{
pack.A obj = new pack.A();
(Fully qualified name)
   obj.msg();
  } }
```

END OF THE TOPIC !!!