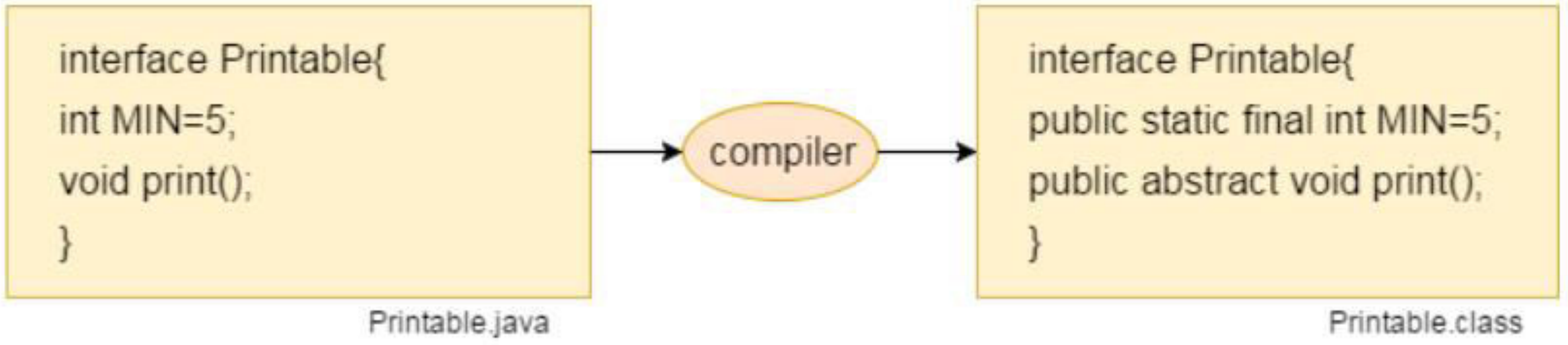
Interface and packages

interface

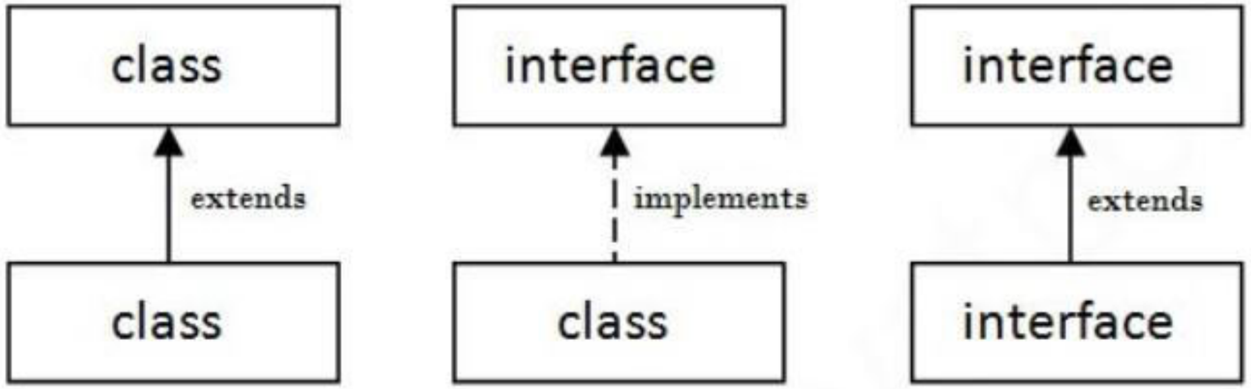
- Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).
- If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract.
- An interface is declared by using the interface keyword.
- All the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
- The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.
- A class that implements an interface must implement all the methods declared in the interface.

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



```
// interface
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void run(); // interface method (does not have a body)
}
```

The relationship between classes and interfaces



```java
// Interface
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    // The body of sleep() is provided here
    System.out.println("Zzz");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

- Like **abstract classes**, interfaces **cannot** be used to create objects
- Interface methods do not have a body - the body is provided by the "implement" class
- On implementation of an interface, you must override all of its methods
- Interface methods are by default `abstract` and `public`
- Interface attributes are by default `public`, `static` and `final`
- An interface cannot contain a constructor (as it cannot be used to create objects)

Accessing Implementations Through Interface References:

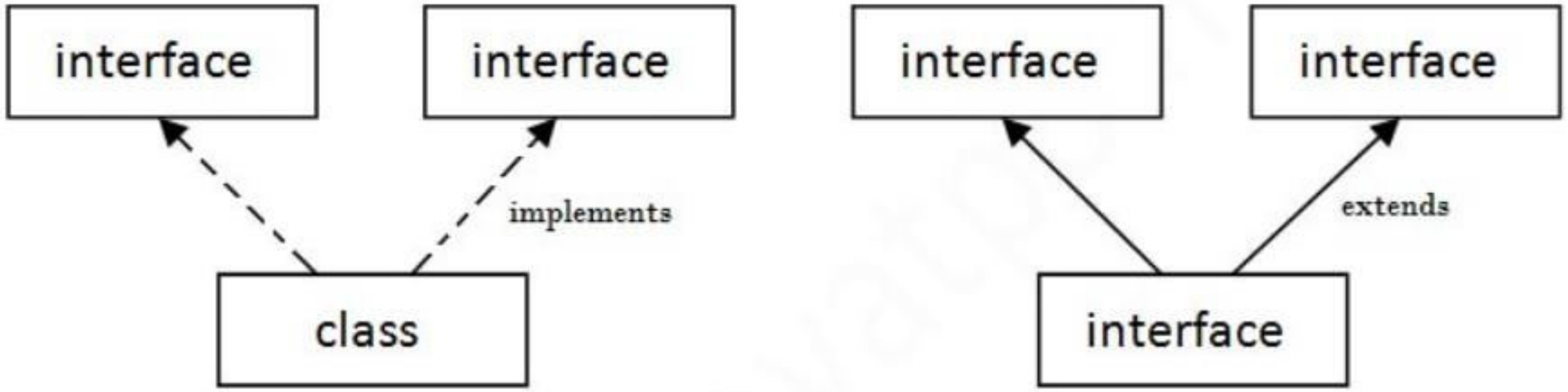
```
//Interface declaration: by first user
interface Drawable{
void draw();
}
//Implementation: by second user
class Rectangle implements Drawable{
public void draw(){System.out.println("drawing rectangle");}
}
class Circle implements Drawable{
public void draw(){System.out.println("drawing circle");}
}
//Using interface: by third user
class TestInterface1{
public static void main(String args[]){
Drawable d=new Circle();//In real scenario, object is provided by method e.g. getDrawable()
d.draw();
}}
```

```
interface Bank{
float rateOfInterest();
}
class SBI implements Bank{
public float rateOfInterest(){return 9.15f;}
}
class PNB implements Bank{
public float rateOfInterest(){return 9.7f;}
}
class TestInterface2{
public static void main(String[] args){
Bank b=new SBI();
System.out.println("ROI: "+b.rateOfInterest());
}}
```

```
ROI: 9.15
```

Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



```java
interface FirstInterface {
  public void myMethod(); // interface method
}

interface SecondInterface {
  public void myOtherMethod(); // interface method
}

class DemoClass implements FirstInterface, SecondInterface {
  public void myMethod() {
    System.out.println("Some text..");
  }
  public void myOtherMethod() {
    System.out.println("Some other text...");
  }
}

class Main {
  public static void main(String[] args) {
    DemoClass myObj = new DemoClass();
    myObj.myMethod();
    myObj.myOtherMethod();
  }
}
```

Multiple inheritance is not supported through class in java, but it is possible by an interface

multiple inheritance is not supported in the case of class because of ambiguity. However, it is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class.

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

class TestInterface3 implements Printable, Showable{
public void print(){System.out.println("Hello");}
public static void main(String args[]){
TestInterface3 obj = new TestInterface3();
obj.print();
 }
}
```

```
Hello
```

Interface inheritance

A class implements an interface, but one interface extends another interface.

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

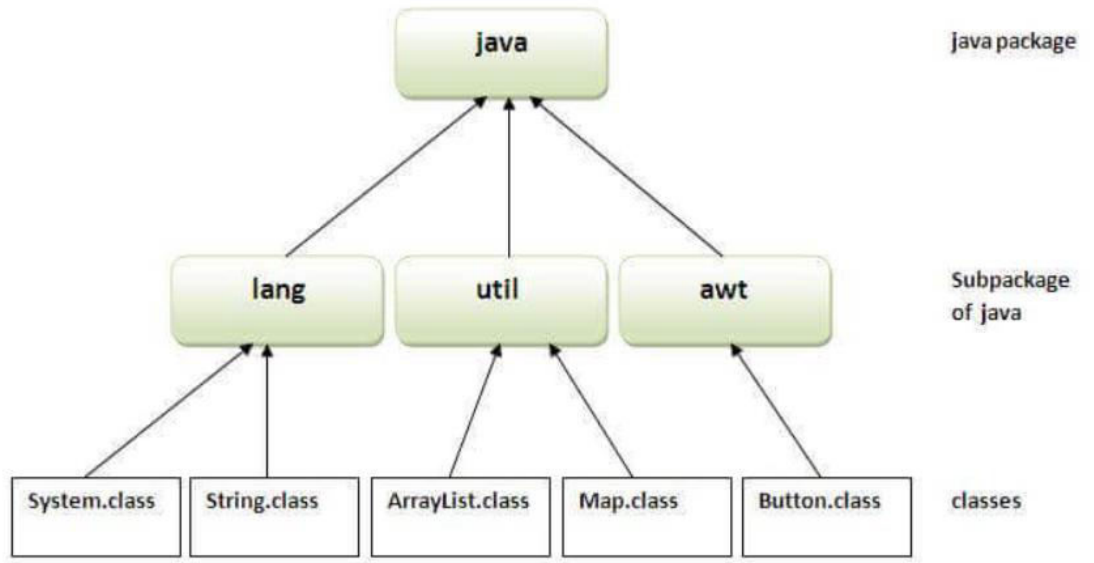
```
Hello
Welcome
```

packages

- A **java package** is a group of similar types of classes, interfaces and sub-packages.
- We can assume package as a folder or a directory that is used to store similar files.
- Package in java can be categorized in two forms:
  - built-in packages:math, util, lang, i/o etc are the example of built-in packages.
  - user-defined packages:Java package created by user to categorize their project's classes and interface are known as 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.

Built-in Packages
These packages consist of a large number of classes which are a part of Java **API**.Some of the commonly used built-in packages are:
1) **java.lang:** Contains language support classes(e.g classed which defines primitive data types, math operations, String, StringBuffer, Thread). This package is automatically imported.
2) **java.io:** Contains classed for supporting input / output operations.
3) **java.util:** Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations,Scanner.
4) **java.applet:** Contains classes for creating Applets.
5) **java.awt:** Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc).
6) r supporting networking



Import a Package

To import a whole package, end the sentence with an asterisk sign (*).

```
import java.util.*;
```

Import a Class

To use a class or a package from the library, you need to use the `import` keyword:

```
import package.name.Class;   // Import a single class
import package.name.*;   // Import the whole package

 import java.util.Scanner;
```

In the example above, `java.util` is a package, while `Scanner` is a class of the `java.util` package.

To use the `Scanner` class, create an object of the class and use any of the available methods found in the `Scanner` class

```
import java.util.Scanner;

class MyClass {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);
    System.out.println("Enter username");

    String userName = myObj.nextLine();
    System.out.println("Username is: " + userName);
  }
}
```

User-defined packages:

These are the packages that are defined by the user.

How to Create a user defined package:

- Choose the name of the package
- Include the package command as the first line of code in your Java Source File.
- The Source file contains the classes, interfaces, etc you want to include in the package
- Compile to create the Java packages

```
package nameOfPackage;
```

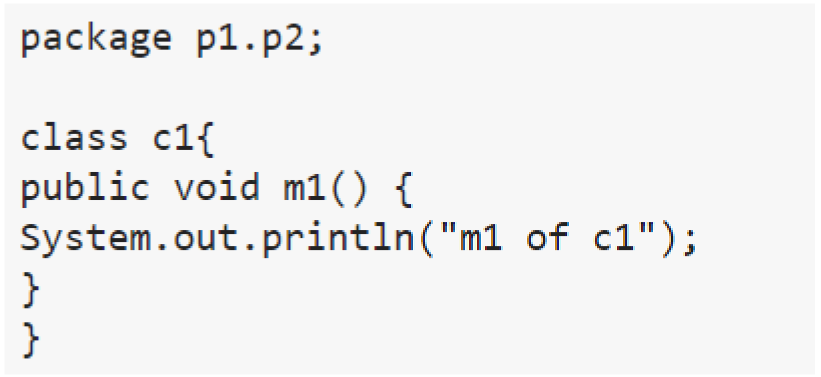
While creating a package, care should be taken that the statement for creating package must be written before any other import statements

```
package p1;
class c1
{
 public void m1()
{
System.out.println("m1 of c1");
}
 public static void main(string
args[])
{
c1 obj = new c1();
 obj.m1();
}
}
```

1.Save the file as c1.java into the folder d:\ECE
  now the file is at location   d:\ECE\c1.java

2. Go to command prompt then Compile and create package
D:\ECE>javac –d . c1.java
The above command forces the compiler to create a package in the current working directory.
-d means create a package(directory)
. means it creates a package p1 in the current working directory ie., d:\ECE   and place the class file in d:\ECE\p1
D:\ECE\p1\c1.class

D:\ECE> javac –d .. C1.java
The above command creates a package in the parent working directory.
D:\p1\c1.class



```
package p1.p2;

class c1{
public void m1() {
System.out.println("m1 of c1");
}
}
```

D:\ECE>javac –d . c1.java
D:\ECE\p1\p2\c1.class
Instead of . We can also specify the path where we want to create a package.

3. Run the program: d:\ECE> java p1.p2.c1

How to Import Package

To create an object of a class (bundled in a package), in your code, you have to use its fully qualified name.

```
java.awt.event.actionListner object = new java.awt.event.actionListner();
```

Instead, it is recommended you use the import statement.

```
import packageName;
```

```
import java.awt.event.*; // * signifies all classes in this package are imported
import javax.swing.JFrame // here only the JFrame class is imported
//Usage
JFrame f = new JFrame; // without fully qualified name.
```

```
package p3;
import p1.*; //imports classes only in package p1 and NOT  in the sub-package p2
class c3{
  public   void m3(){
     System.out.println("Method m3 of Class c3");
  }
  public static void main(String args[]){
    c1 obj1 = new c1();
    obj1.m1();
  }
}
```

1.Save the file with name c3.java in D:\ECE

2.Compile the program

D:\ECE>javac –d . C3.java

3.Create package at d:\ECE\p3\c3.class

4. Run the program

D:\ECE> java p3.c3