Sukkur Institute of Business Administration University

Department of Computer Science

Object Oriented Programming using Java

BS – II (CS/AI/SE)

Spring-2024

Lab # 09: Let's learn about Packages and Interfaces(Abstraction)

Instructor: Nimra Mughal

Objectives

After performing this lab, students will be able to understand:

- Packages
- Interfaces

Package

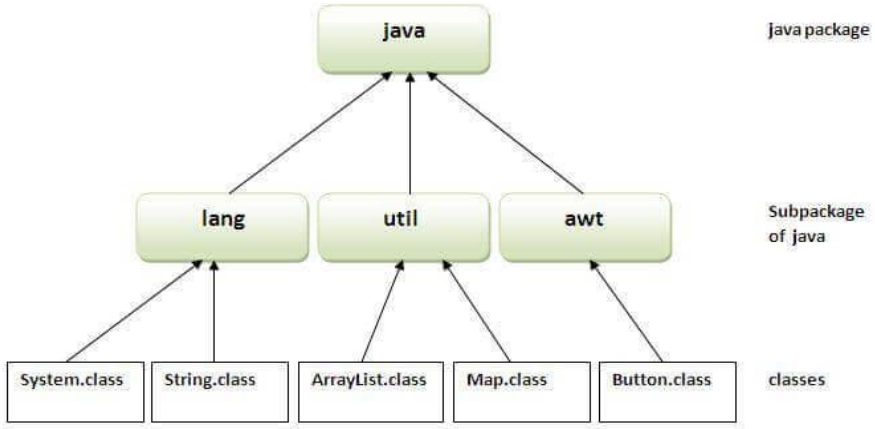
A **package** is a namespace that organizes a set of related classes and interfaces. Conceptually you can think of packages as being similar to different folders on your computer.

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.

Advantages of using Java Packages:

1. **Better organization**: As in large java projects where we have several hundreds of classes, it is always required to group the similar types of classes in a meaningful package name.
2. **Reusability**: While developing a project in java, we often feel that there are few things that we are writing again and again in our code. Using packages, you can create such things in form of classes inside a package and whenever you need to perform that same task, just import that package and use the class.
3. **Name conflicts**: Java package removes naming collision. For example; We can define two classes with the same name in different packages so to avoid name collision, we can use packages



Types of packages in Java

- *User defined package*: The package we create is called user-defined package.
- *Built-in package*: The already defined package like java.io.*, java.lang.* etc are known as built-in packages.

Example

A class Calculator is created inside a package name letmecalculate. To create a class inside a package, declare the package name in the first statement in your program.

A class can have only one package declaration.

Calculator.java file created inside a package letmecalculate

```
package letmecalculate;

public class Calculator {
   public int add(int a, int b){
       return a+b;
   }
}// save this file as Calculator.java. This class is used to create a package as
well as to perform addition
```

Now lets see how to use this package in another program.

```
import letmecalculate.Calculator;

public class Demo{
   public static void main(String args[]){
       Calculator obj = new Calculator();
       System.out.println(obj.add(100, 200));
   }
}// save this file as Demo.java
```

Steps for compiling and running java package

1. Access path of the folder/directory where you have saved the Calculator file
   For example : C:\Users\HP\Desktop\OOP, it suggests that Calculator file is on desktop in OOP folder
2. javac -d . Calculator.java \\ compile the package
3. javac Demo.java
4. java Demo

Access package from another package

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

1. Import package.classname;
2. Fully qualified name.
3. Import package.*

1. Using package.classname

```
import letmecalculate.Calculator;
```

2. Using fully qualified name

```
public class Demo{
```

```
   public static void main(String args[]){
    letmecalculate.Calculator obj = new letmecalculate.Calculator();
    System.out.println(obj.add(100, 200));
   }
}
```

3. **Using package.***

   If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

```
Import java.util.*;
```

Interfaces

An interface in Java is a blueprint of a class. 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.

Since Java 8, we can have **default** and **static methods** in an interface.

Since Java 9, we can have **private methods** in an interface.

Why interfaces?

These are reasons to use interface:

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.

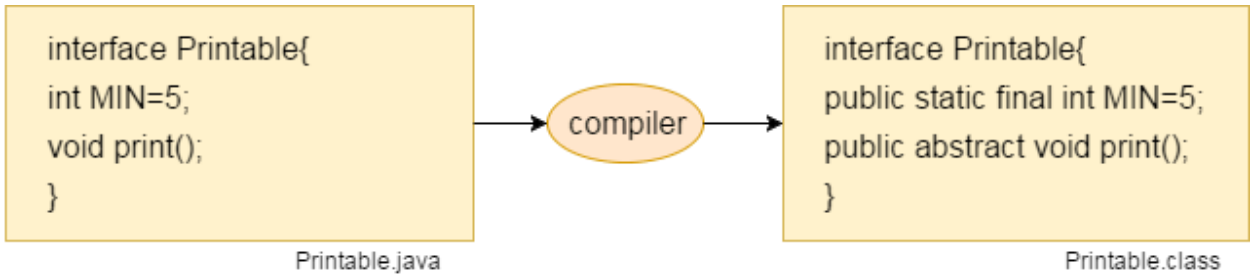
Declaration of an Interface

An interface is declared by using the interface keyword. It provides total abstraction; means 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.

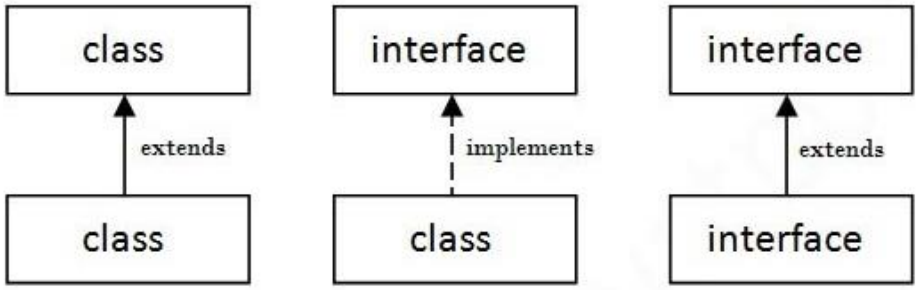
Syntax:

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

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



The relationship between classes and interfaces

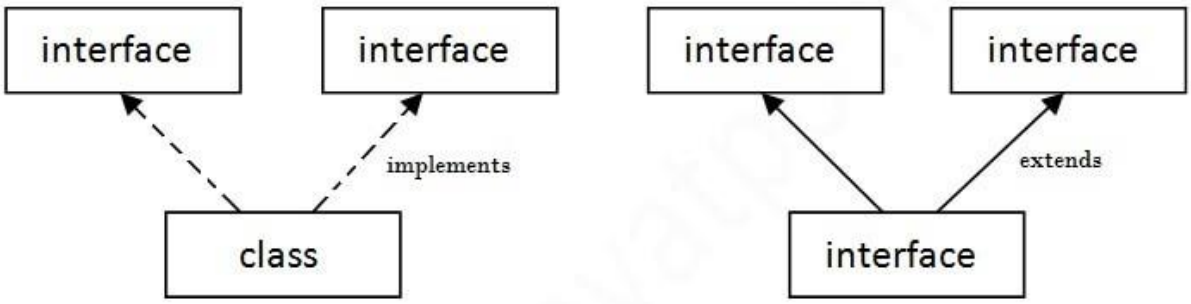


Example:

Open interface_VehicleExample.java

Multiple inheritance in Java by interface

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



Multiple Inheritance in Java

```
//you can now inherit features of both interfaces
Class Test implements Interface1, Interface 2
```

Default Method in Interface

Since Java 8, we can have method body in interface. But we need to make it default method. Let's see an example:

```
interface Drawable{
     void draw();
     default void msg(){System.out.println("default method");}
}
class Rectangle implements Drawable{
     public void draw(){System.out.println("drawing rectangle");}
}
class TestInterfaceDefault{
     public static void main(String args[]){
          Drawable d=new Rectangle();
          d.draw();
          d.msg();
}}
```

Static Method in Interface

```
interface Drawable{
     void draw();
     static int cube(int x){return x*x*x;}
}
class Rectangle implements Drawable{
      public void draw(){System.out.println("drawing rectangle");}
}
class TestInterfaceStatic{
      public static void main(String args[]){
           Drawable d=new Rectangle();
           d.draw();
           System.out.println(Drawable.cube(3));
       }
}
```

Nested Interface

Nested interface which is **declared within the interface**

```
interface interface_name{
 ...
 interface nested_interface_name{
  ...
```

```
  }
}
```

Nested interface which is **declared within the class**

```
class class_name{
 ...
 interface nested_interface_name{
  ...
 }
}
```

Example of nested interface which is declared within the interface

```
interface OuterInterface {

    void anymethod();
    interface InnerInterface {

       void print();
    }

  }
 class NestedInterface implements OuterInterface.InnerInterface {

    public void print() {
       System.out.println("Print method of nested interface");
     }

    public static void main(String args []) {
       NestedInterface obj = new NestedInterface();
       obj.print();

    }
 }
```

Example of nested interface which is declared within the class

```
class OuterClass {
    interface InnerInterface {
       int id = 20;
       void print();
    }
  }
 class NestedInterfaceDemo implements OuterClass.InnerInterface {

    public void print() {

       System.out.println("Print method of nested interface");

     }

    public static void main(String args []) {

       NestedInterfaceDemo obj = new NestedInterfaceDemo();
```

```
        obj.print();
        System.out.println(obj.id);
        // Assigning the object into nested interface type
        OuterClass. InnerInterface obj2 = new NestedInterfaceDemo();
        obj2.print();
    }
}
```

Exercises

Question: 1

1. Create three packages, think of them yourself
2. Put two different classes in each package
3. Import all three of these packages in class named PackagePractice, you will have access to 6 classes
4. Call methods of these 6 classes and use them in PackagePractice

Question: 2 (Interfaces)

What is wrong with the following interface?

```
public interface SomethingIsWrong {
    void aMethod(int aValue){
        System.out.println("Hi Mom");
    }
}
```

1. Fix the interface in question.
2. Is the following interface valid?

```
public interface Marker {
}
```

Question: 3 (Interfaces)

1. Create the Animal interface.
   a. Declare method legs.
   b. Declare a method eat.
2. Create the Spider, Caterpillar and Cat class that implements animal interface.
   a. All classes implement the Animal interface.

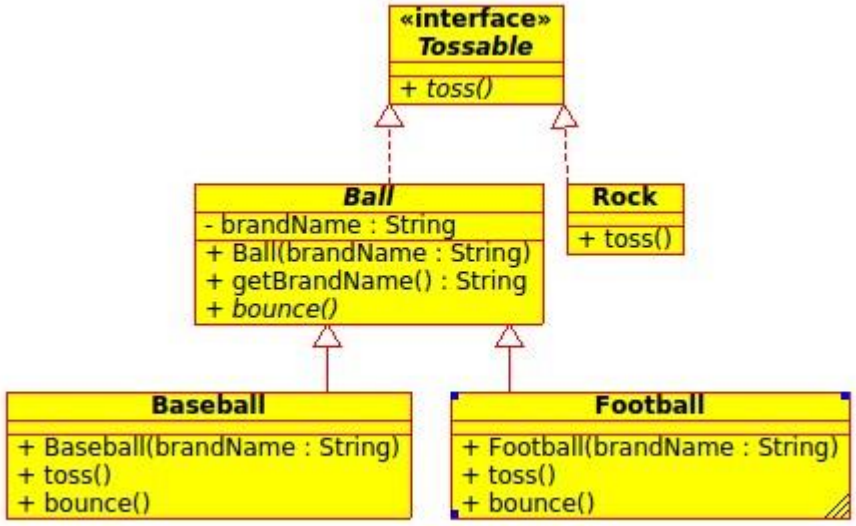
b. Implement the eat and legs method according to the kind of animal (can only provide a printing statement, up to you!)
c. Provide main method and demonstrate each of above methods.

Question: 4 (Multiple Inteheritance)

Create multiple inheritance using interfaces. Create a class Person that derived from class Employee and Officer class. Employee class contains details( ) method and Officer class contains basic info( ) method. You can simply type display any text in details and info methods.

Question: 5

Implement the following class hierarchy. You do not need to fill in the method bodies with actual code for the **toss** or **bounce** methods. You can use string message or whatever suitable you want.



Question: 6

Declare the Rectangle, SportCar, Manager classes as classes that implement the Printable interface and run the given application. The output of this application should be the details of each one of the objects that were instantiated.

```
interface Printable
{
    public void print();
}
public class PrintableDemo
{
     public static void main(String args[])
     {  Printable vec[] =
{new Rectangle(110,80), new SportCar(“Toyota”, 989621),
```

```
                    new Rectangle(34,32), new Manager("John",
40),                    new Rectangle(54,10), new SportCar("Audi",
2365644),                    new SportCar("Mazda", 4322343), new Manager("Joji",
22)};        for(int index=0; index<vec.length; index++)
        {
             vec[index].print();
        }
      }
}
```