Unit 4

Package and Interface

Prepared by: Shiva Bhattarai

Outline

- Packages: defining a package, finding packages and CLASSPATH
- Access Protection
- Interfaces: defining an interface, implementing interfaces, nested interfaces, applying
- Interfaces, variables in interfaces, interfaces can be extended

Package

- Group of similar types of classes, interfaces and sub-packages.
- Mechanism to encapsulate a group of classes, sub-packages and interface.
- Used to provide access protection, namespace management and to make searching/locating and usage of classes, interfaces, enumerations and annotations easier.
- Types of packages
  - Built-in package
    - Existing java package
    - Example: java, lang, awt, io, util, javax, swing net, sql etc. java .io.*, java.util.Scanner.
  - User-defined package
    - Created by user to categorize classes and interface.

Package

- We need to put related classes into package, and can simply import classes from existing packages and use it in our program.
- Package is a container of group of related classes where some of the classes are accessible exposed and other are kept for internal purpose.

Advantage of Package

- Can be considered as data encapsulation(data-hiding)
- Preventing name conflicts
  - Example there can be two classes with same name employee in two package, college.staff.cse.employee and college.staff.ee.employee.
- Making searching/locating and usage of classes, interfaces enumerations and annotations easier.
- Providing controlled assess
  - Protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A default member is accessible by class in the same package only.
- Used to categorize the classes and interfaces so that they can be easily maintained.
- Re-usability of the package.

Creating a package.

- Creating package is easy, include a package command followed by name of the package as the first statement in java source file.
- General form is

  ```
  package package_name;
  ```

  – Here, package_name is name of the package.
- Here, the statement creates a package called mypackage.

  ```
  package mypackage;
  public class student{
      statements;
  }
  ```

  – Above statement creates a package called mypackage.

Creating a package(example)

```
package mypackage;

public class PackageDemo {
   public static void    main(String[] args) {
      System.out.println("Hello good morning");
   }
}
```

Creating a package(example)

```
package animal;

public class cat {
   public void display(){
      System.out.println("mew mew");
   }
}

import animal.*;
public class animalsound {
   public static void main(String[] args) {
      cat c = new cat();
      c.display();
   }
}
```

Creating a package.

- Java uses file system directories to store packages. For example, the .class files for any classes will be part of mypackage and must be stored in a directory called mypackage.
- More than one file can include the same package statement.
- Can create a hierarchy of packages. To do so, simply separate each package name from the one above it by use of period. The general form is

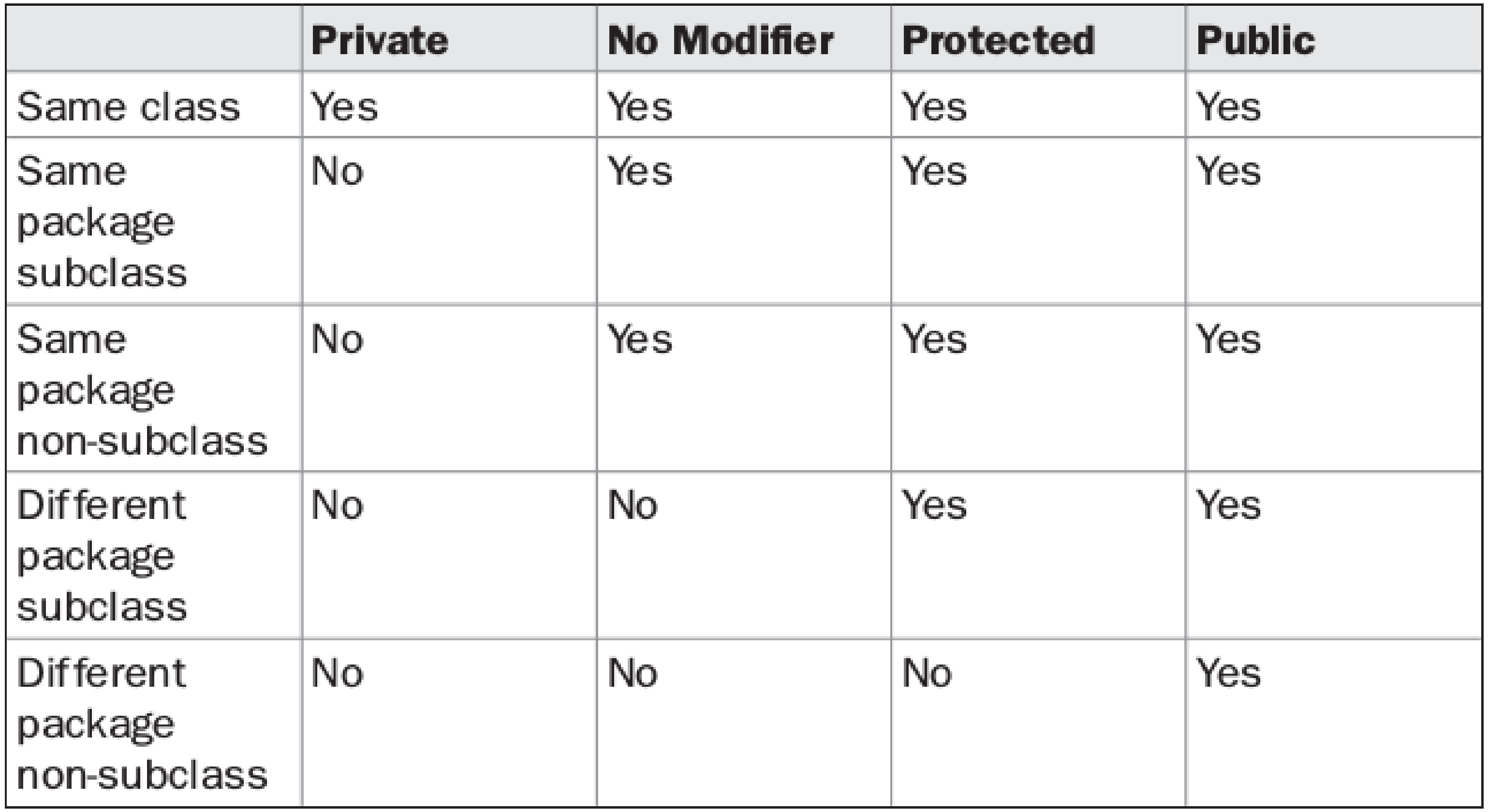
  package pkg1[.pkg2[.pkg3]];
- Example package java.awt.image;
- Needs to be stored in java\awt\image in a window environment.

Finding package and CLASSPATH

- Packages are mirrored by directories, here an important questions raises.
  - How does the java run-time system know where to look for the package that we create?
- Answer has three parts.
  - By default, java run-time system uses current working directory as its starting point. Thus if the package is in sub-directory of the current directory it will be found.
  - We can specify a directory path or paths by setting the CLASSPATH environmental variable.
  - We can use the -**classpath** option with java and javac to specify the path to the classes.

Access protection

Class Member Access



	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Importing package

- Java has import statement that allows us to import an entire package or use certain classes and interface defined in the package.
- The general form is
  - import package.name.ClassName; // to import certain class only.
  - Import package.name.* // to import the whole package.
- Example
  - import java.util.Date; //import onlye date class
  - Import java.io.*; //import everything inside java.io package.

Importing package

- **import** keyword is used to import built-in and user-defined packages into the java source file so that the class can refer to a class that is in another package directly.
- Three ways to access the package from outside the package.
  - Fully qualified name
  - Import all the classes from the particular package i.e. Import package.*;
  - Importing only the class which is needed i.e., Import package.classname;

Fully qualified name

```
package pack;
   public class classA{
      public void display(){
         System.out.println("i am in classA 
package");
   }
}

class classB{
   public static void main(String[] args){
      // using fully qualified name
      pack.classA objA = new pack.classA();
      objA.display();
   }
}
```

Fully qualified name

- In case of fully qualified name than we can declare a class of this package will be accessible.
- No need to use keyword import.
- Need to use fully qualified every-time when we have to use package, which is also a bad practice.

Import package.*;

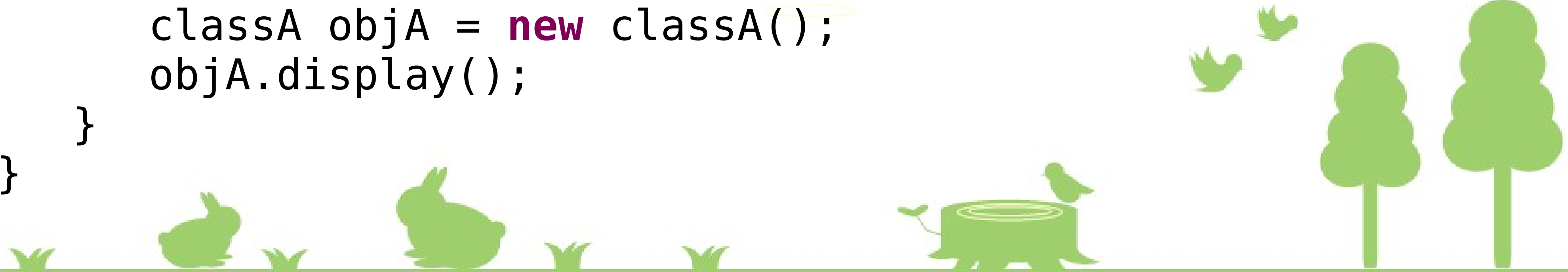
```
package pack;
public class classA{
   public void display(){
      System.out.println("i am in classA package");
   }
}

import pack.*;
class classB{
   public static void main(String[] args){
      classA objA = new classA();
      objA.display();
   }
}
```

Import package.*;

- If we use package.* then all the classes and interfaces of this package will be accessible but not subpackages.
- Import keyword is used to make the classes and interface of another package accessible to the current package.

Import package.classname;

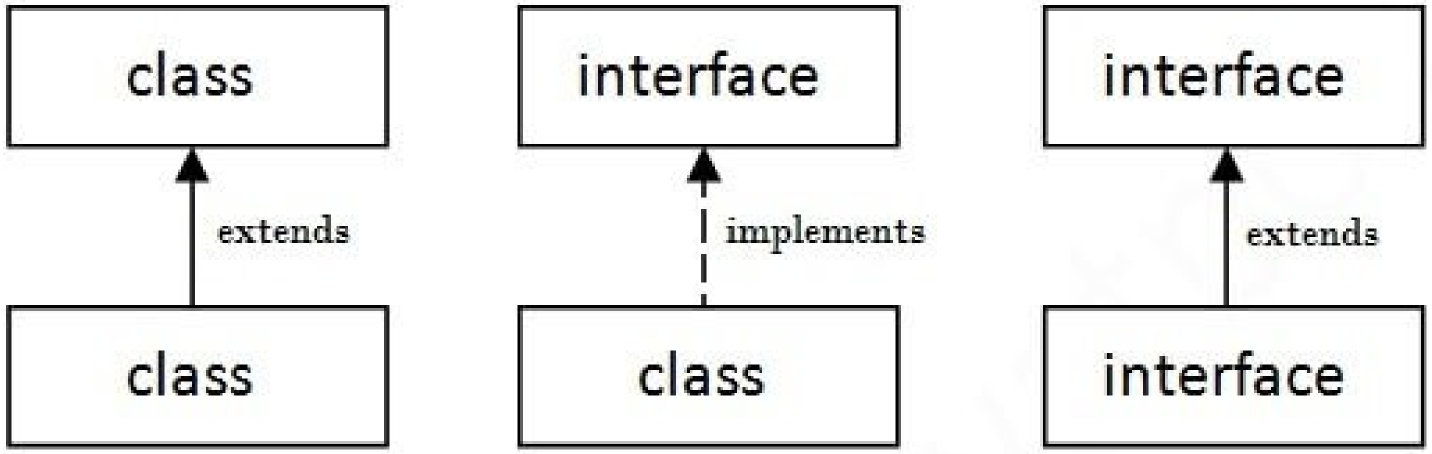
```
package pack;
public class classA{
   public void display(){
      System.out.println("i am in classA package");
   }
}

import pack.classA;
class classB{
   public static void main(String[] args){
      classA objA = new classA();
      objA.display();
   }
}
```



Import package.classname;

- If we want package to access a specified class from the package then we use this technique.
- Here we use import keyword to import the package and class.

Interface


class
extends
class
interface
implements
class
interface
extends
interface

Interface

- Looks like class but not a class, we can say a blueprint of a class, a reference type in java.
  - Cannot have instance variables, however can contain public static final variables i.e., constant class variables.
  - All methods are abstract i.e., method without body.
- Using it, we can specify what a class must do, but not how it does it.
- Can achieve abstraction, since it has static constants and abstract method by default.
- Once it is defined, any number of classes can implement an interface. Also one class implements many numbers of interface i.e., multiple inheritance using interface.

Interface

- To implement an interface, a class must create a complete set of methods defined by the interface, but each is free to implement its own details.
- Designed to support dynamic method resolution at run time.
- To use the interface, class should use keyword **implements**. If interface is used by other interface then implements keyword can be used.

Defining Interface

```
access interface name{
   return-type method-name1(parameter-list);
   return-type method-name2(parameter-list);
   type final-varname1 = value;
   type final-varname2 = value;
   // ...
   return-type method-nameN(parameter-list);
   type final-varnameN = value;
}
```

Defining a Interface

- Where,
  - access
    - is either public or not used, default access applies when no specifier is used. i.e. interface is only available to the members of the package in which it is declared.
    - When interface is public, then the interface can be used by another code.
  - name
    - Name of the interface which is a valid identifier.
  - Method
    - Methods in interface are **abstract** methods.
    - Each class which includes an interface must implement all the methods.
  - Variables
    - Can be declared inside interface declarations, they are implicitly **final** and **static**. i.e. they cannot be changed by the classes which implements the interface.
    - Variables must have constant values.
    - Since interface is public then all the methods and variable are implicitly declared public.

Why Interface?

- To achieve total abstraction.
- Implement multiple inheritance, since java does not support multiple inheritance in case of class, but by using interface, we can achieve multiple inheritance.
- Example

```
interface player{
    final int id = 20;
    int move();
}
```

```java
interface Polygon {
    void getArea(int length, int breadth);
}
class Rectangle implements Polygon {
    public void getArea(int length, int breadth) {
        System.out.println("The area of the rectangle is " + (length * breadth));
    }
}
public class interfaceclass {
    public static void main(String[] args) {
        Rectangle r1 = new Rectangle();
        r1.getArea(5, 6);
    }
}
```

```java
interface Vehicle {
    void changeGear(int a);
    void speedUp(int a);
    void applyBrakes(int a);
}
class Bicycle implements Vehicle{
    int speed;
    int gear;
    public void changeGear(int newGear){
        gear = newGear;
    }
    public void speedUp(int increment){
        speed = speed + increment;
    }
    public void applyBrakes(int decrement){
        speed = speed - decrement;
    }
    public void printStates() {
        System.out.println("speed: " +
speed + " gear: " + gear);
    }
}
```

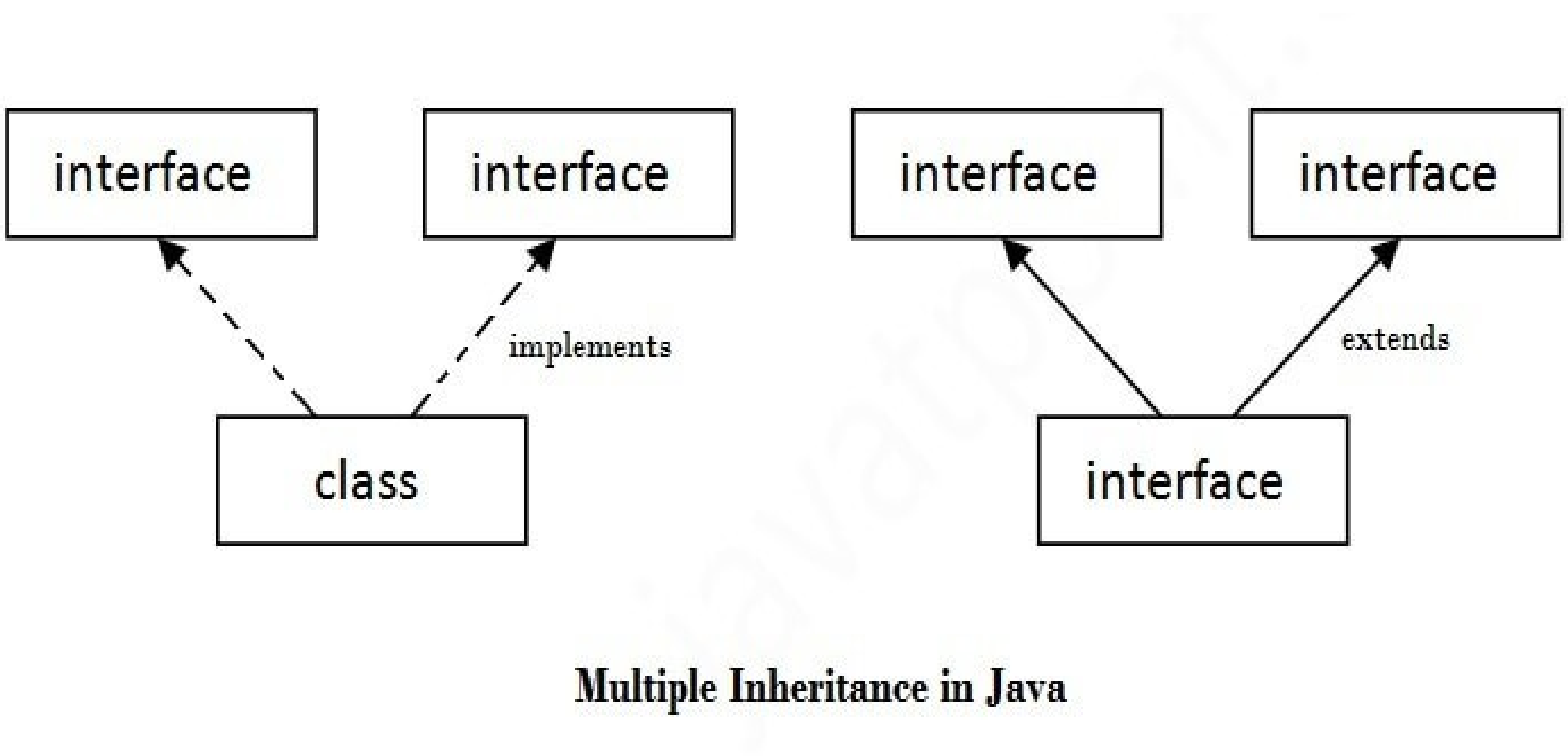
```java
class Bike implements Vehicle {
    int speed;
    int gear;
    public void changeGear(int newGear){
        gear = newGear;
    }
    public void speedUp(int increment){
        speed = speed + increment;
    }
    public void applyBrakes(int decrement){
        speed = speed - decrement;
    }
    public void printStates() {
        System.out.println("speed: " + speed +
gear: " + gear);
    }
}
public class interfaceclass {
    public static void main (String[] args) {
        Bicycle bicycle = new Bicycle();
        bicycle.changeGear(2);
        bicycle.speedUp(3);
        bicycle.applyBrakes(1);
        System.out.println("Bicycle present
state :");
        bicycle.printStates();
        Bike bike = new Bike();
        bike.changeGear(1);
        bike.speedUp(4);
        bike.applyBrakes(3);
        System.out.println("Bike present state
        bike.printStates();
    }
}
```

Why Interface?

- To achieve total abstraction.
- Since java does not support multiple inheritance in case of class, but by using interface, we can achieve multiple inheritance.
- To achieve loose coupling.
- Example

```
interface player{
    final int id = 20;
    int move();
}
```

Multiple inheritance in Java by interface



Multiple Inheritance in Java

```
interface A{
   public void display();
}
interface B{
   public void display();
}
class C implements A, B{
   public void display(){
      System.out.println("Implementing more than one 
interfaces");
   }
}
public class InterfaceExample{
public static void main(String[] args) {
   C c = new C();
   c.display();
   }
}
```

```java
interface Writeable{
    void writes();
}
interface Readable {
    void reads();
}
class student implements Readable,Writeable{
    public void reads(){
      System.out.println("Student reads.. ");
    }
    public void writes(){
      System.out.println("Student writes..");
    }
    public static void main(String args[]){
      student s = new student();
      s.reads();
      s.writes();
    }
}
```

```
interface vehicleone{
    int  speed=90;
    public void distance();
}
interface vehicletwo{
    int distance=100;
    public void speed();
}
class Vehicle  implements vehicleone, vehicletwo{
    public void distance(){
        int  distance=speed*100;
        System.out.println("distance travelled is "+distance);
}
    public void speed(){
        int speed=distance/100;
    }
}
public static void main(String args[]){
        System.out.println("Vehicle");
        Vehicle obj = new Vehicle();
        obj.distance();
        obj.speed();

}
```