JAVA PROGRAMMING

Chap 4 : Packages and Interfaces

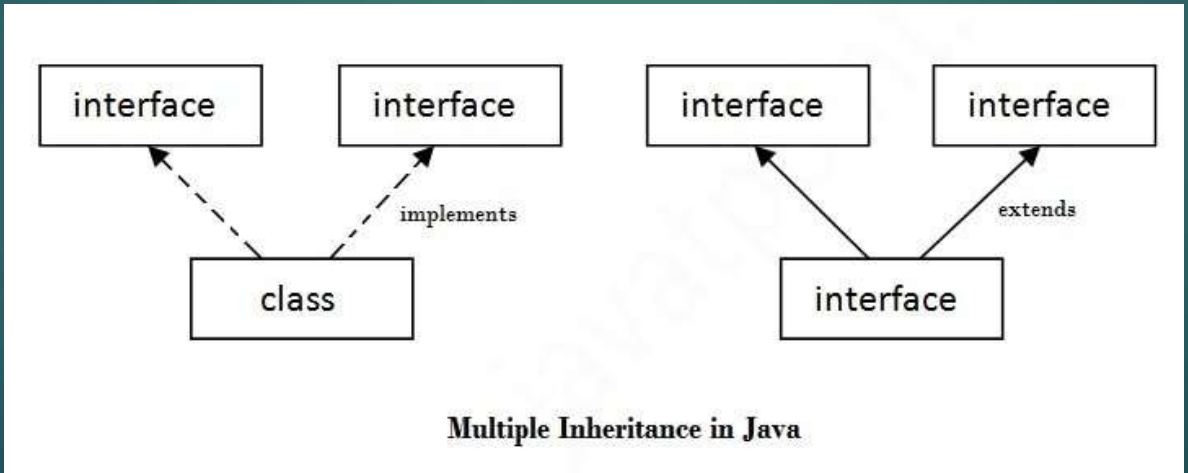
Interfaces

- Multiple Inheritance is not allowed in Java.
- Instead Interface is used to implement multiple inheritance.
- An **interface** in Java is a blueprint of a class. It has static constants and abstract methods.
- There can be only abstract methods in the Java interface, having no method body.
- It is used to achieve abstraction and multiple inheritance in Java.
- Java Interface also represents the IS-A relationship (i.e. Inheritance)
- **Syntax for inheritance**
- 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 are abstract by default.
}
```

- a class extends another class, an interface extends another interface, but a class implements an interface.
- Interface do not have constructor.
- 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 and method definitions may be provided in the subclass.
- Any class can extend only 1 class but can implement infinite number of interface.



Multiple Inheritance in Java

Abstract Class vs Interface

Abstract Class	Interface
It is a class that contains one or more abstract methods, which are to be defined by sub class	It contains only method declarations and no definitions
Abstract class definition begins with keyword “abstract” followed by class definition	Interface definition begins with keyword “interface” followed by interface definition
Abstract classes can have general methods defined along with some methods with only declarations	Interfaces have all methods with only declarations that are defined in subclasses i.e. classes that implements the interface
Variables in abstract classes need not be public, static and final	All Variables in an interface is by default public, static and final
Abstract classes doesn’t support multiple inheritance	Interfaces supports multiple inheritance
An abstract class can contain private and protected members	An interface can only have public members
Abstract classes are fast	Interfaces are slow as it requires extra indirection to find corresponding methods in the actual class

Interface Example

```
import java.util.*;
interface Base {
    public void read(float x);
    public void calculate();
    public void display();
}
class Sphere implements Base {
  protected float r,vol;
  public void read(float x)
 {
   r=x;
 }
 public void calculate()
    {
        vol=3.14f*r*r*r*4/3;
    }
  public void display()
 {
     System.out.println("Volume of Sphere =
"+vol);
 }}
```

```
class Hemisphere implements Base {
  protected float r,vol;
  public void read(float x)
 {
   r=x;
 }
 public void calculate()
    {
        vol=3.14f*r*r*r*2/3;
    }
  public void display()
 {
     System.out.println("Volume of
Hemisphere = "+vol);
 }}
```

```
class Main
{
  public static void main(String args[])
  {
    float rad;
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter Radius : ");
    rad=sc.nextFloat();
    Sphere s=new Sphere();
    s.read(rad);
    s.calculate();
    s.display();
    Hemisphere h=new Hemisphere();
    h.read(rad);
    h.calculate();
    h.display();
}}
```

```
Enter Radius :
10
Volume of Sphere = 4186.6665
Volume of Hemisphere = 2093.3333
```

Packages

- Package is a way to organize code i.e. similar function or related classes must be in a package.
- Thus, one package will have all the classes of similar functionality while another package for another type of classes.
- A **java package** is a group of similar types of classes, interfaces and sub-packages.
- 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.
- 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.
  - Making searching/locating and usage of classes, interfaces, enumerations and annotations easier
- All we need to do is put related classes into packages. After that, we can simply write an import class from existing packages and use it in our program.
- We can reuse existing classes from the packages as many time as we need it in our program.

Creating a Package

Step-1 : To create a package we use the keyword "package" followed by the name of the package

Step-2 : Create a class and write members of the class

Step-3 : Store the program file in a folder with the same name as that of the package and store the file as the name of the class that has main() method. Eg : if the package name is "world", then the folder in which the program is stored should also be "world". If the name of the class having the main() method is "Hello", then the program should be stored as "Hello.java"

Step-4 : Compile the program as usual using javac. This has to be done in the folder of the package. In case of above eg, in the folder "world" we compile the file as javac Hello.java

Step-5 : Traverse out of the package folder and execute the program with syntax –

java package_name.class_name.

In the above eg, java world.Hello

Creating a Package

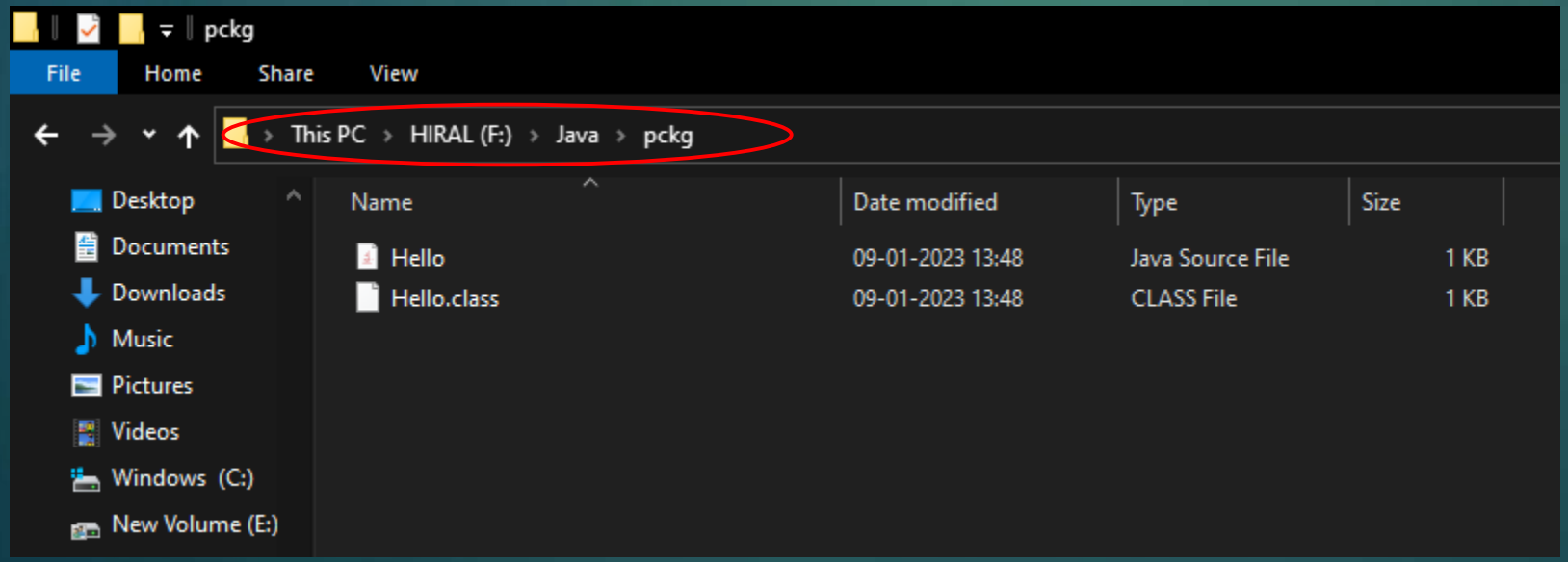
```
package pckg;
class Hello
{
    public static void main(String args[])
    {
        System.out.println("Hello World !!!");
    }
}
```

```
F:\Java\pckg>javac Hello.java

F:\Java\pckg>cd..

F:\Java>java pckg.Hello
Hello World !!!

F:\Java>
```



Creating a Sub-Package

- To create a sub-package we use the same process that we used to create the package.
- The folder of the sub package name has to be stored inside the folder with the package name.

```
package pckg.india;
class Hello
{
    public static void main(String args[])
    {
        System.out.println("Hello Indians !!!");
    }
}
```

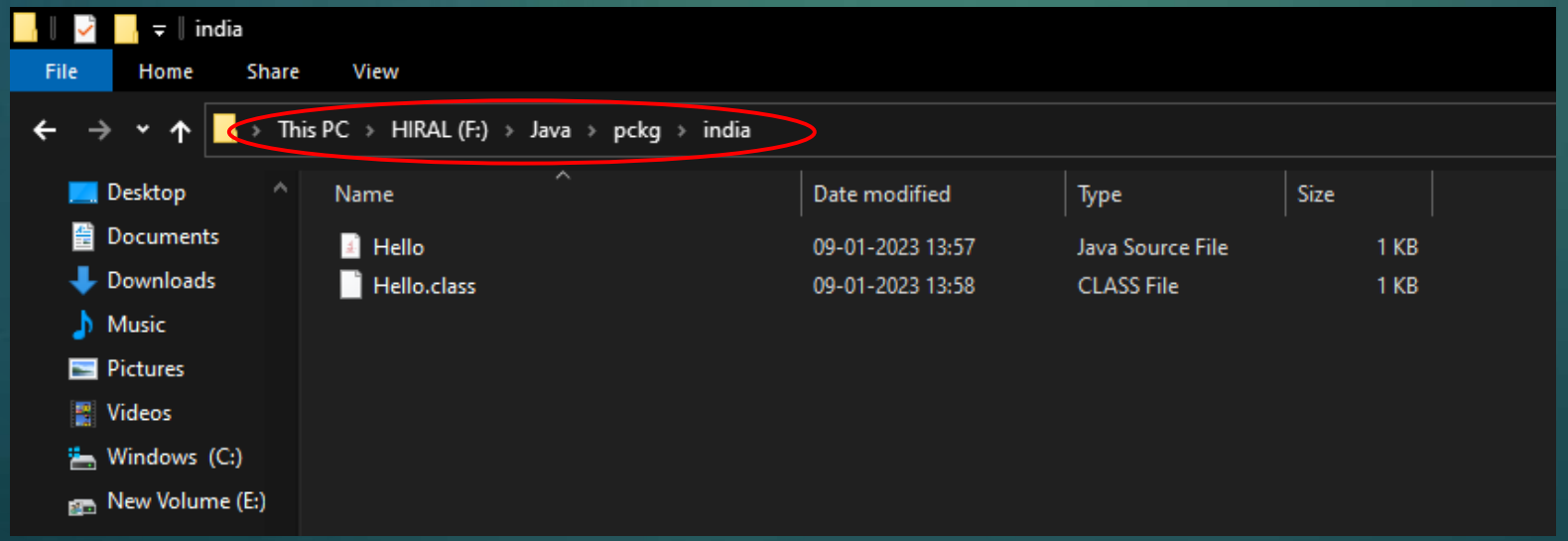
```
F:\Java\pckg\india>javac Hello.java

F:\Java\pckg\india>cd..

F:\Java\pckg>cd..

F:\Java>java pckg.india.Hello
Hello Indians !!!

F:\Java>
```



Importing a Package

- We have imported many built in packages like java.io, java.util,etc. as of now.
- To import self made packages we follow the following steps –
- Create the package as described in previous slides.
- This package is to be imported by another program outside the folder with package name
- That is we write two programs, one the package and another that imports the package. The package i to be stored in the folder with same name.

Importing a Package

The Package (Hello1.java)

```
package pckg.india;
public class Hello1
{
    public void display()
        {
        System.out.println("Hello Indians !!!");
        }
}
```

Importing Package (packimp.java)

```
import pckg.india.*;
class packimp
{
    public static void main(String args[])
    {
        Hello1 h=new Hello1();
        h.display();
    }
}
```

```
F:\Java>cd pckg

F:\Java\pckg>cd india

F:\Java\pckg\india>javac Hello1.java

F:\Java\pckg\india>cd..

F:\Java\pckg>cd..

F:\Java>javac packimp.java

F:\Java>java packimp
Hello Indians !!!
```

Exploring Important Java Packages

- Java.lang
- Java.io
- Java.util

Wrapper Classes

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Programming Practice Questions

- Design an interface Shape with methods calculateArea() and displayArea(). Implement this interface in two classes, Circle and Rectangle. The Circle class should have a constructor that takes the radius, and the Rectangle class should have a constructor that takes width and height. Write a program that creates instances of Circle and Rectangle, and calculates and prints their areas.
- Create an interface SortStrategy with a method void sort(int[] array). Implement this interface in classes, BubbleSort, QuickSort and MergeSort. Each class should provide its own implementation of the sorting algorithm. Write a program that uses different sorting strategies to sort an array of integers.
- Design an interface named TemperatureConverter with a method convertToCelsius() to convert temperatures to Celsius. Implement this converter in classes FahrenheitToCelsiusConverter and KelvinToCelsiusConverter to convert the temperature in different units to Celsius.
- Define an interface Shape with methods getArea and getPerimeter. Define another interface Volume with a method getVolume to calculate Volume of the shape. Implement classes Cube and Sphere that implements both Shape and Volume interfaces, allowing them to calculate their area, perimeter and Volume.

Programming Practice Questions

- Create a basic Java application with three packages: employee, department and company. In employee package, create a class Employee with fields name, id, and department, and methods to get and print these fields. In department package, create a class Department with fields departmentName and location, and methods to get and print these fields. Write a main method in a class located in company Package that creates instances of Employee and Department, sets their fields, and prints their details.
- Create a package hierarchy for a university system with the following structure: university, university.students, and university.courses. In university package, create a base class UniversityMember with common fields like name and id, and methods for displaying member information. In university.students, create classes Student and TeachingAdjunct that extends UniversityMember. Class Student includes additional fields for major and gpa and overrides method to display student details. Class TeachingAdjunct includes fields for no. of hours worked and stipend per hour and method to calculate total stipend alongwith displaying TeachingAdjunct Details by overriding method of base class. In university.courses, create a class Course with fields for courseName and courseCode, and methods to print course details. Write a class in university.main that demonstrates creating instances of Student and Course, and shows how they interact within the university system.