UNIT - II

PACKAGES & INTERFACES

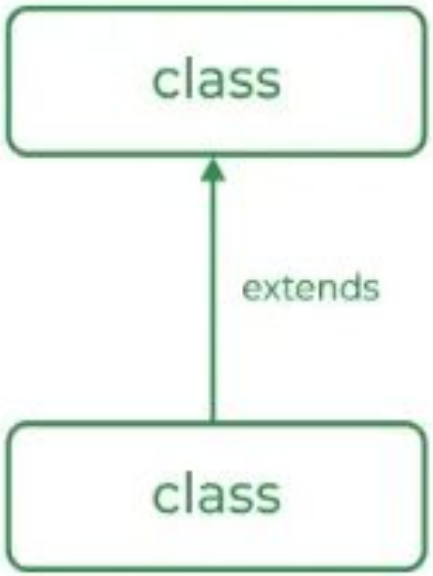
Interface

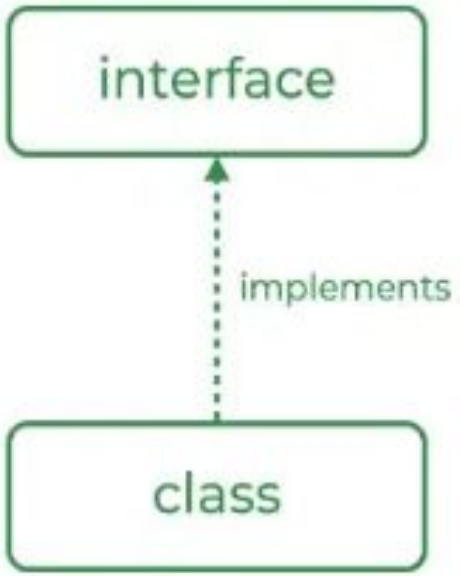
- An interface in the Java programming language serves as an **abstract type that outlines** the expected behavior of a class.
- It functions as a **blueprint,** defining a set of methods that any class **implementing the interface** must provide.
- Java interface may include static constants, which are unchangeable values associated with the interface.
  - **It is used to achieve Abstraction.**
  - **By interface, we can support the functionality of multiple inheritance**
  - **It can be used to achieve loose coupling**

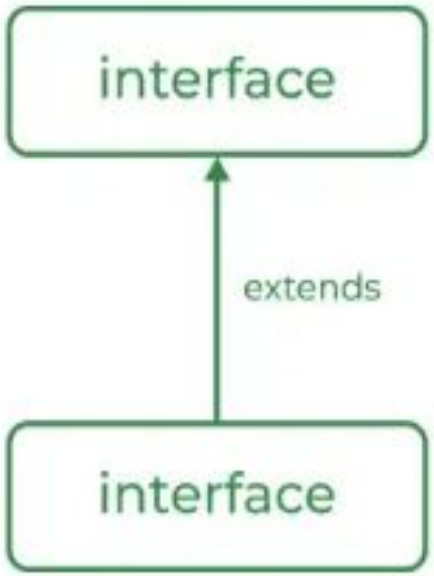
Inheritance in java with Interfaces

- In Java, **interfaces** can be used in inheritance to define a common **set of methods** that must be **implemented by derived classes.**
- When a **class implements an interface,** it must provide **an implementation** for all the methods defined in that interface.

Relationship Between Class and Interface







Sample Programs

MedicalInterfaceExample1

Defining Interfaces

```
public interface HealthStatus {
    public void takeMedicine(String medicineName);
    public void visitDoctor();
    public boolean isHealthy();
}

public class Person implements HealthStatus {
    private boolean healthy;

    public Person() {
        healthy = true;
    }

    public void takeMedicine(String medicineName) {
        // Take medicine
    }

    public void visitDoctor() {
        // Visit doctor
    }

    public boolean isHealthy() {
        return healthy;
    }
}
```

```
public class Pet implements HealthStatus {
    private boolean healthy;

    public Pet() {
        healthy = true;
    }

    public void takeMedicine(String medicineName) {
        // Take medicine
    }

    public void visitDoctor() {
        // Visit veterinarian
    }

    public boolean isHealthy() {
        return healthy;
    }
}
```

Sample Program

```
interface Medicine {
   void displayLabel();
}

class Tablet implements Medicine {
   private String name;
   private String manufacturer;
   private int strength;

   Tablet(String name, String manufacturer, int strength) {
      this.name = name;
      this.manufacturer = manufacturer;
      this.strength = strength;
   }

 public void displayLabel() {
      System.out.println("Name: " + name);
      System.out.println("Manufacturer: " + manufacturer);
      System.out.println("Strength: " + strength + " mg");
   }
}
```

```
class Syrup implements Medicine {
    private String name;
    private String manufacturer;
    private double volume;

    Syrup(String name, String manufacturer, double volume) {
        this.name = name;
        this.manufacturer = manufacturer;
        this.volume = volume;
    }

public void displayLabel() {
        System.out.println("Name: " + name);
        System.out.println("Manufacturer: " + manufacturer);
        System.out.println("Volume: " + volume + " ml");
    }
}

public class HealthcareProgram {
    public static void main(String[] args) {
        Medicine tablet = new Tablet("Paracetamol", "ABC Pharma", 500);
        Medicine syrup = new Syrup("Cough Syrup", "XYZ Pharma", 100);

        tablet.displayLabel();
        syrup.displayLabel();
    }
}
```

- Create an interface called "Sortable" that defines a method called "sort".
- Implement this interface in classes for an array of integers and an array of strings, and provide appropriate implementations for the "sort" method in each class.
- Create an interface named "Logger" with a method named "log" that takes a string message as input and logs the message to a file.
- Create two classes named "ConsoleLogger" and "FileLogger" that implement the "Logger" interface and provide their own implementation of the "log" method.
- The "ConsoleLogger" should log the message to the console and the "FileLogger" should log the message to a file.

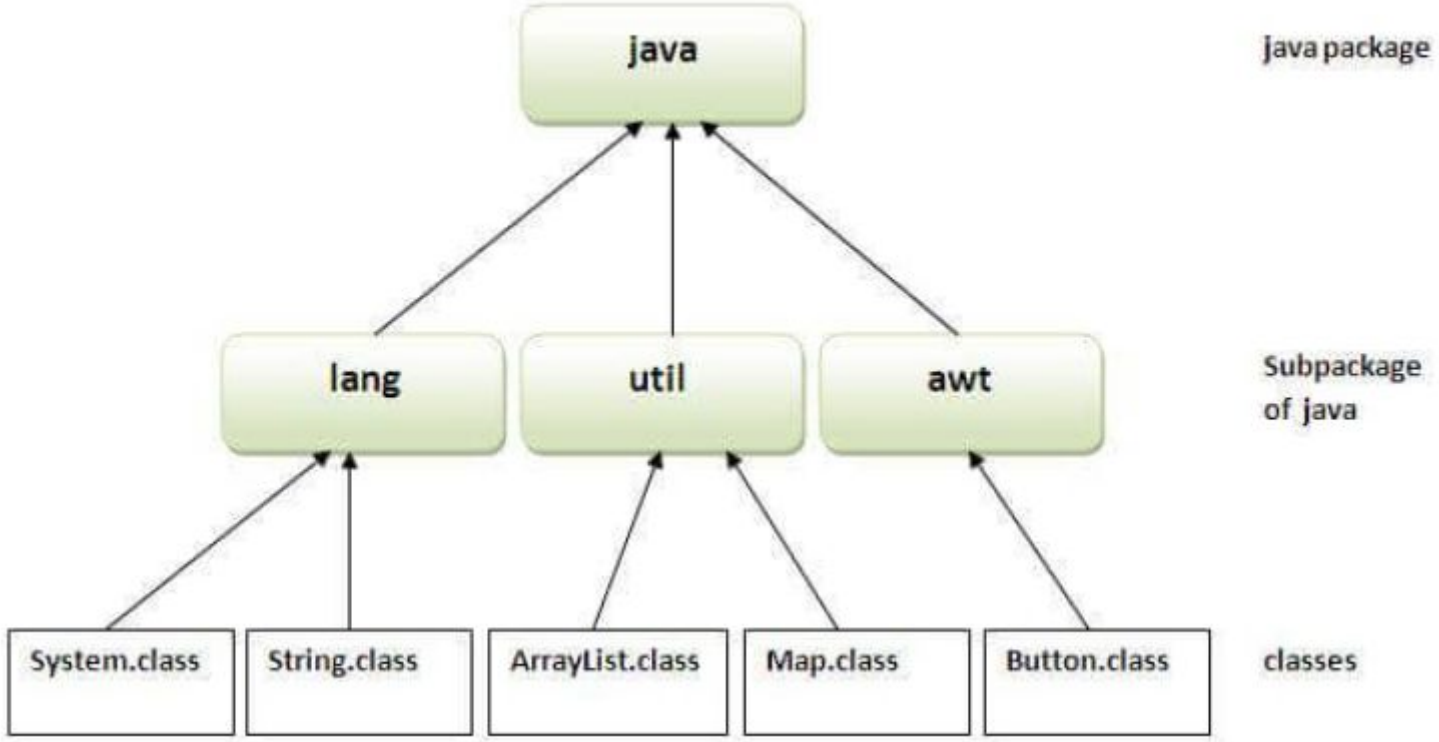
Extending Interfaces (Interface to Interface)

Program based on Interface to Interface

Creating Packages

- Packages are containers for classes.
- They are used to keep the class name space compartmentalized.
- For example, a package allows you to create a class named List, which you can store in your own package without concern that it will collide with some other class named List stored elsewhere.

Package


java
lang
util
awt
System.class
String.class
ArrayList.class
Map.class
Button.class
Java package
Subpackage of java
classes

Sample Program-1

```
package mypack;
public class Simple
{
 public static void main(String args[])
      {
    System.out.println("Package Concepts");
   }
}
```

```
C:\Java Programs>javac -d . Simple.java

C:\Java Programs>java mypack.Simple
Package Concepts
```

Sample Program-2

```
package pack1;
public class P1
{
  public void msg()
      {
      System.out.println("Package Programs");
      }
}
```

```
package mypack1;
import pack1.*;

class P2{
  public static void main(String args[]){
   P1 obj = new P1();
   obj.msg();
  }
}
```

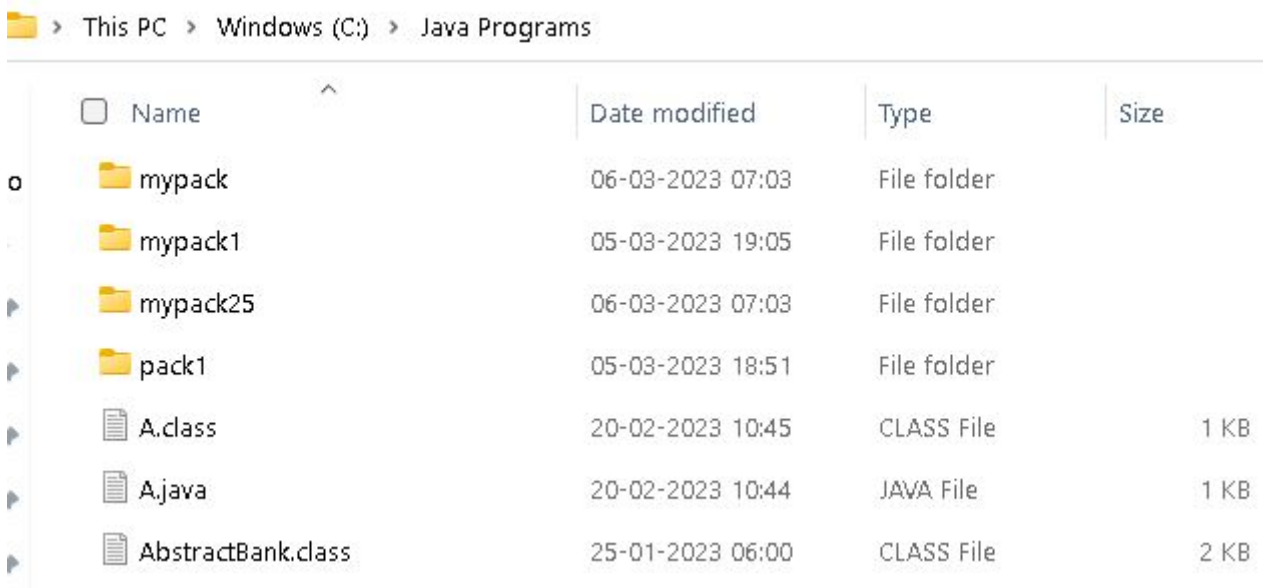
```
C:\Java Programs>javac -d . P2.java

C:\Java Programs>java mypack1.P2
Package Programs
```

CLASSPATH variable

Sample Program

```
package mypack25;

class Balance
{
String name;
double bal;

Balance(String n, double b)
        {
        name = n;
        bal = b;
        }

        void show()
        {
        if(bal<0)
                System.out.print(name + ": $" + bal);
        }
}

class AccountBalance
{
public static void main(String args[])
        {
        Balance current[] = new Balance[3];

        current[0] = new Balance("Smith", 120.5);
        current[1] = new Balance("David", 157.10);
        current[2] = new Balance("Tom", 95.5);

        }

}
```

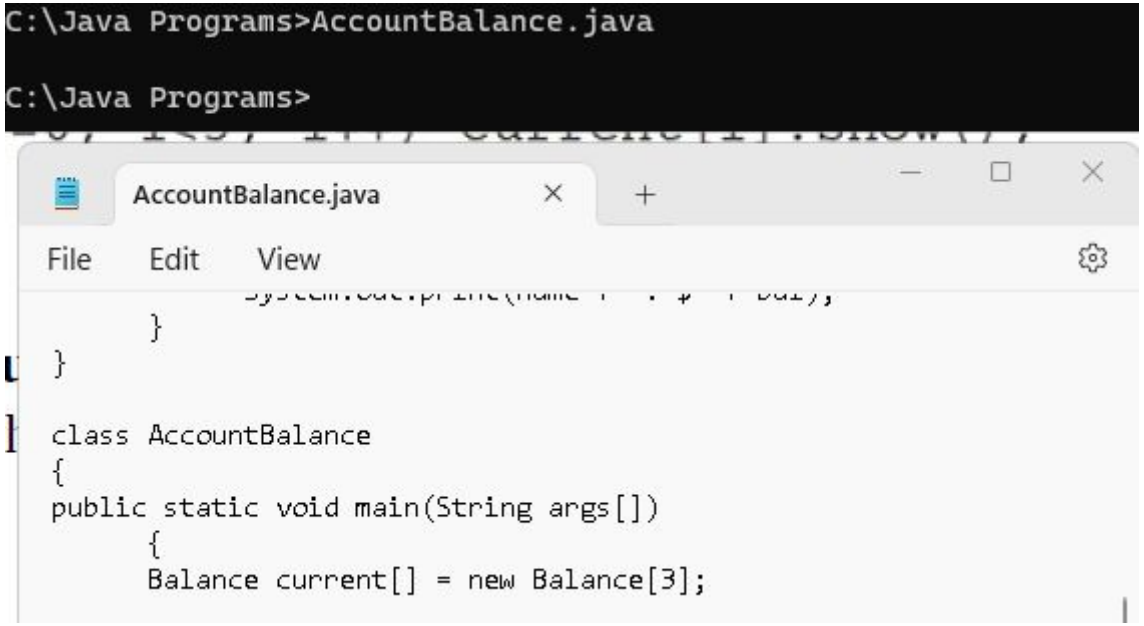
Created Path



Run the program

```
C:\Java Programs>javac -d . AccountBalance.java

C:\Java Programs>java mypack25.AccountBalance
Smith: $120.5David: $157.1Tom: $95.5
C:\Java Programs>
```

Fetch the file from Classpath



Access protection

```
package p1;

public class Protection
{
int n=1;
private int n_pri=5;
protected int n_pro =3;
public int n_pub=4;

public Protection()
        {
        System.out.println("Base Constructor");
        System.out.println("n = " +n);
        System.out.println("n-pri = " +n_pri);
        System.out.println("n_pro = " +n_pro);
        System.out.println("n_pub = " +n_pub);
        }


}
```

```
C:\Java Programs>javac -d . Protection.java

C:\Java Programs>java p1.protection
Error: Could not find or load main class p1.protection
Caused by: java.lang.NoClassDefFoundError: p1/Protection (wrong name: p1/protection)
```

```
package p1;

class Derived extends Protection
{
	Derived()
		{
		System.out.println("Derived Constructor");
		System.out.println("n = " +n);
		//System.out.println("n-pri = " +n_pri);
		System.out.println("n_pro = " +n_pro);
		System.out.println("n_pub = " +n_pub);
		}

}
```

```
C:\Java Programs>javac -d . Derived.java

C:\Java Programs>java p1.Derived
Error: Main method not found in class p1.Derived, please define the main method as:
   public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application
```

```
package p1;

class SamePackage
{
	SamePackage()
	{
	Protection P = new Protection();
	System.out.println("Same Package Constructor");
	System.out.println("n =" + P.n);
	//System.out.println("n-pri = " +P.n_pri);
	System.out.println("n_pro = " +P.n_pro);
	System.out.println("n_pub = " +P.n_pub);
	}

}
```

```
C:\Java Programs>javac -d . SamePackage.java

C:\Java Programs>java p1.SamePackage
Error: Main method not found in class p1.SamePackage, please define the main method as:
   public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application
```

```
package p2;

class Protection2 extends p1.Protection
{
	Protection2()
	{
	System.out.println("Derived Other Package Constructor");
	//System.out.println("n = " +n);
	//System.out.println("n-pri = " +n_pri);
	System.out.println("n_pro = " +n_pro);
	System.out.println("n_pub = " +n_pub);
	}

}
```

```
C:\Java Programs>javac -d . Protection2.java

C:\Java Programs>java p2.Protection2
Error: Main method not found in class p2.Protection2, please define the main method as:
   public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application
```

```
package p2;

class OtherPackage
{
	OtherPackage()
	{
	p1.Protection p = new p1.Protection();
	System.out.println("Other Package Constructor");
	//class or package only
	// System.out.println("n = " +p.n);

	//class only
	//System.out.println("n_pri = " + p.n_pri);

	//class, subclass or Package only
	//System.out.println("n_pro =" + p.n_pro);

	System.out.println("n_pub =" + p.n_pub);
	}

}
```

```
C:\Java Programs>javac -d . OtherPackage.java

C:\Java Programs>java p2.OtherPackage
Error: Main method not found in class p2.OtherPackage, please define the main method as:
   public static void main(String[] args)
or a JavaFX application class must extend javafx.application.Application
```

The test file for p1 is shown here:

```
package p1;

public class Demo5
{
	public static void main(String args[])
	{
	Protection ob1 = new Protection();
	Derived ob2 = new Derived();
	SamePackage ob3 = new SamePackage();
	}
}
```

Output

```
C:\Java Programs>javac -d . Demo5.java

C:\Java Programs>java p1.Demo5
Base Constructor
n = 1
n-pri = 5
n_pro = 3
n_pub = 4
Base Constructor
n = 1
n-pri = 5
n_pro = 3
n_pub = 4
Derived Constructor
n = 1
n_pro = 3
n_pub = 4
Base Constructor
n = 1
n-pri = 5
n_pro = 3
n_pub = 4
Same Package Constructor
n =1
n_pro = 3
n_pub = 4
```

The test file for p2 is shown here:

```
package p2;

public class Demo6
{
	public static void main(String args[])
	{
	Protection2  ob1 = new Protection2();
	OtherPackage ob2 = new OtherPackage();
	}
}
```

```
C:\Java Programs>javac -d . Demo6.java

C:\Java Programs>java p2.Demo6
Base Constructor
n = 1
n-pri = 5
n_pro = 3
n_pub = 4
Derived Other Package Constructor
n_pro = 3
n_pub = 4
Base Constructor
n = 1
n-pri = 5
n_pro = 3
n_pub = 4
Other Package Constructor
n_pub =4
```

Importing Packages

- It could become tedious to type in the long dot-separated package path name for every class you want to use.
- For this reason, Java includes the import statement to bring certain classes, or entire packages, into visibility.
- In a Java source file, import statements occur immediately following the package statement (if it exists) and before any class definitions. This is the general form of the import statement:

import *pkg1* [.*pkg2*].(*classname* | *);

- There is no practical limit on the depth of a package hierarchy, except that imposed by the
- file system.
- Finally, you specify either an explicit classname or a star (*), which indicates that the Java compiler should import the entire package.
- This code fragment shows both forms in use:
  - **import java.util.Date;**
  - **import java.io.*;**

Sample Example

```
package myPackage26;

public class Package11 {
    public void display() {
        System.out.println("Hello from my custom package!");
    }
}
```

```
package mypack27;
import myPackage26.*;

public class PackageDemo1 {
    public static void main(String[] args) {
        Package11 obj1 = new Package11();
        obj1.display();
    }
}
```

```
C:\Java Programs>javac -d . Package11.java

C:\Java Programs>javac -d . PackageDemo1.java

C:\Java Programs>java mypack27.PackageDemo1
Hello from my custom package!
```

Interfaces in a Package

- Using the keyword interface, you can fully abstract a class' interface from its implementation.
- That is, using interface, you can specify what a class must do, but not how it does it.
- Interfaces are syntactically similar to classes, but they lack instance variables, and, as a general rule, their methods are declared without any body.