Course code : **CSE2005**

Course title : **Object Oriented Programming**

Abstract class, Interfaces

Objectives

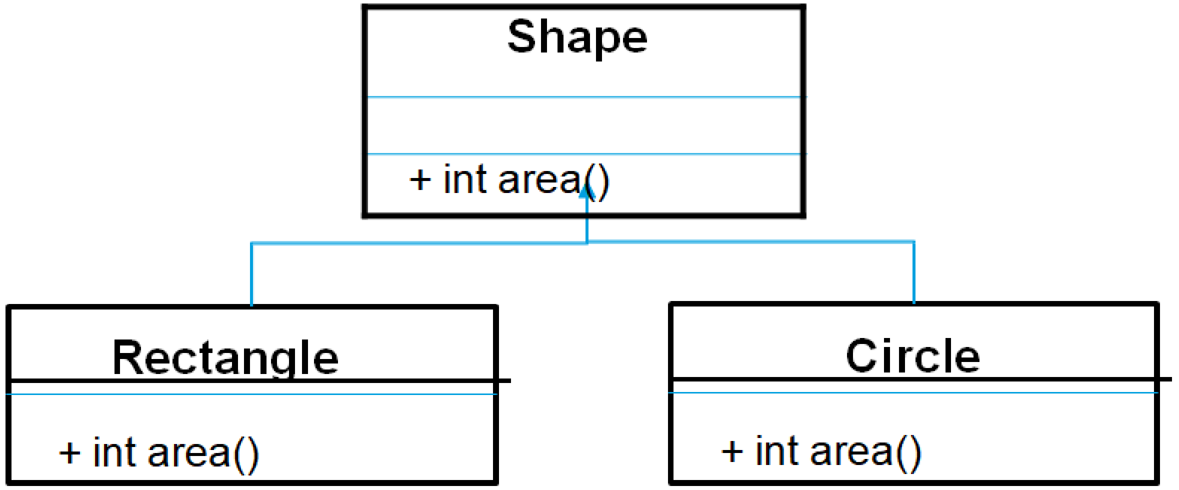
This session will give the knowledge about

- Abstract classes
- Introduction to interfaces
- Applying Interfaces

Abstract class

Abstract Classes

Let us see the below example of Shape class extended by Rectangle and Circle.



In the above example area() for Shape being more generic we cannot define it. At the level of rectangle or Circle we can give the formula for area.

Abstract Classes

Often, you would want to define a superclass that declares the structure of a given abstraction without providing the implementation of every method

The objective is to:

- Create a superclass that only defines a generalized form that will be shared by all of its subclasses
- Leaving it to each subclass to provide for its own specific implementations

Abstract Classes

- Such a class determines the nature of the methods that the subclasses must implement
- Such a superclass is unable to create a meaningful implementation for a method or methods

The importance of abstract classes:

- they define a generalized form (possibly some generalized methods with no implementations) that will be shared by all of its subclasses, so that each subclass can provide specific implementations of such methods.

Abstract Classes

The class Shape in the previous example is such a superclass.

- Shape is a pure geometrical abstraction
- You have only kinds of figures like Rectangle, Triangle etc. Which actually are subclasses of class Shape
- The class Shape has no implementation for the area() method, as there is no way to determine the area of a Shape
- The Shape class is therefore a partially defined class with no implementation for the area( ) method
- The definition of area() is simply a placeholder

Abstract Classes

- Abstract method – It’s a method declaration with no definition
- A mechanism which shall ensure that a subclass must compulsorily override such methods.
- Abstract method in a superclass has to be overridden by all its subclasses.
- The subclasses cannot make use of the abstract method that they inherit directly (without overriding these methods).
- These methods are sometimes referred to as subclasses responsibility as they have no implementation specified in the superclass

Abstract Classes

To use an abstract method, use this general form:

access specifier abstract return-type name(parameter-list);

Abstract methods do not have a body

Abstract methods are therefore characterized by the lack of the opening and closing braces that is customary for any other normal method

This is a crucial benchmark for identifying an abstract class area method of Shape class made Abstract. public abstract int area();

Abstract Classes

Any class that contains one or more abstract methods must also be declared abstract

- It is perfectly acceptable for an abstract class to implement a concrete method
- You cannot create objects of an abstract class
- That is, an abstract class cannot be instantiated with the new keyword
- Any subclass of an abstract class must either implement all of the abstract methods in the superclass, or be itself declared abstract.

Improved Version of the Shape Class Hierarchy

```
package vit.demo;
abstract class Shape{
   int height;
   int width;
   Shape(int x, int y){
            height=x;
            width=y;
   }
   abstract int area();
}
```

```
class Rectangle extends Shape{
   Rectangle(int x, int y){
            super(x,y);
   }
   int area(){
            return height*width;
   }
}
```

Improved Version of the Shape Class Hierarchy

```
class Triangle extends
   Shape{
   Triangle(int x, int y){
         super(x,y);
   }
   int area(){
         return height*width/2;
   }
}
```

```
class Main {
   public static void main(String ar[]) {
         Rectangle rectangle=new Rectangle(2, 3);
         Triangle triangle=new Triangle(5, 10);

         Shape s; //reference variable
         s=rectangle;
         System.out.println(s.area());
         s=triangle;
         System.out.println(s.area());
   }
}
```

Quiz

```
package vit.demo;
class Base{
   int a=10;
   abstract void display();
}
class Main extends Base {
   public static void main(String ar[]) {
            Main main=new Main();
   }
}
```

Quiz

```
package vit.demo;
abstract class Base{
   int a=10;
   void display() {
        System.out.println(a);
   }
}
class Main extends Base {
   public static void main(String ar[]) {
            Base base=new Base();
   }
}
```

Quiz

```
package vit.demo;
class abstract Base{
   int a=10;
   void display() {
        System.out.println(a);
   }
}
class Main extends Base {
   public static void main() {
             Base base=new Base();
   }
}
```

Interfaces

What is an Interface?

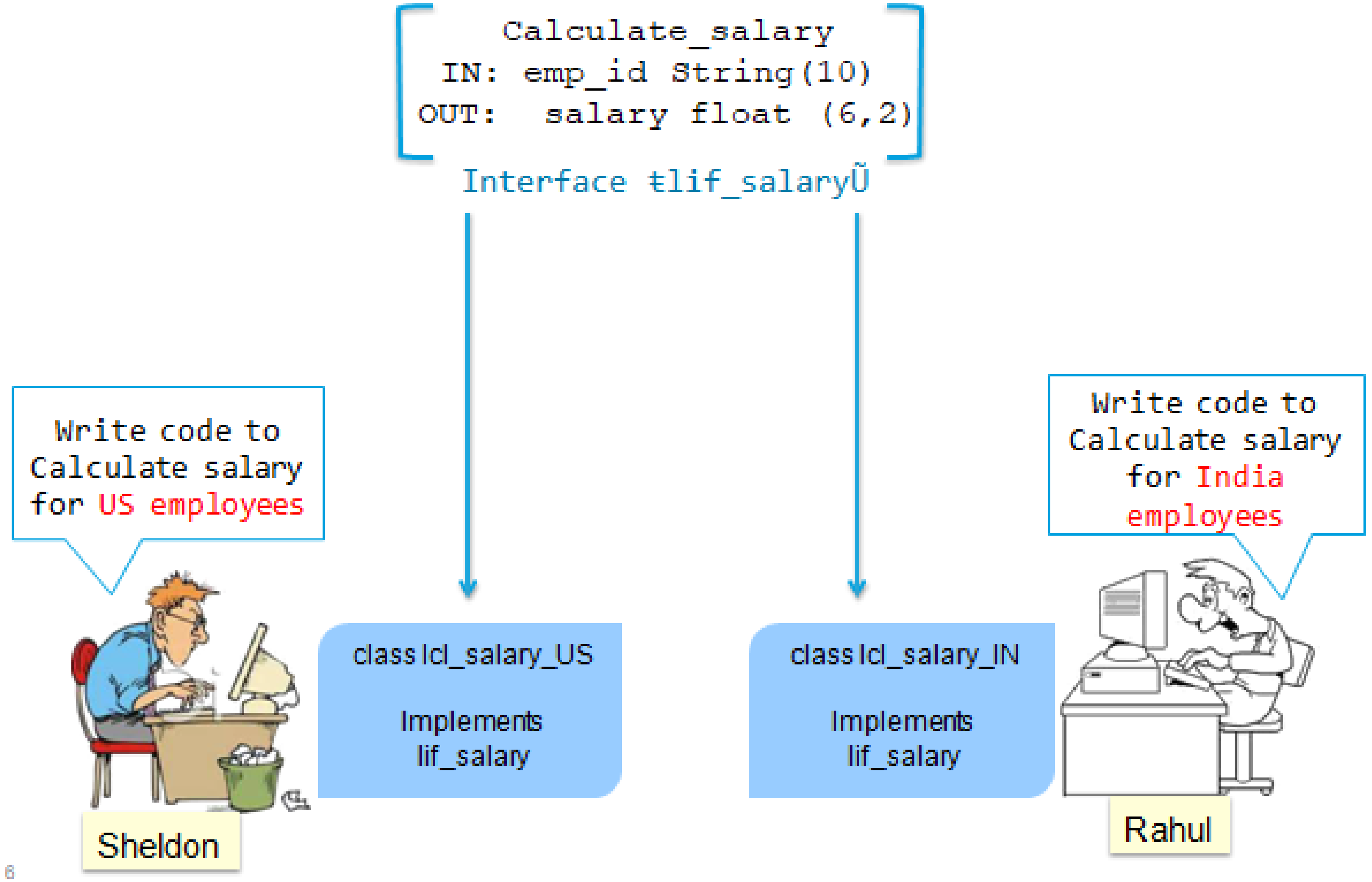
An interface is a named collection of method declarations (without implementations)

- An interface can also include constant declarations
- An interface is syntactically similar to an abstract class
- An interface is a collection of abstract methods and final variables
- A class implements an interface using the implements clause

What is an Interface? (Contd.).

- An interface defines a protocol of behavior
- An interface lays the specification of what a class is supposed to do
- How the behavior is implemented is the responsibility of each implementing class
- Any class that implements an interface adheres to the protocol defined by the interface, and in the process, implements the specification laid down by the interface

Interface: Example



```
    Calculate_salary
IN:  emp_id String(10)
OUT:   salary float (6,2)
```

Interface ±lif_salaryÛ
Write code to Calculate salary for US employees
class lcl_salary_US Implements lif_salary
class lcl_salary_IN Implements lif_salary
Write code to Calculate salary for India employees
Sheldon
Rahul

Why interfaces are required ?

- Interfaces allow you to implement common behaviors in different classes that are not related to each other
- Interfaces are used to describe behaviors that are not specific to any particular kind of object, but common to several kind of objects
- Defining an interface has the advantage that an interface definition stands apart from any class or class hierarchy
- This makes it possible for any number of independent classes to implement the interface

Why interfaces are required ?

- Thus, an interface is a means of specifying a consistent specification, the implementation of which can be different across many independent and unrelated classes to suit the respective needs of such classes
- Java does not support multiple inheritance
- This is a constraint in class design, as a class cannot achieve the functionality of two or more classes at a time
- Interfaces help us make up for this loss as a class can implement more than one interface at a time

Interface members

- All the methods that are declared within an interface are always, by default, public and abstract
- Any variable declared within an interface is always, by default, public static and final

What will you choose..?

What is the behavior which is common among the entities depicted in the pictures above?

Yes..You are right. All of them can fly.

Requirement : You have to develop 3 classes, Bird, Superman and Aircraft with the condition that all these classes must have a method called fly().

What is the mechanism, using which you can ensure that the method fly() is implemented in all these classes? An Abstract class or An Interface?

Defining an Interface

An interface is syntactically similar to a class

It’s general form is:

```
public interface Person{
        String name="myname";
        int age=20;
        void eat();
        void drink(String drinkName);
}
```

Implementing Interfaces

A class implements an interface

A class can implement more than one interfaces by giving a comma-separated list of interfaces

```
public interface Person{
        String name="myname";
        int age=20;
        void eat();
        void drink(String drinkName);
}
```

```
class Student implements Person{
        @Override
        public void eat() {
            System.out.println("student
        eating");
        }
        @Override
        public void drink(String drinkName) {
            System.out.println("student
        drinking "+drinkName);
        }
}
```

Quiz

```
package vit.demo;
interface Animal{
        private int age=300;
        protected void display();
}
class Main implements Animal {
        protected void display(){
             System.out.println(age);
        }
        public static void main(String ar[]) {
        }
}
```

Quiz

```
package vit.demo;
interface Animal{
        static int age=300;
        static void display();
}
class Main implements Animal {
        protected void display(){
              System.out.println(age);
        }
        public static void main(String ar[]) {
        }
}
```

Applying Interfaces

- Software development is a process where constant changes are likely to happen
- There can be changes in requirement, changes in design, changes in implementation
- Interfaces support change
- Programming through interfaces helps create software solutions that are reusable, extensible, and maintainable

Implementing Interfaces

```
package vit.demo;
interface University{
   int hall=301;
   void use(String dept);
}
class Cse implements University{
   @Override
   public void use(String dept) {
     System.out.println(dept+" using "+hall);
   }
}
```

```
class Ece implements University{
     public void use(String dept) {
          System.out.println(dept+" using "+hall);
     }
}
class Main {
     public static void main(String ar[]) {
             Cse cse = new Cse();
             cse.use("bcd");
             Ece ece = new Ece();
             ece.use("bec");
     }
}
```

Interfaces References

- When you create objects, you refer them through the class references. For example :
    - ClassOne c1 = new ClassOne(); /* Here, c1 refers to the object of the class classOne. */
- You can also make the interface variable refer to the objects of the class that implements the interface
- The exact method will be invoked at run time
- It helps us achieve run-time polymorphism

Interfaces References

```
package vit.demo;
interface University{
   int hall=301;
   void use(String dept);
}
class Cse implements University{
   @Override
   public void use(String dept) {
     System.out.println(dept+" using "+hall);
   }
}
```

```
class Ece implements University{
     public void use(String dept) {
          System.out.println(dept+" using "+hall);
     }
}
class Main {
     public static void main(String ar[]) {
             University obj = new Cse();
             obj.use("bcd");
             obj = new Ece();
             obj.use("bec");
     }
}
```

Extending Interfaces

- Just as classes can be inherited, interfaces can also be inherited
- One interface can extend one or more interfaces using the keyword extends
- When you implement an interface that extends another interface, you should provide implementation for all the methods declared within the interface hierarchy

Extending Interfaces

```
package vit.demo;
interface Person{
   void personalDetails();
}
interface Employee extends Person{
   void jobDetails();
}
```

```
class Report implements Employee {
        @Override
        public void personalDetails() {
              System.out.println("my name");
        }
        @Override
        public void jobDetails() {
              System.out.println("manager");
        }
}
```

Multiple Inheritance

```
package vit.demo;
interface Person{
   void personalDetails();
}
class Employee {
   void jobDetails(){
            System.out.println("manager");
   }
}
```

```
class Report extends Employee implements Person
{
       public void personalDetails() {
            System.out.println("my name");
       }
}
class Main {
       public static void main(String ar[]) {
            Report report = new Report();
            report.personalDetails();
            report.jobDetails();
       }
}
```

Marker Interface

- An Interface with no method declared in it, is known as Marker Interface
- Marker Interface is provided as a handle by java interpreter to mark a class, so that it can provide special behavior to it at runtime
- Examples of Marker Interfaces :
  - java.lang.Cloneable
  - java.io.Serializable
  - java.rmi.Remote

Marker Interface

- An Interface with no method declared in it, is known as Marker Interface
- Marker Interface is provided as a handle by java interpreter to mark a class, so that it can provide special behavior to it at runtime
- Examples of Marker Interfaces :
  - java.lang.Cloneable
  - java.io.Serializable
  - java.rmi.Remote

Quiz

Will the following code compile successfully ?

```
interface Base{
   int id=101;
   void personalDetails();
}
class Derived extends Base {
   void personalDetails(){
            System.out.println(id);
   }
}
```

Quiz

Will the following code compile successfully ?

```
interface Base{
  int id=101;
  void personalDetails();
}
interface Derived implements Base {
  void personalDetails();
}
```

Quiz

```
interface Base{
   int id=101;
   void personalDetails();
}
interface Derived extends Base {
   void jobDetails();
}
class Report implements Base {
   public void personalDetails() {
            System.out.println("my name");
   }
}
```

Abstract class VS Interface

Abstract class	Interface
Abstract classes can have non-final non-static variables.	Variables declared within an interface are always static and final.
Abstract Classes can have abstract methods as well as concrete methods.	Interfaces can have only method declarations(abstract methods). You cannot define a concrete method.
You can declare any member of an abstract class as private, default, protected or public. Members can also be static.	Interface members are by default public. You cannot have private or protected members. Interface methods cannot be static.
Abstract class is extended by another class using “extends” keyword.	An interface is “implemented” by a java class using “implements” keyword .

Abstract class VS Interface

Abstract class	Interface
An abstract class can extend another class and it can implement one or more interfaces.	An interface can extend one or more interfaces but cannot extend a class. It cannot implement an interface.
An abstract class can have constructors defined within it.	You cannot define constructors within an interface.
An abstract class cannot be instantiated using “new” Keyword	An interface cannot be instantiated.
You can execute(invoke) an abstract class, provided it has public static void main(String[] args) method declared within it.	You cannot execute an interface

Summary

We have discussed about

- Abstract classes
- Introduction to interfaces
- Applying Interfaces