Sukkur Institute of Business Administration University
Department of Computer Science
BS – II (CS/SE/AI) Spring 2024

Object Oriented Programming

Lab # 09: To become familiar with Abstract Classes and Interfaces

Instructor: Engr. Zainab Umair Kamangar

Lab Report Rubrics (Add the points in each column, then add across the bottom row to find the total score)					**Total Marks**
S.No	**Criterion**	**0.5**	**0.25**	**0.125**	
1	Accuracy	☐ Desired output	☐ Minor mistakes	☐ Critical mistakes	
2	Timing	☐ Submitted within the given time	☐ 1 day late	☐ More than 1 day late	

Submission Profile

Name: Submission date (dd/mm/yy):
Enrollment ID: Receiving authority name and signature:
Comments:

_____
Instructor Signature

Note: Submit this lab hand-out in the next lab with attached solved activities and exercises

Objectives

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

- Abstract classes
- Interfaces

Abstraction in Java

Abstraction is a concept of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

There are two ways to achieve abstraction in java

1. Abstract class
2. Interface

Abstract Class

In some situations, we don't want programmers to instantiate (create an instance of) a particular class, but rather we choose to create the class only so that it can serve as a generic or default superclass of other classes.

For example, we might want to have a class Shape to serve as superclass for a number of "real" subclasses, like Parallelogram or Rhombus: we want to have Parallelogram and Rhombus objects around, but not generic "Shape" objects.

An abstract class is a class that is declared 'abstract'. It can, but does not have to, include abstract methods. Abstract classes cannot be instantiated but they can be subclassed using the extends keyword. An abstract method is a method that is declared without an implementation and it requires the abstract keyword.

If a class includes abstract methods, then the class itself must be declared as abstract.

When a child class extends an abstract class, it must either:

• Provide implementations for all abstract methods from its parent class, or
• Also be abstract
• Child classes can reference the constructor of abstract class by using super()

```
abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end
user
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
class Circle1 extends Shape{
void draw(){System.out.println("drawing circle");}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle1();
//In a real scenario, object is provided through method, e.g., getShape() met
hod
s.draw();
}
}
```

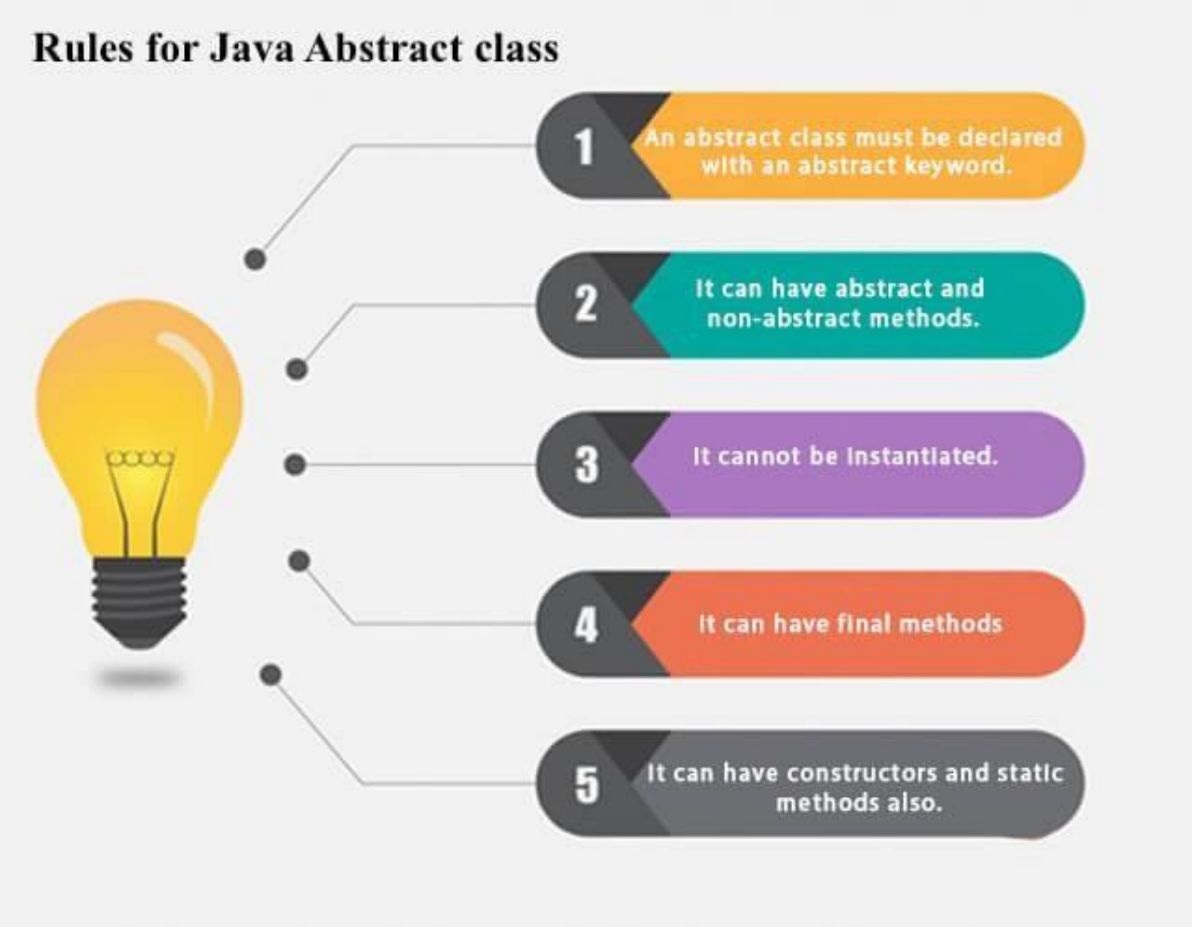
In Java, the following some *important observations* about abstract classes are as follows:

1. An instance of an abstract class can not be created.
2. Constructors are allowed.
3. We can have an abstract class without any abstract method.
4. There can be a final method in abstract class but any abstract method in class(abstract class) can not be declared as final or in simpler terms final method can not be abstract itself as it will yield an error: "Illegal combination of modifiers: abstract and final"
5. We can define static methods in an abstract class
6. We can use the abstract keyword for declaring *top-level classes (Outer class) as well as inner classes* as abstract
7. If a class contains at least one abstract method then compulsory should declare a class as abstract

If the Child class is unable to provide implementation to all abstract methods of the Parent class then we should declare that Child class as abstract so that the next level Child class should provide implementation to the remaining abstract method

Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.



Q1. Why do we use abstract?

One key reason we use abstract concepts is to simplify complexity. Imagine trying to explain the entire universe with every single atom and star! Abstracts let us zoom out, grab the main ideas like gravity and energy, and make sense of it all without getting lost in the details.

Here are some other reasons why we use abstract in Java:

Abstract classes are used to define a generic template for other classes to follow. They define a set of rules and guidelines that their subclasses must follow. By providing an abstract class, we can ensure that the classes that extend it have a consistent structure and behavior. This makes the code more organized and easier to maintain.

Q2. What is the Difference between Encapsulation and Data Abstraction?

Here are some key difference b/w encapsulation and abstration:

Encapsulation	**Abstraction**
Encapsulation is data hiding(information hiding)	Abstraction is detailed hiding(implementation hiding).
Encapsulation groups together data and methods that act upon the data	Data Abstraction deal with exposing the interface to the user and hiding the details of implementation
Encapsulated classes are Java classes that follow data hiding and abstraction	Implementation of abstraction is done using abstract classes and interface
Encapsulation is a procedure that takes place at the implementation level	Abstraction is a design-level process

Q3. What is a real-life example of data abstraction?

Television remote control is an excellent real-life example of abstraction. It simplifies the interaction with a TV by hiding the complexity behind simple buttons and symbols, making it easy without needing to understand the technical details of how the TV functions.

Practice Tasks

1. Write a Java program to create an abstract class Animal with an abstract method called sound(). Create subclasses Lion and Tiger that extend the Animal class and implement the sound() method to make a specific sound for each animal.
   Click me to see the solution

2. Write a Java program to create an abstract class Shape with abstract methods calculateArea() and calculatePerimeter(). Create subclasses Circle and Triangle that extend the Shape class and implement the respective methods to calculate the area and perimeter of each shape.
   Click me to see the solution

3. Write a Java program to create an abstract class BankAccount with abstract methods deposit() and withdraw(). Create subclasses: SavingsAccount and CurrentAccount that extend the BankAccount class and implement the respective methods to handle deposits and withdrawals for each account type.
   Click me to see the solution

4. Write a Java program to create an abstract class Animal with abstract methods eat() and sleep(). Create subclasses Lion, Tiger, and Deer that extend the Animal class and implement the eat() and sleep() methods differently based on their specific behavior.
   Click me to see the solution

5. Write a Java program to create an abstract class Employee with abstract methods calculateSalary() and displayInfo(). Create subclasses Manager and Programmer that extend the Employee class and implement the respective methods to calculate salary and display information for each role.
   Click me to see the solution

6. Write a Java program to create an abstract class Shape3D with abstract methods calculateVolume() and calculateSurfaceArea(). Create subclasses Sphere and Cube that extend the Shape3D class and implement the respective methods to calculate the volume and surface area of each shape.
   Click me to see the solution

7. Write a Java program to create an abstract class Vehicle with abstract methods startEngine() and stopEngine(). Create subclasses Car and Motorcycle that extend the Vehicle class and implement the respective methods to start and stop the engines for each vehicle type.
   Click me to see the solution

8. Write a Java program to create an abstract class Person with abstract methods eat() and exercise(). Create subclasses Athlete and LazyPerson that extend the Person class and implement the respective methods to describe how each person eats and exercises.
   Click me to see the solution

9. Write a Java program to create an abstract class Instrument with abstract methods play() and tune(). Create subclasses for Glockenspiel and Violin that extend the Instrument class and implement the respective methods to play and tune each instrument.
   Click me to see the solution

10. Write a Java program to create an abstract class Shape2D with abstract methods draw() and resize(). Create subclasses Rectangle and Circle that extend the Shape2D class and implement the respective methods to draw and resize each shape.
Click me to see the solution

11. Write a Java program to create an abstract class Bird with abstract methods fly() and makeSound(). Create subclasses Eagle and Hawk that extend the Bird class and implement the respective methods to describe how each bird flies and makes a sound.
Click me to see the solution

12. Write a Java program to create an abstract class GeometricShape with abstract methods area() and perimeter(). Create subclasses Triangle and Square that extend the GeometricShape class and implement the respective methods to calculate the area and perimeter of each shape.
Click me to see the solution

Helpful Sites:

https://javaconceptoftheday.com/java-practice-coding-questions-on-abstract-classes/
https://www.codesdope.com/practice/java-abstract-class/

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.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Java Interface also represents the **IS-A** relationship. It cannot be instantiated just like the abstract class.
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:

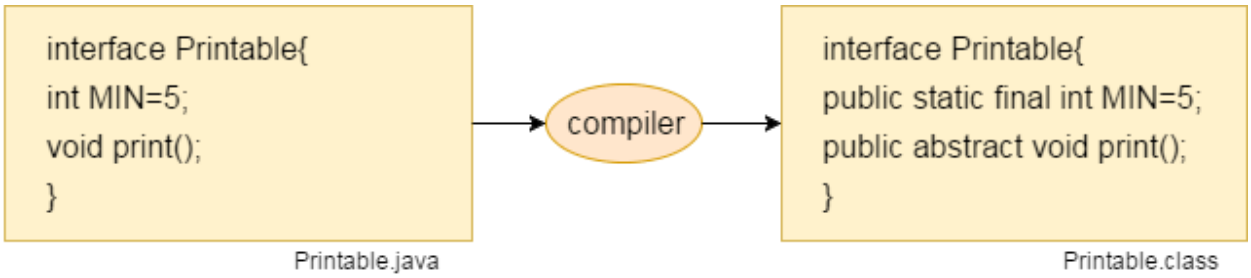
- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.

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:

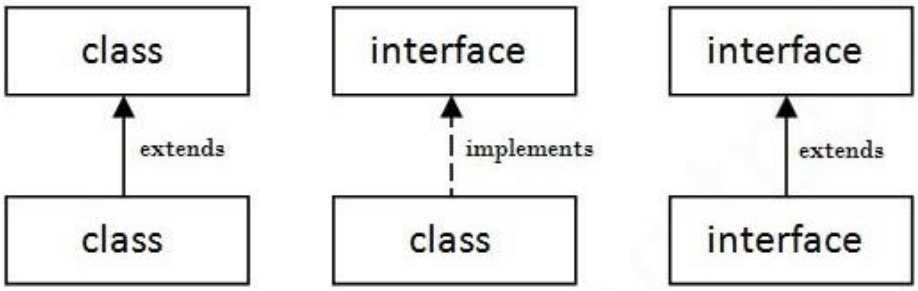
```
interface <interface_name>{

    // 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

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



Example:

Let's consider the example of vehicles like bicycle, car, bike……., they have common functionalities. So we make an interface and put all these common functionalities. And lets Bicycle, Bike, car ….etc implement all these functionalities in their own class in their own way.

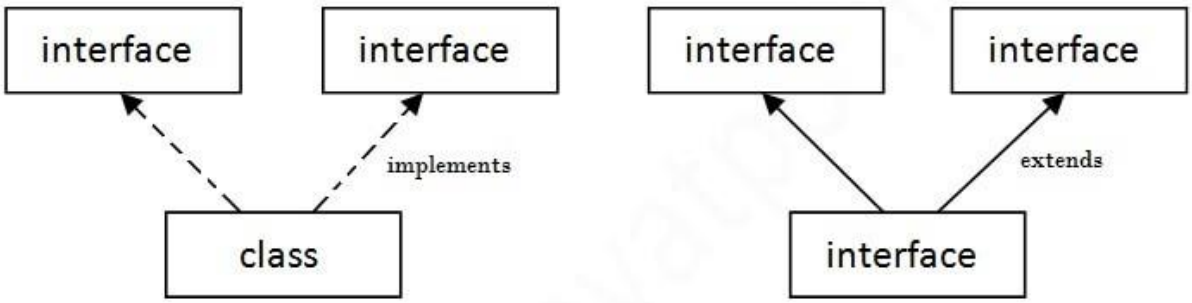
```
import java.io.*;
interface Vehicle {
    // all are the abstract methods.
    void changeGear(int a);
    void speedUp(int a);
    void applyBrakes(int a);
}
```

```
class Bicycle implements Vehicle{
    int speed;
    int gear;

     // to change gear
    public void changeGear(int newGear){

        gear = newGear;
    }

    // to increase speed
    public void speedUp(int increment){

        speed = speed + increment;
    }

    // to decrease speed
    public void applyBrakes(int decrement){

        speed = speed - decrement;
    }

    public void printStates() {
         System.out.println("speed: " + speed
             + " gear: " + gear);
    }
}

class Bike implements Vehicle {

    int speed;
    int gear;

    // to change gear
    public void changeGear(int newGear){
        gear = newGear;
    }

    // to increase speed
    public void speedUp(int increment){

        speed = speed + increment;
    }

    // to decrease speed
    public void applyBrakes(int decrement){

        speed = speed - decrement;
    }

    public void printStates() {
         System.out.println("speed: " + speed
            + " gear: " + gear);
```

```
    }

}

class GFG {

    public static void main (String[] args) {
        // creating an instance of Bicycle
        // doing some operations
        Bicycle bicycle = new Bicycle();

        bicycle.changeGear(2);
        bicycle.speedUp(3);
        bicycle.applyBrakes(1);


        System.out.println("Bicycle present state :");
        bicycle.printStates();
        // creating instance of the bike.

        Bike bike = new Bike();
        bike.changeGear(1);
        bike.speedUp(4);
        bike.applyBrakes(3);
        System.out.println("Bike present state :");
        bike.printStates();
    }
 }
```

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

As we have explained in the inheritance chapter, multiple inheritance is not supported in the case of class because of ambiguity. However, it is supported in case of an interface because there is no ambiguity. It is because its implementation is provided by the implementation class.

```
//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

An interface i.e. declared within another interface or class is known as nested interface. The nested interfaces are used to group related interfaces so that they can be easy to maintain. The nested interface must be referred by the outer interface or class. It can't be accessed directly.

- Nested interface must be public if it is declared inside the interface but it can have any access modifier if declared within the class.
- Nested interfaces are declared static implicitly.

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();
    }
 }
```

Class inside the interface

If we define a class inside the interface, java compiler creates a static nested class.

Syntax:

```
interface M{
  class A{}
}
```

What is the Difference between Abstract Classes and Interfaces in Java?

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below:

Abstract class	***Interface***
*Abstract class can **have abstract and non-abstract** methods.*	*Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.*
*Abstract class **doesn't support multiple inheritance**.*	*Interface **supports multiple inheritance**.*
*Abstract class **can have final, non-final, static and non-static variables**.*	*Interface has **only static and final variables**.*
*Abstract class **can provide the implementation of interface**.*	*Interface **can't provide the implementation of abstract class**.*
*The **abstract keyword** is used to declare abstract class.*	*The **interface keyword** is used to declare interface.*
*An **abstract class** can extend another Java class and implement multiple Java interfaces.*	*An **interface** can extendanother Java interface only.*
*An **abstract class** can be extended using keyword "extends".*	*An **interface** can be implemented using keyword "implements".*
*A Java **abstract class** can have class members like private, protected, etc.*	*Members of a Java interface are public by default.*
Example: *public abstract class Shape{* *public abstract void draw();* *}*	*Example:* *public interface Drawable{* *void draw();* *}*

Exercises

Question: 1 (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 1.
2. Is the following interface valid?

```
public interface Marker {
}
```

Question: 2 (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: 3 (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: 4 (Abstract Class)

We have to calculate the percentage of marks obtained in three subjects (each out of 100) by student A and in four subjects (each out of 100) by student B. Create an abstract class 'Marks' with an abstract method 'getPercentage'. It is inherited by two other classes 'A' and 'B' each having a method with the same name which returns the percentage of the students. The constructor of student A takes the marks in three subjects as its parameters and the marks in four subjects as its parameters for student B. Create an object for each of the two classes and print the percentage of marks for both the students.

Question: 5 (Abstract class)

We have to calculate the area of a rectangle, a square and a circle. Create an abstract class 'Shape' with three abstract methods namely 'RectangleArea' taking two parameters, 'SquareArea' and 'CircleArea' taking one parameter each. The parameters of 'RectangleArea' are its length and breadth, that of 'SquareArea' is its side and that of 'CircleArea' is its radius. Now create another class 'Area' containing all the three methods 'RectangleArea', 'SquareArea' and 'CircleArea' for printing the area of rectangle, square and circle respectively. Create an object of class 'Area' and call all the three methods.