OBject oriented programming with java

Lab manual

Aymen khan

Aymkhan498@gmail.com

**Course Title:** Object Oriented Programming

**Course Code:**

**Course Outcomes:**

At the end of the course the student should be able to:

1. Apply Object Oriented Programming concepts to solve a given problem.
2. Apply design patterns to design a solution for a given problem.
3. Apply inheritance, polymorphism and exception handling mechanism to implement reusable, robust java programs.
4. Implement user interface java programs for a given scenario.

**List of Practical**

|  |  |  |
| --- | --- | --- |
| **Sr.#** | **Week** | **Topics** |
| 1 | One | Introduction to IDE-NetBeans, Getting Started with Java |
| 2 | Two | Java Basics (Input/output, variable declaration and initialization, strings and arrays) |
| 3 | Three | Java Basics (Selection Structure and Iterative Structure, functions) |
| 4 | Four | OOP (creating classes, objects, constructors) |
| 5 | Five | access modifiers, inheritance |
| 6 | Six | multiple/multilevel inheritance |
| 7 | Seven | Function overriding |
| 8 | Eight | Polymorphism |
| 9 | Nine | abstract classes and interfaces |
| 10 | Ten | exception handling and Java file handling |

**Introduction to NetBeans and Java**

**LAB-1**

**OOP (****abstract classes and interfaces)**

**LAB-9**

Data **abstraction** is the process of hiding certain details and showing only essential information to the user. Abstraction can be achieved with either **abstract classes** or interfaces.

The abstract keyword is a non-access modifier, used for classes and methods:

* **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
* **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods:

abstract class Animal {

public abstract void animalSound();

public void sleep() {

System.out.println("Zzz");

}

}

From the example above, it is not possible to create an object of the Animal class:

Animal myObj = new Animal(); // will generate an error

**Example:**

// Abstract class

abstract class Animal {

// Abstract method (does not have a body)

public abstract void animalSound();

// Regular method

public void sleep() {

System.out.println("Zzz");

}

}

// Subclass (inherit from Animal)

class Pig extends Animal {

public void animalSound() {

// The body of animalSound() is provided here

System.out.println("The pig says: wee wee");

}

}

class MyMainClass {

public static void main(String[] args) {

Pig myPig = new Pig(); // Create a Pig object

myPig.animalSound();

myPig.sleep();

}

}

**Interface**

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

**Syntax:**

/\* File name : NameOfInterface.java \*/

import java.lang.\*;

// Any number of import statements

public interface NameOfInterface {

// Any number of final, static fields

// Any number of abstract method declarations\

}

**Example:**

// Interface

interface Animal {

public void animalSound(); // interface method (does not have a body)

public void sleep(); // interface method (does not have a body)

}

// Pig "implements" the Animal interface

class Pig implements Animal {

public void animalSound() {

// The body of animalSound() is provided here

System.out.println("The pig says: wee wee");

}

public void sleep() {

// The body of sleep() is provided here

System.out.println("Zzz");

}

}

class MyMainClass {

public static void main(String[] args) {

Pig myPig = new Pig(); // Create a Pig object

myPig.animalSound();

myPig.sleep();

}

}

**Multiple Interfaces**

To implement multiple interfaces, separate them with a comma:

interface FirstInterface {

public void myMethod(); // interface method

}

interface SecondInterface {

public void myOtherMethod(); // interface method

}

class DemoClass implements FirstInterface, SecondInterface {

public void myMethod() {

System.out.println("Some text..");

}

public void myOtherMethod() {

System.out.println("Some other text...");

}

}

class MyMainClass {

public static void main(String[] args) {

DemoClass myObj = new DemoClass();

myObj.myMethod();

myObj.myOtherMethod();

}

}

**Task-1**

Write a java program to create an abstract class named shape that contains two integers and an empty method named printArea() Provide three classes named Rectangle,, Triangle and Circle such that each one of the classes extends the class shape. Each one of the class contains only the method printArea() that print the area of the given shape.

**Hint:** To create an abstract class that shows the hiding of elements in a class. At the same time inheritance property is used to extend the class shape into different geometrical figures. This represents the reusability of code for a programmer.

**Task-2**

Provide an interface Measurable with a method double getMeasure() that measures an object in some way. Make Employee implement Measurable. Provide a method double average(Measurable[] objects) that computes the average measure. Use it to compute the average salary of an array of employees.

**Assignment Question**

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 eac of the two classes and print the percentage of marks for both the students.