Experiment 1: Constructors and Access Control in Java

Title: Write a Java class Person with constructors for default, parameterized (name and age), and private access control.

Theory: Constructors in Java are special methods that are called when an object is instantiated. They have the same name as the class and do not have a return type. Constructors can be used to set initial values for object attributes. In Java, we can have multiple constructors in a class, each with a different set of parameters (constructor overloading).

Code:

```
public class Person {
    private String name;
    private int age;
    public Person() {
        this.name = "Unknown";
        this.age = 0;
    }
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    private Person(String name) {
        this.name = name;
        this.age = 0;  }
    public String getName() {
        return name;
```

```
    }
    public int getAge() {
        return age;
    }
    // Main method to demonstrate usage
    public static void main(String[] args) {
        // Using the default constructor
        Person defaultPerson = new Person();
            System.out.println("Default Person: Name = " + defaultPerson.getName() + ", Age = " +
defaultPerson.getAge());

        // Using the parameterized constructor
        Person parameterizedPerson = new Person("Shanti", 20);
          System.out.println("Parameterized Person: Name = " + parameterizedPerson.getName() + ",
Age = " + parameterizedPerson.getAge());
    }
}
```

Output:

Conclusion: In the ?main? method of the ?Person? class, we create instances using the default and parameterized constructors and display their values.

Experiment 2: - Polymorphism and Method Overloading

Title: Create a class Shape with methods calculateArea() and calculatePerimeter(). Implement polymorphism by extending this class to create subclasses like Circle,

Rectangle,andTriangle,eachoverridingthesemethodstocalculatetheirrespectiveareas and perimeters.

Theory: Polymorphism in Java is the ability of an object to take many forms. It is a fundamental concept in object-oriented programming (OOP) and can be implemented using method overriding and inheritance. By creating a base class with generic methods and extending this base class to create specialized subclasses, we can override these methods to perform different functionalities.

Code:

```
//Shape.java
publicabstract classShape{
public abstract double calculateArea(); publicabstractdoublecalculatePerimeter();
}
//Circle.java
publicclassCircleextendsShape{ private double radius;
publicCircle(doubleradius){ this.radius = radius;
}
@Override
public double calculateArea() { return Math.PI* radius * radius;
}
@Override
publicdoublecalculatePerimeter(){ return 2 * Math.PI * radius;
}
}
//Rectangle.java

publicclassRectangleextendsShape{ private double length;
privatedouble width;
publicRectangle(doublelength,doublewidth){ this.length = length;
this.width =width;
```

```
}
@Override
publicdoublecalculateArea(){ return length * width;
}
@Override
publicdoublecalculatePerimeter(){ return 2 * (length + width);
}}
//Triangle.java
publicclassTriangleextendsShape{ private double side1;
privatedoubleside2; privatedouble side3;
publicTriangle(doubleside1,doubleside2,doubleside3){ this.side1 = side1;
this.side2=side2; this.side3=side3;
}
@Override
public double calculateArea() { doubles=(side1+side2+side3)/2;
returnMath.sqrt(s * (s-side1)*(s -side2)*(s-side3));
}
@Override
publicdoublecalculatePerimeter(){ return side1 + side2 + side3;
}}

//Main.java
publicclassMain {
publicstaticvoidmain(String[]args){ Shape circle = new Circle(5);
Shaperectangle=newRectangle(4,6); Shape triangle = new Triangle(3, 4, 5);
//Demonstratingpolymorphism
System.out.println("Circle Area: " + circle.calculateArea()); System.out.println("Circle Perimeter: " +
```

circle.calculatePerimeter()); System.out.println("Rectangle Area: " + rectangle.calculateArea()); System.out.println("RectanglePerimeter:"+rectangle.calculatePerimeter()); System.out.println("Triangle Area: " + triangle.calculateArea()); System.out.println("Triangle Perimeter: " + triangle.calculatePerimeter());

}

}

Output:

Conclusion: Bythe end ofthis exercise, you should understand how to useinheritance and polymorphism to extend a base class and override its methods in subclasses. This demonstrates the flexibility and reusability of code, where a common interface (Shape) canbeusedtoperformdifferentimplementationsofthesamemethod(calculateArea()and calculatePerimeter()) depending on the specific type of object (Circle, Rectangle, Triangle).

Experimentno:3-RecursionandAccessControl

Title: Implement a recursive method to calculate the factorial of a number. Ensure the methodisaccessibleonlywithinitspackage.Provideaclassthatdemonstratesthe useof this method with proper access control handling.

Theory: A recursive method calls itself to solve smaller instances of the problem until itreachesthebasecase.Factorialofanumber,denotedasn!,istheproductofallpositive integers less than or equal to n. For instance, 5! = 5 * 4 * 3 * 2 * 1 = 120.

Code://FactorialCalculator.java class FactorialCalculator {

intfactorial(intn){

if(n<0){thrownewIllegalArgumentException("Factorialisnotdefinedfornegative numbers.");

}if(n==0){ return 1;

} elsereturn n * factorial(n - 1); } publicstaticvoidmain(String[]args){

FactorialCalculatorcalculator=newFactorialCalculator(); int number = 5;int result = calculator.factorial(number);

System.out.println("Factorialof "+number+"is: "+result); }}

Output:

Conclusion: In this implementation, the factorial method is designed to be package- private, which means it can only be accessed by other classes in the same package. This restriction ensures proper encapsulation and controlled access, preventing unintended usage from outside the package.

Experimentno:4?NestedandInnerClass

Title: Define an outer class Car with a static nested class Engine and an inner class Wheel. Implement methods in both Engine and Wheel classes to demonstrate their functionality. Create a separate class to instantiate and use objects of both Engine and Wheel classes.

Theory: In Java, a class can be nested within another class. Nested classes can be categorized into two types: static nested classes and inner classes. We'll define an outer class ?Car?with a staticnestedclass ?Engine? and an inner class ?Wheel?. Each class will havemethodstodemonstratetheirfunctionality.AseparateclassCarDemowillinstantiate and use objects of both ?Engine? and ?Wheel?.

Code:

public class Car { privateStringmodel;

publicCar(Stringmodel){ this.model = model;

}

publicStringgetModel(){ return model;

}

//Staticnested class

```
publicstaticclassEngine{ private int horsepower;
public Engine(int horsepower) { this.horsepower=horsepower;}
publicvoidstart(){
System.out.println("Enginewith"+horsepower+ "horsepoweris starting.");

}}

//Inner class
publicclassWheel{ private int size;
publicWheel(int size) {

this.size=size;}

publicvoidrotate(){
System.out.println("Wheelofsize"+size+"inchesisrotatingoncarmodel"+model
+".");
}}
public static void main(String[] args) { CarmyCar=newCar("TeslaModelS");
Car.EnginemyEngine=newCar.Engine(500); myEngine.start();
Car.WheelmyWheel=myCar.newWheel(19); myWheel.rotate();
}}
```

Output:

Conclusion: The use of nested classes in Java helps in logically grouping classes that are only used in one place, increasing encapsulation and readability. This approach simplifiesthedesignandmaintenanceofthecodebykeepingrelatedcomponentstogether and defining clear relationships between them.

Experimentno:5?InheritanceandPackages

Title: Create a superclass Animal with a method makeSound() and subclass Dog that overrides this method to bark. Demonstrate the use of inheritance and packages by importing Animal from a different package and using it in your Dog class.

Theory: Inheritance is a fundamental concept in object-oriented programming that allowsoneclasstoinherittheproperties andbehaviorsofanotherclass.Wewillcreatetwo packages: ?animals? and ?pets?. The ?Animal? class will reside in the ?animals? package, and the ?Dog? class will be in the ?pets? package.

Code:

```
package animals; publicclassAnimal{
public void makeSound() { System.out.println("Somegenericanimalsound");
} }
package pets;
import animals.Animal;
publicclassDogextendsAnimal{ @Override
public void makeSound() { System.out.println("Bark");}
publicstaticvoidmain(String[]args){ Dog dog = new Dog(); dog.makeSound();// Output: Bark
} }
```

Output:

Conclusion: The example demonstrates inheritance and the use of packages in Java. TheAnimal class is a superclass with amethod makeSound(). The Dog class, residingin a different package, extends Animal and overrides the makeSound() method to provide a specific implementation (barking).

Experimentno:6?ExceptionHandling

Title:Writeaprogramthatdemonstratesexceptionhandlingfordivision byzeroand arrayindexoutofbounds(ArrayIndexOutOfBoundsException).Includetry,catch,and finally blocks to handle these exceptions gracefully.

Theory: Exception handlingin Java is a powerful mechanism that allows a program to deal with unexpected situations (exceptions) during runtime. This ensures the program can handle errors gracefully without crashing.

Code:

```
//Exception.java
publicclassException {
publicstaticvoidmain(String[]args){
//DemonstrateArithmeticException(divisionbyzero) try {
inta=10; int b = 0;
intresult=a/ b;// This will throw ArithmeticException
}catch(ArithmeticExceptione){
System.out.println("CaughtanArithmeticException:Divisionby zeroisnot allowed.");
} finally{
System.out.println("ArithmeticExceptionhandlingcomplete.");
}
//DemonstrateArrayIndexOutOfBoundsException try {
int[] array={1, 2, 3};
intvalue=array[5];//This willthrowArrayIndexOutOfBoundsException
}catch(ArrayIndexOutOfBoundsExceptione){
System.out.println("Caught an ArrayIndexOutOfBoundsException: Invalid array index access.");
} finally{
```

System.out.println("ArrayIndexOutOfBoundsExceptionhandlingcomplete.");

System.out.println("Program execution continues smoothly.");

}

}

Output:

Conclusion: The provided program demonstrates how to handle exceptions gracefullyusing?try?,?catch?,and?finally?blocks.Bycatching?ArithmeticException? and ?ArrayIndexOutOfBoundsException?, the program avoids abrupt termination and provides meaningful error messages to the user.

Experimentno:7?StringHandling

Title: Implement a program that compares two strings for equality, ignoring case sensitivity. Additionally, concatenate two strings using StringBuilder for efficient string manipulation.

Theory: Comparingstrings for equalityin Java can bedone usingthe ?equals? method. To ignore case sensitivity, the ?equalIgnoreCase? method is used. This method compares two strings, ignoring the differences in their case (upper or lower).

Code:

public class StringManipulation { publicstaticvoidmain(String[]args){

//Comparingstringsforequality,ignoringcasesensitivity String str1 = "Hello";

Stringstr2 ="hello";

if(str1.equalsIgnoreCase(str2)){

System.out.println("Thestringsareequal,ignoringcase.");

}elseSystem.out.println("Thestringsarenotequal."); String part1 = "Hello, ";

Stringpart2 = "World!";

StringBuildersb=newStringBuilder(); sb.append(part1); sb.append(part2);

System.out.println("Concatenatedstring: "+sb.toString());

}

}

Output:

Conclusion: This program effectivelydemonstrates two keyoperations with strings in Java:comparingforequalitywhileignoringcasesensitivityandefficientlyconcatenating strings using ?StringBuilder?.

Experimentno:8?Threads

Title:CreateaJavaprogramthatdemonstratesmultithreadingusingbothThreadclass and Runnable interface. Implement a scenario where two threads increment a shared integer variable, ensuring thread safety using synchronization.

Theory:MultithreadinginJavaallowstheconcurrentexecutionoftwoormoreparts of a program to maximize the utilization of CPU. Java provides two ways to create a thread:byextendingthe ?Thread?class andbyimplementingthe?Runnable?interface.

Code:

// Shared Resource class Counter { privateintcount=0;

//Synchronizedmethodtoensurethreadsafety public synchronized void increment() { count++;

}

publicintgetCount(){ return count;

} }

//Threadclassimplementation

classThreadCounterextendsThread{ private Counter counter;

publicThreadCounter(Countercounter){ this.counter = counter;

```
}
@Override
publicvoid run() {
for(inti=0;i<1000;i++){ counter.increment();
} }}
classRunnableCounterimplementsRunnable{ private Counter counter;
publicRunnableCounter(Countercounter){

this.counter=counter;
}
@Override
publicvoid run() {
for(inti=0;i<1000;i++){ counter.increment();
}}}
public class MultiThreadDemo { publicstaticvoidmain(String[]args){ Counter counter = new
Counter();
Threadthread1=new ThreadCounter(counter);
Threadthread2=newThread(newRunnableCounter(counter)); thread1.start();
thread2.start(); try {
thread1.join(); thread2.join();
}catch(InterruptedExceptione){ e.printStackTrace();
}
//Print thefinal count
System.out.println("Final count:"+ counter.getCount());
}
}
```

Output:

Conclusion: In the provided example, two threads are created using different approaches: one by extending the Thread class and the other by implementing the Runnable interface. The program's output, ?Final count: 2000?, confirms that both threads successfully increment the counter 1000 times each, totaling 2000.

Experimentno:9?I/OandStreams

Title: Develop a Java application that reads input from the console and writes it to a file (output.txt). Ensure proper handling of file operations using FileWriter or BufferedWriter.

Theory:In Java,fileoperationsareperformed usingthe I/Ostreamclassesprovided in the java.io package. The BufferedReader class reads text from an input stream efficiently, buffering characters to provide efficient reading of characters, arrays, and lines. The InputStreamReader is a bridge from byte streams to character streams.

Code:

```
//ConsoleToFile.java
importjava.io.BufferedReader; import java.io.BufferedWriter; import java.io.FileWriter; import java.io.IOException;
import java.io.InputStreamReader;
publicclassConsoleToFile{publicstaticvoidmain(String[]args){ System.out.println("Enter text (type 'exit' to quit):");
try (BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
BufferedWriter writer = new BufferedWriter(new FileWriter("output.txt"))) {
String line; while(!(line=reader.readLine()).equalsIgnoreCase("exit")){
writer.write(line);
writer.newLine();// Addsanewline aftereachinput
}
```

System.out.println("Datawrittentooutput.txt successfully.");

}

catch(IOExceptione){

System.err.println("AnerroroccurreddduringI/Ooperations:"+ e.getMessage());

}

}

}

Output:

Conclusion: This Java application demonstrates how to read user input from the consoleandwriteittoafileusingBufferedReaderandBufferedWriter.Byusingthese classes, the program efficiently handles character input and output operations.

Experimentno:10?CollectionsFramework

Title: Implement a program that uses ?ArrayList? to store a collection of student names.Providemethodstoadd,remove,anditeratethroughthelistusingiteratorsand enhance loop.

Theory: An ?ArrayList? in Java is a resizable array implementation of the List interface, part of the ?java.util? package. It provides dynamic arrays in Java that can grow as needed. Elements can be added, removed, or accessed efficiently, making ?ArrayList? a flexible and powerful tool for managing collections of objects.

Code:

importjava.util.ArrayList; import java.util.Iterator; public class StudentList {

privateArrayList<String>students; public StudentList() {

students=new ArrayList<>();

}

```
//Method toadd astudent
publicvoidaddStudent(Stringname){ students.add(name);
}
// Method to remove a student by name publicvoidremoveStudent(Stringname){
students.remove(name);
}
//Methodtoiterateusinganiterator public void iterateWithIterator() {
Iterator<String> iterator = students.iterator(); System.out.println("IteratingusingIterator:"); while
(iterator.hasNext()) {
System.out.println(iterator.next());
}
}

//Methodtoiterateusingenhancedforloop public void iterateWithEnhancedLoop() {
System.out.println("IteratingusingEnhancedForLoop:"); for (String student : students) {
System.out.println(student);
}
}
public static void main(String[] args) { StudentListstudentList=newStudentList();
// Adding students studentList.addStudent("Shanti"); studentList.addStudent("Yamuna");
studentList.addStudent("Pamela ");
// Removing a student studentList.removeStudent("Pamela");
//IteratingthroughthelistusingIterator studentList.iterateWithIterator();
//IteratingthroughthelistusingEnhancedForLoop studentList.iterateWithEnhancedLoop();
}
}
```

Output:

Conclusion: This program showcases the use of ?ArrayList? for managing a collection of student names, highlighting the ease of adding and removing elements. Iteration is demonstrated using both ?Iterator? and enhanced for loops, each offering distinct advantages. This example illustrates the versatility and efficiency of ?ArrayList? in handling dynamic collections in Java.

Experimentno:11?AWTandSwing

Title: Create a Java Swing application that includes a JFrame with components like JLabel, JTextField, and JButton. Implement event listeners for the button to display input from the text field in a dialog box.

Theory: Java Swing is a GUI widget toolkit for Java. It is part of Oracle's Java FoundationClasses(JFC)andprovidesarichsetofcomponentsforbuildinggraphical user interfaces (GUIs). Swing offers a more flexible and powerful alternative to the earlier Abstract Window Toolkit (AWT).

Code:

packagecom.mycompany.studentform; import javax.swing.*;

import java.awt.*;

import java.awt.event.ActionEvent; importjava.awt.event.ActionListener;

publicclassStudentFormextendsJFrameimplementsActionListener{ private JLabel nameLabel;

private JLabel facultyLabel; private JLabel emailLabel; private JTextField nameField; privateJTextFieldfacultyField; private JTextField emailField; private JButton submitButton; public StudentForm() {

// Set up the JFrame setTitle("StudentForm"); setResizable(false); setSize(400, 300);

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); setLocationRelativeTo(null); // Center the window

```
//Initializecomponents

nameLabel=newJLabel("StudentName:"); nameField = new JTextField(10); facultyLabel = new JLabel("Faculty:"); facultyField = new JTextField(10); emailLabel = new JLabel("Email:"); emailField = new JTextField(10); submitButton = new JButton("Submit");

// Set layout using GridLayout for labels and text fields JPanelpanel=newJPanel(newGridLayout(4,2,4,6)); panel.add(nameLabel); panel.add(nameField); panel.add(facultyLabel); panel.add(facultyField); panel.add(emailLabel); panel.add(emailField); panel.add(submitButton); //AddActionListenertotheSubmitbutton submitButton.addActionListener(this);

//AddpaneltotheJFrame add(panel);

//DisplaytheJFrame setVisible(true);
}


@Override
publicvoidactionPerformed(ActionEvente){ if (e.getSource() == submitButton) {
StringstudentName=nameField.getText(); String faculty = facultyField.getText(); String email = emailField.getText();
Stringmessage="StudentName: " +studentName +"\n"+

"Faculty:"+faculty+"\n"+ "Email: " + email;
JOptionPane.showMessageDialog(this, message, "Student Information",
```

JOptionPane.INFORMATION_MESSAGE);

}

}

publicstaticvoidmain(String[]args){

//RunGUIconstructionintheEvent-Dispatchingthreadforthreadsafety new StudentForm();

}}

Output:

Conclusion: This program demonstrates the basic event handling mechanism in Swing and highlights the simplicity and effectiveness of Swing for building desktop applications with Java.

Experimentno:12?AppletsandJDBC

Title: Develop a Java applet that connects to a local MySQL database using JDBC. Display records from a table (Students) in an applet window using ResultSet and Statement classes.

Theory: Applet is a special type of program that is embedded in the webpage to generatethedynamiccontent.Itrunsinsidethebrowserandworksatclientside.JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the querywith the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database.

SourceCode:

importjavax.swing.*; import java.awt.*;

import java.awt.event.ActionEvent; importjava.awt.event.ActionListener;

publicclassTextDisplayFrameextendsJFrame{ private JTextField textField;

private JButton button;public TextDisplayFrame() { super("TextDisplayFrame");

```
setLayout(newFlowLayout());
JLabellabel=newJLabel("Entertext:"); textField = new JTextField(20);
button=newJButton("ShowText"); add(label);
add(textField); add(button);
button.addActionListener(newActionListener(){ @Override
publicvoidactionPerformed(ActionEvente){ String inputText = textField.getText();
JOptionPane.showMessageDialog(TextDisplayFrame.this, "Entered text: " + inputText,
});
"Text Displayed", JOptionPane.INFORMATION_MESSAGE);}
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); setSize(300, 150); // Set initial size
setLocationRelativeTo(null);//Centertheframeonthescreen setVisible(true); // Make the frame visible }
publicstaticvoidmain(String[]args){ SwingUtilities.invokeLater(new Runnable() {
@Override
publicvoid run() {
newTextDisplayFrame(); }
});
}
}
```

Conclusion: In this Java Swing application, we successfully demonstrated how to connect to a local MySQL database using JDBC and display records from a ?Students? table in an applet-like window using Swing components.