Object Oriented Programming with JAVA

Practical List

Last updated on 13/09/2014

Lab#	Aim	#Hrs.
1	**Programs to demonstrate printing to console (Hello World), Array and Scanner Class.** 1. Write a program to display "Hello World" on the console. 2. Write a program to find minimum and maximum from the given array. 3. Write a program that shows the use of Scanner class.	2
2	**Programs to demonstrate User-defined Classes, Constructor and this keyword.** 1. Develop a class Rectangle with data members: length, breadth. Write a member function findArea(), that computes and returns area of rectangle. Develop DemoRect class that has the main method. In main, create object of Rectangle class to store length, breadth. Finally, call the findArea() method to get area and display it on the screen. 2. Design a class named CustomTime which represents time: hour, minute, seconds. It provides below mentioned features: • Constructors: ▪ Create empty time, default to 00:00:00 ▪ Create time from hours, minutes and seconds provided. ▪ Create time from number of seconds provided. • Member functions: ▪ toString, returns String time in HH:MM:SS format. ▪ Getters and setters (Use of this keyword is preferred here) ▪ Write a class DemoCustomTime that has the main() and shows the use of class CustomTime. 3. Create a StringBuffer object and illustrate as below: • Refer to documentation via "javap -s java.lang.StringBuffer " • Know length, capacity of object, content using toString • Insert, append character • replace, substring, ==, equals, compareTo 4. Write a program to show the use of **static** keyword. 5. Write a program to show the implementation of **inner class**.	2

3	**Programs to demonstrate Inheritance, super, method overriding, static and final, avoid shadowing for data members, super constructors, interface.**
	1. Design a system for a NAIVE bank. Achieve following:
	• Maintain balance. Support deposit, withdraw and transfer features.
	• Types of accounts: Checking and Savings.
	• Checking account: Maintain transaction count. Deduct fees for each transaction over to allowed free transactions when issued command monthly.
	• Savings account: Maintain interest rate. Add interest to actual balance when issued command monthly.
	• Test as below:
	▪ Create mom's Savings account with SAVINGS INTEREST set to 0.5. No initial balance needed for SAVINGS account;
	▪ Create Harry's Checking account with initial balance as 100;
	▪ Deposit 10000 to mom's Savings;
	▪ Transfer 2000 from mom's Savings to Harry's Checking;
	▪ Withdraw 1500 from Harry's Checking;
	▪ Withdraw 80 from Harry's Checking;
	▪ Transfer 1000 from mom's Savings to Harry's Checking;
	▪ Withdraw 400 from Harry's Checking;//Simulate end of month;
	▪ Issue add interest to mom's Savings;
	▪ Issue deduct fees from Harry's Checking;
	▪ Display the current balance of Mom and Harry.
	▪ Match results with below:
	Mom's savings balance = $7035.0
	Harry's checking balance = $1116.0
	• If you need to match results, find what was the value of FREE_TRANSACTIONS provided the TRANSACTION_FEE applied was 2.0?
	• Note, that setting initial balance and deducting fees do not count towards actual transaction. But transfer to a checking account is actually a deposit, which is counted as transaction.
		2

4	**Programs to demonstrate abstract class and final keyword** 1. Write a java program containing following classes with mentioned features. • Person class : ▪ An instance variable fullName of type String.	**2**

- A constructor with parameter to initialize fullname.
- An abstract method getDescription() with return type String.
- A concrete method getName() which returns fullName of that person.

- Employee class:
  - An instance variable salary.
  - A constructor with two parameters full name and salary.
  - A method getSalary() which returns salary of that employee.
  - An implementation of getDescription() which returns description of type of that person (i.e. employee) with salary value.
- Student class:
  - An instance variable branch.
  - A constructor with two parameters full name and branch.
  - A method getBranch() which returns branch in which that student is studying.
  - An implementation of getDescription() which returns description of type of that person (i.e. student) with his/her branch name.
- DemoAbstract class:
  - In main method, create two variables person1 and person2 of Person type.
  - Initialize person1 with an object of Employee type with necessary values.
  - Initialize person2 with an object of Student type with necessary values.
  - Print description of each person.
- Sample output :
  - Manoj Pandey, an employee with a salary of Rs. 50,000.
  - Rohini Dave, a student studying in Computer Science.

2. Write a java program with classes Person and Employee as in above problem. Also write classes Manager and DemoFinal as per the following description.
   - Manager Class:
     - A String instance variable type which stores value of manager type (i.e. HR, General, Finance..).

	▪ A constructor with three parameters full name, salary and type. ▪ An instance method increaseSalary(), which takes an amount by which manger's salary will be incremented as a parameter and new increased amount is set as a new value to variable salary. ▪ An overridden method getDescription() which also prints the manager type of that manager. • DemoFinal class: ▪ In main method, create an instance manager1 of type Manager with suitable initial values. ▪ Print description of manager1. ▪ Call increaseSalary() on manager1 by passing some appropriate value. ▪ Print description of manager1 again. • Execute a program with different cases: ▪ Case A: Make an Employee class final. ▪ Case B: Make a method getDescription() in Employee class as a final method. ▪ Case C: Make an instance variable salary in Employee class as a final variable.	
5	**Programs to demonstrate use of interface and package.** 1. Write a java program as per the given description to demonstrate use of interface. • Define an interface RelationInterface. • Write three abstract methods: isGreater, isLess and isEqual. All methods have return type of Boolean and take an argument of type Line with which the caller object will be compared. • Write Line class implements RelationInterface interface. It has 4 double variables for the x and y co-ordinates of the line. • Write a constructor in Line class that initializes these 4 variables. • Write a method getLength() that computes length of the line. • [double length = Math.sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1))]. • Implement the methods in interface. • In class CompareLines.java, create two objects of Line class, call the three methods to compare the lengths of the lines. 2. Write a java program as per the given description to demonstrate use of interface for multiple inheritance.	**2**

- Define two interfaces named Terrestrial and Aquatic.
- In Terrestrial interface, write two methods eats() and walks(). Both methods have return type void and take no arguments.
- In Aquatic interface, write two methods eats() and swims(). Both methods have return type void and take no arguments.
- Write a class Amphibian that implements both the interfaces. Write all methods.
- In DemoMultipleInterface class, create Frog object of Amphibian and call all the methods.

3. Write a java program as per the given description to demonstrate concept of package.
   - Create a package studentpackage and create two classes StudentRecord and StudentRecordExample with following specifications.
   - StudentRecord class
     - Create private instance variables: name of String type and mathGrade, englishGrade, scienceGrade and average of type double.
     - Create a private static studentCount variable of type integer and initialize it to zero.
     - Write getter and setter method for name.
     - Write getter method for studentCount.
     - Write increaseStudentCount() that increments the studentCount.
   - StudentRecordExample class
     - Create an object instance of StudentRecord class.
     - Increment the studentCount by invoking method.
     - Create another object instance of StudentRecord class.
     - Increment the studentCount by invoking method.
     - Create the 3rd object instance of StudentRecord class.
     - Increment the studentCount by invoking method.
     - Set the names of the students.
     - Print a name.
     - Print number of students.
   - After creating the above mentioned classes, compile and run the program by providing "-classpath ." option but with the proper path to the class file.

6	**Programs to demonstrate use of multidimensional array and String handling.** 1. Write a program that inputs two 3 X 3 matrices and performs matrix multiplication on them. Display properly formatted output. 2. Write a program that returns the number of times that the string "Hi" appears anywhere in the given string. 3. Write a program that returns the string made of the first two characters of the given string. E.g. for given string "Hello" return "He". [If entered string is shorter than length 2 then return that string itself and return empty string for an empty string.] 4. Write a program that takes two strings and returns a string in the form of short+long+short, with the shorter string on the outside and longer on inside.	**2**
7	**Programs to demonstrate use of Exception handling** 1. Write a program that catches divide-by-zero exception using **try-catch** mechanism. • Take a numeric value and perform division by zero. Catch the ArithmeticException. 2. Write a program that demonstrates use of **multiple catch blocks**. Observe the importance of order of exception being caught in various catch blocks. • Take an integer array a[] of size 5. Divide the element a[5] by zero. Catch ArithmeticException, ArrayIndexOutOfBoundsException and superclass Exception in the respective catch blocks. 3. Write a program that demonstrates use of **finally** block. Observer the output of your program for different cases as mentioned below. • **Case A:** exception does not occur. ▪ Perform 25/5 mathematical operation. Catch NullPointerException • **Case B:** exception occurs but not handled. ▪ Perform 25/0 mathematical operation. Catch NullPointerException • **Case C:** exception occurs and handled. ▪ Perform 25/0 mathematical operation. Catch ArithmeticException 4. Write a program that creates and throws custom exception using **throw** keyword. Handle the custom exception in your program and print it. • Create a class **InvalidAgeException** that extends **Exception**. The	**2**

	constructor of **InvalidAgeException** with a **String** argument calls superclass constructor. • Create a **CustomExceptionDemo** class that has a static method void **validate** that take integer argument age and returns void. This method **throws InvalidAgeException**. Throw a **new InvalidAgeException** if the age is below 18 years. From the main method, call the validate method. 5. Write a program that demonstrates the use of **throws** keyword for propagation of checked exception • Create a class Simple with three methods m(), n(), p() with no arguments and void return type. m() and n() throws IOException. p() uses try-catch block. • In m(), throw new IOException("device error") as a checked exception. Call m() from n(). Call n() from p(). • In the main(), create a Simple class object and call the p() method.	

8	**Programs to demonstrate use of multiple threads and IO library.**
	1. Write a Java program as per the following description, to demonstrate the concept of thread priority.
	• MyThread class :
	▪ Create a class MyThread which extends Thread class.
	▪ It's run method prints number 1 to 5 with that thread name and priority.
	• MyThreadDemo class :
	▪ Create a class MyThreadDemo.
	▪ In the main method create two objects of MyThread class.
	▪ Set priority of first thread object to MIN value and second thread object to MAX value.
	▪ Start both the threads.
	▪ Print numbers 1 to 50 in main method with that thread name and its priority.
	2. Write a Java program which provides a correct solution to producer - consumer problem using separate threads for producer and consumer.
	3. Write a java program to test the following methods of File class.
	• public boolean exists() // Tests if this file/directory exists.
	• public long length() // Returns the length of this file.
	• public boolean isDirectory() // Tests if this instance is a directory.
	• public boolean isFile() // Tests if this instance is a file.
		2

	• public boolean canRead() // Tests if this file is readable.
	• public boolean canWrite() // Tests if this file is writable.
	• public boolean delete() // Deletes file/directory.
	• public void deleteOnExit() // Deletes file/directory when // program terminates.
	• public boolean renameTo(File dest) // Renames this file.
	• public boolean mkdir() // Makes (Creates) this directory
	• public String[] list() // List the contents of directory in a // String-array
	• public File[] listFiles() // List the contents of directory in a //File-array
	4. Write a Java program that creates a copy of a given file using Byte stream classes with buffering.	

9	**Programs to demonstrate the concepts of Serialization, Applet and AWT.** 1. Write a program that demonstrates the use of serialization and deserialization concepts in Java. • Employee class : ▪ This class implements Serializable interface. ▪ It has one String type instance variable name and two integer type instance variables id and salary. ▪ It has a parameterized constructor that initializes the member variables. • DemoSerialization class : ▪ In this class create an object of Employee class and serialize it into a file File1.txt. • DemoDeserialization class : ▪ In this class reconstruct the object of Employee class form File1.txt and display the values of member variables. 2. Write a java applet that displays the string “Hello All” in the center of the applet. Run this applet using appletviewer as well as in the browser. 3. Write a java program using AWT library as per the following description. • Create a Frame with blue background color. • Frame contains two buttons with labels Darker and Lighter.	**2**

	• When button Darker pressed it darkens the background color of frame. • When button lighter pressed it lightens the background color of frame. • When window close button is pressed the frame must be closed.	