**PART A 20 MARKS**

**COMPULSORY QUESTIONS**

**Answer ALL questions from this part.**

**Each question is worth 2.0 marks (10 x 2 = 20 marks).**

1. **2 Marks**

Abstract class vs concrete class

Answer:

Abstract classes cannot be used to instantiate objects. A class becomes an abstract class when it is declared with the key word abstract. An abstract class normally contains one or more abstract methods (but the presence of abstract method is not compulsory in an abstract class). An abstract class can contain concrete methods. A class that contains abstract method(s) must be declared as abstract class.

Concrete classes can be used to instantiate objects. It should not contain any abstract method(s).

1. **2 Marks**

Java language allows implementation of multiple interfaces by one class.

Answer:

Java allows a class to implement multiple interfaces.

The following example shows only part of the definition of the class Invoice that is implementing the interfaces Receivable and Payable.

class Invoice implements Receivable, Payable{

..

}

**Question 3 to Question 10 are similar to the above.**

**PART B 45 MARKS**

**Answer only nine (9) questions from this section.**

**Question 1 4 MARKS**

The following codes define **Person** class in a file named Person.java.

public class Person

{

private String name;

private int age;

public Person(String n, int a)

{

name = n;

age = a;

}

@override

public String toString()

{

return name + “ “ +age;

}

} //end class definition

Your tasks are:

* to define two *set* methods to set the name and age of Person.
* to define two *get* methods to retrieve the name and age of Person.
* to define a method named *isAdult* so that it will return true if the person’s age is greater than 18.
* to create PersonTester.java file wherein PersonTester class helps you
  + to create a Person object with the name value of “Johnson” and the age value of 35.
  + to use toString method to display the above mentioned Person object’s field values

**( Assume PersonTestor.java and Person.java are two separate files )**

Answer:

// codes within Person class

public void setName(String name)

{

this.name=name;

}

public void setAge(int age)

{

this.age=age;

}

public String getName( )

{

return name;

}

public int getAge( )

{

return age;

}

public boolean isAdult( )

{

return (age>18);

}

//PersonTester.java file contents

public class PersonTester

{

public static void main(String [] args)

{

Person aMan = new Person(“Johnson”,35); System.out.println(aMan.toString());

}

}

**Question 2 8 MARKS**

The following Java code demonstrates the concept of inheritance, where two subclasses **SalaryEmployee** and **HourlyEmployee** inherit the properties and methods from the superclass **Employee**.

Your tasks are:

• Write the missing implementation codes for constructors of SalaryEmployee and HourlyEmployee classes.

• What would have been the output produced by the program *DemoInheritance* if it is executed correctly?

import java.text.DecimalFormat;

class Employee

{

protected String empName;

protected String empSSN;

public Employee(String empName, String empSSN)

{

this.empName = empName;

this.empSSN = empSSN;

}

// update the employee name

public void setName(String empName)

{ this.empName = empName; }

// returns a formatted string to display employee information

public String toString()

{ return "Name: " + empName + '\n' + "SS#: " + empSSN; }

} //end of class Employee

class SalaryEmployee extends Employee

{

// new attributes that extends attributes in Employee

private double salary;

public SalaryEmployee(String empName, String empSSN, double salary)

{

super(empName,empSSN);

this.salary=salary;

}

// accessor method to return the salary

public double getSalary()

{ return salary; }

// mutator method to update the salary

public void setSalary(double sal)

{ salary = sal; }

// return a formated string with salaried employee information

// including name, ssn, status (salaried) and monthly pay

public String toString()

{

DecimalFormat fmt = new DecimalFormat("#.00");

return super.toString() + '\n' +

"Status: Salary" + '\n' +

"Salary: $" + fmt.format(salary) + "\n";

}

}//End of class SalaryEmployee

class HourlyEmployee extends Employee

{

// specialattributes for hourly pay

private double hourlyPay;

private double hoursWorked;

public HourlyEmployee(String empName, String empSSN,

double hourlyPay, double hoursWorked)

{

super(empName,empSSN);

this.hoursWorked=hoursWorked;

this.hourlyPay=hourlyPay;

}

// access and update the hourly pay and hours worked

public void setHourlyPay(double hourlyPay)

{ this.hourlyPay = hourlyPay; }

public void setHoursWorked(double hoursWorked)

{ this.hoursWorked = hoursWorked; }

// access and update the hourly pay and hours worked

public double getHourlyPay()

{ return hourlyPay; }

public double getHoursWorked()

{ return hoursWorked; }

// call toString() from superclass and add info type of

// employee (hourly), hourly pay rate and hours worked

public String toString()

{

DecimalFormat fmt = new DecimalFormat("#.00");

return super.toString() + '\n' +

"Status: Hourly" + '\n' +

"Rate: $" + fmt.format(hourlyPay) + "\n" +

"Hours: " + fmt.format(hoursWorked) + "\n";

}

}// End of HourlyEmployee

public class DemoInheritance

{

public static void main(String[] args)

{

HourlyEmployee hEmp = new HourlyEmployee("Steve Howard", "896-54-3217",10.50,40);

SalaryEmployee sEmp = new SalaryEmployee("Moira Dunn", "456-14-3787",800.0);

System.out.println(hEmp);

System.out.println(sEmp);

}

}

Answer:

Name: Steve Howard

SS#: 896-54-3217

Status: Hourly

Rate: $10.50

Hours: 40.00

Name: Moira Dunn

SS#: 456-14-3787

Status: Salary

Salary: $800.00

**Question 3 6 MARKS**

Draw UML diagram with respect to the class Employee and its subclasses which have been mentioned in the above question number 2.

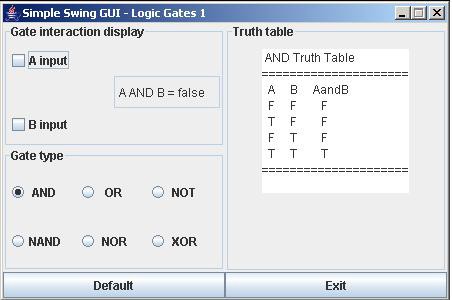
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  | Employee |  |  |
|  |  | -empName: String |  |  |
|  |  | -empSSN: String |  |  |
|  |  | <<constructor>> Employee(name : String, SSN : String) |  |  |
|  |  | + setName( name : String ) |  |  |
|  |  | + toString () : String |  |  |
|  |  | | |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | SalaryEmployee |  | HourlyEmployee |  |
|  | -salary: double |  | Students to complete here |  |
|  |  |  |  |  |
|  | <<constructor>> SalaryEmployee(name : String, SSN : String, salary : double) |  | Students to complete here |  |
|  | + setSalary( salary : double ) |  |  |  |
|  | + getSalary () : double |  |  |  |
|  | + toString () : String |  |  |  |
|  |  |  |  |  |

**Question 4 6 MARKS**

Identify the javax.swing components from the following GUI.

Explain how the layout managers can be used for this GUI.

(Note: The javax.swing components must be written by using correct upper case and lower case letters. i.e., same class names as defined in javax.swing)



**Answer:**

**components:**

JFrame – for the main frame

JPanel – main panel, central panel, bottom panel, central left panel, central right panel, top left

central panel, bottom central left panel, JCheckBox – for A input, B input

JRadioButton – gate type, i.e, AND, OR, NOT, NAND, NOR and XOR ButtonGroup – gate type

JButton – Default, Exit

JTextArea – Truth table display (right panel) JTextField – formula display (top left panel)

JLabel – some dummy labels may need to form Gate interaction display area

**Layout management:**

main panel (BorderLayout) = central panel (CENTER) + bottom panel (SOUTH)

central panel (GridLayout (1,2)) = central left panel + central right panel

central left panel (GridLayout (2,1)) = top left central panel + bottom central left panel

top left central panel (GridLayout(3,2)) = Two input checkboxes + formula text field + some

dummy labels

bottom central left panel (GridLayout (2,3)) = radio buttons for gate type bottom panel (GridLayout (1,2)) =Default button + Exit button

**Question 5 6 MARKS**

Insert the missing codes in the following Java program.

//Class PlayListener is an action listener

import javax.swing.JButton;

import javax.swing.JFrame;

**//Complete the two missing import statements below:**

import java.awt.event. ;

ActionListener

import java.awt.event. ;

ActionEvent

class PlayListener implements ActionListener

{

public void actionPerformed(ActionEvent event)

{

System.out.println("Play is selected");

}

}

// Class PlayTester is a GUI to use PlayListener

public class PlayTester extends JFrame

{

JButton bt;

PlayTester()

{

JButton bt = new JButton("Start Play");

add(bt);

**//Write your code here to install PlayListner**

**//Write your code related to ‘Start Play’ button-click**

**//event handled by the PlayListener**

ActionListener listener = new PlayListener();

bt.addActionListener(listener);

}

public static void main(String [] args)

{

PlayTester aGame = new PlayTester();

aGame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

aGame.setSize(300, 300);

aGame.setVisible(true);

}

}

**Question 6 5 MARKS**

Assume that the following program compiles and runs without any errors. What is the output produced by this program.

The third ‘for’ loop has to be rewritten by using an iterator. The iterator should iterate through the teamList object, retrieve the data from teamList and display the retrieved data on screen.

import java.util.LinkedList;

public class MyProg

{

public static void main(String[] args)

{

String[] team = {"Australia","New Zealand", "India","UK","Pakistan","China"};

LinkedList<String> teamList = new LinkedList<String>();

for (int i=0; i<team.length; i++)

{

teamList.add(team[i]);

}

for (int i=0; i<teamList.size(); i++)

{

String tempTeam = teamList.get(i);

if (tempTeam.equals("New Zealand"))

{

teamList.remove(i);

}

}

for (int j=0; j<teamList.size(); j++)

{

System.out.println(teamList.get(j).toString());

}

}

}//end of class definition

**Answer:**

Output:

Australia India

UK Pakistan China

Iterator <String> countryIterator = teamList.iterator();

while(countryIterator.hasNext())

{

String nextCountry = countryIterator.next();

System.out.println(nextCountry +”\n”);

}

**Question 7 5 MARKS**

Write a program that uses a Stack object to determine whether a string is a Palindrome or not (i.e., a Palindrome is spelled identically backward and forward. E.g “mum”).

**Answer:**

import java.util.Scanner;

import java.util.Stack;

public class Palindrome

{

public static void main( String[] args )

{

Stack< Character > stack =

new Stack< Character >();

// get input string

Scanner scanner = new Scanner( System.in );

System.out.println( "Please enter a string:" );

String input = scanner.nextLine();

// convert to lowercase

input = input.toLowerCase();

// cycle through input one char at a time to

// create stack of all characters

for ( char letter : input.toCharArray() )

stack.push( letter );

// test for palindrome

for ( char letter : input.toCharArray() )

{

// check for non-matching characters if ( letter != stack.pop() )

{

System.out.println( "The input string is not a

Palindrome" );

return;

} // end if

} // end for

// if no non-matching characters were found, string is a palindrome

System.out.println( "The input string is a palindrome" );

} // end main

} // end class Palindrome

**Question 8 5 MARKS**

(a) Find out whether the tree in the following diagram is a binary search tree or not. Justify your answer. Write down the sequence in terms of the inorder, preorder and postorder traversals of this tree. (**2.5 MARKS**)

40

21 65

8 31 52 77

**Answer:**

This is a binary search tree because for each node in this tree, the node’s data is greater than all the data in the node’s left subtree and the node’s data is less than all the data in node’s right subtree. The tree also contains unique values only.

Inorder traversal sequence: 8 21 31 **40** 52 65 77

Preorder traversal sequence: **40** 21 8 31 65 52 77

Postorder traversal sequence: 8 31 21 52 77 65 **40**

(b) Given a Big-O estimate for the running time ***T* (*n*) = *n*2 + 1 + *n* log2 *n ,***

is this algorithm faster than O(*nlog2n*)? Explain your answer. (**2.5 MARKS**)

**Answer:**

The given algorithm is not faster than O(nlog2n).

*n*2 term is the dominant term in the running time term *T* (*n*) = *n*2 + 1 + *n* log2 *n*

We can always find a constant ‘c’ so that when *n*>c, *n*2> *n*log2 *n*. So it is in the order of O(*n*2).

Hence, this algorithm with O(n2) is slower than O(*nlog2n*), because we can find a constant c such that for all n>c, *n*2> *n*log2 *n.* This means O(n2) grows with *n* faster than O(nlog2n).

The program takes more running time means the program is running slowly.

**Question 9 5 MARKS**

**Question 10 5 MARKS**

**End of Paper**