Q1) Refactoring is used to  
  
**1: 4 |||| 17% 2: 0 | 0% 3: 20 ||||||||||||||||| 83% 4: 0 | 0%   
  
Polling options**1: Break down a system into smaller components 2: Introduce anti-patterns in the code 3: Improve code structure but no effect to behav 4: Represent objects in UML

A) 3

Q2) In the context of systems and subsystems, what does a subsystem represent?  
  
**1: 12 ||||||||||| 50% 2: 10 ||||||||| 42% 3: 1 | 4% 4: 1 | 4%   
  
Polling options**1: A collection of related packages 2: A single class within a system 3: An object representation in a database 4: An abstract class

A) 1

Option 2 is correct for component.

Q3) Which of the following is a feature of Object Serialization in Java?  
  
**1: 5 |||||| 26% 2: 2 ||| 11% 3: 8 ||||||||| 42% 4: 4 ||||| 21%   
  
Polling options**1: Representing Objects as Tables 2: Active Objects 3: Persistent Objects 4: Collaboration

A) 4

Serialization is mainly brought into picture to easily read and write data.

Q1) Refactor

class BankAccount {

private String accountNumber;

private double balance;

// Other account-related properties and methods

// Withdraw method currently implemented in the TransactionHistory class

}

class TransactionHistory {

private List<Transaction> transactions;

// Other transaction history properties and methods

public void withdraw(double amount) {

// Logic to handle withdrawals and add a new transaction to the list

}

}

A)

class BankAccount {

private String accountNumber;

private double balance;

// Other account-related properties and methods

public void withdraw(double amount) {

// Logic to handle withdrawals and add a new transaction to the list

}

}

class TransactionHistory {

private List<Transaction> transactions;

// Other transaction history properties and methods

}

Q2)

class Car {

    private String make;

    private String model;

    private int year;

    private int mileage;

    // Other car-related properties and methods

    // Method to calculate car insurance premium in the InsuranceCalculator class

}

class Motorcycle {

    private String make;

    private String model;

    private int year;

    private int engineCapacity;

    // Other motorcycle-related properties and methods

    // Method to calculate motorcycle insurance premium in the InsuranceCalculator class

}

class Truck {

    private String make;

    private String model;

    private int year;

    private int cargoCapacity;

    // Other truck-related properties and methods

    // Method to calculate truck insurance premium in the InsuranceCalculator class

}

class InsuranceCalculator {

    // Other properties and methods

    public double calculateInsurancePremium(Car car) {

        // Logic to calculate car insurance premium based on car-specific attributes

    }

    public double calculateInsurancePremium(Motorcycle motorcycle) {

        // Logic to calculate motorcycle insurance premium based on motorcycle-specific attributes

    }

    public double calculateInsurancePremium(Truck truck) {

        // Logic to calculate truck insurance premium based on truck-specific attributes

    }

}

A)

class Car {

    private String make;

    private String model;

    private int year;

    private int mileage;

    // Other car-related properties and methods

     public double calculateInsurancePremium(Car car) {

        // Logic to calculate car insurance premium based on car-specific attributes

    }

}

class Motorcycle {

    private String make;

    private String model;

    private int year;

    private int engineCapacity;

    // Other motorcycle-related properties and methods

    public double calculateInsurancePremium(Motorcycle motorcycle) {

        // Logic to calculate motorcycle insurance premium based on motorcycle-specific attributes

    }

}

class Truck {

    private String make;

    private String model;

    private int year;

    private int cargoCapacity;

    // Other truck-related properties and methods

    public double calculateInsurancePremium(Truck truck) {

        // Logic to calculate truck insurance premium based on truck-specific attributes

    }

}

Q3)

class ShoppingCart {

    private List<Item> items;

    private double taxRate;

    private double discount;

    // Other shopping cart properties and methods

    public double calculateOrderTotal() {

        double subtotal = 0.0;

        for (Item item : items) {

            subtotal += item.getPrice();

        }

        double tax = subtotal \* (taxRate / 100);

        double discountAmount = subtotal \* (discount / 100);

        double total = subtotal + tax - discountAmount;

        // Apply some additional complex logic here

        return total;

    }

}

class Item {

    private String name;

    private double price;

    // Other item-related properties and methods

}

A)

class ShoppingCart {

    private List<Item> items;

    private double taxRate;

    private double discount;

    public double calculateTax(double subtotal) {

        return subtotal \* (taxRate / 100);

    }

    public double calculateDiscount(double subtotal) {

        return subtotal \* (discount / 100);

    }

    // Other shopping cart properties and methods

    public double calculateOrderTotal() {

        double subtotal = 0.0;

        for (Item item : items) {

            subtotal += item.getPrice();

        }

        double tax = calculateTax(subtotal);

        double discountAmount = calculateDiscount(subtotal);

        double total = subtotal + tax - discountAmount;

        // Apply some additional complex logic here

        return total;

    }

}

class Item {

    private String name;

    private double price;

    // Other item-related properties and methods

    public double getPrice() {

        return this.price;

    }

}

Q4)

import java.util.ArrayList;

import java.util.List;

public class NumberOperations {

    public static void main(String[] args) {

        List<Integer> numbers = new ArrayList<>();

        numbers.add(10);

        numbers.add(5);

        numbers.add(20);

        numbers.add(15);

        numbers.add(25);

        int sum = 0;

        int max = Integer.MIN\_VALUE;

        int min = Integer.MAX\_VALUE;

        for (int num : numbers) {

            sum += num;

            if (num > max) {

                max = num;

            }

            if (num < min) {

                min = num;

            }

        }

        double average = (double) sum / numbers.size();

        System.out.println("Numbers: " + numbers);

        System.out.println("Sum: " + sum);

        System.out.println("Average: " + average);

        System.out.println("Max: " + max);

        System.out.println("Min: " + min);

    }

}

A)

import java.util.ArrayList;

import java.util.List;

public class NumberOperations {

    public static int calculateSum(List<Integer> numbers) {

        int sum = 0;

         for (int num : numbers) {

            sum += num;

        }

        return sum;

    }

    public static int findMax(List<Integer> numbers) {

          int max = Integer.MIN\_VALUE;

         for (int num : numbers) {

            if (num > max) {

                max = num;

            }

        }

        return max;

    }

    public static int findMin(List<Integer> numbers) {

         int min = Integer.MAX\_VALUE;

         for (int num : numbers) {

            if (num < min) {

                min = num;

            }

        }

        return min;

    }

    public static double calculateAvg(List<Integer> numbers) {

        return (double) calculateSum(numbers) / numbers.size();

    }

    public static void main(String[] args) {

        List<Integer> numbers = new ArrayList<>();

        numbers.add(10);

        numbers.add(5);

        numbers.add(20);

        numbers.add(15);

        numbers.add(25);

        int sum = calculateSum(numbers);

        int max = findMax(numbers);

        int min = findMin(numbers);

        double average = calculateAvg(numbers);

        System.out.println("Numbers: " + numbers);

        System.out.println("Sum: " + sum);

        System.out.println("Average: " + average);

        System.out.println("Max: " + max);

        System.out.println("Min: " + min);

    }

}

Q5) Add mentioned method in below code getDaysBetweenInclusive()

import java.time.\*;

class DateUtils {

    public int getYear(LocalDate date){

        return date.getYear();

    }

    public Month getMonth(LocalDate date){

       return date.getMonth();

    }

    public int getDay(LocalDate date){

        return date.getDayOfMonth();

    }

}

public class Main {

    public static void main(String args[]){

    DateUtils dateutils = new DateUtils();

    System.out.println(dateutils.getYear(LocalDate.now()));

    System.out.println(dateutils.getMonth(LocalDate.now()));

}

}

A)

import java.time.\*;

class DateUtils {

    public int getYear(LocalDate date){

        return date.getYear();

    }

    public Month getMonth(LocalDate date){

       return date.getMonth();

    }

    public int getDay(LocalDate date){

        return date.getDayOfMonth();

    }

    // public int getDaysBetweenInclusive(LocalDate startDate, LocalDate endDate) {

    //     int dayCount = 0;

    //     if (isAfter(startDate, endDate)){

    //          for (int i = (Integer) startDate; i <= (Integer) endDate; i++) {

    //              dayCount++;

    //          }

    //     } else {

    //         dayCount = -1;

    //     }

    //     return dayCount;

    // }

    public boolean isAfter(LocalDate startDate, LocalDate endDate) {

        if (startDate < endDate) {

            return true;

        }

        return false;

    }

}

public class Main {

    public static void main(String args[]){

    DateUtils dateutils = new DateUtils();

    System.out.println(dateutils.getYear(LocalDate.now()));

    System.out.println(dateutils.getMonth(LocalDate.now()));

}

}

Q6)

class ElectronicProduct {

    private String name;

    private double price;

    private String brand;

    private String category;

    public ElectronicProduct(String name, double price, String brand, String category) {

        this.name = name;

        this.price = price;

        this.brand = brand;

        this.category = category;

    }

    public void displayProductInfo() {

        System.out.println("Product: " + name);

        System.out.println("Price: $" + price);

        System.out.println("Brand: " + brand);

        System.out.println("Category: " + category);

    }

    // Other electronic-specific methods

}

class BookProduct {

    private String name;

    private double price;

    private String author;

    private String genre;

    public BookProduct(String name, double price, String author, String genre) {

        this.name = name;

        this.price = price;

        this.author = author;

        this.genre = genre;

    }

    public void displayProductInfo() {

        System.out.println("Book: " + name);

        System.out.println("Price: $" + price);

        System.out.println("Author: " + author);

        System.out.println("Genre: " + genre);

    }

    // Other book-specific methods

}

public class OnlineShopping {

    public static void main(String[] args) {

        ElectronicProduct electronicProduct = new ElectronicProduct("Smartphone", 699.99, "Samsung", "Electronics");

        BookProduct bookProduct = new BookProduct("The Alchemist", 14.99, "Paulo Coelho", "Fiction");

        // Code to manage electronic and book products

    }

}

public class StudentManagementSystem {

    public void enrollStudent(String studentName, int age, int studentId, boolean hasScholarship) {

        Student student = new Student(studentName, age, studentId, hasScholarship);

        double tuitionFees = calculateTuitionFees(student);

        displayStudentDetails(student, tuitionFees);

    }

    private double calculateTuitionFees(Student student) {

        // Assume some complex logic to calculate tuition fees based on student details

        // For example, consider a basic calculation based on age and scholarship

        double baseFees = 1000.0;

        double scholarshipDiscount = calculateScholarshipDiscount(student);

        double tuitionFees = calculateTotalFee(baseFees, scholarshipDiscount);

        return tuitionFees;

    }

    private double calculateScholarshipDiscount(Student student) {

        return student.hasScholarship() ? 0.2 : 0.0;

    }

    private double calculateTotalFee(double baseFees, double scholarshipDiscount) {

        return baseFees \* (1 - scholarshipDiscount);

    }

    private void displayStudentDetails(Student student, double tuitionFees) {

        // Assume displaying student details, including tuition fees

        // For example:

        System.out.println("Student Name: " + student.getName());

        System.out.println("Student Age: " + student.getAge());

        System.out.println("Student ID: " + student.getStudentId());

        System.out.println("Tuition Fees: " + tuitionFees);

    }

    public static void main(String[] args) {

        // Example usage of StudentManagementSystem

    StudentManagementSystem sms = new StudentManagementSystem();

    sms.enrollStudent("ABC", 20, 1001, true);

    }

}

class Student {

    private String name;

    private int age;

    private int studentId;

    private boolean hasScholarship;

    public Student(String name, int age, int studentId, boolean hasScholarship) {

        this.name = name;

        this.age = age;

        this.studentId = studentId;

        this.hasScholarship = hasScholarship;

    }

    // Getters and setters for name, age, and studentId

    public int getStudentId() {

        return this.studentId;

    }

    public int getAge() {

        return this.age;

    }

    public String getName() {

        return this.name;

    }

    public boolean hasScholarship() {

        return hasScholarship;

    }

    public void setScholarship(boolean hasScholarship) {

        this.hasScholarship = hasScholarship;

    }

}

public class Shape {

    // Common shape properties and methods

}

public class Circle extends Shape {

    // Circle-specific properties and methods

}

public class Square extends Shape {

    // Square-specific properties and methods

}

// Client code

public void processShape(Shape shape) {

    // Check if the shape is a Circle and downcast

    if (shape instanceof Circle) {

        Circle circle = (Circle) shape;

        // Do something with the circle

    }

    // Check if the shape is a Square and downcast

    else if (shape instanceof Square) {

        Square square = (Square) shape;

        // Do something with the square

    }

    // ... More checks for other subclasses

}

import java.util.Random;

import java.util.Scanner;

public class MonsterBattleArena {

    public static void main(String[] args) {

        System.out.println("Welcome to the Monster Battle Arena!");

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter your name: ");

        String playerName = scanner.nextLine();

        String[] monsters = { "Zog", "Blarg", "Moo", "Grr" };

        String playerMonster = monsters[new Random().nextInt(monsters.length)];

        String enemyMonster = monsters[new Random().nextInt(monsters.length)];

        System.out.println("Hello, " + playerName + "! You have a " + playerMonster + " to fight with!");

        int playerHealth = 100;

        int enemyHealth = 100;

        while (true) {

            System.out.println("Choose your move:\n1. Attack\n2. Defend\n3. Run");

            int playerChoice = scanner.nextInt();

            if (playerChoice == 1) {

                int playerAttack = new Random().nextInt(11) + 10;

                int enemyAttack = new Random().nextInt(11) + 5;

                System.out.println("You attacked " + enemyMonster + " for " + playerAttack + " damage!");

                System.out.println(enemyMonster + " counter-attacked for " + enemyAttack + " damage!");

                playerHealth -= enemyAttack;

                enemyHealth -= playerAttack;

            } else if (playerChoice == 2) {

                System.out.println("You tried to defend, but it didn't work!");

                int playerAttack = new Random().nextInt(6) + 5;

                System.out.println(enemyMonster + " attacked you for " + playerAttack + " damage while you were defending!");

                playerHealth -= playerAttack;

            } else if (playerChoice == 3) {

                System.out.println(enemyMonster + " blocked your escape! You can't run!");

            } else {

                System.out.println("Invalid choice! Try again.");

            }

            // Check if anyone is defeated

            if (playerHealth <= 0) {

                System.out.println("Oh no! You were defeated. Better luck next time!");

                break;

            } else if (enemyHealth <= 0) {

                System.out.println("Congratulations! You defeated the enemy!");

                break;

            }

        }

    }

}