Assessment 4 (30th June 2024)

This assessment covers access specifiers, constructors, control statements, method overloading, and overriding.

Instructions: Please answer the following questions by writing Java code. You can use an online IDE or any preferred code editor.

Note: Aim for clean, readable, and well-commented code.

1. Access Specifiers (10 points)

Create a class called `Employee` with the following attributes:

* `name` (String) - private

* `department` (String) - protected

* `salary` (double) - public

a) Implement a constructor that takes `name` and `department` as arguments and initializes the corresponding attributes. (5 points)

b) Create a public method called `getSalary` that returns the `salary` attribute. (5 points)

Code:

```
public class Employee {
    private String name;
    protected String department;
    public double salary;

    // Constructor that initializes name and department
    public Employee(String name, String department) {
        this.name = name;
        this.department = department;
        this.salary = 0.0;  // Initialize salary to 0.0 by default
    }

    // Public method to get the salary
    public double getSalary() {
        return this.salary;
    }

    // Optional: Public method to set the salary
    public void setSalary(double salary) {
        this.salary = salary;
    }

    // Optional: Public method to get the name
    public String getName() {
        return this.name;
    }

    // Optional: Public method to get the department
    public String getDepartment() {
        return this.department;
    }

    public static void main(String[] args) {
        Employee emp = new Employee("John Doe", "Engineering");
        emp.setSalary(50000.0);
        System.out.println("Employee Salary: " + emp.getSalary());
    }
}
```

2. Constructors (10 points)

Create a class called `Circle` with the following attributes:

* `radius` (double)

a) Implement a constructor that takes a `radius` as an argument and initializes the corresponding attribute. (5 points)

b) Implement a no-argument (default) constructor that sets the `radius` to 1.0 by default. (5 points)

Code:

```
public class GradeCalculator {

    public static String calculateGrade(int marks) {
        if (marks >= 90) {
            return "A";
        } else if (marks >= 80) {
            return "B";
        } else if (marks >= 70) {
            return "C";
        } else if (marks >= 60) {
            return "D";
        } else {
            return "F";
        }
    }

    public static void main(String[] args) {
        int marks = 85;
        System.out.println("Grade: " + calculateGrade(marks));
    }
}
```

3. Control Statements (15 points)

Write a method called `calculateGrade` that takes an integer representing the student's marks as input and returns the corresponding grade based on the following criteria:

* Marks >= 90: A

* Marks >= 80 and less than 90: B

* Marks >= 70 and less than 80: C

* Marks >= 60 and less than 70: D

* Marks less than 60: F

Use appropriate control flow statements (if-else or switch) to achieve this logic. (15 points)

Code:

```
public class GradeCalculator {

    public static String calculateGrade(int marks) {
        if (marks >= 90) {
            return "A";
        } else if (marks >= 80) {
            return "B";
        } else if (marks >= 70) {
            return "C";
        } else if (marks >= 60) {
            return "D";
        } else {
            return "F";
        }
    }

    public static void main(String[] args) {
        int marks = 85;
        System.out.println("Grade: " + calculateGrade(marks));
    }
}
```

4. Method Overloading (10 points)

Create a class called `Calculator` with the following methods:

* `add(int a, int b)` - This method adds two integers and returns the sum.

* `add(double a, double b)` - This method adds two doubles and returns the sum.

Both methods are named `add` but have different parameter types. This is an example of method overloading.

Code:

```
public class Calculator {

    // Method to add two integers
    public int add(int a, int b) {
        return a + b;
    }

    // Method to add two doubles
    public double add(double a, double b) {
        return a + b;
    }

    public static void main(String[] args) {
        Calculator calc = new Calculator();
        System.out.println("Sum of integers: " + calc.add(3, 4));
        System.out.println("Sum of doubles: " + calc.add(3.5, 4.5));
    }
}
```

5. Method Overriding (15 points)

Create a class called `Animal` with a method called `makeSound` that simply prints "Generic animal sound".

Now, create a subclass called `Dog` that inherits from `Animal`. In the `Dog` class, override the `makeSound` method to print "Woof!".

Code:

```
public class Animal {
    public void makeSound() {
        System.out.println("Generic animal sound");
    }
}


public class Dog extends Animal {
    @Override
    public void makeSound() {
        System.out.println("Woof!");
    }


    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        myAnimal.makeSound();


        Dog myDog = new Dog();
        myDog.makeSound();
    }
}
```

Bonus (10 points)

Write a program that simulates a simple ATM machine. The program should allow users to:

1. Check their balance (assume a starting balance of $1000)

2. Withdraw cash (ensure there are sufficient funds)

3. Deposit cash

Use appropriate loops and conditional statements to implement this functionality.

Code:

```
import java.util.Scanner;

public class ATM {
    private double balance;

    // Constructor to initialize the starting balance
    public ATM() {
        this.balance = 1000.0;  // Starting balance
    }

    // Method to check the balance
    public double checkBalance() {
        return this.balance;
    }

    // Method to withdraw cash
    public void withdraw(double amount) {
        if (amount > 0 && amount <= this.balance) {
            this.balance -= amount;
            System.out.println("Withdrawal successful. New balance: $" + this.balance);
        } else if (amount > this.balance) {
            System.out.println("Insufficient funds. Current balance: $" + this.balance);
        } else {
            System.out.println("Invalid amount entered. Please try again.");
        }
    }

    // Method to deposit cash
    public void deposit(double amount) {
        if (amount > 0) {
```

```
        this.balance += amount;
        System.out.println("Deposit successful. New balance: $" + this.balance);
    } else {
        System.out.println("Invalid amount entered. Please try again.");
    }
}

// Main method to run the ATM simulation
public static void main(String[] args) {
    ATM atm = new ATM();
    Scanner scanner = new Scanner(System.in);
    int choice;

    // Loop to keep the ATM running until the user decides to exit
    do {
        System.out.println("\nATM Menu:");
        System.out.println("1. Check Balance");
        System.out.println("2. Withdraw Cash");
        System.out.println("3. Deposit Cash");
        System.out.println("4. Exit");
        System.out.print("Enter your choice: ");
        choice = scanner.nextInt();

        // Switch case to handle user choices
        switch (choice) {
            case 1:
                System.out.println("Current balance: $" + atm.checkBalance());
                break;
            case 2:
                System.out.print("Enter amount to withdraw: ");
                double withdrawAmount = scanner.nextDouble();
                atm.withdraw(withdrawAmount);
                break;
            case 3:
                    System.out.print("Enter amount to deposit: ");
                    double depositAmount = scanner.nextDouble();
                    atm.deposit(depositAmount);
                    break;
                case 4:
                    System.out.println("Exiting... Thank you for using the ATM.");
                    break;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        } while (choice != 4);

        scanner.close();
    }
}
```