Object-Oriented Programming

Subject: CSW2(CSE3141)

Session: Feb 2024 to April 2024

Branch: CSE&CSIT

Section : All

Q1. Write a Java code snippet that comprises a **Car** class and a **CarTester** class. The **Car** class should define private fields for **make** and **model**, along with a parameterized constructor and getter/setter methods for these attributes. In the **CarTester** class, instantiate two instances of the **Car** class: **myCar1** with a specified make and model, and **myCar2** with null values. After instantiation, the **CarTester** class should retrieve and print the initial make and model of both cars. Then, it should set new values for **myCar2** using setter methods and print the updated make and model.

Q2. Design a Java class called **Rectangle** with private attributes **length** and **width**. Include a constructor to initialize these attributes, as well as setter and getter methods for each attribute. Additionally, implement methods to **calculate** the area and perimeter of the rectangle. Write a **main** method to create an object of the **Rectangle** class, set values for its attributes, and display the area and perimeter.

Q3. Write a Java program that defines a '**Point**' class with attributes '**X**' and '**Y**', and includes a **parameterized constructor** to initialize these attributes. Implement a **copy constructor** to create a new point object with the same attribute values. Ensure that modifications made to one object do not affect the other. Utilize **getter** and **setter** methods to retrieve and update the attribute values.

Q4. Write a Java code snippet comprising two classes: **Laptop** and **Main**. The **Laptop** class defines private fields **model** and **price**, alongside setter methods for each attribute. Additionally, it overrides the **toString()** method to return a string representation of the laptop's model and price. In the **Main** class, create an instance of **Laptop**, setting its model using the setter method. Then, print the laptop object using the **toString()** method. Describe the functionality of the toString() method in the Laptop class and explain how it is utilized in the Main class.

Q5. Developing a Simple College Management System in Java

Create a Java program for managing colleges and students. The provided classes model the relationship between colleges and students.

The **College** class represents a college with attributes **collegeName** and **collegeLoc**. The **Student** class represents a student with attributes **studentId**, **studentName**, and a **reference** to a College object. Enhance the Student class by adding a constructor that assigns a student ID, name, and college information. Additionally, add a method named **displayStudentInfo()** that prints the student's ID, name, and the college information in which they are enrolled.

In the **Main** class, instantiate at least two **College** objects and at least two **Student** objects. Enroll each student in one of the colleges. Then, display the information of all colleges and all students using the appropriate methods.

Q6. Library System Assignment:

Task: Develop a Java program for a library system incorporating encapsulation, abstraction, and inheritance.

i. LibraryResource Class:

- Abstract class with private attributes for title and author.
- Constructor, getters, setters, and an abstract method displayDetails().

ii. Book Class:

- Subclass of LibraryResource with additional attribute pageCount.
- Constructor, getters, setters, and displayDetails() method override.

iii. Magazine Class:

- Subclass of LibraryResource with additional attribute issueDate.
- Constructor, getters, setters, and displayDetails() method override.

iv. DVD Class:

- Subclass of LibraryResource with additional attribute duration.
- Constructor, getters, setters, and displayDetails() method override.

v. LibrarySystem Class (Main):

- Instantiate various library resources (e.g., Book, Magazine, DVD).
- Call displayDetails() for each instance to show resource information.

Q7. Consider a scenario where you are tasked with developing a simple banking application using Java, employing the concept of polymorphism. Your application should consist of three classes: **Account**, **SavingsAccount**, and **CurrentAccount**. The **Account** class serves as the base class with private attributes for account number and balance, along with abstract methods for deposit and withdrawal. The **SavingsAccount** class, a subclass of **Account**, should include an additional attribute for interest rate and override the deposit method to calculate interest, as well as override the withdrawal method to ensure a sufficient balance. Similarly, the **CurrentAccount** class, also a subclass of **Account**, should incorporate an overdraft limit attribute and override the withdrawal method to check the overdraft limit. Implement the classes as described, ensuring proper encapsulation and abstraction. Finally, create a **BankingApplication** class (Main) to demonstrate the polymorphic behavior by creating instances of both **SavingsAccount** and **CurrentAccount**, testing deposit and withdrawal operations, and displaying account details.

Q8. Write a Java program that illustrates the concepts of interfaces, method overriding, and method overloading. Begin by defining an interface named **Vehicle** with two abstract methods: **accelerate()** and **brake()**. Then, create two classes, **Car** and **Bicycle**, both of which implement the **Vehicle** interface. In the **Car** class, override the **accelerate()** and **brake()** methods to print appropriate messages indicating the acceleration and braking actions for a car. Similarly, in the **Bicycle** class, override the same methods to display messages specific to a bicycle's acceleration and braking. Additionally, implement method overloading in both classes by providing multiple versions of the **accelerate()** method, each accepting different parameters such as speed and duration. Finally, create a **Main** class to instantiate objects of both **Car** and **Bicycle** classes, test their overridden methods, and demonstrate method overloading by invoking different versions of the **accelerate()** method.

Q9. Design a Java program for managing student enrollment in a university, adhering to the principles of loose coupling and high cohesion. Your program should include classes for representing students (`**Student**`), courses (`**Course**`), and enrollment management (`**Enrollment**`). Ensure that the `**Enrollment**` class interacts with the other classes indirectly through an interface class (`**EnrollmentSystem**`). Implement methods for enrolling and dropping students from courses, and displaying enrollment details. Create a `**Main**` class to demonstrate the system's functionality, with appropriate error handling and user-friendly output messages. Provide comments in your code explaining how loose coupling and high cohesion are maintained throughout the implementation.