Assignment-1 for Object-Oriented Programming

Subject: CSW2 (CSE 3141)
Session: Jan to May 2025
Branch: CSE
Section: All
Course Outcomes: CO1

Create & Learning Levels: Remembering (L1), Understanding (L2), Application (L3), and Analysis (L4).

Q no.	Questions	Learning Levels
✓ Q1.	Write a Java program with a **Car** class having private fields (**make, model**), a parameterized constructor, getter, and setter methods. In the **CarTester** class, instantiate **myCar1** with values and **myCar2** with null. Print their initial details, update **myCar2** using setters, and print the updated details.	L1, L2
✓ 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.	L1, L2
✓ 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.	L2, L3 → pass
✓ Q4.	Write a program to create an **Image** class with attributes **imageWidth**, **imageHeight**, and **colorCode.** Add the required constructor, set methods, get methods, and toString method. Create the object of the image class using the default and parameterized constructor and print the details of the object.	L2, L3
✓ Q5.	Create an **ImageLibrary,** which contains a set of image objects (from Q4) and operations such as searching an image, inserting an image, and getting an image. Create an **ImageController** class to manage the program execution and call the methods to create and manipulate images.	L3, L4 → an
✓ Q6.	Develop a Java-based College Management System to model the relationship between colleges and students. Create a **College** class with attributes **collegeName** and **collegeLoc**, and a **Student** class with **studentId**, **studentName**, and a reference to a **College** object. Implement a constructor in **Student** to initialize these attributes and a **displayStudentInfo()** method to print student and college details. In the **MainApp**	L3, L4

	class, instantiate at least two **College** and **Student** objects, enroll each student in one of the colleges, and display all details using appropriate methods.	
✓ Q7.	Develop a Java program for a library system using encapsulation, abstraction, and inheritance. Create an abstract **LibraryResource** class with private attributes (**title**, **author**), a constructor, getters, setters, and an abstract **displayDetails()** method. Extend it into **Book**, **Magazine**, and **DVD** classes, each adding a specific attribute (**pageCount**, **issueDate**, **duration**), along with constructors, getters, setters, and overridden **displayDetails()** methods. In the **LibrarySystem** class, instantiate various resources and call **displayDetails()** to display their information.	L3, L4 → Ex
✓ Q8.	Develop a Java banking application using polymorphism with three classes: **Account**, **SavingsAccount**, and **CurrentAccount**. The abstract **Account** class has private attributes (**accountNumber**, **balance**) and abstract methods for **deposit** and **withdrawal**. **SavingsAccount** adds an **interestRate** attribute, overrides **deposit** to calculate interest, and ensures sufficient balance in **withdrawal**. **CurrentAccount** introduces an **overdraftLimit** and overrides **withdrawal** to check this limit. In the **BankingApp** class, instantiate both account types, perform transactions, and display account details to demonstrate polymorphism.	L3, L4
✓ Q9.	Write a Java program demonstrating interfaces, method overriding, and method overloading. Define a **Vehicle** interface with abstract methods **accelerate()** and **brake()**. Implement **Car** and **Bicycle** classes that override these methods with specific messages for acceleration and braking. Introduce method overloading in both classes by defining multiple **accelerate()** methods with different parameters (e.g., **speed**, **duration**). In the **VehicleApp** class, instantiate **Car** and **Bicycle** objects, test overridden methods, and invoke overloaded **accelerate()** methods to showcase polymorphism.	L2, L4
✓ Q10.	Design a Java program for university student enrollment, ensuring loose coupling and high cohesion. Create **Student** and **Course** classes, and an **Enrollment** class that interacts with them through an **EnrollmentSystem** interface. Implement methods for enrolling and dropping students from courses, and displaying enrollment details. In the **MainEnrollApp** class, demonstrate functionality by managing student enrollments. Use comments to explain how the design maintains loose coupling (by relying on interfaces) and high cohesion (by keeping related functionalities within appropriate classes).	L3, L4
	-END-	

✓ Student
✓ course

enroll{
-st-
-c-
}

System{
<?Any: enroll
}