University of Windsor SCHOOL OF COMPUTER SCIENCE	**COMP-2120, Fall 2024** **Object Oriented Programming Using Java** **LAB-4** **TOTAL MARKS: 10**

LAND ACKNOWLEDGEMENT

The School of Computer Science at the University of Windsor *sits on the Traditional Territory of the Three Fires Confederacy of First Nations*. We acknowledge that this is the beginning of our journey to understanding the Significance *of the history of the Peoples of the Ojibway, the Odawa, and the Pottawatomie*.

SUBMISSION DEADLINE: OCT-25, 2024

GENERAL INFORMATION

Welcome to the Lab Session of COMP-2120 where you will learn object-oriented programming (OOP) using Java. Please, arrive early to the lab and get settled to start working on the lab. Arriving 20 minutes after the start of the lab will be considered late and can affect your lab participation point. There are eight lab exercises you need to complete throughout the course. Each week on Thursday morning we will publish the lab exercise. You need to upload your code by Thursday, 10 AM in the next week.

LAB GRADING AND ASSESSMENT

You need to explain your code to TAs in the lab to receive the marks. YOU MUST demonstrate your results and explain the code to the TAs to receive marks. Thus, attending the lab sessions is mandatory. Note that your TAs are here in the Lab to help you learn. Feel free to ask questions and talk to them. Remember, the total Labs is worth 16% of your final grade.

Lab-4

In this assignment, you will implement a set of classes representing different roles in a software development firm. You will practice inheritance and polymorphism by creating various employees with different roles and overriding specific methods to match the expected behaviour.

You are given the **Main** class and the **expected output**. Your task is to create the necessary classes, ensuring proper inheritance and method overriding to match the expected behaviour.

Provided Main Class:

```
public class Main {
    public static void main(String[] args) {
        // Create objects
        BackendDeveloper backendDev = new BackendDeveloper("Alice", 1001, 6000.0, "Backend Developer", "Java");
        AutomationTester tester = new AutomationTester("Bob", 1002, 5500.0, "Automation Tester", "Selenium");
        ProjectManager manager = new ProjectManager("Carol", 1003, 8000.0, "Project Manager", 5);
        BackendDeveloper backendDev2 = new BackendDeveloper("Mike", 1004, 6500.0, "Backend Developer", "Java");

        // Polymorphism: Treating each as an Employee
        Employee emp1 = backendDev;
        Employee emp2 = tester;
        Employee emp3 = manager;
        Employee emp4 = backendDev2;

        // Calling performDuties() using polymorphism
        emp1.performDuties();  // This should call BackendDeveloper's performDuties()
        emp2.performDuties();  // This should call AutomationTester's performDuties()
        emp3.performDuties();  // This should call ProjectManager's performDuties()
        emp4.performDuties();  // This should call BackendDeveloper's performDuties()

        // Calling specific methods of each class
        backendDev.optimizeDatabase();  // Specific method for BackendDeveloper
        tester.writeAutomationScript(); // Specific method for AutomationTester
        manager.scheduleMeeting();      // Specific method for ProjectManager
    }
}
```

Expected Output:

```
Alice is building APIs and ensuring database integration.
Bob is writing scripts to automate test cases.
Carol is managing teams and overseeing project deadlines.
Mike is building APIs and ensuring database integration.
Alice is optimizing database queries for better performance.
Bob is writing an automation script for regression testing.
Carol is scheduling a team meeting.
```

Tasks:

1. **Implement the Class Hierarchy**: (6 points)

- Based on the Main class, you will need to implement the following classes:
    - `Employee (base class)`
    - `Developer (inherits from Employee)`
    - `Tester (inherits from Employee)`
    - `ProjectManager (inherits from Employee)`
    - `BackendDeveloper (inherits from Developer)`
    - `AutomationTester (inherits from Tester)`

2. **Override performDuties()**: (2 points)

- In each subclass, override the `performDuties()` method to match the expected behaviour of each role:

- BackendDeveloper: Prints a message about building APIs and database integration.
- AutomationTester: Prints a message about writing scripts for test automation.
- ProjectManager: Prints a message about managing teams and scheduling meetings.

3. **Implement Unique Methods**: (2 points)

- Implement the following specific unique methods for each class:
  - BackendDeveloper: `optimizeDatabase()` should print a message about optimizing database queries.
  - AutomationTester: `writeAutomationScript()` should print a message about writing automation scripts for regression testing.
  - ProjectManager: `scheduleMeeting()` should print a message about scheduling team meetings.

N.B:

You should carefully examine the **Expected Output** section to determine the exact messages that should be printed by the methods. The messages for each employee should reflect their specific duties, as seen in the output, and include the employee's name and role-related information.

WHAT DO YOU NEED TO DO?

1. Complete the program.
2. Attend the lab. Explain the code to TAs.
3. TAs can modify the input to your program.