**UGANDA CHRISTIAN UNIVERSITY**



**NAME: MABIRA CONRAD WILLIAM**

**ACCESS NUMBER: A94170**

**COURSE: BACHELOR OF SCIENCE IN COMPUTER SCIENCE**

**COURSE UNIT: SOFTWARE CONSTRUCTION**

1. **Single Responsibility Principle (SRP):**
   * In the provided code, the **Employee** class has the responsibility of representing an employee's data, such as name and role. The **Report** class is responsible for generating reports based on employee roles. The **BonusCalculator** class calculates bonuses for employees. Each class has a single responsibility, but the **Report** class has multiple reasons to change, violating SRP.
2. **Open/Closed Principle (OCP):**
   * The provided code does not strictly adhere to the OCP. For example, if a new role is introduced, such as "Analyst," modifications will be required in the **Report** and **BonusCalculator** classes to handle this new role. Ideally, classes should be open for extension but closed for modification, allowing for new functionality to be added without changing existing code.
3. **Liskov Substitution Principle (LSP):**
   * The **Manager** and **Developer** classes inherit from the **Employee** class, which is appropriate given that they represent specialized types of employees. However, the **Report** class directly depends on specific roles ("Manager" and "Developer"), which might lead to violations of LSP if, for example, a subclass of **Manager** or **Developer** behaves differently from what **Report** expects.
4. **Interface Segregation Principle (ISP):**
   * The code doesn't explicitly implement interfaces, but we can consider the **ReportGenerator** and **BonusCalculator** as interfaces. However, the **Report** class depends on the entire **Employee** object, violating ISP. Ideally, clients should not be forced to depend on methods or properties they don't use.
5. **Dependency Inversion Principle (DIP):**
   * The code demonstrates a lack of DIP since high-level modules (**Report** and **BonusCalculator**) depend directly on low-level modules (**Employee**, **Manager**, and **Developer**). Instead, they should depend on abstractions. In other words, **Report** and **BonusCalculator** should depend on interfaces (**ReportGenerator** and **BonusCalculator**), not concrete implementations.

**Refactoring Plan**

* Extract report generation logic from the **Report** class into separate **ReportGenerator** implementations for each type of report (e.g., **ManagerReportGenerator**, **DeveloperReportGenerator**).
* Extract bonus calculation logic from the **BonusCalculatorImpl** class into separate **BonusCalculator** implementations for each type of employee (e.g., **ManagerBonusCalculator**, **DeveloperBonusCalculator**).
* Define interfaces (**ReportGenerator** and **BonusCalculator**) to provide contracts for report generation and bonus calculation.
* Refactor classes to depend on these interfaces rather than concrete implementations.
* Ensure that the **Employee**, **Manager**, and **Developer** classes adhere to the SRP and LSP.