
💻 Refactoring Assignment: The Simple Reporting Tool (SRT) (Updated)

This assignment requires you to refactor an existing Java codebase for a simple reporting tool. The current code is functional but suffers from several design smells that make it hard to maintain, extend, and test.

Part 1: Initial Code Analysis

You are given the following initial Java code for a ReportGenerator class and a supporting DataProcessor class.

Initial ReportGenerator.java

Java

```
public class ReportGenerator {

    private String reportType; // e.g., "PDF", "CSV"
    private String rawData;

    public ReportGenerator(String rawData, String reportType) {
        this.rawData = rawData;
        this.reportType = reportType;
    }

    public String generateReport() {
        if (rawData == null || rawData.isEmpty()) {
            return "Error: No data provided.";
        }

        // Processing Logic - Very long method!
        String processedData = DataProcessor.processData(rawData);
```

```
        String header = "";
        String body = "";
        String footer = "--- End of Report ---";

        // Logic for PDF and CSV report formats is tightly coupled
        if ("PDF".equalsIgnoreCase(reportType)) {
            header = "--- PDF Report Header ---";
            body = "PDF formatted data: " + processedData.toUpperCase();
        } else if ("CSV".equalsIgnoreCase(reportType)) {
            header = "--- CSV Report Header ---";
            // The next line simulates complex CSV formatting logic
            String[] dataLines = processedData.split(",");
            StringBuilder csvContent = new StringBuilder();
            for (String line : dataLines) {
                csvContent.append("\"").append(line.trim()).append("\",");
            }
            body = "CSV Data:\n" + csvContent.toString();
        } else {
            return "Error: Unsupported report type: " + reportType;
        }

        return header + "\n" + body + "\n" + footer;
    }

    // Unused method left over from previous development
    public void printStatus() {
        System.out.println("Generator initialized.");
    }

    // New attribute added for additional refactoring task
    public static final String DEFAULT_FOOTER = "--- End of Report ---";
}
```

Initial DataProcessor.java

Java

```
public class DataProcessor {
    public static String processData(String data) {
        // Simulates complex, independent data transformation logic
        return data.replaceAll(" ", "_").toLowerCase();
    }
}
```

Part 2: Refactoring Tasks 🛠️

Your primary goal is to apply the following **required refactoring techniques** to the codebase to improve its design based on the **Single Responsibility Principle (SRP)** and the **Open/Closed Principle (OCP)**.

Required Refactoring Method	Class/Method to Apply To	Design Problem Addressed
1. Extract Method	ReportGenerator.generateReport()	**Long Method** (The method is too large and performs too many steps).
2. Replace Conditional with Polymorphism	ReportGenerator.generateReport()	**Switch Statement/Conditional Complexity** (The large if/else if structure violates OCP).
3. Move Method/Class	Logic within ReportGenerator	**Large Class** and **Feature Envy** (Report formatting logic is too complex for the generator).
4. Extract Interface	New classes created from polymorphism.	To support dependency inversion and testing.
5. Remove Unused Code	ReportGenerator.printStatus()	**Dead Code**.
6. Encapsulate Field (New)	ReportGenerator.reportType	**Public/Exposed Field** (To enforce controlled access and type safety).
7. Replace Magic Number with Symbolic Constant (New)	Hardcoded strings in ReportGenerator (e.g., "PDF", "CSV")	**Magic Strings** (Improves readability and maintainability).
8. Self Encapsulate Field (New)	ReportGenerator.rawData	To allow for future subclassing or more complex data validation without changing

		clients.

Part 3: Deliverables 📝

You must submit the following items for this assignment:

1. **Initial Class Diagram:** Draw the **UML Class Diagram** for the initial code structure (ReportGenerator and DataProcessor).
2. **Refactored Java Code:** The complete, refactored Java source code, split into the appropriate new classes and packages.
3. **Refactored Class Diagram:** Draw the **UML Class Diagram** for the final, refactored code structure. It must show all new classes, interfaces, and their relationships (inheritance, composition, etc.).
4. **Refactoring Report:** A short document (500-750 words) explaining **where** each of the **eight** required refactoring techniques was applied and **how** it improved the code's design (specifically referencing **SRP**, **OCP**, and **Maintainability**).

Would you like me to provide a high-level suggestion on **how** to tackle the "Replace Conditional with Polymorphism" step, as it's the most complex refactoring task?