SOLID Principles: Assignment | Single Responsibility Principle (SRP) in Practice

**Audience:** Mid-level Java developers & solution architects  
 **Goal:** Diagnose SRP violations in a “kitchen-sink” class, explain why they hurt, then redesign the codebase so every class has one—and only one—reason to change.

### **1 Starter Code**

You receive **two files** in src/main/java/aibadcode/legacy/:

EmployeeManager.java // does payroll, persistence, reporting

OrderProcessor.java // validates, stores, emails, logs

Both classes compile but combine several responsibilities (extract from the “Before Applying SRP” snippets above).

### **2 Part A | Problem Discovery**

1. **Static review**
   * Skim each class; list every responsibility you spot (business rule, I/O, formatting, etc.).
   * Fill analysis/srp\_violations.md with a table:

| **#** | **Class** | **Responsibility** | **Why it violates SRP** | **Real-world impact** |
| --- | --- | --- | --- | --- |

1. **Write one failing test per responsibility** (JUnit) that illustrates a pain-point, e.g.  
   * Changing report format breaks salary logic.
   * Mock DB outage causes reporting to fail.

Place tests in src/test/java/aibadcode/legacy/.

### **3 Part B | Refactor to SRP**

*Move the code to the new package aibadcode.clean.\*.*

1. **Extract Classes**
   * From EmployeeManager produce:  
     + SalaryCalculator
     + EmployeeRepository
     + EmployeeReportService
   * From OrderProcessor produce:  
     + OrderValidator
     + OrderRepository
     + CustomerNotifier
2. **Apply Dependency-Injection**
   * Create a coordinator class (EmployeeFacade, OrderService) that receives the new single-purpose collaborators through its constructor.
3. **Refactor tests** so they now pass against the clean API (green bar).

### **4 Part C | Design Justification (Architect Perspective)**

Create docs/design\_notes.md answering:

* For each extracted class:  
  + “What is its single reason to change?”
  + “Which other classes depend on it now?”
* How SRP improved:  
  + **Maintainability** (give a concrete change scenario).
  + **Testability** (show which tests became easier/smaller).
  + **Coupling** (describe any interfaces introduced).

### **5 Deliverables**

swift

CopyEdit

/analysis/srp\_violations.md ← problem list

/src/main/java/aibadcode/clean/\*\* ← refactored code

/src/test/java/aibadcode/clean/\*\* ← updated passing tests

/docs/design\_notes.md ← architectural rationale

/README.md ← build & run instructions

### **6 Grading Rubric (100 pts)**

| **Weight** | **Excellent** | **Satisfactory** |
| --- | --- | --- |
| 30 pts | Violations documented across UI, persistence, logic | Only obvious violations listed |
| 30 pts | Refactor fully respects SRP; each class ≤ 1 responsibility | Classes still mix concerns |
| 20 pts | Tests illustrate problems before & pass after | Tests missing or brittle |
| 20 pts | Clear design notes linking SRP → benefits | Superficial justification |

Bonus (+10 pts): introduce interfaces + DI to enable mocking in tests.

### **7 Suggested Timeline**

| **Timebox** | **Task** |
| --- | --- |
| 30 min | Code walkthrough & fill violation table |
| 45 min | Write failing tests |
| 90 min | Incremental refactor |
| 30 min | Green tests, write design notes, polish README |

### **8 Reference Models**

*The following helper entities are supplied for your convenience (copy or import as needed):*

* Employee, Order, and OrderItem classes from the brief.
* Stubs for EmailSender, DatabaseClient, Logger.

Use them to keep extracted classes laser-focused on a single purpose.

### **9 Turn-in**

Submit a zip or Git repository link to the instructor. CI must show all tests green (mvn test or ./gradlew test) before the deadline.

Good luck—prove you can turn tangled AI output into clean, maintainable architecture by mastering the **Single Responsibility Principle**!