Part (6): Bad Design Example

```
class BadReportService {
    public void export(Report report, String format) {
        if (format.equals("TEXT")) {
            System.out.println("Exporting as TEXT: " +
report.getTitle());
        } else if (format.equals("JSON")) {
            System.out.println("Exporting as JSON: " +
report.getTitle());
        } else if (format.equals("PDF")) {
            System.out.println("Exporting as PDF: " +
report.getTitle());
        }
 else if (format.equals("XML")) {
    System.out.println("Exporting as XML: " + report.getTitle());
}
    }
}
```

1. **Which principles does this violate?**

a.SRP this class has more than one responsibility It decides which format to export and It performs the export itself.
Thats means it has more than one reason to change which violates SRP.

b. OCP this class is not open for extension u must modify it whenever a new export format is added , Every time you add a new format u must change the **`if/else if`** logic inside the method which violates OCP.

2. **What happens when you add a new export type (like XML)?**

If i decide to add new XML export u must edit the class again that means:
More code duplication, more chances of bugs , hard to maintenance and testing, tight coupling.
Every new format breaks the idea “open for extension, closed for modification.”

3. **How would you redesign it to follow SRP and OCP?**

Firstovall create an Interface

```
public interface ReportExporter {
    void export(Report report);
}
```

one common method for all export types

Then create one class per Export Type

```
public class TextReportExporter implements ReportExporter {
    public void export(Report report) {
        System.out.println("Exporting report as TEXT...");
        System.out.println("Title: " + report.getTitle());
        System.out.println("Content: " + report.getContent());
    }
}

public class JsonReportExporter implements ReportExporter {
    public void export(Report report) {
        System.out.println("Exporting report as JSON...");
        System.out.println("{ \"title\": \"" + report.getTitle() +
"\", \"content\": \"" + report.getContent() + "\" }");
    }
}

public class PdfReportExporter implements ReportExporter {
    public void export(Report report) {
        System.out.println("Exporting report as PDF...");
        System.out.println("[PDF] Title: " + report.getTitle());
        System.out.println("[PDF] Content: " +
report.getContent());
    }
}
```

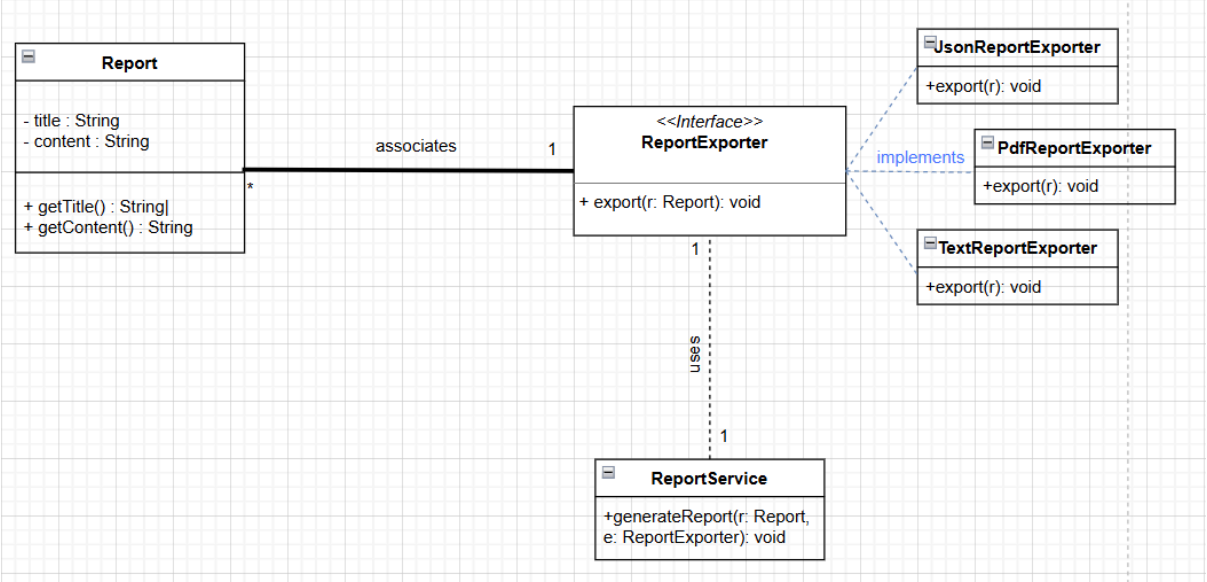
Each class now has a single responsibility.

Then Use a Service Class to Manage Reports

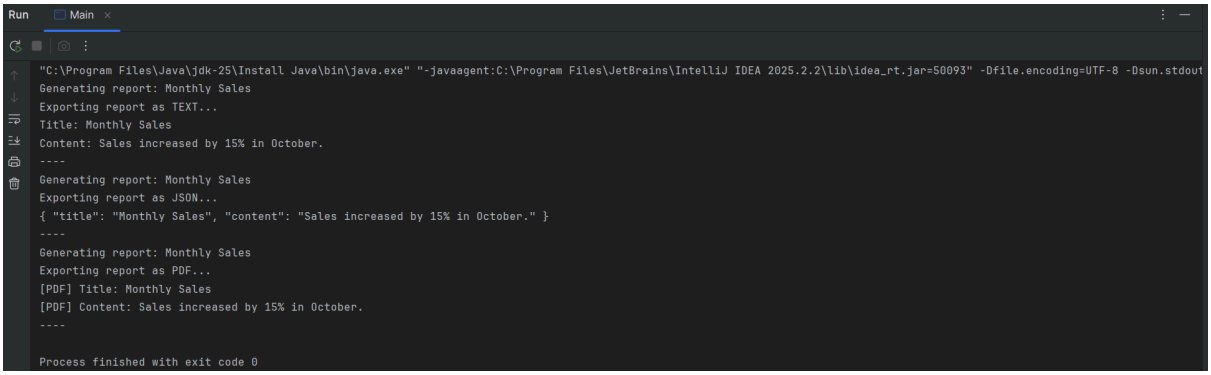
```
public class ReportService {
    public void generateReport(Report report, ReportExporter exporter) {
        System.out.println("Generating report: " + report.getTitle());
        exporter.export(report);
    }
}
```

ReportService only controls the workflow.
It depends on the interface, not on any specific exporter.
U can add new exporters (like XML, Excel, CSV) without changing this class.

Part (7) : UML Class Diagram Design



Part (8): Program Execution (Consol Screenshot)



Did you modify any existing code? Why not?

No, I didn't modify any existing code cuz we use Open/Closed Principle (OCP) , That means I can add a new feature like the PdfReportExporter class without changing any existing once.

all exporters share the same interface ReportExporter then there is no edits required.

Reflection Questions

The Single Responsibility Principle (SRP) improves maintainability by keeping each class focused on one job, so changes are easier and safer.
The Open/Closed Principle (OCP) makes adding new features easier because I can extend the system with new classes instead of editing old ones.
Interface abstraction supports flexibility by letting different exporters use the same rules, so they're easy to switch or add.
If the Report class also handled exporting, it would mix data with output logic, making the system harder to update.
From the class diagram, I saw how interfaces keep things flexible and reduce connections between parts.
If my company wanted Excel or XML export, I'd just create new classes XmlExporter that follow the same interface like ExcelExporter .
SRP and OCP help different teams developers, analysts & designers work at the same time without breaking each other's code.