Sunday, 6 March 2022

Java Design Patterns & SOLID Principles

SOLID Design Principles

SINGLE RESPONSIBILITY PRINCIPLE

What Is It?

- Principle stating that there should never be more than one reason for a class to change.
- Class should be **focused**, provide **single functionality** and address a **specific concern**.

What Does It Mean?

- When you have multiple reasons for a class to change (e.g., protocol change, message format change, communication security change), avoid changing the class itself and **create new class for each change**.
- It allows to change the code in an organised way.

OPEN-CLOSED PRINCIPLE

What Is It?

- Principle stating that software entities (classes, modules, methods, etc.) should be **open for extension**, but **closed for modification**.

What Does It Mean?

- **Open for extension**: you are able to extend the existing behaviour.
- **Closed for modification**: the existing code should remain unchanged.
- Base class that is already written and tested should not be changed but rather extended using the derived class which can derive from base and override methods.

LISKOV SUBSTITUTION PRINCIPLE

What Is It?

- Principle stating that we should be able to substitute base class objects with child class objects and this should not alter behaviour/characteristics of the program.

What Does It Mean?

- If the base class was providing certain functionality/behaviour and it is substituted by a child class, the behaviour should not be altered.

INTERFACE SEGREGATION PRINCIPLE

What Is It?

- Principle stating that clients should not be forced to depend upon interfaces (and methods defined there) that they do not use.

What Does It Mean?

- Avoid classes implementing methods from interface that do not make sense.
- In other words, **avoid Interface Pollution**: avoid large interfaces and unrelated method in the same interface.
- **Signs of Interface Pollution**:
  - Classes have empty method implementations.
  - Method implementations throw UnsupportedOperationException (or similar).
  - Method implementations return null or default/dummy values.
- All in all, write highly cohesive and concise interfaces!

DEPENDENCY INVERSION PRINCIPLE

What Is It?

- Principle stating the following:
  - High level modules should not depend upon low level modules - both should depend upon abstractions.
  - Abstractions should not depend upon details - details should depend upon abstractions.

What Is Dependency?

- When your code has a dependency on the object defined outside of your code - this object is a dependency.

What Does It Mean?

- Instead of tightly coupling the high level modules (i.e., modules that implement business rules) and low level modules (i.e., basic functionality modules like writing to disk, converting java objects to JSON), make them depend on **abstractions** (e.g, abstract classes, interfaces).
- Instead of writing the code that transforms your report specifically into JSON format, pass the formatter and writer interfaces to your method as parameters and write the code using those interfaces.
- Interfaces for different formats and different writers can be then passed as parameters allowing you to easily change where and how you process your report instead of changing the code all the time.
- Instead of instantiating dependencies ourselves, let others give/pass the dependencies.