2. O - Open/Closed Principle (OCP)

Resources

- Video → 1. SOLID Principles with Easy Examples (Hindi) | OOPs SOLID Principles - Low Level Design

What is the Open/Closed Principle (OCP)?

> *Open/Closed Principle(OCP) states that **"A class should be open for extension but closed for modification".***

This means that new functionality can be added through extension (inheritance, composition, interfaces) rather than by modifying existing code. As existing code is already tested and deployed in production, adding modifications would introduce an additional risk and require additional testing effort.

Code Example: Violating OCP

```java
// Responsibility: Managing Database Operations only
public class InvoiceDaoOld {

    Invoice invoice;

    public InvoiceDaoOld(Invoice invoice) {
        this.invoice = invoice;
    }

    public void saveToDB() {
        // Save into the DB the invoice
        System.out.println("Saving to DB...");
    }
}
```

```java

// Responsibility: Managing Database Operations and File Operations
// BAD: This class violates OCP - needs modification for every new kind of save function
public class InvoiceDao {

    Invoice invoice;

    public InvoiceDao(Invoice invoice) {
        this.invoice = invoice;
    }

    public void saveToDB() {
        // Save into the DB the invoice
        System.out.println("Saving to DB...");
    }

    // BAD: This design violates OCP
    // Every time we add a new save function, we need to modify the existing InvoiceDao class
    public void saveToFile() {
        // Save into the file
        System.out.println("Saving to file...");
    }
}

// Usage example - showing the problem
public class Demo {
    public static void main(String[] args) {
        Invoice invoice = new Invoice(new Marker("name", "color", 10, 2020), 10);
        invoice.calculateTotal();

        InvoiceDao databaseFileSave = new InvoiceDao(invoice);
        databaseFileSave.saveToDB(); // Save to DB
        databaseFileSave.saveToFile(); // Save to File

        // Problem: If we want to add a new function like saveToMongoDB(),
        // we need to modify InvoiceDao and all its derived classes(if exists)
        // This violates the "closed for modification" part of OCP
    }
}
```

Problems with the Above Code

- **Modification Required:** Every time we add a new type of save functionality(let's say `saveToMongoDB()`), we must modify the existing `InvoiceDao` class.
- **Risk of Breaking Existing Code:** Changes might introduce bugs in working functionality, i.e., the `InvoiceDao` class that is already deployed in the production environment.
- **Testing Issues:** Need to retest all existing functionality when adding new save operations.

Code Example: Follows OCP

```java
// GOOD: Following OCP using interfaces and polymorphism
public interface InvoiceDao {
    public void save();
}

// Concrete implementation for DatabaseInvoiceDao
public class DatabaseInvoiceDao implements InvoiceDao {
    Invoice invoice;

    public DatabaseInvoiceDao(Invoice invoice) {
        // set the invoice
        this.invoice = invoice;
    }

    @Override
    public void save() {
        // Save into the DB the invoice
        System.out.println("Saving to DB...");
    }
}

// Concrete implementation for FileInvoiceDao
// NEW File Save Operation: An extension without modification!
public class FileInvoiceDao implements InvoiceDao {

    Invoice invoice;

    public FileInvoiceDao(Invoice invoice) {
        // set the invoice
        this.invoice = invoice;
    }

    @Override
    public void save() {
        // Save into the file the invoice
        System.out.println("Saving to file...");
    }
}

// Usage demonstrating OCP compliance
public class Demo {
    public static void main(String[] args) {
        Invoice invoice = new Invoice(new Marker("name", "color", 10, 2020), 10);
        invoice.calculateTotal();

        InvoiceDao databaseInvoiceDao = new DatabaseInvoiceDao(invoice);
        databaseInvoiceDao.save(); // Save to DB

        InvoiceDao fileInvoiceDao = new FileInvoiceDao(invoice);
        fileInvoiceDao.save(); // Save to File

        // The system is:
        // - OPEN for extension (new save functions can be added)
        // - CLOSED for modification (existing code remains unchanged)
    }
}
```

Key Benefits of the Refactored Code

- **Reduced risk:** Existing tested code remains unchanged.
- **Better maintainability:** New features don't break existing functionality.
- **Improved flexibility:** Easy to add new behaviors without touching existing code.
- **Enhanced testability:** New extensions can be tested independently.
- **Supports polymorphism:** Enables dynamic behavior through interfaces and inheritance.

Summary

By adhering to the Open/Closed Principle (OCP), developers can construct software architectures that are both adaptable and resilient, allowing them to evolve and expand without compromising existing functionalities.

This principle emphasizes the importance of abstraction and the creation of well-defined interfaces, which not only accommodate current requirements but also anticipate future enhancements.

Systems designed with OCP in mind enhance robustness and maintainability, increase the longevity and scalability of software solutions, and reduce the risk of introducing defects during changes.