Design Patterns

Table of contents

- 1.Problem Statement: You are developing a game where you need to create different types of enemies (aliens, robots, etc.). Currently, you have a single method that creates enemies based on a string parameter. However, as the game grows, you need a more flexible and extensible way to create new enemy types without modifying the existing code.

```
public class EnemyFactory {
    public Enemy createEnemy(String type) {
        if (type.equals("Alien")) {
            return new Alien();
        } else if (type.equals("Robot")) {
            return new Robot();
        }
        // More cases...
        return null;
    }
}
```

a) Identify the problem with the current approach.

b) Suggest a design pattern that can solve this problem.

c) Provide a high-level UML diagram or pseudocode to illustrate your solution.

- 2. Problem Statement: In a text editor application, you want to provide different formatting options to the text, such as bold, italic, underline, etc. You want to be able to combine these formatting options dynamically without creating a separate class for each combination.

```java
public interface TextComponent {
    void render();
}

public class PlainText implements TextComponent {
    private String text;

    public PlainText(String text) {
        this.text = text;
    }

    @Override
    public void render() {
        System.out.println(text);
    }
}
```

a) Identify the issue with the current implementation.

b) Suggest a design pattern that can solve this problem.

c) Provide a high-level UML diagram or pseudocode to illustrate your solution.

- 3. Problem Statement: You are developing a complex system with multiple subsystems that need to interact with each other. The client code needs to work with these subsystems, but the complexity of their interactions makes it difficult to understand and maintain the code.

```java
// Client code
public class Client {
    public void doSomething() {
        SubsystemA subsystemA = new SubsystemA();
        SubsystemB subsystemB = new SubsystemB();
        SubsystemC subsystemC = new SubsystemC();

        // Complex interactions between subsystems
        subsystemA.operation1();
        subsystemB.operation2(subsystemA.getResult());
        subsystemC.operation3(subsystemA.getResult(), subsystemB.getResult());
    }
}
```

a) Identify the problem with the current approach.

b) Suggest a design pattern that can solve this problem.

c) Provide a high-level UML diagram or pseudocode to illustrate your solution.

- 4. Problem Statement: You are working on a system that requires accessing remote objects or resources. However, you want to provide a local representation or placeholder for these remote objects to simplify the client code and handle potential performance issues.

```
// Remote object interface
public interface RemoteObject {
    void performOperation();
}

// Client code
public class Client {
    public void useRemoteObject(RemoteObject remoteObject) {
        // Code to use the remote object directly
        remoteObject.performOperation();
    }
}
```

a) Identify the potential issues with the current approach.

b) Suggest a design pattern that can solve this problem.

c) Provide a high-level UML diagram or pseudocode to illustrate your solution.

- 5. Problem Statement: You are developing a system that needs to create different types of products, each with its own set of related objects. For example, a product might consist of various components, accessories, and documentation. You want to provide a way to create these families of related objects in a consistent and flexible manner.

```
// Product interfaces
public interface Product { /* ... */ }
public interface Component { /* ... */ }
public interface Accessory { /* ... */ }
public interface Documentation { /* ... */ }

// Concrete product classes
public class ProductA implements Product { /* ... */ }
public class ProductAComponent implements Component { /* ... */ }
```

```
public class ProductAAccessory implements Accessory { /* ... */ }
public class ProductADocumentation implements Documentation { /* ... */ }

// Similar classes for ProductB, ProductC, etc.
```

a) Identify the potential issue with the current approach.

b) Suggest a design pattern that can solve this problem.

c) Provide a high-level UML diagram or pseudocode to illustrate your solution.

- 6. Problem Statement: You are working on a project that involves managing different types of ducks. Each duck has its own behavior for quacking and flying. Initially, you have a simple implementation where each duck type is a separate class that inherits from a base Duck class.

```
public abstract class Duck {
    public abstract void quack();
    public abstract void fly();
}

public class MallardDuck extends Duck {
    @Override
    public void quack() {
        System.out.println("Quack");
    }

    @Override
    public void fly() {
        System.out.println("Flying");
    }
}

public class RubberDuck extends Duck {
    @Override
    public void quack() {
        System.out.println("Squeak");
    }

    @Override
    public void fly() {
        // Rubber ducks cannot fly
        throw new UnsupportedOperationException();
```

```
    }
}
```

a) Identify the potential issue with the current approach.
b) Suggest a design pattern that can solve this problem.
c) Provide a high-level UML diagram or pseudocode to illustrate your solution.

- 7. Problem Statement: You are developing a system that needs to manage different types of weather data sources, such as text files or databases. You want to provide a way to add new features like encryption or compression to these data sources without modifying their existing code.

```
public interface DataSource {
    void writeData(String data);
    String readData();
}

public class FileDataSource implements DataSource {
    private String fileName;

    public FileDataSource(String fileName) {
        this.fileName = fileName;
    }

    @Override
    public void writeData(String data) {
        // Write data to file
    }

    @Override
    public String readData() {
        // Read data from file
        return "";
    }
}

public class DatabaseDataSource implements DataSource {
    private String connectionString;

    public DatabaseDataSource(String connectionString) {
        this.connectionString = connectionString;
```

```
    }

    @Override
    public void writeData(String data) {
        // Write data to database
    }

    @Override
    public String readData() {
        // Read data from database
        return "";
    }
}
```

a) Identify the potential issue with the current approach.

b) Suggest a design pattern that can solve this problem.

c) Provide a high-level UML diagram or pseudocode to illustrate your solution.

- 8. Problem Statement: You are working on a system that involves managing user authentication and authorization. You want to provide a way to add additional security checks or logging mechanisms without modifying the existing authentication and authorization code.

```
public interface AuthenticationService {
    boolean authenticate(String username, String password);
}

public class BasicAuthenticationService implements AuthenticationService {
    @Override
    public boolean authenticate(String username, String password) {
        // Perform authentication logic
        return true;
    }
}

public interface AuthorizationService {
    boolean isAuthorized(String username, String resource);
}

public class BasicAuthorizationService implements AuthorizationService {
```

```
    @Override
    public boolean isAuthorized(String username, String resource) {
        // Perform authorization logic
        return true;
    }
}
```

a) Identify the potential issue with the current approach.

b) Suggest a design pattern that can solve this problem.

c) Provide a high-level UML diagram or pseudocode to illustrate your solution.

- 9. Problem Statement: You are developing a system that involves managing different types of notifications (email, SMS, push notifications, etc.). You want to provide a way to add new notification channels without modifying the existing notification code.

```
public interface NotificationService {
    void sendNotification(String message);
}

public class EmailNotificationService implements NotificationService {
    @Override
    public void sendNotification(String message) {
        // Send email notification
    }
}

public class SMSNotificationService implements NotificationService {
    @Override
    public void sendNotification(String message) {
        // Send SMS notification
    }
}
```

a) Identify the potential issue with the current approach.

b) Suggest a design pattern that can solve this problem.

c) Provide a high-level UML diagram or pseudocode to illustrate your solution