
Design Patterns Lab – Java Implementation

Patterns Covered: Adapter, Flyweight, Proxy, Facade
Objective: Implement each design pattern using simplified Java classes and simulate behavior using printed output.

Part 1: Adapter Pattern – Legacy Sensor Integration

Scenario:
You're developing a weather dashboard that needs to integrate legacy sensors with different APIs.

Task:
Create adapter classes that allow legacy sensors to conform to a common interface.

Suggested Classes:

- Sensor (interface) – defines getData()
- TemperatureSensorLegacy – has readTemp()
- HumiditySensorLegacy – has getHumidityLevel()
- TemperatureSensorAdapter – adapts TemperatureSensorLegacy to Sensor
- HumiditySensorAdapter – adapts HumiditySensorLegacy to Sensor
- Main – uses the adapters to print sensor data

Expected Output:

```
Temperature Sensor: 23°C
Humidity Sensor: 50%
```

Part 2: Flyweight Pattern – Text Editor Formatting

Scenario:
A text editor needs to store and render many characters with shared formatting (font, size, style).

Task:
Use the Flyweight pattern to share TextFormat objects between characters.

Suggested Classes:

- TextFormat – contains font, size, style
- TextFormatFactory – returns shared TextFormat objects
- FormattedCharacter – stores a character and a TextFormat
- Main – prints characters and their formatting

Expected Output:

```
Char: H | Format: Arial, 12pt, Bold
Char: e | Format: Arial, 12pt, Bold
Char: l | Format: Arial, 12pt, Bold
Char: l | Format: Times, 12pt, Italic
Char: o | Format: Times, 12pt, Italic
```

Part 3: Proxy Pattern – Database Access Control

Scenario:
Restrict database access based on user role using a proxy class.

Task:
Create a proxy that allows only "admin" to access the database and blocks "guest" users.

Suggested Classes:

- Database (interface) – defines query(String sql)
- RealDatabase – executes the actual query
- DatabaseProxy – checks user role before allowing access
- Main – simulates both admin and guest users

Expected Output:

```
[Admin] Executing query: SELECT * FROM users
[Guest] Access denied for query: SELECT * FROM users
```

Part 4: Facade Pattern – E-commerce Checkout

Scenario:
You're building an e-commerce checkout system that handles inventory, payment, order logging, and shipping.

Task:
Use a facade to simplify the checkout process into a single method call.

Suggested Classes:

- InventoryService – checks item availability
- PaymentService – processes payments
- OrderService – logs the order
- ShippingService – handles shipping
- CheckoutFacade – coordinates the above services
- Main – uses CheckoutFacade to perform a full checkout

Expected Output:

```
Checking inventory for item: Book
Processing payment of $19.99
Logging order: Book
Scheduling shipping for Book
Checkout complete.
```

Submission Notes:

- Organize each pattern in its own Java package or folder.
- Use `System.out.println()` for simulating functionality.
- Keep the `Main` class clean and only for testing the pattern.
- Use interfaces and proper OOP practices wherever applicable.
