

Textile Management System



HAMZA ATIF (FA20-BSE-018)

SYED ADAN MURAD SHAH (FA20-BSE-036)

**Textile Management System:**

**Builder Pattern:**

* **Inventory Management:**

Builder Pattern can be used for creating complex textile inventory objects with various attributes (e.g., type, size, color).

* **Supplier Management:**

It can be used to create Supplier objects with different attributes.

* **Quality Control and Assurance:**

Build Quality Control objects for tracking quality checks and tests.

* **Batch Tracking:**

Create Batch objects to track the production history and quality information for each batch.

**Observer Pattern:**

* **Real-time Monitoring:**

The Observer Pattern is ideal for real-time monitoring of production processes. Each monitored parameter can be an observer, and the system can notify interested parties of changes.

**Template Method Pattern:**

* **Production Planning and Scheduling:**

Use the Template Method Pattern to create a production scheduling algorithm with fixed steps that can be customized for different production scenarios.

* **Complaint and Return Management:**

Create a template for handling customer complaints and returns with common steps and allow customization for specific types of complaints.

**Strategy Pattern:**

* **Bill of Materials (BOM) Management:**

Use the Strategy Pattern to define various strategies for managing BOMs for different textile products. Each product type can have its strategy for handling materials and processes.

* **Costing and Pricing:**

Implement different pricing strategies as separate strategies that can be dynamically chosen for different products or markets.

* **Sales and Order Management:**

Implement different sales strategies for managing orders, invoicing, and customer relationships.

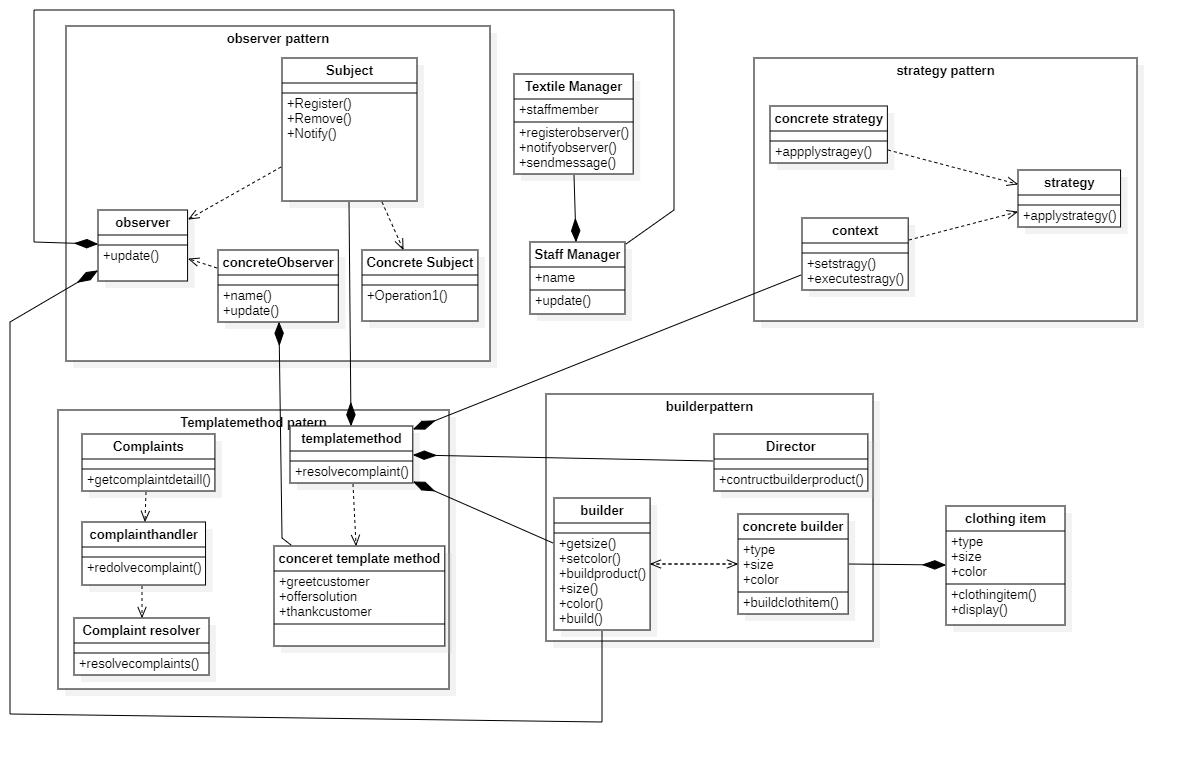
* **Reporting and Analytics:**

Use the Strategy Pattern to define different strategies for generating various types of reports and analytics, such as inventory reports, sales trends, and production efficiency reports.

* **Security and Access Control:**

Implement different security and access control strategies for user roles and permissions. This pattern can help manage various security policies.

**Class Diagram:**

****

**Steps:**

**Observer Pattern:**

* **Subject:** This class represents the subject that is observed. It has methods to register, remove, and notify observers.
* **ConcreteSubject:** A specific implementation of the subject. It maintains a list of observers and notifies them when a change occurs.
* **Observer:** An interface representing observers with a method to update.
* **ConcreteObserver:** A specific implementation of an observer. It has a name and updates when notified.

The ConcreteSubject maintains a list of ConcreteObserver instances and notifies them when notifyObservers is called.

**Template Method Pattern:**

* **TemplateMethod:** An abstract class defining the template method for resolving complaints.
* **ConcreteTemplateMethod:** A specific implementation of the template method with steps for greeting, offering a solution, and thanking the customer.

The ConcreteTemplateMethod class implements the steps for handling complaints, which are defined in the TemplateMethod.

**Builder Pattern:**

* **Builder:** An abstract class defining the builder interface for constructing a product.
* **ConcreteBuilder:** A specific implementation of the builder for building ClothingItem.
* **Director:** This class directs the construction process by using a Builder.
* **Product**: The product created by the builder.

The Director uses the ConcreteBuilder to build a Product.

**Strategy Pattern:**

* **Strategy:** An interface defining the strategy for a particular task.
* **ConcreteStrategy:** A specific implementation of the strategy.
* **Context:** The context class that uses a specific strategy to execute a task.

The Context class is configured with a Strategy instance and can execute a task using the selected strategy.

**Additional Classes:**

* **TextileManager**: Represents a manager with staff members and the ability to send messages to observers.
* **StaffMember:** Represents a staff member who can update in response to messages.
* **ComplaintHandler**: An abstract class representing a complaint handler.
* **ProductComplaintHandler:** A specific implementation of a complaint handler.
* **ClothingItem:** Represents a clothing item with type, size, and color.

The diagram also includes relationships between these classes. For example, ConcreteSubject is a specialization of Subject, and ConcreteObserver implements the Observer interface. Similarly, ProductComplaintHandler extends ComplaintHandler, and ConcreteBuilder implements the Builder interface.