Shopping Cart Project Documentation

1. Overview

The shopping cart application simulates a simple e-commerce system in Java using various design patterns to demonstrate software architectural principles. This documentation covers the project’s architecture, design patterns, UML structure, and usage instructions

3. Implemented Design Patterns

The project employs multiple design patterns, categorized into creational, structural, and behavioral patterns, enhancing modularity, reusability, and scalability

2. Architecture

The project is designed using the **Model-View-Controller (MVC)** architecture pattern to separate the data model (Model), user interface (View), and business logic (Controller). This separation allows for easier testing, maintainability, and flexibility.

1. **Model:** Holds the data (e.g., total price of items)

2. **View**: Displays data to the user

3. **Controller**: Handles interactions and updates the Model and View

Design Patterns: Creational

- **Prototype Pattern: *CartModel***
- **Purpose**: used to encapsulate the process of object creation.
Instead of calling a constructor directly to create objects,
clients call a factory method.
- **Usage:** We create family of objects without specifying through they concrete class

- **Factory Pattern: *ItemFactory***
- **Purpose:** Allows for the creation of different item types without exposing the instantiation logic
- **Usage:** The ***ItemFactory*** class creates item objects (e.g., ***Book***, ***Electronics***) based on the specified type, which is useful for dynamically adding different item types to the cart.

```
Item book = ItemFactory.createItem("book");
Item electronics = ItemFactory.createItem("electronics");
```

Design Patterns: Structural

Adapter Pattern: *PriceAdapter*

Purpose: Converts the price of items into different currencies, adapting the interface of item prices for internationalization.

Usage: The ***PriceAdapter*** converts the price in USD to other currencies like Euros or GBP.
double priceInEuros = adapter.getPriceInCurrency("euro");

Decorator Pattern: *ItemDecorator* and *GiftWrapDecorator*

Purpose: Adds additional functionality (e.g., gift wrapping) to items without altering their structure.

Usage: *GiftWrapDecorator* decorates an item by adding an additional cost for gift wrapping.

Item giftWrappedBook = new GiftWrapDecorator(book

Design Patterns: Behavioral

Observer Pattern: *CartObserver*, *EmailNotification*, and *SMSNotification*

Purpose: Notifies observers whenever the total price of the cart is updated, useful for sending updates to the user.
Usage: CartObserver is implemented by ***EmailNotification*** and ***SMSNotification*** classes to receive notifications of cart updates.

```
Cart cart = new Cart();
cart.addObserver(new EmailNotification());
cart.addObserver(new SMSNotification());
cart.addToTotal(50); // Triggers notifications
```

Strategy Pattern: *PaymentStrategy*, *CreditCardPayment*, *PayPalPayment*, and *DiscountStrategy*

Purpose: Provides multiple payment strategies and discount options that can be dynamically chosen at runtime.
Usage: *CreditCardPayment* and ***PayPalPayment*** implement the ***PaymentStrategy*** interface, while ***DiscountStrategy*** adds various discount methods.

```
PaymentStrategy payment = new CreditCardPayment();
DiscountStrategy discount = new PercentageDiscount(10); // 10% discount
payment.pay(cart.getTotal(), discount);
```

Usage Instructions

1. **Initialize the MVC components** to separate the logic, view, and model.

```
Model model = new Model();
View view = new View();
Controller cartController = new Controller(model, view);
```

2. **Create items** using the ***ItemFactory***.

```
Item book = ItemFactory.createItem("book");
```

3. **Convert prices to different currencies** if needed.

```
double priceInGBP = adapter.getPriceInCurrency("gbp");
```

4. **Add gift wrapping** or other decoration to items using ***ItemDecorator.***

```
Item giftWrappedBook = new GiftWrapDecorator(book);
```

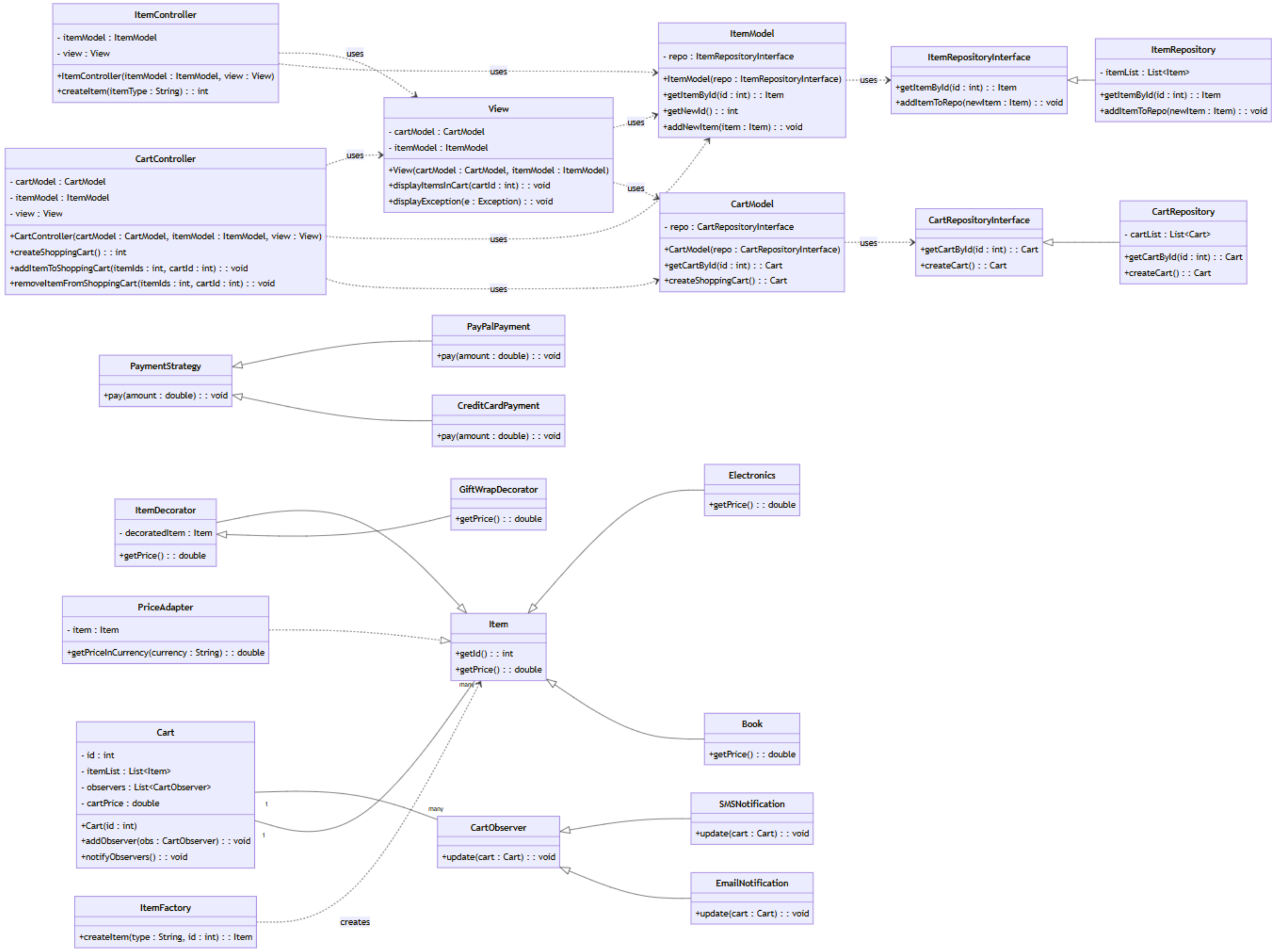
5. **Set up notifications** for the cart using ***CartObserver.***

```
Cart cart = new Cart();
cart.addObserver(new EmailNotification());
```

6. **Choose a payment strategy and apply discounts** if applicable.

```
PaymentStrategy payment = new CreditCardPayment();
DiscountStrategy discount = new PercentageDiscount(15); // 15% discount
payment.pay(cart.getTotal( ), discount);
```

UML


ItemController
- itemModel : ItemModel
- view : View
+ItemController(itemModel : ItemModel, view : View)
+createItem(itemType : String) : : int
uses
uses
View
- cartModel : CartModel
- itemModel : ItemModel
+View(cartModel : CartModel, itemModel : ItemModel)
+displayItemsInCart(cartId : int) : : void
+displayException(e : Exception) : : void
ItemModel
- repo : ItemRepositoryInterface
+ItemModel(repo : ItemRepositoryInterface)
+getItemById(id : int) : : Item
+getNewId() : : int
+addNewItem(item : Item) : : void
uses
ItemRepositoryInterface
+getItemById(id : int) : : Item
+addItemToRepo(newItem : Item) : : void
ItemRepository
- itemList : List<Item>
+getItemById(id : int) : : Item
+addItemToRepo(newItem : Item) : : void
CartController
- cartModel : CartModel
- itemModel : ItemModel
- view : View
+CartController(cartModel : CartModel, itemModel : ItemModel, view : View)
+createShoppingCart() : : int
+addItemToShoppingCart(itemIds : int, cartId : int) : : void
+removeItemFromShoppingCart(itemIds : int, cartId : int) : : void
uses
uses
uses
uses
uses
CartModel
- repo : CartRepositoryInterface
+CartModel(repo : CartRepositoryInterface)
+getCartById(id : int) : : Cart
+createShoppingCart() : : Cart
uses
CartRepositoryInterface
+getCartById(id : int) : : Cart
+createCart() : : Cart
CartRepository
- cartList : List<Cart>
+getCartById(id : int) : : Cart
+createCart() : : Cart
PayPalPayment
+pay(amount : double) : : void
PaymentStrategy
+pay(amount : double) : : void
CreditCardPayment
+pay(amount : double) : : void
GiftWrapDecorator
+getPrice() : : double
Electronics
+getPrice() : : double
ItemDecorator
- decoratedItem : Item
+getPrice() : : double
PriceAdapter
- item : Item
+getPriceInCurrency(currency : String) : : double
Item
+getId() : : int
+getPrice() : : double
many
Book
+getPrice() : : double
Cart
- id : int
- itemList : List<Item>
- observers : List<CartObserver>
- cartPrice : double
+Cart(id : int)
+addObserver(obs : CartObserver) : : void
+notifyObservers() : : void
1
1
many
CartObserver
+update(cart : Cart) : : void
SMSNotification
+update(cart : Cart) : : void
EmailNotification
+update(cart : Cart) : : void
ItemFactory
+createItem(type : String, id : int) : : Item
creates