**1. Design for class “CartScreen”**

A screenshot of a computer program

AI-generated content may be incorrect.

1.1. Attribute design example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Name | Data type | Default Value | Description |
| 1 | placeOrderController | PlaceOrderController | None | Handle “place order” process |
| 2 | cartController | CartController | None | Handle cart-related operations |

1.2. Operation Design example

|  |  |  |  |
| --- | --- | --- | --- |
| # | Name | Return type | Description |
| 1 | chooseRushOrder() | void | Select the rush order option for faster delivery |
| 2 | displayInsufficientProducts() | void | Displays products that are not insufficent and ask customer to update product |
| 3 | choosePlaceOrder() | void | Initiates the “place order” process |
| 4 | updateProduct(product: Product, quantity: int) | void | Update product quantity in cart |
| 5 | removeProduct(product: Product) | void | Remove a product from the cart |

1.3. Parameters

* product (in **updateProduct** and **removeProduct** method) : the product being updated or removed
* quantity (in **updateProduct** method): the new quantity of the product to be updated

1.4. Exceptions

- **InvalidQuantityException:** if the quantity that customer update or choose is invalid or exceeds available stock (negative number)

1.5. How to Use Parameters / Attributes:

* **placeOrderController**: Calls placeOrderController.placeOrder() to process the order.
* **product**: The product whose quantity needs to be updated.
* **quantity**: The new quantity.

1.6. Flow

1. Choose products that want to order
2. Call placeOrderController.placeOrder()

**2. Design for class “PlaceOrderController”**

A screenshot of a computer program

AI-generated content may be incorrect.

2.1. Attribute design example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Name | Data type | Default Value | Description |
| 1 | order | Order | None | The current order in process |

2.2. Operation Design example

|  |  |  |  |
| --- | --- | --- | --- |
| # | Name | Return type | Description |
| 1 | requestRushOrder() | void | Requests a rush order for faster processing |
| 2 | placeOrder(listOfProducts: array<Product>) | void | Places an order with the given list of products |
| 3 | calculateShippingFee() | void | Calculates the shipping fee based on the order details |
| 4 | checkDeliveryInfo() | void | Checks if the delivery information is complete and correct |
| 5 | requestPayOrder() | void | Requests payment processing for the order |
| 6 | checkInventoryQuantity(listOfProducts: array<Product>) | void | Verifies if the requested products are available in stock |
| 7 | updateCart() | void | Updates the cart based on the order status and available inventory |

2.3. Parameters

* **listOfProducts** (in placeOrder and checkInventoryQuantity method): Array of Product objects representing the products to be ordered.

2.4. Exceptions

* **InvalidDeliveryInfoException** if delivery details are incomplete or incorrect.
* **PaymentFailedException** if payment processing encounters an issue.

2.5. How to Use Parameters / Attributes:

* **listOfProducts**: Used to check inventory, calculate total price, and confirm order placement.
* **order**: Stores order details upon successful placement.

2.6. Flow

1. Validate **listOfProducts**.
2. Check inventory availability (**checkInventoryQuantity**). (if invalid quantity step3, else step4)
3. Update cart (**updateCart**).
4. Verify delivery details (**checkDeliveryInfo**).
5. (Optional) Choose rush order (**requestRushOrder**)
6. Calculate shipping fee (**calculateShippingFee**).
7. Process payment (**requestPayOrder**).

**3. Design for class “CartController”**

**A yellow box with black text

AI-generated content may be incorrect.**

3.1. Attribute design example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Name | Data type | Default Value | Description |
| 1 | cart | Cart | None | Represents the shopping cart containing products |

3.2. Operation Design example

|  |  |  |  |
| --- | --- | --- | --- |
| # | Name | Return type | Description |
| 1 | updateQuantities(product: Product, quantity: int) | void | Updates the quantity of a product in the cart |
| 2 | removeFromCart(product: Product) | void | Removes a specified product from the cart |
| 3 | viewCart() | void | Displays the contents of the cart |
| 4 | addToCart(product: Product, quantity: int) | Void | Adds a product to the cart with a specified quantity |

3.3. Parameters

* **product**: The product to be added, updated, or removed.
* **quantity**: The amount of the product to be modified in the cart.

3.4. Exceptions

* **ProductNotFoundException** if the product does not exist in the cart when attempting to update or remove it.

3.5. How to Use Parameters / Attributes:

* **product**: The product to be added, updated, or removed.
* **quantity**: The amount of the product to be modified in the cart.
* **cart**: Uses **cart** to fetch and display all products.

3.6. Flow

**4. Design for class “DeliveryInfo”**

A yellow box with black text

AI-generated content may be incorrect.

4.1. Attribute design example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Name | Data type | Default Value | Description |
| 1 | deliveryAddress | string | None | Stores the full address for delivery |
| 2 | recipientName | string | None | Name of the person receiving the delivery |
| 3 | mail | string | None | Email address of the recipient |
| 4 | phoneNumber | string | None | Contact phone number for delivery |
| 5 | province | string | None | Province/region where the order is being delivered |

4.2. Operation Design example

|  |  |  |  |
| --- | --- | --- | --- |
| # | Name | Return type | Description |

4.3. Parameters

* **deliveryAddress**: The full address where the order should be delivered.
* **recipientName**: The name of the person receiving the order.
* **mail**: The recipient’s email address.
* **phoneNumber**: The recipient’s contact number.
* **province**: The province or region for delivery.

4.4. Exceptions

* **InvalidAddressException** if the deliveryAddress is missing or incorrectly formatted.
* **InvalidContactException** if phoneNumber or mail is missing or invalid.

4.5. How to Use Parameters / Attributes:

* **deliveryAddress**: Used in formatAddress() to ensure a structured format.
* **recipientName**: Included in all communications and delivery labels.
* **mail**: Used for sending order confirmation and tracking updates.
* **phoneNumber**: Used by the delivery service for contact purposes.
* **province**: Helps determine applicable shipping fees and estimated delivery time.

4.6. Flow

**5. Design for class “Invoice”**

A yellow box with black text

AI-generated content may be incorrect.

5.1. Attribute design example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Name | Data type | Default Value | Description |
| 1 | deliveryAddress | string | None | Stores the full address for delivery |
| 2 | recipientName | string | None | Name of the person receiving the delivery |
| 3 | mail | string | None | Email address of the recipient |
| 4 | phoneNumber | string | None | Contact phone number for delivery |
| 5 | province | string | None | Province/region where the order is being delivered |

5.2. Operation Design example

|  |  |  |  |
| --- | --- | --- | --- |
| # | Name | Return type | Description |

5.3. Parameters

5.4. Exceptions

5.5. How to Use Parameters / Attributes:

* **listOfProduct**: Contains the list of products in the invoice.
* **quantity**: Used to calculate the total number of products.
* **productPriceExcludingVAT**: Used to compute the VAT-inclusive price using applyVAT().
* **totalAmount**: Computed using calculateTotalAmount(), including product price and delivery fee.
* **deliveryFee**: Determined based on distance using calculateDeliveryFee().

5.6. Flow

**6. Design for class “Order”**

A yellow box with text

AI-generated content may be incorrect.

6.1. Attribute design example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Attribute Name** | **Data Type** | **Default Value** | **Description** |
| 1 | customerId | string | None | Unique identifier for the customer placing the order |
| 2 | orderId | int | 0 | Unique identifier for the order |
| 3 | customerName | string | None | Name of the customer who placed the order |
| 4 | phoneNumber | string | None | Contact number of the customer |
| 5 | shippingAddress | string | None | Address where the order will be delivered |
| 6 | province | string | None | Province related to the shipping address |
| 7 | totalAmount | int | 0 | Total amount for the order |
| 8 | state | string | "Pending" | Current state of the order (Pending, Approved, Rejected) |

6.2. Operation Design

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Method Name** | **Return Type** | **Description** |
| 1 | save() | void | Saves the order to the database |
| 2 | checkOrderStatus() | void | Checks and returns the current status of the order |
| 3 | changeRejectOrder(order: Order) | void | Changes the order status to "Rejected" |
| 4 | changeApproveOrder(order: Order) | void | Changes the order status to "Approved" |

6.3. Parameters

* order: The order object whose status needs to be changed.

6.4.Exceptions

* OrderNotFoundException if the order ID does not exist.
* InvalidOrderStateException if an invalid state transition is attempted.
* DatabaseSaveException if an error occurs while saving the order.

6.5. How to Use Parameters / Attributes

* orderId: Used to uniquely identify an order.
* state: The current status of the order, updated through changeApproveOrder() or changeRejectOrder().
* totalAmount: Used to verify payment before approval.
* customerId & customerName: Used for customer identification and order tracking.

6.6. Order States

The state attribute follows an enumerated type OrderState, which represents the possible statuses of an order:

|  |  |
| --- | --- |
| **State** | **Description** |
| PENDING | Order is placed but not yet processed |
| APPROVED | Order has been approved for fulfillment |
| REJECTED | Order has been rejected |

**7. Design for class “PaymentController”**

**A yellow box with black text

AI-generated content may be incorrect.**

7.1. Attribute Design

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Attribute Name** | **Data Type** | **Default Value** | **Description** |
| 1 | order | Order | None | Represents the order being processed for payment |

7.2. Operation Design

7.3. Parameters

* order: The order for which payment is being processed.

7.4. Exceptions

* InvalidOrderException if the order is null or invalid.
* PaymentFailedException if the payment processing fails.
* InsufficientFundsException if the user does not have enough balance.

7.5. How to Use Parameters / Attributes

* order: Used to fetch order details and update payment status.

**8. Design for class “Cart”**

A yellow and black text

AI-generated content may be incorrect.

8.1. Attribute Design

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Attribute Name** | **Data Type** | **Default Value** | **Description** |
| 1 | listOfProducts | array<Product> | Empty Array | Holds the list of products in the cart |

8.2. Operation Design

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Method Name** | **Return Type** | **Description** |
| 1 | getCartId() | string | Returns the unique identifier of the cart |
| 2 | removeFromCart(product: Product) | void | Removes a specified product from the cart |
| 3 | updateQuantities(product: Product, quantity: int) | void | Updates the quantity of a product in the cart |
| 4 | calculateTotalPrice() | float | Computes the total price of all products in the cart |
| 5 | addToCart(product: Product, quantity: int) | void | Adds a product to the cart with a specified quantity |
| 6 | empty() | void | Empties the cart, removing all products |
| 7 | checkProductAvailability() | boolean | Checks if all products in the cart are available in stock |
| 8 | checkProductEligible() | boolean | Checks if the products in the cart meet eligibility conditions (e.g., discounts, promotions) |
| 9 | calculateTotalPrice(product: Product, quantity: int) | float | Calculates the total price for a specific product and quantity |

7.3. Exceptions

* ProductNotFoundException if the product does not exist in the cart when attempting to update or remove it.
* OutOfStockException if the product is not available in the required quantity.
* InvalidQuantityException if the specified quantity is not valid (negative or zero).

7.4. How to Use Parameters / Attributes

* product: Used for adding, updating, and removing items from the cart.
* quantity: Used to specify the number of products to be modified.
* listOfProducts: Holds the products currently in the cart and is updated accordingly.

**8. Design for class “Product”**

A screenshot of a computer

AI-generated content may be incorrect.

8.1. Attribute Design

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Attribute Name** | **Data Type** | **Default Value** | **Description** |
| 1 | title | string | None | Name of the product |
| 2 | category | string | None | Product category (e.g., electronics, clothing) |
| 3 | value | float | 0 | The base price of the product |
| 4 | currentPrice | float | 0 | The product price after discounts, offers, or VAT |
| 5 | quantity | int | 0 | Available stock quantity |
| 6 | description | string | None | Detailed description of the product |
| 7 | barcode | string | None | Unique barcode identifier |
| 8 | warehouseEntryDate | date | None | Date when the product was added to the warehouse |
| 9 | dimensions | any | None | Size specifications of the product |
| 10 | weight | int | 0 | Product weight in grams/kilograms |
| 11 | productId | string | None | Unique product identifier |

8.2. Operation Design

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Method Name** | **Return Type** | **Description** |
| 1 | checkProductAvailability(quantity: int) | boolean | Checks if the required quantity is available in stock |
| 2 | getProductInfo() | void | Retrieves all product details |
| 3 | getBarcode() | string | Returns the product barcode |
| 4 | getTitle() | string | Returns the product title |
| 5 | getCurrentPrice() | float | Returns the current price of the product |
| 6 | getDimensions() | any | Retrieves the product dimensions |
| 7 | getValue() | float | Returns the base value of the product |
| 8 | getQuantity() | int | Returns the available quantity in stock |
| 9 | getCategory() | string | Returns the product category |
| 10 | getDescription() | string | Retrieves the product description |
| 11 | checkProductEligible() | boolean | Checks if the product meets eligibility conditions (e.g., discounts, promotions) |
| 12 | getWeight() | int | Returns the product weight |
| 13 | setBarcode() | void | Updates the product barcode |
| 14 | setTitle() | void | Updates the product title |
| 15 | setDimensions() | void | Updates the product dimensions |
| 16 | setQuantity() | void | Updates the stock quantity |
| 17 | setCategory() | void | Updates the product category |
| 18 | setDescription() | void | Updates the product description |
| 19 | setWarehouseEntryDate() | void | Updates the warehouse entry date |
| 20 | setWeight() | void | Updates the product weight |
| 21 | setProductId() | string | Updates the product ID |

8.3. Parameters

* quantity: The amount of the product needed.
* searchAttributes: Criteria for searching the product (e.g., name, category, price range).
* sortRequest: Specifies sorting preferences (e.g., ascending, descending).

8.4. Exceptions

* ProductNotFoundException if the requested product does not exist in the database.
* OutOfStockException if the required quantity is not available.
* InvalidProductException if the product has missing or invalid attributes.

8.5. How to Use Parameters / Attributes

* productId: Unique identifier for each product used to fetch data.
* quantity: Determines stock availability when adding products to a cart or processing orders.
* currentPrice: Used to apply discounts or calculate the final amount for checkout.