Design for class "Cart"

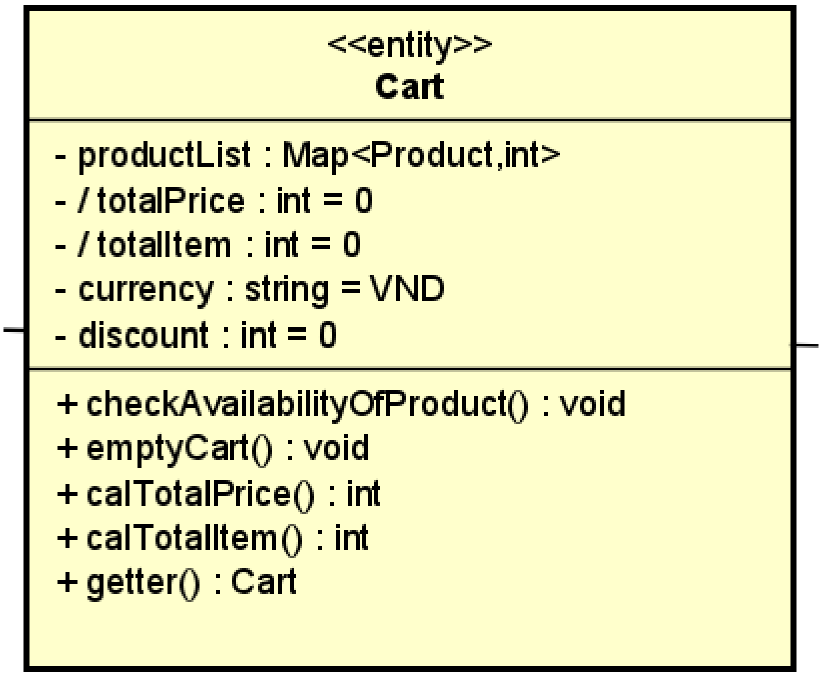


Figure 1: Design Class Diagram of Cart

Table 1: Attribute design of Cart

	Name	Data Type	Default value	Description
1	productList	Map<Product, int>		
2	totalPrice	int	0	
3	totalItem	int	0	
4	currency	int	VND	
5	discount	int	0	

Table 2: Operation design of Cart

	Name	Return Type	Description
1	checkAvailabilityOfProduct()	void	Check if the quantity of products in cart is sufficient
2	emptyCart()		Delete all the products in cart
3	calTotalPrice()	int	Calculate the total price of products in cart
4	calTotalItem()	int	Calculate the total items in cart
5	getter()	Cart	The controller get the information about productList, totalItem, totalPrice, currency and discount to create the Order

1. checkAvailabilityOfProduct()

Exception

Name	Description
InsufficientStockException	Raised when there is not enough stock to fulfill the order.
EmptyCartException	Raised if cart is empty

Method

```
  public void checkAvailabilityOfProduct(int requestedQuantity) throws
ProductUnavailableException, EmptyCartException {
        checkIfEmpty(); // Check if the cart is empty

        // Iterate through each item in the cart and check availability for the requested quantity
        for (CartItem item : items) {
            item.checkAvailabilityOfProduct(requestedQuantity); // Check for each product
        }
    }
```

2. calTotalPrice ()

Method

```
public int calTotalPrice() {
        int totalPrice = 0;
        for (CartItem item : items) {
            totalPrice += item.quantity * item.price;  // Multiply quantity by price for each item
and sum
        }
        return totalPrice;
    }
```

3. emptyCart ()

Method

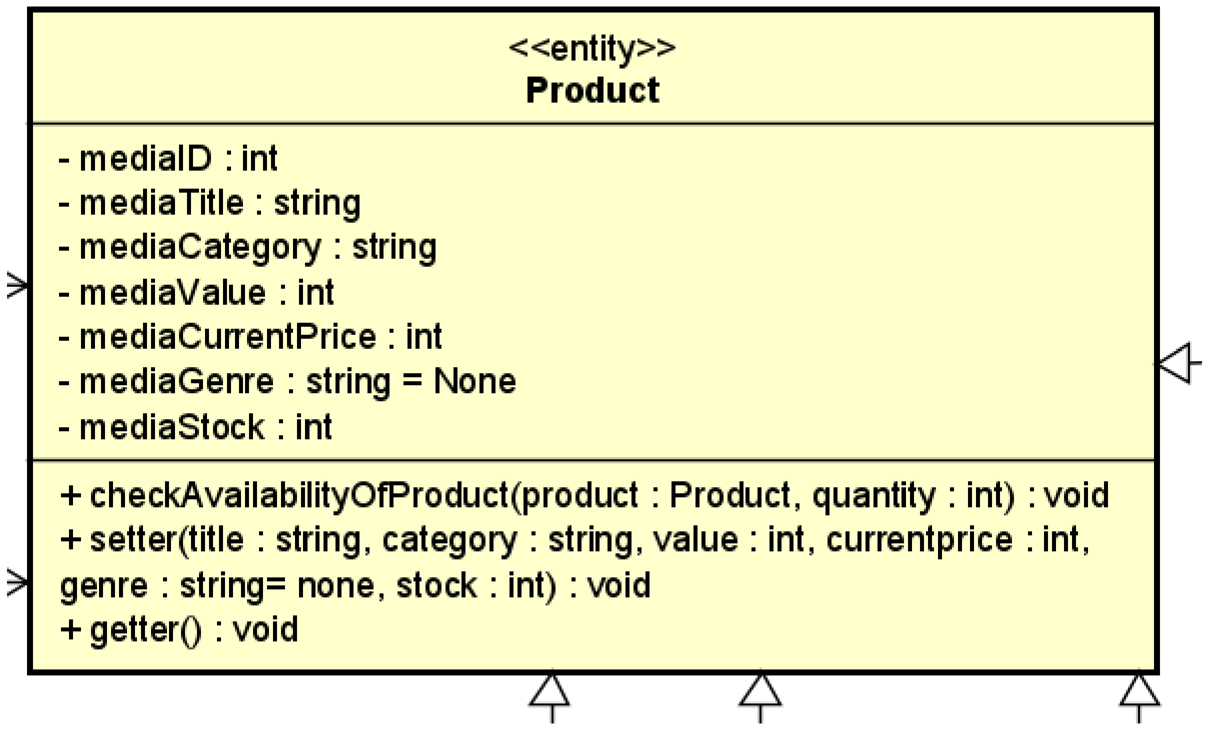
```
public void emptyCart() {
        items.clear();  // where items is a list of product and quantity
}
```

4. calTotalItem ()

Method

```
public int calTotalItem() {
        int totalItems = 0;
        for (CartItem item : items) {
            totalItems += item.quantity;  // Sum the quantities of all items
        }
        return totalItems;
}
```

Design for class “Product”



<<entity>>
Product
- mediaID : int
- mediaTitle : string
- mediaCategory : string
- mediaValue : int
- mediaCurrentPrice : int
- mediaGenre : string = None
- mediaStock : int
+ checkAvailabilityOfProduct(product : Product, quantity : int) : void
+ setter(title : string, category : string, value : int, currentprice : int, genre : string= none, stock : int) : void
+ getter() : void

Figure 2: Design Class of Product

Table 3: Attribute design of Product

#	Name	Data type	Default value	Description
1	mediaID	int		Unique identifier for the product
2	mediaTitle	string		The name of the product
3	mediaValue	int		
4	mediaCurrentPrice	int		The current price of product (VND)
5	mediaGenre	string	None	The list of product’s genres
6	mediaStock	int		The quantity of a particular product available in stock
7	mediaCategory	string		The category of product (DVD, LP, CD, book)

Table 4: Operation design of Product

	Name	Return Type	Description
1	checkAvailabilityOfProduct()	boolean	Used to check if a product is available in stock
2	getter()	v.v	Used to retrieve the values of the attributes in the Product class
3	setter(...)	void	Used by Product Manager

1. checkAvailabilityOfProduct()

Parameter

Name	Default Value	Description
Product		
Quantity		The corresponding amount of chosen product

Exception

Name	Description
InsufficientStockException	Raised when there is not enough stock to fulfill the order.

Method

```
Product product = products.get(productId);
if (product.mediaStock < quantity) {
        throw new InsufficientStockException("Insufficient stock for product " +
product.name + ". Available quantity: " + product.stock);
}
```

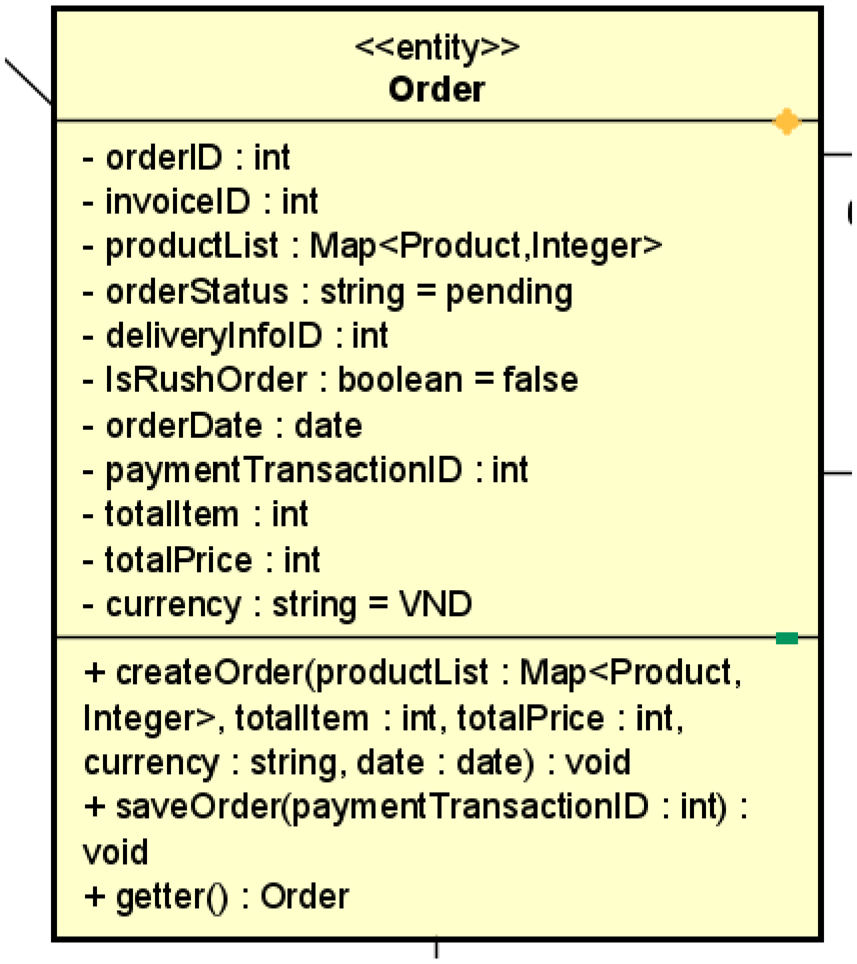
2. setter ()

Name	Default Value	Description
mediaTitle		The name of the product
mediaValue		
mediaCurrentPrice		The current price of product (VND)
mediaGenre	None	The list of product's genres
mediaStock		The quantity of a particular product available in stock
mediaCategory		The category of product (DVD, LP, CD, book)

Method

```
public setter(int mediaID, String mediaTitle, double mediaValue, double
mediaCurrentPrice, String mediaGenre, int mediaStock, String mediaCategory) {
        this.mediaID = mediaID;
        this.mediaTitle = mediaTitle;
        this.mediaValue = mediaValue;
        this.mediaCurrentPrice = mediaCurrentPrice;
        this.mediaGenre = mediaGenre;
        this.mediaStock = mediaStock;
        this.mediaCategory = mediaCategory;
    }
```

Design for class "Order"



```
- orderID : int
- invoiceID : int
- productList : Map<Product,Integer>
- orderStatus : string = pending
- deliveryInfoID : int
- IsRushOrder : boolean = false
- orderDate : date
- paymentTransactionID : int
- totalItem : int
- totalPrice : int
- currency : string = VND
+ createOrder(productList : Map<Product, Integer>, totalItem : int, totalPrice : int, currency : string, date : date) : void
+ saveOrder(paymentTransactionID : int) : void
+ getter() : Order
```

Figure 3: Design Class of Order

Table 5: Attribute design of Order

	Name	Data Type	Default value	Description
1	orderID	int	N/A	Unique identifier for the order
2	invoiceID	int	N/A	The corresponding invoice
3	productList	Map<Product, int>	N/A	A list of media items associated with the order
4	isRushOrder	boolean	N/A	A flag indicating whether the order is a rush order (true or false)
5	orderStatus	char	N/A	Status of the order
6	deliveryInfoID	int	N/A	The corresponding

				delivery information ID
7	paymentTransactionID	int	N/A	The transaction ID used for refunds, if applicable
8	orderDate	date	N/A	The date when the order was placed
9	totalPrice	int		
10	totalItems	int		
11	curency	String		

Table 6: Operation design of Order

	Name	Return Type	Description
1	createOrder()	void	Used to retrieve the values of the attributes in the Product class
2	saveOrder()	void	Update the information about payment
3	getter()	void	Controllers get the information to create Invoice

1. createOrder ()

Parameter

Name	Default Value	Description
productList		A list of media items associated with the order
currency		
orderStatus	new	Status of the order
totalPrice		
totalItems		
orderDate		The date when the order was placed

Exception

Name	Description
NullPointerException	If productList, currency, or date is null
IllegalArgumentException	If totalItem or totalPrice is negative,

Method

```
public void createOrder(Map<Product, Integer> productList, int totalItem, int totalPrice,
String currency, Date date) {
        if (productList == null || currency == null || date == null) {
            throw new IllegalArgumentException("Product list, currency, and date must not be
null");
        }
        if (totalItem < 0 || totalPrice < 0) {
            throw new IllegalArgumentException("Total item and total price must be non-
negative");
        }

        this.productList = productList;
        this.totalItem = totalItem;
        this.totalPrice = totalPrice;
        this.currency = currency;
        this.date = date;}
```

2. saveOrder()

Name	Default Value	Description
paymentTransactionID		The payment transaction ID

Method

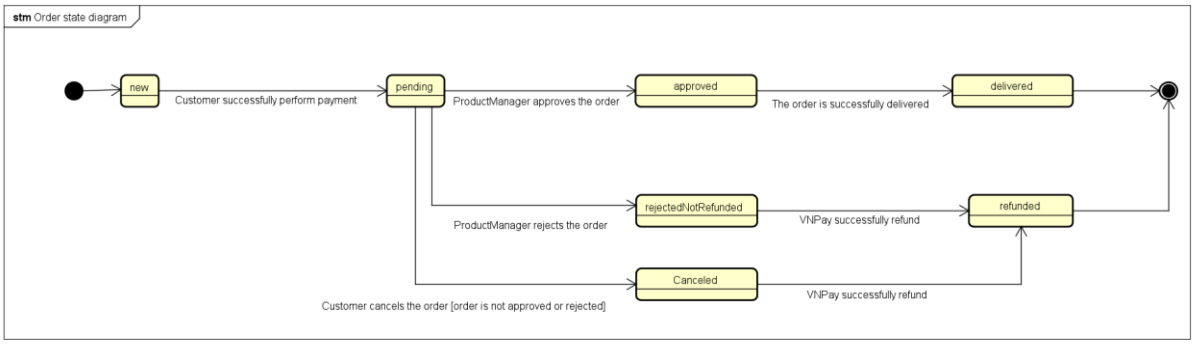
```
public void saveOrder(int paymentTransactionID) {
        this.paymentTransactionID = paymentTransactionID;
        this.orderStatus = "Pending";}
```

3. getter ()

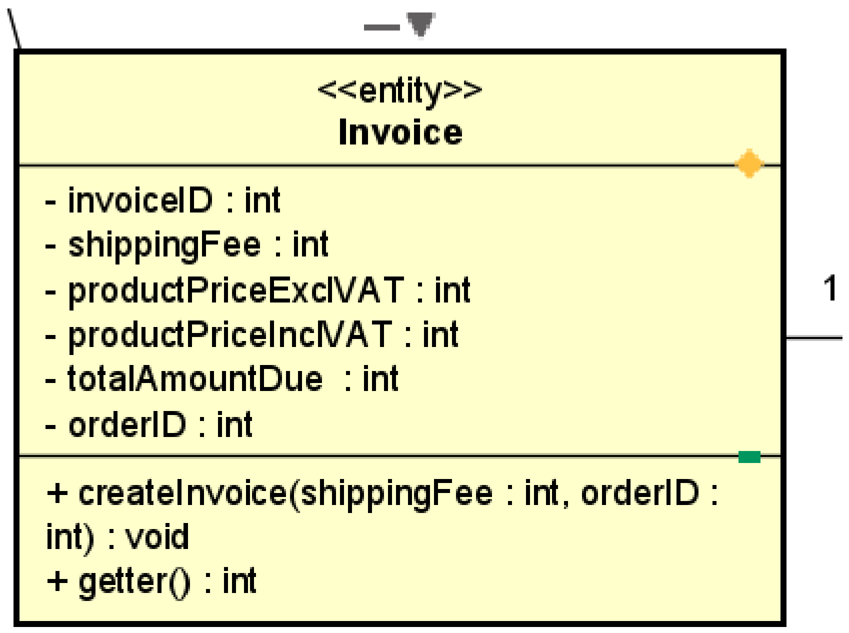
Method

```
public Order getter() {
        return this;
    }
```

State Diagram of order


stm Order state diagram
new
Customer successfully perform payment
pending
ProductManager approves the order
approved
The order is successfully delivered
delivered
ProductManager rejects the order
rejectedNotRefunded
VNPay successfully refund
refunded
Customer cancels the order [order is not approved or rejected]
Canceled
VNPay successfully refund

Design for class “Invoice”



```
<<entity>>
Invoice
- invoiceID : int
- shippingFee : int
- productPriceExclVAT : int
- productPriceInclVAT : int
- totalAmountDue  : int
- orderID : int
+ createInvoice(shippingFee : int, orderID : int) : void
+ getter() : int
```

Figure 4: Design Class of Invoice

Table 7: Attribute design of Invoice

	Name	Data Type	Default value	Description
1	invoiceID	int		
2	shippingFee	Int		The delivery fee
3	productPriceExclVAT	int	0	The total price of products in order before VAT
4	productPriceInclVAT	int	0	The total price of products in order after VAT
5	totalAmountDue	Int	VND	The total amount that customer need to pay
6	orderID	int		

Table 8: Operation design of Invoice

	Name	Return Type	Description
1	createInvoice()	boolean	Controller creates new invoice
2	getter()	Invoice	Get the invoice information

1. createInvoice()

Parameter

Name	Default Value	Description
shippingFee		Shipping fee calculate by controller

orderID		The corresponding order ID

Exception

Name	Description
IllegalArgumentException	If shipping fee or order ID is not valid

Method

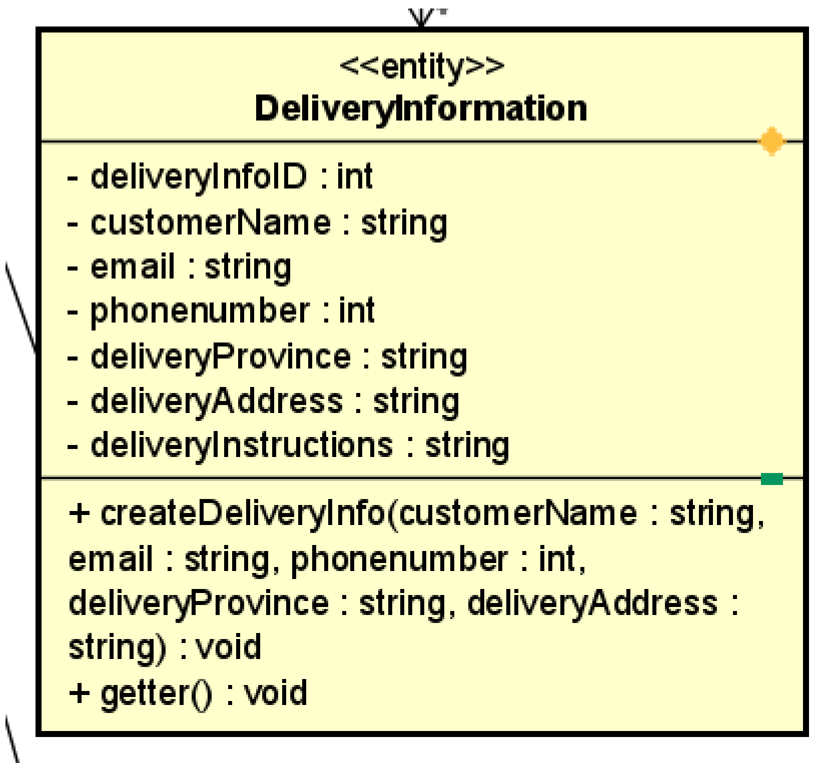
```
public void createInvoice(int shippingFee, int orderID) {
        if (shippingFee < 0) {
            throw new IllegalArgumentException("Shipping fee cannot be negative");
        }
        if (orderID <= 0) {
            throw new IllegalArgumentException("Order ID must be a positive integer");
        }

        this.shippingFee = shippingFee;
        this.orderID = orderID;}
```

2. getter ()

Method

```
public int getter() {
        return this.invoiceID;
    }
```

Design for class "DeliveryInformation"



<<entity>>
DeliveryInformation
- deliveryInfoID : int
- customerName : string
- email : string
- phonenumber : int
- deliveryProvince : string
- deliveryAddress : string
- deliveryInstructions : string
+ createDeliveryInfo(customerName : string, email : string, phonenumber : int, deliveryProvince : string, deliveryAddress : string) : void
+ getter() : void

Figure 5: Design Class of DeliveryInformation

Table 9: Attribute design of DeliveryInformation

	Name	Data Type	Default value	Description
1	deliveryInfoID	int		
2	customerName	String		
3	email	string	0	
4	phonenumber	Int	VND	
5	deliveryProvince	string	0	
6	deliveryAddress	string		
7	deliveryInstruction	string		

Table 10: Operation design of DeliveryInformation

	Name	Return Type	Description
1	createDeliveryInfo()	boolean	Create new delivery information
2	getter()	DeliveryInformation	The controller get the information about deliveryInformation

1. createDeliveryInfo ()

Parameter

Name	Default Value	Description

customerName		
email		The corresponding amount of chosen product
phonenumber		
deliveryProvince		
deliveryAddress		
deliveryInstruction		

Exception

Name	Description
IllegalArgumentException	If any input field is not valid

Method

```
public boolean createDeliveryInfo(String customerName, String email, int phoneNumber,
String deliveryProvince, String deliveryAddress, String deliveryInstruction) {
        if (customerName == null || customerName.isEmpty()) {
            throw new IllegalArgumentException("Customer name cannot be null or empty");
        }
        if (email == null || email.isEmpty()) {
            throw new IllegalArgumentException("Email cannot be null or empty");
        }
        if (phoneNumber <= 0) {
            throw new IllegalArgumentException("Phone number must be a positive integer");
        }
        if (deliveryProvince == null || deliveryProvince.isEmpty()) {
            throw new IllegalArgumentException("Delivery province cannot be null or
empty");
        }
        if (deliveryAddress == null || deliveryAddress.isEmpty()) {
            throw new IllegalArgumentException("Delivery address cannot be null or empty");
        }
}
```

2. getter ()

Method

```
public int getter() {
        return this.deliveryInfoID;
    }
```

Design for class “PlaceOrderController”

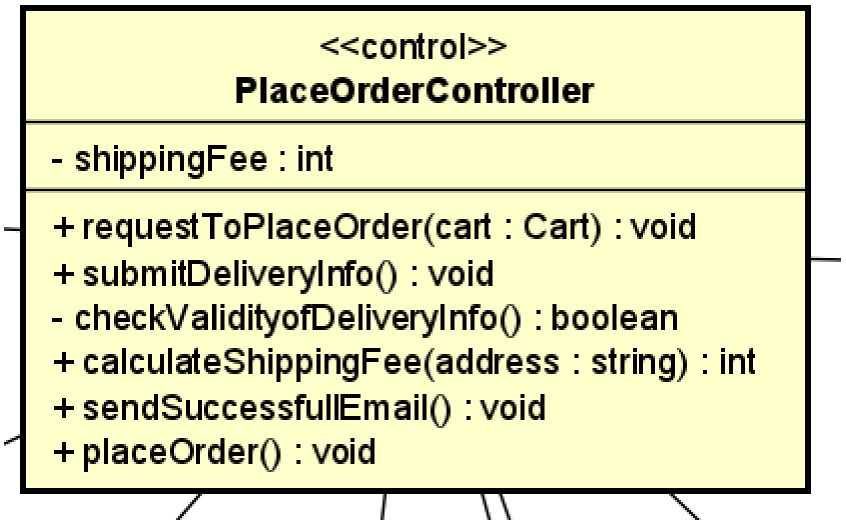

<<control>>
PlaceOrderController
- shippingFee : int
+ requestToPlaceOrder(cart : Cart) : void
+ submitDeliveryInfo() : void
- checkValidityofDeliveryInfo() : boolean
+ calculateShippingFee(address : string) : int
+ sendSuccessfullEmail() : void
+ placeOrder() : void

Figure 6: Design Class of PlaceOrderController

Table 11: Attribute design of PlaceOrderController

	Name	Data Type	Default value	Description
1	shippingFee	int		The controller need to recalculate and save the shipping fee before creating the invoice

Table 12: Operation design of PlaceOrderController

	Name	Return Type	Description
1	requestToPlaceOrder ()	void	Create new delivery information
2	submitDeliveryInfo ()	void	The controller get the information about deliveryInformation
3	checkValidityofDeliveryInfo()	void	Check the validity of delivery information
4	calculateShippingFee()	int	Calculate shipping fee based on submitted addre
5	sendSuccessfullEmail()	void	Send email to customer after place an order successfully
6	placeOrder()	void	Call to saveOrder method

1. requestToPlaceOrder ()

Parameter

Name	Default Value	Description
cart		

Exception

Name	Description
IllegalArgumentException	If any input field is not valid
InsufficientStockException	If any products is insufficient

Method

```
public void requestToPlaceOrder(Cart cart) throws InsufficientStockException {
        checkAvailabilityOfProduct(cart);
    }
```

2. submitDeliveryInfo ()

Exception

Name	Description
IllegalArgumentException	If any input field is not valid

Method

```
public void submitDeliveryInfo(DeliveryInformation deliveryInfo) {
        if (deliveryInfo == null) {
            throw new IllegalArgumentException("Delivery information cannot be null");
        }

        if (checkValidityOfDeliveryInfo(deliveryInfo)) {
            deliveryInfo.createDeliveryInfo(
                deliveryInfo.getCustomerName(),
                deliveryInfo.getEmail(),
                deliveryInfo.getPhoneNumber(),
                deliveryInfo.getDeliveryProvince(),
                deliveryInfo.getDeliveryAddress(),
                deliveryInfo.getDeliveryInstruction()
            );
    }
```

3. checkValidityofDeliveryInfo ()

Method

```
private boolean checkValidityOfDeliveryInfo(DeliveryInformation deliveryInfo) {
        return true; // Placeholder return value
    }
```

4. calculateShippingFee ()

Name	Default Value	Description
address		

5. placeOrder ()

Exception

Name	Description
IllegalArgumentException	If any input field is not valid

Method

```
public void placeOrder(Order order, int paymentTransactionID) {
        if (order == null) {
            throw new IllegalArgumentException("Order cannot be null");
        }
        if (paymentTransactionID <= 0) {
            throw new IllegalArgumentException("Payment transaction ID must be positive");
        }

        order.saveOrder(paymentTransactionID);
    }
```

Design for class "OrderInformationScreen"

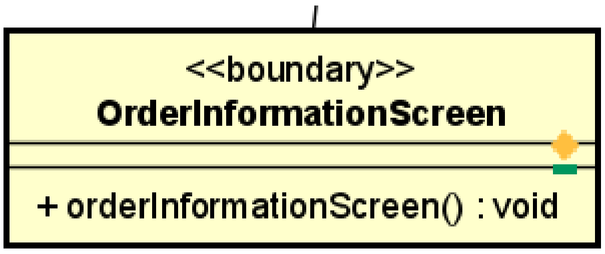


Figure 7: Design Class of OrderInformationScreen

Table 13: Operation design of OrderInformationScreen

	Name	Return Type	Description
1	orderInformationScreen()	void	Display all the related information about order

1. orderInformationScreen ()

Parameter

Name	Default Value	Description
order		

Exception

Name	Description
IllegalArgumentException	If any input field is not valid

Method

```
public void orderInformationScreen (Order order) {
        if (order == null) {
            throw new IllegalArgumentException("Order cannot be null");
        }

        System.out.println("Displaying Order Information:");
        System.out.println("Total Items: " + order.getTotalItem());
        System.out.println("Total Price: " + order.getTotalPrice() + " " + order.getCurrency());
        System.out.println("Order Date: " + order.getDate());
        System.out.println("Order Status: " + order.getOrderStatus());
    }
```

Design for class “DeliveryInfoScreen”

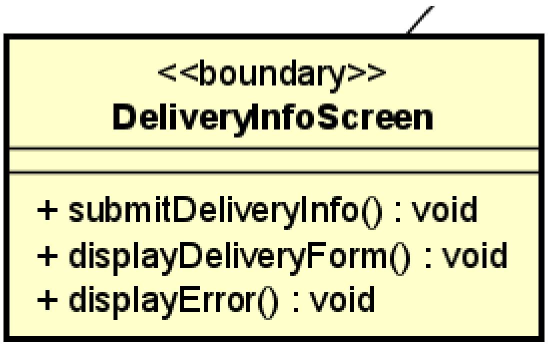


Figure 8: Design Class of DeliveryInfoScreen

Table 14: Operation design of DeliveryInfoScreen

	Name	Return Type	Description
1	submitDeliveryInfo ()	void	User submit the information
2	displayDeliveryForm()	void	Display the delivery information form
3	displayError()	void	

1. displayError ()

Method

```
public void displayError () {
        try {
            controller.checkValidityofDeliveryInfo();
        } catch (Exception e) {
            System.out.println("Unmet Information: " + e.getMessage());
        }
    }
```

Design for class “InvoiceScreen”

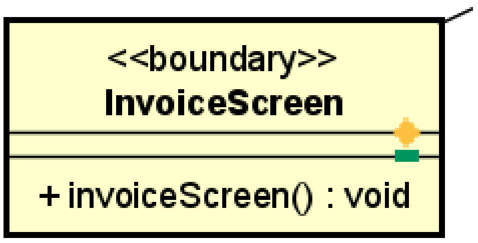


Figure 9: Design Class of InvoiceScreen

Table 15: Operation design of InvoiceScreen

	Name	Return Type	Description
1	invoiceScreen()	void	Redirect customer to payment method

Design for class “CartScreen”

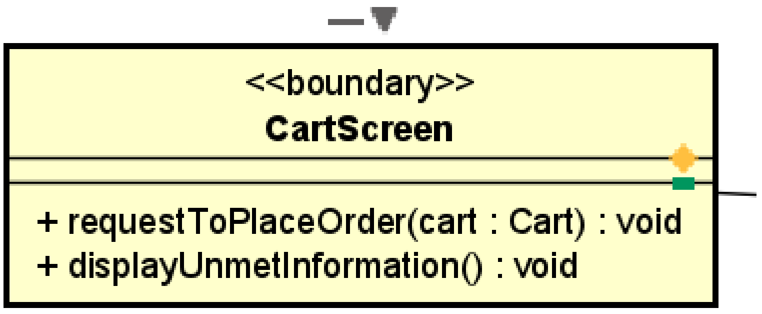

<<boundary>>
CartScreen
+ requestToPlaceOrder(cart : Cart) : void
+ displayUnmetInformation() : void

Figure 10: Design Class of CartScreen

Table 16: Operation design of CartScreen

	Name	Return Type	Description
1	requestToPlaceOrder ()	void	Send to submitted information from user to controller
2	displayUnmetInformation()	void	Display unmet information for user if any product is insufficient

2. displayUnmetInformation()

Method

```
public void displayUnmetInformation(Cart cart) {
        try {
            controller.requestToPlaceOrder(cart);
            System.out.println("All products are available.");
        } catch (InsufficientStockException e) {
            System.out.println("Unmet Information: " + e.getMessage());
        }
    }
```