Name: Nguyen Huu Hai

Student ID: 20215200

REPORT ITSS SOFTWARE MANAGEMENT

OCP IN SOLID PRINCIPLE

1. **At file PaymentController**

package com.springboot.controller;  
  
import java.io.IOException;  
import java.sql.SQLException;  
import java.util.Map;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import com.springboot.model.entity.Invoice;  
import com.springboot.model.entity.Order;  
import com.springboot.model.entity.PaymentTransaction;  
import com.springboot.model.response.InvoiceDetailResponse;  
import com.springboot.model.response.PaymentURLResponse;  
import com.springboot.service.CartService;  
import com.springboot.service.InvoiceService;  
import com.springboot.subsystem.IPaymentSubsystem;  
import com.springboot.subsystem.PaymentSubsystem;  
import com.springboot.subsystem.vnpaysubsystem.VNPaySubsystemController;  
  
@RestController  
@RequestMapping  
public class PaymentController {  
  
 private IPaymentSubsystem payment;  
 private Invoice invoice;  
  
 @Autowired  
 InvoiceService invoiceService;  
 @Autowired  
 private CartService cartService;  
  
 public PaymentController() {  
 this.payment = new PaymentSubsystem(new VNPaySubsystemController());  
 }  
   
 public void payOrder(Order order) {  
 invoice = new Invoice(order);  
 }  
   
 @GetMapping("/payment/invoice")  
 public ResponseEntity<InvoiceDetailResponse> getInvoiceDetail() {  
   
 return ResponseEntity.*ok*(new InvoiceDetailResponse(invoice.getOrder(), cartService));  
 }  
  
 @PostMapping("/payment/result")  
 public ResponseEntity<Void> makePayment(@RequestBody Map<String, String> res) throws IOException, SQLException {  
 PaymentTransaction transaction = payment.getPaymentTransaction(res);  
 invoice.setPaymentTransaction(transaction);  
 invoiceService.save(invoice);  
 return ResponseEntity.*ok*().build();  
 }  
  
 @GetMapping("/payment/VNPayURL")  
 public PaymentURLResponse generateURL() throws IOException {  
 return new PaymentURLResponse(payment.generateURL(invoice.getAmount(), "Payment"));  
 }  
  
}

Problem: This code has some issues with OCP (Open/Closed Principle) compliance:

* The PaymentController class depends hard on the PaymentSubsystem and VNPaySubsystemController classes. If we want to change your payment provider, we will have to modify the PaymentController source code, which violates OCP guidelines.
* Initializing PaymentSubsystem and VNPaySubsystemController objects in the PaymentController constructor makes this class less flexible and difficult to extend.

***Improvement recommendations:***

To comply with the OCP principle, we can apply Dependency Injection and Abstraction techniques as follows:

* Create an IPaymentGateway interface to define common methods for different payment providers:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

* Implement the Payment Gateway interface for specific payment service providers, such as VNPayPaymentGateway:

Ảnh có chứa văn bản, ảnh chụp màn hình

Mô tả được tạo tự động

* Modify the PaymentController class to use Dependency Injection for IPaymentGateway:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

In the Spring Boot configuration file, we can register VNPayPaymentGateway as a bean:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

This way, when we want to change the payment provider, we just need to implement a new class for that provider, then register that class as a bean in the Spring Boot configuration file. The PaymentController class will not be affected by this change, thus complying with OCP guidelines.

1. At PlaceOrderController file:  
   package com.springboot.controller;  
     
   import java.time.LocalDate;  
   import java.time.format.DateTimeFormatter;  
   import java.util.List;  
   import java.util.Map;  
   import java.util.stream.Collectors;  
     
   import org.springframework.beans.factory.annotation.Autowired;  
   import org.springframework.http.ResponseEntity;  
   import org.springframework.web.bind.annotation.GetMapping;  
   import org.springframework.web.bind.annotation.PostMapping;  
   import org.springframework.web.bind.annotation.RequestBody;  
   import org.springframework.web.bind.annotation.RequestMapping;  
   import org.springframework.web.bind.annotation.RestController;  
     
   import com.springboot.model.entity.Cart;  
   import com.springboot.model.entity.CartProduct;  
   import com.springboot.model.entity.DeliveryInfo;  
   import com.springboot.model.entity.Order;  
   import com.springboot.model.response.CartProductResponse;  
   import com.springboot.model.response.RushDeliveryCheckResponse;  
   import com.springboot.service.CartService;  
     
   @RestController  
   @RequestMapping  
   public class PlaceOrderController {  
      
    @Autowired  
    private PaymentController paymentController;  
    private Order order;  
    private Double normalShippingFees = 0.0;  
    private Double rushShippingFees = 0.0;  
    @Autowired  
    private CartService cartService;  
     
    //test for pay order  
    @GetMapping("/pay")  
    public ResponseEntity<Void> payOrder() {  
      
    try {  
    Cart cart = cartService.findById((long) 1);  
    order = new Order(cart,20000,30000,new DeliveryInfo("Ha","0123", "a@gmail.com","HN", false));  
    paymentController.payOrder(order);  
    } catch (Exception e) {  
    e.printStackTrace();  
    }  
    return ResponseEntity.*ok*().build();  
    }  
     
    // test for cart delivery  
    @GetMapping("/cart/delivery")  
    public ResponseEntity<CartProductResponse> getCartDelivery(@RequestBody Map<String, Object> request) {  
    try {  
    Long cartId = Long.*valueOf*(request.get("cartId").toString());  
    List<CartProduct> cartProducts = cartService.getAllProductsInCart(cartId);  
    CartProductResponse response = CartProductResponse.*fromCartProducts*(cartProducts);  
    return ResponseEntity.*ok*(response);  
    } catch (Exception e) {  
    e.printStackTrace();  
    return ResponseEntity.*notFound*().build();  
    }  
    }  
     
    // test for check rush delivery  
    @GetMapping("/cart/delivery/checkRushOrder")  
    public ResponseEntity<RushDeliveryCheckResponse> checkRushOrder(@RequestBody Map<String, Object> request) {  
    try {  
    Integer province = null;  
    Long cartId = Long.*valueOf*(request.get("cartId").toString());  
    if (request.get("province") != null) {  
    province = Integer.*valueOf*(request.get("province").toString());  
    }  
    Boolean isRushDelivery = Boolean.*valueOf*(request.get("isRushDelivery").toString());  
     
   // Cart cart = cartService.findById((long) 1);  
   // order = new Order(cart, 0, 0, new DeliveryInfo("Ha","0123", "a@gmail.com","HN", false));  
    if (province == null) {  
    RushDeliveryCheckResponse response = new RushDeliveryCheckResponse(0, 0, false);  
    return ResponseEntity.*ok*(response);  
    }  
    else if (isRushDelivery == false || province != 1) {  
    List<CartProduct> cartProducts = cartService.getAllProductsInCart(cartId);  
    double normalShippingFee = calculateNormalShippingFee(cartProducts, province);  
    this.normalShippingFees = normalShippingFee;  
    RushDeliveryCheckResponse response = new RushDeliveryCheckResponse(normalShippingFee, 0, false);  
    return ResponseEntity.*ok*(response);  
    }  
    else {  
    List<CartProduct> rushDeliveryProducts = getRushDeliveryProducts(cartService.getAllProductsInCart(cartId));  
    double normalShippingFee = calculateNormalShippingFee(rushDeliveryProducts, province);  
    double rushShippingFee = calculateRushShippingFee(rushDeliveryProducts, province);  
    this.normalShippingFees = normalShippingFee;  
    this.rushShippingFees = rushShippingFee;  
    RushDeliveryCheckResponse response = new RushDeliveryCheckResponse(normalShippingFee, rushShippingFee, false);  
    return ResponseEntity.*ok*(response);  
    }  
    } catch (Exception e) {  
    e.printStackTrace();  
    return ResponseEntity.*notFound*().build();  
    }  
    }  
      
    @SuppressWarnings("unchecked")  
    @PostMapping("/cart/delivery/submit")  
    public ResponseEntity<String> submitDeliveryForm(@RequestBody Map<String, Object> request) {  
    try {  
    DateTimeFormatter formatter = DateTimeFormatter.*ofPattern*("yyyy-MM-dd");  
    Map<String, Object> deliveryFormDTO = (Map<String, Object>) request.get("DeliveryFormDTO");  
    String name = deliveryFormDTO.get("name").toString();  
    String phone = deliveryFormDTO.get("phone").toString();  
    String email = deliveryFormDTO.get("email").toString();  
    String address = deliveryFormDTO.get("address").toString();  
    Long province = Long.*valueOf*(deliveryFormDTO.get("province").toString());  
    String instructions = deliveryFormDTO.get("note").toString();  
    LocalDate date = LocalDate.*parse*(deliveryFormDTO.get("date").toString(), formatter);  
    Boolean isRushDelivery = Boolean.*valueOf*(deliveryFormDTO.get("isRushDelivery").toString());  
      
    DeliveryInfo deliveryInfo = new DeliveryInfo(name, phone, email, province, instructions, address, date, isRushDelivery);  
      
    if (!deliveryInfo.isValid()) {  
    return ResponseEntity.*status*(404).body("Invalid delivery information");  
    }  
      
    Cart cart = cartService.findById((long) 1);  
    this.order = new Order(cart, this.normalShippingFees, this.rushShippingFees, deliveryInfo);  
      
    return ResponseEntity.*ok*("Order created successfully");  
    } catch (Exception e) {  
    e.printStackTrace();  
    return ResponseEntity.*status*(404).body("Failed to create order");  
    }  
    }  
     
    public double calculateNormalShippingFee(List<CartProduct> cartProducts, Integer province) {  
    double totalPrice = calculateTotalPrice(cartProducts);  
    double maxWeight = findProductWithMaxWeight(cartProducts);  
    int baseRate;  
    int additionalFeePerHalfKg = 2500; // Additional fee for every subsequent 0.5kg  
    double shippingFee;  
    // if the province is Hanoi or Ho Chi Minh city  
    if (province == 1 || province == 79) {  
    baseRate = 22000; // Initial price for the first 3kg  
    if (maxWeight <= 3) {  
    shippingFee = baseRate;   
    } else {  
    shippingFee = baseRate + Math.*ceil*((maxWeight - 3) / 0.5) \* additionalFeePerHalfKg;  
    }  
    } else {  
    baseRate = 30000; // Initial price for the first 0.5kg  
    if (maxWeight <= 0.5) {  
    shippingFee = baseRate;  
    } else {  
    shippingFee = baseRate + Math.*ceil*((maxWeight - 0.5) / 0.5) \* additionalFeePerHalfKg;  
    }  
    }  
    // Apply free shipping if total price exceeds 100,000 VND, up to a maximum of 25,000 VND  
    if (totalPrice > 100000) {  
    shippingFee = Math.*max*(shippingFee - 25000, 0);  
    }  
    return shippingFee;  
    }  
      
    public double calculateRushShippingFee(List<CartProduct> cartProducts, Integer province) {  
    int baseRate;  
    int additionalFeePerHalfKg = 2500; // Additional fee for every subsequent 0.5kg  
    double maxWeight = findProductWithMaxWeight(cartProducts);  
    double shippingFee;  
    if (province == 1 || province == 79) {  
    baseRate = 22000; // Initial price for the first 3kg  
    if (maxWeight <= 3) {  
    shippingFee = baseRate;  
    } else {  
    shippingFee = baseRate + Math.*ceil*((maxWeight - 3) / 0.5) \* additionalFeePerHalfKg;  
    }  
    } else {  
    baseRate = 30000; // Initial price for the first 0.5kg  
    if (maxWeight <= 0.5) {  
    shippingFee = baseRate;  
    } else {  
    shippingFee = baseRate + Math.*ceil*((maxWeight - 0.5) / 0.5) \* additionalFeePerHalfKg;  
    }  
    }  
    shippingFee += cartProducts.size() \* 10000;  
    return shippingFee;  
    }  
     
    public double findProductWithMaxWeight(List<CartProduct> cartProducts) {  
    return cartProducts.stream()  
    .mapToDouble(cartProduct -> cartProduct.getProduct().getWeight() \* cartProduct.getQty())  
    .max()  
    .orElse(0.0);  
    }  
     
    public double calculateTotalPrice(List<CartProduct> cartProducts) {  
    return cartProducts.stream()  
    .mapToDouble(cartProduct -> cartProduct.getProduct().getPrice() \* cartProduct.getQty())  
    .sum();  
    }  
     
    public List<CartProduct> getRushDeliveryProducts(List<CartProduct> cartProducts) {  
    return cartProducts.stream()  
    .filter(cartProduct -> cartProduct.getProduct().isRushOrderEligible())  
    .collect(Collectors.*toList*());  
    }  
   }

Problem:

* The PlaceOrderController class is too responsible for calculating delivery fees. If you have a request to change the shipping fee calculation or add a new fee method, you will have to modify the source code of this class, which violates OCP principles.
* Shipping fee calculation methods like calculateNormalShippingFee, calculateRushShippingFee, and findProductWithMaxWeight are in the PlaceOrderController class, which makes the class too complex and difficult to maintain.

Improvement recommendations:  
To comply with the OCP principle, we can apply the Strategy Pattern technique as follows:

* Create an interface ShippingFeeCalculator to define the shipping fee calculation method:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

* Implement the ShippingFeeCalculator interface for different shipping fee calculations:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

* Create a separate class to manage support methods like findProductWithMaxWeight and calculateTotalPrice:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

By this way, if we want to change the shipping fee calculation or add a new fee method, we just need to implement a new class for that fee method and register that class as a bean in the Spring Boot configuration file. The PlaceOrderController class will not be affected by this change, thus complying with OCP guidelines.

1. **At file ViewCartController:**

package com.springboot.controller;  
  
import java.util.List;  
import java.util.Map;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
import org.springframework.web.bind.annotation.RestController;  
  
import com.springboot.model.entity.CartProduct;  
import com.springboot.model.response.StockAvailabilityResponse;  
import com.springboot.model.response.TaxResponse;  
import com.springboot.model.response.UpdateCartResponse;  
import com.springboot.service.CartService;  
import com.springboot.service.ProductService;  
  
@RestController  
@RequestMapping  
public class ViewCartController {  
 @Autowired  
 private CartService cartService;  
 @Autowired  
 private ProductService productService;  
  
 @GetMapping("/cart")  
 public ResponseEntity<List<CartProduct>> getAllProductsInCart() {  
 List<CartProduct> cartProducts = cartService.getAllProductsInCart(1L);  
 if (cartProducts == null) {  
 return ResponseEntity.*notFound*().build();  
 }  
 return ResponseEntity.*ok*(cartProducts);  
 }  
  
 @PostMapping("/cart")  
 public ResponseEntity<UpdateCartResponse> updateCart(@RequestBody Map<String, Object> request) {  
 Long productId = Long.*valueOf*(request.get("product\_id").toString());  
 Integer qty = Integer.*valueOf*(request.get("qty").toString());  
  
 try {  
 List<CartProduct> cart = cartService.updateCart(1L, productId, qty);  
  
 return ResponseEntity.*ok*(new UpdateCartResponse("Cart updated successfully", cart));  
 } catch (Exception e) {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 @GetMapping("/inventory/check")  
 public ResponseEntity<StockAvailabilityResponse> checkQtyInStock(@RequestParam("product\_id") Long productId,  
 @RequestParam("qty") Integer qty) {  
 try {  
 boolean isAvailable = productService.checkInventory(productId, qty);  
 return ResponseEntity.*ok*(new StockAvailabilityResponse(productId, qty, isAvailable) );  
 } catch (Exception e) {  
 return ResponseEntity.*notFound*().build();  
 }  
 }  
  
 @GetMapping(value = "/tax")  
 public ResponseEntity<TaxResponse> getTax() {  
 return ResponseEntity.*ok*(new TaxResponse(10));  
 }  
}

This code does not explicitly violate OCP principles. However, there are some comments and suggestions to improve the source code and better comply with OCP guidelines:

Methods related to handling cart data (getAllProductsInCart, updateCart) and checking inventory quantities (checkQtyInStock) should be separated from the ViewCartController class. This helps the ViewCartController class to focus solely on handling requests from the client and returning the appropriate response.

We can create a CartService interface to define cart-related methods, and a concrete class that implements that interface, for example DefaultCartService. Similarly, we can create an InventoryService interface and a class implementing DefaultInventoryService to handle issues related to inventory management.

The ViewCartController class will use these interfaces through Dependency Injection, helping to better comply with OCP principles. When we want to change the cart handling or inventory management logic, you just need to implement a new class for the corresponding interface, without modifying the ViewCartController.