Concordia University
Engineering and Computer Science

COEN 6312: E-Commerce System

Deliverable 2

Submitted To:

Prof. Abdelwahab Hamou-Lhadj

Submitted By:

Group 10

Hephzibah Pocharam - 40127128

Nikhil Verma - 40160264

Chirag Jhamb - 40169876

Divyaa Mahalakshmi Guruswamy- 40167923

Jayapriya Muthuramasamy - 40184587

Abdul Rahman Koleilat – 40086025

Contents

Table of Figures

1. REVISED CLASS DIAGRAM

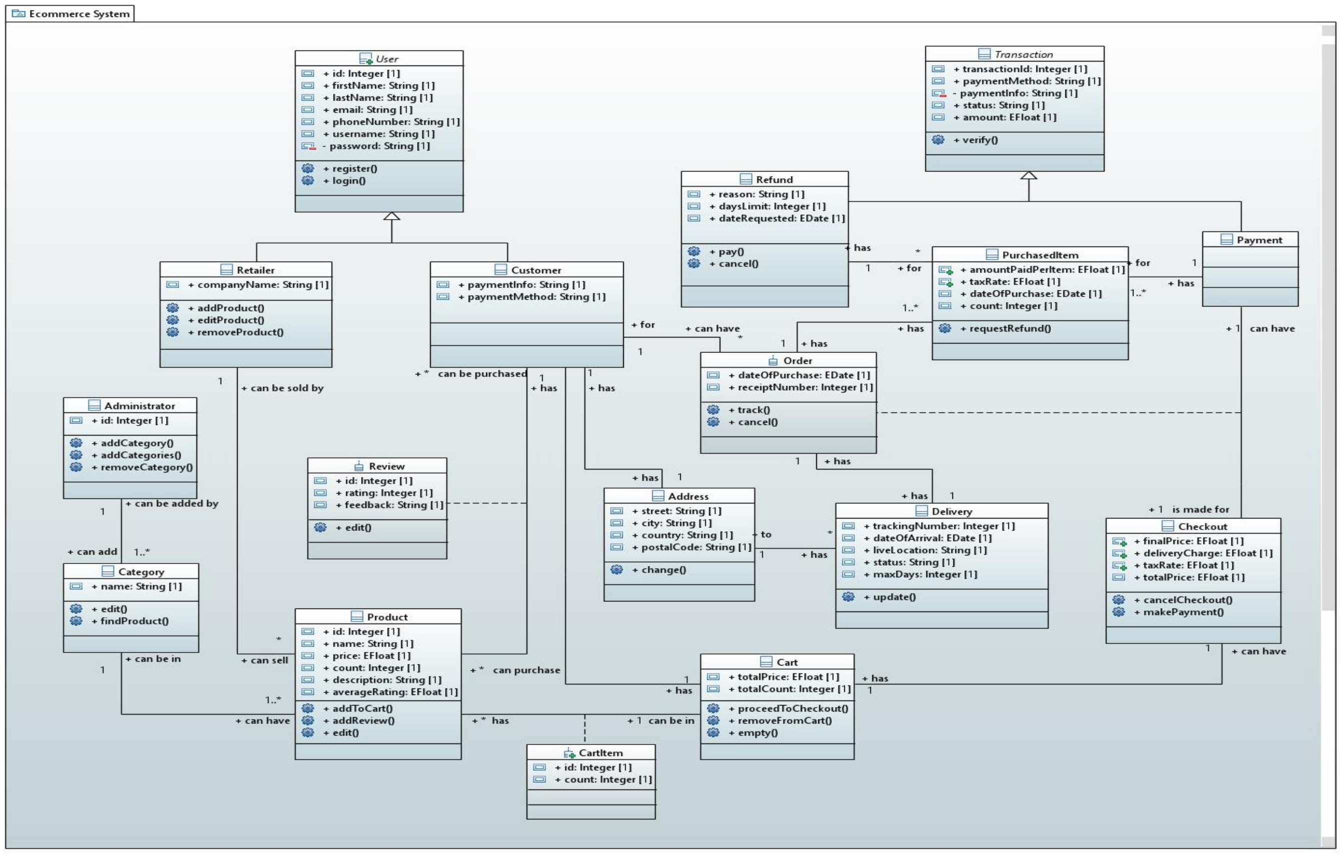


Figure 1 Revised class diagram

2. OCL CONSTRAINTS

1. Users should have a unique ID, email, and username for authentication.

   Context User

   Inv: allInstances()->forAll(u1, u2:User|u1<>u2 implies u1.id<>u2.id AND u1.email<>u2.email AND u1.username<>u2.username)

2. The retailer cannot add the same product again to avoid redundancy.

   Context Retailer::addProduct(P:product)

   pre: self.Product->excludes(P)

post: self.Product->includes(P)

3. The customer can initiate the refund process for the product only after receiving it or if the delivery hasn't arrived on time. The customer can request a refund within 30 days of the product purchase after arrival.

   Context purchased item::requestRefund(r :Refund)

   pre: self.Refund -> excludes(r)

   (self.order.delivery.status="arrived" AND (Day(today) – Day(self.dateOfPurchase)) < 30) OR ((Day(today) - Day(self.Order.Delivery.dateOfArrival)) > 0)

   post: self.Refund -> includes(r) AND r.status = "Requested"

4. For the customer to give review to the product, he/she should be a verified purchaser.

   Context Customer

   Inv: self.product -> includesAll(self.Order.PurchasedItem.Payment.Checkout.Cart.Product)

5. Two orders cannot have the same receipt number.

   Context order

   Inv: allInstances()->forAll(o1,o2 :Order|o1<>o2 implies o1.receiptNumber<>o2.receiptNumber)

6. The customer won't be able to proceed to checkout if the cart is empty.

   Context cart::proceedToCheckout(c :Checkout)

   Pre: self.checkout -> excludes(c) AND self.CartItem -> isnotEmpty()

   Post: self.checkout -> includes(c)

7. An item can only be purchased if the payment is approved

   Context Payment

   Inv: self.purchaseditem -> isnotEmpty() implies self.status = "Approved"

8. Only the customer user can add the product to cart, the cart item count should be equal or less than the product count, and the customer can only add a maximum of 5 of the same product. The total price is then updated.

Context Product::addToCart(c :CartItem, countTobeAdded :int)

pre: self.User -> forAll(u :user | u.oclIsTypeOf(Customer)) AND self.CartItem -> forAll(c :CartItem, p :Product | c.id = p.id implies p.count >= countToBeAdded AND (countToBeAdded + c.count) <=5)

post: self.CartItem -> forAll(c :CartItem, p :Product | c.id = p.id implies p.count >= c.count AND c.count <=5)

totalPrice = totalPrice@pre + c.count * c.Product.price

9. After the customer orders a product, they can add a review. The average rating of the product will be updated.

Context Product::addReview(r :Review)

pre: self.User -> forAll(u :user | u.oclIsTypeOf(Customer)) AND

self.Review -> excludes(r) AND

allInstances() -> includesAll(self.Customer.Order.PurchasedItem.Payment.Checkout.Cart.Product)

post: self.Review -> includes(r) AND

averageRating = (self.Review-> select(r.id = p.id) -> collect(rating) -> sum()) / self.Review-> select(r.id = p.id) -> size()

10. After the customer makes a payment and if payment is approved, their cart will be empty and the count of the purchased items will be subtracted from the count of the product.

Context Checkout::makePayment(p :Payment)

pre: self.Payment -> excludes(p) AND

self.Cart.CartItems -> isNotEmpty()

post:

self.Payment -> includes(p) AND

p.status = “Approved” implies self.Cart.CartItems -> isEmpty() AND self.Cart.Product -> forAll(c :CartItem, p :Product | c.id = p.id implies p.count = p.count@pre - c.count)

11. The delivery should be cancelled when it doesn’t arrive on time.

    Context Delivery

    inv: self.status = “Cancelled” implies ((Day(today) - Day(dateOfArrival)) > 0 AND self.status <> “Arrived”)

12. Maximum of 50 items can be added to the cart.

    Context Cart

    inv: self.CartItems -> collect(count) -> sum() <= 50

13. A review can only have a rating between 0 and 5.

    Context Review

    inv: self.rating >= 0 AND self.rating <= 5

14. When a cart item is removed from cart, its price will be subtracted from the cart’s total price.

    Context Cart::removeFromCart(c :CartItem, countToBeRemoved :int)

    pre: self.CartItem -> includes(c)

    post: c.count = c.count@pre - countToBeRemoved

    totalPrice = totalPrice@pre - c.Product.price * countToBeRemoved

15. If the total price in the checkout is less than $50, then there is a $5 delivery charge.

    Context Checkout

    inv: self..deliveryCharge = if(totalPrice < 50) then 5

    else 0

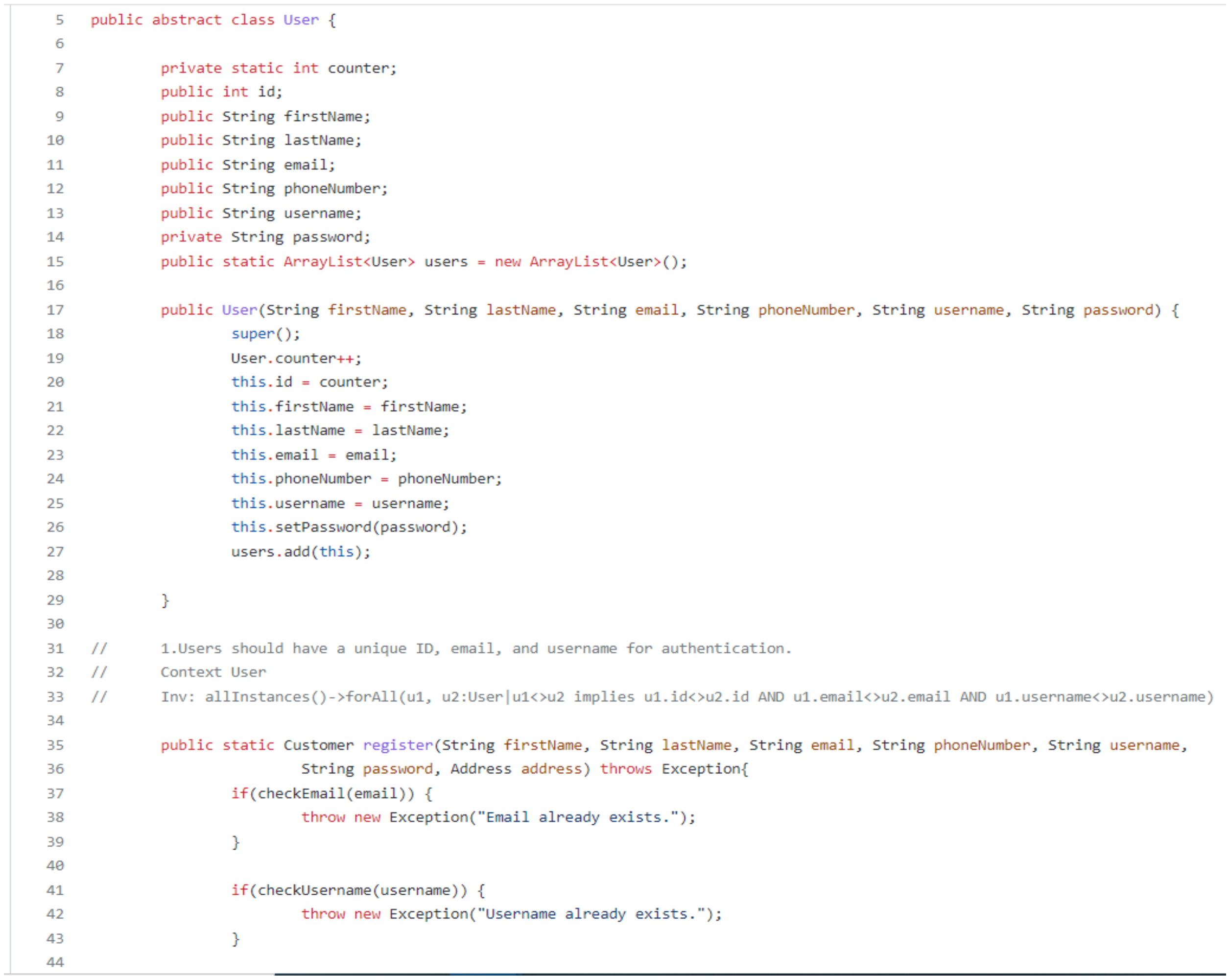
    endif

3. DESCRIPTION OF THE SYSTEM

In this section, the classes developed for the system is discussed. These are implemented in Java with referring to the updated class diagram.

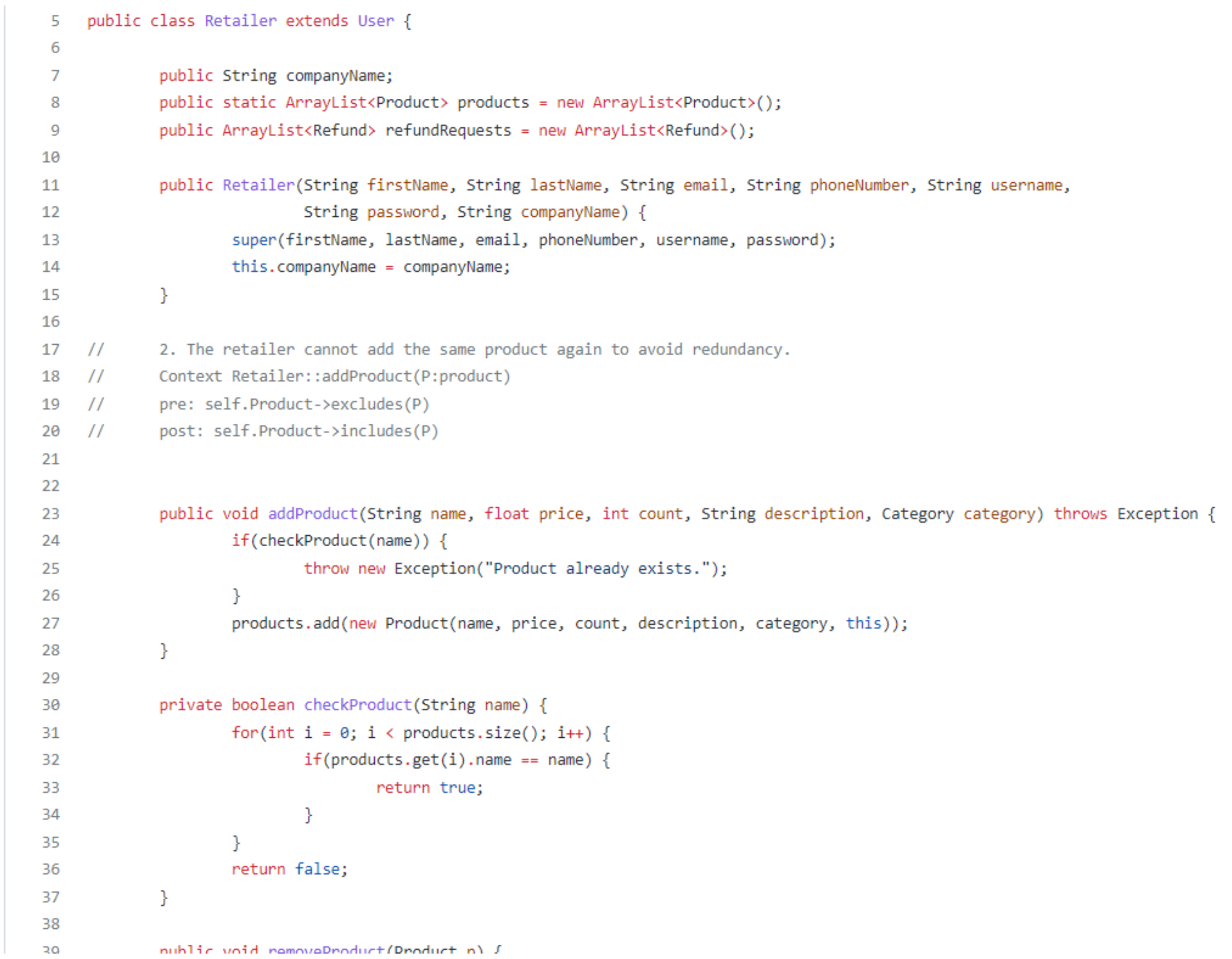
The User class (Customer and Retailer- Derived classes)

It contains the details of all the users accessing the system, which includes the retailers and the customers. It has the user information such as name, email, password and phone number. It also assigns a unique id to each user. It has two derived classes of Retailer and Customer. In the customer class the address along with other user details are stored. Similarly, in the retailer class the company name can be stored. The retailers also have the accessibility to add, modify or delete a product from the system.



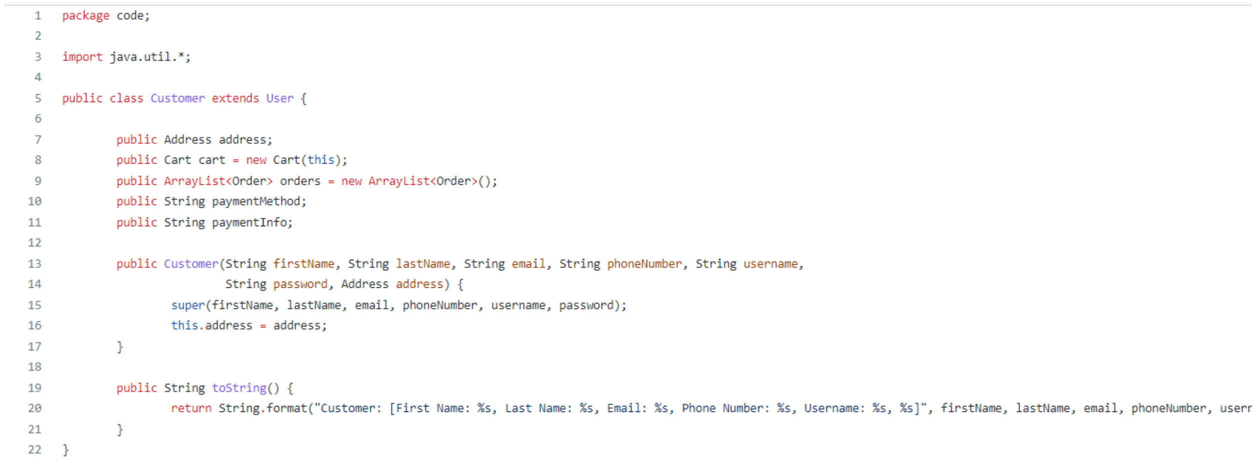
```java
public abstract class User {

        private static int counter;
        public int id;
        public String firstName;
        public String lastName;
        public String email;
        public String phoneNumber;
        public String username;
        private String password;
        public static ArrayList<User> users = new ArrayList<User>();

        public User(String firstName, String lastName, String email, String phoneNumber, String username, String password) {
                super();
                User.counter++;
                this.id = counter;
                this.firstName = firstName;
                this.lastName = lastName;
                this.email = email;
                this.phoneNumber = phoneNumber;
                this.username = username;
                this.setPassword(password);
                users.add(this);

        }

//      1.Users should have a unique ID, email, and username for authentication.
//      Context User
//      Inv: allInstances()->forAll(u1, u2:User|u1<>u2 implies u1.id<>u2.id AND u1.email<>u2.email AND u1.username<>u2.username)

        public static Customer register(String firstName, String lastName, String email, String phoneNumber, String username,
                        String password, Address address) throws Exception{
                if(checkEmail(email)) {
                        throw new Exception("Email already exists.");
                }

                if(checkUsername(username)) {
                        throw new Exception("Username already exists.");
                }

```

Figure 2 User class



```
public class Retailer extends User {

        public String companyName;
        public static ArrayList<Product> products = new ArrayList<Product>();
        public ArrayList<Refund> refundRequests = new ArrayList<Refund>();

        public Retailer(String firstName, String lastName, String email, String phoneNumber, String username,
                        String password, String companyName) {
                super(firstName, lastName, email, phoneNumber, username, password);
                this.companyName = companyName;
        }

//      2. The retailer cannot add the same product again to avoid redundancy.
//      Context Retailer::addProduct(P:product)
//      pre: self.Product->excludes(P)
//      post: self.Product->includes(P)


        public void addProduct(String name, float price, int count, String description, Category category) throws Exception {
                if(checkProduct(name)) {
                        throw new Exception("Product already exists.");
                }
                products.add(new Product(name, price, count, description, category, this));
        }

        private boolean checkProduct(String name) {
                for(int i = 0; i < products.size(); i++) {
                        if(products.get(i).name == name) {
                                return true;
                        }
                }
                return false;
        }

        public void removeProduct(Product p) {
```

Figure 3 Retailer class

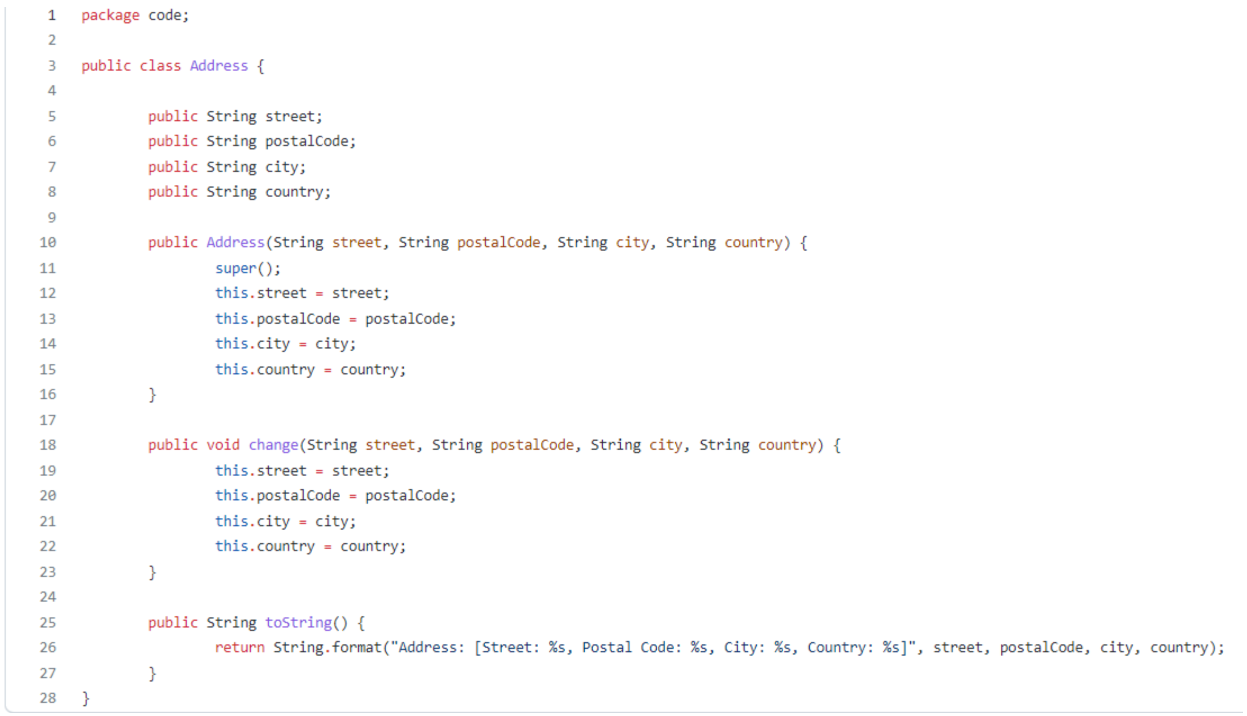


```
package code;

import java.util.*;

public class Customer extends User {

        public Address address;
        public Cart cart = new Cart(this);
        public ArrayList<Order> orders = new ArrayList<Order>();
        public String paymentMethod;
        public String paymentInfo;

        public Customer(String firstName, String lastName, String email, String phoneNumber, String username,
                        String password, Address address) {
                super(firstName, lastName, email, phoneNumber, username, password);
                this.address = address;
        }

        public String toString() {
                return String.format("Customer: [First Name: %s, Last Name: %s, Email: %s, Phone Number: %s, Username: %s, %s]", firstName, lastName, email, phoneNumber, userr
        }
}
```

Figure 4 Customer class

The Address class

The address details are given by the customers to deliver the purchased products. It contains information such as the street, city, country and postalcode. The customer can modify the address by using the change function.

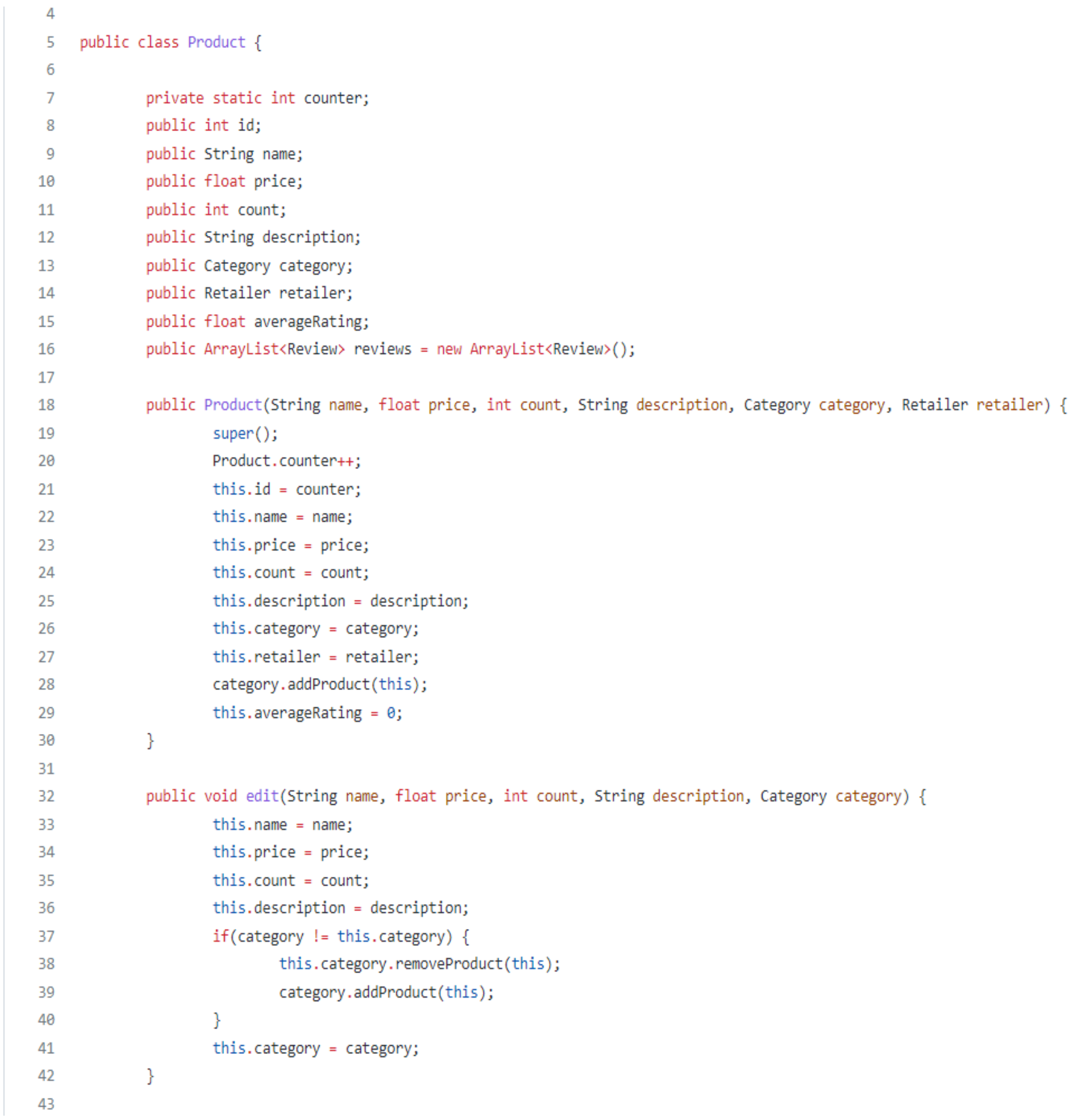


```java
package code;

public class Address {

        public String street;
        public String postalCode;
        public String city;
        public String country;

        public Address(String street, String postalCode, String city, String country) {
                super();
                this.street = street;
                this.postalCode = postalCode;
                this.city = city;
                this.country = country;
        }

        public void change(String street, String postalCode, String city, String country) {
                this.street = street;
                this.postalCode = postalCode;
                this.city = city;
                this.country = country;
        }

        public String toString() {
                return String.format("Address: [Street: %s, Postal Code: %s, City: %s, Country: %s]", street, postalCode, city, country);
        }
}
```

Figure 5 Address class

The Product class

In this class, the product information is stored which includes the product Id, name, price of the product, count, description of the product and the average rating. The category and review for the product is added by the retailer and customer, respectively. The addTocart and addReview is accessed by the customer to add the product to the cart or to add review to the purchased item. The retailer can edit the product by using the edit function.

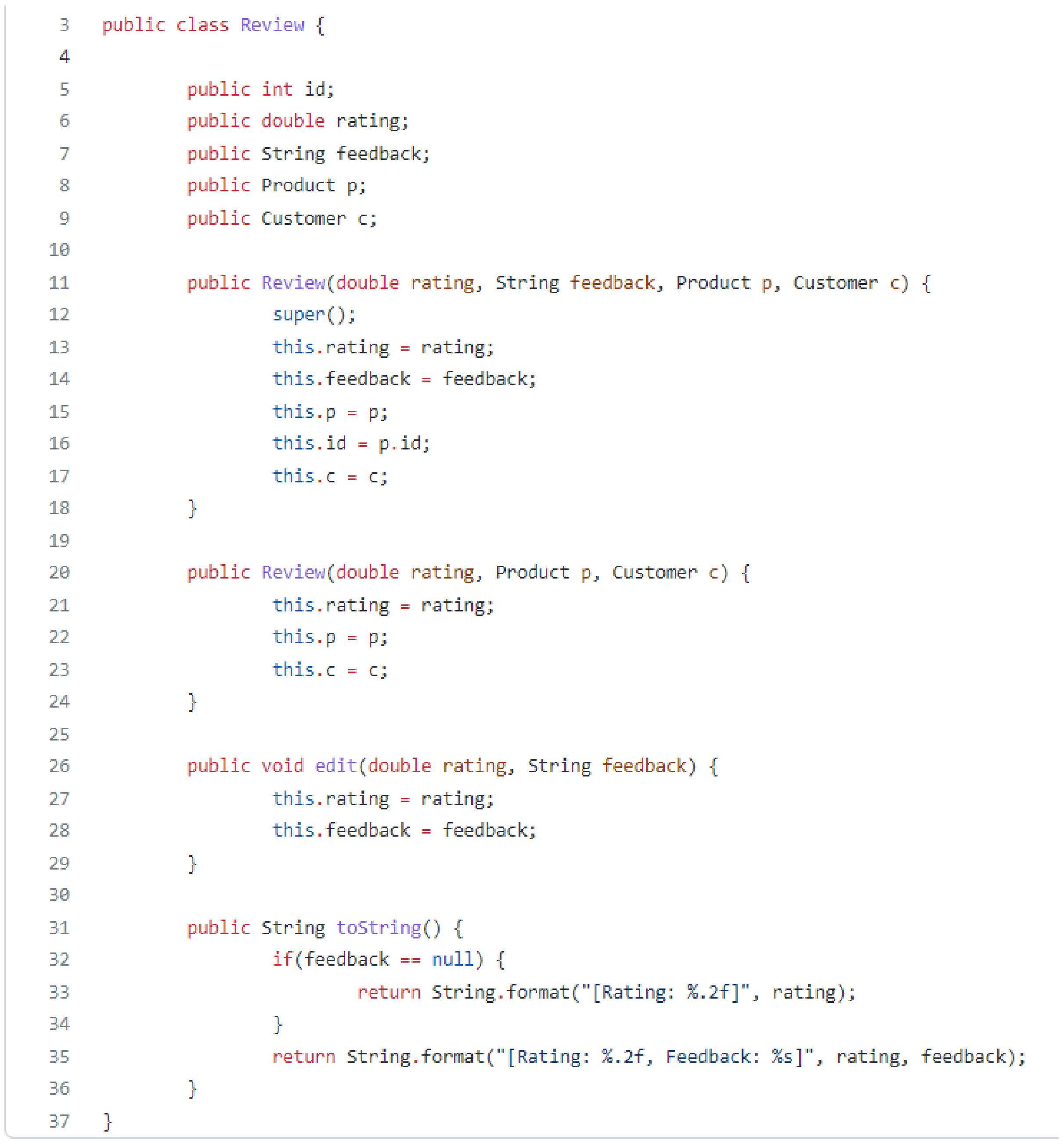


```java

public class Product {

        private static int counter;
        public int id;
        public String name;
        public float price;
        public int count;
        public String description;
        public Category category;
        public Retailer retailer;
        public float averageRating;
        public ArrayList<Review> reviews = new ArrayList<Review>();

        public Product(String name, float price, int count, String description, Category category, Retailer retailer) {
                super();
                Product.counter++;
                this.id = counter;
                this.name = name;
                this.price = price;
                this.count = count;
                this.description = description;
                this.category = category;
                this.retailer = retailer;
                category.addProduct(this);
                this.averageRating = 0;
        }

        public void edit(String name, float price, int count, String description, Category category) {
                this.name = name;
                this.price = price;
                this.count = count;
                this.description = description;
                if(category != this.category) {
                        this.category.removeProduct(this);
                        category.addProduct(this);
                }
                this.category = category;
        }

```

Figure 6 product class

The Review class

It is the associating class between the customer and the product class. It is used for adding the review for each item purchased by the customer. Each review has an ID. There is also a constraint given to the customer to rate the product between 0-5 and also give feedback.

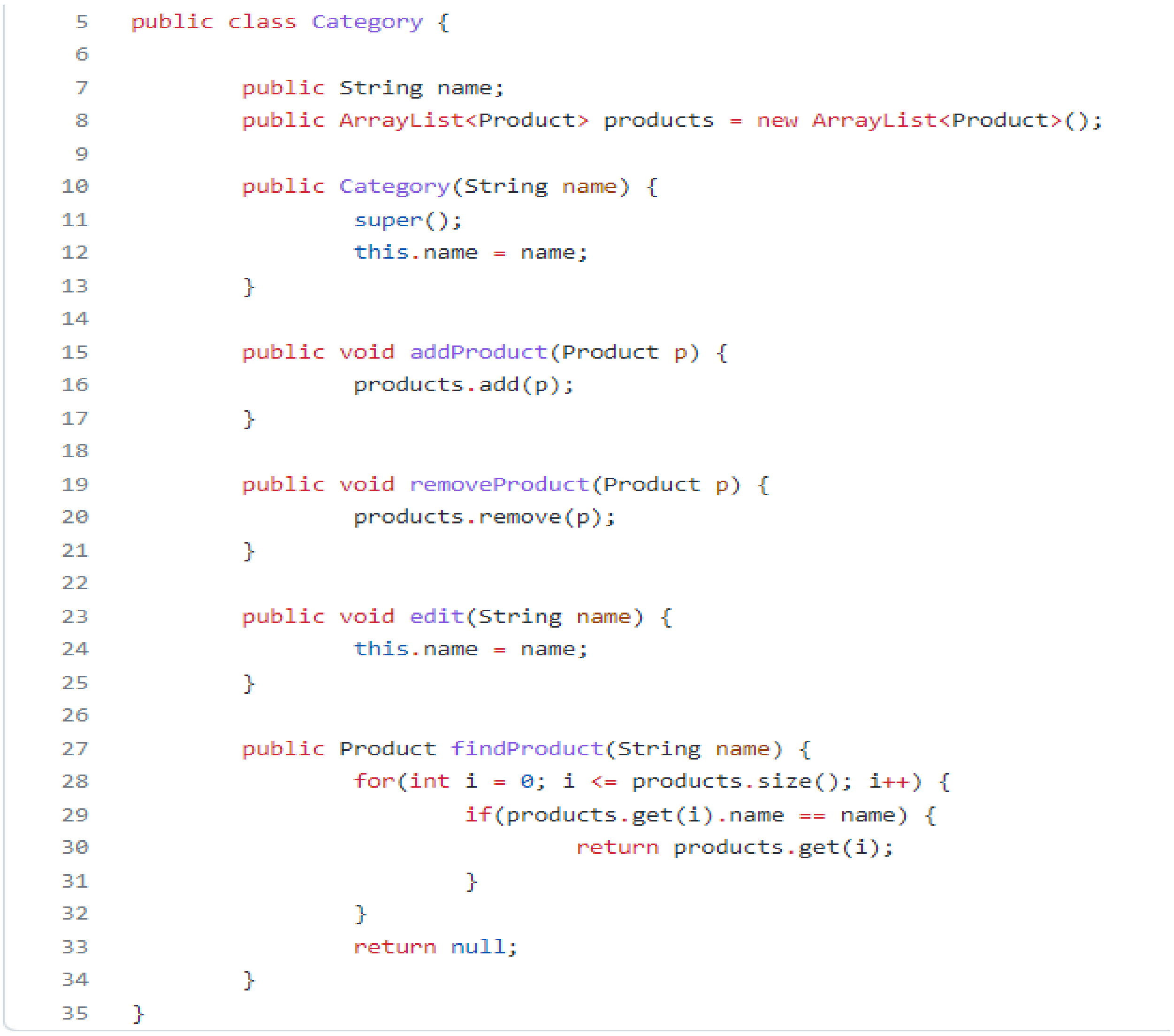


```java
public class Review {

        public int id;
        public double rating;
        public String feedback;
        public Product p;
        public Customer c;

        public Review(double rating, String feedback, Product p, Customer c) {
                super();
                this.rating = rating;
                this.feedback = feedback;
                this.p = p;
                this.id = p.id;
                this.c = c;
        }

        public Review(double rating, Product p, Customer c) {
                this.rating = rating;
                this.p = p;
                this.c = c;
        }

        public void edit(double rating, String feedback) {
                this.rating = rating;
                this.feedback = feedback;
        }

        public String toString() {
                if(feedback == null) {
                        return String.format("[Rating: %.2f]", rating);
                }
                return String.format("[Rating: %.2f, Feedback: %s]", rating, feedback);
        }
}
```

Figure 7 Review class

The Category class

It organizes the products into various categories given by the administrator. The retailer assigns a suitable category to the product. The product can also be added, edited or removed from a category. The product can be searched by the customer with the product name using findProduct function.

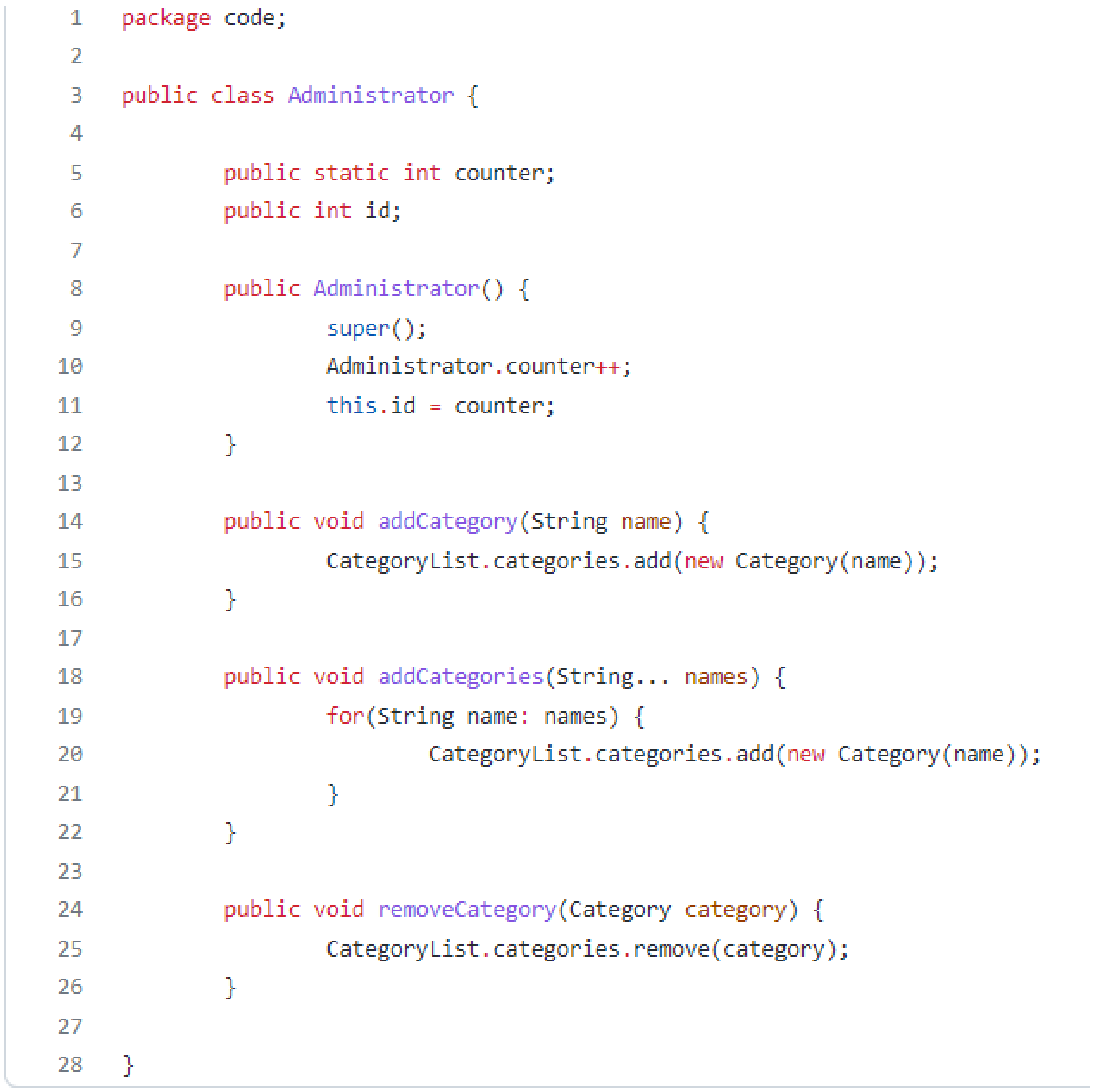


```java
public class Category {

        public String name;
        public ArrayList<Product> products = new ArrayList<Product>();

        public Category(String name) {
                super();
                this.name = name;
        }

        public void addProduct(Product p) {
                products.add(p);
        }

        public void removeProduct(Product p) {
                products.remove(p);
        }

        public void edit(String name) {
                this.name = name;
        }

        public Product findProduct(String name) {
                for(int i = 0; i <= products.size(); i++) {
                        if(products.get(i).name == name) {
                                return products.get(i);
                        }
                }
                return null;
        }
}
```

Figure 8 Category class

The Administrator class

Only the administrator can add a new category to the system or remove an existing category. Each administrator has a unique Id.

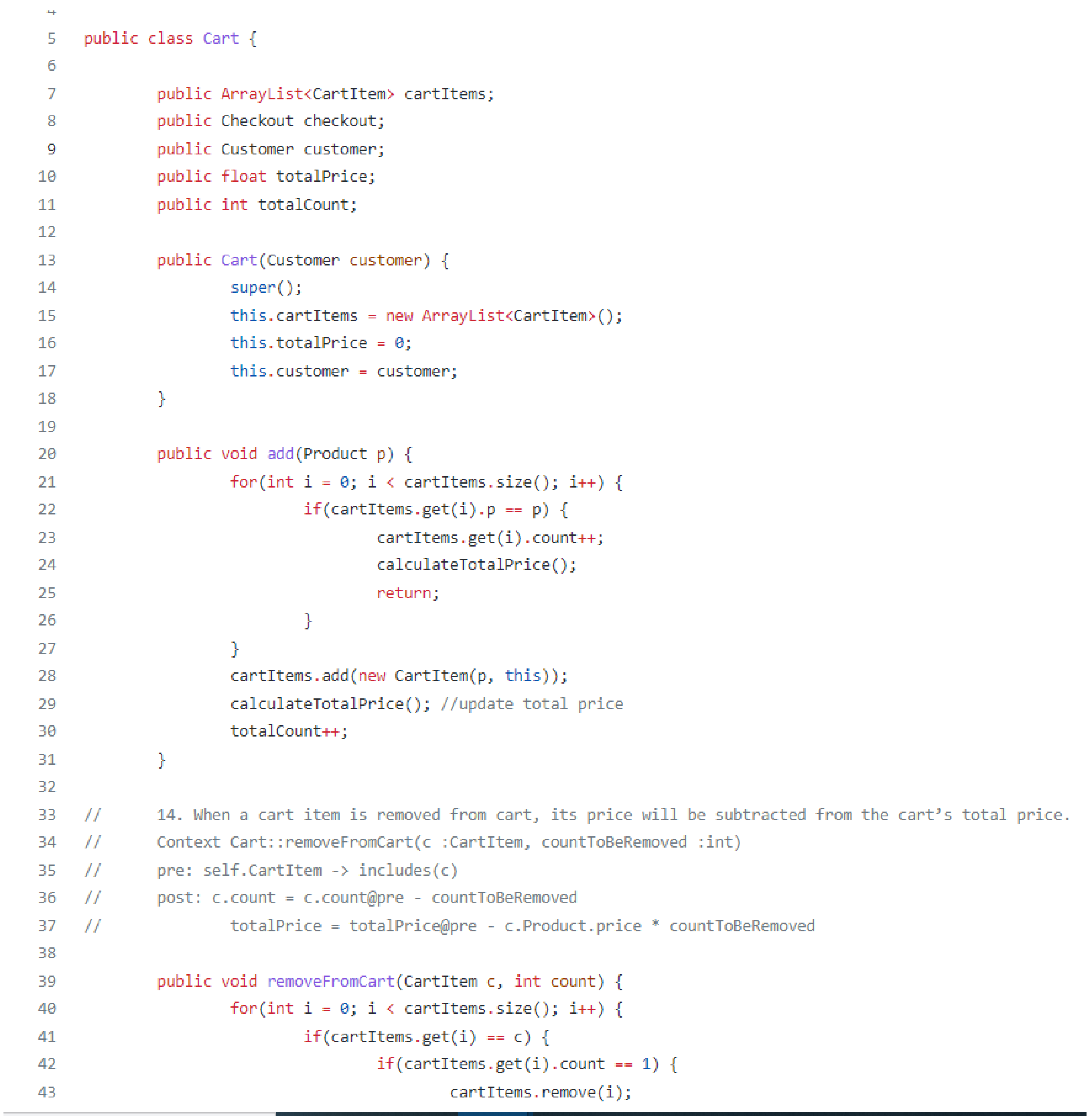


```java
package code;

public class Administrator {

        public static int counter;
        public int id;

        public Administrator() {
                super();
                Administrator.counter++;
                this.id = counter;
        }

        public void addCategory(String name) {
                CategoryList.categories.add(new Category(name));
        }

        public void addCategories(String... names) {
                for(String name: names) {
                        CategoryList.categories.add(new Category(name));
                }
        }

        public void removeCategory(Category category) {
                CategoryList.categories.remove(category);
        }

}
```

Figure 9 Administrator class

The Cart class

The total price of the products added in the cart is given in this class. When the product is added or removed from the cart the price is varied accordingly. Later the customer can also proceed to the checkout using the proceedToCheckout function and the cart is cleared after purchase using empty function.

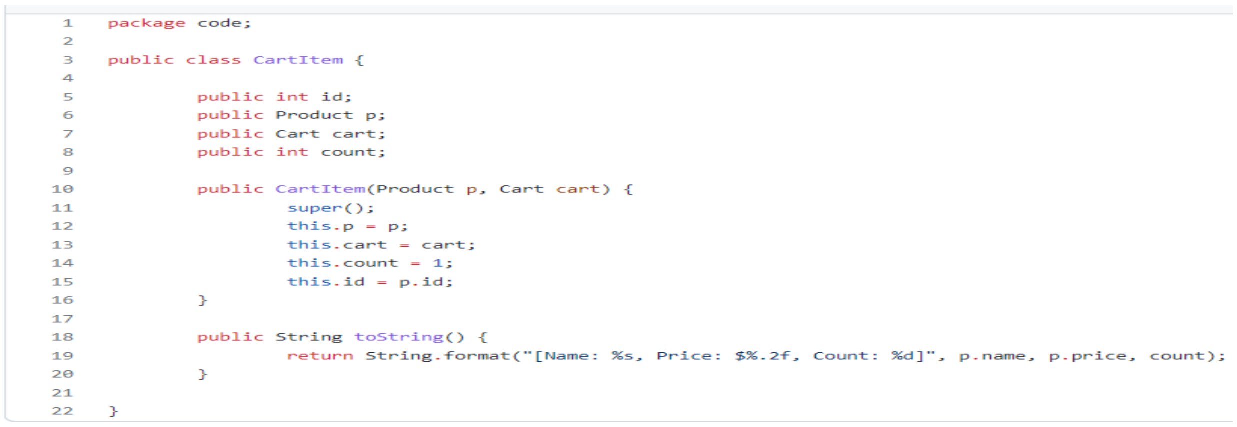


```java
public class Cart {

    public ArrayList<CartItem> cartItems;
    public Checkout checkout;
    public Customer customer;
    public float totalPrice;
    public int totalCount;

    public Cart(Customer customer) {
        super();
        this.cartItems = new ArrayList<CartItem>();
        this.totalPrice = 0;
        this.customer = customer;
    }

    public void add(Product p) {
        for(int i = 0; i < cartItems.size(); i++) {
            if(cartItems.get(i).p == p) {
                cartItems.get(i).count++;
                calculateTotalPrice();
                return;
            }
        }
        cartItems.add(new CartItem(p, this));
        calculateTotalPrice(); //update total price
        totalCount++;
    }

//  14. When a cart item is removed from cart, its price will be subtracted from the cart's total price.
//  Context Cart::removeFromCart(c :CartItem, countToBeRemoved :int)
//  pre: self.CartItem -> includes(c)
//  post: c.count = c.count@pre - countToBeRemoved
//      totalPrice = totalPrice@pre - c.Product.price * countToBeRemoved

    public void removeFromCart(CartItem c, int count) {
        for(int i = 0; i < cartItems.size(); i++) {
            if(cartItems.get(i) == c) {
                if(cartItems.get(i).count == 1) {
                    cartItems.remove(i);
```

Figure 10 Cart class

The CartItem class

This is an association class between the product and cart. The number of same type product to be added to the cart is given by the count attribute. There is also an option to remove the item from the cart. The id is used for subtracting the product count with the cartitem count after the product is being purchased.

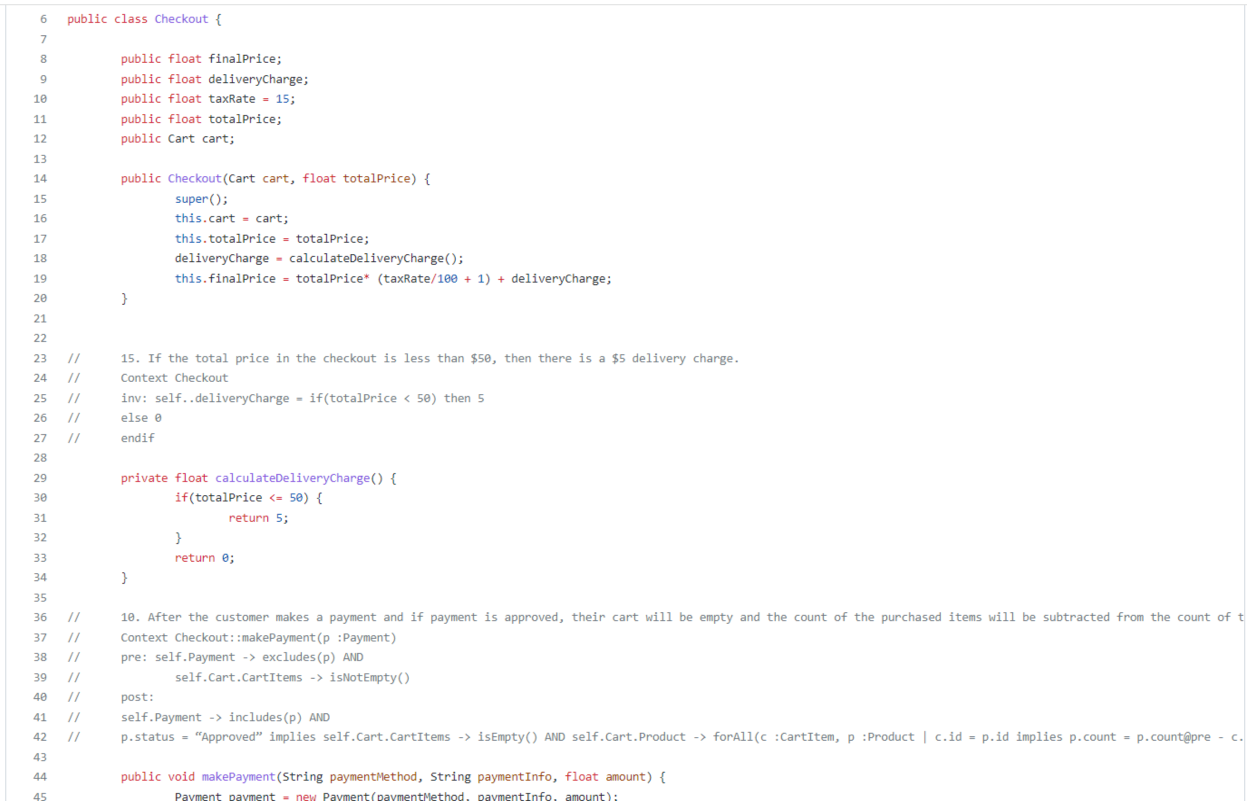


```java
package code;

public class CartItem {

	public int id;
	public Product p;
	public Cart cart;
	public int count;

	public CartItem(Product p, Cart cart) {
		super();
		this.p = p;
		this.cart = cart;
		this.count = 1;
		this.id = p.id;
	}

	public String toString() {
		return String.format("[Name: %s, Price: $%.2f, Count: %d]", p.name, p.price, count);
	}

}
```

Figure 11 Cartitem class

The Checkout class

This class gives the final price of all the products that the customer is going to purchase. The final price is computed after adding the delivery charge and tax amount to the total price. There is also a constraint that the delivery charge will be excluded if the total price of the product is more than $50 else there is a delivery charge of $5. The customer can either cancel his purchase using cancelCheckout function or can further proceed to pay using makePayment function.

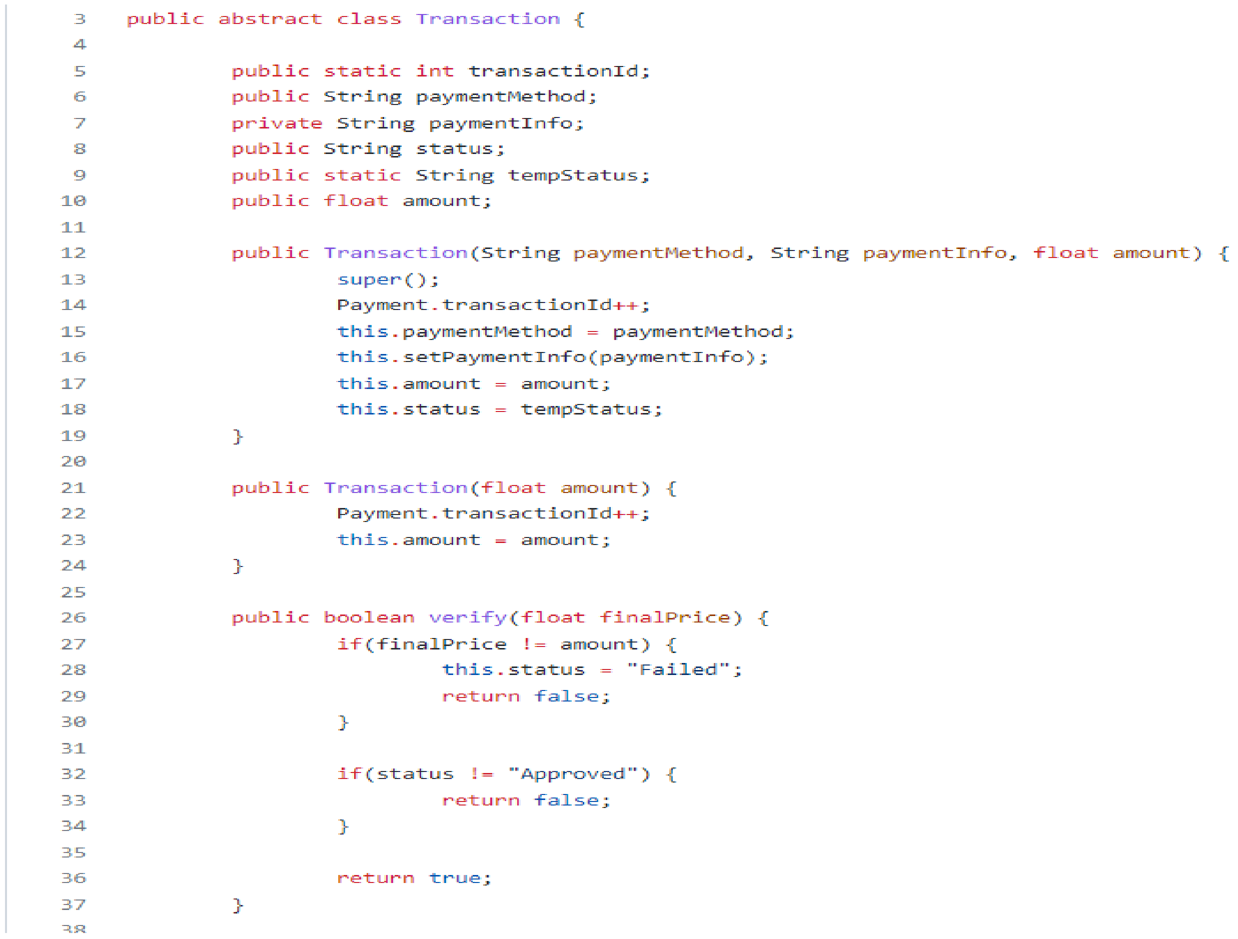


```java
public class Checkout {

	public float finalPrice;
	public float deliveryCharge;
	public float taxRate = 15;
	public float totalPrice;
	public Cart cart;

	public Checkout(Cart cart, float totalPrice) {
		super();
		this.cart = cart;
		this.totalPrice = totalPrice;
		deliveryCharge = calculateDeliveryCharge();
		this.finalPrice = totalPrice* (taxRate/100 + 1) + deliveryCharge;
	}


//	15. If the total price in the checkout is less than $50, then there is a $5 delivery charge.
//	Context Checkout
//	inv: self..deliveryCharge = if(totalPrice < 50) then 5
//	else 0
//	endif

	private float calculateDeliveryCharge() {
		if(totalPrice <= 50) {
			return 5;
		}
		return 0;
	}

//	10. After the customer makes a payment and if payment is approved, their cart will be empty and the count of the purchased items will be subtracted from the count of t
//	Context Checkout::makePayment(p :Payment)
//	pre: self.Payment -> excludes(p) AND
//		self.Cart.CartItems -> isNotEmpty()
//	post:
//	self.Payment -> includes(p) AND
//	p.status = “Approved” implies self.Cart.CartItems -> isEmpty() AND self.Cart.Product -> forAll(c :CartItem, p :Product | c.id = p.id implies p.count = p.count@pre - c.

	public void makePayment(String paymentMethod, String paymentInfo, float amount) {
		Payment payment = new Payment(paymentMethod, paymentInfo, amount);
```

Figure 12 Checkout class

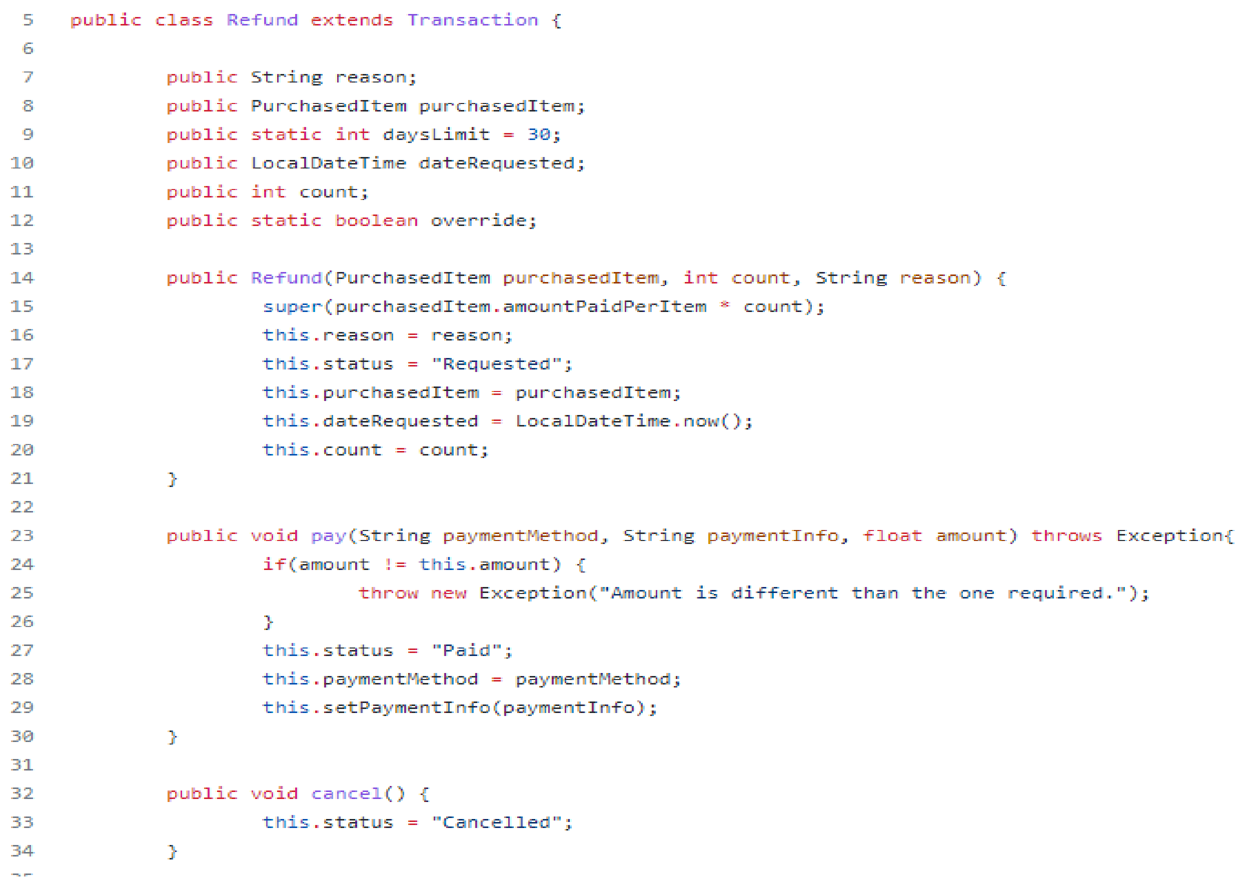
The Transaction class (Refund and Payment- Derived classes)

This contains the customer information of payment method and paymentInfo. For payment of each purchase the amount to be paid and the status is provided. Each transaction has unique transactionId. The verification is done to check the status of the payment and based on it the cartitems are confirmed as purchased items. It has two derived classes of refund and payment. The Payment class is used when the customer would like to purchase the products confirmed in the checkout. Once the product is arrived, if the customer wants to receive a refund for the product he wants to cancel, the refund class is accessed. It includes the reason for the cancelation of the purchased Item. It also has a constraint that the cancellation should be done within the 30 days of the product purchase using the canRequestRefund function. The refund is given back to the customer by adding back the refund amount to the payment account provided by the customer using the payRefund function.



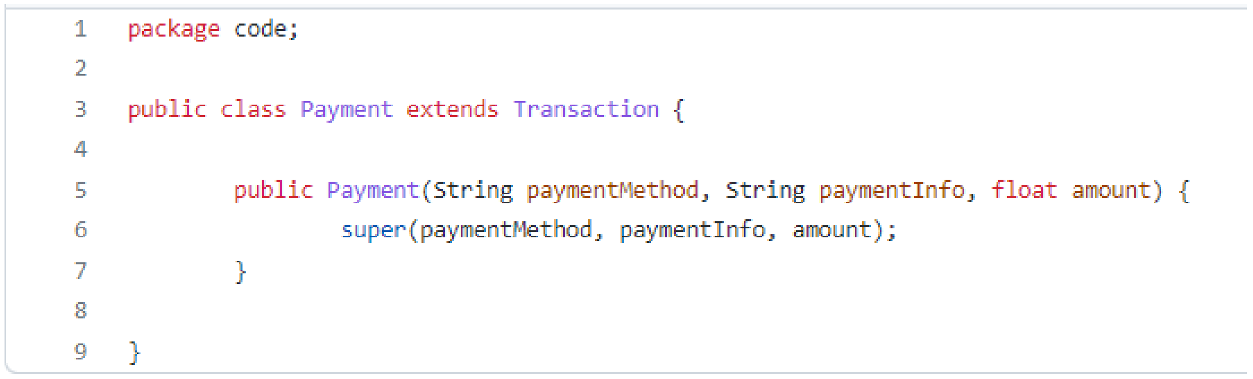
```
public abstract class Transaction {

        public static int transactionId;
        public String paymentMethod;
        private String paymentInfo;
        public String status;
        public static String tempStatus;
        public float amount;

        public Transaction(String paymentMethod, String paymentInfo, float amount) {
                super();
                Payment.transactionId++;
                this.paymentMethod = paymentMethod;
                this.setPaymentInfo(paymentInfo);
                this.amount = amount;
                this.status = tempStatus;
        }

        public Transaction(float amount) {
                Payment.transactionId++;
                this.amount = amount;
        }

        public boolean verify(float finalPrice) {
                if(finalPrice != amount) {
                        this.status = "Failed";
                        return false;
                }

                if(status != "Approved") {
                        return false;
                }

                return true;
        }
```

Figure 13 Transaction class



```java
public class Refund extends Transaction {

        public String reason;
        public PurchasedItem purchasedItem;
        public static int daysLimit = 30;
        public LocalDateTime dateRequested;
        public int count;
        public static boolean override;

        public Refund(PurchasedItem purchasedItem, int count, String reason) {
                super(purchasedItem.amountPaidPerItem * count);
                this.reason = reason;
                this.status = "Requested";
                this.purchasedItem = purchasedItem;
                this.dateRequested = LocalDateTime.now();
                this.count = count;
        }

        public void pay(String paymentMethod, String paymentInfo, float amount) throws Exception{
                if(amount != this.amount) {
                        throw new Exception("Amount is different than the one required.");
                }
                this.status = "Paid";
                this.paymentMethod = paymentMethod;
                this.setPaymentInfo(paymentInfo);
        }

        public void cancel() {
                this.status = "Cancelled";
        }
```

Figure 14 Refund class

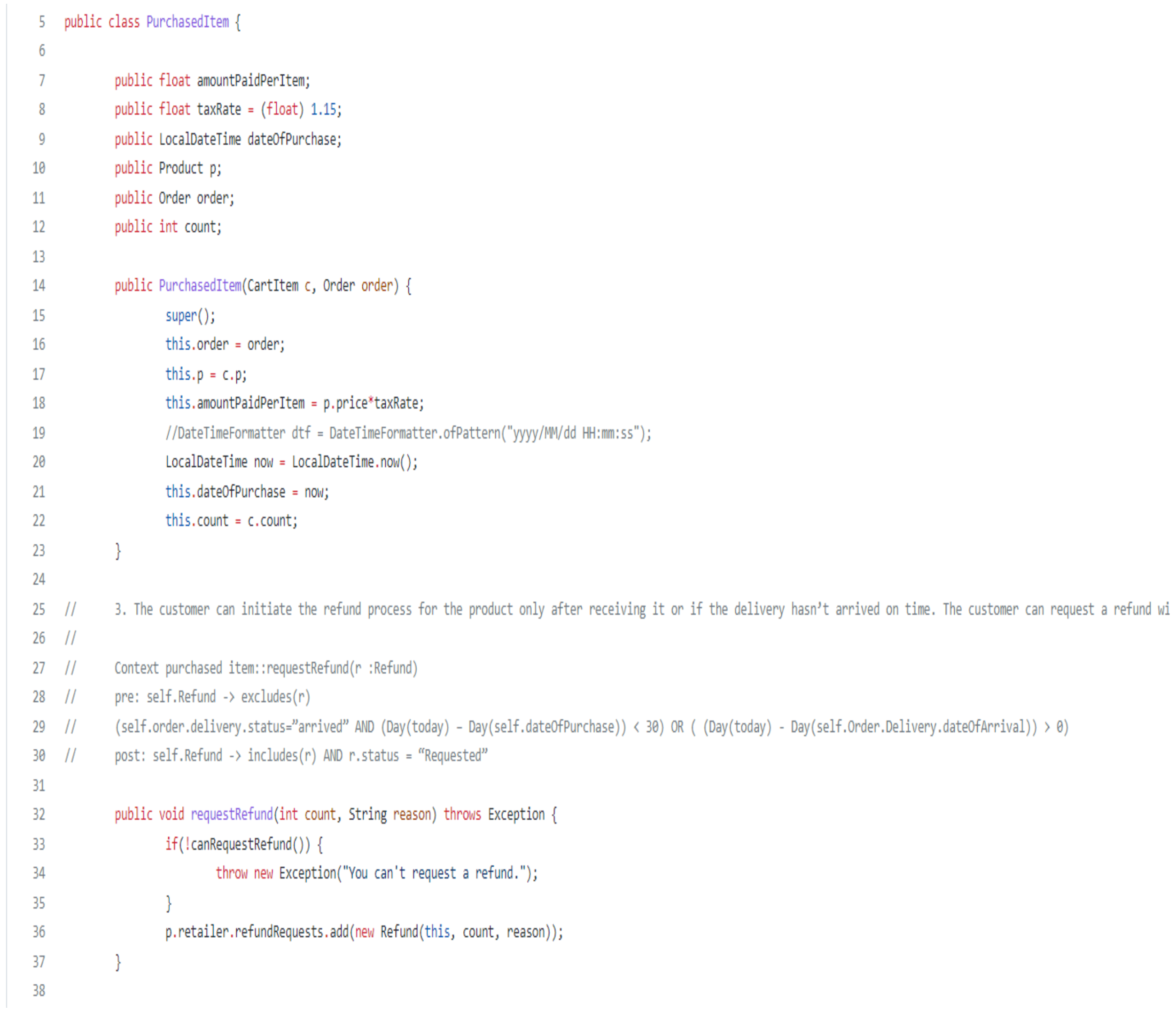


```java
package code;

public class Payment extends Transaction {

        public Payment(String paymentMethod, String paymentInfo, float amount) {
                super(paymentMethod, paymentInfo, amount);
        }

}
```

Figure 15 Payment class

The PurchasedItem class

This class includes the information (amountPaidPerItem, taxRate, dateOfPurchase and count) of the items purchased by customer. The refund for the item purchased can be requested by the customer using the requestRefund function.

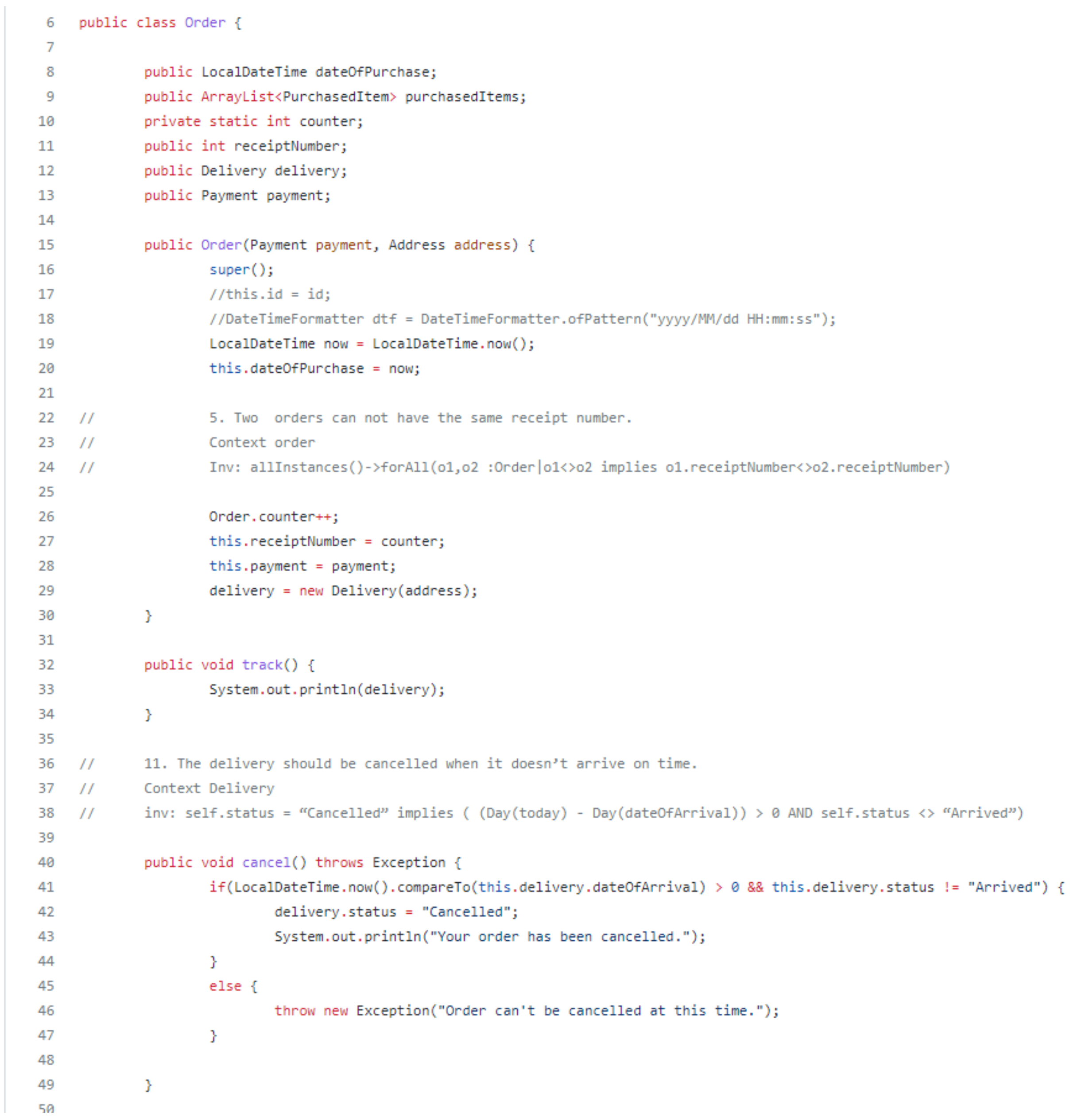


```java
public class PurchasedItem {

        public float amountPaidPerItem;
        public float taxRate = (float) 1.15;
        public LocalDateTime dateOfPurchase;
        public Product p;
        public Order order;
        public int count;

        public PurchasedItem(CartItem c, Order order) {
                super();
                this.order = order;
                this.p = c.p;
                this.amountPaidPerItem = p.price*taxRate;
                //DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");
                LocalDateTime now = LocalDateTime.now();
                this.dateOfPurchase = now;
                this.count = c.count;
        }

//      3. The customer can initiate the refund process for the product only after receiving it or if the delivery hasn't arrived on time. The customer can request a refund wi
//
//      Context purchased item::requestRefund(r :Refund)
//      pre: self.Refund -> excludes(r)
//      (self.order.delivery.status="arrived" AND (Day(today) - Day(self.dateOfPurchase)) < 30) OR ( (Day(today) - Day(self.Order.Delivery.dateOfArrival)) > 0)
//      post: self.Refund -> includes(r) AND r.status = "Requested"

        public void requestRefund(int count, String reason) throws Exception {
                if(!canRequestRefund()) {
                        throw new Exception("You can't request a refund.");
                }
                p.retailer.refundRequests.add(new Refund(this, count, reason));
        }

```

Figure 16 PurchasedItem class

The Order class

It is an association class between payment and checkout. It includes all the purchasedItem details and assigns a unique receiptNumber. It also includes the delivery details of the customer. If the order is cancelled the delivery status is updated to cancelled using the cancel function. The customer can track the order using the track function.

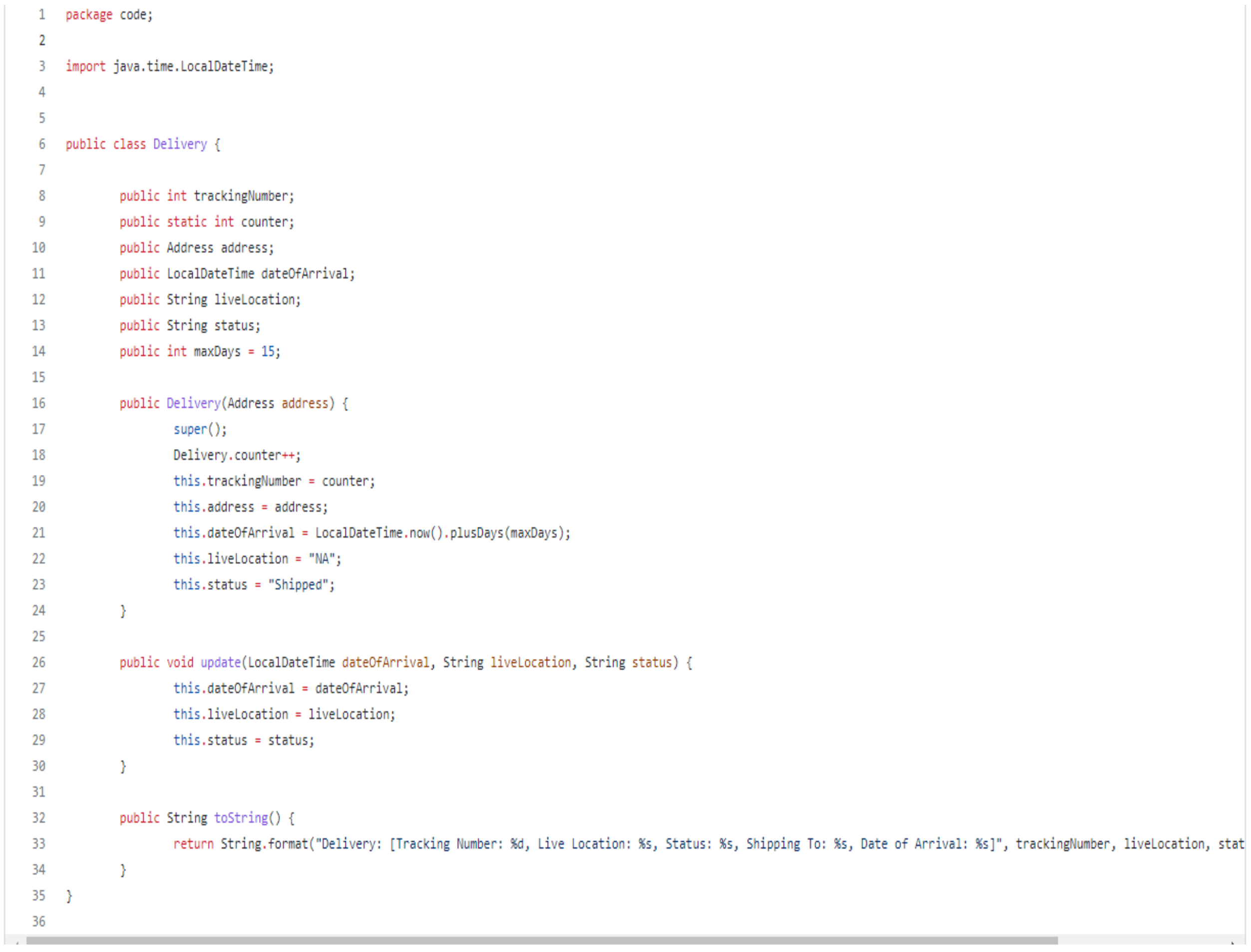


```java
public class Order {

        public LocalDateTime dateOfPurchase;
        public ArrayList<PurchasedItem> purchasedItems;
        private static int counter;
        public int receiptNumber;
        public Delivery delivery;
        public Payment payment;

        public Order(Payment payment, Address address) {
                super();
                //this.id = id;
                //DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");
                LocalDateTime now = LocalDateTime.now();
                this.dateOfPurchase = now;

//              5. Two  orders can not have the same receipt number.
//              Context order
//              Inv: allInstances()->forAll(o1,o2 :Order|o1<>o2 implies o1.receiptNumber<>o2.receiptNumber)

                Order.counter++;
                this.receiptNumber = counter;
                this.payment = payment;
                delivery = new Delivery(address);
        }

        public void track() {
                System.out.println(delivery);
        }

//      11. The delivery should be cancelled when it doesn’t arrive on time.
//      Context Delivery
//      inv: self.status = “Cancelled” implies ( (Day(today) - Day(dateOfArrival)) > 0 AND self.status <> “Arrived”)

        public void cancel() throws Exception {
                if(LocalDateTime.now().compareTo(this.delivery.dateOfArrival) > 0 && this.delivery.status != "Arrived") {
                        delivery.status = "Cancelled";
                        System.out.println("Your order has been cancelled.");
                }
                else {
                        throw new Exception("Order can't be cancelled at this time.");
                }

        }

```

Figure 17 Order class

The Delivery class

It includes the delivery details such as the address, unique trackingNumber, dateOfArrival, liveLocation and maxDays by which the product must be delivered. There is also a status attribute which gets updated based on the delivery.

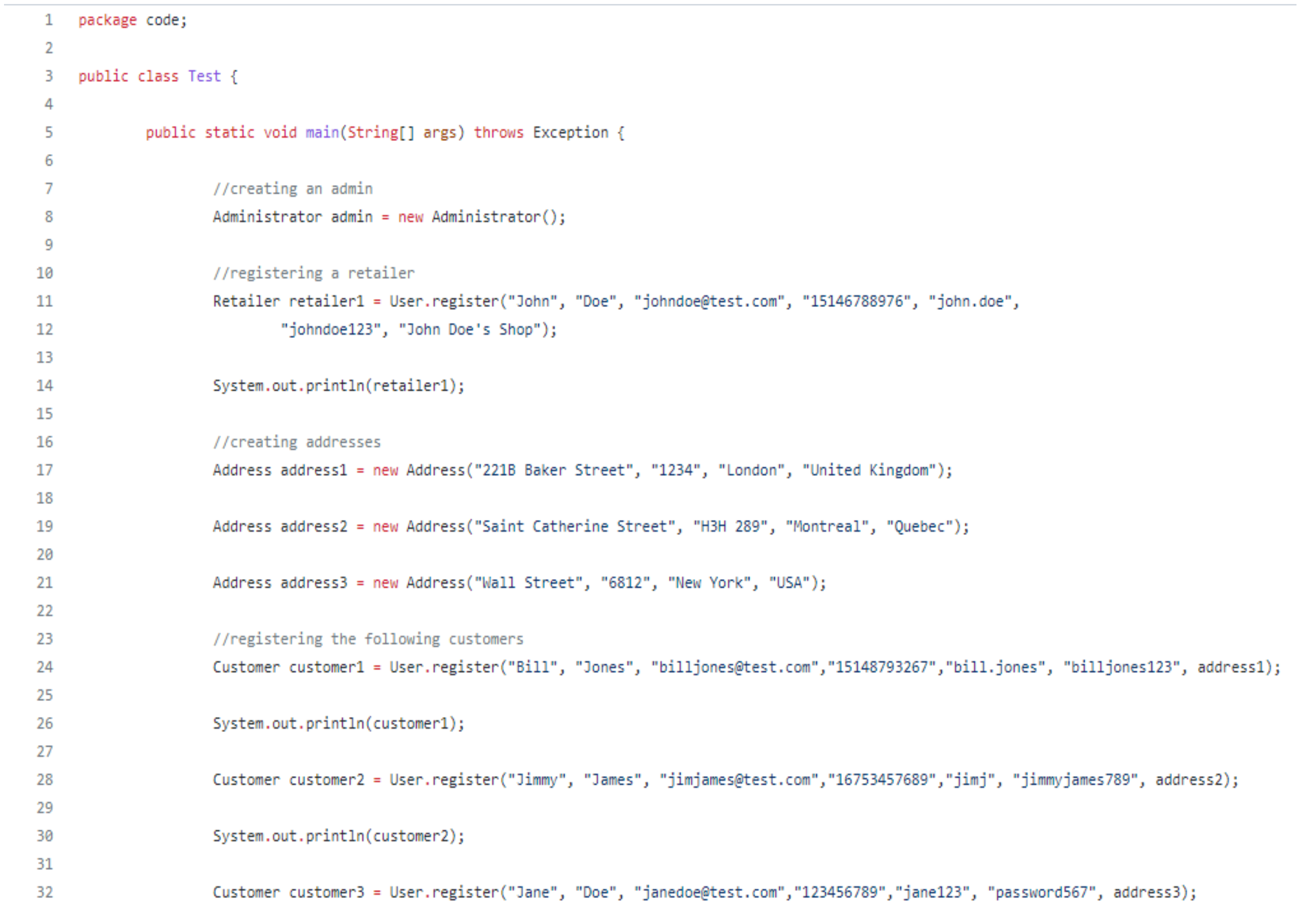


```java
package code;

import java.time.LocalDateTime;


public class Delivery {

        public int trackingNumber;
        public static int counter;
        public Address address;
        public LocalDateTime dateOfArrival;
        public String liveLocation;
        public String status;
        public int maxDays = 15;

        public Delivery(Address address) {
                super();
                Delivery.counter++;
                this.trackingNumber = counter;
                this.address = address;
                this.dateOfArrival = LocalDateTime.now().plusDays(maxDays);
                this.liveLocation = "NA";
                this.status = "Shipped";
        }

        public void update(LocalDateTime dateOfArrival, String liveLocation, String status) {
                this.dateOfArrival = dateOfArrival;
                this.liveLocation = liveLocation;
                this.status = status;
        }

        public String toString() {
                return String.format("Delivery: [Tracking Number: %d, Live Location: %s, Status: %s, Shipping To: %s, Date of Arrival: %s]", trackingNumber, liveLocation, stat
        }
}

```

Figure 18 Delivery Class

The Driver class (Test)

In the driver, we tested the whole code. First, we created the main actors of the system (Administrator, Retailer, and Customer). After that, we tested the main functions of each class. Also, we tested the implementation of the OCL constraints which were represented by exceptions as shown in the driver. Whenever a constraint is violated, an exception is thrown. Please check the comments in the driver file of the code for further details.



```java
package code;

public class Test {

        public static void main(String[] args) throws Exception {

                //creating an admin
                Administrator admin = new Administrator();

                //registering a retailer
                Retailer retailer1 = User.register("John", "Doe", "johndoe@test.com", "15146788976", "john.doe",
                        "johndoe123", "John Doe's Shop");

                System.out.println(retailer1);

                //creating addresses
                Address address1 = new Address("221B Baker Street", "1234", "London", "United Kingdom");

                Address address2 = new Address("Saint Catherine Street", "H3H 289", "Montreal", "Quebec");

                Address address3 = new Address("Wall Street", "6812", "New York", "USA");

                //registering the following customers
                Customer customer1 = User.register("Bill", "Jones", "billjones@test.com","15148793267","bill.jones", "billjones123", address1);

                System.out.println(customer1);

                Customer customer2 = User.register("Jimmy", "James", "jimjames@test.com","16753457689","jimj", "jimmyjames789", address2);

                System.out.println(customer2);

                Customer customer3 = User.register("Jane", "Doe", "janedoe@test.com","123456789","jane123", "password567", address3);
```

Figure 19 Driver class

4. GitHub LINK

https://github.com/abedk21/ecommerce-system

5. REFERENCES

[1] Class lectures of COEN 6312 by Prof. Abdelwahab Hamou-Lhadj

[2[Sommerville, I. (2016). Software engineering