/\*\*

\* The Customer class represents a customer in the e-commerce system. It includes attributes such as

\* customer ID, name, email, phone number, and address. The class also manages the customer's shopping

\* cart, allowing products to be added or removed, and calculates the total cost of the items in the cart.

\* Additionally, it provides methods to place an order and view the shopping cart's contents.

\*/

package com.ecommerce;

import java.util.HashMap;

import java.util.Map;

import com.ecommerce.orders.Order;

public class Customer {

private int customerID;

private String name;

private String email;

private String phone;

private String address;

private Map<Product, Integer> shoppingCart;

public Customer(int customerID, String name, String email, String phone, String address) {

if (customerID <= 0) {

throw new IllegalArgumentException("Customer ID must be positive.");

}

if (name == null || name.isEmpty()) {

throw new IllegalArgumentException("Customer name cannot be null or empty.");

}

if (email == null || email.isEmpty() || !email.contains("@")) {

throw new IllegalArgumentException("Invalid email address.");

}

if (phone == null || phone.isEmpty()) {

throw new IllegalArgumentException("Phone number cannot be null or empty.");

}

if (address == null || address.isEmpty()) {

throw new IllegalArgumentException("Address cannot be null or empty.");

}

this.customerID = customerID;

this.name = name;

this.email = email;

this.phone = phone;

this.address = address;

this.shoppingCart = new HashMap<>();

}

public int getCustomerID() {

return customerID;

}

public String getName() {

return name;

}

public String getEmail() {

return email;

}

public String getPhone() {

return phone;

}

public String getAddress() {

return address;

}

public Map<Product, Integer> getShoppingCart() {

return shoppingCart;

}

public void addToCart(Product product, int quantity) {

if (product == null) {

throw new IllegalArgumentException("Product cannot be null.");

}

if (quantity <= 0) {

throw new IllegalArgumentException("Quantity must be positive.");

}

shoppingCart.put(product, shoppingCart.getOrDefault(product, 0) + quantity);

}

public void removeFromCart(Product product, int quantity) {

if (product == null) {

throw new IllegalArgumentException("Product cannot be null.");

}

if (quantity <= 0) {

throw new IllegalArgumentException("Quantity must be positive.");

}

if (!shoppingCart.containsKey(product)) {

throw new IllegalArgumentException("Product not in cart.");

}

int currentQuantity = shoppingCart.get(product);

if (currentQuantity <= quantity) {

shoppingCart.remove(product);

} else {

shoppingCart.put(product, currentQuantity - quantity);

}

}

public double calculateTotal() {

double total = 0;

for (Map.Entry<Product, Integer> entry : shoppingCart.entrySet()) {

total += entry.getKey().getPrice() \* entry.getValue();

}

return total;

}

public void placeOrder(int orderID) {

if (orderID <= 0) {

throw new IllegalArgumentException("Order ID must be positive.");

}

Order order = new Order(orderID, this);

order.generateOrderSummary();

shoppingCart.clear();

}

@Override

public String toString() {

return "Customer [ID=" + customerID + ", Name=" + name + ", Email=" + email +

", Phone=" + phone + ", Address=" + address + "]";

}

}