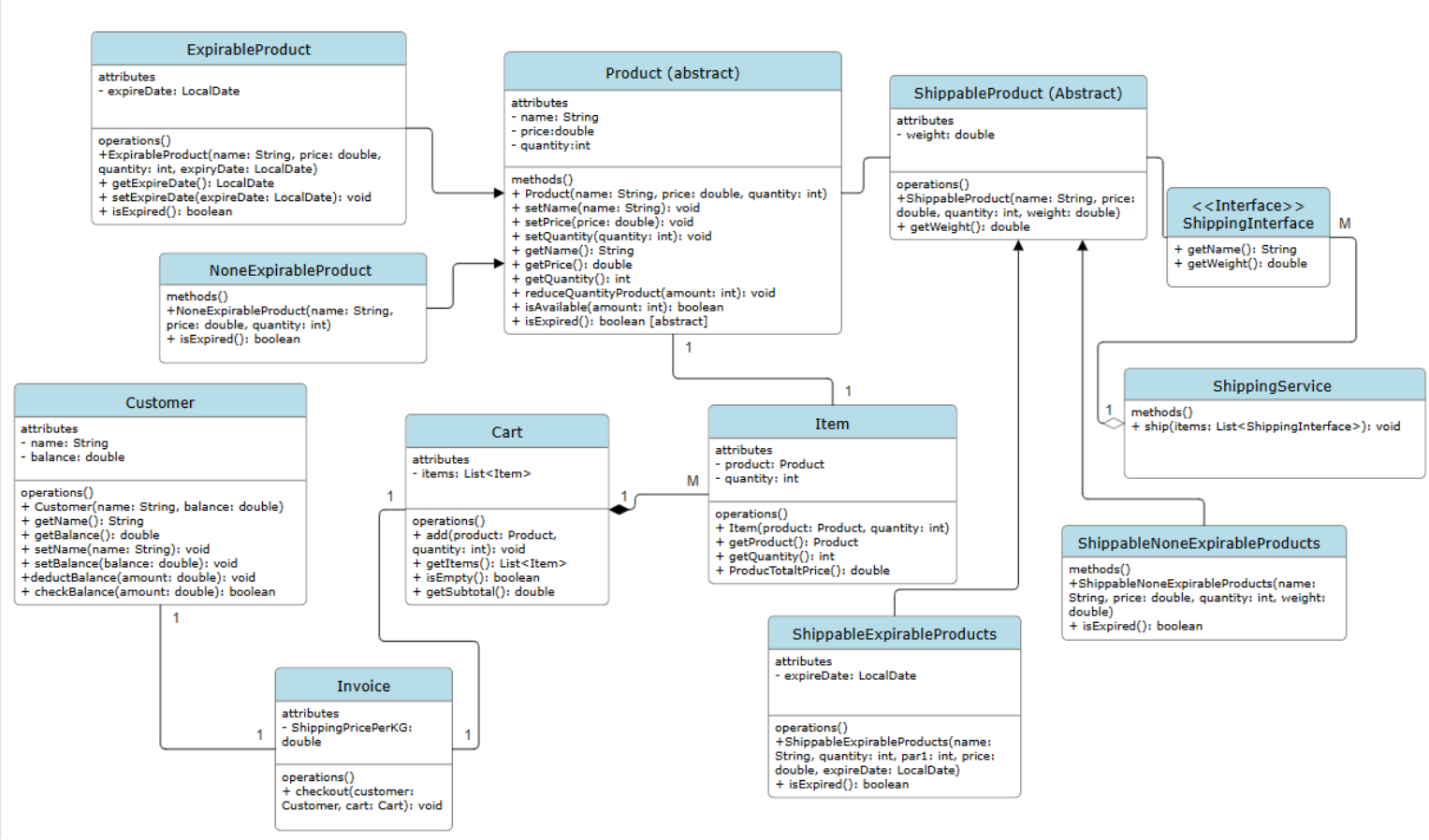
E-Commerce System

Mohamed Ashraf Khalaf Hafez

1st UML Class Diagram:



2nd UML Class Diagram:

Objects for testing:

```
//TESTING
Product cheese = new ShippableExpirableProducts("Cheese", 120, 6, 0.3, LocalDate.now().plusDays(2));
Product biscuits = new ShippableExpirableProducts("Biscuits", 180, 4, 0.8, LocalDate.now().plusDays(1));
Product tv = new ShippableNoneExpirableProducts("TV", 350, 2, 4.5);
Product scratchCard = new NoneExpirableProduct("Scratch Card", 60, 10);
Product eggs = new ShippableExpirableProducts("Eggs", 90, 2, 1.2, LocalDate.now().minusDays(3)); // expired
Product ipad = new ShippableNoneExpirableProducts("Ipad", 220, 1, 1.4); // Limited in stock

Customer Customer1 = new Customer("Mohamed", 10000); // customer with enough money
Customer Customer2 = new Customer("Ashraf", 100); // customer with less money
```

Test 1 : Normal Checkout (customer will not face any problems).

```java
// Test 1 : normal
System.out.println("\nTest 1 : Normal Checkout");
Cart cart1 = new Cart();
cart1.add(cheese, 2);
cart1.add(biscuits, 1);
cart1.add(scratchCard, 1);
try {
    Invoice.checkout(Customer1, cart1);
} catch (Exception e) {
    System.out.println("error!: " + e.getMessage());
```

```
** Shipment notice **
Cheese : 0.3kg
Cheese : 0.3kg
Biscuits : 0.8kg
Total weight: 1.4kg
** Checkout receipt **
2x Cheese 240.00
1x Biscuits 180.00
1x Scratch Card 60.00
----------------------
Subtotal: 480.0
Shipping: 30.0
Amount: 510.0
```

Test 2 : Empty Cart

```java
System.out.println("\nTest 2 : Empty Cart");
Cart cart2 = new Cart();
try {
    Invoice.checkout(Customer1, cart2);
} catch (Exception e) {
    System.out.println("error!: " + e.getMessage());
}
```

```
Test 2 : Empty Cart
error!: cart is empty!
```

Test 3 : Expired Product

```java
// Test 3 : Expired Product
System.out.println("Test 3 : Expired Product");
try {
    Cart cart3 = new Cart();
    cart3.add(eggs,2);
    Invoice.checkout(Customer1, cart3);
} catch (Exception e) {
    System.out.println("error!: " + e.getMessage());
}
```

```
Test 3 : Expired Product
error!: Eggs is expired.
```

Test 4 : Out of Stock

```java
// Test 4 : Out of Stock
System.out.println("Test 4 : Out of Stock");
try {
    Cart cart4 = new Cart();
    cart4.add(ipad, 2);
    Invoice.checkout(Customer1, cart4);
} catch (Exception e) {
    System.out.println("error!: " + e.getMessage());
```

```
Test 4 : Out of Stock
error!: Not enough quantity
```

Test 5 : Insufficient Balance

```java
// Test 5 : Insufficient balance
System.out.println("Test 5 : Insufficient Balance");
Cart cart5 = new Cart();
cart5.add(tv, 1);
cart5.add(biscuits, 2);
try {
    Invoice.checkout(Customer2, cart5);
} catch (Exception e) {
    System.out.println("error!: " + e.getMessage());
}
```

```
Test 5 : Insufficient Balance
Insufficient balance.
error!: insufficient balance!!
```

Test 6 : Non shippable

```java
// Test 6 : Non shippable product
System.out.println("Test 6 : Non shippable");
Cart cart6 = new Cart();
cart6.add(scratchCard, 2);
try {
    Invoice.checkout(Customer1, cart6);
} catch (Exception e) {
    System.out.println("error!: " + e.getMessage());
}       System.out.println("\n=======================
```

```
------------------------
Test 6 : Non shippable
** Checkout receipt **
2x Scratch Card 120.00
----------------------
Subtotal: 120.0
Shipping: 0.0
Amount: 120.0
```

Test 7 : User Order => 1 Cheese

```java
// Test 7 : User Input Order
System.out.println("Test 7 : User Order => 1 Cheese");
Scanner scanner = new Scanner(System.in);
System.out.print("Enter your name: ");
String userName = scanner.nextLine();

System.out.print("Enter your balance: ");
double balance = scanner.nextDouble();

Customer userCustomer = new Customer(userName, balance);
Cart userCart = new Cart();
userCart.add(cheese, 1);
try {
    Invoice.checkout(userCustomer, userCart);
} catch (Exception e) {
    System.out.println("error!: " + e.getMessage());
}
```

```
Test 7 : User Order => 1 Cheese
Enter your name: yassin
Enter your balance: 250
** Shipment notice **
Cheese : 0.3kg
Total weight: 0.3kg
** Checkout receipt **
1x Cheese 120.00
----------------------
Subtotal: 120.0
Shipping: 30.0
Amount: 150.0
```

3rd Classes code:

ECommerce_System (Main)

```
package com.mycompany.ecommerce_system;

import java.time.LocalDate;

import java.util.Scanner;


public class ECommerce_System {

  public static void main(String[] args) {

    //TESTING

    Product cheese = new ShippableExpirableProducts("Cheese", 120, 6, 0.3, LocalDate.now().plusDays(2));

    Product biscuits = new ShippableExpirableProducts("Biscuits", 180, 4, 0.8, LocalDate.now().plusDays(1));

    Product tv = new ShippableNoneExpirableProducts("TV", 350, 2, 4.5);

    Product scratchCard = new NoneExpirableProduct("Scratch Card", 60, 10);

    Product eggs = new ShippableExpirableProducts("Eggs", 90, 2, 1.2, LocalDate.now().minusDays(3)); // expired

    Product ipad = new ShippableNoneExpirableProducts("Ipad", 220, 1, 1.4); // Limited in stock


    Customer Customer1 = new Customer("Mohamed", 10000); // customer with enough money

    Customer Customer2 = new Customer("Ashraf", 100); // customer with less money



    // Test 1 : normal

    System.out.println("\nTest 1 : Normal Checkout");

    Cart cart1 = new Cart();

    cart1.add(cheese, 2);
```

```
cart1.add(biscuits, 1);
cart1.add(scratchCard, 1);
try {
  Invoice.checkout(Customer1, cart1);
} catch (Exception e) {
  System.out.println("error!: " + e.getMessage());
}     System.out.println("\n================================================");



// Test 2 : empty cart
System.out.println("\nTest 2 : Empty Cart");
Cart cart2 = new Cart();
try {
  Invoice.checkout(Customer1, cart2);
} catch (Exception e) {
  System.out.println("error!: " + e.getMessage());
}
System.out.println("\n================================================");



// Test 3 : Expired Product
System.out.println("Test 3 : Expired Product");
try {
  Cart cart3 = new Cart();
  cart3.add(eggs,2);
  Invoice.checkout(Customer1, cart3);
} catch (Exception e) {
  System.out.println("error!: " + e.getMessage());
}
System.out.println("\n================================================");



// Test 4 : Out of Stock
```

```
System.out.println("Test 4 : Out of Stock");
try {
   Cart cart4 = new Cart();
   cart4.add(ipad, 2);
   Invoice.checkout(Customer1, cart4);
} catch (Exception e) {
   System.out.println("error!: " + e.getMessage());
}
System.out.println("\n================================================");



// Test 5 : Insufficient balance
System.out.println("Test 5 : Insufficient Balance");
Cart cart5 = new Cart();
cart5.add(tv, 1);
cart5.add(biscuits, 2);
try {
   Invoice.checkout(Customer2, cart5);
} catch (Exception e) {
   System.out.println("error!: " + e.getMessage());
}
System.out.println("\n================================================");



// Test 6 : Non shippable product
System.out.println("Test 6 : Non shippable");
Cart cart6 = new Cart();
cart6.add(scratchCard, 2);
try {
   Invoice.checkout(Customer1, cart6);
} catch (Exception e) {
   System.out.println("error!: " + e.getMessage());
```

```
    }      System.out.println("\n=================================================");



    // Test 7 : User Input Order
    System.out.println("Test 7 : User Order => 1 Cheese");
    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter your name: ");
    String userName = scanner.nextLine();

    System.out.print("Enter your balance: ");
    double balance = scanner.nextDouble();

    Customer userCustomer = new Customer(userName, balance);
    Cart userCart = new Cart();
    userCart.add(cheese, 1);
    try {
      Invoice.checkout(userCustomer, userCart);
    } catch (Exception e) {
      System.out.println("error!: " + e.getMessage());
    }
    System.out.println("\n=================================================");
  }


}
```

Product

```
package com.mycompany.ecommerce_system;

public abstract class Product {
  private String name;
  private double price ;
  private int quantity;
```

```
public Product(String name, double price, int quantity) {
  this.name = name;
  this.price = price;
  this.quantity = quantity;
}

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

public void setPrice(double price) {
  if (price < 0){
    throw new IllegalArgumentException("price can't be negative");
  }
  this.price = price;
}

public void setQuantity(int quantity) {
  if (quantity < 0){
    throw new IllegalArgumentException("quantity can't be negative");
  }
  this.quantity = quantity;
}

public String getName() {
  return name;
}

public double getPrice() {
  return price;
}

public int getQuantity() {
  return quantity;
}
```

```
  public void reduceQuantityProduct(int amount) {
    if (amount > quantity) {
      throw new IllegalArgumentException("The requested quantity is more than the available in stock.");
    }
    quantity -= amount;
  }

  public boolean isAvailable(int amount) {
    return this.quantity >= amount;
  }

  public abstract boolean isExpired();

}
```

ShippingInterface

```
package com.mycompany.ecommerce_system;


public interface ShippingInterface {
  String getName();
  double getWeight();
}
```

ShippingService :

```
package com.mycompany.ecommerce_system; import java.util.ArrayList; import java.util.List;

public class ShippingService { public static void ship(List items) { System.out.println("** Shipment notice **"); double totalWeight = 0;

   for (ShippingInterface item : items) {
        System.out.println(item.getName() +" : " + item.getWeight()+ "kg");
        totalWeight += item.getWeight();
    }

    System.out.println("Total weight: " + totalWeight + "kg");
```

```
}

}
```

```
package com.mycompany.ecommerce_system;

public abstract class ShippableProduct extends Product implements ShippingInterface {
   private double weight;

   public ShippableProduct(String name, double price, int quantity, double weight) {
      super(name, price, quantity);
      this.weight = weight;
   }

   @Override
   public double getWeight() {
      return weight;
   }
}
```

```
package com.mycompany.ecommerce_system;

public class ShippableNoneExpirableProducts extends ShippableProduct {

   public ShippableNoneExpirableProducts(String name, double price, int quantity, double weight) {
      super(name, price, quantity, weight);
   }
   @Override
   public boolean isExpired() {
      return false;
```

```
    }

}
```

```
package com.mycompany.ecommerce_system;

import java.time.LocalDate;

public class ShippableExpirableProducts extends ShippableProduct {

    private LocalDate expireDate;

    public ShippableExpirableProducts(String name, double price, int quantity, double weight, LocalDate
expireDate) {
    super(name, price, quantity, weight);
    this.expireDate = expireDate;
    }



    @Override
    public boolean isExpired() {
        return LocalDate.now().isAfter(expireDate);
    }
}
```

```
package com.mycompany.ecommerce_system;

public class NoneExpirableProduct extends Product {
    public NoneExpirableProduct(String name, double price, int quantity) {
        super(name, price, quantity);
    }
```

```
    @Override
    public boolean isExpired(){
        return false;
    }
}
```

```
package com.mycompany.ecommerce_system;

public class Item {

    private Product product;
    private int quantity;

    public Item(Product product, int quantity) {
        this.product = product;
        this.quantity = quantity;
    }

    public Product getProduct() {
        return product;
    }

    public int getQuantity() {
        return quantity;
    }

    public double ProducTotaltPrice() {
    return product.getPrice()*quantity;
    }
```

}

---

package com.mycompany.ecommerce_system;

import java.util.ArrayList; import java.util.List;

public class Invoice { private static double ShippingPrice = 30; // constant shipping price

```
public static void checkout(Customer customer, Cart cart) {
    if (cart.isEmpty()) {
        throw new IllegalStateException("cart is empty!");
    }

    List<ShippingInterface> shippables = new ArrayList<>();
    double shippingWeight = 0;

    for (Item item : cart.getItems()) {
        Product product = item.getProduct();
        int quantity = item.getQuantity();

        if (product.isExpired()) {
            throw new IllegalStateException(product.getName() + " is expired.");
        }

        if (!product.isAvailable(quantity)) {
            throw new IllegalStateException(product.getName() + " is out of stock.");
        }

        if (product instanceof ShippingInterface) {
            for (int i = 0; i < quantity; i++) {
                shippables.add((ShippingInterface) product);
                shippingWeight += ((ShippingInterface) product).getWeight();
            }
        }
    }

    double subtotal = cart.getSubtotal();
    double shippingFee;
    if (shippingWeight > 0) {
        shippingFee = ShippingPrice;}
    else {
        shippingFee = 0;
```

} double total = subtotal + shippingFee;

```
   if (!customer.checkBalance(total)) {
        throw new IllegalStateException("insufficient balance!!");
    }

    for (Item item : cart.getItems()) {
        item.getProduct().reduceQuantityProduct(item.getQuantity());
    }

    if (!shippables.isEmpty()) {
        ShippingService.ship(shippables);
    }

    System.out.println("** Checkout receipt **");
    for (Item item:cart.getItems()) {
        System.out.printf("%dx %s %.2f\n", item.getQuantity(),
item.getProduct().getName(), item.ProducTotaltPrice());
    }
    System.out.println("----------------------");
    System.out.println("Subtotal: " + subtotal);
    System.out.println("Shipping: " + shippingFee);
    System.out.println("Amount: " + total);

}


}
```

```
package com.mycompany.ecommerce_system;

import java.time.LocalDate;


public class ExpirableProduct extends Product{

  private LocalDate expireDate;


  public ExpirableProduct(String name, double price, int quantity, LocalDate expiryDate) {

    super(name, price, quantity);

    this.expireDate = expireDate;

  }


  public void setExpireDate(LocalDate expireDate) {

    this.expireDate = expireDate;
```

```
    }

    public LocalDate getExpireDate() {
        return expireDate;
    }

    @Override
    public boolean isExpired(){
        return LocalDate.now().isAfter(expireDate);
    }

}
```

```
package com.mycompany.ecommerce_system;

public class Customer {
    private String name;
    private double balance;

    public Customer(String name, double balance) {
        this.name = name;
        this.balance = balance;
    }

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

    public void setBalance(double balance) {
        this.balance = balance;
```

```
    }

    public String getName() {
        return name;
    }

    public double getBalance() {
        return balance;
    }



    public void deductBalance(double amount) {
        if (amount > balance) {
            throw new IllegalStateException("Insufficient balance for " + name);
        }
        balance -= amount;
    }

        public boolean checkBalance(double amount) {
        if (balance >= amount) {
            balance -= amount;
            return true;
        }
        System.out.println("Insufficient balance.");
        return false;
    }
}
```

```
package com.mycompany.ecommerce_system;
import java.util.ArrayList;
```

```
import java.util.List;


public class Cart {
  private List<Item> items = new ArrayList<>();


  public void add(Product product, int quantity) {
    if (!product.isAvailable(quantity)) {
      throw new IllegalArgumentException("Not enough quantity");
    }
    items.add(new Item(product, quantity));
  }


  public List<Item> getItems() {
    return items;
  }


  public boolean isEmpty() {
    return items.isEmpty();
  }


  public double getSubtotal() {
    return items.stream().mapToDouble(Item::ProducTotaltPrice).sum();
  }
}
```