CSE 102 Programming Assignment 3

Due:

Thursday, 1-June-2023 by 23:59

Deliverables:

The following Java file should be submitted to MS Teams by the due date and time specified above. Submissions received after the deadline will be subject to the late policy described in the syllabus.

- Assignment03_{StudentNumber}.java

Specifications:

Overview: You will continue the program this semester to maintain the inventory for a store. Do not forget your headers with @author and @since information.

Requirements: Write and modify the following set of classes (<u>**All constructors should take parameters in the order given and initialize empty collections – This may mean you will need to change or add constructors from your Assignment 2**</u>):

1. Product
    a. Attributes – remove the quantity parameter (this will be moved to the Store)
    b. Methods – all methods from Assignment 02 with following modifications
        i. Remove purchase(), addToInventory(), remaining()
            1. These will be moved to the Store class
        ii. toString(): String
            1. returns {ID} - {name} @ {price}
2. FoodProduct and CleaningProduct – no change
3. Customer
    a. Attributes – no change (hint: Collections will make this class easier to implement)
    b. Methods
        i. addToCart(store: Store, product: Product, count: int): None
            1. Adds the passed Product and number to the customer cart for the passed Store
            2. If the purchase is not successful (Product is not in the Store or not enough available), this method displays a message to the screen beginning with "ERROR: "
        ii. receipt(store: Store): String
            1. returns a header including the Store name then each Product in the cart on a separate line in the format below
               {Product ID } - {Product Name} @ {Product Price} X {count} … {total for Product}
               Total Due – {Total amount}
            2. If the customer does not have a cart for the passed Store, raises a StoreNotFoundException
        iii. getTotalDue(store: Store): double – returns the total amount due for the passed Store

1. If the customer does not have a cart for the passed Store, raises a StoreNotFoundException

iv. getPoints(store: Store): int – returns the total customer points for the passed Store
   1. If the customer is not in the Customer collection for the passed Store, raises a StoreNotFoundException

v. pay(store: Store , amount: double, boolean: usePoints): double
   1. If amount is greater than or equal to total due, displays a "Thank you" message to the screen and returns the amount that should be given as change
   2. If this customer is a club customer, points should be added as was done in Assignment 2
   3. If usePoints is true, performs the same task as the pay method from ClubCustomer from Assignment 2 as long as the Customer object is in the Customer collection for the passed Store
   4. If usePoints is true and the Customer is not in the Customer collection for the Store, use 0 for the points value
   5. If amount is less than total due, raises a InsufficientFundsException
   6. If the customer does not have a cart for the passed Store, raises a StoreNotFoundException

4. ClubCustomer – remove this class since the Store will now have a collection of customers
5. Store
   a. Attributes – remove the ClubCustomer collection. You also will likely need to modify other collections.
   b. Methods
      i. getCount(): int
         1. returns the number of remaining Product objects in inventory
      ii. remove addProduct() and getProduct()
      iii. addCustomer(customer: Customer): None
         1. Adds the Customer to the Collection
         2. Sets their points to 0
      iv. getProductCount(product: Product): int
         1. returns the count of this Product in the store inventory
         2. if the Product is not found in the collection, raises a ProductNotFoundException
      v. getCustomerPoints(customer: Customer): int
         1. returns the points for the customer passed
         2. if the customer is not found in the collection, raises a CustomerNotFoundException
      vi. removeProduct(product: Product): None

1. removes the passed Product from the Collection
2. if the Product is not found in the Collection, raises a ProductNotFoundException

vii. addToInventory(product: Product, amount: int): None

1. If product is not in the Collection, adds it using the amount
2. If product already exists, adds amount to the inventory
3. raises InvalidAmountException if amount is negative

viii. purchase(product: Product, amount: int): double

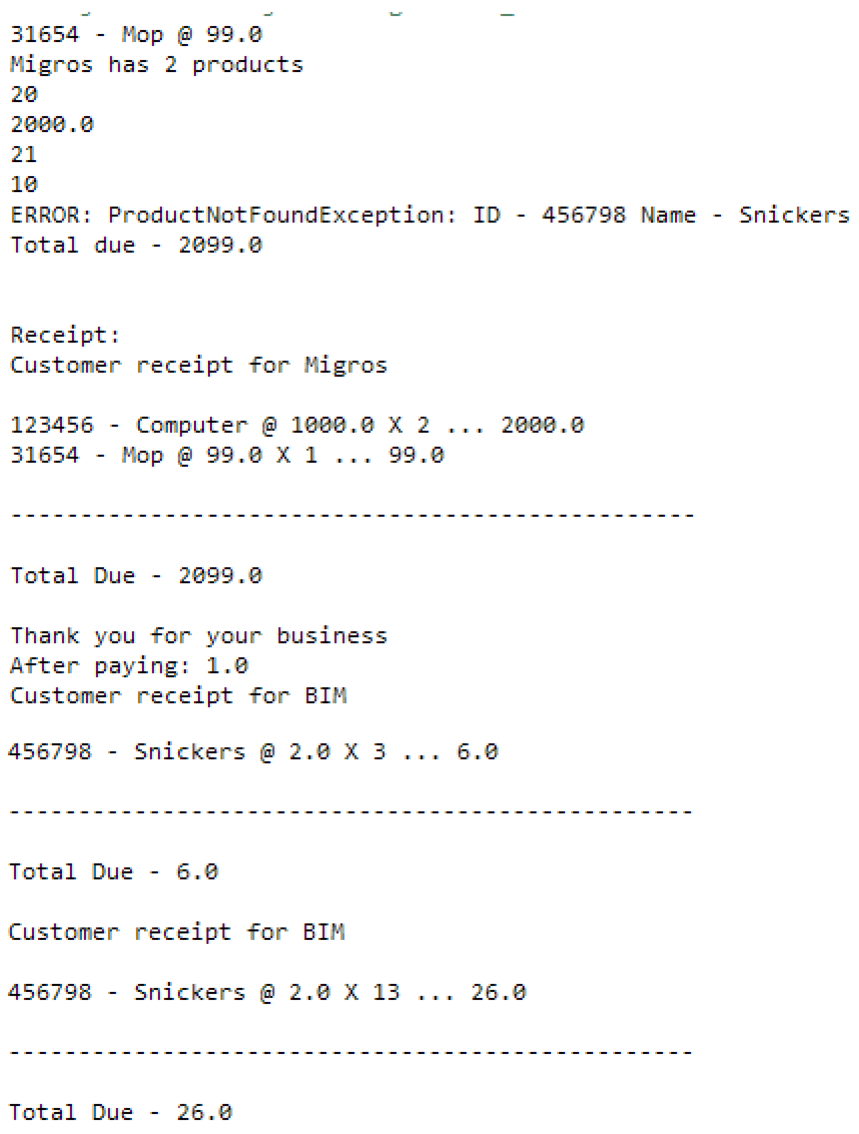
1. reduces the count in the inventory and returns the total price based on (amount X price)
2. if amount is negative or greater than count, do not change count and raise InvalidAmountException
3. if the Product is not found in the Collection, do not change count and raises a ProductNotFoundException

6. Custom Exceptions (All must be instances of RunTimeException) – all from Assignment 2 plus the one given below and the one change below:
    a. StoreNotFoundException – instance of <u>**IllegalArgumentException**</u>
        i. Additional Attribute – name: String
        ii. toString(): “StoreNotFoundException: “ + name
    b. CustomerNotFoundException – instance of <u>**IllegalArgumentException**</u>
        i. Additional Attribute – customer: Customer
        ii. toString(): “CustomerNotFoundException: Name - “ + name
    c. ProductNotFoundException – instance of <u>**IllegalArgumentException**</u>
        i. Additional Attribute – product: Product
        ii. toString(): “ProductNotFoundException: ID - “ + ID + “ Name - “ + name

Design: Your program does not require a main method. You are only responsible for modifying or creating the five (5) classes and six (6) Exceptions described above.
Code: The file you submit will be named Assignment03_{StudentNumber}. You should put all java classes for this assignment inside of this file as discussed in class.
Test: You are responsible for testing your program. It is important to not rely solely on the examples presented in this Assignment description. It would be a very good idea to write your own test cases for this assignment.

Grading:
MS Teams Submission: If anything is ambiguous, it is your responsibility to ask questions. It is also your responsibility to complete this assignment in a timely manner. Questions regarding this assignment will likely not be answered if received after 17:00 on the last weekday before the due date of the assignment.

```java
public class Assignment03_123456789 {
   public static void main(String[] args) {
      Store s1 = new Store("Migros", "www.migros.com.tr");
      Store s2 = new Store("BIM", "www.bim.com.tr");

      Customer c = new Customer("CSE 102");

      Customer cc = new Customer("Club CSE 102");
      s1.addCustomer(cc);

      Product p = new Product(123456L, "Computer", 1000.00);
      FoodProduct fp = new FoodProduct(456798L, "Snickers", 2, 250, true, true, true, false);
      CleaningProduct cp = new CleaningProduct(31654L, "Mop", 99, false, "Multi-room");
      System.out.println(cp);

      s1.addToInventory(p, 20);
      s2.addToInventory(p, 10);
      s2.addToInventory(fp, 100);
      s1.addToInventory(cp, 28);

      System.out.println(s1.getName() + " has " + s1.getCount() + " products");
      System.out.println(s1.getProductCount(p));

      System.out.println(s1.purchase(p, 2));
      s1.addToInventory(p, 3);
      System.out.println(s1.getProductCount(p));
      System.out.println(s2.getProductCount(p));
//      System.out.println(s1.getProductCount(fp)); //results in Exception
//      System.out.println(s2.purchase(fp, 200));  // results in Exception

      c.addToCart(s1, p, 2);
      c.addToCart(s1, fp, 1); // NOTE: This does not stop the program because the Exception is caught
      c.addToCart(s1, cp, 1);
      System.out.println("Total due - " + c.getTotalDue(s1));
      System.out.println("\n\nReceipt:\n" + c.receipt(s1));
//      System.out.println("\n\nReceipt:\n" + c.receipt(s2)); // results in Exception

//      System.out.println("After paying: " + c.pay(s1, 2000, true)); // results in Exception

      System.out.println("After paying: " + c.pay(s1, 2100, true));

//      System.out.println("Total due - " + c.getTotalDue(s1)); // results in Exception
//      System.out.println("\n\nReceipt:\n" + c.receipt(s1)); // results in Exception

      cc.addToCart(s2, fp, 2);
      cc.addToCart(s2, fp, 1);
      System.out.println(cc.receipt(s2));

      cc.addToCart(s2, fp, 10);
      System.out.println(cc.receipt(s2));
   }
}
```



```
31654 - Mop @ 99.0
Migros has 2 products
20
2000.0
21
10
ERROR: ProductNotFoundException: ID - 456798 Name - Snickers
Total due - 2099.0


Receipt:
Customer receipt for Migros

123456 - Computer @ 1000.0 X 2 ... 2000.0
31654 - Mop @ 99.0 X 1 ... 99.0

-------------------------------------------------

Total Due - 2099.0

Thank you for your business
After paying: 1.0
Customer receipt for BIM

456798 - Snickers @ 2.0 X 3 ... 6.0

-------------------------------------------------

Total Due - 6.0

Customer receipt for BIM

456798 - Snickers @ 2.0 X 13 ... 26.0

-------------------------------------------------

Total Due - 26.0
```