# UNIVERSITI TEKNOLOGI MALAYSIA

# PROBLEM SOLVING 2 (2.5%)

# SEMESTER II 2020/2021

**CODE OF SUBJECT** : **SCSJ4383**

**NAME OF SUBJECT** :**SOFTWARE CONSTRUCTION**

**YEAR / COURSE** : **4/SCSJ**

---

**GROUP: Systematic Squad**

**TEAM MEMBERS:**

| Name | Roles |
|---|---|
| Ibrahim Katari (A17CS4018) | Software Engineer |
| Moaid Mohamed Abdelmoneim (A17CS4025) | Senior developer |
| Yasser Ehab Mohieldin (A17CS3013) | Project Manager |
| Dip Jyoti Ghosh (A17CS4001) | Software Tester |
| Shahriar Hasan Sajid (A16CS4022) | Requirements Engineer |

## Design by contract planning

The system chosen for development from the view of design by contract approach is the Shopping cart program using Java. Details about the program is included in APPENDIX I.

## i. Identify 3 FUNCTIONS or more and describe in detail the PRE, PROCESS and POST condition for each function:

1. **addProduct:**

   **class name:** Customer.

   **precondition:** The product exists and the customer has more balance than the product's price

   **process:** The customer adds the product to cart.

   **Postcondition :** The product is added to the cart, quantity and account balance are changed accordingly.

2. **removeProduct:**

   **class name:** Seller

   **precondition:** The specified product exists in the seller's products list.

   **process:** The seller removes that specific product from their list.

   **postcondition**: Product is removed from the seller's products list.

3. **getTotalItemsNo:**

   **class name:** Seller

   **precondition:** seller products list has products and not empty(null).

   **process:** system calculates the number of products in seller products list.

   **postcondition:** products count is returned based on the seller's products list.

4. **calcPrice:**

   **class name:** Order

   **precondition:** Order has products added.

   **process:** The function calculates the total price of the order.

   **postcondition:** Total order price is calculated and returned.

## ii. Apply ASSERT and INVARIANT in the code and shows the output

The system applies the concept of assertion and invariants in three classes: Product, Customer, and User.

### Class Invariants

**Product:** assert and invariant are applied to the Product class to ensure that product's cost can never be less than 0 no matter what other code does. These concepts are also used to ensure that the product quantity can never be a negative number. Figure 1 shows the application of the concept in the code.

```
public class Product {
    private int productId;
    private String productName;
    private int quantity;
    private double productCost;

    public Product(int productId,String productName,double productCost)
    {
        // INVARIANT
        assert this.quantity >= 0 :"Quantity can't be negative";
        assert this.productCost >0 :"Product cost must be more than 0!";

        this.productId=productId;
        this.productName=productName;
        this.productCost=productCost;

    }
```

Figure 1: Product Class Constructor invariant

**Customer:** Assert and invariant are applied in the Customer class to ensure that customer`s account balance is not less than 0 regardless of any other transactions or functionalities in other code. Figure 2 shows the application of these concepts.

```
public Customer(){
    // INVARIANT
    assert this.accountBalance >= 0 : "Account Balance Can not be negative";
}
```

Figure 2: Customer Class Constructor invariant

**User:** Invariant is applied in that class to ensure the user name can never be null. Figure 3 shows the application of the invariant in User class.

```
public User(){
    accountType=null;
    // INVARIANT
    this.name = " ";
}
```

Figure 3: User Class Constructor invariant

## Function Assertions

**addProduct:** The function uses assert to assure that the precondition of the customer having account balance more than the product's price is met. figure 4 shows the application of assert in this function.

```
@Override
public void addProduct(Product p) {
    assert accountBalance>p.getProductCost() : "no balance";
    if(accountBalance<p.getProductCost() && S.myProducts.contains(p)){
        System.out.println("You (" + name + ") Have insufficient balance to add product: "+p.getProductName());
    }

    if (p.getQuantity()== 0){
        System.out.println("Product: "+p.getProductName() + " is sold out.");
    }

    if (!S.myProducts.contains(p)){
        System.out.println("Product: "+p.getProductName()+" is not Sold by " +
                S.getName());
    }
    if((accountBalance>=p.getProductCost()) && (p.getQuantity()>0) && S.myProducts.contains(p)){
        accountBalance-=p.getProductCost();
        p.setQuantity(p.getQuantity()-1);
        cart.add(p);
    }
}
```

Figure 4: Customer Add product assert

**removeProduct:** The function uses assert to assure that the precondition of the seller having the product in their list before removing it is met. figure 5 shows the application of assert in this function.

```
public void  removeProduct(Product p){
    assert myProducts.contains(p) : "The product doesn't exist in your list";
    myProducts.remove(p);
}
```

Figure 5: Seller remove product assert

**getTotalItemsNo:** The function uses assert to assure that the precondition of the seller having a product list before calculating the number of the products is met. figure 6 shows the application of assert in this function.

```
@Override
public void getTotalItemsNo() {
    assert  myProducts != null : "There is no products list";
    productsNum=myProducts.size();
    System.out.println("Number of products: " + productsNum);

}
```

Figure 6: Seller get total items number assert

**calcPrice:** The function uses assert to assure that the precondition of the order having at least one product in the cart before calculating the total price is met. figure 7 shows the application of assert in this function.

```
public double calcPrice(){
    assert orderItems.size()>0:"There are no items added to the order";
    orderPrice=0;
    for (int i=0;i<orderItems.size();i++){
        orderPrice+=orderItems.elementAt(i).getProductCost();
    }
    return orderPrice;
}
```

Figure 7: Order calculate total price assert

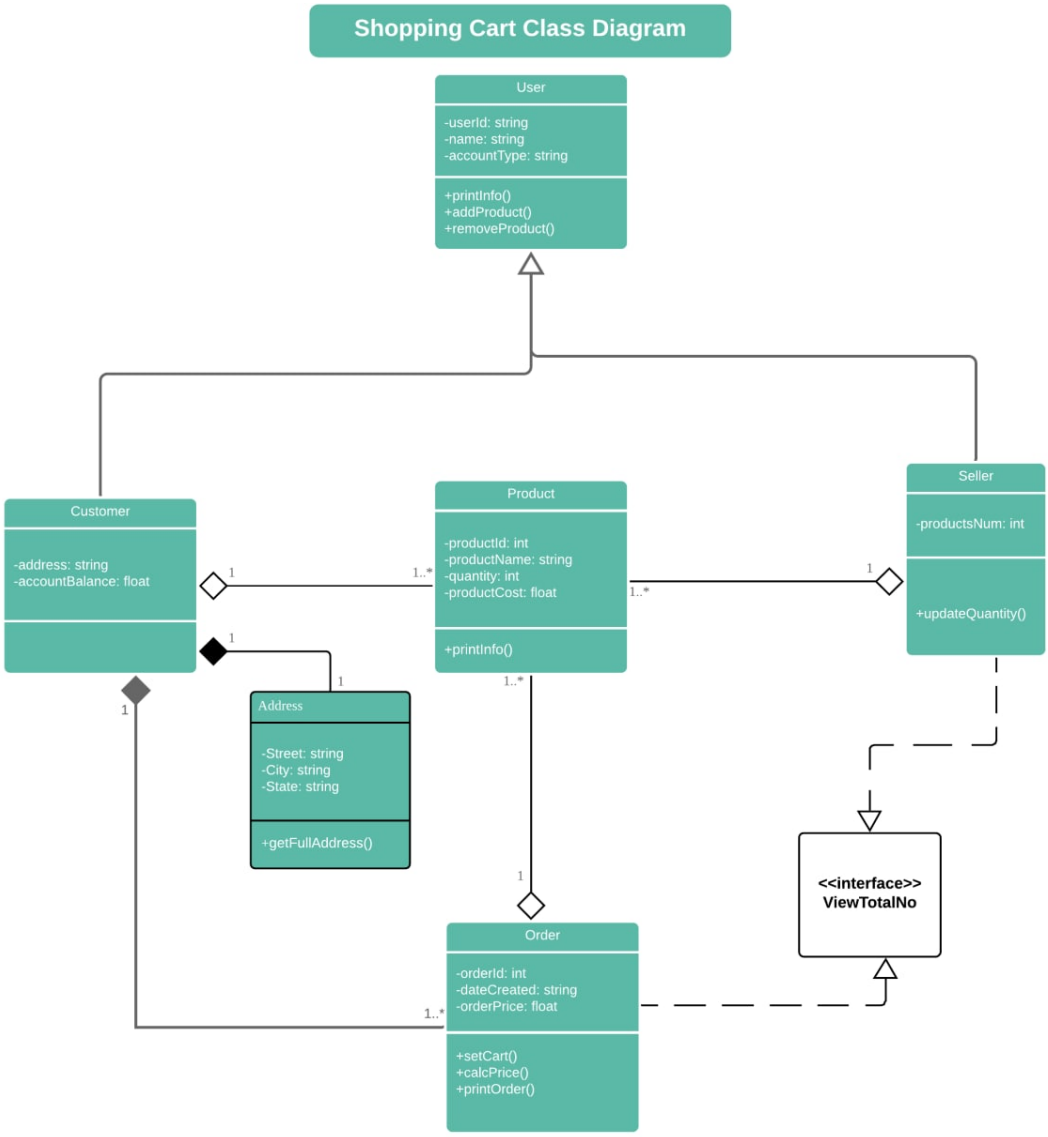
## iii. Git Repository:

**https://bit.ly/3teyvoZ**

# APPENDIX I

## Java Shopping Cart Documentation:

## Class diagram:



The system contains two types of Users: Customer and Seller.
This program has the following functionalities:

- Add, remove products in orders in the online shopping by the customer.
- Add, remove and update quantity by the seller in the system.
- Managing the information of the users (sellers and customers)
- Managing the details of the orders and its products like (Date created, order ID and order price).
- Managing the information of the products such as (product name, product ID, quantity and product cost).
- Printing out all the details of customers, sellers, and orders.

Polymorphism will occur between the parent abstract class (user) and its subclasses (customer) and (seller) specifically in the overridden methods addProduct(), removeProduct(), and printInfo().

**Classes and methods:**

**Class 1: User**
**Data members:** userId, name and account type.
**Methods:** constructors, mutators, printinfo, addproduct and removeproduct.

**Class 2: Address**
**Data members:** street, city, state.
**Methods:** constructors and mutators.

**Class 3: Customer**
Data members: address, accountbalance.
Methods: constructors, accessors, mutators, addProduct, and removeProduct, checkProduct(version 1 only)

**Class4: Seller**
Data members: productsnum.
Methods: constructors, mutators, addProduct , removeProduct, updatequantity and printinfo.

**Class 5: Order**

Data members: Datecreated, orderId and orderPrice.

Methods: constructors, accessors, mutators, calcPrice, and printOrder.

**Class 6: Product**

Data members: product name, product ID, quantity and product cost.

Methods: constructors,accessors,mutators and printinfo.

**Interface: ViewTotalNo**

Abstract Method: getTotalItemsNo