

Question 1:

1.

Defect ID: DF01

Defect Name: Naming convention

Line of Code: 3

Defect Description: variable was initialize with Pascal case instead of camel case

Fixing Solution: change "FilePath" into "filePath"

2.

Defect ID: DF02

Defect Name: Used function

Line of Code: 26

Defect Description: Method 'processFile()' is never used

Fixing Solution: Delete or comment unused function

3.

Defect ID: DF03

Defect Name: Cannot resolve symbol 'BufferedReader'

Line of Code: 6

Defect Description: Missing BufferedReader library

Fixing Solution: Add 'import java.io.BufferedReader;' at the beginning of file

4.

Defect ID: DF04

Defect Name: Cannot resolve symbol 'FileReader'

Line of Code: 6

Defect Description: Missing 'FileReader' library

Fixing Solution: Add 'import java.io.FileReader;' at the beginning of file

5.

Defect ID: DF05

Defect Name: Unhandled exception

Line of Code: 23

Defect Description: Unhandled exception: java.io.IOException

Fixing Solution: Throws java.io.IOException

6.

Defect ID: DF06

Defect Name: Private field 'FilePath' is never assigned

Line of Code: 3

Defect Description: Missing value for FilePath field

Fixing Solution: Assign value to FilePath field

Question 2:

ID: TC1; Test for invalid input; Input parameters: double[]{},"VIP", false, null; Expected result: IllegalArgumentException

Test code:

```
@Test(expected = IllegalArgumentException.class)
public void testNoltemsInOrder(){
    OrderCalculators calculator = new OrderCalculators();
    calculator.calculateTotalPrice(new double[]{},"VIP",false,null);
}
```

ID: TC2; Test for invalid input; Input parameters: double[]{100,0,200}, "VIP", false, null; Expected result: IllegalArgumentException

Test code:

```
@Test(expected = IllegalArgumentException.class)
public void testItemsPriceLowerThanZero(){
    OrderCalculators calculator = new OrderCalculators();
    calculator.calculateTotalPrice(new double[]{100,0,200}, "VIP",false,null);
}
```

ID: TC3; Test for branch 1; Input parameters: double[]{100, 200}, "VIP" ,true ,null ; Expected result: 240.0

Test code:

```
@Test
public void testVipCustomerWithoutDiscount(){
    OrderCalculators calculator = new OrderCalculators();
    double total = calculator.calculateTotalPrice(new double[]{100, 200}, "VIP",true,null);
    assertEquals(240.0,total, 0.01);
}
```

ID: TC4; Test for branch equivalence of discount code; Input parameters: double[]{100, 200}, "VIP", true, "SALE10" ; Expected result: 210.0

Test code:

```
@Test
public void testVipCustomerWithTenPercentDiscount(){
    OrderCalculators calculator = new OrderCalculators();
    double total = calculator.calculateTotalPrice(new double[]{100, 200}, "VIP",true,"SALE10");
    assertEquals(210.0, total, 0.01);
}
```

ID: TC5; Test fo equivalence of discount code; Input parameters: double[]{100, 200}, "VIP", true, "WELCOME5" ; Expected result: 225.0

Test code:

```
@Test
public void testVipCustomerWithFivePercentDiscount(){
    OrderCalculators calculator = new OrderCalculators();
    double total = calculator.calculateTotalPrice(new double[]{100, 200}, "VIP",true,"WELCOME5");
    assertEquals(225.0, total, 0.01);
}
```

ID: TC6; Test for equivalence of role; Input parameters: double[]{100, 200}, "Regular",false,null ; Expected result: 285.0

Test code:

```
@Test
public void testRegularCustomerWithoutDiscount(){
    OrderCalculators calculator = new OrderCalculators();
    double total = calculator.calculateTotalPrice(new double[]{100, 200}, "Regular",false,null);
    assertEquals(285.0,total, 0.01);
}
```

ID: TC7; Test for equivalence of role; Input parameters: double[]{100, 200}, "Regular", false, "SALE10" ;
Expected result: 255.0

Test code:

```
@Test  
public void testRegularCustomerWithTenPercentDiscount(){  
    OrderCalculators calculator = new OrderCalculators();  
    double total = calculator.calculateTotalPrice(new double[]{100, 200}, "Regular", false, "SALE10");  
    assertEquals(255.0, total, 0.01);  
}
```

ID: TC8; Test for equivalence of role; Input parameters: double[]{100, 200}, "Regular", false, "WELCOME5" ; Expected result: 270.0

Test code:

```
@Test  
public void testRegularCustomerWithFivePercentDiscount(){  
    OrderCalculators calculator = new OrderCalculators();  
    double total = calculator.calculateTotalPrice(new double[]{100, 200}, "Regular", false, "WELCOME5");  
    assertEquals(270.0, total, 0.01);  
}
```

Question 3:

Test Case 1 - Normal Flow - successfully order with valid discount code

Test Case ID: TC01

Description: Verify successful order placement with valid discount code.

Preconditions:

- Customer is logged in.
- Customer has at least one item in the cart.

Test Steps:

1. Add items to the cart.
2. Apply a valid discount code.
3. Proceed to payment.
4. Complete payment with a valid method.

Expected Results:

- Discount is successfully applied.
- Payment is processed successfully.
- Customer receives order confirmation.
- Inventory is updated.

Notes: NF (Normal Flow)

Test Case 2 - Alternate Flow - Missing item in cart

Test Case ID: TC02

Description: Prompt user to add item to cart before process payment if their cart is empty

Preconditions:

- Customer is logged in.
- Customer's cart is empty

Test Steps:

1. Attempt to place an order with an empty cart.

Expected Results:

- System displays a message prompting the customer to add items to the cart before proceeding with the order.

Notes: AF (Alternative Flow)

Test Case 3 - Alternate Flow - successfully order with invalid discount code

Test Case ID: TC03

Description: Verify successful order placement with invalid discount code.

Preconditions:

- Customer is logged in.
- Customer has at least one item in the cart.

Test Steps:

1. Add items to the cart.
2. Apply a invalid discount code.
3. Proceed to payment.
4. Complete payment with a valid method.

Expected Results:

- System displays a message prompting the customer that their discount code is invalid
- Payment is processed successfully.
- Customer receives order confirmation.
- Inventory is updated.

Notes: AF (Alternate Flow)

Test Case 4 - Alternate Flow - System testing

Test Case ID: TC04

Description: Verify system to prompt user if their payment process is failed

Preconditions:

- Customer is logged in.
- Customer has at least one item in the cart.

Test Steps:

1. Add items to the cart.
2. Apply a valid discount code.
3. Proceed to payment.
4. Payment didn't process successfully

Expected Results:

- System displays a message prompting the customer to retry their payment process due to the failure of previous payment process

Notes: AF (Alternate Flow)

