**Answer Question 1:**

**1:**

* **Defect ID:** DF001
* **Defect Name:** Wrong Naming Convention.
* **Line Number:** 2
* **Defect Description:** The name fileProcessor for the class is invalid, which require class names to use PascalCase (each word starting with a capital letter).
* **Fixing Solution:** Modify the class name to FileProcessor to suitable with Java’s name conventions.

**2:**

* **Defect ID: DF002**
* **Defect Name:** Resource Leak in openFile.
* **Line Number:** 13-14
* **Defect Description:** BufferedReader is created in the openFile method but is not closed, potentially causing a resource leak if an exception occurs.
* **Fixing Solution:** Use a try-catch statement or ensure reader.close() is called in a finally block.

**3:**

* **Defect ID:** DF003
* **Defect Name:** Inefficient Condition Check.
* **Line Number:** 9
* **Defect Description:** The condition filePath != null || filePath.isEmpty() is incorrect; it should be filePath == null || filePath.isEmpty() to check for both null and empty conditions.
* **Fixing Solution:** Change the condition to filePath == null || filePath.isEmpty().

**4:**

* **Defect ID:** DF004
* **Defect Name:** Lack of Null Check for reader in readFile.
* **Line Number:** 16
* **Defect Description:** In the readFile method, there is no check for reader being null before use, which could lead to a Exception if openFile fails.
* **Fixing Solution:** Add a null check if (reader == null) at the beginning of the readFile method and handle the case where reader is not initialized.

**5:**

* **Defect ID:** DF005
* **Defect Name:** Hardcoded Error Messages.
* **Line Number:** 11, 14, 19, 22, 31
* **Defect Description:** Error messages are hardcoded strings, making it difficult to change or internationalize.
* **Fixing Solution:** Use constants or an external resource file to store error messages.

**6:**

* **Defect ID:** DF006
* **Defect Name:** Unnecessary Method Call in processFile.
* **Line Number:** 33
* **Defect Description:** The processFile method calls readFile without checking if reader has been successfully initialized.
* **Fixing Solution:** Check the state of reader or add a flag to ensure openFile has completed before calling readFile.

**Answer Question 2:**

**ID:** TC1; **Test for** no items in order; **Input parameter**: itemPrices = [], customerType = "VIP", isVIP = false, discountCode = null; **Expected result**: Exception

@Test(expected = IllegalArgumentException.class)

public void testNoItemsInOrder() {

OrderCalculator cal = new OrderCalculator();

cal.calculateTotalPrice(new double[]{}, "VIP", false, null);

}

**ID:** TC2; **Test for** invalid item price; **Input parameter**: itemPrices = [-10.0, 20.0], customerType = "VIP", isVIP = false, discountCode = null; **Expected result**: Exception

@Test(expected = IllegalArgumentException.class)

public void testInvalidItemPrice() {

OrderCalculator cal = new OrderCalculator();

cal.calculateTotalPrice(new double[]{-10.0, 20.0}, "VIP", false, null);

}

**ID:** TC3; **Test for** VIP customer with no discount code; **Input parameter**: itemPrices = [100, 200], customerType = "VIP", isVIP = true, discountCode = null; **Expected result**: 240.0

@Test

public void testVIPCustomerWithNoDiscountCode() {

OrderCalculator cal = new OrderCalculator();

double total = cal.calculateTotalPrice(new double[]{100, 200}, "VIP", true, null);

assertEquals(240.0, total, 0.01); // 20% discount

}

**ID:** TC4; **Test for** regular customer with no discount code; **Input parameter** itemPrices = [100, 200], customerType = "Regular", isVIP = false, discountCode = null; **Expected result**: 300.0

@Test

public void testRegularCustomerWithNoDiscountCode() {

OrderCalculator cal = new OrderCalculator();

double total = cal.calculateTotalPrice(new double[]{100, 200}, "Regular", false, null);

assertEquals(300.0, total, 0.01);

}

**ID:** TC5; **Test for** regular customer with SALE10 discount code; **Input parameter** itemPrices = [100, 200], customerType = "Regular", isVIP = false, discountCode = "SALE10"; **Expected result**: 270.0

@Test

public void testRegularCustomerWithSALE10DiscountCode() {

OrderCalculator cal = new OrderCalculator();

double total = cal.calculateTotalPrice(new double[]{100, 200}, "Regular", false, "SALE10");

assertEquals(270.0, total, 0.01);

}

**ID:** TC6; **Test for** VIP customer with SALE10 discount code; **Input parameter** itemPrices = [100, 200], customerType = "VIP", isVIP = true, discountCode = "SALE10"; **Expected result**: 216.0

@Test

public void testVIPCustomerWithSALE10DiscountCode() {

OrderCalculator cal = new OrderCalculator();

double total = cal.calculateTotalPrice(new double[]{100, 200}, "VIP", true, "SALE10");

assertEquals(216.0, total, 0.01);

}

**ID:** TC7; **Test for** regular customer with WELCOME5 discount code; **Input parameter** itemPrices = [100, 200], customerType = "Regular", isVIP = false, discountCode = "WELCOME5";**Expected result**: 285.0

@Test

public void testRegularCustomerWithWELCOME5DiscountCode() {

OrderCalculator cal = new OrderCalculator();

double total = cal.calculateTotalPrice(new double[]{100, 200}, "Regular", false, "WELCOME5");

assertEquals(285.0, total, 0.01);

}

**ID:** TC8; **Test for** VIP customer with WELCOME5 discount code; **Input parameter** itemPrices = [100, 200], customerType = "VIP", isVIP = true, discountCode = "WELCOME5";**Expected result**: 228.0

@Test

public void testVIPCustomerWithWELCOME5DiscountCode() {

OrderCalculator cal = new OrderCalculator();

double total = cal.calculateTotalPrice(new double[]{100, 200}, "VIP", true, "WELCOME5");

assertEquals(228.0, total, 0.01);

}

**ID:** TC9; **Test for** minimum valid total; **Input parameter** itemPrices = [0.01], customerType = "Regular", isVIP = false, discountCode = null;**Expected result**: 0.01

@Test

public void testMinimumValidTotal() {

OrderCalculator cal = new OrderCalculator();

double total = cal.calculateTotalPrice(new double[]{0.01}, "Regular", false, null);

assertEquals(0.01, total, 0.01);

}

**ID:** TC9; **Test for** largest order total; **Input parameter** itemPrices = [1000.0, 2000.0, 3000.0], customerType = "VIP", isVIP = true, discountCode = "SALE10";**Expected result**: 5040.0

@Test

public void testLargestOrderTotal() {

OrderCalculator cal = new OrderCalculator();

double total = cal.calculateTotalPrice(new double[]{1000.0, 2000.0, 3000.0}, "VIP", true, "SALE10");

assertEquals(5040.0, total, 0.01);

}

**Answer Question 3:**

### Test Case 1: Successful Order with WELCOME5 Discount Code

* **ID**: TC001
* **Description**: Tests a successful order when a new customer uses the WELCOME5 discount code.
* **Preconditions**:

The customer is logged in, and the cart has at least one item.

* **Test Steps**:
  1. Customer adds Product B ($30) and Product C ($20) to the cart.
  2. Customer applies the discount code WELCOME5.
  3. System checks and accepts the discount code.
  4. Customer goes to checkout.
  5. System shows the total with a 5% discount applied.
  6. Customer provides valid credit card details.
  7. System processes the payment.
  8. System confirms the order and sends a confirmation.
* **Expected Result**:

Customer receives order confirmation with the discounted total. Inventory is updated.

* **Note**: NF

### Test Case 2: Successful Order with No Discount and PayPal Payment

* **ID**: TC002
* **Description**: Tests a successful order with no discount code, paid with PayPal.
* **Preconditions**:

The customer is logged in, and the cart has at least one item.

* **Test Steps**:
  1. Customer adds Product A ($50) to the cart.
  2. Customer skips entering a discount code.
  3. Customer goes to checkout.
  4. System shows the total without any discount.
  5. Customer selects PayPal and logs into their PayPal account.
  6. System processes the payment.
  7. System confirms the order and sends a confirmation.
* **Expected Result**:

Customer receives order confirmation with the full price. Inventory is updated.

* **Note**: NF

### Test Case 3: Attempt to Checkout with Only Invalid Discount Code

* **ID**: TC003
* **Description**: Tests the system response when the customer applies only an invalid discount code and tries to proceed.
* **Preconditions**:

The customer is logged in, and the cart has at least one item.

* **Test Steps**:
  1. Customer adds Product C ($20) to the cart.
  2. Customer enters an invalid discount code DISCOUNT100.
  3. System checks the code and shows an error message saying it’s invalid.
  4. Customer proceeds to checkout without the discount.
  5. System shows the full total without any discount.
  6. Customer provides valid credit card information.
  7. System processes the payment.
  8. System confirms the order and sends a confirmation.
* **Expected Result**:

Customer receives order confirmation with the full price. Inventory is updated.

* **Note**: AL

### Test Case 4: Payment Failure due to Expired Credit Card

* **ID**: TC004
* **Description**: Tests payment failure because the customer used an expired credit card.
* **Preconditions**: The customer is logged in, and the cart has at least one item.
* **Test Steps**:
  1. Customer adds Product A ($50) and Product B ($30) to the cart.
  2. Customer skips applying a discount code.
  3. Customer goes to checkout.
  4. System shows the total amount.
  5. Customer provides expired credit card details.
  6. System tries to process the payment.
* **Expected Result**:

System shows a payment failure message due to an expired card and asks the customer to enter valid card details.

* **Note**: EX

### Test Case 5: Checkout with No Items and Redirect to Cart

* **ID**: TC005
* **Description**: Tests that the system redirects the customer to the cart if they try to checkout with an empty cart.
* **Preconditions**:

The customer is logged in, and the cart is empty.

* **Test Steps**:
  1. Customer tries to go to checkout with an empty cart.
* **Expected Result**:

System shows a message telling the customer to add items to the cart before proceeding and redirects them to the cart page.

* **Note**: AL

### Test Case 6: Successful Order with Multiple Items and SAVE10 Discount Code

* **ID**: TC006
* **Description**: Tests a successful order with multiple items and the SAVE10 discount code.
* **Preconditions**:

The customer is logged in, and the cart has at least two items.

* **Test Steps**:
  1. Customer adds Product A ($50) and Product B ($30) to the cart.
  2. Customer applies the discount code SAVE10.
  3. System validates the discount code.
  4. Customer proceeds to checkout.
  5. System displays the total with a 10% discount.
  6. Customer provides valid credit card information.
  7. System processes the payment.
  8. System confirms the order and sends a confirmation.
* **Expected Result**:

Customer receives order confirmation with the total discounted price. Inventory is updated.

* **Note**: NF