Exercise: CS1 – Result Verification

- **Situation:**
  - You have just inherited maintenance of the Flight Management System. The good news is that there are automated unit tests. The bad news is that most of the tests look something like these tests.

- **Instructions:**
  - Examine the code in the handout and determine what code smells you are seeing.

- **Discussion Questions:**
  - Which Code Smells are we having?
  - What are the underlying root causes?
  - Which XUnit Patterns can we apply to alleviate them?

---

The Whole Test

```java
public void testAddItemQuantity_severalQuantity() throws Exception {
    try {
        // Setup Fixture
        final int QUANTITY = 5;
        Address billingAddress = new Address("1222 1st St SW", "Calgary", "Alberta", "T2N 2V2", "Canada");
        Address shippingAddress = new Address("1333 1st St SW", "Calgary", "Alberta", "T2N 2V2", "Canada");
        Customer customer = new Customer(99, "John", "Doe", new BigDecimal("30"),
                                        billingAddress, shippingAddress);
        Invoice invoice = new Invoice(customer);
        // Exercise SUT
        invoice.addItemQuantity(product, QUANTITY);
        // Verify Outcome
        List lineItems = invoice.getLineItems();
        assertEquals("number of items", lineItems.size(), 1);
        LineItem actualLineItem = (LineItem)lineItems.get(0);
        LineItem expectedLineItem = new LineItem(invoice, product, QUANTITY);
        assertEquals(expectedLineItem, actualLineItem);
    } finally {
        deleteObject(expectedLineItem);
        deleteObject(invoice);
        deleteObject(product);
        deleteObject(customer);
        deleteObject(billingAddress);
        deleteObject(shippingAddress);
    }
}
```
Inline Fixture Teardown - Naive

```
try {
    // Setup Fixture
    // Exercise SUT
    // Verify Outcome
} finally {
    deleteObject(expectedLineItem);
    deleteObject(invoice);
    deleteObject(product);
    deleteObject(customer);
    deleteObject(billingAddress);
    deleteObject(shippingAddress);
}
```

Inline Fixture Teardown - Robust

```
try {
    // Setup Fixture
    // Exercise SUT
    // Verify Outcome
} finally {
    try {
        deleteObject(expectedLineItem);
    } finally {
        try {
            deleteObject(invoice);
        } finally {
            try {
                deleteObject(product);
            } finally {
            }
        }
    }
}
```
xUnit Test Patterns and Smells

Pattern

**Automated Fixture Teardown**

```java
public void testAddItemQuantity_severalQuantity () {
    final int QUANTITY = 5;
    Address billingAddress = new Address("1222 1st St SW",
        "Calgary", "Alberta", "T2N 2V2", "Canada");
    addTestObject(billingAddress);
    Address shippingAddress = new Address("1333 1st St SW",
        "Calgary", "Alberta", "T2N 2V2", "Canada");
    :
}
```

```java
public void tearDown() {
    deleteAllTestObjects();
}
```

Aug 16, 2007
©2006, 2007 Gerard Meszaros
TPS-C-27

xUnit Test Patterns and Smells

Pattern

**Automated Fixture Teardown**

```java
public void deleteAllTestObjects() {
    Iterator i = testObjects.iterator();
    while (i.hasNext()) {
        try {
            Deletable object = (Deletable) i.next();
            object.delete();
        } catch (Exception e) {
            // do nothing if the remove failed
        }
    }
}
```

Aug 16, 2007
©2006, 2007 Gerard Meszaros
TPS-C-28
**Smell**

**Hard-Coded Test Data**

```java
public void testAddItemQuantity_severalQuantity() {
    final int QUANTITY = 5;
    Address billingAddress = new Address("1222 1st St SW", "Calgary", "Alberta", "T2N 2V2", "Canada");

    Address shippingAddress = new Address("1333 1st St SW", "Calgary", "Alberta", "T2N 2V2", "Canada");

    Customer customer = new Customer(99, "Dahn", "Doe", new BigDecimal("30"), billingAddress, shippingAddress);


    Invoice invoice = new Invoice(customer);
    // Exercise SUT
    invoice.addItemQuantity(product, QUANTITY);
}
```

**Pattern**

**Distinct Generated Values**

```java
public void testAddItemQuantity_severalQuantity() {
    final int QUANTITY = 5;
    Address billingAddress = new Address(getUniqueString(), getUniqueString(), getUniqueString(),
                                          getUniqueString(), getUniqueString());

    Address shippingAddress = new Address(getUniqueString(), getUniqueString(),
                                          getUniqueString(), getUniqueString());

    Customer customer = new Customer(getUniqueInt(), getUniqueString(),
                                      getUniqueString(), getUniqueDiscount(),
                                      billingAddress, shippingAddress);

    Product product = new Product(getUniqueInt(), getUniqueString(),
                                   getUniqueNumber());

    Invoice invoice = new Invoice(customer);
}
Distinct Generated Values

```java
public void testAddItemQuantity_severalQuantity() {
    final int QUANTITY = 5;
    Address billingAddress = new Address(getUniqueString(),
                                          getUniqueString(),
                                          getUniqueString());
    Address shippingAddress = new Address(getUniqueString(),
                                          getUniqueString(),
                                          getUniqueString());
    Customer customer = new Customer(
                                       getUniqueInt(),
                                       getUniqueString(),
                                       getUniqueString(),
                                       getUniqueDiscount(),
                                       billingAddress,
                                       shippingAddress);
    Product product = new Product(
                                getUniqueInt(),
                                getUniqueString(),
                                getUniqueString());
    Invoice invoice = new Invoice(customer);
}
```

Creation Method

```java
public void testAddItemQuantity_severalQuantity() {
    final int QUANTITY = 5;
    Address billingAddress = createAnonymousAddress();
    Address shippingAddress = createAnonymousAddress();
    Customer customer = createCustomer(billingAddress,
                                        shippingAddress);
    Product product = createAnonymousProduct();
    Invoice invoice = new Invoice(customer);
}
```
### Obscure Test - Irrelevant Information

```java
public void testAddItemQuantity_severalQuantity() {
    final int QUANTITY = 5;
    Address billingAddress = createAnonymousAddress();
    Address shippingAddress = createAnonymousAddress();
    Customer customer = createCustomer(billingAddress, shippingAddress);
    Product product = createAnonymousProduct();
    Invoice invoice = new Invoice(customer);
    // Exercise
    invoice.addItemQuantity(product, QUANTITY);
    // Verify
    LineItem expectedLineItem = newLineItem(invoice, product, QUANTITY, product.getPrice() * QUANTITY);
    List LineItems = invoice.getLineItems();
    assertEquals("number of items", LineItems.size(), 1);
    LineItem actualLineItem = (LineItem) LineItems.get(0);
    assertEquals(expectedLineItem, actualLineItem);
}
```

---

### Remove Irrelevant Information

```java
public void testAddItemQuantity_severalQuantity() {
    final int QUANTITY = 5;

    Customer customer = createAnonymousCustomer();

    Product product = createAnonymousProduct();
    Invoice invoice = new Invoice(customer);
    // Exercise
    invoice.addItemQuantity(product, QUANTITY);
    // Verify
    LineItem expectedLineItem = newLineItem(invoice, product, QUANTITY, product.getPrice() * QUANTITY);
    List LineItems = invoice.getLineItems();
    assertEquals("number of items", LineItems.size(), 1);
    LineItem actualLineItem = (LineItem) LineItems.get(0);
    assertEquals(expectedLineItem, actualLineItem);
}
```
**xUnit Test Patterns and Smells**

### Refactoring: Remove Irrelevant Information

```java
public void testAddItemQuantity_severalQuantity() {
    final int QUANTITY = 5;

    Product product = createAnonymousProduct();
    Invoice invoice = createAnonymousInvoice()
    // Exercise
    invoice.addItemQuantity(product, QUANTITY);
    // Verify
    LineItem expectedLineItem = newLineItem(invoice, product, QUANTITY, product.getPrice() * QUANTITY);
    List lineItems = invoice.getLineItems();
    assertEquals("number of items", lineItems.size(), 1);
    LineItem actualLineItem = (LineItem) lineItems.get(0);
    assertEquals(expectedLineItem, actualLineItem);
}
```

### Refactoring: Introduce Custom Assertion

```java
public void testAddItemQuantity_severalQuantity() {
    final int QUANTITY = 5;

    Product product = createAnonymousProduct();
    Invoice invoice = createAnonymousInvoice()
    // Exercise
    invoice.addItemQuantity(product, QUANTITY);
    // Verify
    LineItem expectedLineItem = newLineItem(invoice, product, QUANTITY, product.getPrice() * QUANTITY);
    List lineItems = invoice.getLineItems();
    assertEquals("number of items", lineItems.size(), 1);
    LineItem actualLineItem = (LineItem) lineItems.get(0);
    assertEquals(expectedLineItem, actualLineItem);
}
```
**Refactoring**

**Introduce Custom Assertion**

```java
public void testAddItemQuantity_severalQuantity() {
    final int QUANTITY = 5;

    Product product = createAnonymousProduct();
    Invoice invoice = createAnonymousInvoice();
    // Exercise
    invoice.addItemQuantity(product, QUANTITY);
    // Verify
    LineItem expectedLineItem = newLineItem(invoice, product, QUANTITY, product.getPrice() * QUANTITY);
    assertExactlyOneLineItem(invoice, expectedLineItem);
}
```

**The Whole Test – Done**

```java
public void testAddItemQuantity_severalQuantity() {
    final int QUANTITY = 5;
    Product product = createAnonymousProduct();
    Invoice invoice = createAnonymousInvoice();
    // Exercise
    invoice.addItemQuantity(product, QUANTITY);
    // Verify
    LineItem expectedLineItem = newLineItem(invoice, product, QUANTITY, product.getPrice() * QUANTITY);
    assertExactlyOneLineItem(invoice, expectedLineItem);
}
```

```java
Customer createAnonymousCustomer() {
    BigDecimal uniqueId = getUniqueIntForTest();
    Address billingAddress = createAnonymousAddress();
    Address shippingAddress = createAnonymousAddress();
    Customer customer = new Customer(
        getUniqueInt(), getUniqueString(),
        getUniqueString(), getUniqueString(), billingAddress,
        shippingAddress);
}
```
**Test Coverage**

`TestInvoiceLineItems extends TestCase {
    TestAddItemQuantity_oneItem {..}
    TestAddItemQuantity_severalItems {..}
    TestAddItemQuantity_duplicateProduct {..}
    TestAddItemQuantity_zeroQuantity {..}
    TestAddItemQuantity_severalQuantity {..}
    TestAddItemQuantity_discountedPrice {..}
    TestRemoveItem_noItemsLeft {..}
    TestRemoveItem_oneItemLeft {..}
    TestRemoveItem_severalItemsLeft {..}
} `