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

```
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);
    Product product = new Product(88, "SomeWidget", new BigDecimal("19.99"));
    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 = newLineItem(invoice, product, QUANTITY);
    assertLineItemsEqual(expectedLineItem, actualLineItem);
  } finally {
    deleteObject(expectedLineItem);
    deleteObject(invoice);
    deleteObject(product);
    deleteObject(customer);
    deleteObject(billingAddress);
    deleteObject(shippingAddress);
```

Pattern

Inline Fixture Teardown - Naive

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

Pattern

Inline Fixture Teardown - Robust

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

Pattern

Implicit Fixture Teardown - Naive

```
public void testAddItemQuantity_severalQuantity ()
   throws Exception {
    // Setup Fixture
    // Exercise SUT
    // Verify Outcome
}


public void tearDown() {
    deleteObject(expectedLineItem);
    deleteObject(invoice);
    deleteObject(product);
    deleteObject(customer);
    deleteObject(billingAddress);
    deleteObject(shippingAddress);
}
```


Pattern

Implicit Fixture Teardown - Robust

```
public void testAddItemQuantity_severalQuantity ()
   throws Exception {
    // Setup Fixture
    // Exercise SUT
    // Verify Outcome
}


public void tearDown() {
    try {
        deleteObject(expectedLineItem);
    } finally {
       try {
          deleteObject(invoice);
       } finally {
           try {
              deleteObject(product);
           } finally {
                       :
```

Pattern

Automated Fixture Teardown

```
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");
   addTestObject(shippingAddress );

          :
}

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


Pattern

Automated Fixture Teardown

```
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
         }
     }
}
```

Pattern

Transaction Rollback Teardown

```
public void setUp() {
    TransactionManager.beginTransaction();
}

public void tearDown() {
    TransactionManager.abortTransaction();
}
```

Important: SUT must not commit transaction

- DFT Pattern: Humble Transaction Controller

The Whole Test

```
public void testAddItemQuantity_severalQuantity() throws Exception {
    // Setup Fixture
    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");
    addTestObject(billingAddress);
    Customer customer = new Customer(99, "John", "Doe", new BigDecimal("30"),
         billingAddress, shippingAddress);
    addTestObject(billingAddress);
    Product product = new Product(88, "SomeWidget", new BigDecimal("19.99"));
    addTestObject(billingAddress);
    Invoice invoice = new Invoice(customer);
    addTestObject(billingAddress);
    // Exercise SUT
    invoice.addItemQuantity(product, QUANTITY);
    // Verify Outcome
    assertEquals("number of items",lineItems.size(),1);
    LineItem actualLineItem = (LineItem)lineItems.get(0);
    LineItem expectedLineItem = newLineItem(invoice, product, QUANTITY);
    assertLineItemsEqual(expectedLineItem, actualLineItem);
}                                          // No Visible Fixture Tear Down!
```

The Whole Test

```
public void testAddItemQuantity_severalQuantity() throws Exception {
    // Setup Fixture
    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");
    addTestObject(billingAddress);
    Customer customer = new Customer(99, "John", "Doe", new BigDecimal("30"),
         billingAddress, shippingAddress);
    addTestObject(billingAddress);
    Product product = new Product(88, "SomeWidget", new BigDecimal("19.99"));
    addTestObject(billingAddress);
    Invoice invoice = new Invoice(customer);
    addTestObject(billingAddress);
    // Exercise SUT
    invoice.addItemQuantity(product, QUANTITY);
    // Verify Outcome
    assertEquals("number of items",lineItems.size(),1);
    LineItem actualLineItem = (LineItem)lineItems.get(0);
    LineItem expectedLineItem = newLineItem(invoice, product, QUANTITY);
    assertLineItemsEqual(expectedLineItem, actualLineItem);
}
```

// No Visible Fixture Tear Down!

The Smells Seen Thus Far

- **Complex Undo Logic**
  - Complex fixture teardown code
  - More likely to leave test environment corrupted leading to Erratic Tests (Causes: Unrepeatable Tests or Interacting Tests)

The Patterns Used So Far

- **Inline Teardown**
  - Hand-coded tear down logic within the Test Method
- **Implicit Teardown**
  - Hand-coded tear down logic in a tearDown method
- **Automated Teardown**
  - Tear down all registered test objects programatically
- **Transaction Rollback Teardown**
  - Get the database to undo all the changes made by test
  - SUT must not commit transaction

The Whole Test

```
public void testAddItemQuantity_severalQuantity () throws Exception {
    // Setup Fixture
    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");
    addTestObject(billingAddress);
    Customer customer = new Customer(99, "John", "Doe", new BigDecimal("30"),
         billingAddress, shippingAddress);
    addTestObject(billingAddress);
    Product product = new Product(88, "SomeWidget", new BigDecimal("19.99"));
    addTestObject(billingAddress);
    Invoice invoice = new Invoice(customer);
    addTestObject(billingAddress);
    // Exercise SUT
    invoice.addItemQuantity(product, QUANTITY);
    // Verify Outcome
    assertEquals("number of items",lineItems.size(),1);
    LineItem actualLineItem = (LineItem)lineItems.get(0);
    LineItem expectedLineItem = newLineItem(invoice, product, QUANTITY);
    assertLineItemsEqual(expectedLineItem, actualLineItem);
}
```

Smell

Hard-Coded Test Data

```
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, "John", "Doe", new
        BigDecimal("30"), billingAddress, shippingAddress);

    Product product = new Product(88, "SomeWidget",
        BigDecimal("19.99"));

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

Hard-coded Test Data (Obscure Test)

Unrepeatable Tests

Pattern

Distinct Generated Values

```
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(), getUniqueString());
    Customer customer = new Customer(
        getUniqueInt(), getUniqueString(),
        getUniqueString(), getUniqueDiscount(),
        billingAddress, shippingAddress);
    Product product = new Product(
        getUniqueInt(), getUniqueString(),
        getUniqueNumber());
    Invoice invoice = new Invoice(customer);
```

Pattern

Distinct Generated Values

```
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(), getUniqueString());
   Customer customer = new Customer(
       getUniqueInt(), getUniqueString(),
       getUniqueString(), getUniqueDiscount(),
       billingAddress, shippingAddress);
   Product product = new Product(
       getUniqueInt(), getUniqueString(),
       getUniqueNumber());
   Invoice invoice = new Invoice(customer);
```

Irrelevant Information (Obscure Test)

Pattern

Creation Method

```
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);
```

Smell

Obscure Test - Irrelevant Information

```
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);
    assertLineItemsEqual(expectedLineItem, actualLineItem);
}
```

Irrelevant Information (Obscure Test)

Refactoring

Remove Irrelevant Information

```
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);
    assertLineItemsEqual(expectedLineItem, actualLineItem);
}
```

Irrelevant Information (Obscure Test)

Refactoring

Remove Irrelevant Information

```
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);
    assertLineItemsEqual(expectedLineItem, actualLineItem);
}
```


Refactoring

Introduce Custom Assertion

```
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);
    assertLineItemsEqual(expectedLineItem, actualLineItem);
}
```

Mechanics hides Intent

Refactoring

Introduce Custom Assertion

```
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

```
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 );
}

Customer createAnonymousCustomer() {
   BigDecimal uniqueId = getUniqueIdForTest()
   Address billingAddress = createAnonymousAddress();
   Address shippingAddress = createAnonymousAddress();
   Customer customer = new Customer(
     getUniqueInt(), getUniqueString(),
     getUniqueString(), getUniqueDiscount(), 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 {..}
}
```

Pattern: Testcase Class per Feature

Rapid Test Writing

```
public void testAddItemQuantity_severalItems () {
   final int QUANTITY = 1 ;
   Product product1 = createAnonymousProduct();
   Product product2 = createAnonymousProduct();
   Invoice invoice = createAnonymousInvoice();
   // Exercise
   invoice.addItemQuantity(product1, QUANTITY);
   invoice.addItemQuantity(product2, QUANTITY);
   // Verify
   LineItem expectedLineItem1 =  newLineItem(invoice,
      product, QUANTITY, product.getPrice()*QUANTITY );
   LineItem expectedLineItem2 =  newLineItem(invoice,
      product, QUANTITY, product.getPrice()*QUANTITY );
   assertExactlyTwoLineItems(invoice,
                  expectedLineItem1, expectedLineItem2 );
}
```

The Smells Seen Thus Far

- **Obscure Test**

  The test is hard to understand. Specific causes:

  – Hard-Coded Test Data
    » Literal Constants
  – Irrelevant Information
    » Information in test unrelated to SUT behavior

The Patterns Used so Far

- **Generated Value**
  – Variation: Distinct Generated Value
    » Generate a unique value for each test run
- **Creation Method**
  – Anonymous Creation Method
    » Sets all attributes/references to default values
  – Parameterized Creation Method
    » Tests specifies relevant values only
- **Testcase Class per Feature**
  – Group all Test Methods for a feature or concept on a single class
  – Alternatives: Testcase Class per Class, Testcase Class per Fixture
- **Custom Assertion**
    » (again)