Mocks and Stubs

d'après Martin Fowler –

http://www.martinfowler.com/articles/mocksArentStubs.html

Légèrement incrémenté par
M. Blay-Fornarino

Example – Electronic Store

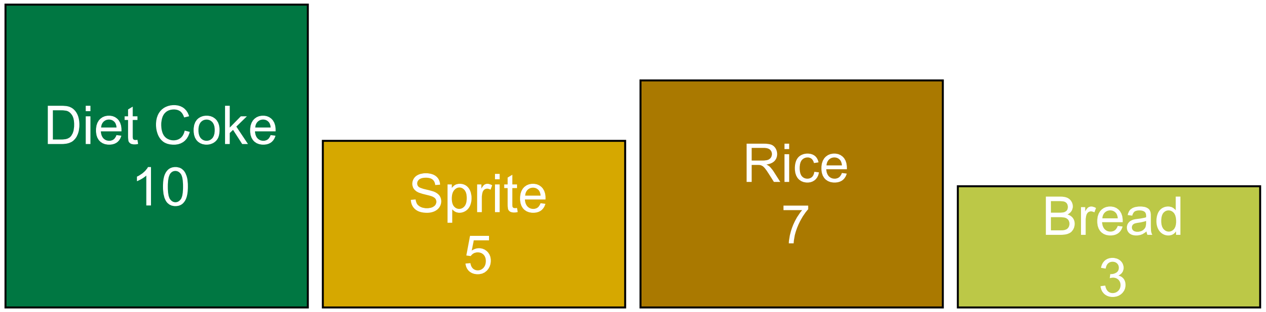
- *Orders* and a *Warehouse*

Order1: Diet Coke - 5

Order2: Diet Coke - 2

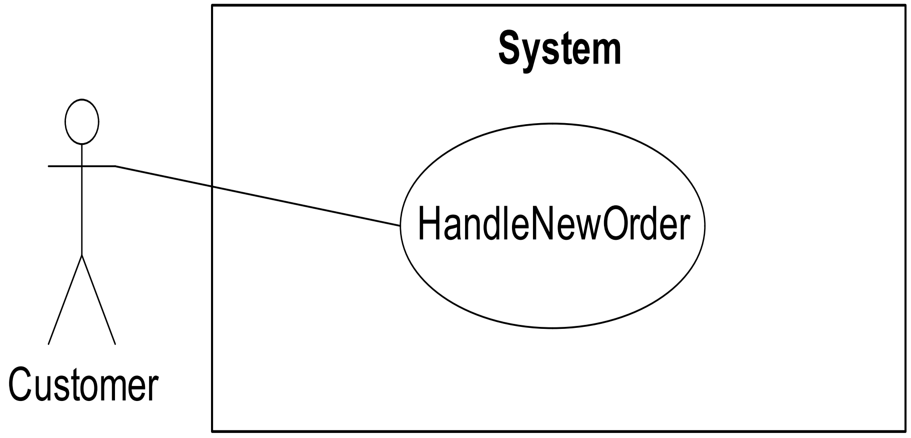
Order3: Sprite - 3

Order4: Bread - 1



Example – Electronic Store

- Use Case Model



- System Sequence

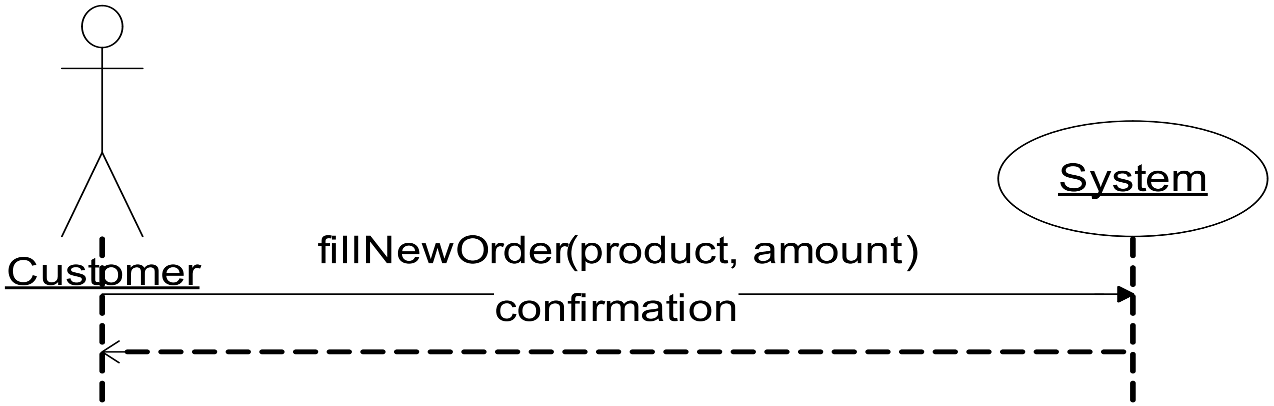


Diagramme de séquence (Conception)

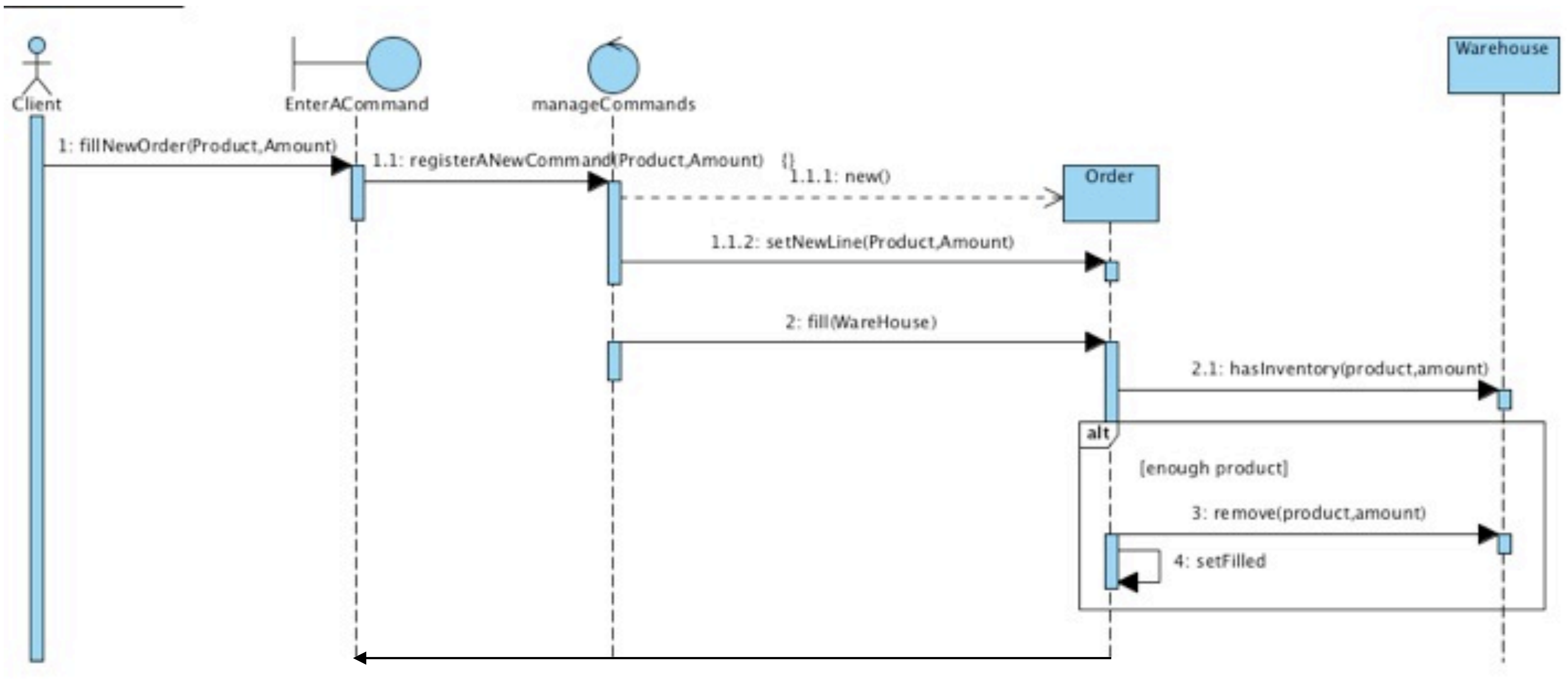
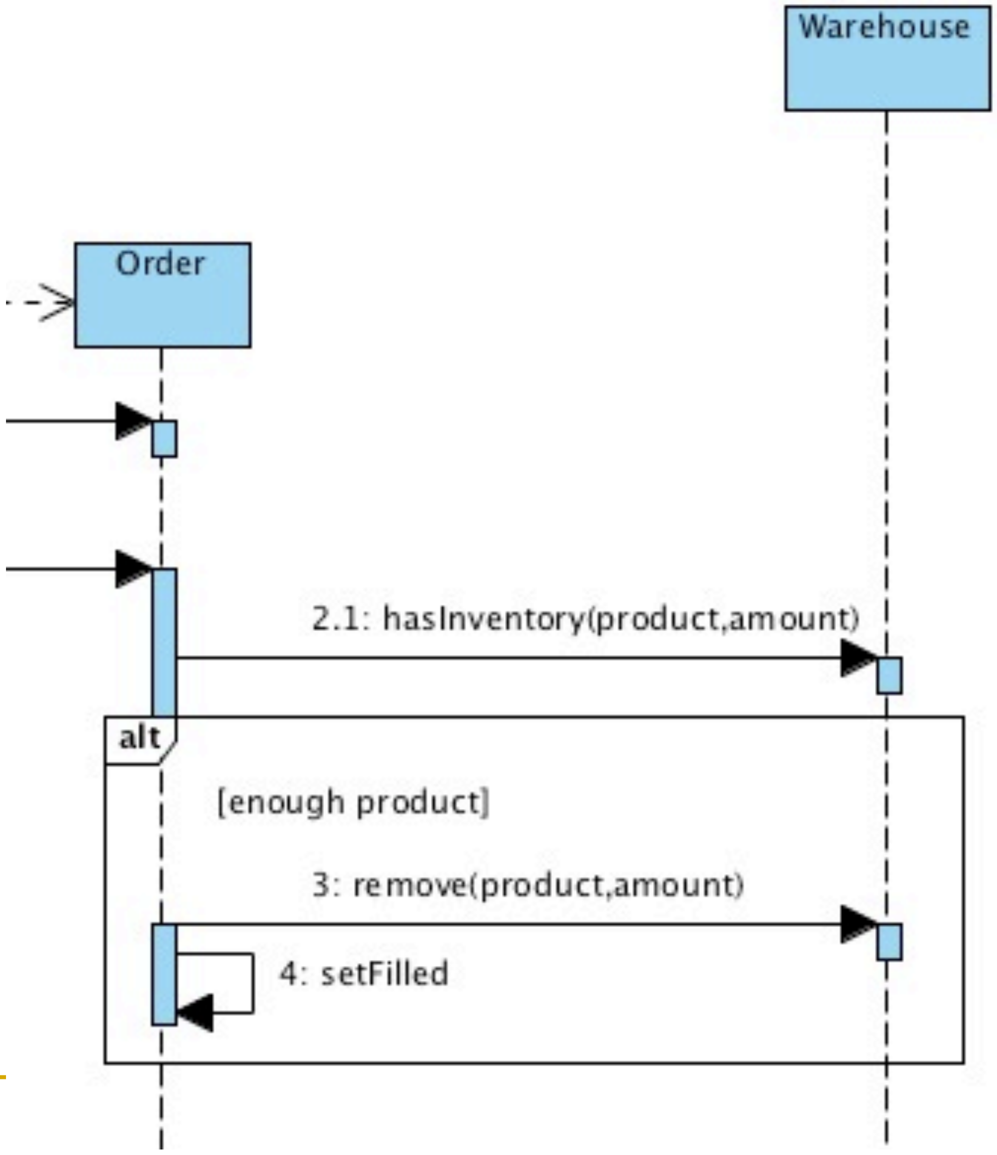
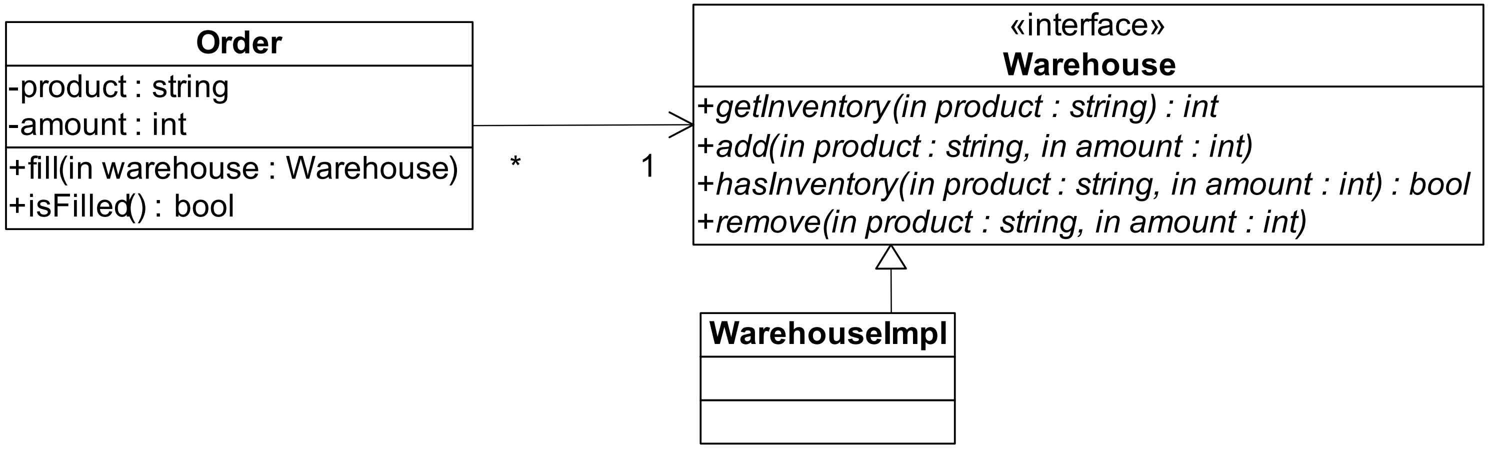

Client
EnterACommand
manageCommands
Warehouse
1: fillNewOrder(Product,Amount)
1.1: registerANewCommand(Product,Amount) {}
1.1.1: new()
Order
1.1.2: setNewLine(Product,Amount)
2: fill(WareHouse)
2.1: hasInventory(product,amount)
alt
[enough product]
3: remove(product,amount)
4: setFilled

Diagramme de séquence (Conception)



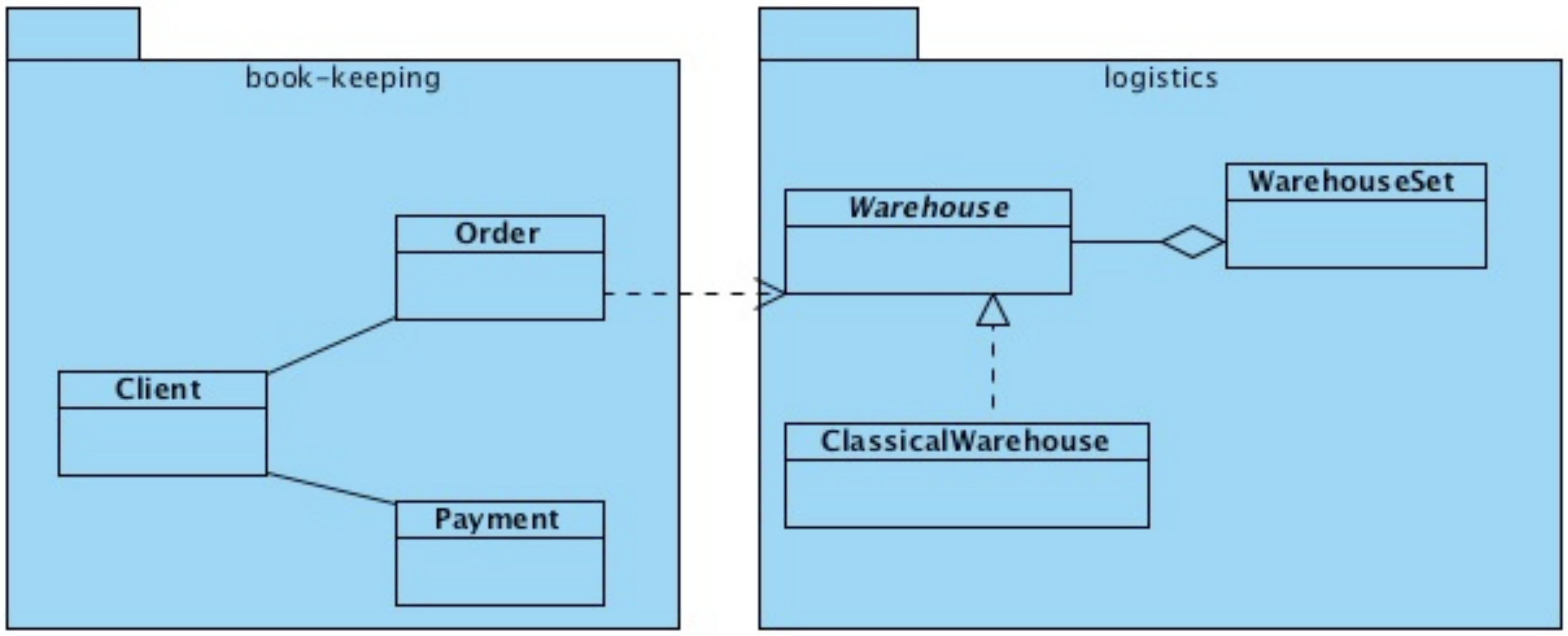
Example – Electronic Store

- Domain Model



Order
-product : string
-amount : int
+fill(in warehouse : Warehouse)
+isFilled() : bool
*
1
«interface»
Warehouse
+getInventory(in product : string) : int
+add(in product : string, in amount : int)
+hasInventory(in product : string, in amount : int) : bool
+remove(in product : string, in amount : int)
WarehouseImpl

Packages & Séparation des responsabilités



Example – Electronic Store

- Testing the **Order** class

  …….

```
public class Order {
		…
		public Order(String product, int i) {
				this.product = product;
				this.amount = i;
				this.isFilled = false;
		}

		public void fill(Warehouse warehouse) {
				if (warehouse.hasInventory(product,amount)) {
						warehouse.remove(product,amount);
						isFilled = true;
				}
		}

		public boolean isFilled() {
				return isFilled;
		}
}
```

Example – Electronic Store

- Testing the **Order** class

```
public class OrderStateTester extends TestCase {
        …
        public void testOrderIsFilledIfEnoughInWarehouse(){
                Order order = new Order(DIET_COKE,5);
                order.fill(warehouse);
                // Primary object test
                assertTrue(order.isFilled());
                // Secondary object test(s)
                assertEquals(0,warehouse.getInventory(DIET_COKE));
        }

        public void testOrderDoesNotRemoveIfNotEnough(){
                Order order = new Order(SPRITE,11);
                order.fill(warehouse);
                // Primary object test
                assertFalse(order.isFilled());
                // Secondary object test(s)
                assertEquals(10, warehouse.getInventory(SPRITE));
        }
}
```

Example – Electronic Store

- Testing the **Order** class:

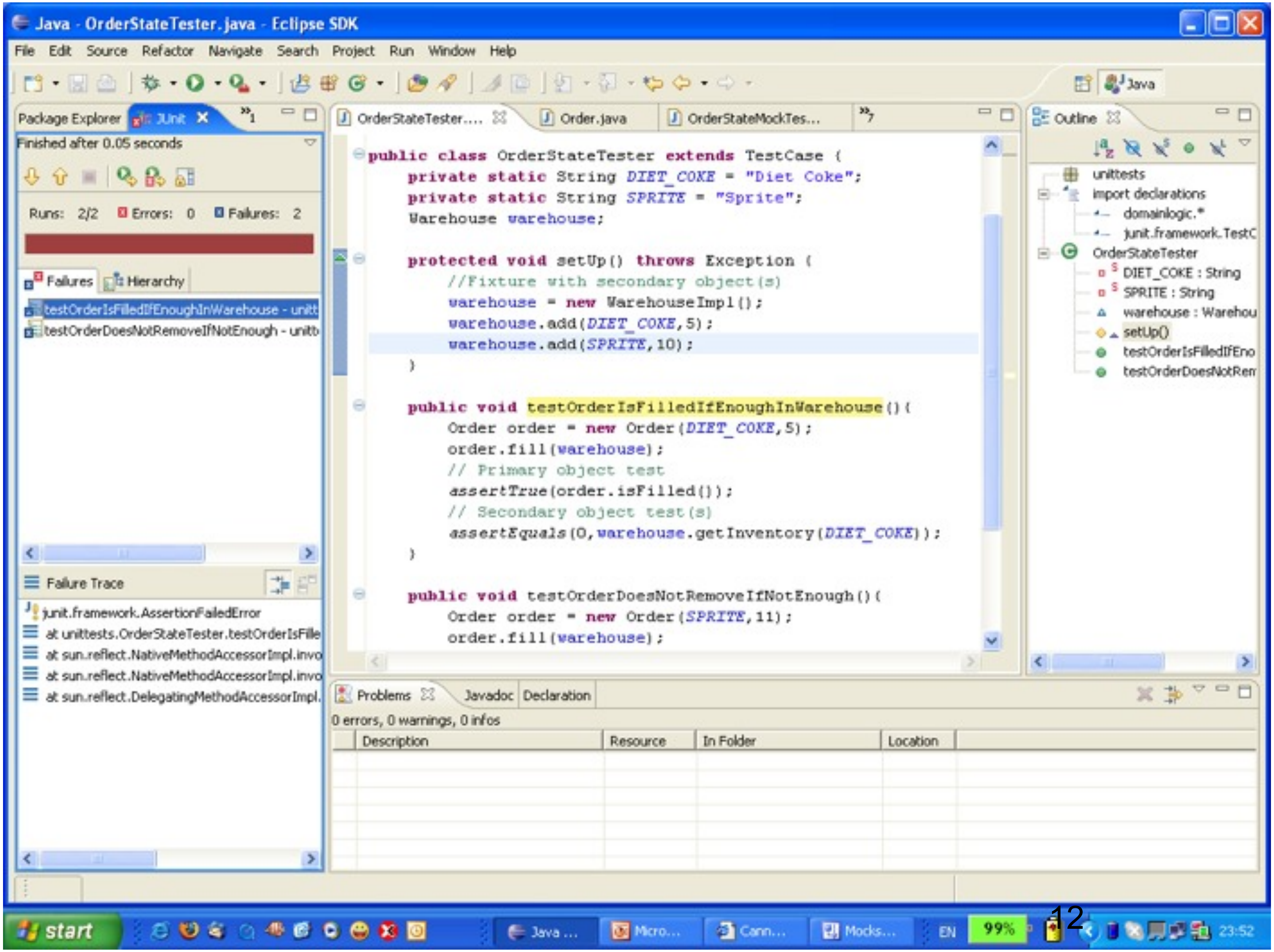
```
public class OrderStateTester extends TestCase {
        private static String DIET_COKE = "Diet Coke";
        private static String SPRITE = "Sprite";
        Warehouse warehouse;

        protected void setUp() throws Exception {
                //Fixture with secondary object(s)
                warehouse = new WarehouseImpl();
                warehouse.add(DIET_COKE,5);
                warehouse.add(SPRITE,10);
        }
        …
```

Example – Electronic Store

Stub

- Using a *stub* to run the tests -
  - Stubs return canned data to methods calls:

```
public class WarehouseImpl implements Warehouse {

        public void add(String product, int i) {}

        public int getInventory(String product) {
                return 0;
        }

        public boolean hasInventory(String product) {
                return false;
        }

        public void remove(String product, int i) {}
}
```



testOrderIsFilledIfEnoughInWarehouse - unitt
testOrderDoesNotRemoveIfNotEnough - unitt

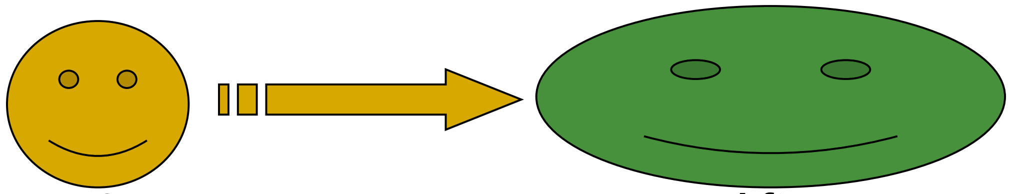
```
junit.framework.AssertionFailedError
at unittests.OrderStateTester.testOrderIsFille
at sun.reflect.NativeMethodAccessorImpl.invo
at sun.reflect.NativeMethodAccessorImpl.invo
at sun.reflect.DelegatingMethodAccessorImpl.
```

```java
public class OrderStateTester extends TestCase {
    private static String DIET_COKE = "Diet Coke";
    private static String SPRITE = "Sprite";
    Warehouse warehouse;

    protected void setUp() throws Exception {
        //Fixture with secondary object(s)
        warehouse = new WarehouseImpl();
        warehouse.add(DIET_COKE,5);
        warehouse.add(SPRITE,10);
    }

    public void testOrderIsFilledIfEnoughInWarehouse(){
        Order order = new Order(DIET_COKE,5);
        order.fill(warehouse);
        // Primary object test
        assertTrue(order.isFilled());
        // Secondary object test(s)
        assertEquals(0,warehouse.getInventory(DIET_COKE));
    }

    public void testOrderDoesNotRemoveIfNotEnough(){
        Order order = new Order(SPRITE,11);
        order.fill(warehouse);
```

Example – Electronic Store

- The tests fail since the stub object – **warehouse** (secondary) misses the required functionality
- Remember: the intension is to test the behavior of the primary object - **Order**, all other objects are tested in their own corresponding tests.
- The test is only *State-Based*
  - E.g., was the *remove()* method invoked? Other methods of the warehouse object?

Example – Electronic Store



- State Based tests:

  - All objects involved must be created – **complex fixture**
  - After the primary object was “kicked” with the behavior that is being tested, the **result** is evaluated against:
    - The primary object
    - All secondary objects
  - If the test fails, its source might be fuzzy between the primary and the secondary objects
  - No interaction is being tested!
- A possible solution – *Mock* objects

Example – Electronic Store

- Interaction based test:
  - Tests check which methods were called on the mock
  - All secondary objects are replaced with mocks
  - Expectations → provide the specification for the interface of the secondary objects
  - Test Isolation: Bugs will be reflected in tests related to primary objects
  - Strongly coupled with the implementation → may interfere with refactoring
  - Don’t check objects integration

Example – Electronic Store

- Tests basés sur les interactions
  - Les tests doivent vérifier quelles méthodes ont été appelées sur les objets secondaires.
  - Tous les objets secondaires sont remplacés par des «mocks»
  - => spécification des interfaces des objets secondaires
  - Test en Isolation: Les Bugs détectés dans les tests sont uniquement liés aux objets primaires

  - Fortement couplés avec la mise en œuvre => peuvent interférer avec la refactorisation

Using EasyMock

- Define only the interface of the Mock object:

```
public interface Warehouse {

        void add(String product, int i);
        int getInventory(String product);
        boolean hasInventory(String product,int amount);
        void remove(String product, int i);
}
```

Using EasyMock

- Create the Mock object:

```
protected void setUp() throws Exception {
        //Fixture with secondary object(s)
        mock = createMock(Warehouse.class);
}
```

You need to :

- Add the EasyMock jar file (easymock.jar) from this directory to your classpath
- *import static org.easymock.EasyMock.*;*

Using EasyMock

- Running tests with expectations:

```
public void testOrderIsFilledIfEnoughInWarehouse(){
        //Expectations
        expect(mock.hasInventory(DIET_COKE,5)).andReturn(true);
        mock.remove(DIET_COKE,5);
        replay(mock);

        Order order = new Order(DIET_COKE,5);
        order.fill(mock);
        // Primary object test
        assertTrue(order.isFilled());

       // Secondary object test(s)
        verify(mock);
}
```

Using EasyMock

- Verifying Behavior
  - If the method is not called on the Mock Object

```
public void testDemo(){
		mock.remove("cola",2);
		replay(mock);

		verify(mock);
}
```

```
java.lang.AssertionError:
  Expectation failure on verify:
    remove("cola", 2): expected: 1, actual: 0
```

Using EasyMock

- Verifying Behavior
  - If the method is not called on the Mock Object

```
public void testDemo(){
        mock.remove("cola",2);
        replay(mock);
        Order order = new Order(SPRITE,11);
        order.fill(mock);
        verify(mock);
}
```

```
java.lang.AssertionError:
  Unexpected method call hasInventory("Sprite", 11):
    remove("cola", 2): expected: 1, actual: 0
```