Contents

[1. JUnit & Mockito 1](#_Toc181617004)

[1. **TestInMemoryDatabaseRetrieval** 2](#_Toc181617005)

[2. **TestReceiptProcessing**: 2](#_Toc181617006)

[3. **TestDonationFunctionality** 3](#_Toc181617007)

[4. **Test Nummer 1** 3](#_Toc181617008)

[2.Mockito 6](#_Toc181617009)

[1. testNeverCalledMethods : 6](#_Toc181617010)

[2. testGetItemByBarcode\_InvalidBarcode : 6](#_Toc181617011)

[3. Test4 : 7](#_Toc181617012)

[3. BDD (IGiven) 8](#_Toc181617013)

[1. Test Nummer 7 8](#_Toc181617014)

[**2.** Test Nummer 8 8](#_Toc181617015)

## 1. JUnit & Mockito

**Definition:** Zuerst müssen wir die Klassen und Methoden importieren und deklarieren, die wir brauchen.

@ private BottleDepositMachine bottleDepositMachine;  
private CardScanner cardScanner;  
private Display display;  
private SupervisoryModule supervisoryModule;  
private InMemoryDatabase inMemoryDatabase;  
private ReceiptProcessor receiptProcessor;  
private List<Item> testItems;

@BeforeEach  
void setUp() {  
 receiptProcessor = new ReceiptProcessor();  
 display = new Display();  
 bottleDepositMachine = new BottleDepositMachine(display,receiptProcessor);  
 cardScanner = new CardScanner(bottleDepositMachine);  
 supervisoryModule = new SupervisoryModule(bottleDepositMachine, cardScanner, display,receiptProcessor);  
 inMemoryDatabase = new InMemoryDatabase();  
 testItems = new ArrayList<>();  
 simulator = new Simulator(bottleDepositMachine);  
  
}

**Aufgaben des Tests:**

1. **TestInMemoryDatabaseRetrieval:** Wir prüfen, ob ein Artikel mit dem Barcode „r8yz7clkz4“ in inMemoryDatabase existiert.  
    assertNotNull(item)  
    Wenn ja, dann ist das Frontlabel gleich „ABC Can | 0.33L“  
   assertEquals("ABC Can | 0.33L", item.getFrontLabel());

@Test  
void testInMemoryDatabaseRetrieval() {  
 Item *item* = inMemoryDatabase.getItemByBarcode("r8yz7clkz4");  
 *assertNotNull*(*item*);  
 *assertEquals*("ABC Can | 0.33L", *item*.getFrontLabel());  
}

1. **TestReceiptProcessing**: Wir fügen einen Artikel hinzu   
    String[] itemData = {"ABC Can | 0.33L", "r8yz7clkz4", "Disposable", "Metal", "0.25"};  
   und versuchen zu sehen, ob wir z. B. in unserem Beleg die gleiche Summe haben wie die Summe der einzelnen Artikel.  
   *assertEquals*("0.25", receiptProcessor.getTotal());

@Test  
void testReceiptProcessing() {  
 String[] *itemData* = {"ABC Can | 0.33L", "r8yz7clkz4", "Disposable", "Metal", "0.25"};  
 receiptProcessor.addItemToReceipt(*itemData*);  
 *assertEquals*("0.25", receiptProcessor.getTotal());

1. **TestDonationFunctionality**: Wir fügen einen Artikel hinzu und tätigen eine Spende. Anschließend prüfen wir, ob die Spendenbasis nicht leer ist bzw dieser Artikel hat assertTrue(donationDatabase.hasDonation());

@Test  
void testDonationFunctionality() {  
 receiptProcessor.addItemToReceipt(new String[]{"ABC Can | 0.33L", "r8yz7clkz4", "Disposable", "Metal", "0.25"});  
 supervisoryModule.donate();  
 // You might want to assert the donation state or database state here  
 *assertTrue*(donationDatabase.hasDonation());  
}

1. **Test Nummer 1 :** Das war meine erste Lösung für diesen Test. Ich habe diese 6 Elemente zu meiner Array-Liste hinzugefügt und dann einige Variablen als Zähler erstellt, zum Beispiel „disposableItemsCount“. Dies war die gleiche Art und Weise, wie die Anwendungsklasse funktioniert.Anschließend werden alle Artikel über einen Flaschenpfandautomaten importiert und anschließend mit AsserEquals getestet. Weil im letzten Schritt viele Texte und Codes vorhanden waren und das zu kompilieren war, habe ich auch einen Simlator der Klasse geschrieben, damit man das ganze Programm testen kann.
2. A screenshot of a computer

   Description automatically generated

Accepted", "Glas", 0.00));  
 testItems.add(new Item("ABC Can | 0.5L", "tmvkrw69le", "Disposable", "Metal", 0.25));  
 testItems.add(new Item("FG Bottle | 1L", "8hgij9rqv5", "Reusable", "Plastic", 2.00));  
 testItems.add(new Item("X Can | 0.5L", "3a6mr6o7sl", "Not Accepted", "Metal", 0.00));  
 testItems.add(new Item("DE Bottle | 0.75L", "4xpokcvb7c", "Reusable", "Glas", 0.50));  
  
 // Unlock the machine  
 bottleDepositMachine.scanCard("employeeID1-Alex");  
  
 // Counters for processed items  
 int *disposableItemsCount* = 0;  
 int *reusableItemsCount* = 0;  
 int *nonAcceptedItemsCount* = 0;  
 double *disposableTotal* = 0.0;  
 double *reusableTotal* = 0.0;  
 double *totalForAll* = 0.0;  
  
 // Process bottles  
 for (Item *item* : testItems) {  
 bottleDepositMachine.inputBottle(*item*);  
 // Update counters based on the item type  
 if (*item*.getDepositAmount() > 0) {  
 *totalForAll* += *item*.getDepositAmount();  
 if (*item*.getRecyclingType().equalsIgnoreCase("Disposable")) {  
 *disposableItemsCount*++;  
 *disposableTotal* += *item*.getDepositAmount();  
 } else if (*item*.getRecyclingType().equalsIgnoreCase("Reusable")) {  
 *reusableItemsCount*++;  
 *reusableTotal* += *item*.getDepositAmount();  
 }  
 } else {  
 *nonAcceptedItemsCount*++;  
 }  
 }  
  
 // Finalize the transaction  
 bottleDepositMachine.clickOn("Finish");  
 bottleDepositMachine.clickOn("Print the receipt");  
  
 // Retrieve totals from the receipt processor  
 ReceiptProcessor *rp* = bottleDepositMachine.getReceiptProcessor();  
  
 // Assertions  
 *assertEquals*(1, *rp*.getDisposableCount()); // Disposable Count  
 *assertEquals*(3, *rp*.getReusableCount()); // Reusable Count  
 *assertEquals*(2, *rp*.getNonAcceptedCount()); // Not Accepted Count  
 // assertEquals(0, 0); // Not Accepted Total is always 0.00  
 *assertEquals*(0.25, *rp*.getDisposableTotal()); // Disposable Total price  
 *assertEquals*(3.25, *rp*.getReusableTotal()); // Reusable Total price  
 *assertEquals*(3.50, *rp*.getTotalForAll()); // Total for All  
}

1. package BottleDepositMachine.Software;  
     
   import BottleDepositMachine.BottleDepositMachine;  
   import Items.Item;  
     
   import java.util.List;  
     
   public class Simulator {  
    private BottleDepositMachine bottleDepositMachine;  
     
    public Simulator(BottleDepositMachine *bottleDepositMachine*) {  
    this.bottleDepositMachine = *bottleDepositMachine*;  
    }  
     
    public ReceiptProcessor runSimulation(List<Item> *items*) {  
    bottleDepositMachine.scanCard("employeeID1-Alex");  
     
    // Counters for processed items  
    int *disposableItemsCount* = 0;  
    int *reusableItemsCount* = 0;  
    int *nonAcceptedItemsCount* = 0;  
    double *disposableTotal* = 0.0;  
    double *reusableTotal* = 0.0;  
    double *totalForAll* = 0.0;  
     
    // Process bottles  
    for (Item *item* : *items*) {  
    bottleDepositMachine.inputBottle(*item*);  
     
    if (*item*.getDepositAmount() > 0) {  
    *totalForAll* += *item*.getDepositAmount();  
    if (*item*.getRecyclingType().equalsIgnoreCase("Disposable")) {  
    *disposableItemsCount*++;  
    *disposableTotal* += *item*.getDepositAmount();  
    } else if (*item*.getRecyclingType().equalsIgnoreCase("Reusable")) {  
    *reusableItemsCount*++;  
    *reusableTotal* += *item*.getDepositAmount();  
    }  
    } else {  
    *nonAcceptedItemsCount*++;  
    }  
    }  
     
    // Finalize the transaction  
    bottleDepositMachine.clickOn("Finish");  
    bottleDepositMachine.clickOn("Print the receipt");  
     
    // Retrieve totals from the receipt processor  
    return bottleDepositMachine.getReceiptProcessor();  
    }  
   }
2. **Test 2 :** Jetzt können wir viel Code speichern und das Programm einfach durch den Simulator ausführen. **A screen shot of a computer

   Description automatically generated**

@Test  
void testBottleProcessingWithSimulator() {  
 // Prepare test items  
 List<Item> *testItems* = new ArrayList<>();  
 *testItems*.add(new Item("DE Bottle | 0.33L", "bzfi339nsy", "Reusable", "Glass", 0.25));  
 *testItems*.add(new Item("Z Bottle | 0.33L", "j03eiqtm2t", "Not Accepted", "Glass", 0.00));  
 *testItems*.add(new Item("DE Bottle | 0.5L", "3jdwml7w52", "Reusable", "Glass", 0.30));  
 *testItems*.add(new Item("Y Bottle | 1L", "5hqoa0ean4", "Not Accepted", "Glass", 0.00));  
 *testItems*.add(new Item("FG Bottle | 0.5L", "js92hp13rp", "Reusable", "Plastic", 1.00));  
 *testItems*.add(new Item("ABC Can | 0.33L", "r8yz7clkz4", "Disposable", "Metal", 0.25));  
 *testItems*.add(new Item("Y Bottle | 0.5L", "447ds7u9j4", "Not Accepted", "Glass", 0.00));  
  
 // Run the simulation  
 ReceiptProcessor *rp* = simulator.runSimulation(*testItems*);  
  
 // Assertions based on the receipt processor  
 *assertEquals*(3, *rp*.getReusableCount()); // Expected reusable count  
 *assertEquals*(1, *rp*.getDisposableCount()); // Expected disposable count  
 *assertEquals*(3, *rp*.getNonAcceptedCount()); // Expected non-accepted count  
 *assertEquals*(0.25, *rp*.getDisposableTotal(), 0.01); // Expected disposable total  
 *assertEquals*(1.55, *rp*.getReusableTotal(), 0.01); // Expected reusable total  
 *assertEquals*(1.80, *rp*.getTotalForAll(), 0.01); // Expected total for all  
}

1. Test3 : das habe ich auch mit Gjiven geschrieben  
   A screen shot of a computer

   Description automatically generated

## 2.Mockito

1. testNeverCalledMethods : Dieser Test überprüft, dass bestimmte Methoden des receiptProcessor-Objekts niemals aufgerufen werden, wenn die Simulation ohne akzeptierte Artikel durchgeführt wird. Dies stellt sicher, dass unerwartete Aufrufe nicht stattfinden.
2. testGetItemByBarcode\_InvalidBarcode : In diesem Test wird geprüft, dass beim Abrufen eines Artikels mit einem ungültigen Barcode (invalid\_barcode) der Rückgabewert null ist. Dies bestätigt, dass der Artikel nicht gefunden wird, wenn der Barcode nicht existiert.
3. Test4 : Ich habe 4 bis 6 Mockto getest hier ist zb Test Nummer 4

**A screen shot of a computer

Description automatically generated**

@Test  
void test3() {  
 // Prepare test items  
 testItems.add(new Item("DE Bottle | 0.5L", "3jdwml7w52", "Reusable", "Glass", 0.30));  
 testItems.add(new Item("ABC Can | 0.33L", "r8yz7clkz4", "Disposable", "Metal", 0.25));  
 testItems.add(new Item("X Can | 0.5L", "3a6mr6o7sl", "Not Accepted", "Metal", 0.00));  
 testItems.add(new Item("FG Bottle | 1L", "8hgij9rqv5", "Reusable", "Plastic", 2.00));  
 testItems.add(new Item("Z Bottle | 0.5L", "5jrsqeg201", "Not Accepted", "Plastic", 0.00));  
 testItems.add(new Item("DE Bottle | 1L", "xvjix0xaue", "Reusable", "Glass", 0.75));  
 testItems.add(new Item("Y Bottle | 1L", "5hqoa0ean4", "Not Accepted", "Glass", 0.00));  
 testItems.add(new Item("ABC Can | 0.5L", "tmvkrw69le", "Disposable", "Metal", 0.25));  
  
 // Set up mock behavior for the receipt processor  
 *when*(receiptProcessor.getReusableCount()).thenReturn(3);  
 *when*(receiptProcessor.getDisposableCount()).thenReturn(2);  
 *when*(receiptProcessor.getNonAcceptedCount()).thenReturn(3);  
 *when*(receiptProcessor.getDisposableTotal()).thenReturn(0.50);  
 *when*(receiptProcessor.getReusableTotal()).thenReturn(3.05);  
 *when*(receiptProcessor.getTotalForAll()).thenReturn(3.55);  
  
 // Run the simulation  
 ReceiptProcessor *rp* = simulator.runSimulation(testItems);  
  
 // Assertions based on the mocked receipt processor  
 *assertEquals*(3, *rp*.getReusableCount()); // Expected reusable count  
 *assertEquals*(2, *rp*.getDisposableCount()); // Expected disposable count  
 *assertEquals*(3, *rp*.getNonAcceptedCount()); // Expected non-accepted count  
 *assertEquals*(0.50, *rp*.getDisposableTotal(), 0.01); // Expected disposable total  
 *assertEquals*(3.05, *rp*.getReusableTotal(), 0.01); // Expected reusable total  
 *assertEquals*(3.55, *rp*.getTotalForAll(), 0.01); // Expected total for all  
  
 // Verify interactions with mocks  
 *verify*(receiptProcessor, *times*(1)).getReusableCount();  
 *verify*(receiptProcessor, *times*(1)).getDisposableCount();  
 *verify*(receiptProcessor, *times*(1)).getNonAcceptedCount();  
 *verify*(receiptProcessor, *times*(1)).getDisposableTotal();  
 *verify*(receiptProcessor, *times*(1)).getReusableTotal();  
 *verify*(receiptProcessor, *times*(1)).getTotalForAll();  
}

## 3. BDD (IGiven)

1. Test Nummer 7:
   * *Dieser Test besteht einfach aus 3 Teilen, die gegeben given, when und then. in wenn wir unser Testelement deklarieren, in wenn wir den Simulator ausführen und in dann überprüfen wir unsere Werte.*

A screen shot of a computer code

Description automatically generated

@Test  
@DisplayName("Bottle deposit machine simulation test 7")  
public void bottle\_deposit\_machine\_simulation\_test\_7() {  
 given().the\_machine\_is\_ready\_with\_test\_items\_for\_test\_7();  
 when().the\_simulation\_is\_run();  
 then().all\_counts\_and\_totals\_should\_be\_correct\_for\_test\_7(2, 2, 3, 0.50, 1.75, 2.25);  
}

1. Test Nummer 8**:**
   * *Es ist das gleiche wie 7, aber mit anderen Werten-  
       
     so sieht das ganze test aus :*
2. @Test  
   @DisplayName("Bottle deposit machine simulation test 8")  
   public void bottle\_deposit\_machine\_simulation\_test\_8() {  
    given().the\_machine\_is\_ready\_with\_test\_items\_for\_test\_8();  
    when().the\_simulation\_is\_run();  
    then().all\_counts\_and\_totals\_should\_be\_correct\_for\_test\_8(3, 1, 4, 0.25, 3.05, 3.30);  
   }
3. public TestStage the\_simulation\_is\_run() {  
    resultReceiptProcessor = simulator.runSimulation(testItems);  
    return self();  
   }
4. public TestStage the\_machine\_is\_ready\_with\_test\_items\_for\_test\_8() {  
    // Setup  
    ReceiptProcessor *receiptProcessor* = new ReceiptProcessor();  
    Display *display* = new Display();  
    bottleDepositMachine = new BottleDepositMachine(*display*, *receiptProcessor*);  
    simulator = new Simulator(bottleDepositMachine);  
    testItems = new ArrayList<>();  
     
    testItems.add(new Item("Y Bottle | 1L", "5hqoa0ean4", "Not Accepted", "Glass", 0.00));  
    testItems.add(new Item("ABC Can | 0.5L", "tmvkrw69le", "Disposable", "Metal", 0.25));  
    testItems.add(new Item("DE Bottle | 0.5L", "3jdwml7w52", "Reusable", "Glass", 0.30));  
    testItems.add(new Item("Z Bottle | 1L", "pv4og90ymi", "Not Accepted", "Plastic", 0.00));  
    testItems.add(new Item("FG Bottle | 1L", "8hgij9rqv5", "Reusable", "Plastic", 2.00));  
    testItems.add(new Item("DE Bottle | 1L", "xvjix0xaue", "Reusable", "Glass", 0.75));  
    testItems.add(new Item("X Can | 0.5L", "3a6mr6o7sl", "Not Accepted", "Metal", 0.00));  
    testItems.add(new Item("Y Bottle | 0.33L", "ha3pven757", "Not Accepted", "Glass", 0.00));  
     
    return self();  
   }  
     
   public TestStage all\_counts\_and\_totals\_should\_be\_correct\_for\_test\_8(int *expectedReusableCount*, int *expectedDisposableCount*,  
    int *expectedNonAcceptedCount*, double *expectedDisposableTotal*,  
    double *expectedReusableTotal*, double *expectedTotalForAll*) {  
    *assertNotNull*(resultReceiptProcessor, "ReceiptProcessor should not be null");  
     
    *assertEquals*(*expectedReusableCount*, resultReceiptProcessor.getReusableCount(), "Reusable count does not match");  
    *assertEquals*(*expectedDisposableCount*, resultReceiptProcessor.getDisposableCount(), "Disposable count does not match");  
    *assertEquals*(*expectedNonAcceptedCount*, resultReceiptProcessor.getNonAcceptedCount(), "Non-accepted count does not match");  
     
    *assertEquals*(*expectedDisposableTotal*, resultReceiptProcessor.getDisposableTotal(), 0.01, "Disposable total does not match");  
    *assertEquals*(*expectedReusableTotal*, resultReceiptProcessor.getReusableTotal(), 0.01, "Reusable total does not match");  
    *assertEquals*(*expectedTotalForAll*, resultReceiptProcessor.getTotalForAll(), 0.01, "Total for all does not match");  
     
    return self();  
   }