Test-Driven Development – Is it as good as it seems?

Exercise

Design goals

- How to teach TDD in **30 minutes**?
  - tutorials usually take much longer
  - live-programming in 30 minutes: not a good idea

- **Main goal** of the exercise:
  - to get you into the rhythm of the “red, green, refactor” cycle

The task

- Create a class “**Euro**” that represents the currency (€)
- We will add requirements as we go through the exercise

- **Note**: This exercise may seem trivial.
  - The difficulty is not in the solution of the problem
  - This exercise will give you a brief glimpse into how to apply TDD

Project setup

```
import org.junit.Test;
import static junit.framework.Assert.*;

/*
    Euro class - to do:
 */

public class EuroTest {

}
```

A new requirement appears!

We want to create Euro objects and get string representations for them, e.g. “EUR 2.00”

```
import org.junit.Test;
import static junit.framework.Assert.*;

/*
    Euro class - to do:
        - convert to string
 */

public class EuroTest {

}
```

Write a new test

```
import org.junit.Test;
import static junit.framework.Assert.*;

/*
    Euro class - to do:
        - convert to string
 */

public class EuroTest {
    @Test
    public void testToString() {
        assertEquals("EUR 2.00", new Euro(2).toString());
    }
}
```

Run it ...

Cannot find symbol class Euro

Make the code compile

```
@Test
public void testToString() {
    assertEquals("EUR 2.00", new Euro(2).toString());
}
```

```
public class Euro {
    public Euro(int amount) {
    }

    @Override
    public String toString() {
        return null;
    }
}
```

Run it again ...

junit.framework.ComparisonFailure:
Expected: EUR 2.00
Actual: <null>

Add a tiny bit of production code

```
@Test
public void testToString() {
    assertEquals("EUR 2.00", new Euro(2).toString());
}
```

```
public class Euro {
    public Euro(int amount) {
    }

    @Override
    public String toString() {
        return null;
    }
}
```

Add a tiny bit of production code

```
@Test
public void testToString() {
    assertEquals("EUR 2.00", new Euro(2).toString());
}
```

```
public class Euro {
    public Euro(int amount) {
    }

    @Override
    public String toString() {
        return "EUR 2.00";
    }
}
```

Run once more ...

Passed!

What about fractions of Euros (e.g. Cents)? What about other values than “2 €”?

```
import org.junit.Test;
import static junit.framework.Assert.*;

/*
    Euro class - to do:
        - convert to string
 */

public class EuroTest {
    @Test
    public void testToString() {
        assertEquals("EUR 2.00", new Euro(2).toString());
        assertEquals("EUR 7.50", new Euro(7.50).toString());
    }
}
```

Run it ...

Cannot find symbol constructor Euro(double)

Make the code compile

```
@Test
public void testToString() {
    assertEquals("EUR 2.00", new Euro(2).toString());
    assertEquals("EUR 7.50", new Euro(7.50).toString());
}
```

```
public class Euro {
    public Euro(double amount) {
    }

    @Override
    public String toString() {
        return "EUR 2.00";
    }
}
```

junit.framework.ComparisonFailure:
Expected: EUR 7.50
Actual: EUR 2.00
at EuroTest.testToString

Add more production code

```
@Test
public void testToString() {
    assertEquals("EUR 2.00", new Euro(2).toString());
    assertEquals("EUR 7.50", new Euro(7.50).toString());
}
```

```
public class Euro {
    private double amount;

    public Euro(double amount) {
        this.amount = amount;
    }

    @Override
    public String toString() {
        return String.format("EUR %.2f", amount);
    }
}
```

Run ...

Passed!

A new requirement appears ...

We want to check Euro objects for equality

```
/*
    Euro class - to do:
        X convert to string
        - equality
 */
```

```
public class EuroTest {
    @Test
    public void testToString() {
        assertEquals("EUR 2.00", new Euro(2).toString());
        assertEquals("EUR 7.50", new Euro(7.50).toString());
    }

    @Test
    public void testEquality() {
        Euro sevenFifty = new Euro(7.50);
        Euro sevenFiftyToo = new Euro(7.50);
        assertTrue(sevenFifty.equals(sevenFiftyToo));
    }
}
```

Add a tiny bit of production code

```
@Test
public void testEquality() {
    Euro sevenFifty = new Euro(7.50);
    Euro sevenFiftyToo = new Euro(7.50);
    assertTrue(sevenFifty.equals(sevenFiftyToo));
}
```

```
public class Euro {
    private double amount;

    public Euro(double amount) {
        this.amount = amount;
    }

    @Override
    public String toString() {
        return String.format("EUR %.2f", amount);
    }

    @Override
    public boolean equals(Object o) {
        return true;
    }
}
```

Run ...

Passed!

We should also check for inequality … :)

```
public class EuroTest {
    @Test
    public void testToString() {
        assertEquals("EUR 2.00", new Euro(2).toString());
        assertEquals("EUR 7.50", new Euro(7.50).toString());
    }

    @Test
    public void testEquality() {
        Euro sevenFifty = new Euro(7.50);
        Euro sevenFiftyToo = new Euro(7.50);
        assertTrue(sevenFifty.equals(sevenFiftyToo));
    }

    @Test
    public void testInequality() {
        Euro sevenEuros = new Euro(7);
        Euro threeEuros = new Euro(3);
        assertFalse(sevenEuros.equals(threeEuros));
    }
}
```

junit.framework.AssertionFailedError
at EuroTest.testInequality

Let’s do this properly…

```java
@Test
public void testInequality() {
    Euro sevenEuros = new Euro(7);
    Euro threeEuros = new Euro(3);
    assertFalse(sevenEuros.equals(threeEuros));
}
```

```java
public class Euro {
    protected double amount;

    public Euro(double amount) {
        this.amount = amount;
    }

    @Override
    public String toString() {
        return String.format("EUR %.2f", amount);
    }

    @Override
    public boolean equals(Object o) {
        return (o instanceof Euro) && amount == ((Euro) o).amount;
    }
}
```

Run ...

Passed!

A new requirement appears ...

We want to be able to subtract Euros from each other

```
/*
    Euro class - to do:
        X convert to string
        X equality
        - subtraction
 */

public class EuroTest {

    /* ... */

    @Test
    public void testSubtraction() {
        assertEquals(new Euro(1), new Euro(3).minus(new Euro(2)));
        assertEquals(new Euro(2), new Euro(5).minus(new Euro(3)));
    }
}
```

Write production code

```
@Test
public void testSubtraction() {
    assertEquals(new Euro(1), new Euro(3).minus(new Euro(2)));
    assertEquals(new Euro(2), new Euro(5).minus(new Euro(3)));
}
```

```
public class Euro {
    protected double amount;

    public Euro(double amount) {
        this.amount = amount;
    }

    @Override
    public String toString() {
        return String.format("EUR %.2f", amount);
    }

    @Override
    public boolean equals(Object o) {
        return (o instanceof Euro) && amount == ((Euro) o).amount;
    }

    public Euro minus(Euro subtrahend) {
        return new Euro(amount - subtrahend.amount);
    }
}
```

Run the tests ...

Passed!

We have just finished a book on numerical computing...

"One should never use floats or doubles to store currencies" -
Example: `1.03 - 0.42 == 0.6100000000000001 != 0.61`

Improve numeric safety!

```
/*
    Euro class - to do:
        X convert to string
        X equality
        X subtraction
        - numeric safety?
 */

public class EuroTest {

    /* ... */

  @Test
    public void testNumericSafety() {
        assertEquals(new Euro(0.61), new Euro(1.03).minus(new Euro(0.42)));
    }
}
```

junit.framework.AssertionFailedError:
Expected: EUR 0.61
Actual: EUR 0.6100000000000001

How to fix this numeric problem?

Solution: Store the amount in Cents,
e.g. 7.50 € == 750 Cents

Refactor amount from **double** to **int**

```
public class Euro {
    protected int amount;

    public Euro(double amount) {
        this.amount = (int) (amount * 100.0);
    }

    @Override
    public String toString() {
        return String.format("EUR %.2f", (double) amount / 100.0);
    }

    @Override
    public boolean equals(Object o) {
        return (o instanceof Euro) && amount == ((Euro) o).amount;
    }

    public Euro minus(Euro subtrahend) {
        return new Euro(amount - subtrahend.amount);
    }
}
```

Run it ...

junit.framework.AssertionFailedError:
Expected: EUR 1.00
Actual: EUR 100.00
at EuroTest.testSubtraction

junit.framework.AssertionFailedError:
Expected: EUR 0.61
Actual: EUR 61.00
at EuroTest.testNumericSafety

What happened?

junit.framework.AssertionFailedError:
Expected: EUR 1.00
Actual: EUR 100.00
at EuroTest.testSubtraction

junit.framework.AssertionFailedError:
Expected: EUR 0.61
Actual: EUR 61.00
at EuroTest.testNumericSafety

What happened:

We broke existing functionality (subtraction)
… but our test suite warned us!

(That's good)

Fix subtraction ...

```
public class Euro {
    protected int amount;

    public Euro(double amount) {
        this.amount = (int) (amount * 100.0);
    }

    @Override
    public String toString() {
        return String.format("EUR %.2f", (double) amount / 100.0);
    }

    @Override
    public boolean equals(Object o) {
        return (o instanceof Euro) && amount == ((Euro) o).amount;
    }

    public Euro minus(Euro subtrahend) {
        Euro result = new Euro(0);
        result.amount = amount - subtrahend.amount;
        return result;
    }
}
```

Run ...

Passed!

Great, now let's refactor ...

```
public class Euro {
    protected int amount;

    private static final double CENTS_PER_EURO = 100;

    public Euro(double amount) {
        this.amount = (int) (amount * CENTS_PER_EURO);
    }

    @Override
    public String toString() {
        return String.format("EUR %.2f", (double) amount / CENTS_PER_EURO);
    }

    @Override
    public boolean equals(Object o) {
        return (o instanceof Euro) && amount == ((Euro) o).amount;
    }

    public Euro minus(Euro subtrahend) {
        Euro result = new Euro(0);
        result.amount = amount - subtrahend.amount;
        return result;
    }
}
```

Run ...

Still passes!

Final test code

```
public class EuroTest {
    @Test
    public void testToString() {
        assertEquals("EUR 2.00", new Euro(2).toString());
        assertEquals("EUR 7.50", new Euro(7.50).toString());
    }

    @Test
    public void testEquality() {
        Euro sevenFifty = new Euro(7.50);
        Euro sevenFiftyToo = new Euro(7.50);
        assertTrue(sevenFifty.equals(sevenFiftyToo));
    }

    @Test
    public void testInequality() {
        Euro sevenEuros = new Euro(7);
        Euro threeEuros = new Euro(3);
        assertFalse(sevenEuros.equals(threeEuros));
    }

    @Test
    public void testSubtraction() {
        Euro twoEuros = new Euro(2);
        Euro threeEuros = new Euro(3);
        assertEquals(new Euro(1), threeEuros.minus(twoEuros));
        assertEquals(new Euro(2), new Euro(5).minus(new Euro(3)));
    }

    @Test
    public void testNumericSafety() {
        assertEquals(new Euro(0.61), new Euro(1.03).minus(new Euro(0.42)));
    }
}
```

Final production code

```
public class Euro {
    protected int amount;

    private static final double CENTS_PER_EURO = 100;

    public Euro(double amount) {
        this.amount = (int) (amount * CENTS_PER_EURO);
    }

    @Override
    public String toString() {
        return String.format("EUR %.2f", (double) amount / CENTS_PER_EURO);
    }

    @Override
    public boolean equals(Object o) {
        return (o instanceof Euro) && amount == ((Euro) o).amount;
    }

    public Euro minus(Euro subtrahend) {
        Euro result = new Euro(0);
        result.amount = amount - subtrahend.amount;
        return result;
    }
}
```

Discussion