Selbststudium OOP8 & ALG3

Aufgaben

Kapitel 7.3

1. zu bearbeitende Aufgaben: 7.1 bis 7.11

7.1:

done

7.2:

done

7.3:

done

7.4:

done

7.5:

Rating mit 0 funktioniert, obwohl nur 1 bis 5 erlaubt wäre.

7.6:

Rating mit 0 funktioniert, obwohl nur 1 bis 5 erlaubt wäre.

7.7:

done

7.8:

done

7.9:

Nein, noSuchElementException

7.10:

```
private boolean ratingInvalid(int rating)
{
   return rating <= 0 || rating > 5;
}

public Comment findMostHelpfulComment()
{
   Iterator<Comment> it = comments.iterator();
   if (it.hasNext()) {
      // [code]
   }
   Else {
      return null;
   }
}
```

Nein, es muss wieder alles getestet werden.

7.11:

Positiv: 7.1, 7.2, 7.3, 7.4, 7.7, 7.8, 7.9

Negativ: 7.4, 7.6

Kapitel 7.4

2. zu bearbeitende Aufgaben: 7.12 bis 7.19

7.12:

done

7.13:

done

7.14:

setUp() und tearDown()

7.15:

```
@Test
public void testTwoComments()
{
   SalesItem salesIte2 = new SalesItem("article1", 123);
   assertEquals(true, salesIte2.addComment("autor1", "comm1", 1));
   assertEquals(false, salesIte2.addComment("autor1", "comm2", 2));
   assertEquals(1, salesIte2.getNumberOfComments());
}
```

7.16:

```
@Test
public void testIllegalRating()
{
   SalesItem salesIte1 = new SalesItem("Java For Idiots", 19900);
   assertEquals(false, salesIte1.addComment("autor1", "comm1", 0));
   assertEquals(false, salesIte1.addComment("autor2", "comm2", 6));
}
```

7.17:

Mittels „Show Source“ kann zur fehlerhaften Zeile gesprungen werden.

7.18:

```
@Test
public void testAuthor()
{
   Comment c1 = new Comment("Author1", "comment1", 1);
   assertEquals("Author1", c1.getAuthor());
}

@Test
public void testRating()
{
   Comment c2 = new Comment("Author2", "comment2", 2);
   assertEquals(2, c2.getRating());
}

@Test
public void testUpvote()
{
   Comment cUp = new Comment("Author", "comment", 2);
   cUp.upvote();
   assertEquals(1, cUp.getVoteCount());
}

@Test
public void testDownvote()
{
   Comment cDown = new Comment("Author", "comment", 2);
   cDown.downvote();
   assertEquals(-1, cDown.getVoteCount());
}
```

7.19:

```
@Test
public void testfindMostHelpfulCommentNull()
{
   SalesItem salesIte3 = new SalesItem("article1", 123);
   assertEquals(null, salesIte3.findMostHelpfulComment());
}
```

Kapitel 7.6

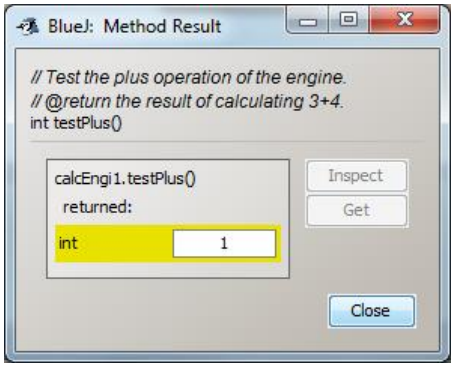
3. zu bearbeitende Aufgaben: 7.21 bis 7.23

7.21:

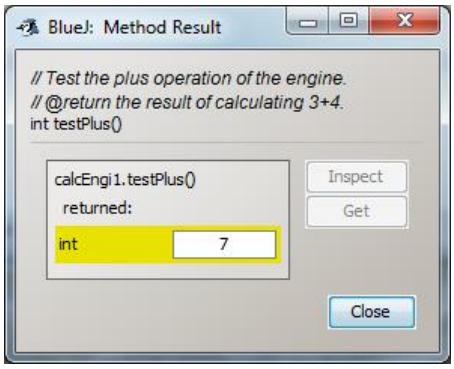
```
Testing the addition operation.
The result is: 7
Testing the subtraction operation.
The result is: 5
All tests passed.
```

⇨ *Es ist nicht bekannt was gerechnet wurde, somit ob das Ergebnis überhaupt richtig ist.*

7.22:



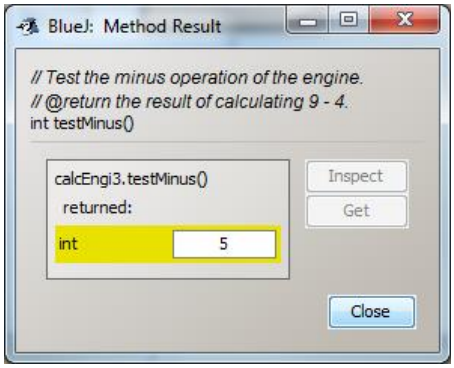
Es ist ein anderes Ergebnis, eigentlich wäre 3 + 4 = 7 und nicht = 1.

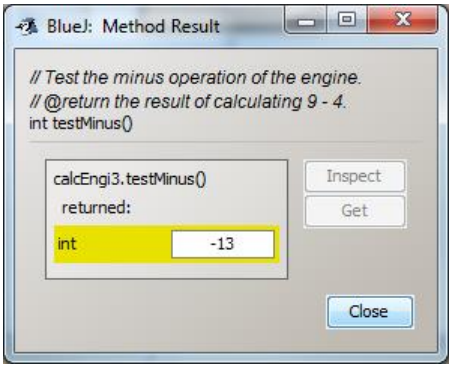


Beim zweiten Aufruf wird das richtige Resultat (7) zurück geliefert.

Es sollte immer das gleiche Resultat geben wenn man diese Methode aufruft.

7.23:





Nein, es gibt ebenfalls unterschiedliche Resultate zurück.

Kapitel 7.7

4. zu bearbeitende Aufgaben: 7.24 bis 7.27

7.24:

Ist vergleichbar mit der Plus Methode.

7.25:

Method called	*displayValue*	*leftOperand*	*previousOperator*
initial state	*0*	*0*	*´ ´*
Clear	*0*	*0*	*´ ´*
numberPressed(3)	*3*	*0*	*´ ´*
plus	*0*	*3*	*´+´*
numberPressed(4)	*4*	*3*	*´+´*
equals()	*7*	*0*	*´+´*

7.26:

Nein, es wird nur überprüft ob es ein + war, alle anderen werden als – (Subtraktion) interpretiert.

7.27:

- equals() überprüft nur das + Zeichen

- clear() löscht nicht alle Felder

```
public void equals() {
  if(previousOperator == '+') {
    displayValue = leftOperand + displayValue;
  }
  if(previousOperator == '-') {
    displayValue = leftOperand - displayValue;
  }
  else {
    //do nothing
  }
  leftOperand = 0;
}

public void clear()
{
  displayValue     = 0;
  leftOperand      = 0;
  previousOperator = ' ';
}
```

Kapitel 7.8

5. zu bearbeitende Aufgaben: 7.30 bis 7.33

7.30:

done

7.31:

Nein, es zeigt die jeweiligen Resultate an, ohne wirklich auf Fehler zu überprüfen.

7.32:

Für manuelles Debuggen ist es optimal in diesem Umfang, tendiert jedoch in die Richtung „zu viel Output“.

7.33:

Debug Befehle müssen nachträglich wieder entfernt/auskommentiert werden.

Debug Befehle liefern nur Informationen, sie überprüfen dabei nicht zwingend die Richtigkeit

Für „schnelles“ Debugen sind Debug-Befehle hilfreich.

Kapitel 7.9

6. zu bearbeitende Aufgabe: 7.34

7.34:

done

Kapitel 7.10

7. zu bearbeitende Aufgabe: 7.36

7.36:

```
//Auszug aus Testklasse HackersTests
@Test
public void testPlus()
{
   // Make sure the engine is in a valid starting state.
   engine.clear();
   // Simulate the key presses: 3 + 4 =
   engine.numberPressed(3);
   engine.plus();
   engine.numberPressed(4);
   engine.equals();
   // Return the result, which should be 7.
   assertEquals(7, engine.getDisplayValue());
}
```

Optional zu Kapitel 7.11

8. zu bearbeitende Aufgabe: 7.37

7.37:

Klasse Brick:

- *getSurfaceArea(): side1 sollte side2 sein bei der Summe*
- *getWeight(): 1000 sollte 1000d oder 1000.0 sein*

Klasse Palette:

- *getHeight(): % ist falsch*
- *getWeight(): + baseWeight fehlt*

Vorgehen:

- *Programmlogik erkennen / was soll das Programm tun?*
- *Fehler mit „code walkthrough" finden*
- *Debbuggen*

Testen

9. Schreiben Sie eine Testspezifikation für ein Programm, das das Volumen V eines Quaders bei gegebenen drei Seiten a, b und c berechnet. Verwenden Sie dazu folgende Tabelle:

	Testfall	Eingabe	Ausgabe	Bestanden?
Normalfälle	Nr. 1	*a=1, b=2, c=3*	*V=6*	
	Nr. 2	*a=5, b=5, c=2*	*V=50*	
	Nr. 3	*a=500, b=321, c=123*	*V=19'741'500*	
Zulässige Spezialfälle	Nr. 4	*a=1, b=1, c=1*	*V=1*	
	Nr. 5	*a=5, b=5, c=5*	*V=125*	
	Nr. 6	*a=0, b=2, c=20*	*V=0 (kein Quader)*	
Unzulässige Speziallfälle	Nr. 7	*a=-5, b=10, c=5*	*V=0 (negative Zahlen)*	
	Nr. 8	*Alle Seiten sehr gross, dass es einen Overflow gibt*		
	Nr. 9	*Falls es Float sind: sehr kleine Zahlen wählen damit es gegen Null geht*		

Algorithmen

10. Implementieren Sie den Pseudocode "Türme von Hanoi" in Java.

Pseudocode:

```
moveDisks(String from, String via, String to, int n)
   Falls n == 1:
      print ("move disk from " + from + " to " + to) ;
   sonst, d.h. n > 1:
      moveDisks(from, to, via, n-1);   // A -> C -> B
      moveDisks(from, "", to, 1);      // A -> C
      moveDisks(via, from,   to, n-1); // B -> A -> C
```

Java-Implementation:

```
/**
 * Türme von Hanoi
 *
 * @author Felix Rohrer
 * @version 29.03.2013
 */
public class TowerOfHanoi
{
    // Anzahl schritte
    private int countMove;

    /**
     * Constructor for objects of class TowerOfHanoi
     */
    public TowerOfHanoi()
    {
        //nothing
    }


    /**
     * moveDisks von Quell auf den Ziel-Stapel
     * @param Quell-Stapel
     * @param Via-Stapel
     * @param Ziel-Stapel
     * @param Anzahl Scheiben die verschoben werden müssen
     */
    private void moveDisks(String from, String via, String to, int n)
    {


        // Verschieben ausführen (rekursiv)
        if (n == 1) {
            // Counter für die Anzahl Schritte erhöhen
            countMove++;
            System.out.println("Step " + countMove + ": move disk from " + from + " to " + to); //
                                                                                    // Rekursionsbasis
        }
        else {
            // obere Scheiben auf den "via-Stapel" verschieben (A -> C -> B)
            moveDisks(from, to, via, n-1);
            // Eine Scheibe auf den Ziel-Stapel verschieben (A -> C)
            moveDisks(from, via, to, 1);
            // restliche Scheiben verschieben (B -> A -> C)
            moveDisks(via, from, to, n-1);
        }
    }

    /**
     * moveTower
     * Test für moveDisks
     * @param Anzahl Scheiben
     */
    public void moveTower(int numberDiscs)
    {
        // reset countMove
        countMove = 0;

        // move Tower from "a" to "c" via "b"
        moveDisks("a", "b", "c", numberDiscs);
    }
}
```

11. Beobachten Sie im Debugger wie die "offenen" Methoden (deren Ausführung unterbrochen ist) auf dem Stack "eingefroren" werden. Verwenden Sie dazu die im BlueJ Debugger angezeigte Aufrufkette der Methodenaufrufe (Call Graph).

```
moveTower(4)

Step 1: move disk from a to b
Step 2: move disk from a to c
Step 3: move disk from b to c
Step 4: move disk from a to b
Step 5: move disk from c to a
Step 6: move disk from c to b
Step 7: move disk from a to b
Step 8: move disk from a to c
Step 9: move disk from b to c
Step 10: move disk from b to a
Step 11: move disk from c to a
Step 12: move disk from b to c
Step 13: move disk from a to b
Step 14: move disk from a to c
Step 15: move disk from b to c
```