Exercises 7.15-7.19

Answer to 7.17: Once a test fails, we can inspect the statement that caused it to fail. Eclipse will highlight that line of code. There appears to be a quick way (by right-clicking on the test case) to create a new task associated with the failure, so that other team members can look at it.

SalesItemTest

```java
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

/**
 * The test class SalesItemTest.
 *
 * @author  mik
 * @version 0.1
 */
public class SalesItemTest
{
    /**
     * Default constructor for test class SalesItemTest
     */

    @Test
    public void addCommentTwiceTest()
    {
        SalesItem salesIte1 = new SalesItem("Java for complete Idiots", 21998);
        assertEquals(true, salesIte1.addComment("James Duckling", "This book is great. I learned all my Java from it.", 4));
        assertEquals(false, salesIte1.addComment("James Duckling", "This book is great. I learned all my Java from it.", 4));
    }
    @Test
    public void negativeRatingTest()
    {
        SalesItem salesIte1 = new SalesItem("Java for complete Idiots", 21998);
        assertEquals(false, salesIte1.addComment("alice", "This book is great. I learned all my Java from it.", 0));
        assertEquals(false, salesIte1.addComment("bob", "This book is great. I learned all my Java from it.", 6));
    }



    @Test
    public void mostUseFulCommentTest()
    {
        SalesItem salesIte1 = new SalesItem("Java for complete Idiots", 21998);

        assertEquals(true, salesIte1.addComment("name", "This book is great", 4));
```

```
        assertEquals(true, salesIte1.addComment("name2", "Too simple!", 2));
        assertEquals(true, salesIte1.addComment("name3", "Why don't people switch to mono instead?", 1));
        assertEquals(true, salesIte1.addComment("name4", "Really pedagogical. I recommend this book to freshman", 5));

        salesIte1.upvoteComment(3);
        salesIte1.upvoteComment(3);
        salesIte1.upvoteComment(3);
        salesIte1.upvoteComment(1);
        salesIte1.upvoteComment(0);
        salesIte1.downvoteComment(2);

        assertEquals("name4", salesIte1.findMostHelpfulComment().getAuthor());

     }


    public SalesItemTest()
    {
    }

    /**
     * Sets up the test fixture.
     *
     * Called before every test case method.
     */
    @Before
    public void setUp()
    {
    }

    /**
     * Tears down the test fixture.
     *
     * Called after every test case method.
     */
    @After
    public void tearDown()
    {
    }

    /**
     * Test that a comment can be added, and that the comment count is correct afterwards.
     */
    @Test
    public void testAddComment()
    {
        SalesItem salesIte1 = new SalesItem("Java for complete Idiots", 21998);
        assertEquals(true, salesIte1.addComment("James Duckling", "This book is great. I learned all my Java from it.", 4));
        assertEquals(1, salesIte1.getNumberOfComments());
```

```
    }

    /**
     * Test that a comment using an illegal rating value is rejected.
     */
    @Test
    public void testIllegalRating()
    {
        SalesItem salesIte1 = new SalesItem("Java For Complete Idiots, Vol
            2", 19900);
        assertEquals(false, salesIte1.addComment("Joshua Black", "Not
            worth the money. The font is too small.", -5));
    }

    /**
     * Test that a sales item is correctly initialised (name and price).
     */
    @Test
    public void testInit()
    {
        SalesItem salesIte1 = new SalesItem("test name", 1000);
        assertEquals("test name", salesIte1.getName());
        assertEquals(1000, salesIte1.getPrice());
    }
}
```

CommentTest

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

import org.junit.Test;


public class CommentTest {

    @Test
    public void commentStorageTest()
    {
        Comment comment1 = new Comment("James Duckling", "This book is
            great. I learned all my Java from it.", 4);

        assertEquals(4, comment1.getRating());
        assertEquals("James Duckling", comment1.getAuthor());
    }

    @Test
    public void upvoteDownvoteTest()
    {
        Comment comment1 = new Comment("Henry Higgins", "It is marvelous
            indeed", 5);

        comment1.upvote();
        assertEquals(1, comment1.getVoteCount());
        comment1.downvote();
        assertEquals(0, comment1.getVoteCount());
    }
```

```
}
```

Exercise 'insertion sort'

Sort class

```
import java.util.Arrays;

/**
 * A collection of sorting algorithms for arrays of integers.
 *
 * @author Stefan Nilsson
 * @version 2009-10-22
 */
public class Sort
{
    private static final boolean DEBUGGING = false;

    private void debugPrint(String s) {
        if (DEBUGGING) {
            System.err.println("Sort: " + s);
        }
    }

    public int[] insertionSort(int[] v)
    {
        for (int j = 1; j < v.length; j++)
        {
            int key = v[j];
            int i = j-1;

            while(i>= 0 && v[i] > key)
            {
                v[i+1] = v[i];
                i--;
            }
            v[i+1] = key;
        }

        return v;
    }


    /**
     * Sort the elements in ascending order.
     * This algorithm has time complexity Theta(n*n), where n is
     * the length of the array.
     *
     * @param v     An array of integers.
     * @return      The same array sorted in ascending order.
     */
    public void selectionSort(int[] v) {
        int n = v.length;
        debugPrint("selection sort, n=" + n);
        for (int i = 0; i < n - 1; i++) {
            // find index m of min element in v[i..n-1]
```

```
            int m = i;
            for (int j = i + 1; j < n; j++) {
                if (v[j] < v[m]) {
                    m = j;
                }
            }
            if (DEBUGGING && n < 10) {
                debugPrint(Arrays.toString(v));
                debugPrint("i=" + i + ", m=" + m);
            }
            // swap v[i] and v[m]
            int temp = v[i];
            v[i] = v[m];
            v[m] = temp;
        }
    }
}
```

SortTest class

```
import java.util.Arrays;
import java.util.Random;

/**
 * Test class for Sort.
 *
 * @author  Stefan Nilsson
 * @version 2011-10-23
 */
public class SortTest extends junit.framework.TestCase
{
    /**
     * t: test case,  s: expected solution.
     */
    private int[] t0, s0, t1, s1, t2, s2, t7, s7;

    /**
     * Big array of random numbers.
     * tr: test case, sr: expected solution.
     * R_SIZE is the size of the array.
     */
    private static final int R_SIZE = 10000;
    private int[] tr, sr;

    private Random rand;

    /**
     * Constructs a new test case.
     */
    public SortTest() {
        rand = new Random();
    }

    /**
     * Sets up the test fixture.
     * Called before every test case method.
```

```
     */
    protected void setUp() {
        t0 = new int[0];
        s0 = new int[0];

        t1 = new int[] {1};
        s1 = new int[] {1};

        t2 = new int[] {2, 1};
        s2 = new int[] {1, 2};

        t7 = new int[] {9, 5, 2, 7, 1, 6, 6};
        s7 = new int[] {1, 2, 5, 6, 6, 7, 9};

        tr = new int[R_SIZE];
        sr = new int[R_SIZE];
        for (int i = 0; i < R_SIZE; i++) {
            tr[i] = sr[i] = rand.nextInt();
        }
        Arrays.sort(sr);
    }

    /**
     * Tears down the test fixture.
     * Called after every test case method.
     */
    protected void tearDown() {
    }

    public void testSelectionSort() {
        Sort sort = new Sort();

        sort.selectionSort(t0);
        assertTrue(Arrays.equals(t0, s0));

        sort.selectionSort(t1);
        assertTrue(Arrays.equals(t1, s1));

        sort.selectionSort(t2);
        assertTrue(Arrays.equals(t2, s2));

        sort.selectionSort(t7);
        assertTrue(Arrays.equals(t7, s7));

        sort.selectionSort(tr);
        assertTrue(Arrays.equals(tr, sr));
    }

    public void testInsertionSort()
    {
        Sort sort = new Sort();

        sort.insertionSort(t0);
        assertTrue(Arrays.equals(t0, s0));

        sort.insertionSort(t1);
```

```
        assertTrue(Arrays.equals(t1, s1));

        sort.insertionSort(t2);
        assertTrue(Arrays.equals(t2, s2));

        sort.insertionSort(t7);
        assertTrue(Arrays.equals(t7, s7));

        sort.insertionSort(tr);
        assertTrue(Arrays.equals(tr, sr));
    }
}
```

Exercise 'reverse order' of a vector

The algorithm that is to be described below work as following: *Given that an array contains n elements, we take the first item and replace it with the last. Later on, we take the second item and replace it with the second last and so on. We continue this procedure until will reach* $\lfloor n/2 \rfloor - 1$.

Algorithm 1: Reverse order of an integer array

input : An array A of n integers
output: An array of integers in reveresed order

for $i \leftarrow 0$ **to** $\lfloor n/2 \rfloor - 1$ **do**
    `Swap` ($A[i]$, $A[n-i]$)
return A

Exercise 'order according to Big Oh'

$$\begin{aligned} n + 100 \\ n \log(n) \\ n^{1.5} \\ 2^n \\ 10^n \end{aligned}$$

Exercise 'Big Oh'

Paul Bachmann's O-notation is defined as following:

$$f(n) = O(g(n)) \qquad \forall n \tag{1}$$

which means that there is a constant C such that:

$$|f(n)| \leq C|g(n)| \qquad \forall n \tag{2}$$

Big Omega notation is defined as

$$f(n) = \Omega(g(n)) \quad \iff \quad |f(n)| \geq C|g(n)| \quad \text{for some } C > 0 \tag{3}$$

- $n(n+1)/2 = O(n^3)$ is true because $n(n+1)/2 = 0.5n^2 + 0.5n$ which grows slower than n^3. Using definition, we can always pick a constant C to make it work.
- $n(n+1)/2 = O(n^2)$ is also true, similar to the reason above. When $n \to \infty$, only the highest degree term will matter. Again, we can always pick a constant to make this relationship valid.
- $n(n+1)/2 = \Theta(n^3)$ isn't true because n^3 is not the lower bound, i.e. $n(n+1)/2 \neq \Omega(n^3)$
- $n(n+1)/2 = \Omega(n)$ is true as n grows slower than $0.5n^2 + 0.5n$

Exercise 'time complexity of algorithm'

- The time complexity is $O(n^2)$
- There is no 'good' case nor 'bad' case. The algorithm will have iterate the same number of times for a given size of n. Thus, $\Omega(f(n))$ of this algorithm is the same as the Big Oh.

Algorithm 2: Returns partial sums of a given array

---

input : An array A of n integers
output: An array B of partial sums of A

for $i \leftarrow 1$ **to** $n-1$ **do**
 $B[i] = B[i-1] + A[i]$
return B

This algorithm has time complexity $O(n)$.

Exercise 'function that is neither Oh nor Omega

For example,

$$f(x) = x^2 \times |sin(x)|$$