Hikari Sophia Stölzle 570683
Pavel Tsvyatkov 559632

04. June 2019

LAB REPORT NO. 7

JUnit

PROF. DR. WEBER-WULFF / INFORMATIK 1 GROUP 1 / ROOM C579 / 12:15 - 13:45

Introduction

This weeks laboratory acquainted us dealing with test cases and writing assertions.

The given diary-testing-junit project simulates a system for keeping track of appointments on specific days and hours. Per hour one single appointment can be taken starting at 9 a.m. with the last appointment going from 5 p.m. to 6 p.m.

Index

Hikari Sophia Stölzle 570683
Pavel Tsvyatkov 559632

04. June 2019

Assignments

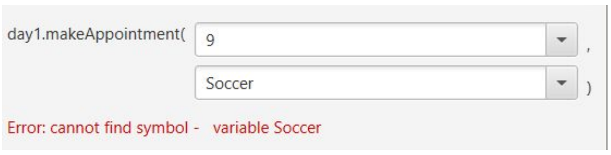
1 Create a test method in DayTest to check that findSpace returns the value of 10 for a one-hour appointment, if a day already has a single one-hour appointment at 9 a.m.

Use makeAppointment for the first appointment and findSpace for the second appointment. Specify assertions for the results of both calls.

Before we started with the exercises, we discussed for a while about doing tests when using BlueJ. We also took a look at each other's pre-lab and saw that we had similar examples. We opened the diary in BlueJ and started checking and playing around with it. We were only supposed to develop test cases and then describe the errors that occur, so we made sure to understand how the program works exactly before we moved forward with the exercises.

First we created a new *Day* object named day1 and looked at the available methods.

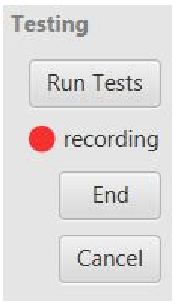
Our first attempt to make an appointment failed because we entered a String instead of the required parameter which is of type Appointment.


day1.makeAppointment(9 , Soccer)

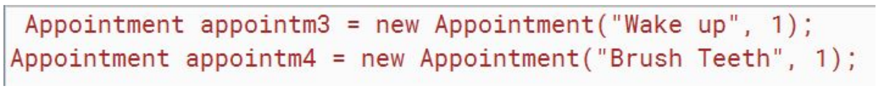
Error: cannot find symbol - variable Soccer

So we closed the window and created an Appointment object which had "Soccer" as *description* and 1 as duration in *int*.

We created a new test method named *testFindSpaceTen* by choosing *Create Test Method..* in the option menu of DayTest.

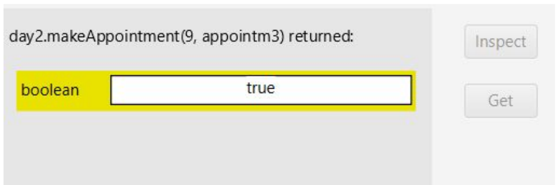


After we made sure that BlueJ started recording we created a new *Day* object and two appointments.

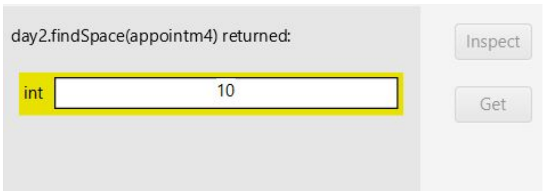


```
 Appointment appointm3 = new Appointment("Wake up", 1);
Appointment appointm4 = new Appointment("Brush Teeth", 1);
```

First we made an appointment at 9 a.m. and received a *Method Result* window in which we could decide what the result should be equal to. Since 9 was a valid time for an appointment we entered *true*.
At first we made a mistake because we thought that we only have the option to assert equals to true or false. We checked the methods that were already part of the DayTest class and discussed about it. We remembered that we spoke in class that we can do tests by either letting BlueJ record them for us or we could manually write them.
So we repeated the same procedures with *findSpace* for *appointm4*, typed in 10 as assertion and ended the recording by clicking End.

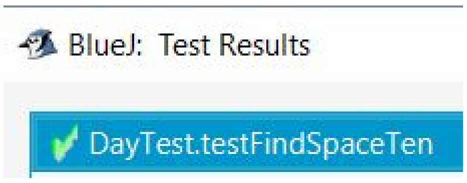

day2.makeAppointment(9, appointm3) returned:

boolean true


day2.findSpace(appointm4) returned:

int 10

Now the new test case method was automatically added to *DayTest* and we could invoke it.
In order to see the result we opened the Test Results window and received a green tick, which showed us that we did it correctly.


BlueJ: Test Results

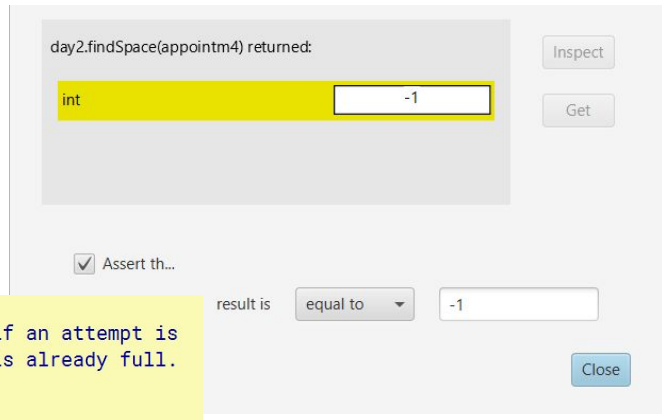
✓ DayTest.testFindSpaceTen

estimated time used: **10** minutes

2 Create a test to check that findSpace returns a value of -1, if an attempt is made to find an appointment in a day that is already full.

We talked about the exercise and agreed to create a single appointment that lasts for 9 hours, so that it fills up our day. First we created a new test method and called it *testNoSpaceLeft*.
We created a new day object, a 9 hour appointment called "School" and another appointment "After School" with duration of 1 hour. We made an appointment at 9 a.m for "School", which returned true, and after that we called findSpace( ) to search for available time for the "After School" appointment.

BlueJ returned a value of -1, which was correct. We tested if we can set the assertion to equal -1 from here and it worked.

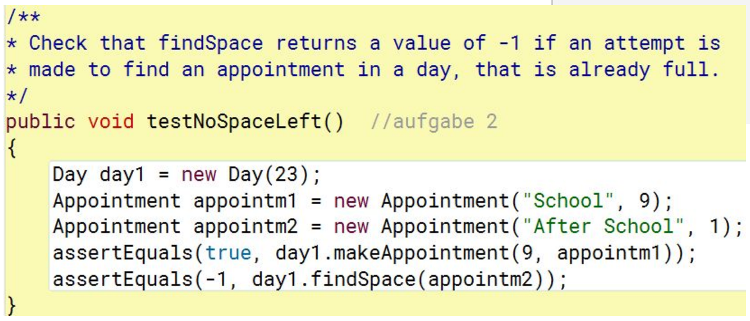

day2.findSpace(appointm4) returned:

int -1

✓ Assert th...

result is equal to -1

We opened the DayTest class and our test method looked like that.


```
/**
* Check that findSpace returns a value of -1 if an attempt is
* made to find an appointment in a day, that is already full.
*/
public void testNoSpaceLeft()  //aufgabe 2
{
    Day day1 = new Day(23);
    Appointment appointm1 = new Appointment("School", 9);
    Appointment appointm2 = new Appointment("After School", 1);
    assertEquals(true, day1.makeAppointment(9, appointm1));
    assertEquals(-1, day1.findSpace(appointm2));
}
```

Hikari Sophia Stölzle 570683
Pavel Tsvyatkov 559632

04. June 2019

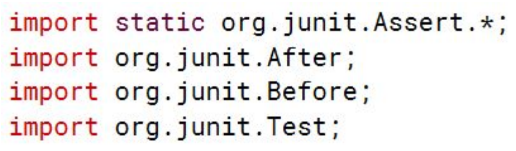
estimated time used: 4 minutes

3 Create a test class that has Appointment as its reference class

Record, separate test methods within it, that check that the description and duration fields of an Appointment object are initialized correctly following its creation.

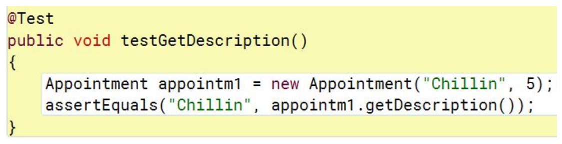
We selected *Create Test Class* from the pop up menu of the Appointment class and it automatically appeared with the name AppointmentTest.

Interesting to note is that there were already some classes imported from the java library by default which were not imported in the DayTest class.

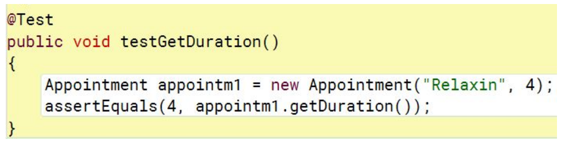


```
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
```

First we tested the basic functionality of *getDescription* and *getDuration*. For each of the test methods we created a new appointment with a description as a String and the duration as an int.

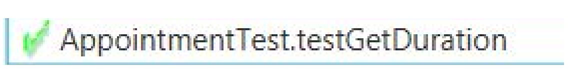


```
@Test
public void testGetDescription()
{
    Appointment appointm1 = new Appointment("Chillin", 5);
    assertEquals("Chillin", appointm1.getDescription());
}
```

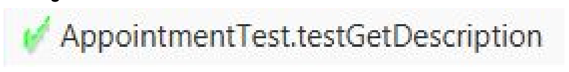


```
@Test
public void testGetDuration()
{
    Appointment appointm1 = new Appointment("Relaxin", 4);
    assertEquals(4, appointm1.getDuration());
}
```

Testing both methods via Test All option in the pop up menu of the test class, we received a green tick next to the tests. They both worked.

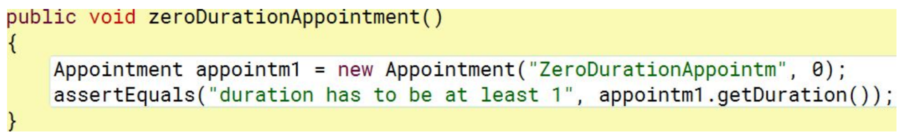


✓ AppointmentTest.testGetDuration

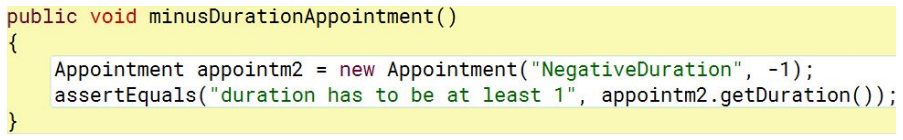


✓ AppointmentTest.testGetDescription

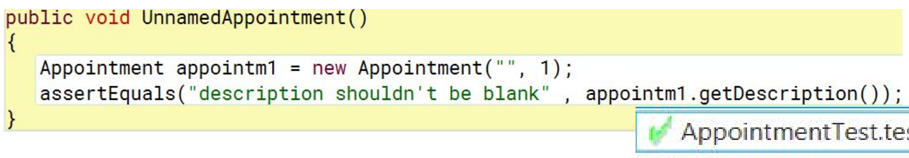
Later we added some other test methods which showed that especially the duration method needed some changes. We could select any integer number for the duration which could cause errors when using the Diary. Also we noticed that an empty String was possible as description which should be prevented in the future, because after calling showAppointments(), then it would look like the hour we added the appointment to is free, while there actually is an appointment.



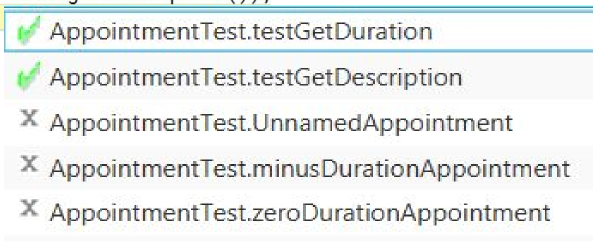
```
public void zeroDurationAppointment()
{
    Appointment appointm1 = new Appointment("ZeroDurationAppointm", 0);
    assertEquals("duration has to be at least 1", appointm1.getDuration());
}
```



```
public void minusDurationAppointment()
{
    Appointment appointm2 = new Appointment("NegativeDuration", -1);
    assertEquals("duration has to be at least 1", appointm2.getDuration());
}
```



```
public void UnnamedAppointment()
{
    Appointment appointm1 = new Appointment("", 1);
    assertEquals("description shouldn't be blank" , appointm1.getDescription());
}
```



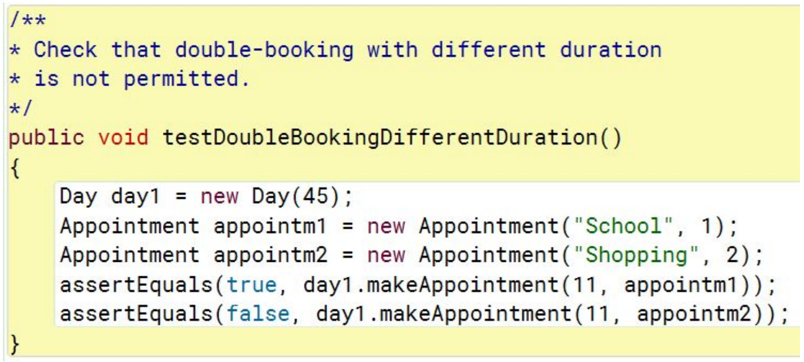
✓ AppointmentTest.testGetDuration
✓ AppointmentTest.testGetDescription
✗ AppointmentTest.UnnamedAppointment
✗ AppointmentTest.minusDurationAppointment
✗ AppointmentTest.zeroDurationAppointment

estimated time used: **7** minutes

4 Create a negative test in the DayTest class

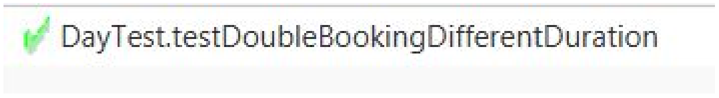
Try and set up a one-hour appointment and then put in a two-hour appointment at the same time.

We created a new *day1* object and added an one-hour appointment and an appointment with the duration of 2 hours to the object bench. When we first invoke the method *makeAppointment* we assert the result to true. The second appointment which was also booked on the same hour should assert to false.



```
/**
* Check that double-booking with different duration
* is not permitted.
*/
public void testDoubleBookingDifferentDuration()
{
    Day day1 = new Day(45);
    Appointment appointm1 = new Appointment("School", 1);
    Appointment appointm2 = new Appointment("Shopping", 2);
    assertEquals(true, day1.makeAppointment(11, appointm1));
    assertEquals(false, day1.makeAppointment(11, appointm2));
}
```

After we tested the new method we could see in the Test Result window that it worked properly.



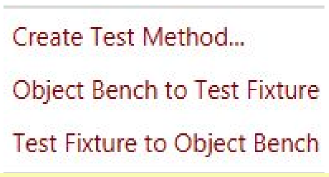
DayTest.testDoubleBookingDifferentDuration

estimated time used: **9** minutes

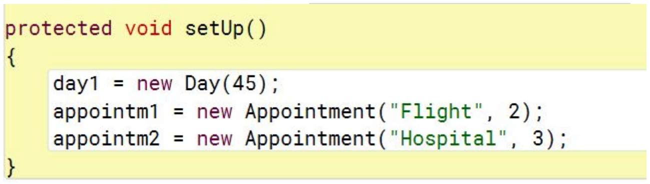
5 Set up a fixture for an additional test case

If several test methods require the same object or set of objects in a defined state, a fixture can help to save you some time.
First we created the objects *day1*, *appointm1* and *appointm2*.

In the menu of DayTest we selected the option *Object Bench to Test Fixture*. The instances disappeared from the object bench and were now in the body of the *setUp* method.

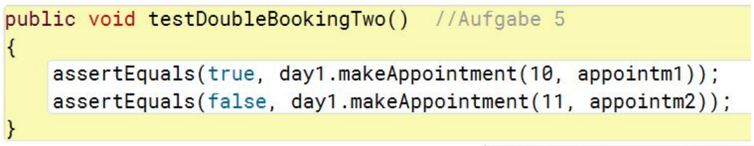


We wanted to test if an appointment can be made which starts before another appointment has ended (Prelab T 4.).



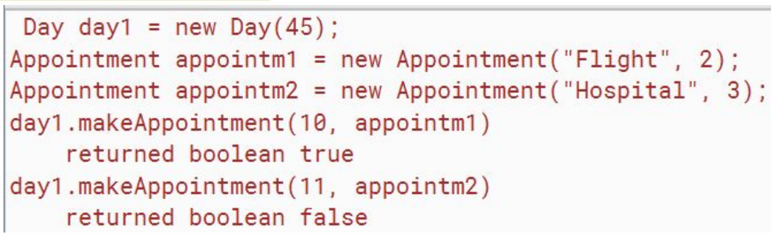
```
protected void setUp()
{
    day1 = new Day(45);
    appointm1 = new Appointment("Flight", 2);
    appointm2 = new Appointment("Hospital", 3);
}
```

After we had decided the name of the test case method *setUp* was called automatically and placed the objects on the object bench. Now the only thing we had to do was to add the appointments to a day and assert properly.

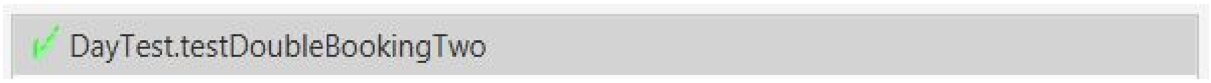


```
public void testDoubleBookingTwo()  //Aufgabe 5
{
    assertEquals(true, day1.makeAppointment(10, appointm1));
    assertEquals(false, day1.makeAppointment(11, appointm2));
}
```

Testing the new test case gave us a positive result.



```
 Day day1 = new Day(45);
Appointment appointm1 = new Appointment("Flight", 2);
Appointment appointm2 = new Appointment("Hospital", 3);
day1.makeAppointment(10, appointm1)
    returned boolean true
day1.makeAppointment(11, appointm2)
    returned boolean false
```

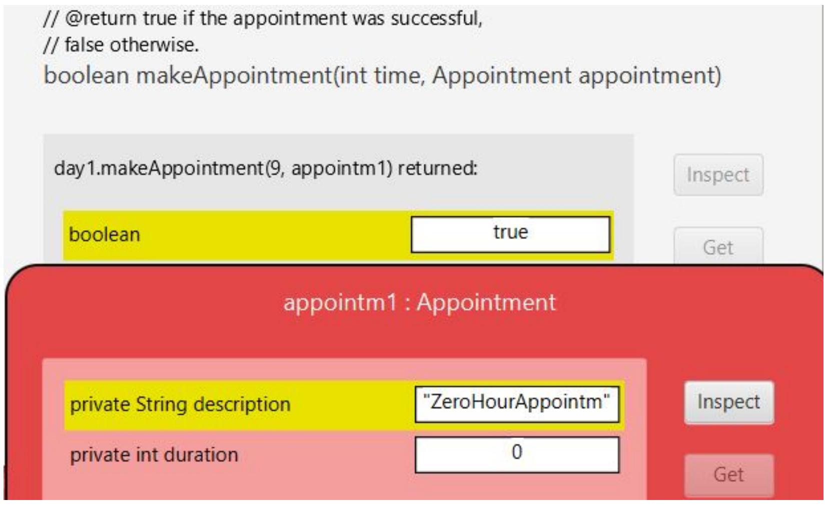


DayTest.testDoubleBookingTwo

estimated time used: **8** minutes

6 Add further automated tests (positive and negative) to the DayTest class

When we got to this exercise, we started discussing about what other tests we could possibly need to make sure our program works correctly. To make sure everything works fine, we had to further test all methods with different values, so we can see if any errors would occur. The first thing we were suspicious about was to test if it was possible to make an appointment that has a negative or 0 hour duration. To our surprise, the program returned true when we tried to make an appointment that has 0 as a value for duration.



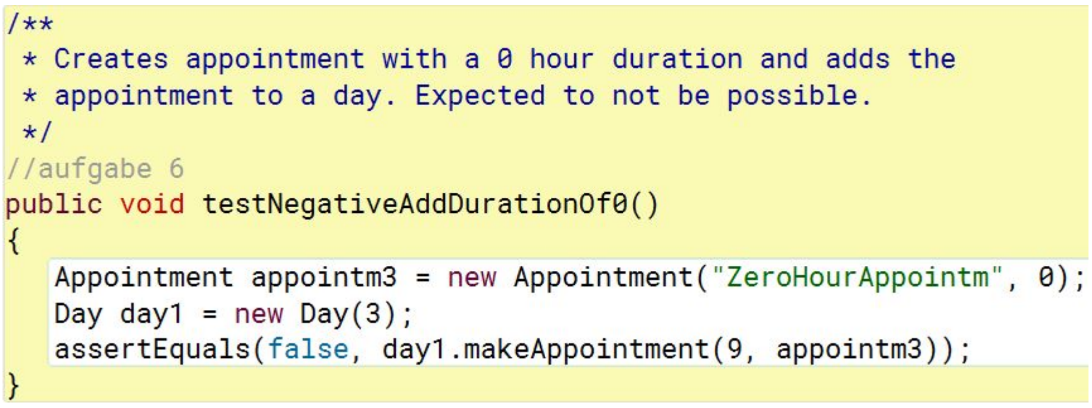
```
// @return true if the appointment was successful,
// false otherwise.
boolean makeAppointment(int time, Appointment appointment)

day1.makeAppointment(9, appointm1) returned:
boolean    true
```

appointm1 : Appointment

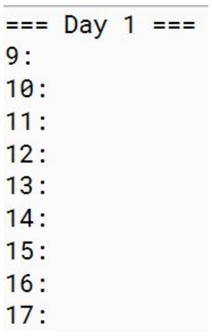
private String description	"ZeroHourAppointm"
private int duration	0

Then we went on and created a new negative test method about it, where we said that the assertion should be equal to false.



```
/**
 * Creates appointment with a 0 hour duration and adds the
 * appointment to a day. Expected to not be possible.
 */
//aufgabe 6
public void testNegativeAddDurationOf0()
{
    Appointment appointm3 = new Appointment("ZeroHourAppointm", 0);
    Day day1 = new Day(3);
    assertEquals(false, day1.makeAppointment(9, appointm3));
}
```

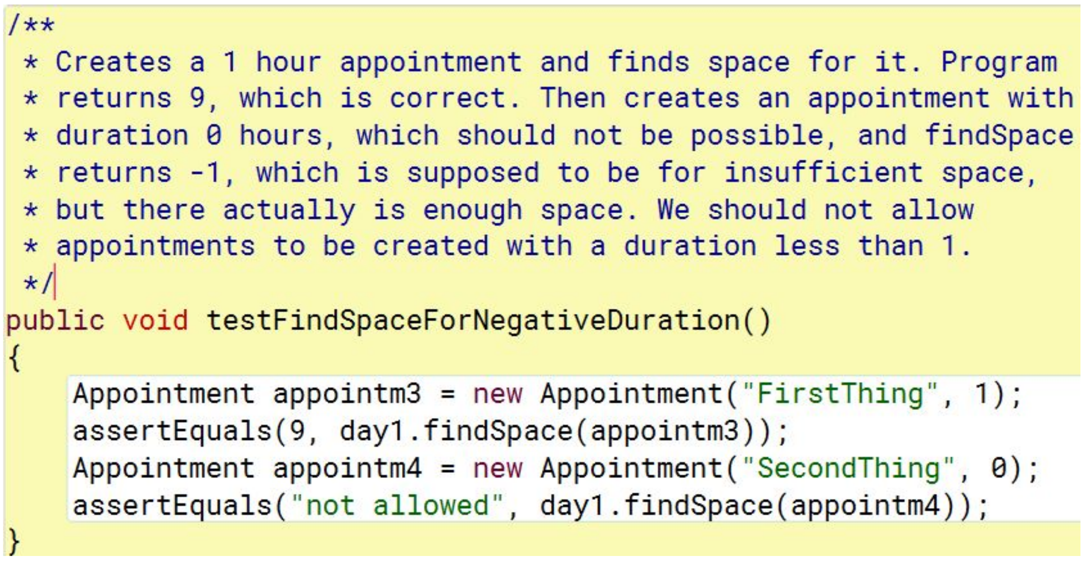
Additionally, after making a 0 hour appointment, we called the showAppointments method and saw that even if the program returned true, the appointment was actually not listed.



```
=== Day 1 ===
9:
10:
11:
12:
13:
14:
15:
16:
17:
```

That appeared to be very weird to us, so we tried to make an appointment again, using the zero hour appointment, and the program returned true. We noticed that we could keep making appointments and the program was always returning true, but it wasn't actually getting added to our list of appointments. We weren't able to create a test method for the showAppointments, but we saw that the program had some errors about that.

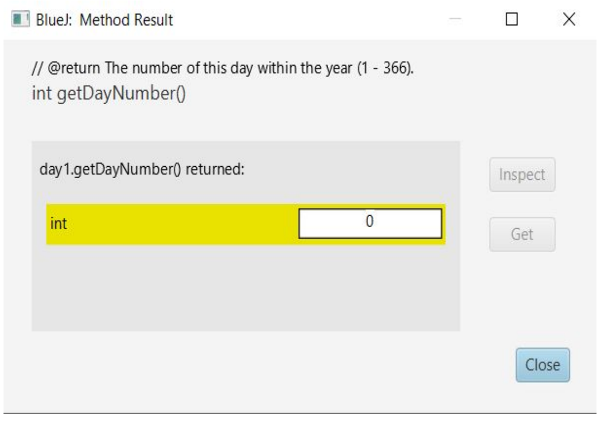
Shortly after this, we came up with an idea to call findSpace() and pass the 0 hour appointment as a value. We found another error in the program, because it was returning -1, which is supposed to be for insufficient space, even though there was free space. We made sure to create a test method about it and document it thoroughly.



```
/**
 * Creates a 1 hour appointment and finds space for it. Program
 * returns 9, which is correct. Then creates an appointment with
 * duration 0 hours, which should not be possible, and findSpace
 * returns -1, which is supposed to be for insufficient space,
 * but there actually is enough space. We should not allow
 * appointments to be created with a duration less than 1.
 */
public void testFindSpaceForNegativeDuration()
{
    Appointment appointm3 = new Appointment("FirstThing", 1);
    assertEquals(9, day1.findSpace(appointm3));
    Appointment appointm4 = new Appointment("SecondThing", 0);
    assertEquals("not allowed", day1.findSpace(appointm4));
}
```

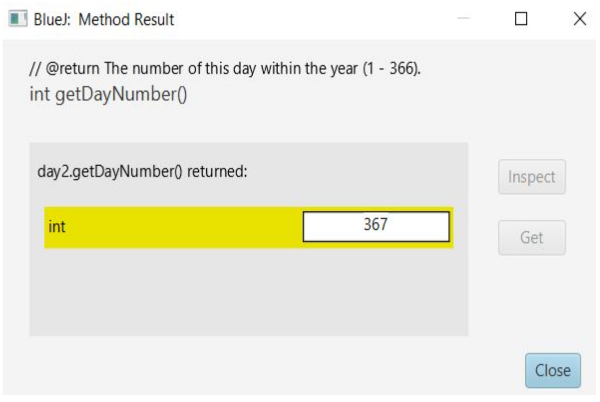
At this point we saw there are many errors occuring and they were all happening, because it was possible to create an appointment with zero or negative duration so that had to be fixed.
The next thing we noticed was that when creating a day, the day number should

be between 1 and 366, so we tried to create a day with values that are out of bound. We checked if it was possible to pass a value less than 1 and bigger than 366 and it was possible, we noticed we could even pass a negative number as a value.



```
// @return The number of this day within the year (1 - 366).
int getDayNumber()

day1.getDayNumber() returned:
int    0
```



```
// @return The number of this day within the year (1 - 366).
int getDayNumber()

day2.getDayNumber() returned:
int    367
```

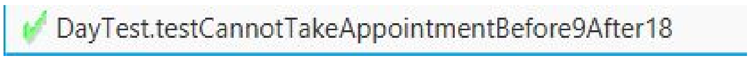
We both agreed that this shouldn't be allowed, so we created another negative test method.

```
/**
 * Checks if it's possible to create a "-1","0" or anything
 * above "366" day. Only days between 1 and 366 should be allowed.
 */
public void testNegativeNotAllowedDayNumber()
{
    Day day1 = new Day(0);
    assertEquals(false, day1.getDayNumber());
    Day day2 = new Day(-1);
    assertEquals(false, day2.getDayNumber());
    Day day3 = new Day(367);
    assertEquals(false, day2.getDayNumber());
}
```

After finding those errors, we were still curious if there is anything else that doesn't appear to be working correctly. We looked at the methods and checked if an appointment could be made before 9 or after 18 o'clock and created a negative test about it.

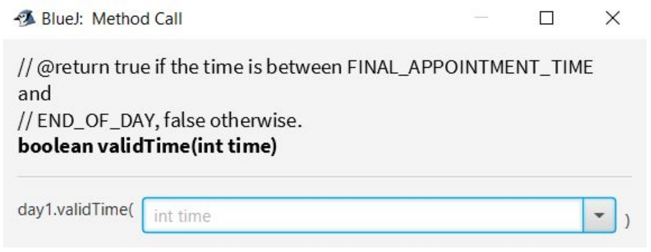
```
 /**
 * Creates a 1 hour appointment and checks that it can't be
 * added to a day before 9 or after 18 o'clock.
 */
public void testCannotTakeAppointmentBefore9After18()
{
    Appointment appointm3 = new Appointment("Playing", 1);
    Day day2 = new Day(4);
    assertEquals(false, day2.makeAppointment(8, appointm3));
    assertEquals(false, day2.makeAppointment(18, appointm3));
}
```

Then we ran a test and it was marked as correct.



✓ DayTest.testCannotTakeAppointmentBefore9After18

After that we noticed that that there is something weird about the documentation of the validTime method, because it was saying it returns true if the time is between final appointment time and end of day, which should be the same thing. We thought it was just mistyped instead of start of day.

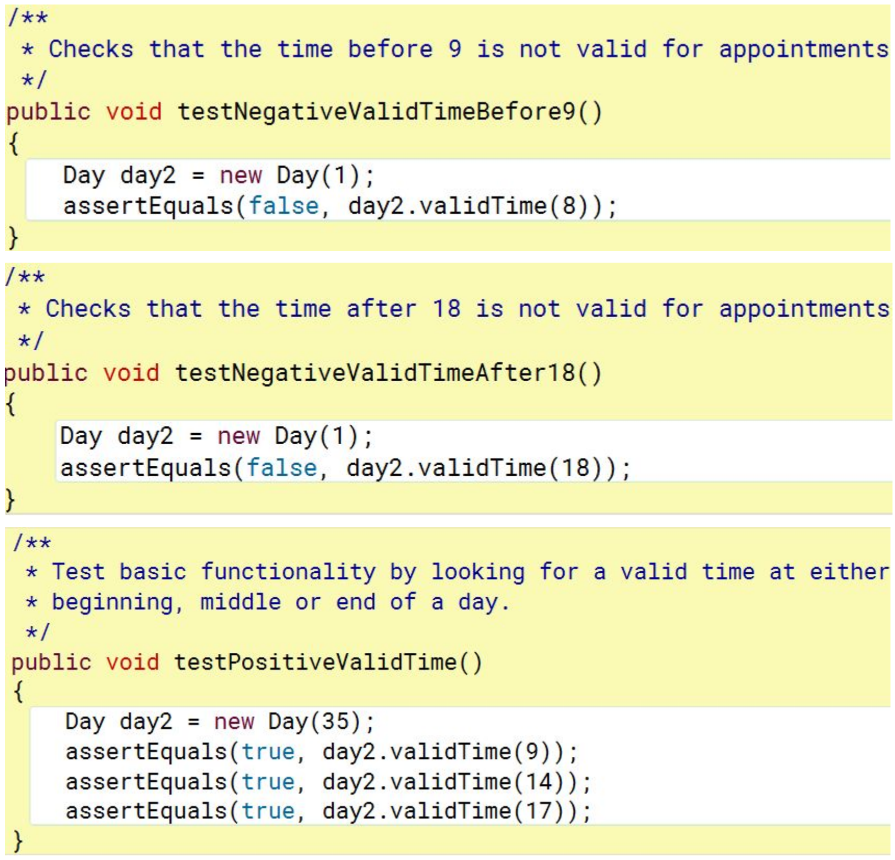


BlueJ: Method Call

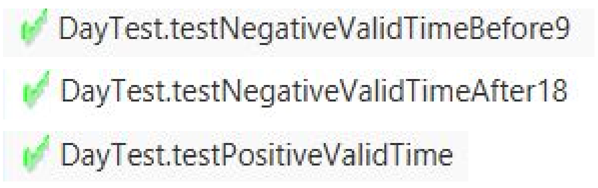
```
// @return true if the time is between FINAL_APPOINTMENT_TIME
and
// END_OF_DAY, false otherwise.
boolean validTime(int time)

day1.validTime( int time )
```

Then we created some negative and positive test methods to make sure it works well.

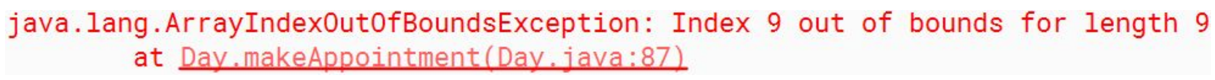


```
/**
 * Checks that the time before 9 is not valid for appointments
 */
public void testNegativeValidTimeBefore9()
{
    Day day2 = new Day(1);
    assertEquals(false, day2.validTime(8));
}
/**
 * Checks that the time after 18 is not valid for appointments
 */
public void testNegativeValidTimeAfter18()
{
    Day day2 = new Day(1);
    assertEquals(false, day2.validTime(18));
}
/**
 * Test basic functionality by looking for a valid time at either
 * beginning, middle or end of a day.
 */
public void testPositiveValidTime()
{
    Day day2 = new Day(35);
    assertEquals(true, day2.validTime(9));
    assertEquals(true, day2.validTime(14));
    assertEquals(true, day2.validTime(17));
}
```



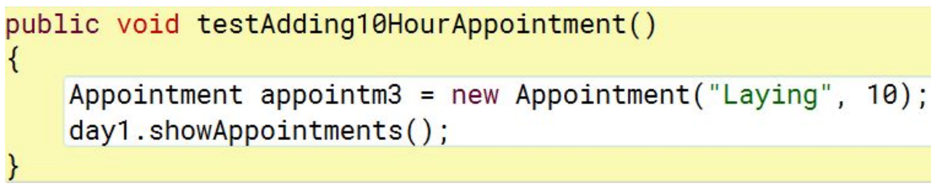
✓ DayTest.testNegativeValidTimeBefore9
✓ DayTest.testNegativeValidTimeAfter18
✓ DayTest.testPositiveValidTime

The last thing we wanted to see is what happens if we add an appointment with duration of 10 hours to a day. We created a new test method and after we added the 10 hour appointment to a day we noticed some weird behavior.



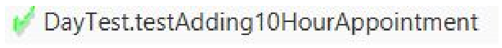
```
java.lang.ArrayIndexOutOfBoundsException: Index 9 out of bounds for length 9
	at Day.makeAppointment(Day.java:87)
```

BlueJ returned out of bound exception in the terminal which we thought is correct. What we thought is weird was that it didn't return true or false about making the appointment. We also didn't get prompted about the assertion so that we can write it should be equal to false. We checked what was recorded and the test method looked like that.



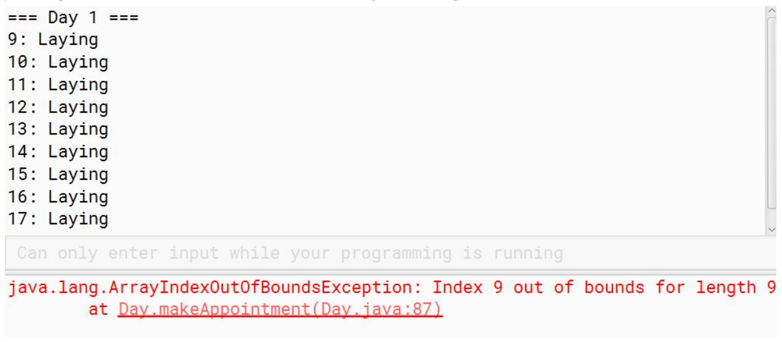
```
public void testAdding10HourAppointment()
{
    Appointment appointm3 = new Appointment("Laying", 10);
    day1.showAppointments();
}
```

When we ran a test, the test had a green tick, but we thought that it's not correct and there was an error.



✓ DayTest.testAdding10HourAppointment

We called the showAppointments method and we were surprised that all the hours were filled, which meant that we added the appointment to a day, but the program didn't return anything about its creation.

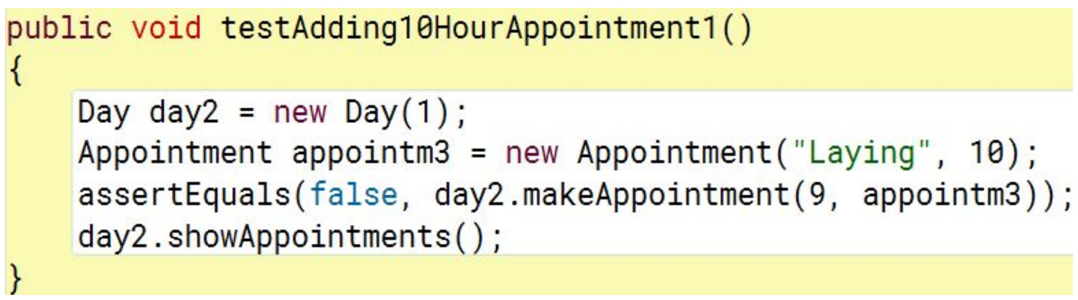


```
=== Day 1 ===
9: Laying
10: Laying
11: Laying
12: Laying
13: Laying
14: Laying
15: Laying
16: Laying
17: Laying
```

Can only enter input while your programming is running

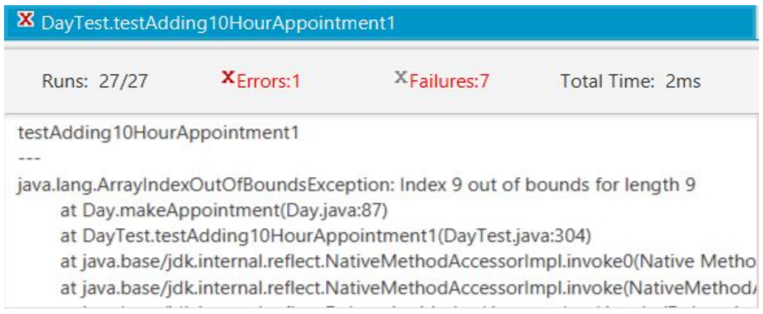
```
java.lang.ArrayIndexOutOfBoundsException: Index 9 out of bounds for length 9
	at Day.makeAppointment(Day.java:87)
```

We created a new test method and manually typed in the assertion, because we were sure there was something wrong about it.



```
public void testAdding10HourAppointment1()
{
    Day day2 = new Day(1);
    Appointment appointm3 = new Appointment("Laying", 10);
    assertEquals(false, day2.makeAppointment(9, appointm3));
    day2.showAppointments();
}
```

This time after running a test, we had an error about it.



```
DayTest.testAdding10HourAppointment1

Runs:  27/27        Errors:1        Failures:7        Total Time:  2ms

testAdding10HourAppointment1
---
java.lang.ArrayIndexOutOfBoundsException: Index 9 out of bounds for length 9
	at Day.makeAppointment(Day.java:87)
	at DayTest.testAdding10HourAppointment1(DayTest.java:304)
	at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Metho
	at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethod/
```

estimated time used: **40** minutes

After the Lab

We had few minutes left and we weren't ready with all the exercises yet. We exchanged e-mails and sent the BlueJ file and our notes to each other. We both agreed to work on the exercises at home.

Evaluation

Sophia:

I used the Unit Test before but now I learned how to use it the right way. First I was unsure about assertEquals and thought that only true and false statements are possible.
It is very important to write down the ideas for the test cases and keep track of everything, otherwise it gets very confusing.
The fixture is a very useful tool which I will use in future project to save me some time while testing.
I still need some more time, in order to look more into positive and negative testing.

Pavel:

While working on the exercises I learned more about how to do proper testing and understood what exactly BlueJ does when we create a test method. I could go inside the Test class, read the code and understand clearly what each of our test methods was doing. I also learned how to look at the code from a different perspective and think about what errors there might be in a program. I learned how to set up a fixture and why it can be useful when we are doing tests. I had to create both positive and negative tests and it helped me think more about the process of testing. I learned how to find errors and think of a way to handle them so the program can be improved.

estimated time used for report: **2** hours

Appendix

Pre-Lab Exercises

P1. Write down six test cases for a diary that takes Appointments on a specific Day. Give each test case a name like in the lecture.

T 1. Can there be more than one appointment on a specific day at a specific time?
T 2. Is there a way to see all the appointments for a specific day?
T 3. Is there a way to see all appointments for a number of days?
T 4. Can an appointment start before another appointment has ended?
T 5. Is there a way to see all appointments at a specific time for the whole week?
T 6. Can an appointment be removed from the specific day?

P2. What assertions do you need in order to be able to automate the tests?

T 1. assertEquals(true, makeAppointment (9, appointm1));
assertEquals(true, makeAppointment (14,appointm2));
T 2. no assertion - by invoking showAppointments( );
T 3. no assertion - only works for one day object
T 4. //create appointm1 with duration 2h ; appointm2 with 3h
assertEquals(true, makeAppointment(9, appointm1));
assertEquals(false, makeAppointment(10,appointm2));
T 5. no assertion - only works for one day object
T 6. no assertion - this option is not given

Code

DayTest class

```
/**
 * The test class DayTest.
 *
 * @author  David J. Barnes and Michael Kolling
 * @version 2008.03.30
 */
public class DayTest extends junit.framework.TestCase
{
    private Appointment appointm1;
    private Appointment appointm2;
    private Day day1;
    /**
     * Default constructor for test class DayTest
     */
    public DayTest()
    {
    }

    /**
     * Sets up the test fixture.
     *
     * Called before every test case method.
     */
    // fixture for Aufgabe 5
    protected void setUp()
    {
        day1 = new Day(45);
        appointm1 = new Appointment("Flight", 2);
        appointm2 = new Appointment("Hospital", 3);
    }

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

    /**
     * Test basic functionality by booking at either end
     * of a day, and in the middle.
     */
    public void testMakeThreeAppointments() //already in here
    {
        Day day1 = new Day(1);
        Appointment appointm1 = new Appointment("Java lecture", 1);
        Appointment appointm2 = new Appointment("Java class", 1);
        Appointment appointm3 = new Appointment("Meet John", 1);
        assertEquals(true, day1.makeAppointment(9, appointm1));
        assertEquals(true, day1.makeAppointment(13, appointm2));
        assertEquals(true, day1.makeAppointment(17, appointm3));
    }

    /**
     * Check that double-booking is not permitted.
     */
    public void testDoubleBooking() //already in here
    {
        Day day1 = new Day(1);
        Appointment appointm1 = new Appointment("Java lecture", 1);
        Appointment appointm2 = new Appointment("Error", 1);
        assertEquals(true, day1.makeAppointment(9, appointm1));
        assertEquals(false, day1.makeAppointment(9, appointm2));
    }

    /**
    * Check that the first appointment for an empty diary is at 9 a.m.
    */
    public void testFindSpaceEmpty() //already in here
    {
        Day day2 = new Day(13);
        day2.showAppointments();
        Appointment a = new Appointment("First", 1);
        assertEquals(9, day2.findSpace(a));
    }

    /**
    *
    */
    public void testFindSpace9() //already in here
    {
        Day day1 = new Day(13);
        Appointment a = new Appointment("first", 1);
        assertEquals(9, day1.findSpace(a));
        day1.showAppointments();
    }

    /**
    *
    */
    public void testNotFull() //already in here
    {
        Day day1 = new Day(13);
        Appointment appointm1 = new Appointment("first!", 1);
        assertEquals(true, day1.makeAppointment(11, appointm1));
        Appointment appointm2 = new Appointment("second", 1);
        assertEquals(9, day1.findSpace(appointm2));
        day1.showAppointments();
        Appointment b3 = new Appointment("BIg", 3);
        assertEquals(12, day1.findSpace(b3));
    }

    /**
    * Checks that findSpace method returns 10 for the second
    * appointment in a day, if there is already a single
    * one-hour appointment.
    */
    //aufgabe 1
    public void testFindSpace()
    {
        Day day1 = new Day(1);
        Appointment appointm1 = new Appointment("Swimming", 1);
        Appointment appointm2 = new Appointment("Soccer", 1);
        assertEquals(true, day1.makeAppointment(9, appointm1));
        assertEquals(10, day1.findSpace(appointm2));
    }

    /**
    * Check that findSpace returns a value of 10 for an one-hour
    * appointment if there is already an one-hour appointment at 9am.
    */
    public void testFindSpaceTen() //aufgabe 1
    {
        Day day2 = new Day(2);
        Appointment appointm3 = new Appointment("Wake up", 1);
        Appointment appointm4 = new Appointment("Brush Teeth" , 1);
        assertEquals(true, day2.makeAppointment(9, appointm3));
        assertEquals(10, day2.findSpace(appointm4));
    }

    /**
    * Check that findSpace returns a value of -1 if an attempt is
    * made to find an appointment in a day, that is already full.
    */
    public void testNoSpaceLeft()  //aufgabe 2
    {
        Day day1 = new Day(23);
        Appointment appointm1 = new Appointment("School", 9);
        Appointment appointm2 = new Appointment("After School", 1);
        assertEquals(true, day1.makeAppointment(9, appointm1));
        assertEquals(-1, day1.findSpace(appointm2));
    }

    /**
    * Check that double-booking with different duration
    * is not permitted.
    */
    public void testDoubleBookingDifferentDuration() //aufgabe 4
```

```
{
    Day day1 = new Day(45);
    Appointment appointm1 = new Appointment("School", 1);
    Appointment appointm2 = new Appointment("Shopping", 2);
    assertEquals(true, day1.makeAppointment(11, appointm1));
    assertEquals(false, day1.makeAppointment(11, appointm2));
}

public void testDoubleBookingTwo() //using fixture for aufgabe 5
{
    assertEquals(true, day1.makeAppointment(10, appointm1));
    assertEquals(false, day1.makeAppointment(11, appointm2));
}

/**
 * Check that findSpace returns 9 for the first available time
 * for an appointment in a free day.
 */
//aufgabe 6
public void testPositiveFirstAvailableTime()
{
    Appointment appointm3 = new Appointment("firstThingForToday", 1);
    Day day2 = new Day(2);
    assertEquals(9, day2.findSpace(appointm3));
}

 /**
 * Check that an appointment with a duration of -1
 * cannot be added to a day.
 */
 //aufgabe 6
public void testNegativeAppointmentDurationMinusOne()
 {
    Appointment appointm3 = new Appointment("durationMinusOne", -1);
    Day day2 = new Day(1);
    assertEquals(false, day2.makeAppointment(9, appointm3));
}

/**
  * Creates an appointment with duration 8 hours ( 9 to 17 ), then
  * finds space for another appointment with 1 hour duration and
  * should return 17.
  */
 //aufgbe 6
public void testFindSpaceAfter8hAppointment()
{
    Appointment appointm3 = new Appointment("Studying", 8);
    Day day2 = new Day(2);
    Appointment appointm4 = new Appointment("Gaming", 1);
    assertEquals(true, day2.makeAppointment(9, appointm3));
    assertEquals(17, day2.findSpace(appointm4));
}

/**
 * Creates appointment with a 0 hour duration and adds the
 * appointment to a day. Expected to not be possible.
 */
//aufgabe 6
public void testNegativeAddDurationOf0()
{
   Appointment appointm3 = new Appointment("ZeroHourAppointm", 0);
   Day day1 = new Day(3);
   assertEquals(false, day1.makeAppointment(9, appointm3));
}

 /**
 * Creates a 1 hour appointment and checks that it can't be
 * added to a day before 9 or after 18 o'clock.
 */
 //aufgabe 6
public void testCannotTakeAppointmentBefore9After18()
{
    Appointment appointm3 = new Appointment("Playing", 1);
    Day day2 = new Day(4);
    assertEquals(false, day2.makeAppointment(8, appointm3));
    assertEquals(false, day2.makeAppointment(18, appointm3));
}

/**
 * Checks if it's possible to create a "-1","0" or anything
 * above "366" day. Only days between 1 and 366 should be allowed.
 */
//aufgabe 6
public void testNegativeNotAllowedDayNumber()
{
    Day day1 = new Day(0);
    assertEquals(false, day1.getDayNumber());
    Day day2 = new Day(-1);
    assertEquals(false, day2.getDayNumber());
    Day day3 = new Day(367);
    assertEquals(false, day2.getDayNumber());
}

/**
 * Checks that the time before 9 is not valid for appointments
 */
//aufgabe 6
public void testNegativeValidTimeBefore9()
{
    Day day2 = new Day(1);
```

```
        assertEquals(false, day2.validTime(8));
    }

    /**
     * Checks that the time after 18 is not valid for appointments
     */
    //aufgabe 6
    public void testNegativeValidTimeAfter18()
    {
        Day day2 = new Day(1);
        assertEquals(false, day2.validTime(18));
    }

    /**
     * Test basic functionality by looking for a valid time at either
     * beginning, middle or end of a day.
     */
     //aufgabe6
    public void testPositiveValidTime()
    {
        Day day2 = new Day(35);
        assertEquals(true, day2.validTime(9));
        assertEquals(true, day2.validTime(14));
        assertEquals(true, day2.validTime(17));
    }

    /**
     * Creates a 1 hour appointment and finds space for it. Program
     * returns 9, which is correct. Then creates an appointment with
     * duration 0 hours, which should not be possible, and findSpace
     * returns -1, which is supposed to be for insufficient space,
     * but there actually is enough space. We should not allow
     * appointments to be created with a duration less than 1.
     */
    //aufgabe 6
    public void testFindSpaceForNegativeDuration()
    {
        Appointment appointm3 = new Appointment("FirstThing", 1);
        assertEquals(9, day1.findSpace(appointm3));
        Appointment appointm4 = new Appointment("SecondThing", 0);
        assertEquals("not allowed", day1.findSpace(appointm4));
    }

    public void testAdding10HourAppointment()
    {
        Appointment appointm3 = new Appointment("Laying", 10);
        day1.showAppointments();
    }
}
```

AppointmentTest class

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

/**
 * The test class AppointmentTest.
 *
 * @author  (your name)
 * @version (a version number or a date)
 */
public class AppointmentTest
{
    /**
     * Default constructor for test class AppointmentTest
     */
    public AppointmentTest()
    {
    }

    /**
     * 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()
    {
    }
```

```
/**
 * Checks if the getDescription method returns the
 * description of an appointment correctly.
 */
//aufgabe 3
@Test
public void testGetDescription()
{
   Appointment appointm1 = new Appointment("Chillin", 5);
   assertEquals("Chillin", appointm1.getDescription());
}

/**
 * Checks if the getDuration method returns the
 * duration of an appointment correctly.
 */
//aufgabe 3
@Test
public void testGetDuration()
{
   Appointment appointm1 = new Appointment("Relaxin", 4);
   assertEquals(4, appointm1.getDuration());
}

/**
 * Checks that creating an appointment with duration of 0
 * is possible which should not be.
 */

@Test
public void zeroDurationAppointment()
{
   Appointment appointm1 = new Appointment("ZeroDurationAppointm", 0);
   assertEquals("duration has to be at least 1", appointm1.getDuration());
}

/**
 * Checks that creating an appointment with a negative duration
 * is possible which should not be.
 */

@Test
public void minusDurationAppointment()
{
   Appointment appointm2 = new Appointment("NegativeDuration", -1);
   assertEquals("duration has to be at least 1", appointm2.getDuration());
}

/**
```

```
     * Checks if an appointment can be unnamed. It should
     * at least have some sort of description.
     */

    @Test
    public void UnnamedAppointment()
    {
        Appointment appointm1 = new Appointment("", 1);
        assertEquals("description shouldn't be blank" , appointm1.getDescription());
    }
}
```