Exercise 6.6

It should be good enough to check just a few of the possibilities. It would be wise to check the boundaries as well as at least one non-boundary value. Examples of times to test: 9, 13, 17

Exercise 6.7

Yes, it is possible to use the same appointment object at different times within a single day.

The tests are not as legitimate anymore, because we can no longer see the difference between the objects (because there is none) and therefore we can't be sure that the appointments are in the correct time slots.

It would make sense to use the same appointment object if you have a recurring appointment. For instance you might have a daily meeting with someone.

Exercise 6.8

Several of the tests will fail.

makeAppointment only uses the starting time of an appointment to check if the slot is free, and it can therefore overwrite other appointments or exceed the boundary (which results in an exception). For instance, try booking a two-hour appointment at 1700 or one at 0900 when the 1000 time is already booked.

Exercise 6.9

	Positive	**Negative**
6.1	X	
6.2	X	
6.3	X	
6.4		X
6.5	X	
6.6		X

Other negative tests:
- test how the application handles a 'null' object as an appointment.
- test zero and negative values for duration and time.

A positive test could be to test that appointments longer than one hour have the correct duration after being inserted.

Exercise 6.13

The regression test we have done so far still requires manual inspection of the output. This would be very hard and time consuming to do if we had many hundreds or thousands of tests. The difficulty means that, in practice, manual checking will not be done thoroughly, which calls into question the value of the testing.

Exercise 6.15

A test class initially contains the methods:

The constructor
setUp()
tearDown()

Exercise 6.16

1. Start recording a new test method called testFindSpace10
2. Create a Day object
3. Create an Appointment with an one-hour duration
4. Make this appointment on the day object at time 9
5. Assert that this result is equal to true
6. Make a new appointment object with an one-hour duration
7. Call the findSpace method on the Day object with the new assignment object
8. Assert that this result is equal to 10

Exercise 6.19

The test result window reports one failure and shows a big red bar.

Exercise 6.21

The terminal window shows:

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

There is no way to know (just from the test output) that the tests passed, because we only get the test result value. So we should not trust it.

Exercise 6.22

Calling testPlus() returns the value 1.

It is not the same result as was returned by testAll().

Calling testPlus() one more time returns 7.

It should always give the same answer.

According to the source code it should return 7.

Exercise 6.23

No.

Exercise 6.24

The testMinus() method is similar in structure to the testPlus() method. Making a walk through of testMinus() obviously makes us want to take a closer look at the minus() method. Furthermore we could make a note to check that negative values are handled correctly.

Exercise 6.25

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	'+'

Exercise 6.26

No. The equals method does not does not explicitly check the value of the previousOperator when it is '-'. Therefore anything other than a '+' sign will result in a subtraction.

Exercise 6.27

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	'+'
clear	0	0	'+'

It is not in the same state as the previous clear() (the previousOperator differs).

This could have a big impact on subsequent calls. This is most likely the source of the inconsistent result we saw in exercise 6.22.

Exercise 6.28

Two things has come to our attention while doing the walk through:

1. equals() does not explicitly test for the '-' value of previousOperator.
2. clear() does not clear all the fields.

The clear method should be changed to this:

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

The equals method should be changed to this:

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

Exercise 6.29

Method called	displayValue	leftOperand	previousOperator
initial state	0	0	' '
clear	0	0	' '
numberPressed(9)	9	0	' '
plus	0	9	'+'
numberPressed(1)	1	9	'+'
minus	0	10	'-'
numberPressed(4)	4	10	'-'
equals	6	0	'-'

The result should be 6, which is indeed what is in the displayValue at the end of the walk through.

Exercise 6.31

Yes. The output shows the same information as we collected in the table in the previous exercises.

Exercise 6.33

Debug statements have to be removed again.

Debug statements might be less error prone than the manual walk through.

Debug statements might be easier/faster to do (this is probably subjective)

Debug statements are easier to use when you need to verify a correction.

The activity of doing the manual walk through might give a better understanding of the program.

Exercise 6.35

See the project from the cd: calculator-full-solution

Exercise 6.36

The bugs in the bricks project are:

class Brick:

getSurfaceArea(): *side1 should be side2 in sum*
getWeight(): *1000 should be 1000.0*

class Palette:

getHeight(): *% should be **
getWeight(): `+ baseWeight` *is missing*