Assignment 10

Grades for this assignment come from the quiz: false  
[**Click here to go to the quiz**](https://learn-slcc.uen.org/courses/61946/quizzes/%7B%7B%20quiz_id%20%7D%7D)

This assignment is linked to the discussion, false. Grading will be based on posts in the topic.  
[**Click here to go to the discussion**](https://learn-slcc.uen.org/courses/61946/discussion_topics/%7B%7B%20discussion_topic_id%20%7D%7D)

Web Resources

* [Wikipedia](http://en.wikipedia.org/wiki/Unit_testing)
* [Official JUnit Website](http://www.junit.org/)
* [JUnit 4.x Quick Tutorial](http://code.google.com/p/t2framework/wiki/JUnitQuickTutorial)
* [JUnit Primer](http://www.clarkware.com/articles/JUnitPrimer.html)
* [Unit Testing Java Programs - Playing Golf](http://javaboutique.internet.com/tutorials/UnitTesting/)
* <http://users.csc.calpoly.edu/~djanzen/tdl/tddintro/JGraspIntegration/jgrasp_junit_integration.html>
* <http://sqa.fyicenter.com/FAQ/JUnit/How_To_Group_Test_Cases_Class_using_JUnit_TestSu.html>

Problem Statement

A parking garage charges a $2.00 minimum fee to park for up to three hours. The garage charges an additional $0.50 per hour for each hour, or part thereof, in excess of three hours. The maximum charge for any given 24-hour period is $10.00. Assume that no car parks for longer than 24 hours.

Specifications

* Write a program that calculates and displays the parking charges for each customer who parked in the garage on a given day. Your program should:
  1. Ask the user to enter the hours parked for each customer.
  2. Display the charge for the current customer.
  3. Calculate and display the running total of the day's receipts.
  4. Create method calculateCharges() that accepts the number of hours parked and returns the parking charges for each customer.
* Create a ParkingChargesTest class with a series of test methods to test calculateCharges() for all possible cases, boundary values, etc.
* Create a ParkingChargesTestSuite class that uses jUnit to run the test class to verify that calculateCharges() is performing correctly.

Admin

* Grading
  + 0 points if your program does not compile.
  + -5 for comments, indentation and placement of {} not per [Style Guide](http://www.cs.slcc.edu/style-guide.shtml).
  + -10 for each specification not met.
* Submission
  + Attach an executable .jar file that also contains your .java source code files and submit.

**J-Unit Code Demo 1**

public class TestAdder {

public void testSum() {

Adder adder = new AdderImpl();

// can it add positive numbers?

assert(adder.add(1, 1) == 2);

assert(adder.add(1, 2) == 3);

assert(adder.add(2, 2) == 4);

// is zero neutral?

assert(adder.add(0, 0) == 0);

// can it add negative numbers?

assert(adder.add(-1, -2) == -3);

// can it add a positive and a negative?

assert(adder.add(-1, 1) == 0);

// how about larger numbers?

assert(adder.add(1234, 988) == 2222);

}

}

In this case the unit test, having been written first, acts as a design document specifying the form and behaviour of a desired solution, but not the implementation details, which are left for the programmer. Following the "do the simplest thing that could possibly work" practice, the easiest solution that will make the test pass is shown below.

**J-Unit Code Demo 2**

interface Adder {

int add(int a, int b);

}

class AdderImpl implements Adder {

int add(int a, int b) {

return a + b;

}

}

Another Demo

public class Subscription {  
  
 private int price ; // subscription total price in euro-cent  
 private int length ; // length of subscription in months  
  
 // constructor :  
 public Subscription(int p, int n) {  
 price = p ;  
 length = n ;  
 }  
  
 /\*\*  
 \* Calculate the monthly subscription price in euro,  
 \* rounded up to the nearest cent.  
 \*/  
 public double pricePerMonth() {  
 double r = (double) price / (double) length ;  
 return r ;  
 }  
  
 /\*\*  
 \* Call this to cancel/nulify this subscription.  
 \*/  
 public void cancel() { length = 0 ; }  
  
}

For example, new Subscription(1000,2) will create a new subscription of 1000 Euro-cent for the total period of 2 months.

By the way, the class has a number of bugs; e.g. pricePerMonth is supposed to return the price per month in euro. However it calculates the price in cent.

**Let’s write a test**

Let us write two simple tests to check if pricePerMonth correctly calculates the price per month:

1. If we have a subscription of 200 cent for a period of 2 month, its monthly price should be 1 euro, right?
2. The monthly price is supposed to be rounded up to the nearest cent. So, if we have a subscription of 200 cent for a period of 3 month, its monthly price should be 0:67 euro.

Of course we can write tests without JUnit; but this tutorial is about JUnit. So here is how we write them in JUnit.

Each of the tests above will be implemented as a test method; you then group your test methods in a test class. Usually you would group all the test methods that test a certain target class *C* is a test class called *CTest*, but you can also have multiple test classes if you want, and call them whatever you want.

Here is my test class implementing the above two tests:

import org.junit.\* ;  
import static org.junit.Assert.\* ;  
  
public class SubscriptionTest {  
  
 @Test  
 public void test\_returnEuro() {  
 System.out.println("Test if pricePerMonth returns Euro...") ;  
 Subscription S = new Subscription(200,2) ;  
 assertTrue(S.pricePerMonth() == 1.0) ;  
 }  
  
 @Test  
 public void test\_roundUp() {  
 System.out.println("Test if pricePerMonth rounds up correctly...") ;  
 Subscription S = new Subscription(200,3) ;  
 assertTrue(S.pricePerMonth() == 0.67) ;  
 }  
}

The marker @Test is called an annotation in Java. When we later execute JUnit’s test runner it needs to know which methods in your test class are test methods (e.g. you may have several helper methods in your test class). The @Test is used to mark that a method is a test method.

In the first test (test\_returnEuro) we first create a Subscription; we call it S. Then we want to check that S.pricePerMonth() will return the expected value of 1.0. The checking is done by the code:

assertTrue(S.pricePerMonth() == 1.0)

By the way, the annotation @Test and the method assertTrue are things exported by the JUnit library; so you need the imports as in the above code to use them.