

**Software Engineering**

 **Cal 2**

|  |  |
| --- | --- |
| **Teacher:** | **Daniel Rodríguez** |
| **Subject:** | **Software Engineering** |
| **Students:** | **Carlos Javier Prado Vázquez Javier José Guzmán Rubio** |
| **Group:** | **18** |
| **Date:** | **16/05/2021** |

Index

[1º Junit Test 3](#_Toc72022864)

[Rectangle Junit Tests: 5](#_Toc72022865)

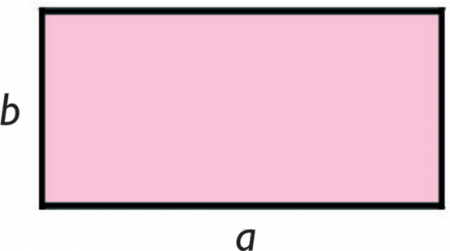
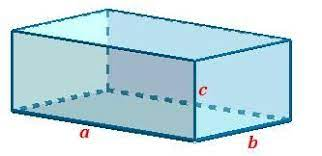
[Parallelepiped Junit Tests: 7](#_Toc72022866)

[2ºMock Test 10](#_Toc72022867)

[3ºMeasure the Code 11](#_Toc72022868)

# 1º Junit Test

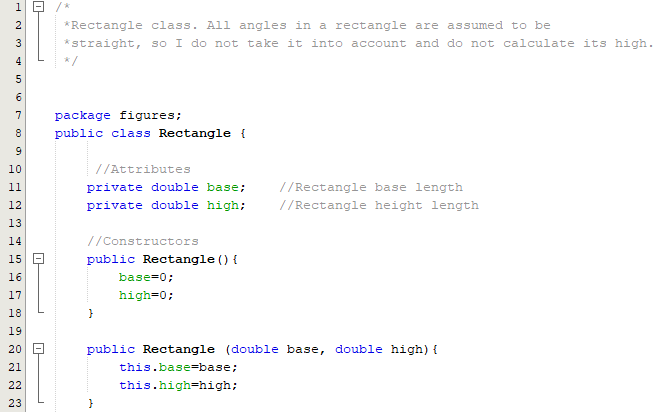
First, we needed to choose a mathematical function, in this case we selected the parallelepiped volume and surface, for that we thought of doing 1st the rectangle method and the re use it for the parallelepiped.

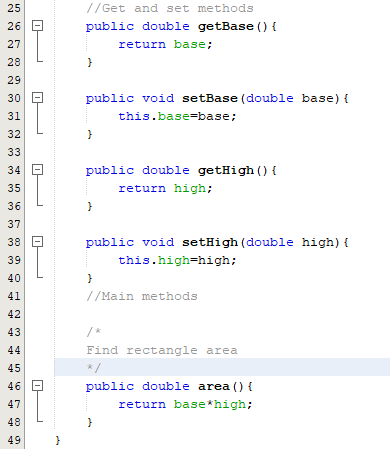


Parallelepiped

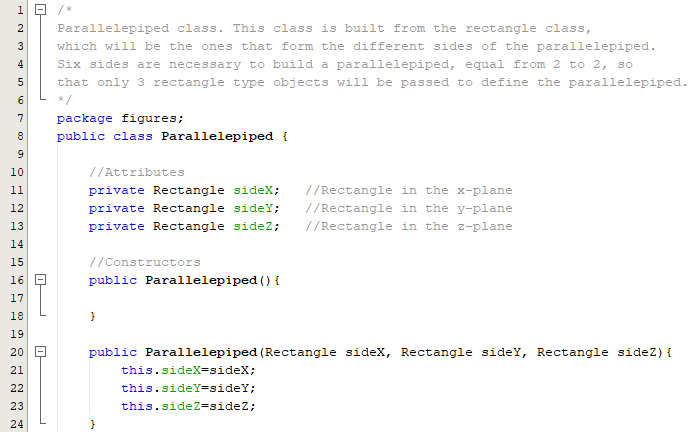
Rectangle

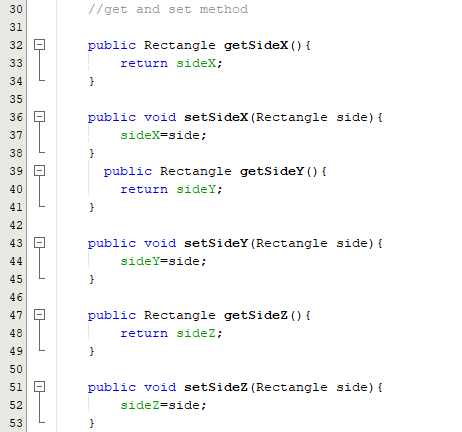
As we know a parallelepiped is made by different rectangles, so we did this method for the following methods for then and then we reused them in the parallelepiped.

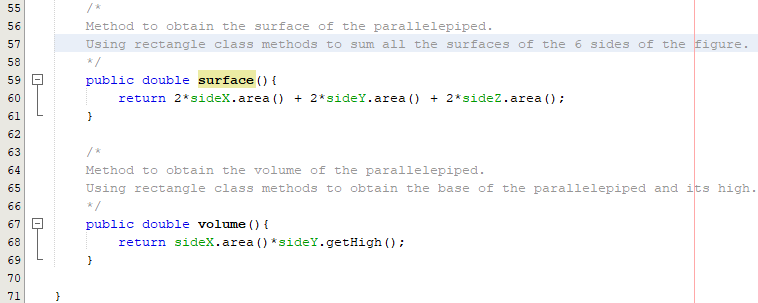




Then we reused this for the Parallelepiped:



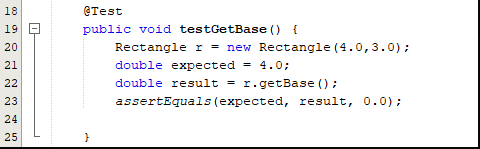


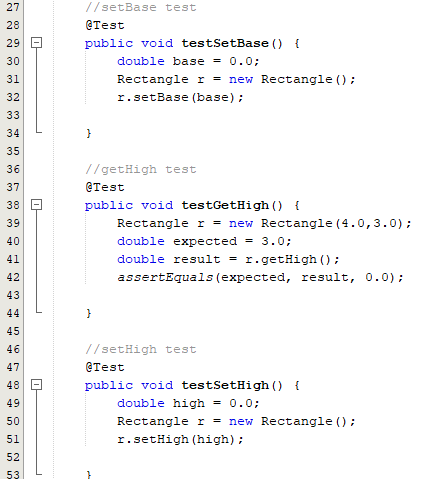


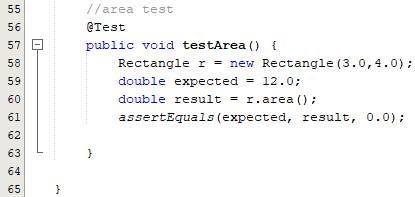
Then we created the test cases with *Junit* to be able to evaluate if the operation of each of the methods of the class behaves as expected.

## Rectangle Junit Tests:

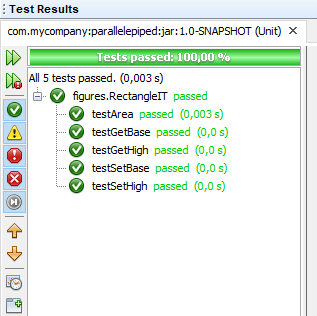
The getter and setter tests are not very relevant for these geometric functions, but we have calculated them the same to check that there were no problems with them.







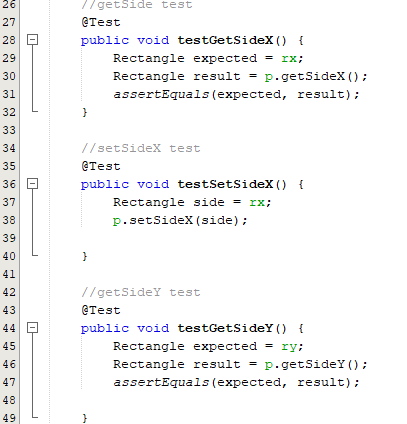
To obtain these results, we can right click on a test class and select the "Test File" option.

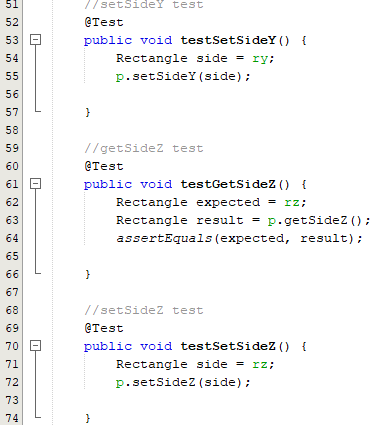


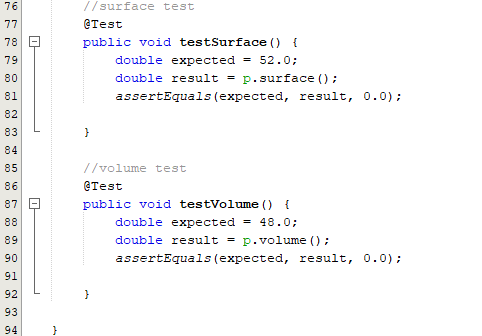
As we can see we obtain the expected values.

## Parallelepiped Junit Tests:

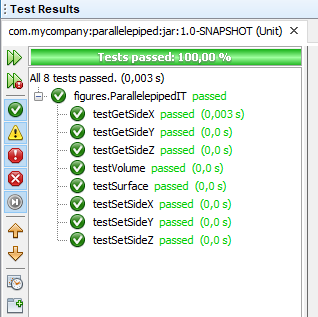
The getter and setter tests are not very relevant for these geometric functions, but we have calculated them the same to check that there were no problems with them.







To obtain these results, we can right click on a test class and select the "Test File" option.

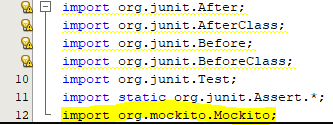


As we can see, the methods are working as expected.

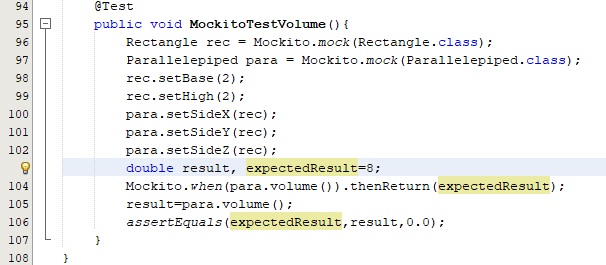
# 2ºMock Test

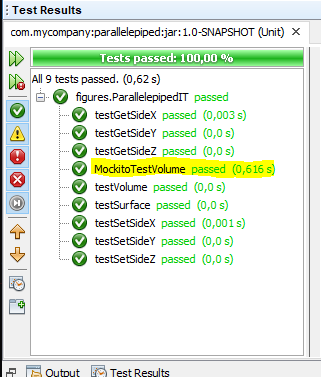
To use Mockito First we installed this library in NetBeans, for that we went to Tool-> Libraries -> New Library and We imported the jar we downloaded.

Remember always to import the library you are going to use:



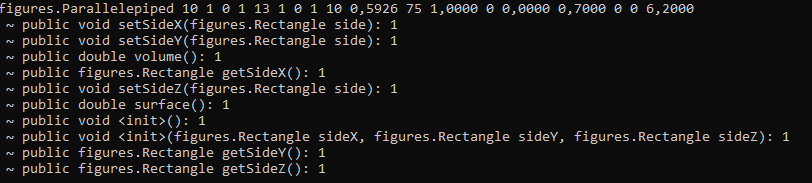
And then we applied a Mockito test in one of our methods (the volume of the parallelepiped:





# 3ºMeasure the Code

To Measure the code, we installed and used the ckjm lribrary, the wewent to the terminal and we used the command:



These different values are the:

WMC - Weighted methods per class

A class's *weighted methods per class* WMC metric is simply the sum of the complexities of its methods. As a measure of complexity we can use the cyclomatic complexity, or we can abritrarily assign a complexity value of 1 to each method. The *ckjm* program assigns a complexity value of 1 to each method, and therefore the value of the WMC is equal to the number of methods in the class.

DIT - Depth of Inheritance Tree

The *depth of inheritance tree* (DIT) metric provides for each class a measure of the inheritance levels from the object hierarchy top. In Java where all classes inherit Object the minimum value of DIT is 1.

NOC - Number of Children

A class's *number of children* (NOC) metric simply measures the number of immediate descendants of the class.

CBO - Coupling between object classes

The *coupling between object classes* (CBO) metric represents the number of classes coupled to a given class (efferent couplings and afferent couplings). This coupling can occur through method calls, field accesses, inheritance, arguments, return types, and exceptions.

RFC - Response for a Class

The metric called the *response for a class* (RFC) measures the number of different methods that can be executed when an object of that class receives a message (when a method is invoked for that object). Ideally, we would want to find for each method of the class, the methods that class will call, and repeat this for each called method, calculating what is called the *transitive closure* of the method's call graph. This process can however be both expensive and quite inaccurate. In *ckjm*, we calculate a rough approximation to the response set by simply inspecting method calls within the class's method bodies. The value of RFC is the sum of number of methods called within the class's method bodies and the number of class's methods. This simplification was also used in the 1994 Chidamber and Kemerer description of the metrics.

LCOM - Lack of cohesion in methods

A class's *lack of cohesion in methods* (LCOM) metric counts the sets of methods in a class that are not related through the sharing of some of the class's fields. The original definition of this metric (which is the one used in *ckjm*) considers all pairs of a class's methods. In some of these pairs both methods access at least one common field of the class, while in other pairs the two methods to not share any common field accesses. The lack of cohesion in methods is then calculated by subtracting from the number of method pairs that don't share a field access the number of method pairs that do. Note that subsequent definitions of this metric used as a measurement basis the number of disjoint graph components of the class's methods. Others modified the definition of connectedness to include calls between the methods of the class. The program *ckjm* follows the original (1994) definition by Chidamber and Kemerer.

Ca - Afferent couplings

A class's afferent couplings is a measure of how many other classes use the specific class. Coupling has the same definition in context of Ca as that used for calculating CBO.

Ce - Efferent couplings

A class's efferent couplings is a measure of how many other classes is used by the specific class. Coupling has the same definition in context of Ce as that used for calculating CBO.

NPM - Number of Public Methods

The NPM metric simply counts all the methods in a class that are declared as public. It can be used to measure the size of an API provided by a package.

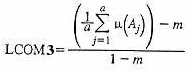
LCOM3 -Lack of cohesion in methods.

LCOM3 varies between 0 and 2.

m - number of procedures (methods) in class

a - number of variables (attributes in class

µ(A) - number of methods that access a variable (attribute)



The constructors and static initializations are taking into accounts as separately methods.

LOC - Lines of Code.

The lines are counted from java binary code and it is the sum of number of fields, number of methods and number of instructions in every method of given class.

DAM: Data Access Metric

This metric is the ratio of the number of private (protected) attributes to the total number of attributes declared in the class. A high value for DAM is desired. (Range 0 to 1)

MOA: Measure of Aggregation

This metric measures the extent of the part-whole relationship, realized by using attributes. The metric is a count of the number of data declarations (class fields) whose types are user defined classes.

MFA: Measure of Functional Abstraction

This metric is the ratio of the number of methods inherited by a class to the total number of methods accessible by member methods of the class. The constructors and the java.lang.Object (as parent) are ignored. (Range 0 to 1)

CAM: Cohesion Among Methods of Class

This metric computes the relatedness among methods of a class based upon the parameter list of the methods. The metric is computed using the summation of number of different types of method parameters in every method divided by a multiplication of number of different method parameter types in whole class and number of methods. A metric value close to 1.0 is preferred. (Range 0 to 1).

IC: Inheritance Coupling

This metric provides the number of parent classes to which a given class is coupled. A class is coupled to its parent class if one of its inherited methods functionally dependent on the new or redefined methods in the class. A class is coupled to its parent class if one of the following conditions is satisfied:

 One of its inherited methods uses a variable (or data member) that is defined in a new/redefined method.

 One of its inherited methods calls a redefined method.

 One of its inherited methods is called by a redefined method and uses a parameter that is defined in the redefined method.

CBM: Coupling Between Methods

The metric measure the total number of new/redefined methods to which all the inherited methods are coupled. There is a coupling when one of the given in the IC metric definition conditions holds.

AMC: Average Method Complexity

This metric measures the average method size for each class. Size of a method is equal to the number of java binary codes in the method.

CC - The McCabe's cyclomatic complexity

It is equal to number of different paths in a method (function) plus one. The cyclomatic complexity is defined as:

CC = E - N + P

where

E - the number of edges of the graph

N - the number of nodes of the graph

P - the number of connected components

These values are displayed in order as we can see here:



# 4ºGitHub Link

https://github.com/guzman9601/SOFT\_CAL2