Test-Driven Teaching – Object Oriented Programming

http://java.dzone.com/articles/test-driven-teaching-%E2%80%93-object

In the previous sections we learned how to create a simple Java program and tested it. We mainly did procedural programming, while now we will learn how to better encapsulate functionality and data with the help of object oriented programming (*OOP*). The aim of OOP is mainly to provide better code re-use than it would be the case with procedural programming. With OOP it is easier to write large software and personally I think, it leads to better readable code.
A nice introduction to OOP with Java could be given via:
An object is a self-contained entity which has its own collection of properties (ie. data) and methods (ie. operations) that encapsulate functionality into a reusable and dynamically loaded structure. After a class definition has been created as a prototype, it can be used as a template for creating new subclasses (via extends *keyword) that* ***add*** *functionality. … Applications can request new objects of a particular class on demand via the* new *keyword.*
What you will learn in this section:

- Get closer to object oriented programming
  - we don't want static methods and variables
  - we don't need a reset method for different tests
  - we don't want public properties
  - private, public, protected and package access
- IDE:
  - Refactor: rename
  - Diff-tool to compare these versions
  - refactor public field to private and create getter
- Explain via tests:
  - references (aka pointers) compared to values
  - loops

Get Started

So check out the previous example at the following URL with subversion:
https://timefinder.svn.sourceforge.net/svnroot/timefinder/branches/tdteaching/TestDrivenTeaching2/
Open it in your favourite IDE and we will now apply some necessary refactorings (code changes) to transform a procedural program into a more object oriented one.

Refactoring (static)

What is refactoring? Code refactoring means code changes, which have the aim to make the code better readable or introduce a cleaner design. But refactoring does not change how the applications works.
First: let us get rid of the static keyword:

1. Rename the InitialBlockGame class to BlockMan, for that you have to rename the file name too. This procedure is very easy with an IDE. In NetBeans just place your cursor under the class name right click->Refactor->Rename or just press CTRL R
2. Remove the *static* keyword from all occurences of the BlockMan class, except from the main(String[]) method

3. Remove the entire reset() method. We don't need it, because we simple create a new object for every test
4. The line with 'BlockMan.start();' should be replaced with
5. view sourceprint?
6. 1.BlockMan man1 = new BlockMan();
7. 2.man1.start();
8. or with a shorter version:
9. view sourceprint?
10. 1.new BlockMan().start();
11. Now we need to refactor the test cases and after the test compiles, it should pass as well. This shows one advantage and one disadvantage of test-driven development: you have to refactor all tests after refactoring which costs some time, but at the same time you can run those tests and make sure that all works as before.

Instead of calling the static start method of the BlockMan **class** we can now create a BlockMan object and invoke the start method of this **object**. Now it is simple to create and manage not only one man but several men:

view sourceprint?

```
1.BlockMan man1 = new BlockMan();
2.man1.move(0, 1);
3.BlockMan man2 = new BlockMan();
4.man2.move(0, 2);
```

The first man *man1* is located after this operation at x=0 and y=1 (0; 1) but *man2* is located at a different location (0; 2) after the move method call.

We should have added a test case before some code. So here it is:

view sourceprint?

```
01.BlockMan man1 = new BlockMan();
02.BlockMan man2 = new BlockMan();
03.
04.man1.move(0, 1);
05.man2.move(0, 2);
06.
07.assertEquals(1, man1.y);
08.assertEquals(2, man2.y);
09.
10.man2.move(0, 2);
11.
12.assertEquals(1, man1.y);
13.assertEquals(4, man2.y);
```

(A good but not ideal comparison of Class/Object in the real world: A class could be compared to a rubber stamp and an object to the resulting images on the paper.)

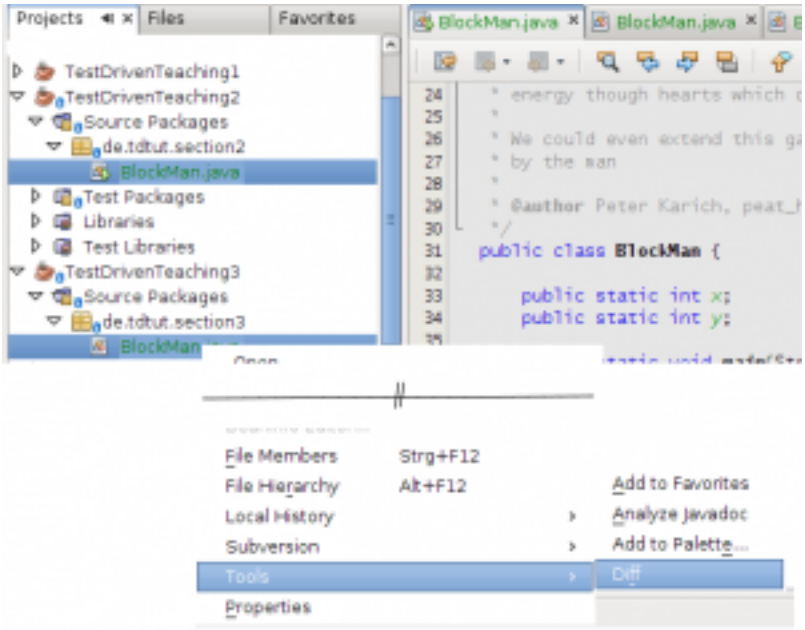
Additionally we should change the *public* access of the x and y variable. What does *access* mean? We learned in the previous post that we can use variables like x only in its block and sub-blocks, where it was defined. Now we used *public* before the variable declaration so that we can access (or 'see') the variable from all external code, which is called *world* in the following table. A short overview of this can be given:

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y

protected	Y	Y	Y	N
no keyword == package access	Y	Y	N	N
private	Y	N	N	N

This table was taken from sun, where you can get more information about this topic.
With *public* acces we can retrieve y (the same is true for x)
view sourceprint?
1.assertEquals(1, man1.y);
and even initialize y:
view sourceprint?
1.man1.x = -100;
This is not good in the most cases and as you see here: the y value is not in its correct domain [0; 10). External code could change the variables without knowing the bounds or sth. else. E.g. the move-method is a better approach to change the variables x and y.
So, just make all variables private and provide **public methods** to manage initialization and retrieval of such a variable via setter and getter. Instead 'public int x;' you should do:
view sourceprint?
01.private int x;
02.
03.// the getter is used for the retrieval
04.public int getX() {
05.return x;
06.}
07.
08.// the setter is used for the initialization
09.public void setX(int someX) {
10.x = someX;
11.}
Now you can simply remove the setX method and 'external code' is not able to change x without the appropriate move method.
Now x and y variables are sometimes called properties. To create such property-access-methods (getter and setter) with NetBeans press ALT INS and select Generate 'Getter and Setter…'
Now we are near the finish of refactoring so check out the code of this **third** section to compare your refactored code with my code:
https://timefinder.svn.sourceforge.net/svnroot/timefinder/branches/tdteaching/TestDrivenTeaching3/
More details on subversion was given in the second section.
To compare the files go to the Projects window and select both files (hold CTRL). Then right-click->Tools->Diff



You will notice that I introduced two methods too: moveX and moveY to make our class easier to use. Also the border values could be now accessed from outside via getBorderX and getBorderY

Look at Unit Tests

In the unit tests of the third example
https://timefinder.svn.sourceforge.net/svnroot/timefinder/branches/tdteaching/TestDrivenTeaching3/
the following things will be further explained:

- Value vs. reference (‘pointers’) are explained in the ValueOrReferenceTest class
- Loop constructs in the LoopExamples class

From http://karussell.wordpress.com