Development manual for TicTacToe

All new employees should read this

Everyone should clone into the repository with the `git clone` command. Click here for the repository link.

Introduction

This n00bs project is the making of the popular game of Tic Tac Toe. If you are not familiar with Tic Tac Toe, you can find the rules of the game here.

What do you need for the project?

- GitHub account.
  - You need to be a member of the n00bs-Hugb2016 organization.
- Java 1.8.0_101 or newer (the version can be found with the command `java -version`).
  - Can be installed here: Click me!.
- Git Bash needs to be installed if you're working on Windows.
- Account at Advania's Qstack (received from project manager).

Test Driven Development

This project uses test driven development (TDD). For each function of the project, start by writing the test for the function and then go into writing the code. Go back to the test cases and add all the tests you can think of and go back to the code. Repeat until all possible tests have been written, coded and passed. After a test is passed, go back to the code and see if you can refactor something. Remember our mantra: "Red, green, refactor, repeat".

Branching

The project has three major branches; *master*, *develop* and *featureDoc*. All other branches are workingstations.

master is the main branch where the final project will be hosted when all has been merged.

develop is the branch where all working code is stored. This is the place where all changes of code should be pushed on to.

featureDoc is a branch that stores the documentation for the project.

When making a major change to the project, create new branch. When everything has finished on the branch and it is time to merge to *master* or *develop*, create a pull request (see how to create a pull request here and one of your team members will go over the changes and accept or decline the request.

Commit comment

When pushing a code that has been coded with pair programming, set the first initial of your name and your partner's in the beginning of the commit comment.

Example: If Tinna and Drífa pair program, their comment would look like this: `"T&D comment"`

Coding rules

1. Naming
   - Naming functions:
     - Names of functions should be written in *camelCasing*.
       - Example: `startGame();`
     - Names of functions should be verbs.
   - Naming variables; private and public:
     - *Private* variables should have underscore in front and be written in *camelCasing*
       - Example: `var _humanPlayer;`
     - *Public* variables should be written in *camelCasing*
       - Example: `var computerPlayer;`
   - Naming classes:
     - Should be nouns, in singular.
     - Should be describing for their content.
     - Should be written in *PascalCasing*
       - Example: `GameController`
2. Code architecture
   - Declarations:
     - Only one declaration should be in each line of code.
       - Example: `var player;`
       - Example: `var winner = getWinner();`
   - Declarations of variables:
     - Should be in the beginning of each function.
     - Arranged by type first, then alphabetically.
       - Example:private getWinner(){`int numOfWins = 0;`

         `string pleyer1Name = "Bob";`

         `string player2Name = "Grub";`
         }
   - Brackets
     - Brackets are always used after a conditional statement, even though it is only around one expression, to maintain readability.
       - Example: if (*condition*) { `foo();`

}

- Indentation
  - Code should be indented with four character spacings which equals the push of the "tabs" button.
- Commentation
  - When writing a doc comment that should appear in the javadoc, it should be written in HTML and must precede a class, field, constructor or method declaration. For more details you can visit How to write doc comments for the javadoc tool
    - Example: /**
      - this is
      - a doc comment */ getWinner() {

code;

}
  - When writing a long comment, use the multiline commentation according to Java rules. This should always be done above the subject being commented on.
    - Example: /* comment
      that
      is
      very
      long */ getWinner() { `code;`
      }
  - When commenting a single line comment, use the single line commentation according to Java Rules. This should always be done above the subject being commented on.
    - Example: // comment getWinner() { `code;`
      }
- Character spacing
  - Between a condition and a opening parenthesis there should be one space.
  - Between the parenthesis and the condition, there should be no space.
  - One space should separate classes and variables from operators, unary operators excluded.
    - Example:

```
a = b + c;
a++;
```

- Line spacing
  - One empty line should separate the definitions of two functions.
  - One empty line should separate the final variable declaration from the rest of the function's code.
  - Two empty lines should separate the definitions of classes.

3. Cooperative Control
   - Coordination of code

- Consult the rest of the programming team before starting to change a file.