Basic Programming: Project

Practical Guidelines

- Students work on the project **individually!** Code duplication between students will result in severe penalties for all parties involved; no code copying from online sources allowed.

- Questions regarding the project assignment should be asked in the Toledo forum or during the exercise sessions. Only these questions will be answered; we will not help with technical questions (i.e. fixing bugs in your code).

- The deadline for the final project submission is **Wednesday, December 30th 2020, 23:59**. No projects received after this date will be accepted.

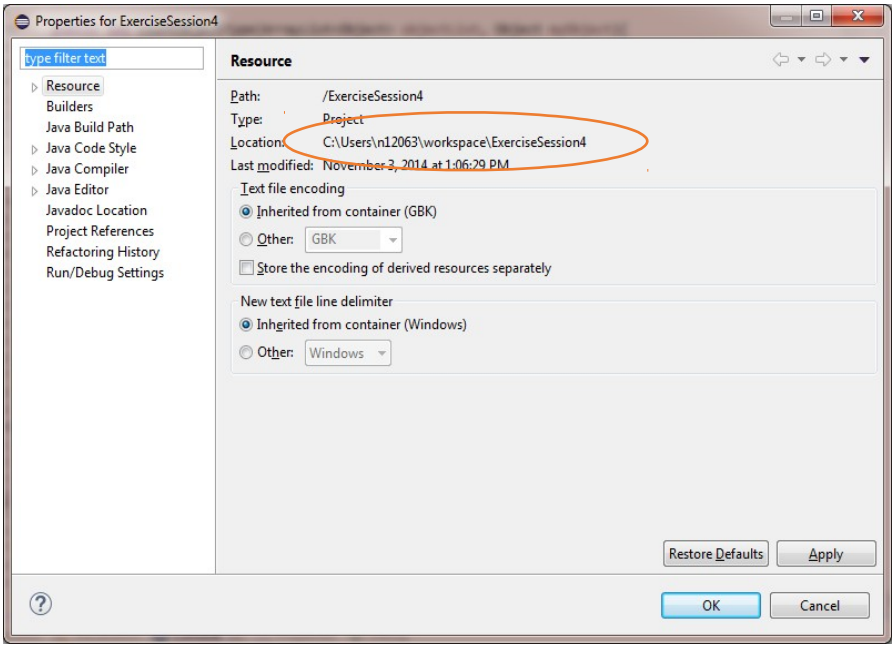
- An electronic version of your project should be submitted on Toledo, the module to do so will be created by the end of the practice sessions and this will be announced via Toledo as well. As discussed during the first class, multiple submissions will be allowed. However, Toledo should not serve as your daily backup service, and hence **the number of submissions allowed will be restricted to 5! No exceptions** will be made (regardless of any reasons you might have); the idea is that you upload your project code when it's complete and fully functional, but you can still submit corrections in case you find a bug or another type of problem. Projects that are late will be discarded; **the final submission will be used for grading and for the oral defense** (again, **no exceptions**).

- The Toledo submission system will indicate a mark of 10 as the system requires a maximum mark to be provided. As indicated in the introductory presentation for the course, the Java project counts for 10 marks out of 20 for students of the Master of Artificial Intelligence and the Master of Information Management, and for 7,5 marks out of 20 for all other students (i.e. those that also take the R part of the course; hence the marks will be rescaled).

- You must include the following components in your project submission, preferably in a zipped file with **surname and first name** as the name of the file: **surname_firstname.zip:**
  - Java project source code copied from your Eclipse workspace, including all the required images (when applicable) for your project to run in the correct location; no .class files or .jar files should be submitted; your project should contain exactly **one main method**.
  - Short report describing your project (**maximum** 4 pages, see guidelines) with strong focus on the structure and design of your code; no UML diagrams should be provided.

- Prepare your report and code (names, comments, etc.) in **English**.

- To copy your project from Eclipse:
  - From Eclipse, right click on your project, select Properties. Under Resource, you will find the location of the project folder.
  - Navigate to this location on your computer and include a copy of the entire project folder in your project submission.



- To copy your project from IntelliJ:
  - In IntelliJ, you can see the location of your project in the Project view/panel (next to the project name that's listed in bold).
- It's a good test to see if you can copy your project to another computer and see if you can run it there from within an IDE such as Eclipe, IntelliJ or Netbeans.

Coding Guidelines

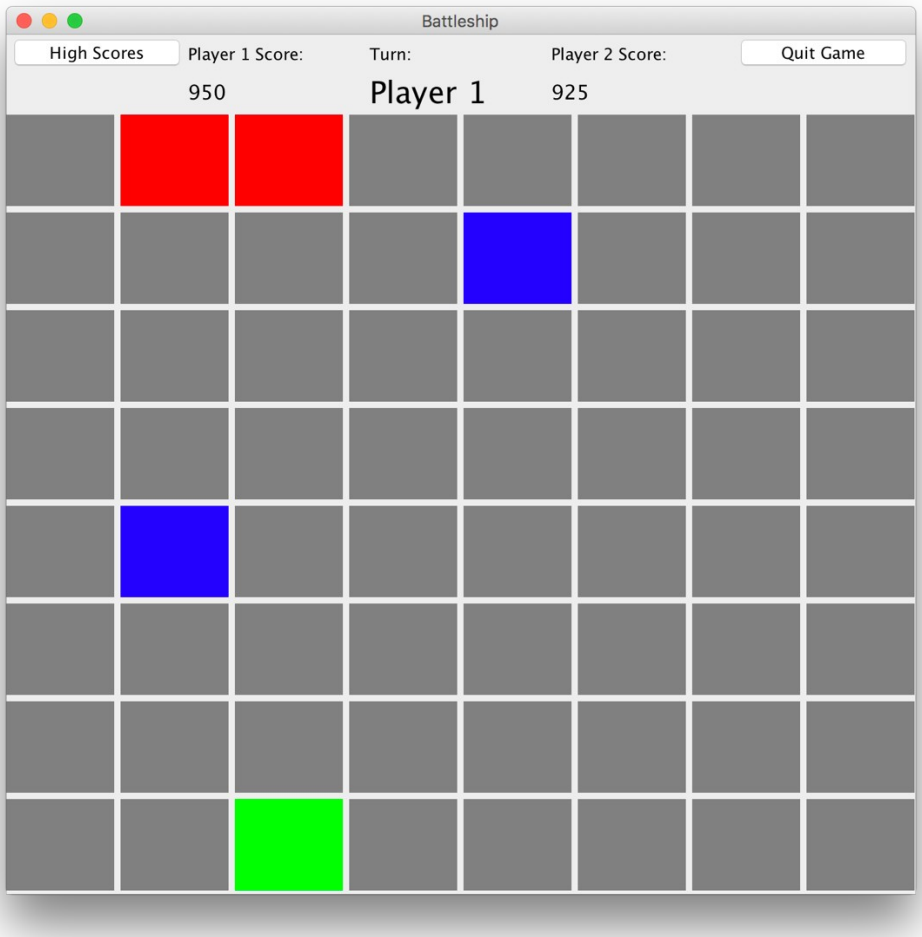
The code guidelines below are a selection from a more extensive set of guidelines provided by Oracle (can be found on the oracle website). We expect you to follow at least these guidelines:

- Use a separate file for each class and interface
- Assign classes that are not related to each other to different packages
- Write at most 1 statement per line
  ```
  int x = 5;      //(good)
  int y = 10;
  ```
  is preferred over
  ```
  int x = 5; int y = 10;    //(bad)
  ```
- Naming conventions
  - **Classes**
    - Class names should be nouns, in mixed case with the first letter of each word capitalized. Try to keep your class names simple and descriptive. Use whole words—avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form).
      - Start with upper case, e.g.: `Person`, `Car`, `Game`, `BMICalculator`
  - **Interfaces**
    - Start with capital letter, e.g.: `Nameable`, `Capable`, `Pettable`
  - **Methods**
    - Names are usually verbs, and should start with lower case and use camelCase if it consists of multiple words, e.g.: `getValue()`, `add()`, `performSomeAction()`
  - **Variables**
    - Variable names should be short (where possible) yet **meaningful**!. Choose variable names which indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are: `i`, `j`, `k`, `m`, and `n` (for `int`) and `c`, `d`, and `e` (for `char`).
    - Name starts with lower case and use camelCase if it consists of multiple words, e.g.: `name`, `dateOfBirth`, `ageLimit`, `stomachContent`
  - **Constants**
    - Should be all uppercase, use "_" to separate words, e.g.: `PI`, `DAYS_IN_WEEK`, `GRADE_TO_PASS`
- Provide comments for each variable, method, and class so that it is 100% clear to the reader of your code what its purpose is.
- Use proper indentation for your code. Every time you start a block of code (when making a class/method/if-else if-else/switch/…) indent the code within the block.
  - TIP: Eclipse and IntelliJ can do this for you.
    Eclipse: Select the code you want to indent properly > right-click > source > Correct Indentation
    IntelliJ: Select the code, Code > Auto-Indent Lines

Project Guidelines

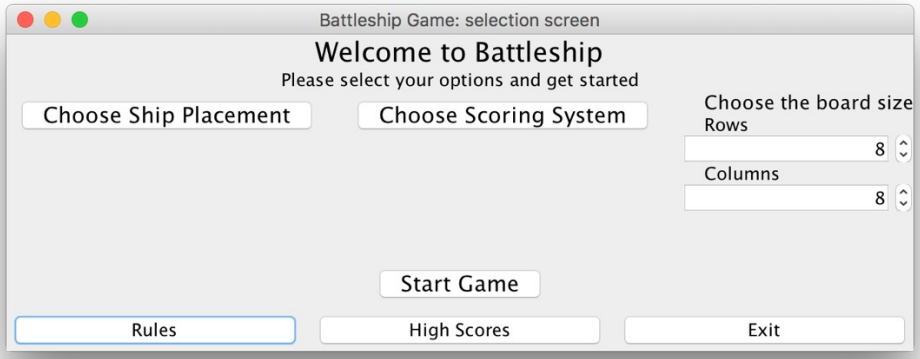
For this project, you will create an implementation of the popular board game Battleship. Battleship is a guessing game for two (human) players, no AI or computer opponents are required for this project assignment. The most basic version of the game is played on a square board of dimension 8-by-8 consisting of 64 grey-colored tiles. A number of ships are located on the board. During each player's turn, the player clicks on a grey tile and it will be revealed whether she/he hit a ship or the water. If the player hit the water, the grey tile turns blue. If the player hit a ship, the tile turns a specific color, depending upon the color of the ship that has been hit. There are four different types of ships, with each ship having a unique name, size, and color (you can decide upon the colors yourself or even use images, although be warned that this might complicate your implementation considerably):

- Carrier: this ship is five tiles long
- *Battleship*: this ship is four tiles long
- Submarine: this ship is three tiles long
- *Destroyer*: this ship is two tiles long



When a player hits a ship, they receive a set amount of points, depending on which ship is hit. It is up to you to determine the points-per-hit for the specific ships, but the points-per-hit must be different for each ship. In any case, a player receives double the usual amount of points if their hit results in a ship sinking (i.e., their hit means that all tiles corresponding to the ship have now been hit). The two players take turns clicking grey tiles until all ships have been sunk, at which point the game ends. The player with the most points at the end of the game wins the game.

Several options need to be provided to the users at the start of the program, before the game begins. These options appear on a starting panel that is shown when the application is started.



First, there needs to be a “rules” button that can be called from within the game that explains the rules in a new window. Second, it should be possible to choose different board sizes; incase this would prove overly difficult, focus on making sure your implementation makes use of the default 8x8 board size. Third, two options for scoring systems must be offered: one which assigns the same points-per-hit to each player, and one which provides a unique points-per-hit to each player to accommodate going first or second to start the game. Fourth, the placement of the ships on the board should either be determined randomly (important: each tile on the board can only be used once), or through a user-provided text file. The text file’s first line will be a positive integer that specifies the dimension of the board. After that, the file will list one ship per line, and each line will start with the name of the ship, followed by the coordinates for the tiles corresponding to the location of the ship. The name and coordinates are each separated from one another by a semicolon. A coordinate is specified by two numbers separated by a “*” with the first number representing the row and the second number representing the column. For example, “2*3” represents the tile at row 2 and column 3 of the board. An example of such a user-provided text file is as follows:

```
8
Carrier;3*2;3*3;3*4;3*5;3*6
Battleship;5*6;6*6;7*6;8*6
Submarine;5*2;6*2;7*2;
Destroyer;1*7;1*8
```

If the user-provided text file has errors (for example: overlapping ships, tile coordinates and board dimension specifications that are inconsistent, incorrect ships names, too few or too many ships, etc.), an error message must appear on screen and the game cannot be started. Finally, there should also be a “high scores” button that shows a list of high scores of previous game winners; you can make use of a fixed file name of your choosing to store this information.

Summary of the key requirements:

- A playing window that contains the actual level / game, with an accompanying score board that indicates each player’s score, which player currently has to make a move, access to the high scores list and the possibility to quit the game.
- The playing field/window consists of a grid of tiles:
  - Grey tiles indicate tiles that have not yet been turned over
  - Blue tiles indicate tiles that have been turned over and correspond to a “missed shot” that did not hit a ship and ended up in the ocean
  - Red, green, yellow, and white tiles (for example) that indicate parts of ships that have been “hit”
  - The locations of the ships should be able to be assigned in two ways: randomly, or via a user-provided text file
  - The game ends when all ships have been sunk
- Scoring systems:
  - Each player receives a set amount of points for hitting a specific ship, and the usual amount of points received for a specific player hitting a specific ship is doubled if the hit sinks the ship. The points-per-hit must be different for the different ships. There must be two scoring systems: in the first, the points-per-hit are the same for each player, and in the second, the points-per-hit are different for each player in order to compensate the player going second.

Key points when implementing the assignment:

The structure of your code is the most important evaluation for the project. This means you should take care to include as many of the object-oriented concepts covered in the course whenever applicable. However, use them in intelligent ways, just forcing something for the sake of including it does not add value to your project.

Watch out for plagiarism! Online you will find similar game implementations which you may use for inspiration, but you must write your own code! Note that many code examples you can find online are poorly written and/or poorly designed.

A nice GUI and a creative setting will count towards your final grade, but is not required to pass. On the other hand, an exceptional GUI without well-developed underlying logic is not sufficient. So, to summarize: a well-designed but more limited project will lead to a higher score than a poorly designed but very graphically appealing interface since we want

you to focus first on the logic! We realize that design patterns such as Model-View-Controller are not part of the course material, and you are not meant to study this material for the project; you are however required to think about properly structuring your project code.

Finally, aim for a well-designed / well-structured project with cleanly written code. As stated before, clearly state in the project if you were unable to provide certain requested aims of the project and provide some information (what did you try and where did it go wrong).

Report Guidelines

You should prepare a short report of **maximum 4 pages** detailing important components of your project. It should include the following information:

- Provide an explanation of your game, including the story/setting and the rules of play; a new user unfamiliar with your game should be able to read this and understand how to play.
- Write a short description of every class, indicating what functionality is included in each class.
- Describe the relationship between your classes (this may be text and/or a simple diagram, but not a UML diagram), for example, inheritance relationships or method calls from one class to another.
- If you are unable to implement a certain part of the game, we encourage you to make a sensible decision, be upfront about this decision (i.e. explain it in your report) and explain what the main difficulty was. Additionally, **if you encounter implementation difficulties or time constraints, it's better to fully and correctly implement a subset of the functionality** rather than to implement small parts of each of the targeted program parts.
- Discuss what you think are the strengths and weaknesses in your project, but focus on code design when doing so and not on how the application looks. Describe any difficulties you faced while working on the project.