# 02324 Advanced Programming F24
# Mandatory Assignment 2

**TO BE SUBMITTED ON DTU LEARN - SEE DEADLINES ON DTU LEARN**

You are strongly recommended to work in groups, but you must clearly identify the contributions of each group member, and you will be jointly responsible for the finished report. Your group must be registered on DTU Learn.

The assignment must be submitted electronically as a PDF report. It must clearly mention who are the group participants (use DTU student unique IDs). Materials that are handwritten and scanned **will not be accepted**.

## Conceptual Modeling

In Assignment 1, you were asked to focus on class diagrams related to the domain model of RoboRally.

In this part of the assignment, your task is to complete the domain model as well as to focus on the clear representation of the requirements for your version of RoboRally game from the user's perspective. Specifically, you are asked to

1. **Provide the first brief description** (only a few paragraphs) of your RoboRally game project. It should focus on providing some background on the game (by relying on your understanding of the rules) and your initial vision of the project (you can also base it on the code provided for this assignment as well as the completed domain model and features).
2. **Describe the main use-cases** (how the user perceives them) for your RoboRally game. Every use-case must come with a textual description, where actors, success and failure scenarios are clearly identified. You may represent such use-cases graphically by resorting to designated UML diagrams.
3. **Identify and prioritise features of your game**. Use the MoSCoW categories (Must, Should, Could, Won't) for the prioritisation. You have to provide a short description of each feature and every MoSCoW category.
4. **Produce** at least one **activity** and one **state diagrams** representing (activity- and behaviour-related) concepts of the RoboRally domain model.
5. **Describe non-functional requirements** such as usability, maintainability etc.

Notice that the outcome of this part of the assignment is not final: you might need to have a few more iterations over it before submitting the final project report.

# Programming

This part of the assignment focuses on the programming-related tasks.

## Preparations

Download and familiarise yourself with the initial RoboRaally project (`RoboRally-roborally-1.1.0-java17.zip`) from the Assignments tab on Learn.

The software uses JavaFX UI framework. The RoboRally initial project includes Maven setup to work with JavaFX, but before you can run the project you will have to change the run configuration:

6. Start IntelliJ and open this project (with "`File→Open...`").
7. Edit the configuration with "`Run → Edit Configurations...`", select "`RoboRally`" configuration, and set a module path to your local folder where JavFX is installed ( `--module-path "\path\to\javafx-sdk-17\lib" --add-modules javafx.controls,javafx.fxml`)
8. Try to build and start the project (the main class is RoboRally in the package dk.dtu.compute.se.pisd.roborally.
9. If you have problem with building the software and maven's JavaFX installation, here you can find detailed information how to install it manually and setup the project properly: https://openjfx.io/openjfx-docs/#IDE-Intellij

## Javadoc task

Inspect this project and find out where the different components are located and how they play together. Document all the main classes and public methods with JavaDoc. For more information on Javadoc see `https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html` (an introduction to Javadoc by Oracle) and/`https://www.jetbrains.com/help/idea/javadocs.html` (an introduction to Javadoc by Jetbrains)

## Programming tasks

For all the programming tasks, **you will have to respect the following requirements:**

1. Every implemented method/feature has to be documented by including information for the following tags: `@auhtor`, `@param` and `@return`. Should there be any need in including information for other tags, please feel free to do so.
2. Every implemented method has to be tested using acceptance tests, including the descriptions of scenarios. This does not apply to cases for which tests are explicitly supplied.

## Task 1

Test whether the project works by starting it through `StartRoboRally` class.

When you initiate a new game, a game board with players' checkers pops up, and there are also some fields for programming cards and buttons. But when you press the game board or buttons, nothing happens. Moreover, at the bottom there is a progress bar which shows whose player's turn it is right now.

You're going to change this by implementing the following functionalities.

**Functionality 1.** When the user clicks on a field where there is no player (or a little more accurate a player's robot), the current player's robot must move to that field; and afterwards it should be the next player's turn.

The software that you were provided is built up already in such a way that when the player clicks on a field, then a method `moveCurrentPlayerToSpace` in `GameController` class is called, but this method does not do anything. Here you have to program exactly what it says in the `TODO` comments section by using the model classes `Board`, `Player` and `Space`. To do so, one must understand what information these model classes offer and how this information can be changed (this relates to the outcomes of your Javadoc task).

Once you have implemented this method, you must test whether your solution works as intended. As an intermediate solution, use the debugger to find the error. Do not continue with the next step until this feature is working as intended.

**Functionality 2.** So far, in the status bar you can only see if it is the player's turn. You also need to add a little more information to the status bar (see below) and must show how many moves have been in the game.

Once you have implemented it, you can expand the code that you have programmed in `moveCurrentPlayerToSpace` method of `GameController` so that the counter is counted every time a player moves.

Finally, you need to change the method `getStatusMessage()` in the class `Board` so that it returns string that represents not only the player's name, but also the number of moves that were in the game (using counter). Related parts of the code are marked with `TODO Task1`.

## Task 2

When a new game is started, apart from has been described in Task 1, the game's programming phase is activated, with each player receiving some random command cards (on the bottom line of the card boxes). And players can drag some of them to the five card fields on the top line that represent this player's robot program (register).

Once every player has programmed their robot, you should be able to press "`Finish Programming`". But nothing will happen. The reason for this is that the "`Finish Programming`", "`Execute Program`", and "`Execute Current Register`" buttons are still associated with the controller's method `notImplemented()`.

To solve the above problem, you first need to make sure that all buttons are attached to the respective methods in `GameController`. Then it should be the case that when the player presses "`Finish Programming`", the buttons "`Execute Program`" and "`Execute Current Register`" are activated. And when you press "`Execute Program`", all programs of all robots are executed. When you press "`Execute Current Register`", only the current robot's current map is executed; in this way, you can click step by step throughout the program. And it is recommended to use this step-by-step execution at the beginning.

When you press "`Finish Programming`" and "`Execute Current Register`" you will see which card will be executed; but the commands themselves have no effect. The reason for this is that the methods that represent the different commands in are not implemented yet. Therefore, you should implement the methods that implement the commands of the cards. They are marked in `GameController` with `TODO Task2`.

To test if your program works now, implement a test for each of the four commands. For this, you can use and implement a test method for each command in `GameControllerTest`. Execute all the tests. When all these tests run through, Task 2 is done.