Basic Programming: Final Project

For **all** students enrolled in Basic Programming, there is a project to be completed as part of the evaluation for the course. The project constitutes 7/20 points for the students that also take the R part of the course and 10/20 points for the students that only do the Java part. Students should work on their projects **individually**.

# Practical Guidelines

* Questions about the project should be asked in the Toledo forum or during the exercise sessions. Only these questions will be answered.
* Registration for the project is obligatory. You must send an email before **November 23rd, 2017, 23:59 GMT+1** confirming your participation to the following three email addresses.

[klaas.nelissen@kuleuven.be](mailto:klaas.nelissen@kuleuven.be),

[maria.oskarsdottir@kuleuven.be](mailto:maria.oskarsdottir@kuleuven.be)

[michael.reusens@kuleuven.be](mailto:michael.reusens@kuleuven.be?subject=Basic%20Programming%20Project)

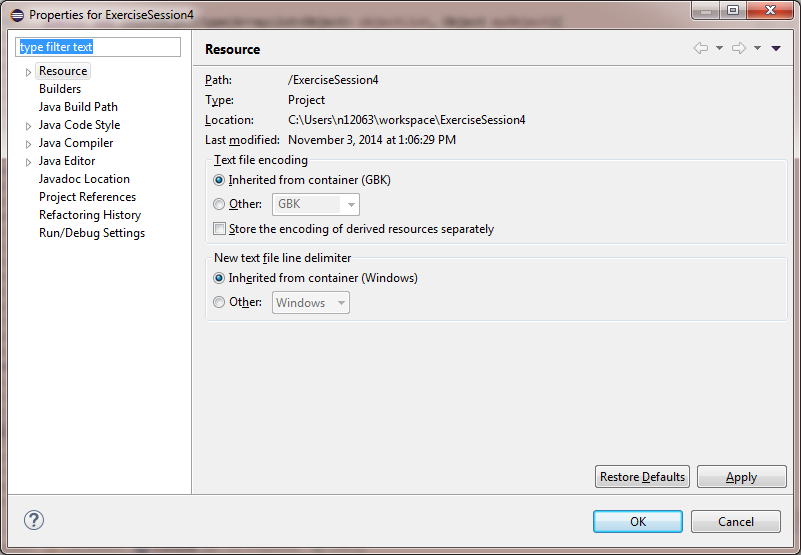
* The deadline for the final submission is **January 2nd**, **2018, 23:59 GMT+1 AND at least 48 hours before your defense, whichever comes first**. No projects received after the deadline will be accepted.
* An electronic version of your project should be sent to all three emails:

[klaas.nelissen@kuleuven.be](mailto:klaas.nelissen@kuleuven.be),

[maria.oskarsdottir@kuleuven.be](mailto:maria.oskarsdottir@kuleuven.be)

[michael.reusens@kuleuven.be](mailto:michael.reusens@kuleuven.be?subject=Basic%20Programming%20Project)

* You must include the following components in your email, preferably in a zipped file with **your name** as the name of the file:
  + Java project copied from your Eclipse workspace
  + Short report describing your project (see guidelines)
* Prepare your report and code (names, comments, etc.) in **English**.
* You will also need to defend your project orally. During the defense of max 20 minutes you will be asked to explain parts of your code and answer Java related questions.
* By December 2nd, 2017 defense registration and moments will be announced on Toledo.
* 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.



# Coding Guidelines

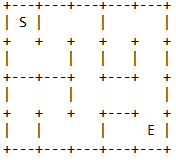
The code guidelines below are a selection from a more extensive set of guidelines provided by Oracle (can be found on [the oracle website](http://www.oracle.com/technetwork/java/codeconvtoc-136057.html)). We expect you to follow at least these guidelines:

* 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 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, populationSize
  + **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 can do this for you.   
    *Select the code you want to indent properly -> right-click -> source -> Correct Indentation*

# Project Guidelines

## A Maze(ing)

For this project, you will create a maze like game where a player tries to get from the starting point to the end point of a maze in as few moves as possible. The walls in the maze are unusual in the sense that some of them can be broken down, some contain doors and some are fake, i.e. the player can walk right through them. There are also hidden objects in the maze, such as keys that can be used to open doors and hammers that the player can use to break down walls. The player does not know about the location of these objects and they are only revealed and added to the player’s possession when he comes across them during his journey through the maze. The game is over when the player reaches the end point. An example of a maze with normal walls, a starting point (S) and an end point (E) is shown below.



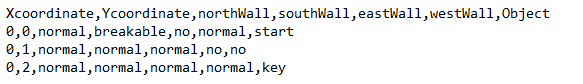
You will need to code the maze, the player and functionality of playing the game. We encourage you to express your creativity when building the game by picking a specific theme, implementing additional constraints or functionalities. However, we do expect you to include several specific components which are designed to make use of the concepts you’ve learned in this course.

#### Minimal Requirements (you may go above and beyond!):

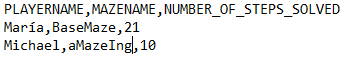
For students also taking R as part of Basic Programming this project counts for a smaller part of your grade. These students are not obliged to implement the minimal requirements that are **bold and underlined**.

##### Your solution should include

1. A maze

* A grid of squares representing the maze.. The dimensions of the grid depend on the information in an input file (see below).
* Each square in the maze has four sides where each side has one of the following characteristics:
  + Open
  + Wall
  + Breakable wall
  + Door
  + Looks like a wall but isn’t (if the player tries to walk through it he can)
* Some squares contain a hidden object with special abilities. These are only revealed when the player enters the square. When a player comes across a hidden object, it is added to his possession. Hidden objects include
  + a key to open doors
  + hammer to break down walls
  + trophy for additional points
  + …
* Exactly one square is the entry point (S) of the maze and exactly one square is the exit point (E).
* Game setup
  + The setup of the game is based on a text based input file,
  + We will provide a few files that can be used for initializing a game.
  + The game should work with any file of the same format.
  +  For example, the file

would result in a maze that looks like

* End of game
  + When the player reaches the end point, his results are written to a ‘High Score’ file, as shown below

1. A player traversing the maze
   * His initial position is the entry point(S) of the maze
   * He can travel through the maze by moving up/down/left/right
   * He can not walk through walls
   * His goal is to reach the end point (E)
   * He can collect and use the hidden objects, open doors, break down walls etc.
   * **Has a move history**
     + **The player has an option to ‘Undo’ a move in which case the game moves back one step This can be done several times. You need to implement a way to trace back the player’s actions.**
2. A text based interface (in the console). It should show
   * A two dimensional representation of your maze, including
     + Entry point (S)
     + End point (E)
     + Different types of walls
       - Breakable/unbreakable
       - Doors
       - Looks like a wall but isn’t one
     + Current position of the player (P)
     + Statistics of the game
       - How many steps has the player taken
       - Which objects are in his possession
3. **Algorithm for automatically solving the maze**
   * **Simple maze AI with at least two different strategies for traversing the maze, e.g.:**
     + **Random walk**
     + **Least amount of steps**
     + **Dijkstra**
     + **Depth/breadth first**
     + **…**

This assignment still leaves you with some room to implement your own rules. Whenever something is not specified clearly in the assignment, you are free to fill the blanks in as you choose, as long as it makes for an interesting game which illustrates your grasp on the programming concepts seen in this course.

#### Key Points:

Although we will provide you with a few input files to initialize the game, it should also work with any input file following the specification.

You can make your own input files and share them on the Toledo discussion board! If each student makes a maze and shares it on Toledo, all students will have more mazes to test their code on.

Watch out for plagiarism! Online you will find maze implementations which you may use for inspiration but you must write your own code!

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 as you can. Be careful to properly define your classes, use inheritance instead of enums and include polymorphism where appropriate. However, use these in intelligent ways, just forcing something for the sake of including it does not add value to your project.

We do not expect you to add a GUI: a textual interface can be enough to earn full credit. Going above and beyond the minimal requirements (e.g. by implementing a GUI) will of course lead to a better grade, however, an exceptional GUI without well-developed underlying code is not sufficient.

# Report Guidelines

You should prepare a report of max 6 pages detailing important components of your project. It should include the following information:

* Provide a **short** explanation of your game
* 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 diagram), for example, inheritance relationships or method calls from one class to another.
* Discuss what you think are the strengths and weaknesses in your project, and describe any difficulties you faced. Provide at least 3 strengths and 3 weaknesses.