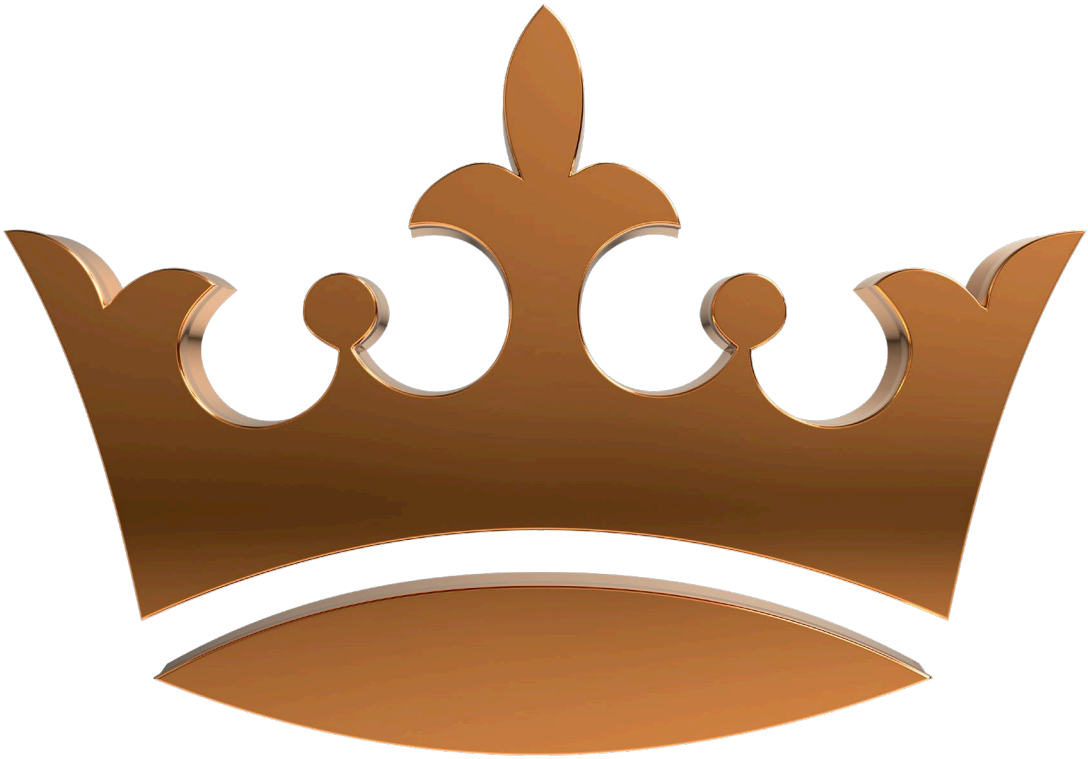
Assignment 3

Due Date

This assignment is to be completed in pairs. The assignment is due at 11:55PM Wednesday May 28th 2025 and should be completed with a partner. You and your partner should work together on all of the design and programming. It should be done using the *pair-programming* methodology and not by division of labour.



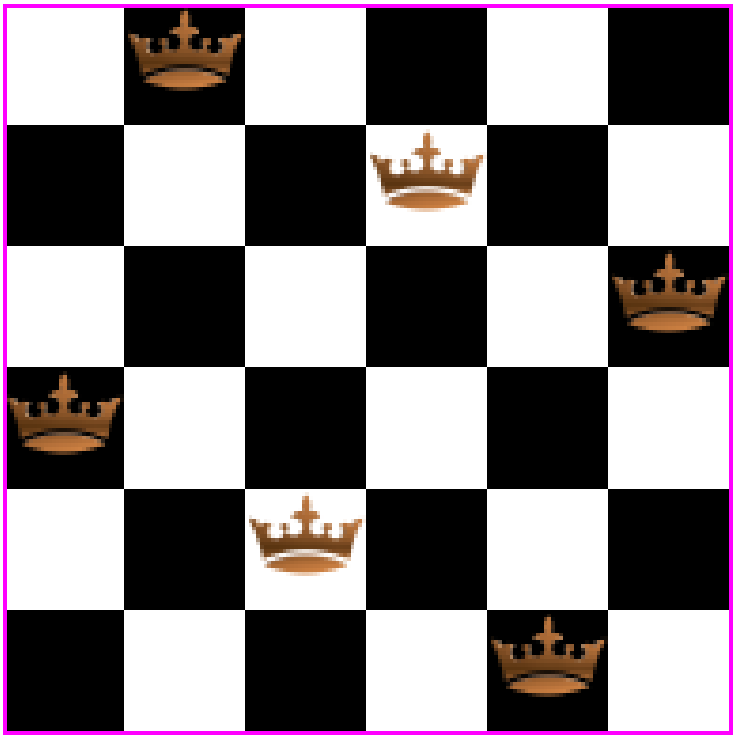
Use the "Assignment 3 Pairs" groups on MyLO to find and register your pair:

- if you already have a partner — start at the bottom of the list of pairs and when you find an empty group, both you and your partner should enrol in that group.
- if you do not have a partner — start at the top of the list of pairs and when you find a group with 0 or 1 member(s), add yourself to that group. If you are the second person to join, you now have a partner and so email your partner to get started.

Context

Famous people don't like to share the lime-light with other famous people. COVID-19 restrictions required social distancing. Table plans for wedding receptions need to have family members that don't get along kept away from each other. N-Queens is a problem from the board game of Chess which has many such real-life applications!

In N-Queens, a board of N rows and N columns must have N Queens placed on the NxN squares of the board such that the Queens don't challenge each other, i.e. there is no more than one Queen on each row, each column, and on the diagonals of each square. An example of 6-Queens is shown below.



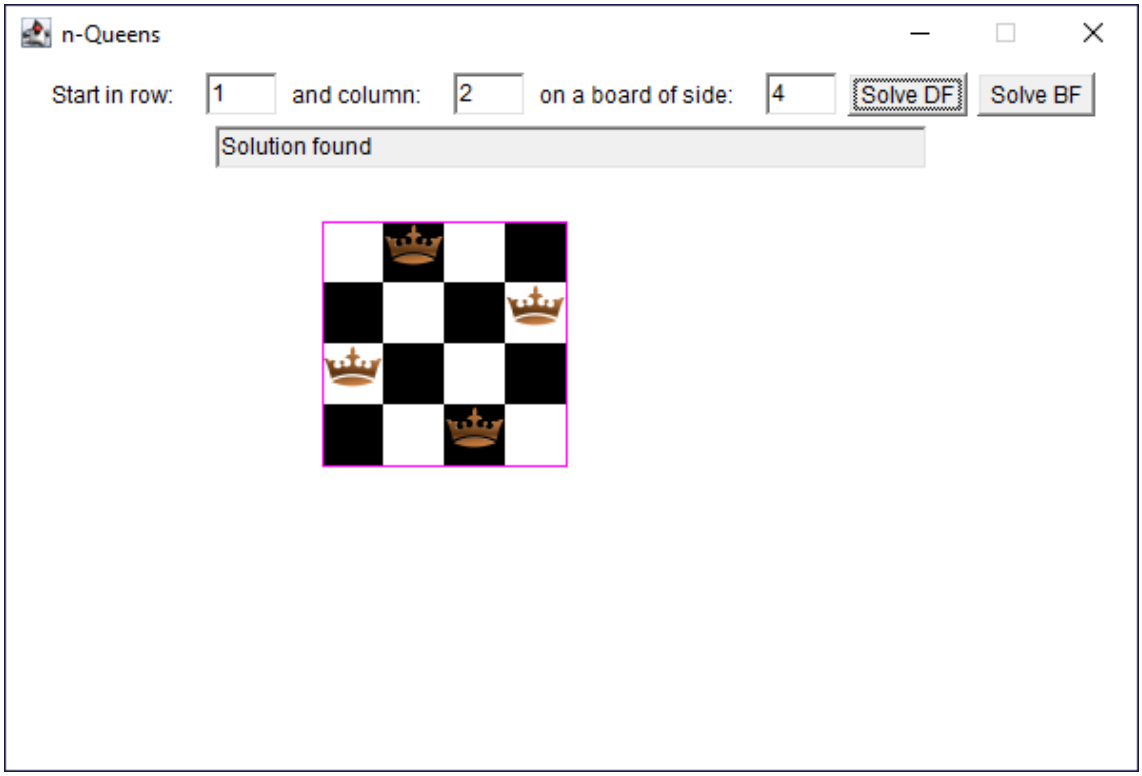
Task

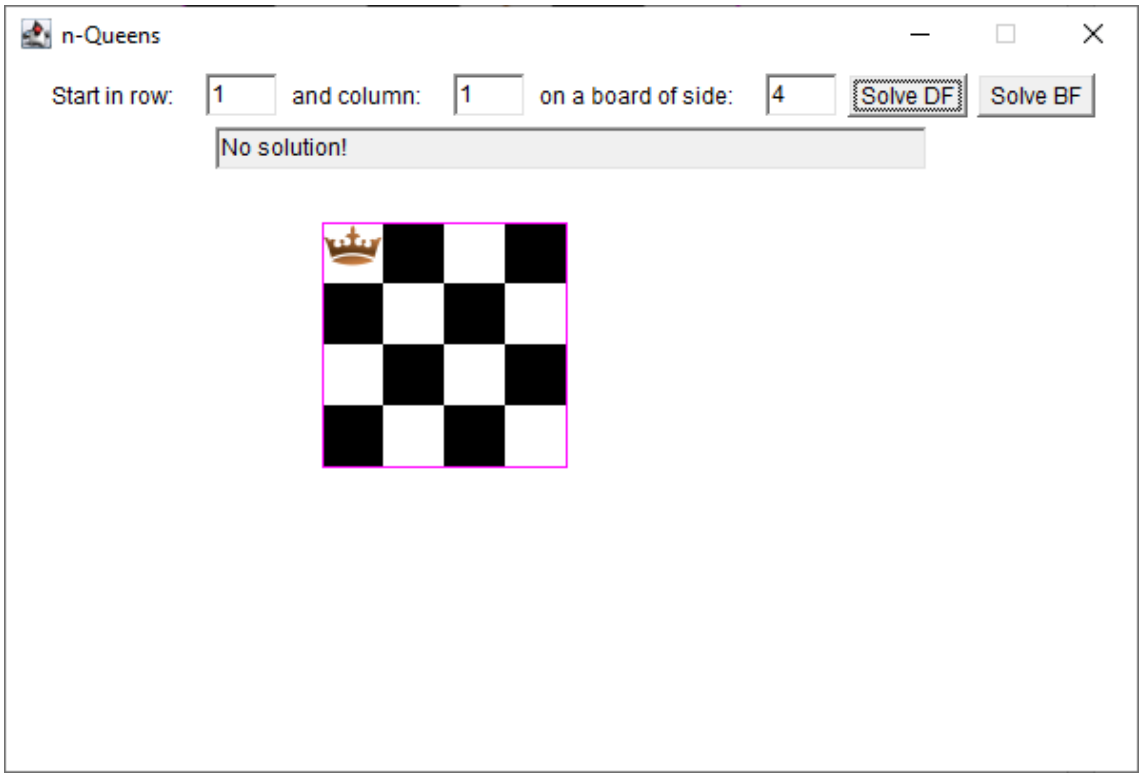
Your task is to implement the N-Queens problem in Java so that the computer will find the solution for you*. Overleaf are the output of two different executions.

The computer will select moves by creating a *game tree*. A game tree consists of all the possible states of the game and in this assignment the computer determines any possible solution to be the best one.

---

* There is insufficient memory available for large grids and/or deep game trees. There's not much you can do about it, you'll know it when you see it—`java.lang.StackOverflowError` — and it's not your problem to fix!





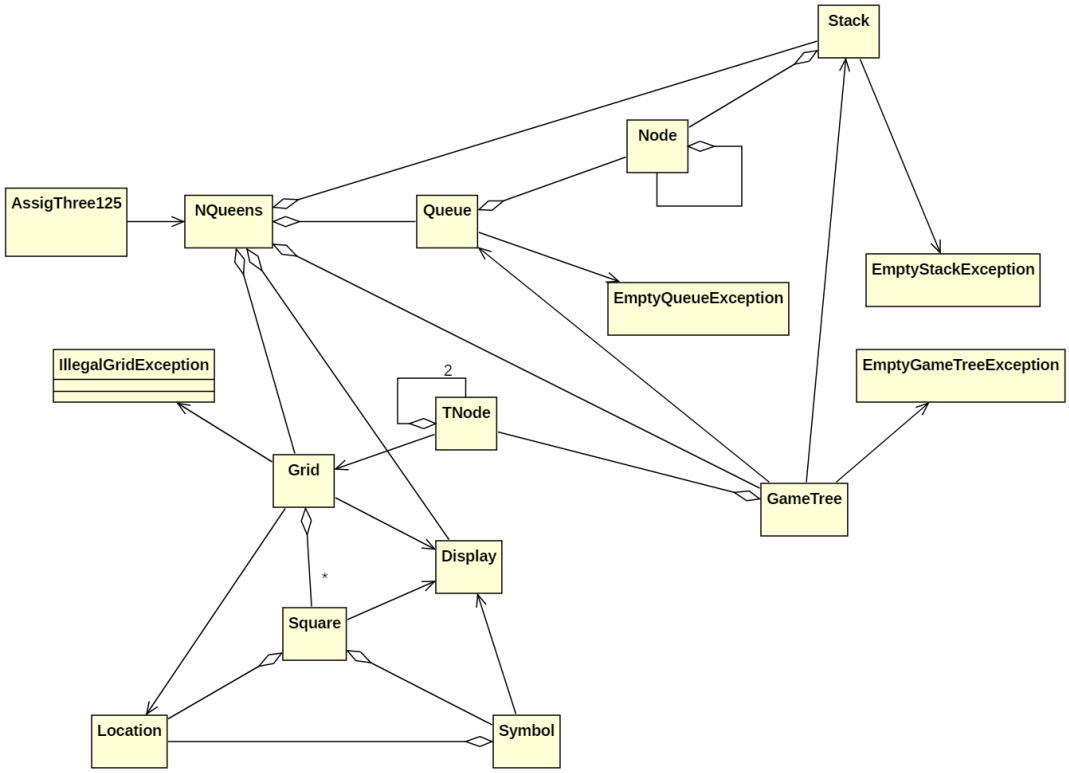
Game trees are general trees in which each tree node is a particular state (i.e. grid) of the game. Each level in the tree possesses as many sibling nodes as there are possible moves from the parent node, i.e. it is technically possible that every node will have NxN child nodes, but as moves are made, the number of possible moves lessens as we move down the game tree.

Each node in our game tree will also have a level number; the root node will be at level 1, its children will be at level 2, their children will be at level 3, and so on. (This gives us a simple way to count how many queens are on each board.)

This assignment uses many ADTs and data structures (trees, arrays, linked-lists, Stacks, and Queues) to solve the game. You should create and traverse the game tree using a stack and queue as intermediate data structures as done in §10 (Slides 226–230) of the lectures.

Program Specification

The overall software architecture is shown below.

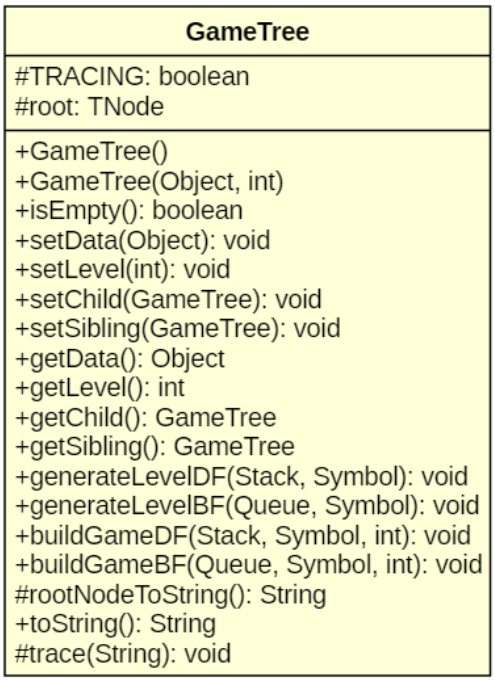


There are eleven interface–class pairs of program files:

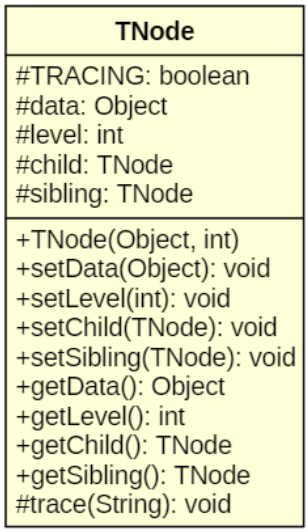
- `NQueens` — the GUI and all the user interactions

NQueens
#TRACING: boolean #LIMIT: int #STANDARD_SIZE: int #dfst: Stack #bfst: Queue #window: Display #game: GameTree #board: Grid #queen: Symbol
+NQueens(String) +paint(Graphics): void +actionPerformed(ActionEvent): void #showStatus(String): void #trace(String): void

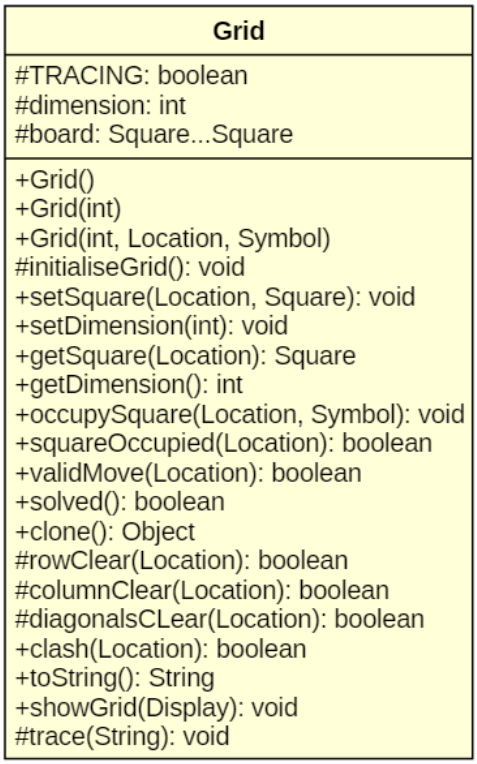
- `GameTree` — the game tree of possible grids in the game



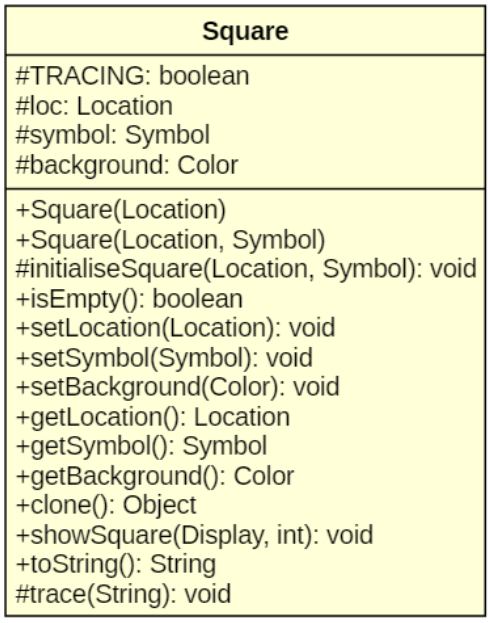
- `TNode` — tree nodes (which will contain grids)



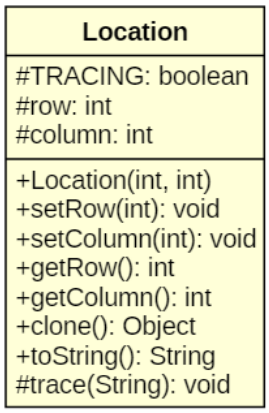
- `Grid` — the collection of squares



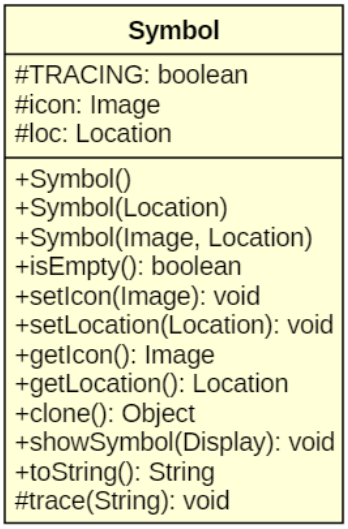
- `Square` — a square on the grid that can contains a symbol



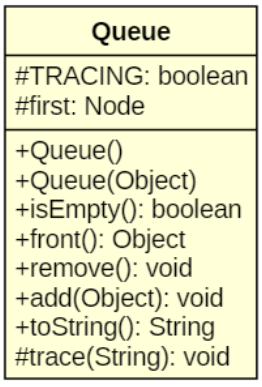
- `Location` — a row and column combination



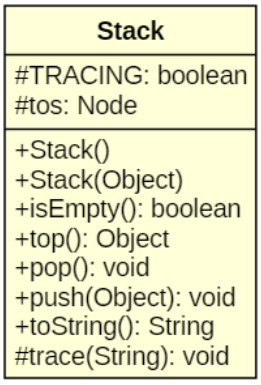
- `Symbol` — either empty, or a queen



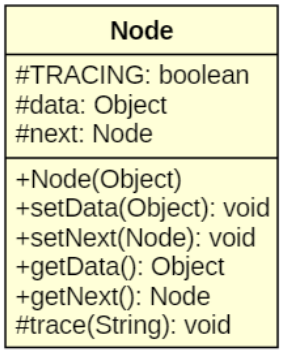
- `Queue` — a queue of game trees yet to be processed (breadth-first)



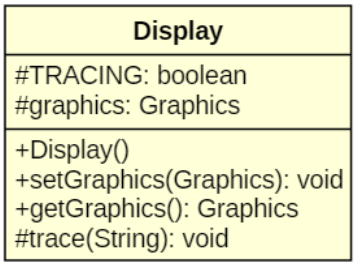
- `Stack` — a stack of game trees yet to be processed (depth-first)



- `Node` — singly-linked-list nodes (which will contain game trees)



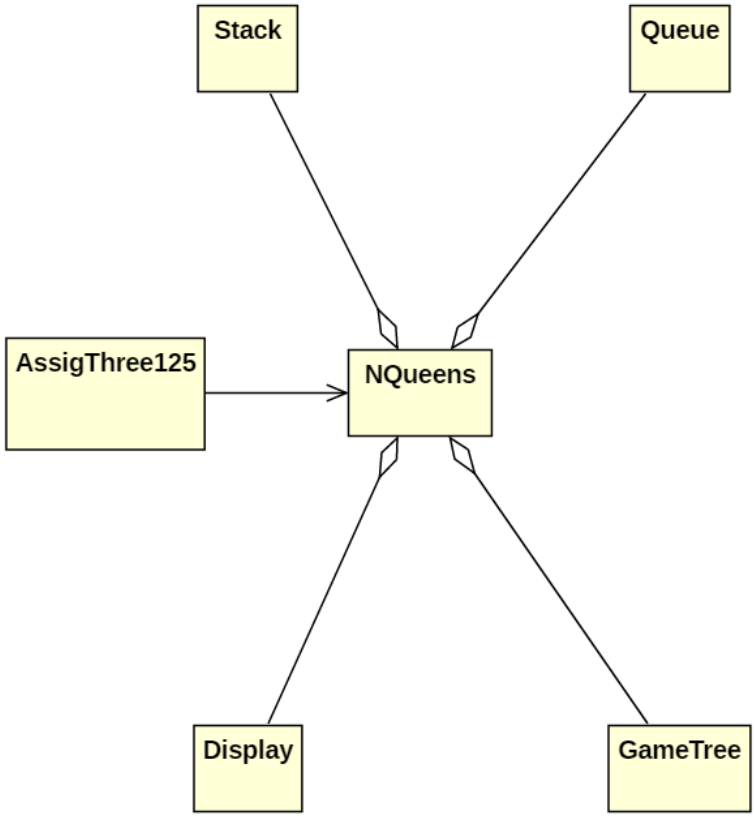
- `Display` — the window in which the app is running



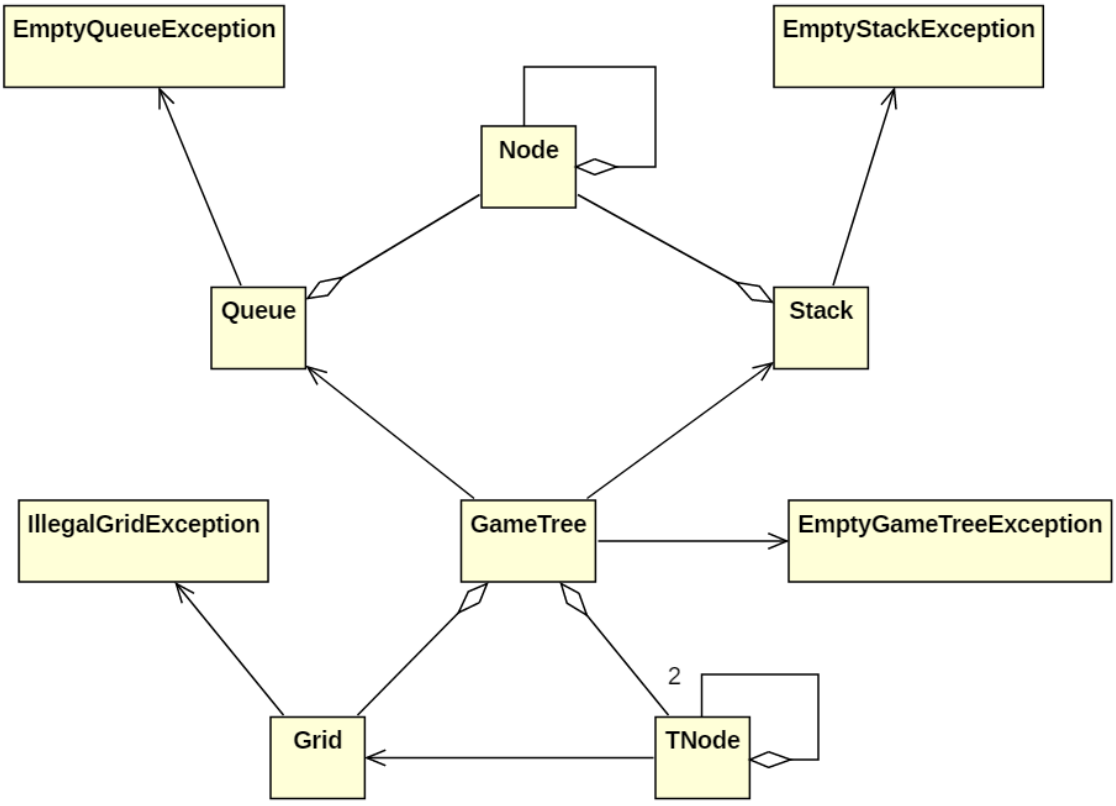
In addition, there are four exception classes (`EmptyGameTreeException`, `EmptyQueueException`, `EmptyStackException`, and `IllegalGridException`) and the harness class: `AssigThree125`.

The classes work together to complete different parts of the task, for example:

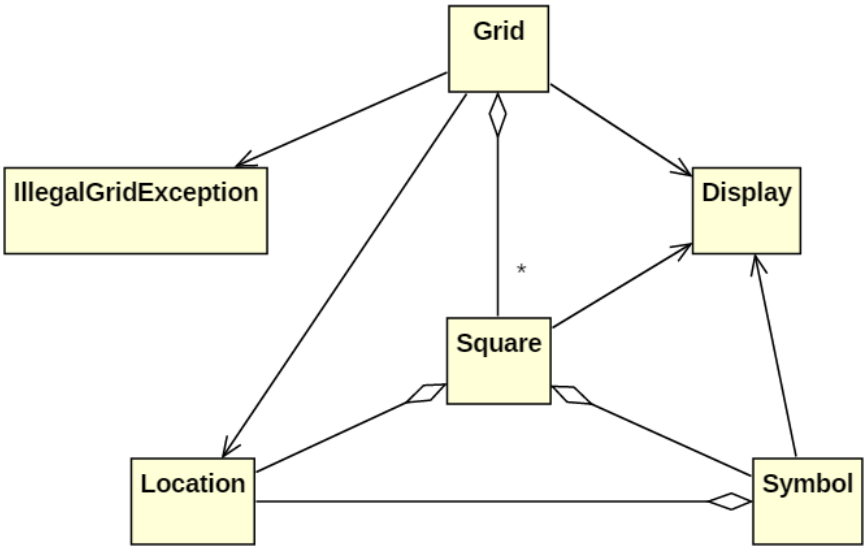
- the 'big picture' showing how the GUI drives the solution:



- the classes involved in building the game tree:



- and the classes that comprise the composition of the grid:



Your task is to complete the implementation of the abstract data types (ADTs) required to implement N-Queens.

This assignment will require you to utilise the implementation of three high-level data structures: a stack, a queue and a general tree. The tree will be used to store the encoded grid representation representing the current state of the game with successive potential moves being the children of each game tree node. As discussed in lectures, the stack will be used for the depth-first traversal and the queue for the breadth-first traversal.

All the interfaces are given to you and are complete. You will need to complete the implementation of the `Grid`, `Location`, `Square`, `Symbol`, `Stack`, `Queue`, and `GameTree` classes by completing the methods which have been declared but for which the method bodies are mostly/completely missing.

By default, the program will display lots of output to help with debugging — set `TRACING` to `false` in each class file to prevent this. As you develop your code, put in calls to `trace()` so that you can debug/verify your code.

Procedure

I suggest a staged approach:

1. complete `Location`, `Symbol`, and `Square`.
2. complete `Grid` noting that real-world row and column numbers commence at 1 while array elements commence at 0.
3. get it working so that you can see the starting grid.
4. implement `Stack` and the 'minor' `GameTree` methods and change `LIMIT` to `2` in `NQueens.java`.
5. get depth-first traversal working (see Slide 227 for hints on `buildGameDF()`!) by implementing `generateLevelDF()` and

`buildGameDF()`. Progressively increase `LIMIT` in `NQueens.java` until you are happy that depth-first generation is working properly.
6. implement `Queue` and change `LIMIT` back to `2` in `NQueens.java`.
7. get breadth-first traversal working (see Slide 229 for hints on `buildGameBF()`!) by implementing `generateLevelBF()` and `buildGameBF()`. Progressively increase `LIMIT` in `NQueens.java` until you are happy that breadth-first generation is working properly.
8. Set all `TRACING` variables to `false` and you're done!

Do not alter any supplied code. ***The only modifications required are the completion of methods where you find the words `COMPLETE ME!!!`.*** *All required classes are defined. All required interfaces are defined completely.*

You are given method declarations for all required functions. Each has a four-part header comment:

- Preliminary information comprising method name and purpose;
- **Pre-Condition** — this can be assumed to be true and lists what needs to be in place for the function to correctly execute;
- **Post-Condition** — this is what the method should have done when it completes (assuming the Pre-Condition was met); and
- Informally — a less detailed description of the method's role

You should attempt to think about this assignment early. Come and seek help when you require it, do not procrastinate.

Program Style

No method that you write should be longer than a screen-full of code. *You cannot change any distributed code.*

Your program should follow the following coding conventions:

- `final` variable identifiers should be used as much as possible, should be written in all upper case and should be declared before all other variables in a method;
- identifiers should be meaningful and variable and method names should be expressed in `camelCase`;
- local variables should only be defined at the beginning of methods and their identifiers should start with a lower case letter;
- every `if` and `if-else` statement should have a block of code (i.e. collections of lines surrounded by `{` and `}`) for both the `if` part and the `else` part (if used);
- every `do`, `while`, and `for` loop should have a block of code (i.e. { }s);
- the keyword `continue` should not be used;
- the keyword `break` should only be used as part of a `switch` statement;
- opening and closing braces of a block should be vertically aligned;
- all code within a block should be aligned and indented 1 tab stop (or 4 spaces) from the braces marking this block;
- commenting:
    - the header comment for each class should include at least

        - file name
        - student names
        - student identity numbers
        - a statement of the purpose of the program
        - date
        - the percentage of the work completed by the authors — 50:50 is expected and assumed but reasons should be given if it is more/less than this
    - each variable declaration should be on a single line and should be commented;
    - there should be a comment identifying groups of statements that do various parts of the task; and
    - comments should describe the strategy of the code (i.e. the algorithm) and should not simply translate the Java into English.

Style marks will be awarded proportionally, i.e. if you attempt only half the coding you can expect only half the style marks.

Marking Scheme

Task/Topic	Maximum mark
Program operates as specified	
`Location.java` correctly completed	3
`Symbol.java` correctly completed	3
`Square.java` correctly completed	3
`Grid.java` correctly completed	8
`GameTree.java` correctly completed	21
`Stack.java` correctly completed	5
`Queue.java` correctly completed	5
Program Style	
Does not unnecessarily repeat tests or have other redundant/confusing code	6
Uses correctly the Java naming conventions	6
Alignment of code and use of white space makes code readable	6
Always uses blocks in branch and loop constructs	6
Meaningful identifiers	6
Variables (other than loop counters) declared at the top of methods	6
Each variable declaration is commented	6
Comments within the code indicate the purpose of sections of code (but DO NOT just duplicate what the code says)	4
Header comments include student IDs and names	2

The program will be marked out of 96.

What and How to Submit

You should submit the entire folder of source files compressed as a ZIP file. *Do not submit a RAR archive — it will not be marked.* You should submit the files to the "Assignment 3" assignment drop-box on *KIT107*'s MyLO site.

You may make as many submissions as you wish; if you want to re-submit, simply do so.

Please note: only one submission is required for the pair. You cannot submit as an individual only as a registered member of a pair. ***If you can't see the Submission drop-box, it's because you're not registered in a pair.*** *Fix that problem first.*

Plagiarism and Cheating:

Practical assignments are used by the School of ICT for students to both reinforce and demonstrate their understanding of material which has been presented in class. They have a role both for assessment and for learning. It is a requirement that work you hand in for assessment is your own.

Working with others

One effective way to grasp principles and concepts is to discuss the issues with your peers and/or friends. You are encouraged to do this. We also encourage you to discuss aspects of practical assignments with others. However, once you have clarified the principles *of the question*, you must express your algorithm/code in writing or electronically entirely by yourself in your pair. In other words, you and your partner must develop the algorithm to solve the problem and write the program that implements this algorithm yourselves. *You can discuss the question with others, but not the solution.* Assistance with the solution can be provided by staff during tutorials or consulting times.

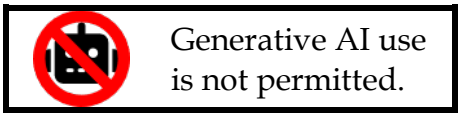
Cheating

- Cheating occurs if you claim work as your own when it is substantially the work of someone else.
- Cheating is an offence under the Ordinance of Student Academic Integrity within the University. Furthermore, the ICT profession has ethical standards in which cheating has no place.
- Cheating involves two or more parties.
    - If you allow written work, program print-outs, or electronic versions of your code to be viewed, borrowed, or copied by another student then you are *an equal partner* in the act of cheating.
    - You should be careful to ensure that your work is not left in a situation where it may be used/stolen by others.
- Where there is a reasonable cause to believe that a case of cheating has occurred, this will be brought to the attention of the Unit Coordinator. If they consider that there is evidence of cheating, then no marks will be given to any of the students involved and the case will be referred to an Academic Integrity Advisor for consideration of further action.

Generative AI

Under the University's Assessment and Results Procedure, all work that you submit as your own must be your own work. You can use generative artificial intelligence (AI) to learn, just like you would study with a classmate or ask a friend for advice — i.e. to learn concepts and grasp principles. You are *not permitted* to consult generative AI tools (such as ChatGPT) to learn how to

complete assessment tasks, or to provide partial/full solutions to assessment tasks. In particular, you are not permitted to present the output of generative AI as your own work for your assignments or other assessment tasks. This constitutes an academic integrity breach.



Julian Dermoudy, May 7th 2025.