Tic-tac-toe (aka, Tris) game

The goal of the exercise is to implement an engine to play Tic-tac-toe (https://en.wikipedia.org/wiki/Tic-tac-toe, tris in Italian, https://it.wikipedia.org/wiki/Tris_(gioco)). All the classes implementing the game are in "it.unive.dais.po1.exercise1" package.

You can find all the classes and tests of this exercise in the GitHub repository of the course (https://github.com/pietroferrara/PO1_2020).

In particular, class `Mark` represents a mark (that might be either a cross or a circle) in the tic-tac-toe board. The implementation of this class is given: it contains two static public methods, `getCircle` and `getCross`, that returns the instance of `Mark` class representing a circle and a cross, respectively.

You are asked to implement two classes:

- Class `Player` storing the mark of the player (represented as a `Mark` field `mark`) providing:
    - a constructor receiving as parameter the mark of the player (either cross or circle, represented as instance of `Mark`), and
    - a method `play` that, given a board, tries to play a move (that is, it marks one of the cells with his/her mark if the board is not full and there is not yet a winner), and returns true if he/she was able to play his/her round. Note that there is no assumption about how the player plays the game (e.g., randomly, taking the "first" empty cell, considering what is already on the board, etc..), but the player should always play his/her move if possible.
- Class `TicTacToeBoard` storing the status of the board (represented as a `Mark[][]` matrix in field `board`) and providing:
    - an empty constructor,
    - `put` method that given a `Mark`, and the coordinates x and y of a cell in the board, returns false if there is already a winner or the cell is not empty, or otherwise puts the mark in the given cell returning true,
    - `getMark` method that given the coordinates x and y of a cell returns the mark in the cell, or null if the cell is empty,
    - `isFull` method (returning true if and only if all the cells of the board are not empty), and
    - `winner` method that returns the mark of the winner, or null if there is no winner yet.

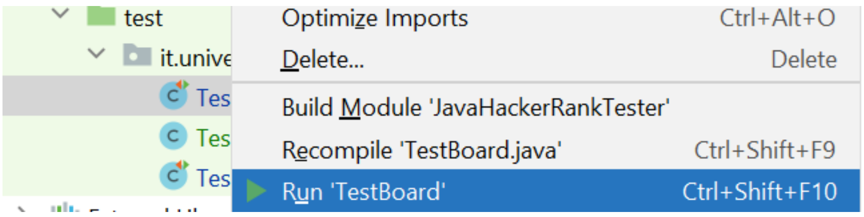
  The coordinates of cells in the board ranges from 0 to 2 (e.g., cell at (0, 0) is the upper left cell, while (2, 2) is the lower right cell).

The GitHub repository contains the skeleton of these two classes (where the aforementioned methods are not implemented), and you should implement them. Note that one of the goal of this exercise is to maximise the information hiding of your code.

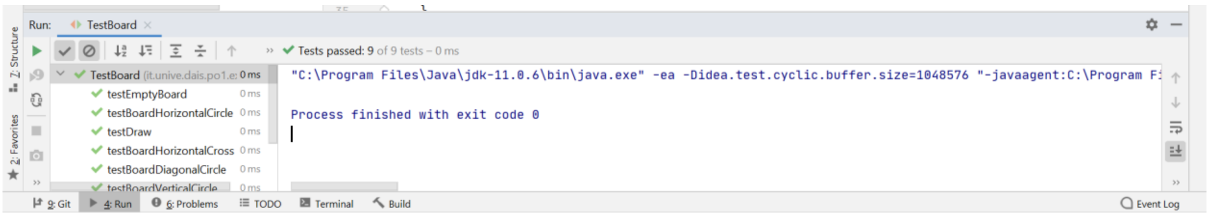
Note that class `Game` provides some utilities to debug the implementation: method `playGame` plays a game end-to-end and returns the mark of the winner (printing in the console all the rounds of the game), `print` prints in the console the status of a board (representing crosses by X, circles by

O, and empty cells by e), while `main` simply plays a full game. Therefore, class `Game` can be used to run the whole implementation and debug it.

You can self-evaluate your exercise with the tests that are in the `/test` directory. In particular, there are three distinct sets of tests: one to evaluate the functionalities of the board, one for the player, and one checking if the structure is correct (e.g., contains the fields and methods expected, there is no unneeded public method, etc..). The tests are implemented as JUnit tests. You can run them by right-clicking on the class file, and selecting "Run '<..>Test'":


test
it.unive
Tes
Tes
Tes
Optimize Imports Ctrl+Alt+O
Delete... Delete
Build Module 'JavaHackerRankTester'
Recompile 'TestBoard.java' Ctrl+Shift+F9
Run 'TestBoard' Ctrl+Shift+F10

The results of the tests are then visualized in the lower part in the Run view:



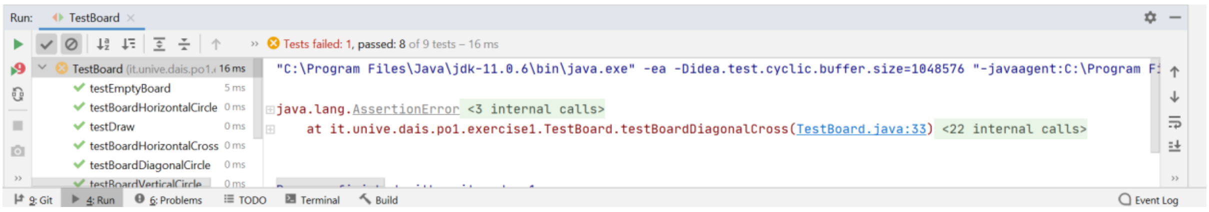
Tests passed: 9 of 9 tests – 0 ms

```
TestBoard (it.unive.dais.po1.e: 0 ms
  testEmptyBoard                 0 ms
  testBoardHorizontalCircle      0 ms
  testDraw                       0 ms
  testBoardHorizontalCross       0 ms
  testBoardDiagonalCircle        0 ms
  testBoardVerticalCircle        0 ms
```

```
"C:\Program Files\Java\jdk-11.0.6\bin\java.exe" -ea -Didea.test.cyclic.buffer.size=1048576 "-javaagent:C:\Program F

Process finished with exit code 0
```

If a test caused a runtime error it is marked as red, and if it did not pass one of the checks of the test (e.g., the board didn't return the expected winner) it is marked as orange. Selecting the failing tests, you can visualize the line of the test that failed:



Tests failed: 1, passed: 8 of 9 tests – 16 ms

```
TestBoard (it.unive.dais.po1.e 16 ms
  testEmptyBoard                 5 ms
  testBoardHorizontalCircle      0 ms
  testDraw                       0 ms
  testBoardHorizontalCross       0 ms
  testBoardDiagonalCircle        0 ms
  testBoardVerticalCircle        0 ms
```

```
"C:\Program Files\Java\jdk-11.0.6\bin\java.exe" -ea -Didea.test.cyclic.buffer.size=1048576 "-javaagent:C:\Program F

java.lang.AssertionError <3 internal calls>
    at it.unive.dais.po1.exercise1.TestBoard.testBoardDiagonalCross(TestBoard.java:33) <22 internal calls>
```