**Base exercise (9 points)**

WalkingBoardBasicTestSuite contains the tests of the basic exercise. Make it go all green, starting with the structural tests.

The WalkingBoard class represents a board with a figure standing on position x,y. Each position also has a value associated with it.

Initialise the board in two different ways: either make it size×size or use the given dimensions and initial values.

* About the second way:
  + The rows may not be uniform: they may have differing numbers of columns in them.
  + Don’t simply take the argument array: create its copy in the field.
* All values are set to BASE\_TILE\_SCORE, or to the given value if that’s bigger.

Operations.

* getPosition: returns the values of x and y in an array.
* getTile: returns the value of the of the board position.
  + If it is an invalid position, throw an IllegalArgumentException.
    - Let this condition be checked by the helper method isValidPosition.
* getTiles: returns the values of all the board positions.
  + Don’t let the field contents be accessed from the caller: create and return an exact copy of the structure.
* getXStep and getYStep: helper methods, they indicate by how much (-1, 0, or 1) the x and y coordinates respectively need to change to take one step in the given direction.
* moveAndSet: it moves the piece and changes the value of the new position.
  + Use the above helper methods in the code of the method.
  + If the move would make the figure leave the board, cancel the move: the figure remains where it was, the state of the board doesn’t change at all, and the method immediately returns 0.
  + Otherwise, the method returns the old value of the new position, and its value is set to the second argument.
  + setAndMove: it is similar to moveAndSet but it changes the value of the position before changing position

JUnit 5 tests. Where arguments are given, use a parameterized test.

* testSimpleInit(size): the first constructor works as expected.
  + getTiles shows suitable values.
  + The edges of the board (the smallest and biggest conceivable values) are accessible and contain the values that they have to (BASE\_TILE\_SCORE).
* testCustomInit(x, y, expected): the second constructor works as expected.
  + On positions where values smaller than three were passed, the board contains the value BASE\_TILE\_SCORE.
  + If you pass an array to the constructor and later modify a value in it, the respective tile retains the originally passed value.
  + If you modify an element in the return value of getTiles(), and get the value of the respective tile again, this newly received content has to be the originally set value.
* testMoves(): take four or five steps and check that the board’s contents are changed just right.
* Include a step that tries to move to the x coordinate Integer.MIN\_VALUE and another one that moves to the y coordinate 666.
  + Include a step that tries to move outside of the board. In this case, check that both the position and the board’s contents are unchanged.

**Extended exercise (6 points)**

Now we have players. As usual, start by turning all structural tests (see WalkingBoardExtendedTestSuite) go green.

The class has two constructors that call their equivalents in the parent class, and then call initPlayers that works like this.

* If the given number of players is less than two, raise an IllegalArgumentException.
* Otherwise, create the first player as a MadlyRotatingBuccaneer and the others as simple Players.

Implement move the following way.

* The players get to play in order. After the last player, the first one is up again.
* The player begins by calling turn() once.
  + When turning, the player’s direction changes to the next Direction (or to the first one if the current one is the last).
  + The MadlyRotatingBuccaneer is a bit different: it considers how many times he has come to play already, and turns that much.
    - So, when he plays for the first time, he doesn’t change his direction at all.
    - The next time he’s playing, he turns just once, like a regular Player.
    - Then he turns back a full 180°.
    - And so on…
* After turning, he takes as many steps forwards (in the direction that he’s facing) as the next element in the argument array tells him to.
  + As the second argument to moveAndSet, use the number of steps taken in total during this execution of move but no more than SCORE\_EACH\_STEP.
    - Here, steps that would leave the board count, too.
  + Increase the score of the current player by the value read from the board position.
* The return value contains the scores of the players in an array.

JUnit 5 tests: create walk1 and walk2 that walk on two separate boards that feature different numbers of players. In each one, let each players take a step at least three times. Check the resulting contents of each board and also check the final scores of the players.