Victoria University of Wellington
School of Engineering and Computer Science

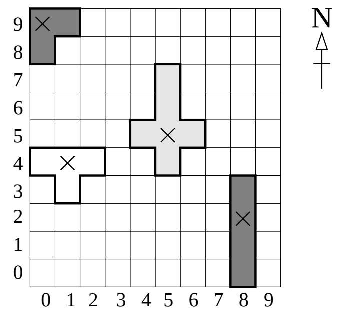
SWEN221: Software Development

Mid-term Test (worth 10% of overall mark)

Cathedral

In this test you will work on a program for checking rules of the *Cathedral* board game. You may read about this game here: `http://en.wikipedia.org/wiki/Cathedral_(board_game)`. In the game, two players (White and Black) play by *placing buildings* onto the board and *capturing areas* of the board. The program reads in a file which represents a *game of cathedral.* An example game file is given below on the left, along with a graphical visualisation of the final board on the right:

```
4
P(5,5;N;C) P(0,9;E;L)
P(1,4;S;T) P(8,2;E;S)
```



The first line of the file gives the number of moves, whilst the remaining lines give the moves of the game. Moves for the White player are in the left column, whilst those for Black are in the right column. Only two kinds of move supported:

- `P(x,y;N/S/E/W;C/L/S/T)` — A move where the player *places a building* on the board. The building is centered at position `x,y` (as indicated by X's in the diagram above). Then, the *direction* in which it is placed is either: `N` (North), `S` (South), `E` (East) or `W` (West). Finally, the *type of building* is either: `C` (for Cathedral), `L` (for Corner Building), `S` (for Straight Building), or `T` (for T-Shaped Building).
- `A(x,y;N/S/E/W;C/L/S/T;x1,y1-x2,y2;...)` — A move where the player places a building as before, and also *captures an area.* The area is given as a sequence of rectangles `x1,y1-x2,y2`, where `x1,y1` is the position of one corner and `x2,y2` the position of the opposing corner.

In our example above, White initially placed the Cathedral centered at `5,5` facing North. Black then placed an `L`-building centered at `0,9` facing East, White placed a `T`-building at `1,4` facing South and, finally, Black placed an `S`-building at `8,2` facing East. No areas were captured during this game.

Download. You can download the code provided for the Cathedral program here:

`http://ecs.victoria.ac.nz/~djp/files/test_7may2014_.jar`

You will find several Java source files, including a JUnit test file.

Submission. You should submit your solutions through the usual assignment submission system. Please make sure you submit to the correct session. The URL for submission is:

`http://ecs.victoria.ac.nz/cgi-bin/auth/submit?course=SWEN221`

Late submissions will get zero marks (unless you have arranged this with us, which will only be in exceptional circumstances).

PLEASE TURN OVER

1 Debugging the Parser (worth 10%)

Begin by importing the code provided into Eclipse and running the JUnit tests. At least the first eleven tests should fail, and helpful information regarding these errors is printed into the Eclipse console. There are two bugs in `Parser.java`, and your aim is to find and correct these mistakes. Upon completing this, you should find that tests `test_01`, . . ., `test_03` now pass. **NOTE:** you should also find that many or all of the other tests now fail!

2 Rotation of Cathedral (worth 10%)

You should find that some or all of the tests `test_04`, ..., `test_06` currently fail. This is because the method `BuildingTemplate.rotate()` is implemented incorrectly, and your aim is to fix this. Once finished, you should find that tests `test_04`, ..., `test_06` now pass.

3 Building Placement (worth 15%)

You should find that some or all of the tests `test_07`, ..., `test_11` currently fail. This is because the class `L_Building` is not implemented correctly, and your aim is to fix this. Once finished, you should find that tests `test_07`, ..., `test_11` now pass.

4 Invalid Building Placement (worth 15%)

You should find that some or all of the tests `test_12`, ..., `test_18` currently fail. This is because the program does not currently prevent buildings from being placed off the board. Furthermore, the program does not currently prevent buildings from being placed on each other. Your aim here is to implement the necessary checks to ensure that a `GameError` exception is thrown when an invalid placement move is made. Once finished, you should find that tests `test_12`, ..., `test_18` now pass.

5 Use of Invalid Building (worth 20%)

You should find that some or all of the tests `test_19`, ..., `test_23` currently fail. This is because the program does not keep track of how many buildings each player has left to play. Furthermore, it does not enforce the following rule:

> Whichever player is using the ”white” pieces begins by placing the Cathedral anywhere within the play area, aligned with the squares. Next the ”black” player places one of their game pieces on the board.

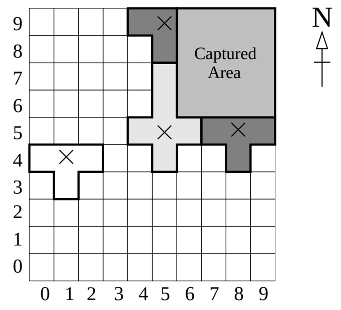
At the beginning of the game, each player is given exactly six buildings, made up from two of each kind (i.e. two `L_Buildings`, two `S_Buildings` and two `T_Buildings`. Your aim here is to implement the code necessary to keep track of the many buildings each player has played, and to throw a `GameError` if an attempt is made to play more than two of any kind. Note also that the `Cathedral` is not owned by either player. Once finished, you should find that tests `test_19`, ..., `test_23` now pass.

PLEASE TURN OVER

6 Capture Area (worth 20%)

You should find that some or all of the tests `test_24`, ..., `test_27` currently fail. This is because the program does not currently record which areas have been captured by either player.

```
4
P(5,5;N;C) P(5,9;S;L)
P(1,4;S;T) A(8,5;S;T;6,9-9,6)
```



Here, we can see that the area from `6,9` to `9,6` (inclusive) is captured by the Black player. Your aim is to implement the code for recording which areas have been captured. Note the following rules regarding captured areas:

> "If the captured territory contains one and only one of the opponent's pieces or the cathedral, that piece may be removed and the opponent may no longer place pieces in that area. An opponent's piece so removed can return to play at a later stage, but the cathedral will remain absent for the remainder of the game."

At this stage, you can simply remove any of the opponent's buildings located within the captured area. The empty methods `Game.capture()` and `Game.getHolderOfSquare()` have also been provided as a starting point. Once finished, you should find that tests `test_24`, ..., `test_27` now pass.

7 Invalid Captures (worth 10%)

You should find that some or all of the tests `test_28`, ..., `test_31` currently fail. This is because the program does not currently enforce the rules regarding captured areas. Note the following rules regarding captured areas:

> "Players may capture territory within the city by completely enclosing it with their pieces alone, or with the help of the city walls. Boundaries of these areas must be *wall to wall* that is, if two of the surrounding pieces only touch on their corners, that's not a capture. If the territory contains more than one piece (including the Cathedral), it is not captured and remains available for the opponent to use."

For example, the program currently does not check whether a captured area is appropriately enclosed by the player's pieces, the cathedral and/or the walls of the board. A function `Area.isInternal()` has been provided as a starting point. This can be used to distinguish the *external facing* squares of an area versus the *internal ones*. Once finished, you should find that tests `test_28`, ..., `test_31` now pass.