Victoria University of Wellington, School of Engineering and Computer Science

SWEN221: Software Development

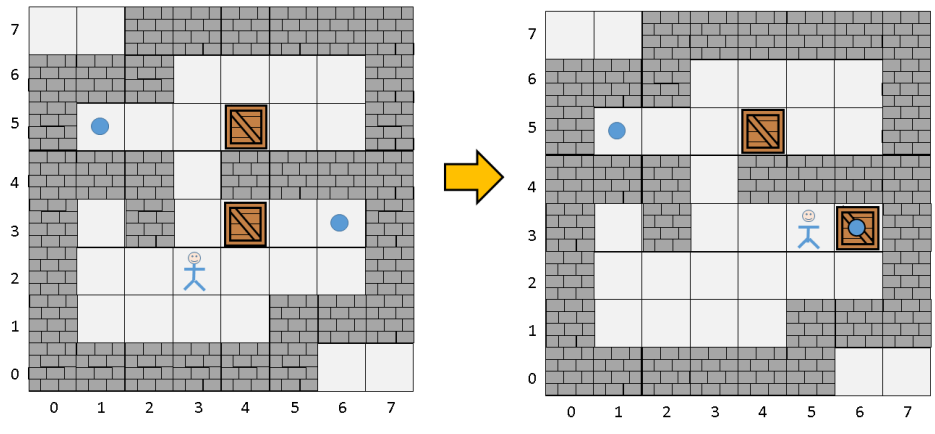
Mid-term Test (worth 10% of overall mark)

Sokoban

This test is about the *Sokoban* computer game. You may read about this game here: `https://en.wikipedia.org/wiki/Sokoban`. In the game, the player moves around the board pushing *crates* into *storage locations*. When every *crate* is placed in a *storage location*, then game is over. The player cannot move through *crates* or *walls* and cannot push more than one *crate* at a time. The aim of this test is to complete a program that checks a sequence of moves in the game of Sokoban is valid.

The program reads in a file which represents a *game of Sokoban*. An example game file is given below on the left, along with a graphical visualisation of the starting and final boards on the right:

```
4 8,8
W(N;1)
P(E;2)
```



The first line of the file gives the number of moves, followed by the *width* and *height* of the board. The remaining lines give the moves of the game. Only three kinds of move are supported:

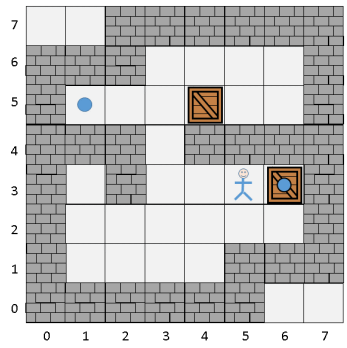
- `W(N/S/E/W;1-9)` — The player moves a number of steps from his/her current position in the given *direction*, which is: `N` (North), `S` (South), `E` (East) or `W` (West). Finally, the *number of steps* is a number between 1 and 9 (inclusive).
- `P(N/S/E/W;1-9)` — The player pushes a crate for a number of steps from his/her current position in the given *direction*. The *direction* and *number of steps* are the same as above.
- `F` — Signals that the game is *finished* and that every crate has been pushed into a storage location.

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

`http://ecs.victoria.ac.nz/~djp/files/test16_sokoban.jar`

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

ASCII Representation. In the JUnit tests, the board is represented using an array of ASCII characters. The following illustrates such a board:



```
7|_|_|#|#|#|#|#|#|
6|#|#|#|_|_|_|_|#|
5|#|+|_|_|X|_|_|#|
4|#|#|#|_|#|#|#|#|
3|#|_|#|_|_|O|*|#|
2|#|_|_|_|_|_|_|#|
1|#|_|_|_|_|#|#|#|
0|#|#|#|#|#|#|_|_|
  0 1 2 3 4 5 6 7
```

Here, "`O`" indicates the player, "`X`" indicates a crate, "`+`" indicates a storage location and "`#`" indicates a wall. Finally, "`*`" indicates a crate placed on a storage location.

1 Valid Walking Moves (worth 5%)

Begin by importing the code provided into Eclipse and running the JUnit tests. Several tests should be failing, and helpful information regarding these errors is printed into the Eclipse console. There is one bug in `Parser.java`, and your aim is to find and correct this mistake. Upon completing this, you should find that tests `test_01` and `test_02` now pass. **NOTE:** you should also find that many or all of the other tests now fail!

2 Valid Pushing Moves (worth 10%)

You should find that some or all of the tests `test_03`, ..., `test_06` currently fail. This is because the method `PushingMove.apply()` does not update the position of the crate, only the position of the player. Upon fixing this, you should find that tests `test_03`, ..., `test_06` now pass.

3 Invalid Walking Moves (worth 10%)

You should find that some or all of the tests `test_07`, ..., `test_11` currently fail. This is because the method `WalkingMove.apply()` does not check for the following invalid moves:

1. *Trying to move player into a position that is already occupied by a wall or crate*
2. *Trying to move player through a position that is already occupied by a wall or crate*

Having implemented these rules, you should find that tests `test_07`, ..., `test_11` now pass.

4 Invalid Pushing Moves (worth 20%)

You should find that some or all of the tests `test_12`, ..., `test_17` currently fail. This is because the method `PushingMove.apply()` does not check for the following invalid moves:

1. *Player trying to push something in unoccupied position*
2. *Player trying to push crate into wall or another crate*
3. *Player trying to push more than one crate at a time*

Having implemented these rules, you should find that tests `test_12`, ..., `test_17` now pass.

5 Storage Locations (worth 30%)

You should find that some or all of the tests `test_18`, ..., `test_25` currently fail. This is because the implementation for storing pieces is incomplete. In particular, it should support the following:

1. *Players and crates can occupy the same position as a storage piece*
2. *When a player or crate occupies a storage position, it should be flagged as being "stored"*
3. *When a player or crate moves out of a storage position, the storage piece should remain as before*

Having done this, you should find that tests `test_18`, ..., `test_25` now pass.

HINT: When a crate occupies a storage location, it is drawn as "`*`" on the board.

HINT: When the player occupies a storage location, it is drawn as "`@`" on the board.

HINT: To implement these rules, you should add a `boolean` flag `isStored` to both `CratePiece` and `PlayerPiece`. This should be `true` when the pieces are stored in a storage location, and `false` otherwise. We recommend that you do this using an `abstract` class which both `CratePiece` and `PlayerPiece` extend.

6 Finishing Moves (worth 15%)

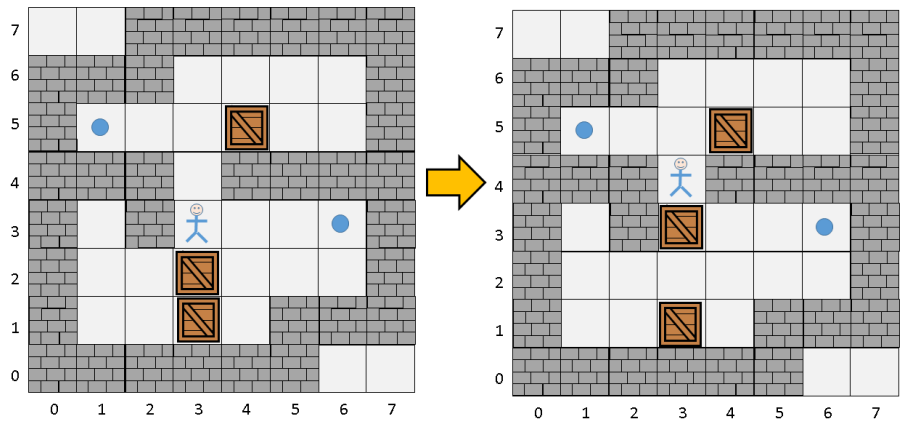
The implementation does not currently support the concept of the "finishing move". In particular, it should support the following:

1. *The finishing move should succeed if every crate is stored in a storage location*
2. *The finishing move should report a* `GameError` *if at least one crate is not in a storage location*

To implement these rules, you should first update `Parser` to create `FinishingMove` objects when required. You should also update `FinishingMove` appropriately, in particular to implement the rules above. Having completed this, you should find that tests `test_26`, ..., `test_32` now pass.

7 Pulling Moves (worth 10%)

The implementation does not currently support a "pulling move" which is similar to a "pushing move", but where the crate is initially located "behind" the player. An example of a pulling move is `L(N;1)` where the player pulls a crate one step north. The following illustrates this move on an example board:



Here we see that the crate is initially located to the south of the player, and is moved one step north as a result of being "pulled".

To implement the pulling move, you will need update the `Parser` to create objects of a new class `PullingMove` as appropriate. As with `PushingMove`, your new class `PullingMove` needs to implement the logic of pulling appropriately. Having completed this, you should find that tests `test_33`, ..., `test_40` now pass.

Submission. Your test solution should be submitted electronically via the *online submission system*:

`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). The minimum set of required files is:

```
sokoban/Game.java
sokoban/io/GameError.java
sokoban/io/Parser.java
sokoban/moves/AbstractMove.java
sokoban/moves/Move.java
sokoban/moves/PushingMove.java
sokoban/moves/WalkingMove.java
sokoban/pieces/CratePiece.java
sokoban/pieces/Piece.java
sokoban/pieces/PlayerPiece.java
sokoban/pieces/StoragePiece.java
sokoban/pieces/WallPiece.java
sokoban/util/Position.java
```