The University of Queenland, School of ITEE
CSSE2002/7023 — 2018
Assignment 1 (10%)
Due: August 31, 2018
Revision: 1.2 (as at August 20, 2018)

Introduction

The goal of this assignment is to implement a set of classes and interfaces — themed around a block-world — to be used in later assignments. You will implement precisely the public and protected items described in the supplied documentation (no extra members or classes). Private members will be for you to decide.
Language requirements: Java version 1.8, JUnit 4

Context

The block world in this assignment (inspired by a popular video game) includes:

- a set of tiles containing blocks,
- a variety of different block types, and
- a character (a builder) who can manipulate the blocks.

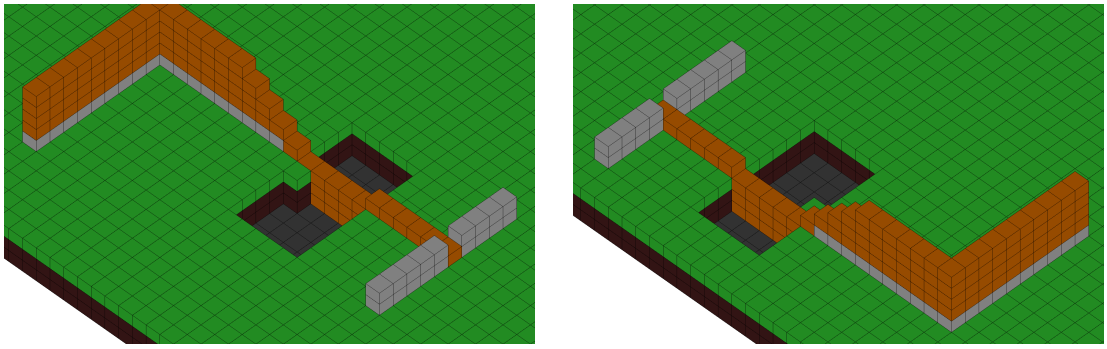

Figure: Two ways of viewing a simple block world with 4 simple block types — wood, stone, grass and soil.

The Builder can perform a set of tasks on the world including digging, moving blocks, carrying blocks, placing blocks, and moving around the world.

- Digging – the Builder can remove the top block from a tile if the block is diggable. If the block is also carryable, the builder will add it to their inventory, otherwise the builder will discard it.
- Moving blocks – the Builder can shift a block from one square to an adjacent square, but only downhill, and only if the block is moveable.
- Carrying blocks – the Builder has an inventory of blocks, which can be added to by digging, or utilised by placing.

- Placing blocks – the Builder can take a block from their inventory and add it to a tile.
- Moving around the world – the Builder can move to any tile that is connected by an "exit", but only if the heights are compatible (a maximum of 1 step upwards or downwards).

In this assignment, you will create classes to represent the builder, the tile, and different blocks types within the block world. You will also implement utility classes, such as Exceptions, for handling the interactions between the elements in the block world.

Ethical obligations

All work on this assignment is to be your own individual work. As detailed in Lecture 1, code supplied by course staff is acceptable but there are no other exceptions.

You are expected to be familiar with "What not to do" from Lecture 1 and `http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`.

If material is found to be "lacking academic merit", that material may be removed from your submission prior to marking. No attempt will be made to repair any code breakage caused by doing this.

If you have questions about what is acceptable, please ask.

Supplied material

- This task sheet
- A `.zip` file containing html documentation (javadoc) for the classes and interfaces you are to write (also on Blackboard). Unzip the bundle somewhere and start with `doc/index.html`.

Tasks

1. Implement each of the following classes and interfaces (described in the javadoc):
    - Builder
    - Tile
    - Block
    - StoneBlock
    - WoodBlock
    - GroundBlock
    - GrassBlock
    - SoilBlock
    - BlockWorldException
    - TooHighException
    - TooLowException
    - NoExitException
    - InvalidBlockException
2. Write JUnit4 tests for the methods in the following classes:
    - `Tile` as `TileTest`
    - `GrassBlock` as `GrassBlockTest`

Marking

The 100 marks available for the assignment will be divided as follows:

Symbol	*Marks*	*Marked*	*Description*
F	55	Electronically	Implementation and functionality: Does the submission conform to the documentation?
S	25	By "humans"	Style and clarity.
J	20	Electronically	Student supplied JUnit tests: Do the tests correctly distinguish between correct and incorrect implementations?

The overall assignment mark will be $A_1 = F + S + J$ with the following adjustments:

1. If $F < 5$, then $S = 0$ and $J = 0$ and "style" will not be marked.
2. If $S > F$, then $S = F$.
3. If $J > F$, then $J = F$.

For example: $F = 22, S = 25, J = 17 \Rightarrow A_1 = 22 + 22 + 17$.

The reasoning here is not to give marks to cleanly laid out classes which do not follow the specification.

Functionality marking

The number of functionality marks given will be

$$F = \frac{\text{Tests passed}}{\text{Total number of tests}} \cdot 55$$

Each of your classes will be tested independently of the rest of your submission. Other required classes for the tests will be copied from a working version. Functionality testing does not apply to your JUnit tests.

Note: Where "cannot be X" (e.g., "cannot be null") is used in the specification, it indicates that X is not a valid input to the function and the function will not be tested with input X.

Style marking

As a style guide, we are adopting[1] the Google Java Style Guide `https://google.github.io/styleguide/javaguide.html` with some modifications:

4.2 Indenting is to be **+4** chars not +2.

4.4 Column limit for us will be 80 columns.

4.5.2 First continuation is to be +8 chars.

There is quite a lot in the guide and not all of it applies to this course (eg no copyright notices). The marks are broadly divided as follows:

Naming	5
Commenting	6
Structure and layout	10
Good OO implementation practices	4

Note that this category does involve some aesthetic judgement (and the marker's aesthetic judgement is final).

[1] There is no guarantee that code from lectures complies.

Commenting

All functions and member variables need to be commented for this assignment. Comments for functions need to explain how the function is used, what the function returns, and what changes the function makes to its calling instance.

Additionally, any section of code (inside a function) that would be difficult for another reader to understand needs comments that explain what the code does.

As we have not yet covered Javadocs in the course, ***Javadocs are not required for this assignment***. Javadocs can be used if desired, and for functions and variables specified in the assignment, descriptions can be taken from the specification and added as comments.

Test marking

Marks will be awarded for test sets which distinguish between correct and incorrect implementations[2]. A test class which passes everything (or fails everything) will receive a mark of zero.

There will be some limitations on your tests:

1. If your tests take more than 20 seconds to run, they will be stopped and a mark of zero given.
2. Each of your test classes must be less than 300 (non-empty) lines. If not, that test will not be used.
3. ***Your tests must only call functions specified in the Javadocs for classes that you are testing***.

These limits are very generous, (eg your tests shouldn't take anywhere near 20 seconds to run).

Electronic Marking

The electronic aspects of the marking will be carried out in a virtual machine. The VM will not be running Windows and neither IntelliJ nor Eclipse will be involved. For this reason, it is important that you name your files correctly.

It is critical that your code compiles. If one of your classes does not compile, you will recieve zero for any electronically derived marks for that class.

It is critical that your class, interface and public member names match the Javadocs. If the names do not match, you could lose marks for all functionality associated with that class, interface or member. No corrections of typos will be made by staff. It is your responsibility to ensure that your submission is error-free and meets with the specification.

Submission

Submission is via the course blackboard area `Assessment/Ass1/Ass1 Submission`.

Your submission is to consist of a single `.zip` file with the following internal structure:

`src/`	`.java` files for classes described in the javadoc
`test/`	`.java` files for the test classes

A complete submission would look like:

[2] And get them the right way around

```
src/BlockWorldException.java
src/Block.java
src/Builder.java
src/GrassBlock.java
src/GroundBlock.java
src/InvalidBlockException.java
src/NoExitException.java
src/SoilBlock.java
src/StoneBlock.java
src/Tile.java
src/TooHighException.java
src/TooLowException.java
src/WoodBlock.java
test/TileTest.java
test/GrassBlockTest.java
```

Any submission which does not comply with this structure will receive a penalty of 10% of the maximum marks available. Your classes must not declare themselves to be members of any package. Do not submit any other files (eg no `.class` files). Remember that java filenames are case sensitive when your filesystem isn't.

Late submission

As stated in the ECP:

- No late submissions will be accepted.
- Requests for extensions must be made at least 48 hours prior to the submission deadline. Extension requests received after this point may not be able to be considered.
- Due to the incremental nature of the assessment items in this course, extensions to Assignment 1 are extremely unlikely. If you apply for an extension, you may be asked to meet with the course coordinator to discuss alternatives. Failure to attend this meeting before the assignment due date will result in your extension request being denied, unless the medical or other circumstances are such that you could not reasonably be expected to attend the meeting.

Revisions

If it becomes necessary to correct or clarify the task sheet or javadoc, a new version will be issued and a course announcement will be made on blackboard. No changes will be made on or after Monday of Week 6 (ie August 27, 2018). ***This means that you need to periodically check blackboard until this date***.

Version 1.1 — August 10, 2018

- Correct some typos in the task sheet
- Added missing files to submission:
  - `src/NoExitException.java`
  - `src/InvalidBlockException.java`

- Renamed `src/BlockException.java` to `src/BlockWorldException.java`
- Added section about commenting
- Added a sentence about "cannot be X"
- Changes to the specification:
  - Changed spec for `Tile` constructor, `Tile.placeBlock`, and `Builder.dropFromInventory` to clarify that the number of blocks for the conditions are 8 or more and 3 or more.
  - Changed spec for Block.dig to remove null as a possible return value.

Version 1.2 — August 20, 2018

- Changed the following in the Javadoc specifications:
  - Changed `Tile` constructor to clarify the number of blocks needs to be "more than 8" to throw a TooHighException.
  - Changed `Tile` constructor to clarify the ***index*** of a GroundBlock needs to be "$\geq$ 3" to throw a TooHighException.
  - Changed `Tile.dropFromInventory` to clarify that the block needs to be removed from the Builder's inventory.
  - Changed `Tile.getTopBlock` so that it no longer says it can return null.
  - Fixed typo in `Tile.placeBlock` changed "target block" to "target tile".
  - Fixed minor typo in `Tile.moveBlock` changed "getExits(exitName)" to "exitName in getExits".