The University of Queenland, School of ITEE
CSSE2002/7023 — 2018
Assignment 2 (15%)
Due: 4 May 18 17:00
Revision: 1.1.

Abstract

In this assignment you will add classes and methods to a base set developed in Assignment 1.
As before, you are to implement exactly the public and protected aspects described in the javadoc.
Also as before, you may add private members as you see fit.
Note you may add additional helper classes if you wish. However, they must be either nested/annonymous or package private classes and *only used within the file in which they are declared.*
For some items in the javadoc you will need to write them from scratch, for others, you will be given code which you may need to add extra features to (See below). In those cases, you are not to modify any supplied methods.
Language requirements: Java version 1.8, JUnit 4

Preamble

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[1].

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 you are to write. Unzip the bundle somewhere and start with `doc/index.html`.
- A `.zip` file containing supplied source code for you to build on. Do not modify the supplied files except to add required methods. Note that the supplied code will require you to implement `DigException` before it will compile.

Tasks

1. Implement the following classes and all described methods:
   (a) Pair
   (b) MapWalker
   (c) BoundsMapper
   (d) MapIO
2. Implement the following methods:
   (a) Room.makeExitPair

[1] No attempt will be made to repair any code breakage caused by doing this.

3. Declare `repr` in Thing.
4. Implement `repr()` in the following classes: `Critter, Builder, Explorer, Treasure`.
5. Implement `decode()` in the following classes: `Builder, Explorer, Critter, Treasure`.
6. Write JUnit4 tests for the methods in the following classes:
   - `MapIO` as `MapIOTest`
   - `BoundsMapper` as `BoundsMapperTest`

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 $A_2 = F + S + J$ 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: $F = 22, S = 25, J = 17 \Rightarrow A_2 = 22 + 22 + 17$.

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.

Style marking

As a style guide, we are adopting[2] 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.

[2]There is no guarantee that code from lectures, pracs and tutes complies.

4.5.2 First continuation is to be `+8` chars.

- All private/"package private" members must be commented (you may use javadoc comments but are not required to).
- Java source files must be encoded as either ASCII or UTF-8.
- All public and protected comments are expected to use javadoc markup (eg @param etc).

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	8
Good OO implementation practices	6

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

Test marking

Marks will be awarded for test sets which distinguish between correct and incorrect implementations[3]. A test class which passes everything (or fails everything) will likely get a low mark.

When testing `loadMap` and `saveMap` you should test single room cases separately but for multiple room cases, test them together.

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.

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 or linux box. 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 also 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.

Submission

Submission is via the course blackboard area `Assessment/Ass2/Ass2 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

Note: you must submit supplied classes as well.
Your classes must not declare themselves to be members of any package. Do not submit any other files (eg no `.class` files). Any files or directories which do not match the above structure will be deleted before marking. Remember that java filenames are case sensitive even when your filesystem isn't.

[3] And get them the right way around

Late submission

See the ECP for rules and penalties regarding late submission. Note that, for example, that submitting at 2am on 5th of May would incur a 30% penalty.

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 2018-04-30.

Changes from 1.0

1. Changes to Javadoc:
   (a) Changed text on the MapWalker constructor: the constructor is not supposed to call visit. The constructor should not do any actual walking.
   (b) Added note to walk() that reset should be called at the beginning of the routine.
   (c) Removed classes from javadoc that were removed from the assignment (eg DigException).
2. Changes to supplied files (these are all to make sure the supplied code complies with the style guide):
   (a) Added static final to (Critter): `private static final int MAX_HEALTH = 10;`
   (b) Wrapped Explorer constructor line (3rd constructor)
   (c) Wrapped long line in Explorer class javadoc.
   (d) Reordered imports in Player.
3. Changes to task sheet:
   (a) Abstract changed to make it clear where helper classes can be used.

Changes from draft version

1. A typo in the example Explorer.repr output has been corrected.
2. I've dropped Builder.digExit() from the assignment. The javadoc was confusing and there is enough to do with the MapWalker and BoundsMapper. The supplied files have been updated.
3. I've dropped the DigException class and its subclasses RoomExistsException and OffTheMapException since they were only used by Builder.digExit()
4. Added missing getDamage() to Explorer in supplied files.
5. You will need to make sure that any subclass of MapWalker which overrides reset() calls super.reset(), to ensure that state is cleared properly.
6. I will not test the MapWalker constructor with a `null` argument.
7. Order of items in a Pair matters.
8. Critter has been modified to have a `MAX_HEALTH` to make saving and loading consistant.