UNIVERSITY OF EXETER

COLLEGE OF ENGINEERING, MATHEMATICS
AND PHYSICAL SCIENCE

ECM2414

Software Development CA

Continuous Assessment

Handed out date: 21 October 2019
Handed in date: 21 November 2019

This CA comprises 40% of the overall module assessment.

This is a paired assessment, and you are reminded of the University's Regulations on Collaboration and Plagiarism (https://intranet.exeter.ac.uk/emps/?nav=288).

In this assignment you will implement an application using the various techniques that you are learning about during this module. You will tackle this task using the pair programming development paradigm that will reinforce the idea of programming as a team exercise.

1. Development Paradigm

Pair programming is a popular development approach, primarily associated with agile development, but used across the software industry. In pair programming, two programmers work together to generate the solution to a given problem. One (the driver) physically writes the code while the other (the navigator) reviews each line of code as it is generated. During the development, the roles are switched between the two programmers regularly. The aim of the split role is for the two programmers to concern themselves with different aspects of the software being developed, with the navigator considering the strategic direction of the work (how it fits with the whole, and the deliverables), and the driver principally focussed on tactical aspects of the current task at hand (block, method, class, etc.), as well as allowing useful discussion between the developers regarding different possible solutions and design approaches.

Research has indicated that pair programming leads to fewer bugs and more concise (i.e., shorter) programs. Additionally, it also facilitates knowledge sharing and flow between developers, which can be crucial for a software house, with pair programming often cycling through developers on a team so everyone is eventually paired with everyone else at some point. This assignment will introduce you to the paired programming approach in a practical fashion, through the development of a threaded Java game. As such you will need to form pairs. At this stage, you should have already had your partner, either found by yourself or paired by me. The pairing information is available on module ELE page.

2. Task Specification

In this assignment you are to model a game involving multiple competing players, in a thread-safe fashion. The game being modelled has a strictly positive number of players. There are six bags of pebbles in the game, three white bags (A, B and C) and three black bags (X, Y and Z). At the start of the game the white bags are empty and the black bags are full. Each player takes 10 pebbles from a black bag (the black bag each player selects is chosen at random). Each pebble has an integer weight value.

If the weight of pebbles held by a player is 100, then they have won. They should immediately announce this fact to other players, and all other simulated players should stop playing (once they have ensured they only have 10 pebbles in their possession). If a player does not hold a winning hand they should discard a pebble to a white bag. They should then select one of the three black bags at random and draw a new pebble from this bag. This process should continue until either the player has won (has 10 pebbles with a total combined weight of 100), or until another player has won and the game has ended.

On starting, the command-line program should request the number of players, and after this has been entered it should then request the location of three files in turn containing the weight of each of the pebbles used in the black bags at the start of the game. An example of this is

```
Yuleis-MacBook-Pro:CA1_example yw433$ java PebbleGame
Welcome to the PebbleGame!!
You will be asked to enter the number of players.
and then for the location of three files in turn containing comma seperated inte
ger values for the pebble weights.
The integer values must be strictly positive.
The game will then be simulated, and output written to files in this directory.

Please enter the number of players:
3
Please enter location of bag number 0 to load:
range_1.csv
Please enter location of bag number 1 to load:
range_1.csv
Please enter location of bag number 2 to load:
range_1.csv
```

The legal format of these files is as a comma-separated list of integers. Examples of these files can be found on module ELE page.

The program should perform exception-handling to cope with illegal player number entry and illegal file formatting/values and request the user to provide legal values until both inputs are legal, or until the user enters 'E' (in this case the program should exit).

The simulated game should have the following properties:

- Once a black bag is empty, all the pebbles from a white bag should be emptied into it. Bag X should be filled by bag A, bag Y filled by bag B, and bag Z filled by bag C.
- The process of drawing pebbles from a black bag should be *uniformly at random* – that is it should be equally probable to draw *any* of the pebbles in the bag.
- The players should act as *concurrent threads*, with the threading commencing *before* drawing their initial pebbles.
- Players should be implemented as nested classes within a `PebbleGame` application.

- Drawing and discarding should be an atomic action – additionally, the bag a pebble is discarded to, must be the paired white bag of the black bag that the last pebble draw was from. Specifically, if the last pebble was drawn from X, the next discard should be to A, if the last pebble drawn was from Y, the next discard should be to B, and if the last pebble drawn was from Z, the next discard should be to C.
- On loading the bag files, the program should ensure that each black bag contains at least 11 times as many pebbles as players. For example, if there are three players then there must be at least 33 pebbles.
- If a player attempts to draw from an *empty* black bag, the player should attempt to select another bag until they select a bag with pebbles.
- A player should never find an empty black bag if its paired white bag has pebbles within it.
- Pebbles must have a strictly positive weight – therefore the program should detect and warn the user if they are trying to use files where this is not the case.

After each draw and discard a player should write to a file the drawn/discarded value and the associated bag, and then the current values of the pebbles the player holds. I.e., if player 1 draws a pebble of weight 17 the following two lines should be written to `player1_output.txt`:

```
player1 has drawn a 17 from bag Y
player1 hand is 1, 2, 45, 6, 7, 8, 56, 23, 12, 17
```

and if subsequently player 1 discards a pebble of weight 45 the following two lines should be written to `player1_output.txt`:

```
player1 has discarded a 45 to bag B
player1 hand is 1, 2, 6, 7, 8, 56, 23, 12, 17
```

There are no restrictions on the standard Java libraries that you may use.

3. Submission

Your submission consists of two parts: an electronic submission (E-submit) and a report submission (BART-submit).

E-submit: Your electronic submission includes the following two parts (i.e., `pebbles.jar` and `pebbelsTest.zip`). You need put these two files in one zip file, and submit this zip file, at empslocal.ex.ac.uk/submit, to the folder 2019-11-21~ECM2414~Yulei Wu~JavaGame, by 12 noon on the 21st of November 2019.

- A copy of your finished classes in an executable *jar* file named `pebbles.jar`. The jar file should include **both** the bytecode (*.class*) and source files (*.java*) of your submission.
- A copy of your finished classes, and associated test classes and test suites, and any supporting files, in a *zip* file named `pebbelsTest.zip`. The zip file should include **both** the bytecode (*.class*) and source files (*.java*) of your submission (plus any testing files the tests may rely upon), and a README file, detailing how to run the test suite (and what the expected output should be).

BART-submit: You should also **hand in a paper report together with your E-submit receipt**, using BART, to the Education Office (Harrison Hub Receiption), by 12 noon on the 21st of November 2019. **You will need to attach the BART sheets of both members to the physical submission.** The report, with minimum 2 cm margins and 11 point text, should contain the items listed below.

- A cover page which details how you would like the final mark to be allocated to the developers, based upon your agreed input (i.e., 50:50 if both parties took equal roles,

or perhaps 55:45 if you both agree that one party may have contributed a bit more than the other – do remember however that in pair programming both the *driver* and *observer* roles are vital, and should be switched frequently between developers). The maximum divergence allowed is 60:40, although non-contributors will receive a zero.

- A (max 1 page) development log, which includes date, time and duration of pair programming sessions, and which role(s) developers took in these sessions, with each log entry signed (using your candidate number) by both members. This part of the document should be no more than one side of A4.
- A (max 3 page) document detailing your design choice and reasons with respect to both your production code, and any known performance issues (also including addition features (see marking criteria in Section 4)). This part of the document should be no more than two sides of A4 (If you can't detail all the design choice within the page limit, you need to include only the key design choice).
- A (max 3 page) document detailing the design choice and reasons with respect to your tests, performance assessment, and code coverage of the tests[1] (also including the testing of addition features). You may use either of the JUnit 3.x or 4.x frameworks, but you should explicitly detail which framework you are using in your document. This part of the document should be no more than three sides of A4 (Again, if you can't detail all the testing design choice within the page limit, you need to include only the key testing design choice).

Your CA will be marked based on the criteria in Section 4.

If you have any question about this CA, please ask Yulei Wu (Y.L.Wu@exeter.ac.uk).

[1] TikiOne JaCoCoverage Plugin can be installed in the NetBeans IDE in the machines of Harrison 207 and 208. The latest version of NetBeans IDE doesn't support this plugin.

4. Marking Criteria

This assessment will be marked using the following criteria.

Marking Scheme	**Description**	**Mark**
The Report		
Structure and contents of the report	The report is well structured and presented. The design is well explained, matches the specification provided and the implemented code.	25%
The Electronic Submission		
Code comments	Code comments are useful and informative, and at the appropriate level (i.e., it should not contain spurious comments, or ones that do not serve an explanatory purpose).	10%
Production Code. Implementation & Testing.	The code is well structured and presented, with a coherent design and clear management of object states. The program is thread-safe, produces output and takes input in the formats specified. The Junit tests are well formulated and cover the code well.	35%
README file.	The file details all that is required to run the test suite, and details the expected results.	5%
Production Code. Handling exception states.	The code deals smoothly with exceptional inputs, and is robust in use.	10%
Additional Useful Features	Successful implementation of additional useful features at your discretion	15%
Penalties		
Penalty	Non-submission of cover page with weightings (a 50:50 will be assumed)	–5%
Penalty	Not attaching the BART sheets of both members to the physical submission	–5%
Penalty	Non-submission of development log	–10%
Penalty	The submitted jar file `pebbles.jar` cannot be executed from command line as specified.	–20%
Penalty	The test suite cannot be run following the instructions in README file.	–20%
Penalty	Submission is greater than the stated number of pages (8)	No penalty applied, but only the first 8 pages will be marked.