CS-110 2018/19 - Coursework 1: Last Biscuit

Neal Harman, Oct 2018

The first part of the marking process is automated - your code will be automatically compiled and run against a test script. *Therefore it's essential that you follow the instructions **exactly** - or your application will be marked 'does not compile' and/or 'does not work'.* The second part of the marking process involves me reading the code - and this will **not** be done automatically (but will be marked against a scheme that you will be able to read). This process makes the marking more consistent and systematic - there is no 'judgement' or 'opinion' about whether it works or no. *Also this is how your code will be tested in industry - it will be automatically run against at set of test by dedicated testing code, and it will have to be exactly right.*

Unfortunately this does mean the instructions are very long - but please take time to understand them to ensure you submit the correct work.

First I'll describe what you have to do - with examples. Next I'll explain how to get and use the sample test files you can use to make sure your program will pass my automated test. Then I'll give you some guidance notes and hints. Finally I'll tell you exactly how to submit your work. **You will have access to the automatic test script so you can check your program works - the biggest problem with automatic testing is not doing this before submitting work.**

- **If you DO run the automatic tests and they say your code PASSES, you are guaranteed to get credit for it working;**
- **if you DO run the automatic tests and they say your code FAILS, you are guaranteed NOT to get credit for it working;**
- **If you DON'T run the automatic tests, you will have NO IDEA if it works or not (experience shows usually it doesn't).**
- **Please see below to find out how to tell if your code passes or fails.**

The Game of Last Biscuit

The Game of Last Biscuit is a very simple variation of the very old Game of Nim. Like lots of games it comes in different versions. *It's essential that you follow the instructions here - not any others you might find or already know that are different* because you are almost certain then to fail the automatic tests.

- The game is for two players
- Players take it in turns to play
- The players have two barrels of biscuits (or piles of cookies; or bags of sweets; or whatever - but they have two 'groups' of 'something' - but in your code, they must be called biscuits)
- Each player takes it in turn to take a number of biscuits (cookies, sweets)
- They can take as many as they want BUT
- They must EITHER take that number from ONE barrel (pile, bag)
- OR they must take the SAME number from BOTH barrels (piles, bags)
- *The player who is takes the last biscuit wins*.

The trick, obviously, to winning is not to leave your opponent with the same number of biscuits/cookies/sweets in both piles/barrels/bags; and with at least one biscuit/cookie/sweet in each barrel.

Example - two piles of 6 and 8 biscuits

- Player 1 (who is not very clever) takes two biscuits from pile 2
- This leaves both piles with 6 biscuits
- Which means that Player 2 can take ALL the biscuits from both piles - because there are the same number in each.
- So Player 2 wins

Example - two bags of 6 and 8 biscuits

All our examples will have 6 and 8 because that's what the coursework will be using

- Player 1 takes 1 from bag 1 (leaving 5 and 8)
- Player 2 takes 4 from bag 1 (leaving 1 and 8)
- Player 1 takes 6 from bag 2 (leaving 1 and 2)
- Player 2 thinks carefully…
- Player 2 realises that there are FOUR possible outcomes:
    - They could take 1 from bag 2 - leaving 1 in each - so Player 1 will take both and win
    - They could take 2 from bag 2 - leaving 1 in bag1 - so Player 1 will take it and win
    - They could take the last one from bag 1 - and Player 1 can take both from bag and win
    - They could take one from each bag - leaving 1 in bag 2 so Player 1 can take it and win
- Player 2 cannot win.. and realises they messed up when they took 4 from bag 1

There's a Flash version of this game here:
https://nrich.maths.org/2656/index
It plays against the computer (we won't do that). Also, because it's Flash, it's a bit irritating to play. But you can get a better idea about the game

Your task is to write an implementation for two human players who share the same computer and keyboard and take turns to play - there is no 'AI' in this.
Because of the varying level of programming expertise, there are three levels of implementation.

Level 1 - Minimum Acceptably Correct - Assumes all Input is Correct

If you attempt this version you do not need to do any checking of input data - you can **assume** that the users will always enter correct data. That is,

- You can assume users only ever type in integers where integers are expected
- The user is expected to type **one**, **two** or **both** to say if they want to take biscuits from one or both barrels - but you don't need to check if they actually do
- You can assume the integers they type will always be >0 and < number left in the barrels

It is acceptable that this version crashes (or otherwise doesn't work properly) if the users does not enter integers, or if they enter integers outside the allowed ranges. **80% of the marks are available for this version.**

Level 2 - Checks Input to Ensure Integer; Does not Check Ranges

The second level adds checks to ensure that the user has entered only integers when integers are expected, and does not crash if this happens. Instead, it asks the user to re-enter data. However, as above, you can **assume** that the users will always otherwise enter correct data. That is,

- The user is expected to type **one**, **two** or **both** to say if they want to take biscuits from one or both barrels - but you don't need to check if they actually do
- You can assume the integers they type will always be >0 and < number left in the barrels

It is acceptable in this version that the program crashes (or otherwise doesn't work properly) if the user enters an integer *outside the acceptable range* of if they type something other than **one**, **two** or **both** when selecting a barrel. BUT if the user types a non-integer when an integer is expected, the program should print some message and ask them to re-enter the data. **90% of the marks are available for this version**.

Level 3 - No Assumptions about Input; Checks Ranges

The final version both checks that input items are integers where integers are expected, ***and*** *that they are all legal values;* ***and*** *that other inputs are legal.*

- If an integer input is expected, only an integer will be accepted - otherwise the user will be asked to re-enter an integer.
- Integers must be > 0 and < no. of biscuits that can be taken at that point in the game - otherwise the user will be asked to re-enter the data.
- When selecting one or both barrels, the user must enter **one**, **two** or **both** - otherwise, the user will be asked to reenter the data.

In all cases, if the user enters incorrect data, the program will ask for the data to be re-entered. This version should not crash or behave incorrectly in any circumstances. **100% of the marks are available for this version.**

Critical Messages - ESSENTIAL FOR CORRECT OPERATION.

Because the marking is partly automated, some of the messages your program prints must conform **exactly** to a specific pattern. The messages and the way you print them are chosen so they both make sense when you play the game manually and when you run the test script to play automatically. **Note the case! this is significant and getting it wrong means your code won't work.**

These messages are:

- "**Biscuits taken by player 1:** " - this is the message you should print to ask player 1 to choose a bag. Note the case and the space after the ":" You should use `System.out.print` and not `System.out.println`
- "**Biscuits taken by player 2:** " - this is the message you should print to ask player 2 to choose a bag. Again, note the case and the space after the ":" And again, you should use `System.out.print`
- **"`From barrel1 (one), barrel2 (two), or both (both)? `"-** this is the message you should print when you are asking either player which barrel they should take biscuits from - again note the case and the space after the ?
- "**`Biscuits Left - Barrel 1: X`**
  `Biscuits Left - Barrel 2: Y`" - this is the message you should print (over two lines) after each player has taken their turn, where X is the number of biscuits remaining in barrel 1 and Y in barrel 2. In this case you should use `System.out.println` for each line **DO NOT** attempt to format the numbers being generated with the `System.out.printf or System.out.format` operations.
- "**`Winner is player 1`**" and "**`Winner is player 2`**" are the messages you should print if player 1 and 2 win respectively. Again in this case you should use `System.out.println`

Note that you should not print the " (quote) characters, but you should pay attention to the spaces that appear at the end of messages which ask the user for input.

Example Dialogue
Here's an example of the message sequences when playing a very simple game. Messages in bold are printed by the computer; italic are entered by the player.

```
Biscuits Left - Barrel 1: 6
Biscuits Left - Barrel 2: 8
Biscuits taken by player 1: 1
From barrel1 (one), barrel2 (two), or both (both)? two
Biscuits Left - Barrel 1: 6
Biscuits Left - Barrel 2: 7
Biscuits taken by player 2: 2
From barrel1 (one), barrel2 (two), or both (both)? one
Biscuits Left - Barrel 1: 4
Biscuits Left - Barrel 2: 7
Biscuits taken by player 1: 3
From barrel1 (one), barrel2 (two), or both (both)? two
Biscuits Left - Barrel 1: 4
Biscuits Left - Barrel 2: 4
Biscuits taken by player 2: 4
From barrel1 (one), barrel2 (two), or both (both)? both
Biscuits Left - Barrel 1: 0
Biscuits Left - Barrel 2: 0
Winner is player 2
```

Messages for Level 2 and Level 3 Solutions
If you attempt the more advanced Level 2 or Level 3 solutions (see above) then you obviously have to print out error messages when the user does not enter the correct data. However, because these more advanced functions are not automatically checked, you can choose whatever messages you like (though make sure they are clear and sensible).

Naming Your Program, How to Submit and Identifying Yourself
To ensure that the automatic scripting works as intended, *you need to call your Java source code file **LastBiscuit.java** (note case!)*. You should only submit **one** file. In most cases, your solution should only have one class file - if you *really* want to submit more than one class it's essential that you see me first. *Your code should also include a statement in a comment saying which Level (1, 2 or 3) you have attempted.*

Starting State
As stated above you should start your program with 6 biscuits in barrel 1 and 8 biscuits in barrel 2.

Testing Your Program - Manual
You should first test your program manually - entering values for the player choices - to see if the output meets your expectations. *If you have used an IDE like Netbeans or Eclipse, please make sure your code runs from the command line* - that is, you can compile and run your code by typing:

```
javac LastBiscuit.java
java LastBiscuit
```

Testing Your Program - Automatic
Because of the automatic testing, it's important that you have a way to try this yourself to make sure your code will pass. I will make available to you all the test files I will use to make sure your

program works (or not). The files you need for this are part of the zip file you can download from Blackboard.
I recommend that you put all the files in the same folder you are using to develop your program.
The testing system consists of two script files that compile and run your program, read in inputs from a test input file, and then check it matches a test output file. The test input files are called cwtest1.txt, cwtest2.txt and so on. The output files are called `cwtest1output, cwtest2output` and so on (there are five). You can open them to see the input and expected output.

Testing Using Windows

You need to use the file `biscuittest_win.bat` - simply **type at the command line**:

```
biscuittest_win cwtest1
```

to run `cwtest1` (and obviously you can type `cwtest2` and so on instead). The output you see should be the commands being run - the final line should be

```
FC: no differences encountered
```

if your code is correctly generating the required output. If not, then the final line (or lines) should tell you what is different so you can track down the problem in your code. *If you do not see the line above at the end of the output your code is not correct.*

Testing Using A Mac or Linux

In this case you need to use the file `biscuittest_nix.sh.` instead of `biscuittest_win.bat`, and you need to **type at the terminal**

```
sh biscuittest_nix.sh cwtest1
```

Again, you can put `cwtest2` etc. to run the other tests
In this case there is ***no*** *output if the program works correctly.* If there is *any* output, it tells you how your program output differs from what it expects - *meaning your program is not correct*.

Seeing the Actual Test Output

In both cases, the output from your program is sent to a file called ‘`output.txt`’ - you can look in this and compare it with the output expected in the `biscuittest` file.

Do Not Ignore the Automatic Testing

If your code does not pass the automatic testing, you are guaranteed to loose 40% of the marks! If you do not do the automatic testing you might get lucky, and your code might pass - but that’s actually quite unlikely in practice *and so you will, again, loose 40% of the marks!*
If your code does not pass the automatic testing and you do not know why, then contact me - do not ignore it.

Automatic Testing and Scanners

Because of the way this testing and Scanners work, you need to obey a couple of rules.
You should only have one Scanner in your code.
You should not create that Scanner in a loop.

So this is wrong:

```
Scanner in = new Scanner(System.in);
Scanner in2 = new Scanner(system.in)
```

and so is this:

```
while(…) {
      Scanner in = new Scanner(system.in);
      …
}
```

Both of these will, unfortunately, work when you test from the keyboard but fail the automatic testing.

Guidance Notes

- *The biggest mistake you can make is failing to start the work early enough* - you need to not only get the code written and compiling, but also working and matching the output of the automatic tests. You need to give yourself time to deal with any hitches, and see me if you need to.
- *The second biggest mistake you can make is just to start writing code without thinking about what you are doing.* Don't just jump in and start writing - make a plan!
- *Break the program into parts* - there are three steps in this program: player one chooses and decides which barrel(s); player two chooses and decides which barrel(s); the program decides if the game is over and if so, who has won. Note that the code for the first two parts is nearly exactly the same.
- *Write and test in parts* - there is no need to try to do everything in one go: write the program in parts so you can test and check each part on it's own without worrying about the rest of it.
- *Don't be afraid to write test code* - if you're going to write your program in parts, you need write some extra test code. For example if you are writing the bit that decides if the game is over and who has won, you need to write some code to set the numbers of tokens in the bags to dummy values. Sometimes people shy away from writing this code when they start - thinking the work is 'wasted' if it's not code they are going to submit. But it makes it easier and quicker to solve the problems you face, so it's **not** wasted effort - it's less effort
- *Be careful if you use an IDE like Eclipse or Netbeans.* You are welcome to do this, but make sure that you can run your code against the test scripts as described above. There are videos on Blackboard showing you how to (a) create projects without package statements and (b) get your code out of Eclipse/Netbeans so you can test it against the scripts from the command line.
- *Don't be afraid to ask for help* - it's my job to help and guide you (not do it for you though) so please ask if you have problems
- *Don't be tempted by unfair practice* - it's sometimes tempting to submit someone else's solution (maybe changed a bit…), or get something off the internet, or ask someone else to do it. If you do this you risk serious penalties for breaching academic integrity. Remember we run submissions though software systems that detect these practices.
- *Read the marking criteria* - you have access to the marking rubric on Blackboard that tells you what will be assessed, how many marks are for each part, and what you need to do to get a mark in each category. Please pay attention to this.
- *Don't get carried away* - every year I get advanced solutions to coursework that gets a lower mark than some simpler solutions. This is because the advanced solutions focus too much on the 'advanced' things and forget the simpler basics (which are in the marking scheme).

Submission

Please note the submission details very carefully.

- You must **submit** your work via the coursework link on Blackboard by **13.00 on Friday 9th November 2018**. Do not email your work to me - also, ignore the submission time on science.swan.ac.uk which only permits a limited number of choices - *the time on Blackboard is correct.*
- You must upload a file called `LastBiscuit.java` containing your Java code. Make sure you submit a file and do not just cut and paste your code into the text box on the submission page. Also do not submit the `.class` file - just the `.java` file
- ***Late submissions will receive a mark of zero*** - although you will get feedback. If you have **extenuating circumstances** which prevents you from submitting on time, *you must inform me as soon as possible* in order to make a case to remove the submission penalty. If you don't you will almost certainly not get the penalty lifted.
- I *strongly suggest* you make sure you are consistently using *either* sequences of four spaces *or* tabs for indenting. Do not mix both as this can often look like inconsistent indenting after it's been through Blackboard. You can use Notepad++ (or whatever other editor you are using) to automatically do this for you.
- You will receive **feedback** and a **mark** for your work on or before the **24th November**.
- On the same date I will release a range of **sample solutions** and **general feedback** - after that date I cannot accept any late submissions *regardless* of the reason.
- The **assessment criteria** for your work are defined by the **marking rubric on Blackboard**. This defines: the categories you will be assessed against; how many marks are available for each category; and what you need to do to get those marks. *It's important you study this carefully before you attempt the coursework - no matter how good your code is, if it does not meet the assessment criteria you will not get a high mark.*
- The **feedback** you will receive will consist of the marking rubric, identifying how well you have done in each category together with additional comments if appropriate.
- If you have **any problems with submission**, send me an email with **[submission]** (that *exact* text) in the title, with details of your problem and I'll advise you what to do - but do not include your actual coursework submission unless you are asked to.