THE UNIVERSITY OF WESTERN AUSTRALIA

School of Computer Science and Software Engineering

CITS1001 Object-oriented Programming and Software Eng

Document version number: 1.05.
(Updated advice on using *FileIO.java*)
(Updated advice on submission, assessment, and *allEqual*)
Check the CITS1001 web-site to ensure that you are aware of the latest version.

Project 1: Solving Number Sequence Puzzles

Submission deadline: 5pm, Friday 20 April 2018.
Value: 10% of CITS1001.
To be done individually.

You should construct a Java program containing your solution to the following problem. You must submit your program electronically using *cssubmit*. No other method of submission is allowed.

You are expected to have read and understood the University's guidelines on academic conduct. In accordance with this policy, you may discuss with other students the general principles required to understand this project, but the work you submit must be the result of your own effort.

You must submit your project before the submission deadline above. The penalty for late submission is described here.

Overview

A *number sequence puzzle* gives you a sequence of numbers and asks you to supply the next few numbers in the sequence, by inferring the rule that was used to generate the given numbers. For example, what are the next few numbers in the following sequence?

1, 4, 19, 52, ...

This sequence could be generated by the polynomial

x^3 - 4x + 4

in which case the next few numbers would be

109, 196, 319, ...

Every number sequence of length *n* can be generated by a polynomial of degree *n-1* (or less). In this project you will write a Java program that takes a number sequence puzzle as input, and that derives and displays the simplest generating polynomial for that sequence.

The Algorithm

The algorithm that you will implement is iterative. Each iteration of the main loop derives one term of the polynomial. For example, starting from the sequence of numbers <1, 1, 5, 13, 25>:

1. Repeatedly calculate the differences between adjacent numbers in the sequence, until every number in the sequence is the same:

   <1, 1, 5, 13, 25> ⇒ <0, 4, 8, 12> ⇒ <4, 4, 4>

2. The exponent of the new term is the number of steps above, i.e. 2.

3. The coefficient of the new term is the number on the final sequence divided by the factorial of the exponent, i.e. 4/2! = 2. Thus the new term is 2x^2.

4. Subtract the new term from each element of the original sequence. From the first element subtract 2 * 1^2, from the second subtract 2 * 2^2, from the third subtract 2 * 3^2, etc. This leaves <-1, -7, -13, -19, -25>, which is the sequence of numbers to start the next iteration.

The second iteration with this example yields the term -6x, and leaves the sequence <5, 5, 5, 5, 5>. Now all of the numbers in the sequence are the same, so the main loop terminates. The final term in the polynomial is the number on the final sequence, i.e. 5, and the complete polynomial is

2x^2 - 6x + 5

Make sure that you can perform this algorithm yourself (on paper) before you start writing your program. Practice on <1, 4, 19, 52> from above and on sequences from the sample file until you are sure.

Tasks

You are required to write three Java classes:

- *Term.java*
- *Polynomial.java*
- *Sequence.java*

Each link above gives you a skeleton of the required class, including instance variables, constructors, and methods. You are required to complete the methods whose bodies are marked with the comment TODO. Make sure you understand the connections between the classes, and where each method fits into the algorithm described above.

Sequence.java provides the (completed) method *solveFileSequences* so that you can easily test your program on a file like the sample file. To read files, you also need to download this class:

- *FileIO.java*

Add *FileIO.java* into your project, download project1sample.txt into the same folder, then just apply *solveFileSequences* to the filename.

Here is a suggested order for tackling the methods, and some tips about specific methods. If we add to this list, we will update the document version number.

- *factorial*
- *allEqual* - this method will be marginally easier with a for-each loop, rather than a for loop.
- *displaySimple* - look up the library method *Math.abs*.
- *differences* - this method needs to create a new data structure to hold the result, then calculate the differences and store them for returning.
- *nextTerm* - this method implements the first three steps of the algorithm described above: repeatedly calculate differences until all the numbers on sequence are equal as described in Step 1, then calculate the coefficient and exponent of the new term as described in Steps 2 and 3. *nextTerm* should call *allEqual* and *differences*.
- *termUpdate* - look up the library method *Math.pow*.
- *updateSequence* - this method implements Step 4 of the algorithm: use the argument term to update each element in the sequence as appropriate. *updateSequence* should call *termUpdate*.
- *displayImproved* - this method has several special cases to get it exactly right.
- *displayPolynomial* - look up the library methods *String.charAt* and *String.substring*. This one has only one special case.
- *solveSequence* - this method implements the entire algorithm: repeatedly create terms and add them to the polynomial until all the numbers on the sequence are equal. Don't forget the last term or the display! *solveSequence* should call *allEqual*, *nextTerm*, and *updateSequence*.

Submission

Submit your three completed files *Term.java*, *Polynomial.java*, and *Sequence.java* via *cssubmit*.

Your file names, class names, and method names and signatures must match the specifications *exactly*. The marking process is partly automated and any deviation in the names or signatures will cause problems, which will be penalised. It is ok to add other methods if you like, as long as you do not change the names or signatures of existing methods.

Common mistakes are to submit *.class* files in place of *.java* files, or to submit our test classes in place of your code. If you do one of these, you will be notified as soon as we become aware, but **you will be due for any applicable late penalty**. It is easy to check your submitted files after you have submitted them - do it!

Assessment

Your submission will be assessed on

- completeness: how many of the methods you have written;
- correctness: whether your methods implement the specifications exactly, and whether your output is in the same format as the sample file;
- clarity: whether your code is clear to read and well-constructed.

Testing classes are provided for you to validate your program before submission:

USEFUL LINKS

- Unit outline
- Unit timetable 2018
- Java 8 API
- help1001
- csentry
- cssubmit
- csmarks
- IT help for students
- Compulsory online modules

- *TermTest.java*
- *PolynomialTest.java*
- *SequenceTest.java*

Note however that the correctness of your program is your responsibility. **The testing classes are (intentionally) quite brief, and a program that passes all of the tests is not guaranteed to be free of errors.** "Testing shows the presence, not the absence of bugs." -- E.W. Dijkstra.

Feel free to augment the testing classes with more tests of your own. Also compare your program's output with the sample file.

A good place to start wrt clarity is the CSSE Java style guide.

Marks breakdown

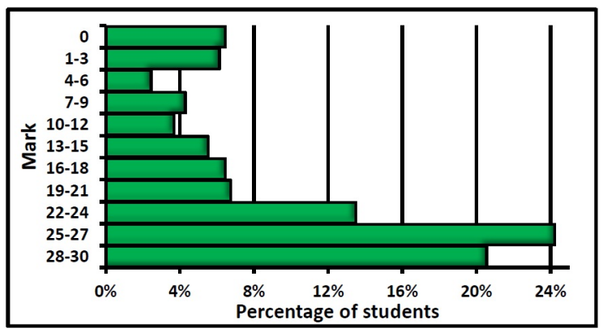
There were 25 marks for correctness, distributed as follows.

- *factorial* - 1
- *allEqual* - 2
- *displaySimple* - 1
- *differences* - 2
- *nextTerm* - 4
- *termUpdate* - 1
- *updateSequence* - 2
- *displayImproved* - 4
- *displayPolynomial* - 3
- *solveSequence* - 5

Methods were assessed independently, e.g. if your *allEqual* had a problem, that wouldn't affect other methods that called it. If your method got all green ticks from the marking class, you would have got full marks for that method; otherwise you would have been penalised according to the severity of your error.

There were 5 marks for code clarity. Mostly you would have lost a mark if you did something significant contrary to the CSSE Java style guide, or if the marker deemed that your code for a method was significantly more complicated than it needed to be.

The overall average was 20/30. The distribution of marks is shown below.



My solution

This is my solution. Take a look, and compare the way my solution works with the way that yours works.

- *Term.java*
- *Polynomial.java*
- *Sequence.java*

This is the marking class that was used.

- *Project1Test.java*

Try out the marking class, and try to understand how your program goes wrong (if it does!). Please direct questions to help1001.

The method *testdifferences* has a known weakness that we discovered after marking started; it gives out green ticks for some methods that have minor errors. (We have decided to let this weakness go.) There will be a prize** for the first person to email Lyndon with a concise description of this weakness.

**All prizes are awarded or not solely at Lyndon's discretion.

**Update: this prize has been won by Matthew Chidlow. Well done! The weakness is that in all of the tests performed on differences, the argument has the same value as the field variable. Therefore a method that processes the field variable instead of the argument will pass the tests, even though it shouldn't. The version on this page has one extra test which covers this gap; but in marking we will stick with the original version, for equity reasons. We believe that this error will be rare and that it will show up in other tests anyway.

Help!

The quickest way to get help with the project is via help1001. You can ask questions, browse previous questions and answers, and discuss all sorts of topics with both staff and other students.

Please read and follow these guidelines.

- **Do not post project code on help1001.** For obvious reasons, this behaviour undermines the entire assessment process: as such you will be liable for punishment under the University's guidelines on academic conduct.
- Before you start a new topic, check first to see if your question has already been asked and answered in an earlier topic. This will be the fastest way to get an answer!
- When you do start a new topic, give it a meaningful subject. This will help all students to search and to make use of information posted on the page.
- Feel free to post answers to other students' questions. And don't be afraid to rely on other students' answers: we read everything and we will correct anything inaccurate or incomplete that may be posted from time to time.
- Be civil. Speak to others as you would want them to speak to you. Remember that when you post anonymously, this just means that your name is not displayed on the page; your identity is still recorded on our systems. Poor behaviour will not be tolerated.
- **Do not post project code on help1001.** This one is worth saying twice.

If you have a question of a personal nature, do not post to help1001: please email Lyndon or Arran instead.

Good luck!