Curtin University — Discipline of Computing

Software Engineering Concepts (COMP3003/6007)

Semester 2, 2020

Assignment 2

Due: Thursday 29 October, 23:59:59 AWST / 21:30:00 IST

Weight: 30% of the unit mark.

Your task is to create an equation evaluator; that is, a program that inputs a mathematical expression from the user that includes a variable `x`, and then evaluates that expression for different values of `x`. (What happens with the resulting calculations depends on the plugins loaded...)

1 Basic Operation

You may implement a purely text-based user interface if you wish. You may alternatively implement a GUI, but there will be no marks awarded for doing so.

The program must be Java-based, and must give the user two basic options:

(a) Manage plugins. The user must be able to show currently-loaded plugins and add new plugins. The program *does not* necessarily have to "remember" the plugins from one execution to the next. (You may also implement the ability to remove plugins at runtime if you wish, although this is not a requirement.)

See Section 3 for what a plugin should be able to do.

(b) Evaluate an equation for a range of different values of `x`. The user must be asked to enter the following:

(i) An expression – a string containing numbers, mathematical operators, and the letter "x".

(ii) A range of `x` values; specifically: the minimum value, the maximum value, and the increment. All of these should be real numbers.

The program must then invoke the following algorithm:

```
input expression
input minX, maxX, incX

for x = minX; x <= maxX; x += incX
{
    y = evaluate expression for x
    ...
}
```

1.1 Update (2020-10-09)

You will need to add the following to your `build.gradle` file:

```
run {
    standardInput System.in
}
```

This allows your application to access console input when run using "`./gradlew run`", which will be critical for any console-based UI. (By default, for obscure reasons, "`./gradle run`" doesn't allow your application to access console input when run like this.)

2 Scripting

How should the application "evaluate expressions"? For the purposes of this assignment, it is required that you do this by invoking a scripting language[1]. When the user enters an expression containing "`x`" (such as "`3*x*x - 5*(x + x*x)`"), this should constitute a valid programming language expression. Your application must set a value for `x`, ask the scripting engine to evaluate the expression, and obtain the result.

You do not necessarily have to use Jython, but it is recommended simply because it is the one covered in the notes. (If you choose to use a different scripting engine, it must be Python or JavaScript-based.)

The mechanics of getting a value *out* of the scripting engine aren't specifically covered in the notes, but you can refer to the following code snippet (for Jython):

```
import org.python.core.*;
import org.python.util.*;


...
PythonInterpreter py = ...;
String expression = ...;
...
double result = ((PyFloat) py.eval("float(" + expression + ")")).getValue();
```

Specifically, the `PythonInterpreter.eval()` method will evaluate a Python *expression* and return the result. In this case, we're wrapping the expression in a Python `float()` call, in order to guarantee the resulting datatype. However, we then need to convert that result (a `PyFloat` object) to a normal Java `double`.

Another trick that you will (eventually) need is the ability to run Java methods as if they were functions in Python. There are several approaches to this, but the simplest is to first declare a series of static methods:

```
package myPackage;
public class MyStaticMethods
{
    public static ... myFunction(...) { ... }
    ...
```

[1] To be fair, this is also probably the easiest way to do it.

```
}
```

Then, execute a Python import statement as follows:

```
PythonInterpreter py = ...;
...
py.exec("from myPackage.MyStaticMethods import myFunction");
...
```

At this point, any Python code will be able to call `myFunction()`.

3 Plugins

The application must support a Java-based plugin system, and this is designed to support key features not present in the core application. (Indeed, if you look closely at Section 1, you'll see that the core system doesn't *output* anything!)

You must create a plugin API, a plugin loading mechanism, and a set of specific plugin implementations.

3.1 The Plugin API

The plugin API is a specification, written as a set of Java interfaces (and/or abstract classes) for how the core system and the plugins interact with each other. It must be workable and demonstratable, but no more complex than necessary.

(A warning: although it will be written in code, you cannot create an API through a "run-it-and-see-if-it-works" approach. You must be able to reason about your API design as a separate issue from whether your code works, with some foresight into what kinds of plugins *someone in the future* might want to create.)

Here is a set of things that your plugin API must allow plugins to do:

- Receive the four different input values: the equation, and the minimum/maximum/increment values for `x`.
- Be notified *when* each individual value of `y` has been calculated (noting that this is a *timing* requirement).
- Receive those values of `y` (e.g., so that they can be output in some form, although the manner of outputting is beyond the scope of the API itself).
- Create additional mathematical functions, with any names, that may appear in the expression to be evaluated.

3.2 Plugin Implementations

You must provide a series of three Java-based plugins to demonstrate your plugin API, as follows:

- One plugin should provide a running progress indicator that displays the percentage complete (starting at 0%, and eventually reaching 100%). If there is a large range between

the minimum and maximum `x`, and/or the increment is very small, then the total time to calculate all `y`'s may be substantial.

- One plugin should write out all values of `x` and `y` to a two-column CSV file. For the purposes of this assignment, you are required to use Apache Commons CSV[2] to do this.
- One plugin should provide additional mathematical functions `factorial` and `fibonnaci` for use in expressions.

  These normally take an non-negative integer, so they will first have to round their parameter, and return 0 for any negative parameter values. (In fact, I'm not particularly worried about the exact implementation of these functions, as long as they're reasonably within the spirit of their usual definitions.)

3.3 Plugin Loading Mechanism

As per Section 1, your core application will need a way to add plugins at runtime.

Your application must support zero-or-more plugins being loaded at the same time in any combination with one another. It *must not* hard-code the details of any specific plugins. However, you may assume that the user knows the class name for each plugin to be loaded/unloaded (or, more precisely, whichever class in each plugin serves as that plugin's "starting point").

4 Native Plugin Implementation

Provide a fourth plugin, that re-implements one of the other three plugins in C/C++, using JNI. The choice of which of the three plugins to re-implement is up to you, but the C/C++ version will have the same requirements as the Java version.

It is allowed and expected to have one or more Java classes here, in order to interact with the Java API. However, those classes shouldn't contain any actual Java code other than to load the native library. All methods in those classes should be declared `native`.

5 Build System

You are required to use a Gradle-based build system, with the following characteristics:

- Your project should have a subproject for each of the four plugin implementations mentioned in sections 3.2 and 4. (In fact, the C/C++ plugin may need to occupy two subprojects.)
- Your project should have another single subproject for the API. This is expected to consist mostly of Java interfaces or abstract classes. It should contain no actual code, or extremely minimalistic code.
- You may place the core application either in the root project, or in its own subproject.
- Each third-party library used must be specified as a dependency, for the appropriate part of the project. (The project *must not* depend on installed software other than the OS, the JDK, and a C/C++ compiler.)

[2] See the documentation at commons.apache.org/proper/commons-csv/apidocs/index.html.

The marker will initially assume that they can compile and run your project using "`./gradlew run`". You should ensure that this works.

6 Documentation

You are *not* required to write a report. However:

- Your API should be well documented within the code, at the point of declaration.

  It should be clear from your comments how to use the API to write another plugin (although you do not need to describe build-related issues, just coding issues).

- The rest of your code should be well commented in general.

- If you need to clarify anything important for the marker, put your remarks inside a "`README.txt`" file.

  In particular, if you cannot make "`./gradlew run`" work by itself, then you're strongly advised to say exactly what the marker needs to do (within reason) to run your code.

7 Mark Allocation

This assignment will be marked out of 30 marks, as follows:

(a) **[5 marks]** – Use of scripting (Section 2).
(b) **[5 marks]** – API design (sections 3.1 and 6).
(c) **[5 marks]** – Java plugin implementations (Section 3.2).
(d) **[3 marks]** – Plugin loading mechanism (sections 1 and 3.3).
(e) **[5 marks]** – Native plugin implementation (Section 4).
(f) **[7 marks]** – Build logic and project structure (Section 5).

Your code should actually work, and adhere to reasonable standards of readability and maintainability. If this is not the case, some/all of the above criteria may be considered unfulfilled.

8 Submission

Submit your assignment electronically, via Blackboard, before the deadline. To submit, do the following:

(a) Fill out and sign a declaration of originality. A photo, scan or electronically-filled out form is fine. Whatever you do, ensure the form is complete and readable! Place it (as a .pdf, .jpg or .png) inside your project directory.

(b) Zip up your entire project directory as-is (as a .zip or .tar.gz file). Leave nothing out.

(c) Submit your zip/tar.gz file to the assignment area on Blackboard.

(d) Re-download, open, and run your submitted work to ensure it has been submitted correctly.

You are responsible for ensuring that your submission is correct and not corrupted. You may make multiple submissions, but only your newest submission will be marked. The late submission policy (see the Unit Outline) will be strictly enforced.

Please note:

- DO NOT use WinRar (or the .rar format in general).
- DO NOT have nested zip/tar.gz files. One is enough!
- DO NOT try to email your submission as an attachment. Curtin's email filters are configured to *silently discard* emails with potentially executable attachments.

In an emergency, if you cannot upload your work to Blackboard, please instead upload it to Google Drive, or a private Github repository, or another online service that *preserves immutable timestamps* and is *not publicly accessible*.

9 Academic Integrity

Please see the *Coding and Academic Integrity Guidelines* (available alongside this specification on Blackboard).

In summary, this is an assessable task. If you use someone else's work or assistance to help complete part of the assignment, where it's intended that you complete it yourself, you will have compromised the assessment. You will not receive marks for any parts of your submission that are not your own original work. Further, if you do not *reference* any external sources that you use, you are committing plagiarism and/or collusion, and penalties for academic misconduct may apply.

Curtin also provides general advice on academic integrity at
http://academicintegrity.curtin.edu.au/.

The unit coordinator may require you to provide an oral justification of, or to answer questions about, any piece of written work submitted in this unit. Your response(s) may be referred to as evidence in an academic misconduct inquiry.

End of Assignment