Assignment 1

Due Date

This assignment is to be completed in pairs. The assignment is due at 3PM Wednesday March 25th 2020 and should be completed with a partner. You and your partner should work together on all of the design and programming. It should be done using the *pair-programming* methodology and not by division of labour. Use the "Assignment 1 Pairs" groups on MyLO to find and register your pair.

Background

The local chocolate factory is going to try a curry flavoured dark chocolate. They need to trial the amount of curry which they add so that the best flavour can be determined. This concentration can range from no curry-all chocolate (represented numerically as 0.0) up to all curry-no chocolate (represented numerically as 1.0).

Banana is sometimes eaten with curry to reduce the 'heat' of the flavour. The chocolatier plans to add very small banana pieces to the chocolate too. The amount needs to be determined to match the curry and this will be trialled for a defined range too.

Modelling will be used to simulate the trials of different amounts of curry and banana to optimise taste. To reduce cost, a 'sweet spot' of 500 Sweet Flavour Accents (SweetFAs — the well-known international standard unit) has been set as the target. Once a combination of curry, banana, and dark chocolate reaches/exceeds 500 SweetFAs the trial will conclude.

The food scientists have derived an equation for the tastiness of the chocolate:

> Banana factor = 120 – number of banana pieces
> Flavour = curry concentration * (800 – (banana factor * banana factor))

Program specification

You and your partner must write a Java program to calculate and display all possible flavours for a range of banana pieces and curry concentrations.

First, your program should obtain the following information from the user:

- the minimum number of pieces of banana desired to be included;
- the maximum number of pieces of banana desired to be included; and
- a step size for the curry concentration

Once these values have been entered the program should check if the values entered are legal (all positive, the minimum less than or equal to the maximum, and with the difference between the minimum and maximum number of banana pieces not exceeding 10).

If the values entered are not legal, the program should print an error message.

If the values entered are valid, the program should produce a table of results displaying the flavour of the chocolate for each possible number of banana pieces in the range specified by

the user. Each column in the table should show the results for a specific number of banana pieces, and should have this number displayed at the top of the column. Each row in the table should show the results for a particular curry concentration step, and should have that value displayed on the left to 2 decimal places.

The first row should have a curry concentration equal to the step size given by the user, and the curry concentration should increase by that step size for each subsequent row. The program should keep adding rows to the table until either the curry concentration reaches/exceeds 1.0, or until a chocolate bar is produced reaching the threshold flavour of 500 SweetFAs.

All calculations of flavour should be done using `doubles`, and the results should be displayed to 1 decimal place.

Your solution should run in *Dr Java* and comprise the following files:

- `SweetFAInterface.java` and `SweetFA.java` — the *Flavour* ADT. You should have the following methods: a constructor, 'getters' and 'setters' for the curry concentration object and number of banana pieces, and 'doer' methods to calculate the flavour, determine whether the flavour exceeds the threshold, and convert the flavour to each of a `double` and a `String`.
- `CurryInterface.java` and `Curry.java` — the specification and implementation (respectively) of a curry concentration. A curry concentration should be implemented as a class with instance variables which consist of:
    - the current curry concentration value (as a `double`); and
    - the curry step value (as a `double`).

  The class should also contain methods as specified in the interface, i.e. a constructor, setter and getter for the step value, getter for the concentration value, a doer which checks whether the concentration value is at the maximum (and if so returns `false`, and if not increases the concentration value by the step and returns `true`), and a method to convert the concentration value to a `String`.
- `AssigOne120.java` — the file which contains the `main()` method.

A starting point is available on MyLO for you to download. You should complete all of the required files.

Two example executions are shown below:

```
The Local Chocolate Factory
---------------------------

Enter the minimum number of banana pieces: 100
Enter the maximum number of banana pieces: 200
Enter the curry concentration step size: 0.14

Sorry, these are not legal.
```

```
The Local Chocolate Factory
---------------------------

Enter the minimum number of banana pieces: 102
Enter the maximum number of banana pieces: 109
Enter the curry concentration step size: 0.14

        Number of banana pieces
Curry   102     103     104     105     106     107     108     109
0.14     66.6    71.5    76.2    80.5    84.6    88.3    91.8    95.1
0.28    133.3   143.1   152.3   161.0   169.1   176.7   183.7   190.1
0.42    199.9   214.6   228.5   241.5   253.7   265.0   275.5   285.2
0.56    266.6   286.2   304.6   322.0   338.2   353.4   367.4   380.2
0.70    333.2   357.7   380.8   402.5   422.8   441.7   459.2   475.3
0.84    399.8   429.2   457.0   483.0   507.4   530.0   551.0   570.4
```

Program Style

Your program should follow the following coding conventions:

- `final` variable identifiers should be used as much as possible, should be written all in upper case and should be declared before all other variables
- variable identifiers should start with a lower case letter
- every `if` and `if-else` statement should have a block of code (i.e. collections of lines surrounded by `{` and `}`) for both the `if` part and the `else` part (if used)
- every `do`, `for`, and `while` loop should have a block of code (i.e. `{}`s)
- the keyword `continue` should not be used
- the keyword `break` should only be used as part of a `switch` statement
- opening and closing braces of a block should be aligned
- all code within a block should be aligned and indented 1 tab stop (or 4 spaces) from the braces marking this block
- instance variables should be used sparingly with parameter lists used to pass information in and out of functions
- local variables (excluding loop counters) should only be declared at the beginning of methods (either as parameters or otherwise)
- commenting:
    - There should be a block of header comment which includes at least
        - file name
        - student names
        - student identity numbers
        - a statement of the purpose of the program
        - date
        - the percentage of the work completed by the authors — 50:50 is expected and assumed but reasons should be given if it is more/less than this
    - Each variable declaration should be commented
    - There should be comments that identify groups of statements that do various parts of the task
    - Comments should describe the strategy of the code and should not simply translate the Java into English

Marking scheme

Task/Topic	Maximum mark
Program submitted in form required	
Program submitted correctly	1
Program compliles without error and runs to completion with test data	1
Program operates as specified	
`SweetFA.java` correctly completed	5
`Curry.java` correctly completed	7
`AssigOne120.java` correctly completed	10
Program Style	
Does not unnecessarily repeat tests or have other redundant/confusing code	3
Uses `final` variables where appropriate	2
Uses pre-existing methods (`format`/`nextInt`/`nextDouble`/`min`/`print`/`println`/etc)	2
Uses correctly the Java naming conventions	2
Alignment of code and use of white space makes code readable	3
Always uses blocks in branch and loop constructs	3
Meaningful identifiers	2
Header comments for the program (author, purpose, date, filename etc)	2
Each variable declaration is commented	3
Comments within the code indicate the purpose of sections of code (but DO NOT just duplicate what the code says)	2

What and how to submit

What to submit

- You should submit all `.java` files.

How to submit

- Log in to MyLO and navigate to the `Assignments` tool under the `Assessments` menu in the top tool bar.
- Select `Assignment 1` from the list of available drop-boxes.
- Click on `Add a File` and follow the instructions to attach your source code files and then click `Add`. Then click `Submit`.

If you want to re-submit, please just do so.

Please note: only submission is required for the pair.

Plagiarism and Cheating:

Practical assignments are used by the Discipline of ICT for students to both reinforce and demonstrate their understanding of material which has been presented in class. They have a role both for assessment and for learning. It is a requirement that work you hand in for assessment is your own.

Working with others

One effective way to grasp principles and concepts is to discuss the issues with your peers and/or friends. You are encouraged to do this. We also encourage you to discuss aspects of

practical assignments with others. However, once you have clarified the principles *of the question*, you must express them in writing or electronically entirely by yourself in your pair. In other words you and your partner must develop the algorithm to solve the problem and write the program which implements this algorithm yourselves. You can discuss the question, but not the solution. Assistance with the solution should be provided by staff during tutorials or consulting times.

Cheating

- Cheating occurs if you claim work as your own when it is substantially the work of someone else.
- Cheating is an offence under the Ordinance of Student Academic Integrity within the University. Furthermore, the ICT profession has ethical standards in which cheating has no place.
- Cheating involves two or more parties.
    - If you allow written work, program print-outs, or electronic versions of your code to be viewed, borrowed, or copied by another student then you are *an equal partner* in the act of cheating.
    - You should be careful to ensure that your work is not left in a situation where it may be used/stolen by others.
- Where there is a reasonable cause to believe that a case of cheating has occurred, this will be brought to the attention of the unit lecturer. If the lecturer considers that there is evidence of cheating, then no marks will be given to any of the students involved and the case will be referred to the Head of School for consideration of further action.

Julian Dermoudy, March 3rd 2020.