University of Stirling
Computing Science and Mathematics
CSCU9A2 Java assignment Spring 2019
A simple interactive product sales database

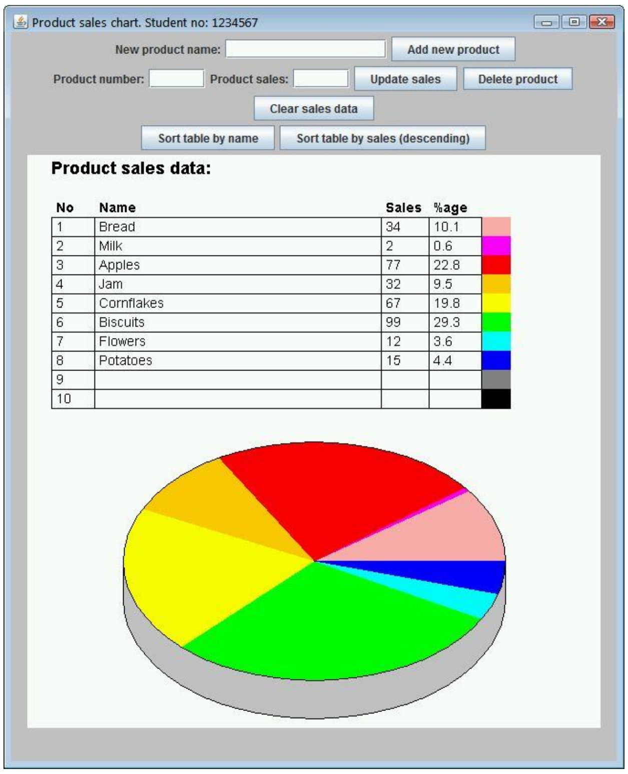
Deadline: Tuesday 26 March, 5.00 pm **Worth: 40% of final mark**

This description and the starter code can be found on the module Canvas pages, and also on Groups on Wide\CSCU9A2\Assignment

Please read this document right through before starting your work.

Introduction

In this assignment you will be completing a Java program that runs a very simple interactive product sales database – as in the example shown to the right: The program holds up to 10 product names and sales data. A table of the sales data is always displayed, together with a coloured pie chart showing the relative proportions of the sales of the different products. At the top of the window there are buttons and text fields for functions for managing the products and sales data – it is the *methods and algorithms* that implement these functions behind the scene that you are required to complete.



Product sales data:

No	Name	Sales	%age	
1	Bread	34	10.1	🟫
2	Milk	2	0.6	🟪
3	Apples	77	22.8	🟥
4	Jam	32	9.5	🟧
5	Cornflakes	67	19.8	🟨
6	Biscuits	99	29.3	🟩
7	Flowers	12	3.6	🟦
8	Potatoes	15	4.4	🟦
9				⬜
10				⬛

You are given an almost-complete Java program, `ProductDatabase.java`*, on the Canvas Assignment page, and in the folder* `Groups on Wide\CSCU9A2\Assignment` *You must take your own copy of this program to work on in a BlueJ project in the usual way. In these locations you will also find a collection of images illustrating the intended functions of the program, to be viewed in conjunction with the description that follows.*

- The user interface and event handling are fully complete and correct, *and you must not change them*.
- Behind the scene, inside the program, the product database is held in a collection of arrays – again the arrays are complete, *and you must not change their declarations.*
- The event handlers for the advanced function buttons use **six different methods** whose purpose is to access or update the data held in the arrays (for example to add a new product, or to update the sales figure for a specified product). These methods are present in the code but are *incomplete*, and it is your task to complete as many of them as you can: to function correctly, and in good programming style. ***The method headers for these methods must not be changed* (as they are called from the event handlers), but *you may add further helper methods if you wish.***
- There is a **seventh method** (`computePercentages`) that is not called directly, but needs to be called from your methods to calculate the percentage sales of each product (see the example above). Again, this method is present in the code but is *incomplete*, and it is your task to complete it to function correctly, and in good programming style.

Your submission for the assignment should complete as many of the incomplete methods as possible — you do not need to complete all of them in order to get credit. The seven methods contribute up to the following number of marks each to the overall mark for the assignment:

`addNewProduct`	10
`updateSales`	10
`deleteProduct`	20
`sortByName`	20

`computePercentages`	10
`clearAllSales`	10
`sortBySalesDescending`	20

Your solution for each of the seven methods will be assessed according to the following criteria (based on the University's Common Marking Scheme):

Mark range	**Criteria**
70-100 (1st)	Fully correct results in all situations; well coded, with well chosen identifiers and comments
60-69 (Upper 2nd)	More or less fully correct results; perhaps with occasional faults; competent coding and comments
50-59 (Lower 2nd)	More or less fully correct results; perhaps with occasional faults
40-49 (3rd)	Partial attempt that mostly works correctly
30-39 (Marginal Fail)	Partial attempt with substantial faults in the results
0-29 (Clear Fail)	Little or nothing attempted

In judging whether the Java code that you have written is of "high quality" we shall take into account the following **Assessment criteria**:

- correctness of operation,
- appropriate use of Java constructs,
- suitably informative choices of variable and method names,
- consistency, legibility and tidiness of program layout,
- effective use of comment text.

Independent work

Note that we require that the work you submit is your own, independent work. "Plagiarism" means presenting the work of others as your own. The University takes a very serious view of plagiarism, and the penalties can be severe. The University has a formal policy on plagiarism which can be found at

http://stir.ac.uk/1x0

However, this should not exclude discussion of general issues. Please read the "Note on Independent Work" available via the CSCU9A2 Canvas assignment page. Be warned: copying or borrowing code does not necessarily guarantee a quality product, and may damage your grade, as may showing or giving your code to someone else! We will be submitting all the solutions to a "similarity" checking service, and **any apparent cases of collusion will be investigated carefully, and the appropriate penalties applied.**

General description of the problem

The product sales database program contains a 'database' of products: their names, the most recent sales figure for the product, and the percentage that product has of the total sales. The database is quite small in this version, but in principle it could be quite large. Of course, in this exercise, the 'database' will be a little unrealistic: the information is built-in to the program (whereas in a 'serious' system it would, perhaps, be read in from a file or held in a proper database). The database inside the program is a parallel collection of arrays of Strings (for the names), ints (for the sales figures) and floats (for the percentages). In order to avoid a product "number" of 0, the arrays are only used from index 1 onwards. The values in the elements at index 1 of each array comprise the information about one product, the values in elements at index 2 comprise the information about another product, and so on. There is a single variable, `actualProducts`, that holds the number of products for which details are currently held in the database, and a single variable, `totalSales`, that should always be updated to contain the current total of all sales in the table.

User interface buttons are provided to allow the user:

- to add a new named product (with zero sales initially)
- to update the sales figure for a particular product
- to delete a specified product
- to clear all the sales data (but keeping the product names)
- to sort (re-order) the sales table in alphabetic order of product names
- to sort (re-order) the sales table in descending order of the sales figures

The Add new product button *almost* functions correctly – it does not detect when the user is attempting to add another product to a *full* database. This is because the method called by the button's event handler is incomplete.

The other five buttons *do not work*: The user interface code for them is fine, but it calls core data processing methods to carry out the necessary accesses/updates to the data in the arrays, ***and those methods require completing***. The incomplete methods are listed in the table at the top of page 2. ***You will find them in the code below line 473 (see ////////////////////////////////).***

You should insert your student number in place of 1234567 in line 111.

All other work is on the seven method bodies: You must complete the `addNewProduct` method, and implement full method bodies for `computePercentages`, `updateSales`, `deleteProduct`, `clearAllSales`, `sortByName` and `sortBySalesDescending`.

Note: You MUST NOT alter any other parts of the program:

- the GUI parts are complete and correct
- the array declarations are complete and correct
- the method headers are complete and correct

Advice — Incremental Development

Be very wary of tackling programming using the "Big Bang" approach: don't try to create the entire final program before entering it and testing out your ideas. That way lies frustration and poor progress.

We advise that you develop programs one step at a time – in this assignment perhaps one method at a time. At each step you can design and enter a little bit more of the final program. Of course, when you run the program at each step it won't do everything that the final program is supposed to, but you can check that the design so far is functioning correctly. [Further, if you run out of time doing the assignment, then you can submit your most recent correctly functioning version, confident that it does something properly — a large but non-functioning program is hard to give credit for.] This is an important technique in developing big programs: we can build a complete, compilable and runnable program without having designed all the details.

The incremental development approach is valuable for various reasons: If you have a working version of the program and you add only a little new code at a time, when something turns out to be faulty then it should be fairly clear where to look for the fault! (Clearly, if you have just typed in 100 lines of Java and it doesn't work, then locating the fault might be like looking for a needle in the proverbial haystack!) It is also very reassuring to see some facilities coming “on-line” at an early stage — rather than working to complete the whole program before trying it out! Good psychology and good methodology.

Keeping backups of your work

I **very strongly recommend** that you make backup copies of your work from time to time — perhaps after each successfully completed development step, or more frequently such as every day. It is easy to do this: Find the folder *containing* your project folder, click *once* on the icon for your project folder to highlight it, select **Copy** from the **Organize** menu (or right-click, **Copy**), and then select **Paste** from the **Organize** menu. You will see a new folder appear called `Copy of ...` (or `Copy (2)...`, etc for subsequent copies). If you ever need to backtrack, you can just copy your `.java` file from the most recent `Copy...` folder to the main project folder. I suggest that you treat the `Copy ...` folders as your *safe archive — leave them alone and continue your work in your main project folder*.

Keeping regular print-outs of your work can also be a life saver.

In past years a number of students have found themselves in extremely distressing circumstances as a result of ignoring the advice above.

Please remember that computers and related devices are not perfectly reliable, and people do make mistakes, so simple precautions are appropriate. If you carry out this assignment work on your own computer, remember to keep backups on removable disks, pen drives or in the cloud, but do not keep your only copy of your work on removable disks or pen drives.

Submission instructions

You must submit your assignment work through Canvas: *You should submit exactly one file, your final version of `ProductDatabase.java`*, through the assignment submission link on the CSCU9A2 assignment page before the deadline stated above. [Note that this is not a Turnitin submission, as it does not deal with code, and so there will be no originality report.]

We will compile and test your applications, so you must make sure that what you submit compiles without errors — we are *not* going to correct errors nor delete incomplete code in order to see if anything works properly! You can delete or comment out incomplete code before submitting your application (check that it does then compile). If you submit an application that doesn’t compile, then we will do our best to award some credit for what we can see in the Java text, but you are unlikely to be awarded a mark higher than 40.

Anonymous marking: You are required above to include your student number in a String in the `main` method, but you ***must not*** include your name nor your username.

Extensions and non-submission

If you need an extension for a good reason (perhaps for medical reasons) then this can be arranged, but it *must* be discussed with the module organizer, Dr Simon Jones, as soon as possible after the problem becomes known.

In accordance with University policy, assessed coursework submitted late will be accepted up to seven days after the submission date (or expiry of any agreed extension) but the mark will be lowered by three marks per day or part thereof. After seven days the piece of work will be deemed a non-submission, the student will have failed to meet module requirements, *and this will result in the award of No Grade for the module as a whole*.

SBJ February 2019