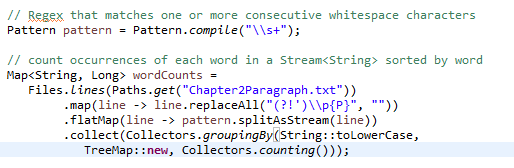
1. Streams of lines

This program uses lambdas and streams to summarize the number of occurrences of each word in a file then display a summary of the words in alphabetical order grouped by starting letter.

1. In the parsing phase:



* Define a pattern that matches one or more consecutive spaces.



* Read data from Chapter2Paragraph.txt into stream



* Replace all the characters that are punctuations but not “ ‘ ” (apostrophe) with empty string



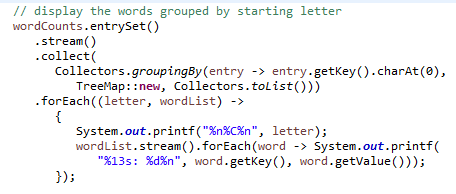
* Split the intermediate stream (including the whole article) into small pieces (words) by using pre-defined pattern



* Reduce the intermediate stream (words) into result map. Use the lowercase of each word as the grouping attribute (key), count the number of words of each group as the value for each group, and put the result into a new TreeMap. Note that the TreeMap will maintain its key in sorted order.



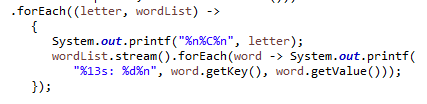
1. In the display phase



* Put all the entries of the result map into stream and use the first character of the word of each entry as grouping attribute, add all the entries of each group into a list and use this list as the “value” of the result map. Put the result into a new TreeMap whose key is the first character of each word and value is the entry list associated with that key.

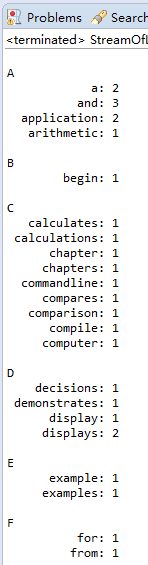
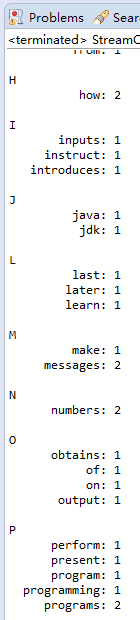
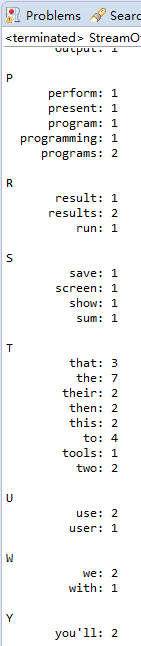


* Traverse the TreeMap: for each loop, first print the key (first letter) of the map, and then print all the corresponding words and the number of their occurrences.



1. Final running result:

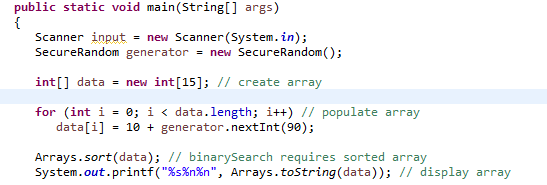
(The result was too long to show it in one screenshot)

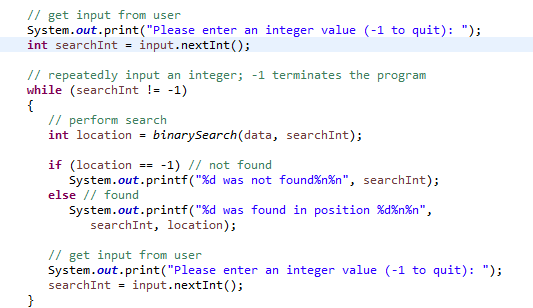
1. Binary Search

The program implements a binary search method including the implementation of binary search algorithm and corresponding print methods to show the searching process.

1. In the main method, it first generates an integer array (ranges from 10 to 99) randomly and sorts the array, so that we can use binary search to search an element in that array.

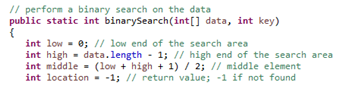


1. Then it gets searching element from user input and prints the searching result. User can keep searching elements for many times until entering a “-1”.



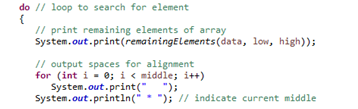
1. In the binary search method, it receives as parameters the array to search and the searching key.

* It first defines the two searching bounds and calculates the middle position.



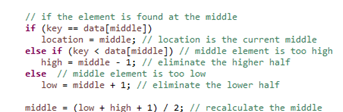
* Use a do - while loop to keep searching until no elements left or the searching element has been found in the array.

For each loop, first print the remaining elements to be searched, and print an indicator (“\*”) and spaces (for alignment) to show the position of middle element.

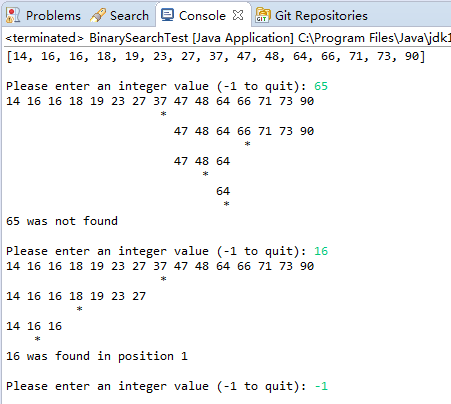




* Do a binary search algorithm, each time compare the middle element with the searching key, if they are equal, we found the searching key; if the searching key is less than the middle element, there’s no way to find the searching key on the right, so we go left (set “high” to middle - 1); otherwise, go right (set “low” to middle + 1).



1. Final running result:



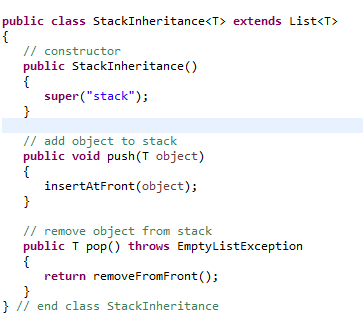
1. Stack Inheritance

“StackInheritance” class shows one way to implement a new data structure (stack) from an old data structure (List), which is called inheritance. “StackInheritanceTest” class builds some test cases to show how “StackInheritance” works

1. StackInheritance

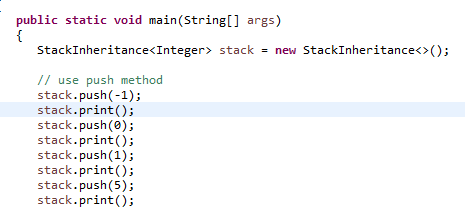
This class extends List, so that it can reuse all the methods (except private methods) of List class, such as isEmpty, print, and the two methods we use below.

It uses insertAtFront method and removeFromFront method to implement a stack; it always pushes the new element into the front and removes elements from front, so that the first pushed element will be at the end and will be popped at last.

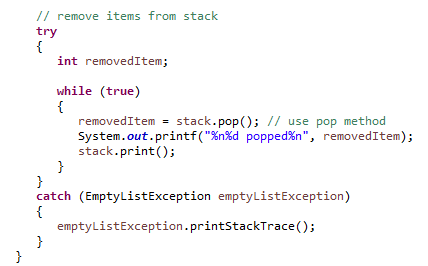


1. StackInheritanceTest

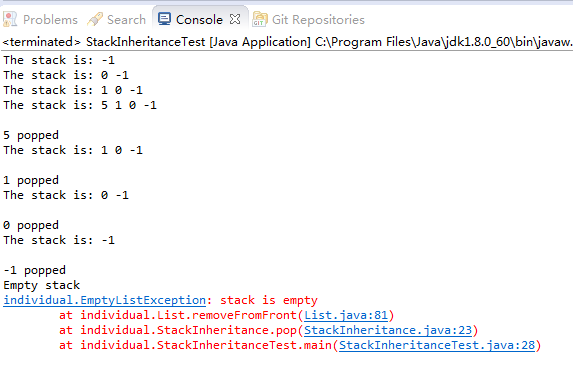
It first pushes four elements to the stack and prints them (inherit print method from List, which will print the name of the List and all the elements in the List), so that we can use them later on.



Then it keeps popping element one by one until the stack is empty. After each popping, it prints the popped element and the remaining stack. When the stack is empty, throws an exception and prints the exception stack trace information.



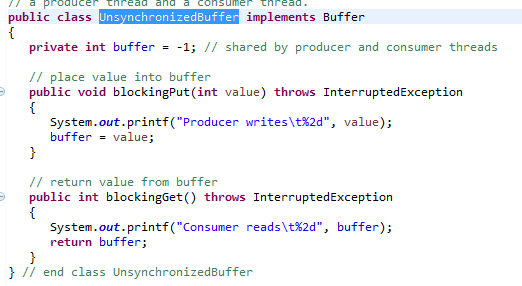
1. Final running result:



1. Shared Buffer

This program shows how an unsynchronized buffer leads to an unpredictable and inaccurate result.

1. In the unsynchronized buffer, it simply implements an integer value to represent the shared data, and corresponding get and put methods to get, modify and print the value involved.

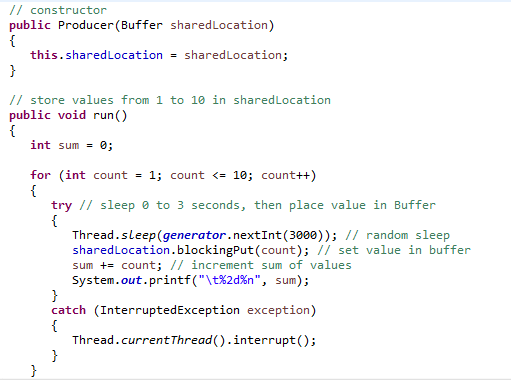


1. Also define a producer and a consumer for add new value and use current value.

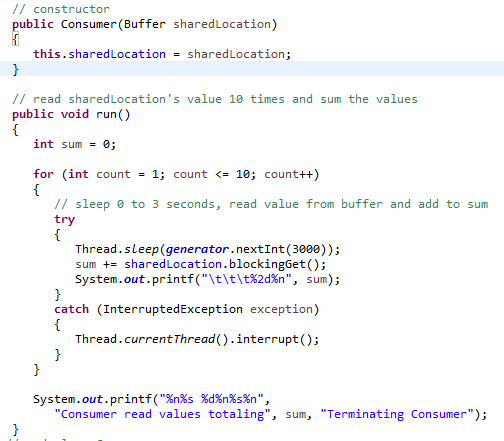
* Producer:

It produces numbers from 1 to 10 with a random time interval from 0 to 3 seconds.

After each production, set the producing value to the shared buffer, sum the value produced so far and print corresponding information.



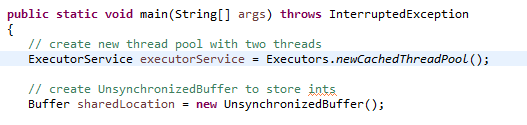
* Consumer: it gets current value from the shared buffer with a random time interval from 0 to 3 seconds, sum the value got so far, and prints corresponding information



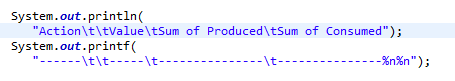
1. In the test class,

* It first creates a new thread pool to manage threads

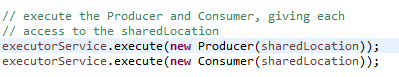
And also create a new shared buffer to play with.



* Create table header:



* Create and execute a producer thread and a consumer thread.



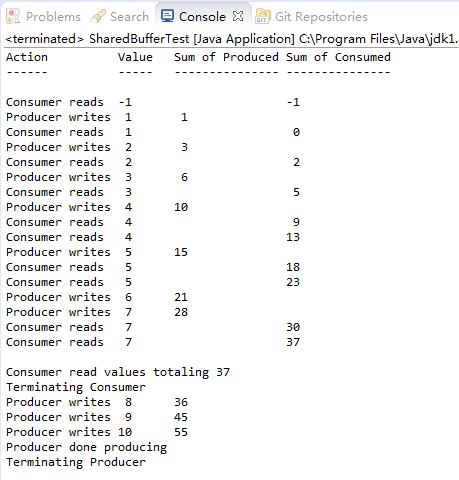
* Refuse new tasks and terminate the service if all tasks are completed or a times-out occurs (go beyond 1 minute); otherwise, block here to wait for the completion of all tasks.



1. Final running result:

We can see that the producing process and consuming process are not done alternatively, which means the consumer may consume the duplicate elements or miss some elements. Also we can see the total values of producing and consuming are not the same. The consumer even starts consuming before producing, although the producing process starts first.

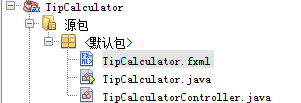
All of this kind of chaos is due to using unsynchronized way to access shared data. The access order of different threads will be determined by CPU schedule which is kind of unpredictable for the user.



1. Tip Calculator

This program uses JavaFX to implement a tip calculator

1. GUI part, use Scene Builder to create the GUI and store the GUI configuration in TipCalculator.fxml.



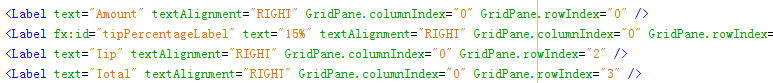
I won’t show the creating process here (since this program was done by the author); instead, I will explain the final setting result stored in the setting file.

* Set the GUI’s layout to “gridPane”, and set the alignment, and gap between each grid. Also assign a controller for this GUI.





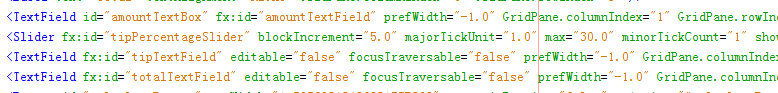
* Set labels and corresponding alignment information and position (the index of horizontal grid and vertical grid)



* Set controls for displaying user information, including three textFields and a slider.

Besides location information,

For textFields, set the “tip” and “total” field not editable and not traversable. Set the PrefWidth property to USE\_COMPUTED\_SIZE (-1) to indicate that the column’s width should be based on the widest child (the Amount Label).

For the slider, set its default value to 15 and its max value to 30; set tick unit to 1. Do not show the tick marks.





* Set button and its corresponding action

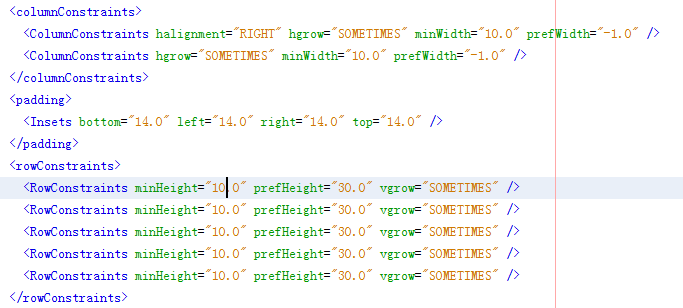
Set the Max Width property to MAX\_VALUE. This causes the Button’s width to grow to fill the column’s width



Set the action to be taken (method name in the controller) when clicking the button and its position.

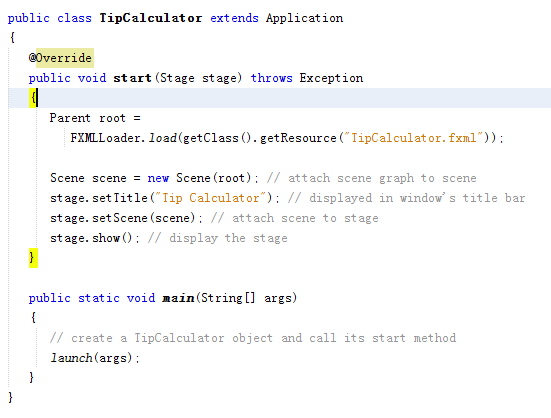


* Set padding around the panel, and also set the constraints of each row and column, mainly including alignment, prefer height/width and minimum height/width.



1. TipCalculator

* The class’s main method calls class Application’s static launch method to begin executing a JavaFX app. This method, in turn, causes the JavaFX runtime to create an object of the Application subclass and call its start method.
* This program overrides the start method and adds its own logic: Loads UI settings from TipCalculator.fxml, creates the GUI, attaches it to a Scene and places it on the Stage
* In the load method, it returns a Parent object reference to the scene graph’s root node (GridPane), and it creates an object of the controller class(defined in the FXML file) and calls its “initialize” method and registers all the event handlers for any events specified in the FXML.

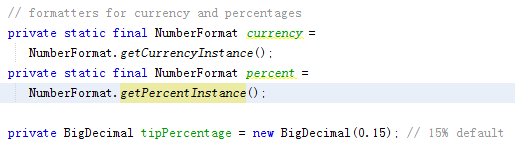


1. TipCalculatorController.java

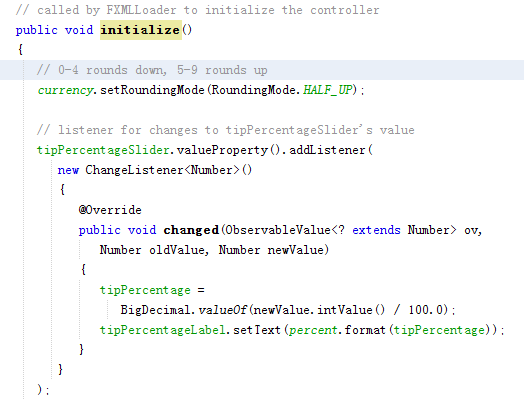
Initializing process called by the FXML Loader.

* Define two formats for currency and percentage.

Use BigDecimal object to store the percentage with a default value 0.15.



* Set rounding mode for currency and add a listener for value changing of slider. After each changing, get the new value in the integer format and change it to percentage by dividing 100, format it to percentage format and set the new value to the label.



1. Action to be taken when clicking the button. (registered during the initializing process)

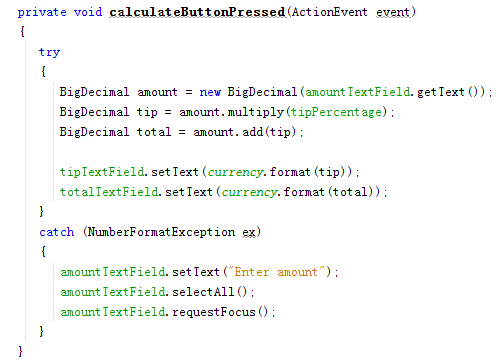
Calculate: Tip = amount \* tip percentage.

Total = amount + tip

After calculation, set corresponding text fields.

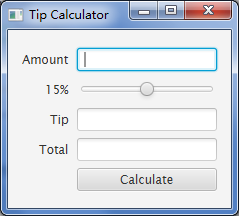
Note that the value displayed is formatted by currency (we have set its rounding mode)

Set selection and focus on the text field and set the value to “Enter amount” when entering a non-numeric value.

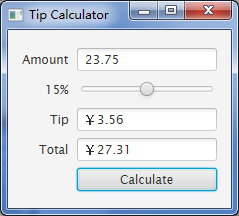


1. Final running result:

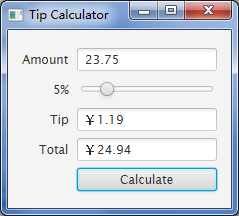
Start:



Input an amount and click “calculate”:



Drag the slider, and click the button:



Enter an invalid value:

