Programming Project Group 22

Members: Oisín Tong, Eoin Donnelly Maguire, Adam Lysaght, Stephen Rowe

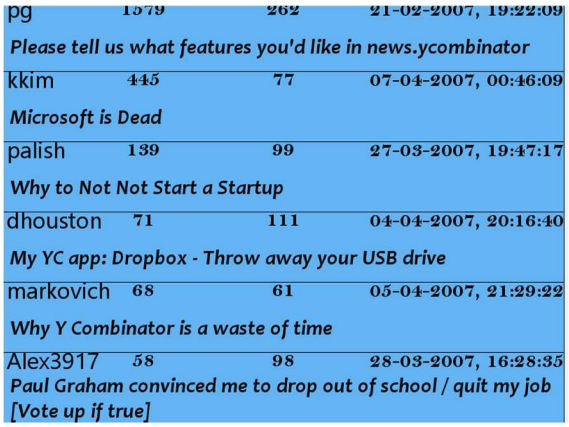
Approach:

The brief for the assignment was to create an app to explore data from "HackerNews.net". When we began this project, our team agreed to achieve two main goals with our program:

- Create an interface that could easily be used on a day-to-day basis by a Hacker News user with all the functionality expected for casual browsing.
- All the features of our program, including all aspects of the user interface, would be coded ourselves, without the use of any imported libraries.

We started out by creating a rough draft in an image editor of what each screen should look like, and how we should present the content from the JSON on screen.

We had several potential user interface drafts, for example, the design shown below –



However, after modifying and optimising our most favourite ones, we decided on the following widget design:

We based our designs on both the original Y Combinator News site and the Reddit interface. We followed the guideline goals for our labs every week as a basis for the overall picture, so that we wouldn't go on any tangents with small and time-consuming features that would eventually detract from the project as we got closer to the deadline. We would agree on what aspects of the program we would each work on over the weekend and discussed through our group-chat and in our weekly meetings on Wednesday.

Next, we went about breaking down what the core problems we needed to address were. We categorized our core problems under three different headings: loading, sorting and displaying.

Our Program:

We suggest using the file "hackerMedium.json" from blackboard for demonstrating our program.

Our program uses four arrays to hold JSON (aka StoryData) content:

- storyList
- queryStoryBackup
- queryCommentBackup
- currentQueryList

storyList contains all the data from the JSON, with each line stored as an instance of StoryData. This array is filled in loadData. currentQueryList contains only the data relevant to the current screen, and is the array that is used when the user is sorting data. queryStoryBackup is a clone of the content in currentQueryList *when we are on SCREEN_STORY only.* This also applies to queryCommentBackup, but with SCREEN_COMMENT. This is done so that, if a use enters a comment screen, or a subcomment screen, or a graph screen, then they can return to either comment screen or story screen, and then immediately sort the data correctly and immediately.

The arrays storyDisplay, commentDisplay, subCommentDisplay are used to store the StoryWidget and Comment Widgets created from the above StoryData arrays.

loadData()

This function made use of a buffered reader in order to read in each individual line of the JSON file. The information was then parsed into the storyList and currentQueryList and a "story" was created, using the StoryData class which will be explained later. The loadData function continued to read through the file until it's completion, then the app interface began.

StoryData class

The StoryData class sorted the information that was loaded and made on object from it, using get() methods to return the relevant information and set the variables within the setVariables() function. This information included: ID, time of posting, who the post was made by, and when present the url to the article. Each element created was then stored in the relevant ArrayList.

changeQuery()

This method was vital for sorting the data in the relevant query list, as well as searching for a string entered in the search bar. An event constant would be passed as a parameter, and the appropriate if statement would dictate how the data was sorted. The parameters EVENT_SEARCH, and the string "" were when the user clicked the home button, and this happened to achieve the same result we were looking for.

Banner Class

This method will draw the Big banner on screen, the "Hacker News" title, and the search bar, and accepts user text input for searching a query. The string textInput is set to "", and float scrollEffect is set to 0 to start (will explain this variable below). The height of the entire banner is always 3 times the height of the search bar. The method typing() is called in main when the user presses a key, accepting the pressed key as parameter. If the key is a printable character, it will be added to the

string textInput, which will then also be printed on screen in the search bar. If the key entered in Backspace, it will find the width of the last character of the string, and subtract this from scrollEffect. Then it removes the last character from the string. If the key entered is delete, then textInput is made blank and scrollEffect returns to 0.

If the width of the string is wider than the searchBar, then we increment the scrollEffect by the width of the key. What scrollEffect does is adds up the width of all the characters that will make the string textInput longer than the searchBar, and then pushes back the entire string being printed on screen. The activateBlinker() method is used to determine if the blinking text cursor should be drawn when the user clicks on the search bar, and if it should stay solid while the user is typing.

I recommend trying to type a very long series of characters into the search bar to fully see the effect of the scrollEffect. Characters entered into the search bar are also limited to 200.

makeStoryWidgets(), makeCommentWidgets()

These methods in main are used to draw the StoryWidget or Comment widgets for a given element in the relevant ArrayList. At the same time, it updates the totalHeightAllContent value to determine exactly what the y-position of a given story should be, before storing it in the storyDisplay arraylist. This totalHeightAllContent value is then also used for the scroll bar of the relevant screen. Each story and each comment's height is adjusted to suit the height of the text in the widget. You may spot a few comments here and there that are outside the bounds of the comment widgets, but this is due to how the user on the original site formatted their text (e.g. lots of whitespace).

When the user clicks a story, they are entered into a comment screen, which shows only the immediate descendants of that story (i.e. top-level comments). If the user then clicks on a comment, then are entered into the subcomment screen, in which they can see all the subcomments (and subcomments of subcomments), all indented appropriated. This is done through the recursive function **getCommentDescendants()**, which calls **getImmediateDescendants()** for a given comment, then enters into recursion to get each of those immediate descendants, and so in. In each "layer" of recursion, a comment is assigned an "indentingValue", which is calculated by counting how far into recursion we had to go to get this particular comment.

All the comments found through recursion are then concatenated together in one arraylist, forming the subcomment screen the user will see.

Screen Class

Each type of content, i.e. story, comment, subcomment and graph, are given their own screen. Each screen has a unique scroll bar, and comment and subcomment screens are given a **star button especially for post that are of type "poll".** While the draw() in main determines which screen should be drawn, the draw in the screen class is used to draw all the widgets, the relevant stories/comments, and the scrollbar. It also carries over relevant mouse functions, within mousePressed().

SortBy and Themes Classes

The stories displayed would then change in order to satisfy each of the queries. The drop-down was created within SortBy and Theme class, and the mousePressed() method in Screen class would satisfy the user's instructions. A theme class was created in order to allow the user to choose from a selection of colour schemes including: Blue/White, Red/Black, Orange/Black.

Scroll Bar Class

Easily the most difficult aspect of our program to manually create, we decided to use the totalHeightAllContent variable from the makeCommentWidgets and makeStoryWidgets methods to determine (a) the height of the slider (through the ratio of slider to bar), (b) how much the slider has been moved by the user (we find the percentage of the middle point of the scroll bar in relation to the entire scroll bar), and (c) how much the shift in the scroll bar should shift up and down each individual story widget, with relation to totalHeightAllContent. Once we had success in scrolling the content, we then decided to include mouse wheel functionality, and clicking and dragging the scroll bar, so that it became completely natural to use this scrollbar like any other.

If there is anything that you feel needs to be explained further, please consult the comments in the program file, which also include credits to who wrote the relevant parts of the code.

Testing our program:

When testing our program, we suggest trying the following features:

- Searching by user with **user:** , and searching by date with **date:DD-MM-YYYY.**
- Switching between many screens, then immediately sorting the content (comments can only be sorted by latest and oldest).
- Home button: returns the user to the latest stories page, same as when the user first opens the program.
- Try out the themes, and switch through different screens
- Search for the story "facebook is not really that special", then click the top-level comment by user "menloparbum", and scroll through the subcomments. This is a good example of the comment indentation, and you can compare it to the original comment page on HackerNews here: https://news.ycombinator.com/item?id=160408
- Hover your mouse over a URL on the main screen, the URL will be underlined by a line the length of the text width of the URL.
- Search "hero" and move through all the pages with next-page. You'll notice that the option to go to the next page will disappear on the last page, and the ratio of the slider has changed to suit the content on screen.
- Search for the story "how old are you", and enter the comments. You will see a star button on the top of the screen, where you can view the poll results.

Our final design came together after six weeks of collaborative brainstorming, coding, and iterating. Over the course of our time working together we learned to communicate clearly amongst one another and work together with a common focus. Our proficiency at problem solving and coding also increased greatly thanks to the work we put into our project.

In the end we reached a final product that we believe achieves everything we set out to do, and we are all proud of the work we have done.