Documentation

Rohin Arya, Y11 HL DP Computer Science, May 31.

The client has requested a piece of software to manage information about clients with a requirement of a specific set of features. To fulfil these requirements, the programming language java is most ideal for a variety of reasons.

Firstly, Java is very quick relative to other common programming languages. This is because the language has low-level control over the computer allowing to complete computational tasks more quickly. Secondly, Java is included natively on nearly all modern devices. In 2016, more than 15 billion devices ran Java[1], which makes it an excellent language to build potentially-corporate software that must run on a host of devices. Finally, Java is statically-typed which makes the developer experience superior in many ways such as IDE autocomplete, syntax highlighting, and catching errors before runtime. While subjective, I find these benefits very useful and an important factor when selecting a programming language.

When designing the software, I paid close attention to file structure and insuring extensibility when writing code. I created the following files:

- App.java
  - This includes the main entry point into the program, password verification, and calls various other methods within other files.
- Client.java
  - This class serves as the blueprint for all Client instances to allow for custom instance methods such as `save()`, `delete()`, and others.
- Loop.java
  - To better organize my code, I separated the loops into a different file so that it can be better managed. This file manages most of the input from the user and delegates methods from the Client and Search classes.
- Search.java
  - To allow for high-quality searches, I created a Search class with methods like `addFilter()` so that multiple filters can be layered.

Within each file, there are code comments, appropriate method headers, readable spacing/tabbing, and evidence of attentiveness to other Java conventions to allow for future improvements authored by anyone. More specifically, ArrayLists were used frequently to manage client data well. The reason for this is because ArrayLists have great helper methods such as `.contains()`, `remove()`, `forEach`, and many more . However, when gathering data from files (ex. `Search.java:62`), stacks were used (with ArrayList syntax) to demonstrate the benefit of reading the stack in reverse order. The reason for this is to ensure that the more recent files will show up first without the added time/complexity of manually reversing an ArrayList.

To conclude, I used various classes to abstract away some of the complexity with its functionality. For example, to save a client, instead of manually writing all the data to the file, I can simply call `x.save()`, where `x` is an instance of Client. Finally, fields, and some methods include the `private` modifier to encapsulate important data for security and privacy.

[1]: Oracle. (n.d.). Moved by Java Timeline | Oracle. Retrieved June 3, 2022, from https://www.oracle.com/java/moved-by-java/timeline/.
hi