Name: Daryan Chan

ID: 113973192

Section: NAA

Course: JAC444

Workshop 6

Task 1

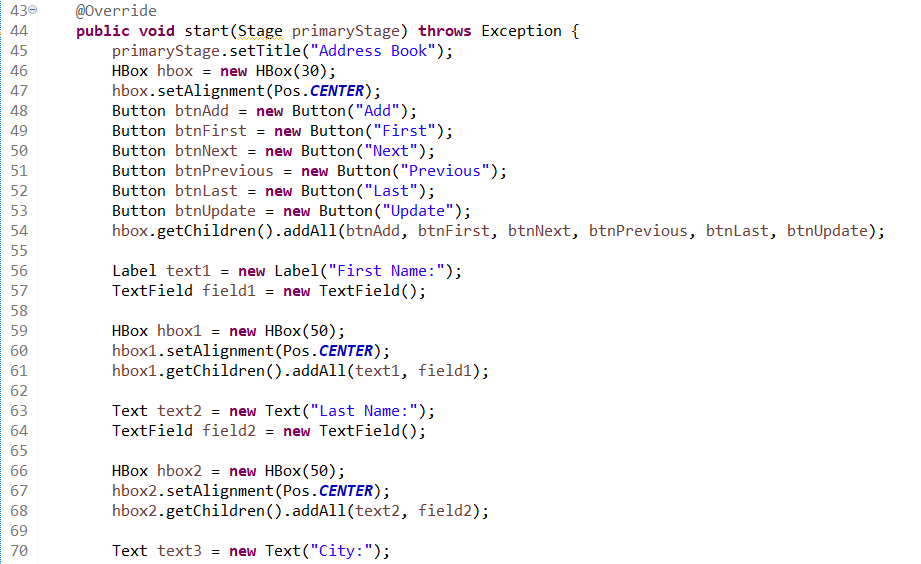


Figure 1 AddressBook class with JavaFX functionality

The first thing I did was to create the GUI for allowing user to input data. This was done by titling the application using setTitle(). Next thing I did was creating all the buttons. Since they will be arranged in a horizontal row, I had to use the HBox. Then I created all the buttons, Add, First, Next, Previous, Last, and Update. Since they were all part of the same group, I had to use getChildren().addAll() and attach all those button to the hbox.

Next came the textfields. As there were texts beside the textfields, I used 2 different methods. The first one was using Label to create the text I would place next to the textbox. TextField are used to create the textbox. Since the texts and textfields will become part of the same group, I used another HBox to group them together. The concept is the same as the previous paragraph but only thing that was different was that I used setAlignment as well. I utilized this method to position those elements in the middle.

The same idea were used for the rest of the textbox and texts. However for the rest of those, instead of using Label, I just used Text class.



Figure 2 Task 1 continue

One notable difference is that I used a ChoiceBox to create a dropdown menu. To populate the options, I used getItems().add().

Once all the texts, choicebox and textfields were created, I had to group them all together. To do this, I used a VBox to arrange them in a vertical pattern and spaced them out using setSpacing(30).

Once those things were done, I had to put everything together, hence why I used BorderPane. With this functionality, I placed the vbox on the top and hbox on the bottom, and added a beige colour background.

Next, to create the application when the code runs, the last 3 lines of codes were included to facilitate that. The end result was this.

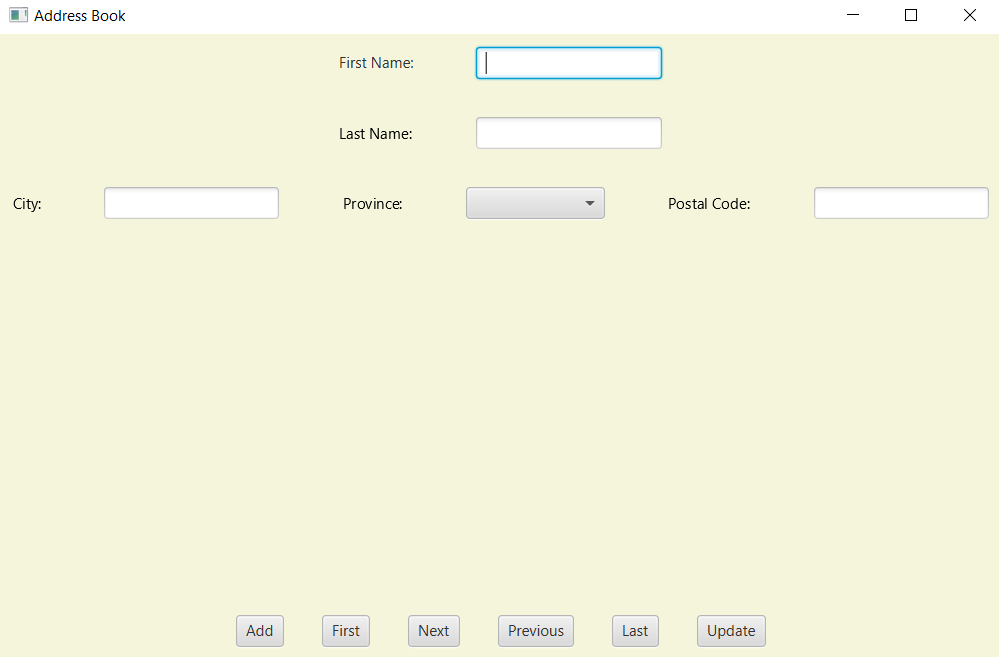


Figure 3 task 1 continue

This was the finished product.

Task 2

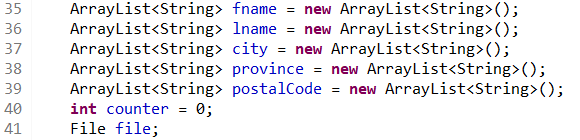


Figure 4 Task 2 declarations

Since the user will populate the file based on what they entered, I used ArrayList since this array function will expand the size automatically to store new info. Counter and file will be explained later.

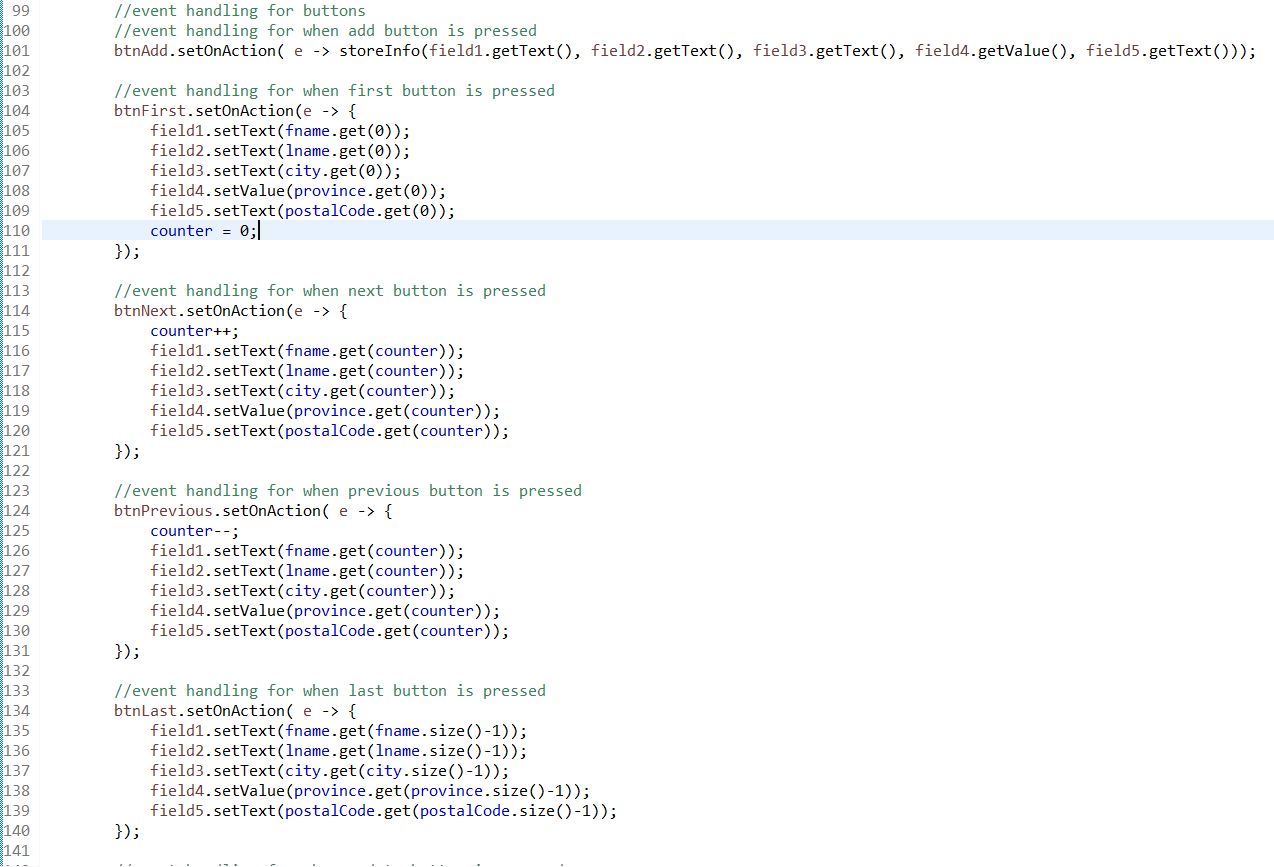


Figure 5 adding eventhandling

For the second task, I had to make sure each button did as it implied. For example, the add button should ensure that the data entered by the user should be saved. Therefore, I used setOnAction for the btnAdd, which will call the storeInfo method and all the parameters will come from the textfields and choicebox.

The second is button first, which needs to show the first info. To do this, I used setText for the textfields and setValue for the choicebox and passed in the first element of each array for each field. The counter resets back to 0.

For the next button, the counter will add 1 to it, and then use the setText and setValue again, and retrieve the element from the array using the counter as the index.

The previous button uses the same concept as the next button. Only thing that is different is that the counter will decrease by 1 every time it is pressed.

The last button has the same concept as the First button, but to retrieve the last element of each array, I had to use size()-1.

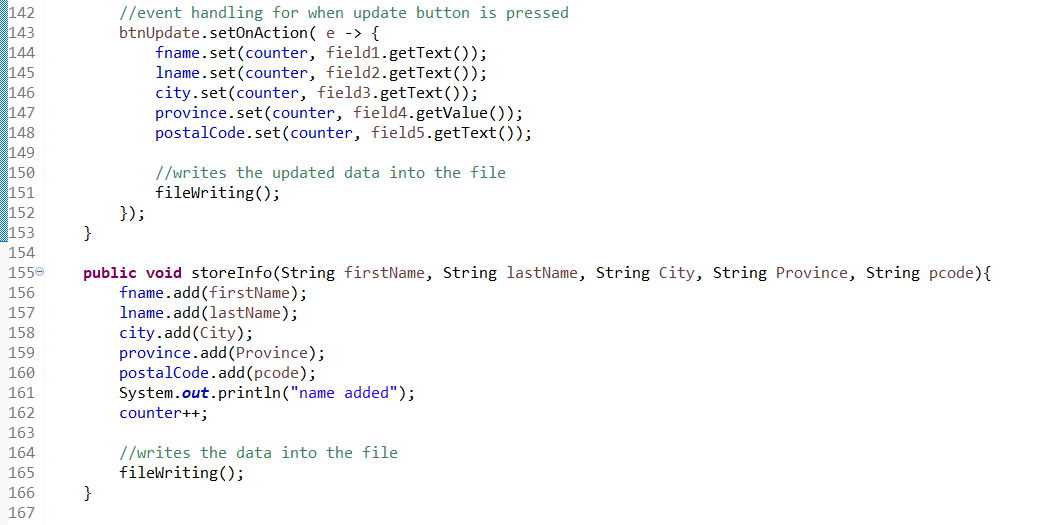


Figure 6 task 2 continuation

The final button is the update button and is supposed to update the file with revision from the user. First, each array will retrieve the data from the new user input and replace the original data. This is done by using the set functionality. It takes the counter as the first parameter, which is supposed to help find the index of the array that needs to be updated and use getText and getValue from each textfield/ choicebox. After, it calls the filesWriting method.

The storeInfo method is used by both the add button. When called, it retrieves the String datas and adds them to the arrays. Counter will be increased by 1 as well. Next, it will call the fileWriting method.

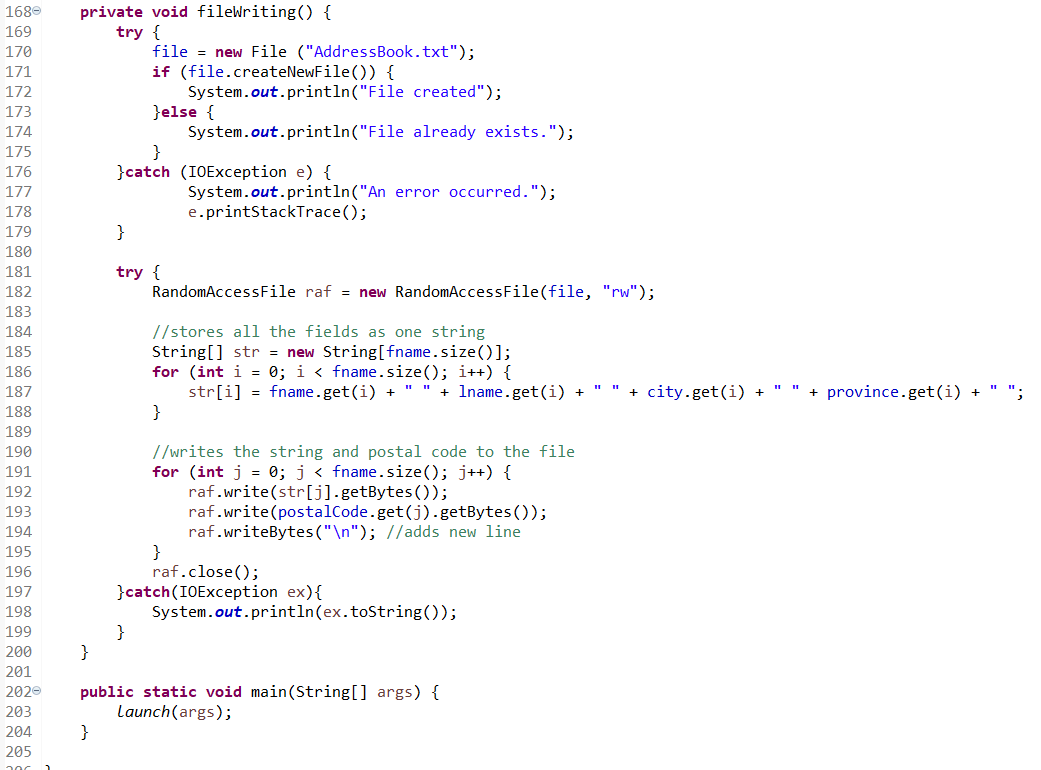


Figure 7 task 2 continuation

The fileWriting method will use the try catch method. It will retrieve AddressBook text file and will go through an if else statement. If file doesn’t exist, then it will create it. If it does, it will alert the user that it exists. Should an error occur, it will catch it and alert the user. Once that is done, it will do another try method and this time, it is for writing and reading the file using RandomAccessFile. A String array will be created and go through a for loop. The purpose of this is to copy each array’s element into a single string. Next, another for loop is initiated to write the String data into the file. This was done using getBytes. When everything is written into the file, it will close the file. Again, the catch is there to receive any errors.

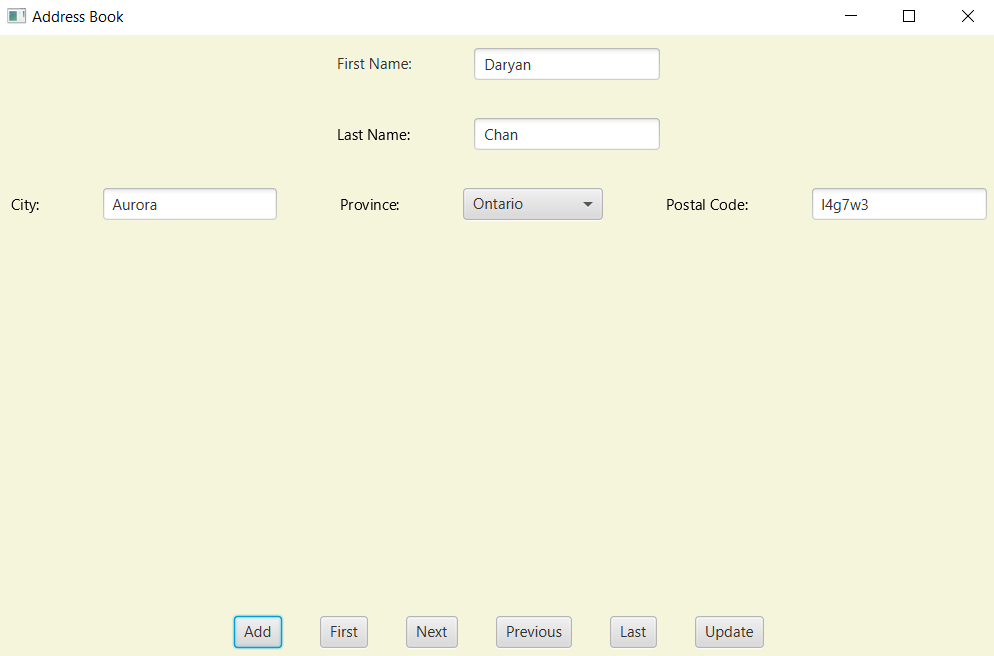


Figure 8 task 2 entering data

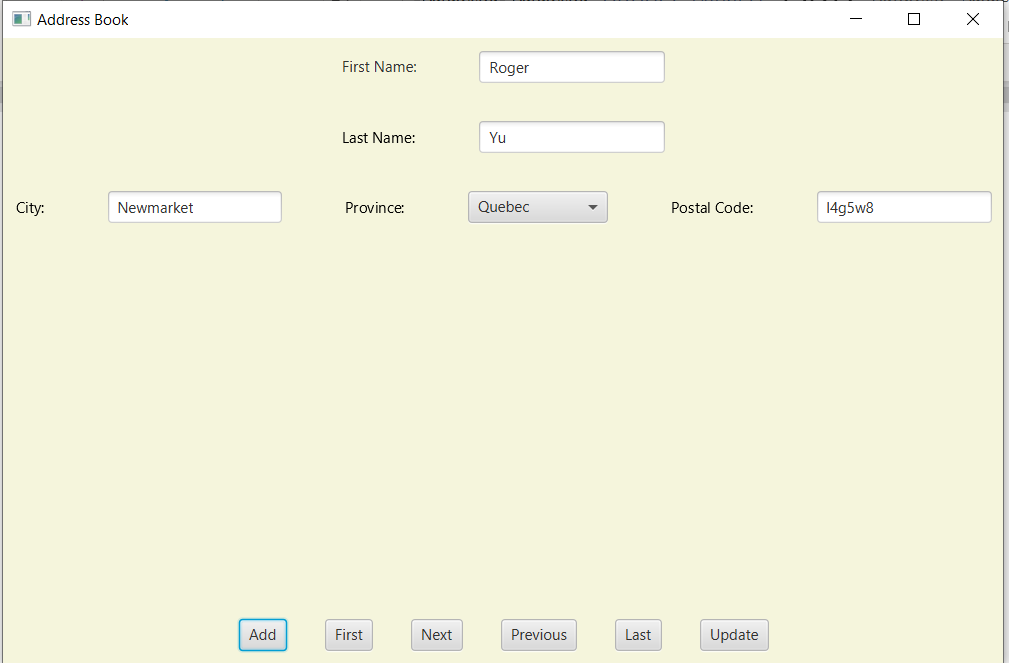


Figure 9 entering data

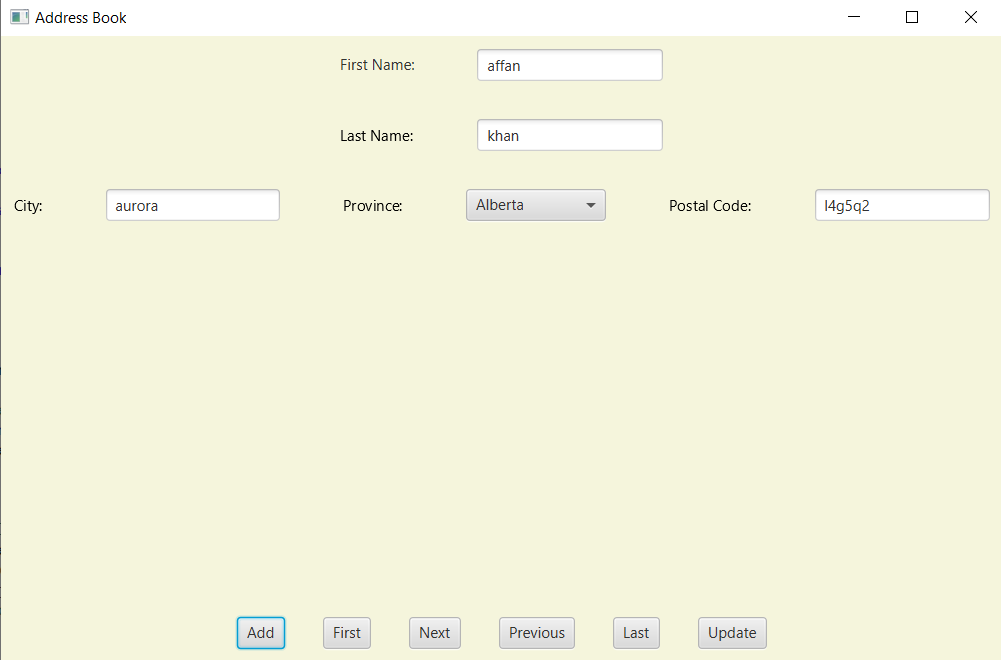


Figure 10 entering data

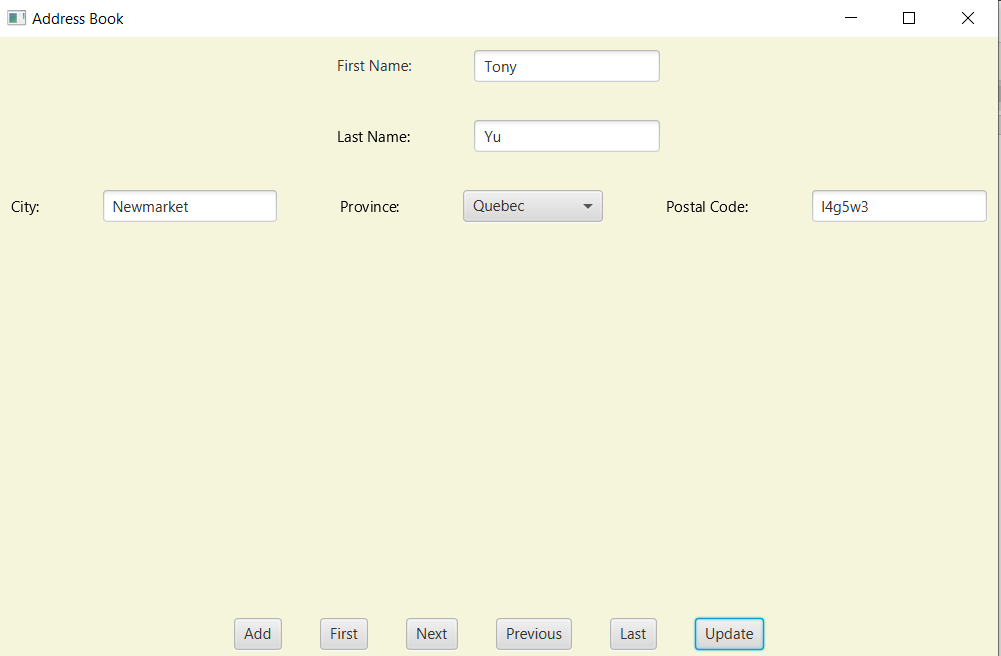


Figure 11 clicking previous and updating

Going back on element and updating the file. Changing name Roger to Tony and updating postal code.

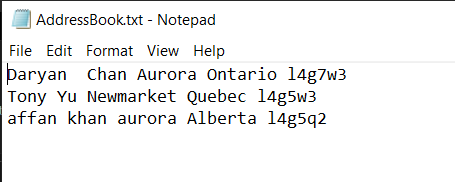


Figure 12 textfile creation

Data inside the textfile.

Task 2 Research

1. The advantage of using RandomAccessFile is that data is stored in byte format and makes fields that contain data types such as integer and doubles much smaller. It is also easy to navigate if you want to access a particular location (seek) within the file. It allows files to be opened and read and written to it sequentially, therefore it is much easier and efficient.
2. The disadvantage is that RandomAccessFile is somewhat disconnected from the input/ output stream in java.io. It’s disadvantage because you cant use the same filters to RandomAccessFile as you can to streams.
3. Usage of RandomAccessFile is understanding that it treats the file as an array of bytes. You can also write your data in any position of the array. This is done by using a pointer that holds the current position. It does that by using
   1. getFilePointer() to get the current position of the pointer
   2. seek(int) to set the position of the pointer
   3. read(byte[] b) to read up to b.length bytes of data from the file into an array of bytes
   4. write(byte[] b) to write b.length bytes from the specified byte array into the file, starting from the pointer.
4. Syntax, I assume are the different type of modes and mandatory methods to use RandomAccessFile. To start, we will need [**RandomAccessFile**](https://docs.oracle.com/javase/7/docs/api/java/io/RandomAccessFile.html#RandomAccessFile(java.io.File,%20java.lang.String))([**File**](https://docs.oracle.com/javase/7/docs/api/java/io/File.html) file, [**String**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) mode). File is the file name and mode can be the following:
   1. r – open for reading only
   2. rw – open for reading and writing and if file doesn’t exist, then attempt to create it will be made
   3. rws – open for reading and writing and requires that every update to the file’s content/ metadata by written synchronously to underlying storage device.
   4. rwd – open for reading and writing and requires that every update to the file’s content by written synchronously to underlying storage device.

At the end, it must close the file with close().

1. Constructor can be [**RandomAccessFile**](https://docs.oracle.com/javase/7/docs/api/java/io/RandomAccessFile.html#RandomAccessFile(java.io.File,%20java.lang.String))([**File**](https://docs.oracle.com/javase/7/docs/api/java/io/File.html) file, [**String**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) mode) or [**RandomAccessFile**](https://docs.oracle.com/javase/7/docs/api/java/io/RandomAccessFile.html#RandomAccessFile(java.lang.String,%20java.lang.String))([**String**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) name, [**String**](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html) mode)

The first constructor creates a random access file stream to read from and optionally to write to the file specified in the file argument. The second one is the same but instead, it can optionally write to a file with the specified name.

1. There are many methods. These include:

[**close**](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html#close())()

Closes this random access file stream and releases any system resources associated with the stream.

 o  **[getFD](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "getFD())**()

Returns the opaque file descriptor object associated with this stream.

 o  **[getFilePointer](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "getFilePointer())**()

Returns the current offset in this file.

 o  [**length**](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html#length())()

Returns the length of this file.

 o  [**read**](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html#read())()

Reads a byte of data from this file.

 o  [**read**](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html#read(byte[]))(byte[])

Reads up to b.length bytes of data from this file into an array of bytes.

 o  [**read**](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html#read(byte[],%20int,%20int))(byte[], int, int)

Reads up to len bytes of data from this file into an array of bytes.

 o  **[readBoolean](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "readBoolean())**()

Reads a boolean from this file.

 o  **[readByte](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "readByte())**()

Reads a signed 8-bit value from this file.

 o  **[readChar](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "readChar())**()

Reads a Unicode character from this file.

 o  **[readDouble](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "readDouble())**()

Reads a double from this file.

 o  **[readFloat](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "readFloat())**()

Reads a float from this file.

 o  **[readFully](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "readFully(byte[]))**(byte[])

Reads b.length bytes from this file into the byte array.

 o  **[readFully](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "readFully(byte[],%20int,%20int))**(byte[], int, int)

Reads exactly len bytes from this file into the byte array.

 o  **[readInt](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "readInt())**()

Reads a signed 32-bit integer from this file.

 o  **[readLine](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "readLine())**()

Reads the next line of text from this file.

 o  **[readLong](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "readLong())**()

Reads a signed 64-bit integer from this file.

 o  **[readShort](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "readShort())**()

Reads a signed 16-bit number from this file.

 o  **[readUnsignedByte](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "readUnsignedByte())**()

Reads an unsigned 8-bit number from this file.

 o  **[readUnsignedShort](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "readUnsignedShort())**()

Reads an unsigned 16-bit number from this file.

 o  **[readUTF](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "readUTF())**()

Reads in a string from this file.

 o  [**seek**](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html#seek(long))(long)

Sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.

 o  **[skipBytes](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "skipBytes(int))**(int)

Skips exactly n bytes of input.

 o  [**write**](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html#write(byte[]))(byte[])

Writes b.length bytes from the specified byte array starting at offset off to this file.

 o  [**write**](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html#write(byte[],%20int,%20int))(byte[], int, int)

Writes len bytes from the specified byte array starting at offset off to this file.

 o  [**write**](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html#write(int))(int)

Writes the specified byte to this file.

 o  **[writeBoolean](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "writeBoolean(boolean))**(boolean)

Writes a boolean to the file as a 1-byte value.

 o  **[writeByte](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "writeByte(int))**(int)

Writes a byte to the file as a 1-byte value.

 o  **[writeBytes](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "writeBytes(java.lang.String))**(String)

Writes the string to the file as a sequence of bytes.

 o  **[writeChar](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "writeChar(int))**(int)

Writes a char to the file as a 2-byte value, high byte first.

 o  **[writeChars](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "writeChars(java.lang.String))**(String)

Writes a string to the file as a sequence of characters.

 o  **[writeDouble](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "writeDouble(double))**(double)

Converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the file as an 8-byte quantity, high byte first.

 o  **[writeFloat](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "writeFloat(float))**(float)

Converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the file as a 4-byte quantity, high byte first.

 o  **[writeInt](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "writeInt(int))**(int)

Writes an int to the file as four bytes, high byte first.

 o  **[writeLong](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "writeLong(long))**(long)

Writes a long to the file as eight bytes, high byte first.

 o  **[writeShort](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "writeShort(int))**(int)

Writes a short to the file as two bytes, high byte first.

 o  **[writeUTF](http://www.cs.yorku.ca/tech/other/java/docs/api/java.io.RandomAccessFile.html" \l "writeUTF(java.lang.String))**(String)

Writes a string to the file using UTF-8 encoding in a machine-independent manner.

References

Kiourtzoglou, B. (2013, February 16). Java RandomAccessFile Example. Retrieved July 02, 2020, from https://examples.javacodegeeks.com/core-java/io/randomaccessfile/java-randomaccessfile-example/

Working with Random Access Files. (n.d.). Retrieved July 02, 2020, from https://www.cs.princeton.edu/courses/archive/spr96/cs333/java/tutorial/java/io/rafs.html