

CSE 174 - Lab 13

Part 1 (8 points): Learning a new class.

1. Study the Customer class:

- For this lab you are to use the Customer class and in order to finish this lab successfully you need to study this class very carefully, and you will be tested for that.
- Download the [API Documentation for Customer Class](#).
- Start studying the class description first.
- Scroll down and study the constructors.
- After learning about constructors, scroll down again and study the methods. You can click on each method name to see more details.

★ You should not go any further if there is something in this class that you don't understand. You can go back and study the material, talk to your friends, or you can ask your instructor in case you need help with understanding this class.

2. Using the Customer class:

- In jGRASP create a java file called **Lab13**.
- Download the [Customer.java](#) file (DO NOT CHANGE ANYTHING INSIDE THIS FILE) and put it in the same folder as your Lab13.java file.
- Inside the main method, create a few Customer objects, call different methods and print the results. For instance: create two Customer objects, call different methods on each object and print the results. Compare two objects, see if the result is acceptable for you. Print the objects to see how the output looks like. Now, try to print the object in a different format.

★ Again, if you have any questions please ask your instructor. You should not go to the next part of the lab without understanding all details of this class.

3. Take the quiz 13.1:

- Now it's time to evaluate your understanding of this class. Please go ahead and take the quiz 13.1

Part 2 (12 points): Array of Objects & Search algorithms

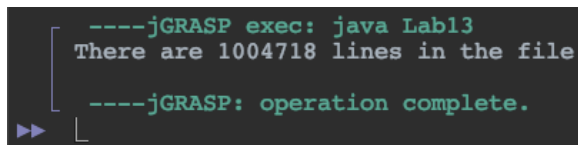
In this Part of the lab you are to create an array of Customer objects, and initialize each object with the data from a file. Then you are to write 2 methods, one for linear search and one for binary search and look for different keys to see which search method can find the keys faster.

1. Reading data from a file:

- Download the file [customer_list.txt](#) and put it in the same folder as your Lab13.java file.
- Try **not** to open the file, since this text file contains more than a million lines and could cause your jGRASP to crash. As you can see in the following image, in each line there is a nickname and id number separated by space which is necessary for creating one Customer object:

```
gwstikg 100005949
nzjxuz 100009509
uvhhzehay 100010722
aly 100015911
sdqctkgc 100022212
dknp 100026342
ysxmpomd 100037474
zuhvkcbmmk 100040373
aaesgjomw 100073737
bqabh 100076373
lyladzg 100088624
qif 100093541
nlba 100125412
urps 100128450
```

- Now let's write some code. Inside the main method (if you like you can write a separate method for that, it's up to you), write some code to open the file and count how many lines this text file has and print the result similar to the following line:

A screenshot of a terminal window with a dark background. It shows the output of a Java program. The first line is '----jGRASP exec: java Lab13' in green. The second line is 'There are 1004718 lines in the file' in white. The third line is '----jGRASP: operation complete.' in green. There are green arrow icons on the left side of the first and third lines.

```
----jGRASP exec: java Lab13
There are 1004718 lines in the file
----jGRASP: operation complete.
```

(you should **not** hardcode that number, your code should calculate the number)

2. Creating an array of Customer objects:

- Now that you know how many lines the text file has, that means you know the length of your array. So, go ahead and define an array of Customers using that number as the length of your array.
- You can not define an array without knowing the length. In addition, after defining one, you can not change the length of the array directly. Therefore, having a fixed size list/array is one of the weaknesses of using arrays.
- After defining the array, java will initialize each element of the array with a **null** value, which means each element inside the array (each element is like a Customer variable) is pointing to nothing and it is ready to point to an actual Customer object.
- The only difference between arrays of primitive types and objects is that after defining an array of objects you should assign an object to each element. So, it's time to read from the file line by line again, but this time for creating a Customer object from each line and assigning it to each element of the array.
- Reopen your text file again. The file needs to be reopened because in the previous part we already went through the file once to count how many lines the file has.

Hint: for reopening the file you can create a new Scanner object and re-assign it into your previous Scanner variable.

- Using a loop, start reading from the file as long as there is something inside the file. **Use hasNext()** method over hasNextLine(). Keep that in mind during each iteration you need to read **one String** and **one long** value.
- During each iteration after reading two elements from a line (one String, one long), use them to create a Customer object (you practiced that in the Part1 of Lab13), and use that Customer object to initialize the current element of the array with. So, the following is the summary of what you need to do after defining an array of Customers:

```
loop (as long as there is something inside the file) {  
    Read one string and one long value  
    Create a new Customer object  
    arr[i] = The Customer object created in the previous line  
}
```

- Now let's test your array to see if you were able to read from the file, create Customer objects, and save them inside the array correctly. Print the elements at the index of 0, 10, 1000, 1000000, and the last element of the array. Before printing each object, print the index as well. So, your result so far should look like the following:

```

----jGRASP exec: java Lab13
There are 1004718 lines in the file
[0]: [gwstikg, 100005949]
[10]: [lyladzg, 100088624]
[1000]: [zqtao, 109878166]
[1000000]: [vdni, 9953749496]
[last index]: [emdapyo, 9999993624]

----jGRASP: operation complete.

```

3. Writing a linear search method:

- So far, you were able to read from a file and create a list/array of Customer objects.
- Now the question is how will you find a specific Customer object between millions of elements inside the array?
- As a solution, let's write a method called **linearSearch** that accepts an array of Customer objects, and one Customer object as the key that needs to be found inside the array. And if you recall, the linear search method should return an index.
- The only difference between this method and the one in the videos is that here you are dealing with Customer objects. So, every time you need to check each element of the array with the given key to see whether they are equal or not (you practiced comparing two Customer objects in the Part1 of this lab).
- When you are done with the method, let's test it to see if it works. Inside the main method create a Customer object called key using the nickname: "gwstikg" and id: 100005949
- Call the linearSearch method and give it your array and the key you just created, and print the result. Add some messages to have your output as following:

```

----jGRASP exec: java Lab13
There are 1004718 lines in the file
[0]: [gwstikg, 100005949]
[10]: [lyladzg, 100088624]
[1000]: [zqtao, 109878166]
[1000000]: [vdni, 9953749496]
[last index]: [emdapyo, 9999993624]
The object [gwstikg, 100005949] located at the index of: 0

----jGRASP: operation complete.

```

- Now, **inside the linearSearch method** add a counter to count every time it compares the Customer objects. Then, before returning the index, print the number of times that comparison has happened. The goal is to see how many times elements are compared inside this method. So, you need to count and print that inside this method. So, change your code to generate an output as below:

```

----jGRASP exec: java Lab13
There are 1004718 lines in the file
[0]: [gwstikg, 100005949]
[10]: [lyladzg, 100088624]
[1000]: [zqtao, 109878166]
[1000000]: [vdni, 9953749496]
[last index]: [emdapyo, 9999993624]
Linear Search: The key is found after 1 comparison
The object [gwstikg, 100005949] located at the index of: 0

----jGRASP: operation complete.

```

4. Writing a binary search method:

- Repeat the whole process from the last section this time to write the **binarySearch** method. Write a binarySearch method that gets an array of Customers and a key, and returns an index. Count how many comparisons happen inside this method and print it inside this method right before returning any values.
- The question that you need to ask here is whether the array is sorted since the binary search algorithm only works with sorted lists. The good news here is that the array is sorted based on id numbers, therefore inside the binarySearch method where you need to compare the key with the value of the middle element to see if it's greater or smaller, you only need to compare the idNumbers.
- Comment out some of the previous prints and make sure your output matches the following output:

```

----jGRASP exec: java Lab13
Linear Search: The key is found after 1 comparison
The object [gwstikg, 100005949] located at the index of: 0

Binary Search: the key is found after 19 comparison
The object [gwstikg, 100005949] located at the index of: 0

----jGRASP: operation complete.

```

- Repeat the process for finding the following Customers (the numbers are long type):
 - ["mqzhfygjuk", 6001073675]
 - ["gnv", 7412760286]
 - ["CSE174", 1111111111]
- After printing the results your output should look like the following:

```

----jGRASP exec: java Lab13
Linear Search: The key is found after 1 comparison
The object [gwstikg, 100005949] located at the index of: 0

Binary Search: the key is found after 19 comparison
The object [gwstikg, 100005949] located at the index of: 0

Linear Search: The key is found after 598734 comparison
The object [mqzhfygjuk, 6001073675] located at the index of: 598733

Binary Search: the key is found after 17 comparison
The object [mqzhfygjuk, 6001073675] located at the index of: 598733

Linear Search: The key is found after 160817 comparison
The object [gnv, 7412760286] located at the index of: 160816

Binary Search: the key is found after 17 comparison
The object [gnv, 7412760286] located at the index of: 742000

Linear Search: The key is found after 1004718 comparison
The object [CSE174, 111111111] located at the index of: -1

Binary Search: the key is found after 20 comparison
The object [CSE174, 111111111] located at the index of: -1

----jGRASP: operation complete.

```

5. Analyzing the results:

- Looking at the first test, you can see the linear search only did one comparison compared to binary search that did 19 comparisons, meaning if you are looking for something at the very beginning of the array, using the linear search could be a better option over the binary search.
- For the second test, you can see the fast result of binary search by doing only 17 comparisons compared to the linear search which did 598734 comparisons.
- In the third test, you can see each method returns a different index, and that is because there are duplicate keys inside the file. Since each algorithm searches through the array differently, each algorithm finds the key at a different index.
- The last test shows the worst case scenario where the key is not in the list. The linear search has to check all n or 1004718 elements to realize that the key is not in the list, while the binary search did only $\log_2(n)$ or 20 comparisons to get the same result.

6. Submit your Code on canvas:

- If your code generated the last results as shown above, you are ready to submit your work on canvas.
- There are two tests: one that tests your linear search, and one for testing your binary search.

Rubric

<u>Criteria</u>	<u>Full Credit</u>	<u>Partial Credit</u>	<u>No Credit</u>
Successful submission via Code	A fully successful submission to CODE that passes all of the required tests will earn full credit. 8 Points	Complete each test. 4 Points	if your submission is not accepted by code, you will receive no credit. 0 Points
Writing comments for the class and linear and binary search methods	Writing comments for the class and the linear and binary search methods. explaining the purpose, parameters, and the return type will earn full credit. 1.5 Point	Complete comments for each part. 0.5 Point	Not writing comments or incomplete comments receive no credit. 0 Points
Correct Style	If your submission has 0 style errors you will earn full credit. 2.5 Point		If there are any style errors present, you will receive no credit. 0 Points