University of New Brunswick
CS1083 – Open Access Version

Module 3 Assignment

The purpose of this assignment is to give you practice:

- applying the binary search algorithm.

For all assignments:

- Follow the instructions in the document "Java Coding Guidelines.pdf" available within the "Start Here" module, under the "Assignments" topic.
- For each .java file you create, include a javadoc comment (one that begins with /**) at the beginning of the file that describes that Java class. This comment block should include a line that begins with @author followed by your name and student number on the same line. (Note: inclusion of this comment is part of the instructions given in "Java Coding Guidelines.pdf")
- Include comments throughout your programs to explain any non-obvious portions of your code.
- It is recommended that you create a separate folder on your computer for each assignment. You might even wish to create separate sub-folders within an assignment folder if the assignment has multiple parts. Keeping your work organized makes it easier to find things later when you want to review what you have done.
- Few things in life are more frustrating than losing your work while working on an assignment. Get in the habit of saving frequently when working on an assignment. Also, regularly make backup copies of any files you create as part of your work for this course.
- Each module has a link to the discussion forum where you can ask questions and seek help in completing your assignments.
- Submitting your assignment involves creating a pdf file and uploading that single file to the assignment drop box on D2L. In general, that file will tend to include all Java code you wrote for the assignment, plus any output from running your programs that the assignment instructions specify you should capture. Specific submission instructions are included at the end of each assignment.
- To create the pdf file for each assignment, begin by opening a new document using the word processing program of your choice. Save the document using the name “Your Name CS1083 Module x Assignment Submission.docx” Replace x with the correct module number. “docx” may be different depending on which word processing software you choose to use.

- At the beginning of your submission document enter your name, your student number, CS1083, the assignment name (this one is “Module 2 Assignment”), and the date. It doesn’t matter if the date is when you started working on the assignment or when you submit it – either will do.
- You can add content to your submission document as you work on the various questions in the assignment. Clearly label each part of the assignment (“Part A” etc.) in your document. When your document is complete, save / export as a pdf file. Always make sure your pdf file opens properly before uploading it.
- To include Java code in your submission document, copy all the text in your .java file and then paste that text into your submission document. Use a monospaced font (e.g.: Consolas, Courier) for your code to maintain proper indentation.
- To include output from running your program in your submission document, you have two choices. You can either (a) copy and paste the text from your command prompt window, or (b) capture a screen shot of the output and include that as a picture / image in your submission document. Either approach is fine.
- To copy text from your command prompt window, try selecting the desired text and then pressing either command-c (Mac) or control-c (Windows or Linux). If you have issues, you can always use Google to see how to do this on your specific type of computer.
- To capture a screen shot of a selected portion of your command prompt window, try command-shift-4 (Mac), WindowsKey-shift-s (Windows), or shift-PrtScrn (Linux).
- Once you have pasted your output text into your submission document, make sure it is in a monospaced font (e.g.: Consolas, Courier) so you retain the alignment of output.

Searching Student Records

The file Student.java is provided for download with this assignment. This class represents a student record that includes a student id, surname, given name(s), and grade point average (gpa).

The file BinarySearch.java is also provided for download earlier in this module. This class provides a binarySearch() method used to locate a given integer value in an array of type int.

Modify this BinarySearch class into a new **StudentBinarySearch** class, containing a single static method called studentBinarySearch(). This method accepts three parameters:

1. A sorted array of Student objects, in ascending order according to their student id values. (This means the smallest student id value will be within the object in position [0] of the array.) This array cannot be assumed to be full. It might be empty, partially filled, or completely filled with Student objects;
2. A counter indicating how many Student objects are in the array; and
3. A student id value.

As an example, if the length of a given array is 100 and it contains 30 Student objects, then those objects would be in positions 0 to 29 of the array. Positions 30 to 99 would be unused.

The studentBinarySearch() method returns the index where a Student object with the given student id value is located. You can assume no two objects contain the same student id.

If no object in the array contains the given student id, then studentBinarySearch() returns -1 (which is represented by the NOTFOUND class variable).

Your method should make sure the given counter contains a valid value – between 0 and the length of the array – otherwise conduct no search and return NOTFOUND.

Then create a **StudentSearchTest** class that contains two static methods.

The **generateStudentArray()** method returns a randomly generated array of student objects. The following is a display created from an example array containing 10 Student objects:

```
[0] Student[studentId=3170520, surname=Norrad, givenNames=Claudia, gpa=3.4]
[1] Student[studentId=3211274, surname=Norrad, givenNames=Lynda, gpa=3.6]
[2] Student[studentId=3242897, surname=Gilroy, givenNames=Krista, gpa=2.0]
[3] Student[studentId=3243897, surname=Gilroy, givenNames=Matt, gpa=2.4]
[4] Student[studentId=3296985, surname=Elkins, givenNames=Robert, gpa=3.3]
[5] Student[studentId=3307085, surname=Swanson, givenNames=Krista, gpa=3.6]
[6] Student[studentId=3321249, surname=Norrad, givenNames=Tom, gpa=3.8]
[7] Student[studentId=3328191, surname=Kent, givenNames=Reynold, gpa=2.8]
[8] Student[studentId=3347208, surname=Gilroy, givenNames=Reynold, gpa=3.6]
[9] Student[studentId=3350844, surname=Curtis, givenNames=Tom, gpa=2.0]
```

The generateStudentArray() method accepts three parameters:

1. An integer **baseId** value used as a starting point for generating pseudo-random student id values. For the above example, I used a baseId value of 3108743. There was no particular reason why I chose this number – I just thought it would result in realistic student id numbers.
2. The size of the desired array; and
3. The number of array elements to fill in other words, the number of Student objects to be created and inserted into the array.

For each Student object created by this method, generate the required values as follows:

- student id – Pick some integer increment (I used 80000). Generate a random number between 1 and this increment, and add this number to the previous student id (starting with baseId). This guarantees the student id values will be in ascending order in the array.
- surname – Create an array of different surnames (I simply picked 10 names out of the phone book). Randomly generate an index for one of those surnames. As you can see from the example above, it's okay if different students end up with the same name.
- givenNames – Create an array of different given names (I simply picked 10 names out of the phone book). Randomly generate an index for one of those given names.
- gpa – Randomly generate a double value between 2.0 and 4.0. If you wish to limit the gpa to one fractional digit as shown in the examples above, consider generating an int value between 20 and 40 and then divide by 10.0.

Also write a **main()** method that does the following. Call generateStudentArray() to create an array of length 20 that contains 10 Student objects. Use toString() to display the Student objects along with their index values – [0] to [9].

Use the following six student Id values to test your studentBinarySearch method:

1. The student id from position [0] of the generated array *(note that test values 1, 2, and 3 are guaranteed to be found in the array)*
2. The student id from position [9] of the generated array
3. The student id from a randomly selected position in the generated array
4. The first test value minus 1 *(note that test values 4, 5, and 6 should not be found in the array)*
5. The second test value plus 1
6. The third test value plus 1 *(technically speaking, this value could possibly be found in the array, but it is very unlikely to happen)*

Use your studentBinarySearch method to search in the generated array for each of those test values.

The output from executing StudentSearchTest should look exactly like the following (with different randomly generated values, of course):

```
[0] Student[studentId=3170520, surname=Norrad, givenNames=Claudia, gpa=3.4]
[1] Student[studentId=3211274, surname=Norrad, givenNames=Lynda, gpa=3.6]
[2] Student[studentId=3242897, surname=Gilroy, givenNames=Krista, gpa=2.0]
[3] Student[studentId=3243897, surname=Gilroy, givenNames=Matt, gpa=2.4]
[4] Student[studentId=3296985, surname=Elkins, givenNames=Robert, gpa=3.3]
[5] Student[studentId=3307085, surname=Swanson, givenNames=Krista, gpa=3.6]
[6] Student[studentId=3321249, surname=Norrad, givenNames=Tom, gpa=3.8]
[7] Student[studentId=3328191, surname=Kent, givenNames=Reynold, gpa=2.8]
[8] Student[studentId=3347208, surname=Gilroy, givenNames=Reynold, gpa=3.6]
[9] Student[studentId=3350844, surname=Curtis, givenNames=Tom, gpa=2.0]

St. Id   Result
=======  ======
3170520     0
3350844     9
3243897     3
3170519     -1
3350845     -1
3243898     -1
```

Note that in this example, the first three test values were found in the appropriate positions and the last three test values were not found. Thus all six tests showed that studentBinarySearch() worked as expected. Also, as you can tell, the results need not be vertically aligned.

Submission Instructions

Include the following in your submission document:

- Your name, student number, CS1083, Module 3 Assignment, and the date
- Complete code for your StudentBinarySearch and StudentSearchTest classes *(There is no need to include the Student class)*
- Two (2) samples of output from running StudentSearchTest

D2L DROPBOX SUBMISSION INSTRUCTIONS

Remember to save / export your submission document as a pdf file, and then upload that one pdf file to D2L as follows:

1. In the top-navigation bar on the course screen, select 'Assessments' and then 'Assignments'.
2. Select the assignment title and follow the instructions to upload your submission document.