EECS 233 Programming Assignment #1
100 points
Due October 4, 2019 before 11:59pm

Create a class named *Course* that contains a string data field named *courseID* and an int data field named *capacity*. Provide an abstract data type *CourseList*, that can be used to represent sequences of objects of class *Course*.

The abstract data type *CourseList* must support the following operations:

- *int size()* – Returns the current size of the list (total number of courses).
- *void addCourse(int i, Course course)* - Adds a new element before the i-th element of the list (the index of the first entry is 0). If *i* is greater than the number of the elements in the list, adds it to the end.
- *boolean changeCapacity(String ID, int cap)* – Changes the capacity of the course whose courseID equals ID and returns true. Returns false if ID does not exist in the list.
- *boolean removeCourse(int i)* - Deletes the i-th element of the list and returns true. In the case when the list has less than *i* elements returns false.
- *Course search(int i)* - Returns the i-th element of the list. In the case when the list has less than *i* elements returns null.

Provide two implementations of this abstract data structure: a class named *CourseArrayList* and another class named *CourseLinkedList*. The first one must use an array to store the sequence of courses and the second a singly linked list.

Notes regarding the implementation of these ADT:

- Using built-in Java classes, such as ArrayList and LinkedList, is not allowed.
- You are allowed to use helper classes (e.g. Node).

You are given the demo application utilizing this abstract data structure (called Demo.java).
You should implement the following components:

- Two static methods: *int prefixCountArrayList(CourseArrayList list, String prefix)* and *int prefixCountLinkedList(CourseLinkedList list, String prefix)* which return the total number of courses in the list with the *courseID* starting with that prefix.

In the main method of Demo.java we create an instance of *CourseArrayList*, insert several elements into it, test the *CourseArrayList* methods and then use the *prefixCountArrayList* method to print the number of elements in the sequence with a given prefix. The same steps are repeated with a *CourseLinkedList.* The sample output of the main method is given below (it will be printed twice):

```
----------------------------------------
Capacity of MATH444 is 30

Capacity of EECS233: 140
Change capacity method: true
Change capacity method: false
Capacity of EECS233: 100

Number of courses: 4
Course removed:true
Course removed:false
Number of courses: 3

Number of courses starting with EECS: 2
----------------------------------------
```

Submission and Grading: The submissions will be evaluated on completeness, correctness, and clarity. Please provide sufficient comments in your source code to help the TAs read it. Please generate a single zip file containing all your *.java files needed for this assignment and optionally a README.txt file with explanation about added classes and other extra changes you may have done. Name your file P1_YourCaseID_YourLastName.zip. Submit your zip file electronically to canvas. Grading:

- Abstract data type specification: 10 points
- *CourseArrayList* implementation: 30 points
- *CourseLinkedList* implementation: 30 points
- Demo application: 10 points
- Proper encapsulation/information hiding: 10 points
- Design and style: 10 points
    - Style: 3 points. Are variable names descriptive and convey their purpose? Of course, simple concepts like a loop variable do not require descriptive names; e.g., it is perfectly fine, even preferable, to use i for a loop variable. Is formatting clear and consistent?
    - Comments: 4 points. Comments should aid the reader to understand the code. Comments that restate what is already clear from the code are redundant and not helpful. Nor are comments that are not consistent with the code. For each method implementation, state in the comments its worst-case running time.
    - Design: 3 points. Are there lines that never execute? Inelegant constructions? Convoluted or unnecessarily inefficient ways to achieve some result?