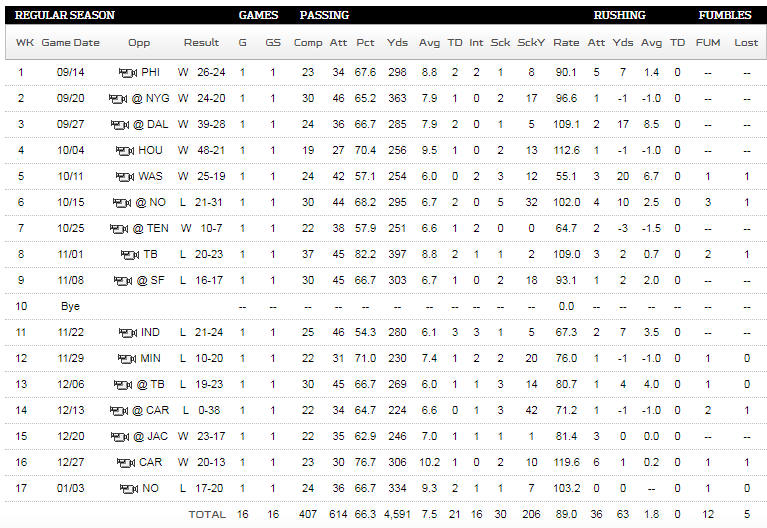
**Lab Assignment – 12  
Due Date: Nov 9 (Friday), 11.59 pm  
Total Points: 150**

**Objective: Working with Arrays**

1. Create a java class that calculates the various statistics of your favorite quarterback in NFL. If you don’t have one, you could use pick Matt Ryan of Atlanta Falcons from his 2015 season. Use either visit nfl.com or their team website for some of the data you would need to complete this assignment.

Implement the main method in the class and do the following in the main method

* + Create an appropriate type of array (yards) to store a number of yards that the QB threw in each of the games they played last season. The length of the array would be set to the number of games he played. The score for game 1 would be in 0th index position and so.
  + Create another array (attempts) that has the same size as the earlier array which will store the number of throw attempts in each of the games. The number of attempts in game 1 would be in 0th index position and so on.
  + Create another array (attempts) that has the same size as the earlier array which will store the number of completions in each of the games. The number of completions in the game 1 would be in 0th index position and so on.
  + Use a for loop to calculate the total yards thrown in the entire season
  + Use a for loop to calculate the number of games where the number of yards was over 275.
  + Create a 3rd array that will store the average number of yards per attempt. Use a for loop to populate this array.
  + Use a for loop to print the average completion percentage for each of the games.
  + Use a for loop to provide the highest yards thrown in a game during the season.



Make sure to keep your output verbose**. (30 points)**

1. Design a user-defined class called *Course*. Each course object will hold data about a course. The data includes *courseID, courseName*, *credits* and *instructorName*. Define the instance variables required of the appropriate type. Provide *only get* methods along with *toString( )* and *equals( )* methods. Use JavaDoc style comments.

Design another user-defined class called *Student.* Each student object will hold data on *studentID, name, GPA, gender*, and *courseList*. Make sure to define *studentID* of *int* type, *name* as *String*, *gender* as a char type and *courseList* as an array of *Course*. Declare a static variable named *nextID* initialized to 1000. *nextID* will be incremented inside the constructor and it’s value would be assigned to *studentID*. Please note the constructor will not require a separate parameter to initialize *studentID.*

Provide *the following get methods: getStudentID( ), getName( ), getGPA(), getGender(), getCourseList( )*

Provide the following set methods: *setName( ) , setGPA( ) , setGender( ), setCourseList( )*

Also implement the *toString( )* and *equals( )* methods. Use JavaDoc style comments.  
***Hint:*** check out the CellPhone.java program (page 499 – 500 in text) and CellPhoneClient (pages 503 – 504).

Create a client program with the main method. Start by creating Course objects for each of the courses you are taking this semester. Next, create two Student objects. Use the course objects you created to populate the courseLists for the two student objects. Demonstrate the various methods of the Student class. **(60 points)**

1. The queue is a container of elements that are inserted and removed according to the first-in-first-out (FIFO) principle – the first element that goes into the queue is the first one that is out of the queue. An excellent example of a queue is a line of students in the food court at NDSU. New additions to the queue are made at the back of the queue, while removal (or serving) happens in the front. In the queue, only two operations are allowed enqueue and dequeue. Enqueue means inserting an element into the back of the queue, and dequeue means removing the front item.

In this programming exercise, you will implement a user-defined class called as Queue to store integer values and provide methods for enqueue and dequeue operations. Implement the following in the Queue class:

***Instance variables***

* + An int array called list to act as a container of the elements of the queue
  + An int value count that will keep count of elements in the queue. (Please remember an array can be partially filled. An example would be the following:

int [ ] x = new int[5];

x[0] = 10;

x[1] = 5;

the count of this array would be 2 as two elements were added to the array.

***Constructors and Methods***

* + A default constructor that initializes the list instance variable to an array of length 5
  + An overloaded constructor that takes an int array as a parameter, and an int value that provides a count of the elements in the array that is being passed as a parameter.
  + Get method -getCount( ) to return the count
  + Get method – getList( ) to return a copy of list. Note this method should return a copy of the list with elements copied from the original list to preserve encapsulation.
  + enqueue() method that takes an int value as a parameter which will be added to the queue. Please note this method would have to first check if the array is full before adding the element to the queue. If the array is full then the method will call a helper method (private method) addCapacity( ) that will increase the capacity of the list to double the original size and copy the values from the original array.
  + dequeue() method that will remove the first element of the value and return that first element and shift the contents of the array to the left by one place.
  + Method maxValue( ) that returns the maximum value that is currently stored in the queue.
  + toString( ) method that returns the contents of the queue as a String.
  + equals( ) method that checks if the contents of two Queue objects are the same.

Next implement a test program, QueueTest with the main( ) method and do the following in the test program

1. Create an int array with length 10 and store 10 random int values in the array between 1 and 100. Make use of the Random class to generate these random values.
2. Create a Queue object using the overloaded constructor and pass the array you created in the first step.
3. Print the contents of the queue.
4. Add the following values to the queue by calling the enqueue method: 32, 5, 73, 82, 19
5. Print the contents of the queue.
6. Print the maximum value in the queue
7. Perform a dequeue operation on the queue and print the value the dequeue method returns
8. Print the contents of the queue.
9. Perform another dequeue operation on the queue and print the value the dequeue method returns
10. Print the contents of the queue.
11. Perform one more dequeue operation on the queue and print the value the dequeue method returns
12. Print the contents of the queue.

**(60 points)**

**Things to turn in:**

* Word document saved using the same convention from the last few lab assignments

**Program - 1**

* + Copy and paste the source code, output, and screenshot

**Program - 2**

* + Copy and Paste the source code of the user-defined classes Student and Course
  + Generate APIs for Student and Course class using JavaDoc utility and copy and paste the APIs for Student and Course class.
  + Copy and Paste the source code of the client program
  + Copy and paste the output of the client program
  + Copy and paste the screenshot of the client class
  + Include the ***Class diagram*** for Student and Course classes.

**Program - 3**

* + Copy and Paste the source code of the user-defined classes Queue
  + Copy and Paste the source code of the QueueTest program
  + Copy and paste the output of the QueueTest program
  + Copy and paste the screenshots of the output window.
* Zip your NetBeans project folder and submit the same with a word document.