CSC 2720 – Spring 2019
Homework #1
Due 16/Feb/2019 11:59 pm

Submission Requirements
You must turn work at the SPECIFIED TIME so you can receive credit for Homework! Homework 1 ***must be submitted*** *by the due date and time. Late homework will be subject to a penalty(of 10%), as stated in the course grading policy. No email or hard copies of homework will be accepted. All the solutions must be submitted through iCollege.*

You may discuss the assignments with other students in the class, but (as stated in the academic honesty policy) your written answers ***must be your own****, and you must list the names of other students you discussed the assignment with.*

Optional: Attach a read me file for how to execute your code.

How to Submit

This homework must be done using java programming language and your .java files must be uploaded to icollege along with the screenshots of the output of your code. Please follow the directions carefully for each question.

1. **(10 points)** Write a class called **Date** that represents a date consisting of a year, month, and day. A **Date** object should have the following methods:

public Date(int year, int month, int day)
Constructs a new Date object to represent the given date.
public void addDays(int days)
Moves this Date object forward in time by the given number of days.
public void addWeeks(int weeks)
Moves this Date object forward in time by the given number of seven-day weeks.
public int daysTo(Date other)
Returns the number of days that this Date must be adjusted to make it equal to the given other Date.
public int getDay()
Returns the day value of this date; for example, for the date 2006/07/22, returns 22.
public int getMonth()
Returns the month value of this date; for example, for the date 2006/07/22, returns 7.
public int getYear()
Returns the year value of this date; for example, for the date 2006/07/22, returns 2006.
public boolean isLeapYear()
Returns true if the year of this date is a leap year. A leap year occurs every four years, except for multiples of 100 that are not multiples of 400. For example, 1956, 1844, 1600, and 2000 are leap years, but 1983, 2002, 1700, and 1900 are not.
public String toString()
Returns a String representation of this date in year/month/day order, such as 2006/07/22".

2. **(10 points)** Given two integers ***start*** and ***end***, where ***end*** is greater than ***start***, write a recursive java method that returns the sum of integers from ***start*** through ***end***, inclusive. Name the java class as **RecursiveSummation.java**

3. **(30 points)** Create an application called **Registrar** that has the following classes:

   A **Student** class that minimally stores the following data fields for a student:
   • name
   • student id number
   • number of credits
   • total grade points earned

   The following methods should also be provided:
   ■ A constructor that initializes the name and id fields
   ■ A method that returns the student name field
   ■ A method that returns the student ID field
   ■ A method that determines if two student objects are equal if their student id numbers are the same (override equals from the class object)
   ■ Methods to set and retrieve the total number of credits
   ■ Methods to set and retrieve the total number of grade points earned
   ■ A method that returns the CPA (grade points divided by credits)

   An **Instructor** class that minimally stores the following data fields for an instructor:
   • name
   • faculty id number
   • department

   The following methods that should be provided
   ■ A constructor that initializes the name and id fields
   ■ Methods to set and retrieve the instructor's department

   A Course class that minimally stores the following data for a course:
   • name of the course
   • course registration code
   • maximum number of 35 students
   • instructor
   • number of students
   • students registered in the course (an array)

   The following methods should also be provided:
   ■ A constructor that initializes the name, registration code, and maximum number of students
   ■ Methods to set and retrieve the instructor
   ■ A method to search for a student in the course; the search should be based on an ID number.
   ■ A method to add a student to the course. If the course is full, then an exception with an appropriate message should be raised (try creating your own exception class for this). Also, be sure that the student is not already registered in the course. The list of students should be in the order that they registered.

■ A method to remove a student from the course. If the student is not found, then an exception with an appropriate message should be raised (use the same exception class mentioned above).
■ A method that will allow Course objects to be output to a file using object serialization
■ A method that will allow Course objects to be read in from a file created with Object serialization

You will note that the **Student** and **Instructor** classes described above have some commonality. Create a **Person** class that captures this commonality and uses it as a base class tor **Student** and **Instructor**. This class should be responsible for the *name* and *id* fields and also provide a *toString* method that returns a string of the form *name, id*. This will be the inherited *toString* method for the **Student** and **Instructor** classes.

a) Draw a UML diagram this application.
b) Implement the previous classes in Java. Write a main program that can serve as a test class that tests all of the methods created and demonstrates that they are working.
c) Write a second main program that allows user to:
  i. create a course, prompting the user for all the course information
  ii. add students to the course
  iii. check to see if a student is registered in the course, and
  iv. remove a student from the course