# COP5007 Java Programming

## Programming Project # 2

*See Dropbox for due date*

### Project Outcomes:

### Develop a Java program that uses:

* decision constructs
* looping constructs
* basic operations on an Array of objects (find, change, access all elements)
* more than one class and has multiple objects
* static variables

### Prep Readings:

Absolute Java textbook, Chapters 1 - 6.

**Project Requirements:**

1. We are going to develop a program that will handle a Golfer and his scores.  The program will be comprised of three classes: a Golfer class that will manage all the golfer's scores, a Score class and a Tester class to drive the other classes.
2. The Golfer Class
   1. This is a class designed to represent a golfer including his name, home course, id number and an Array of Score Objects. The golfers score information consist of the score, course name where the score was shot, the par score for the course, the courses rating and course slope, and the date score was shot. Each field is explained in detail below in the explanation of the Score Class.  Remember part of being a strong programmer is being flexible enough to learn about new domains, you don't have to be a golfer or a doctor or a banker to write application in those domains.
   2. Instance Variables
      * name - A String representing the golfer's name
      * homeCourse - A String representing the golf course where the player's handicap is keep.
      * idNum - A unique integer that identifies each golfer.
      * scores – an Array - stores all the golfer's scores. **(NOTE:  MUST USE AN ARRAY, CANNOT USE ANY OTHER TYPE OF JAVA COLLECTION OBJECT)**
   3. Static Variable - int nextIDNum - starts at 1000 and is used to generate the next Golfer's IDNum
   4. Methods
      * Constructor - sets name and homeCourse from parameters and uses the static variable nextIDNum to retrieve the next available ID number. Creates Array.
      * Constructor - default constructor, sets all instance field to a default value. Creates Array.
      * Accessor and mutator methods for all variables. NOTE Mutator method for IDNum should use the static variable nextIDNum. Mutator method should be used to set all instance fields
      * addScore - create a Score object from the parameters that represent the course, course rating, course slope, date and score.  Adds the newly created Score object to the Array of Scores.
      * deleteScore - delete a score from the Array based on score date,  Assume only one score per day. Returns true if score found and deleted, false if not found.
      * getScore - returns a score object based on the score date. If not found returns null;
      * findScore - private method given a parameter representing the score's date, returns the Array index of a score. (Use in deleteScore and getScore). Return constant NOTFOUND if not found,  NOTFOUND is set to -1;
      * lowestScore - returns the Score object of the lowest score. Note in golf the lower the score the better. Returns null if no scores entered.
      * toString - returns a nicely formatted string of a Golfer's information and all their scores.  Use the Score toString method to assist.  It should looks something like

John Smith   ID number: 1234 Home Course:  Bay Hill CC    
  
Score        Date            Course            Course Rating    Course Slope  
75            6/3/12         Bay Hill CC      69.5                  123  
77            7/23/12      AC Read           70.4                  128

1. The Score Class
   1. Definitions for non-golfers ( for more definitions go to usga.org)
      1. course -  the specific place where golf is played, has a name such as Augusta National or St. Andrews or A,C. Read.
      2. score- The number of strokes taken during a round(18 holes) of golf.
      3. course rating - indicates the evaluation of the playing difficulty of a course for a scratch golfer (zero handicapp) under normal course and weather conditions. It is expressed as strokes taken to one decimal place, and is based on yardage and other obstacles to the extent that they affect the scoring ability of a scratch golfer.
      4. course slope - indicates the measurement of the relative difficulty of a course for players who are not scratch golfers.  The lowest Slope Rating is 55 and the highest is 155. A golf course of standard playing difficulty has a Slope Rating of 113.
   2. Instance Variables
      1. courseName - a String representing the name of the course.
      2. score - an int representing a 18 hole score, it must be between 40 and 200 (inclusive)
      3. date - a String representing the date in format mm/dd/yy
      4. courseRating - a double representing the course rating, it must be between 60 and 80. (inclusive)
      5. courseSlope - an int representing the course slope, it must be between 55 and 155 (inclusive).

NOTE:  Your code should ensure all instance field are valid as described above.   If not write an error message to the standard output and continues.  Set field to 9999

* 1. Methods
     1. Constructor - sets all instance fields from parameters
     2. Default Constructor - sets all instance fields to a default value.
     3. Accessor and mutator methods for all instance variables. Mutator method should be used to set all instance fields
     4. toString - returns a nicely formatted String that contains the score and its instance fields that looks something like:

75            6/3/12         Bay Hill CC      69.5                  123

1. The GolferTester Class
   1. This class should consists of a main method that tests all the methods in each class, either directly by calling the method from the Tester or indirectly by having another method call a method.
   2. The Tester should include valid and invalid data that fully tests the application. No user input, create enough Golfer object to fully test your program.
   3. Ensure you display the contents of the Golfer's array prior to exiting.
2. Create UML Class Diagram for the final version of your project.  The diagram should include
   1. All instance variables, including type and access specifier (+, -);
   2. All methods, including parameter list, return type and access specifier (+, -);
   3. The Golfer and Score class, the GolferTester does not need to be included.
   4. Refer to the UML Distilled pdf on the content page as a reference for creating class diagrams.
   5. A Link to a drawing program, Dia, is also posted on the content page
3. Challenge (bonus 5 points) - use a GregorianCalendar object to handle the date in the score class.
4. Bonus (5 points) - Turn in a self graded grade sheet.

**Submission Requirements:**

Your project must be submitted using the instructions below. Any submissions that do not follow the stated requirements will not be graded.

1.     You should have 5 files for this assignment:

* + Golfer.java  The Golfer class
  + Score.java  The Score class
  + GolfterTester.java  A driver program for your Golfer class
  + Simply UML Class diagram of your two Classes (Golfer and Score)(Dia file or image file , jpg, gif, pdf etc)
  + Self graded Grade sheet.
  + Golfer.html - The javadoc file for the Golfer Class. (Do not turn in)
  + Score.html - The javadoc file for the Golfer Class(Do not turn in)

2. Remember to compile and run your program one last time before you submit it. If your program will not compile, the graders will not be responsible for trying to test it.

3. Follow the submission requirements posted on elearning.

**Important Notes:**

1. Projects will be graded on whether they correctly solve the problem, and whether they adhere to good programming practices.
2. Projects must be submitted by the time specified on the due date. Projects submitted after that time will get a grade of zero.
3. Please review UWFs academic conduct policy. Note that viewing another student's solution, whether in whole or in part, is considered academic dishonesty. Also note that submitting code obtained through the Internet or other sources, whether in whole or in part, is considered academic dishonesty. All programs submitted will be reviewed for evidence of academic dishonesty, and all violations will be handled accordingly.