HOMEWORK 1 - Fall 2015

HOMEWORK 1 - due Wednesday, September 9th no later than 6:00PM

REMINDERS:

- **Be sure your code follows the coding style for CSE214.**
- **Make sure you read the warnings about academic dishonesty. *Remember, all work you submit for homework or exams MUST be your own work.***
- **Login to your grading account and click "Submit Assignment" to upload and submit your assignment.**
- **You may not use any Java API Data Structures to implement this assignment.**
- **You may use Scanner, InputStreamReader, or any other class that you wish for keyboard input.**

In this assignment, you will write a course planner for Stony Brook students. The planner is simply a list where students can record the courses they want to take in order of preference. A course contains general information such as course name, course code, course section, instructor, etc. We will simplify the course to include these basic information. The planner should be able to record a total of 50 courses, a number which should be declared as a final variable.

1. Write a fully-documented class named `Course` which contains the course name (String), department (String), code (int), section (byte), and instructor (String). You should provide accessor and mutator methods for each variable. The mutator methods for code and section should throw an appropriate exception for invalid values (e.g. a negative value). Also, include a constructor for the class, as well as the `clone()` and `equals()` methods. The full list of required methods is:

- **`public Course` - constructor (you may include a constructor with parameters)**
- **getter/setter methods for each variable**
- **`public Object clone()` - the return value is a copy of this Course. Subsequent changes to the copy will not affect the original and vice versa. Note that the return value must be typecasted to a Course before it can be used.**
- **`public boolean equals(Object obj)` - a return value of true indicates that obj refers to a Course object with the same attributes as this Course. Otherwise, the return value is false.**

2. Write a fully-documented class named `Planner` which stores an ordered list of Course objects. The array indices determine the preference of the courses (do not use ArrayList, Vector, or any other Java

API Data Structures). A student can insert a course at any position within the range of the list. Make sure there are no gaps between courses on the list. Note that arrays in Java are 0 indexed, but your listing preference should start from 1. The Planner can record a total of 50 courses, so use the final variable `MAX_COURSES = 50` (and refrain from using the number 50 within your code). The class will be based on the following ADT specification:

`public class Planner`
The Planner class implements an abstract data type for a list of courses supporting some common operations on such lists.

- **`public Planner`**
  Constructs an instance of the Planner with no Course objects in it.
  Postcondition:
  This Planner has been initialized to an empty list of Courses.

- **`public int size()`**
  Determines the number of courses currently in the list.
  Preconditions:
  This Planner has been instatiated.
  Returns:
  The number of Courses in this Planner.

- **`public void addCourse(Course newCourse, int position)`**
  Parameters:
  `newCourse` - the new course to add to the list
  `position` - the position (preference) of this course on the list
  Preconditions:
  This Course object has been instantiated and 1 ≤ `position` ≤ items_currently_in_list + 1. The number of Course objects in this Planner is less than MAX_COURSES.
  Postconditions:
  The new Course is now listed in the correct preference on the list. All Courses that were originally greater than or equal to `position` are moved back one position. (e.g. If there are 5 Courses in a Planner, positioned 1-5, and you insert a Course in position 4, the new Course would be placed in position 4, the Course that was in position 4 will be moved to position 5, and the Course that was in position 5 will be moved to position 6.)
  Throws:
  `IllegalArgumentException`
  Indicates that `position` is not within the valid range.
  `FullPlannerExpcetion`
  Indicates that there is no more room in the Planner to record an additional Course.
  Note 1:
  `position` refers to the position in the Planner and not the position in the array.
  Note 2:
  Inserting an item in position (items_currently_in_list + 1) is effectively the same as adding the item to the end of the list.

- **`public void addCourse(Course newCourse)`**

Works just like `public void addCourse(Course newCourse, int position)`, except adds to the end of the list.
Note:
This method can be written in one line using the `addCourse()` and `size()` methods above.

- **`public void removeCourse(int position)`**
  Parameters:
  `position` - the position in the Planner where the Course will be removed from.
  Preconditions:
  This Planner has been instantiated and 1 ≤ `position` ≤ items_currently_in_list.
  Postconditions:
  The Course at the desired position has been removed. All Courses that were originally greater than or equal to `position` are moved backward one position. (e.g. If there are 5 Courses in a Planner, positioned 1-5, and you remove the Course in position 4, the item that was in position 5 will be moved to position 4.)
  Throws:
  `IllegalArgumentException`
  Indicates that `position` is not within the valid range.
  Note:
  `position` refers to the position in the Planner and not the position in the array.

- **`public Course getCourse(int position)`**
  Parameters:
  `position` - position of the Course to retrieve.
  Preconditions:
  The Planner object has been instantiated and 1 ≤ `position` ≤ items_currenyly_in_list.
  Returns:
  The Course at the specified position in this Planner object.
  Throws:
  `IllegalArgumentException`
  Indicates that `position` is not within the valid range.
  Note:
  `position` refers to the position in the Planner and not the position in the array.

- **`public static void filter(Planner planner, String department)`**
  Prints all Courses that are within the specified department.
  Parameters:
  `planner` - the list of courses to search in
  `department` - the 3 letter department code for a Course
  Preconditions:
  This Planner object has been instantiated.
  Postconditions:
  Displays a neatly formatted table of each course filtered from the Planner. Keep the preference numbers the same.

- **`public boolean exists(Course course)`**

Checks whether a certain Course is already in the list.
Parameters:
`course` - the Course we are looking for
Preconditions:
This Planner and Course has both been instantiated.
Returns:
True if the Planner contains this Course, false otherwise.

- **`public Object clone()`**
  Creates a copy of this Planner. Subsequent changes to the copy will not affect the original and vice versa.
  Preconditions:
  This Planner object has been instantiated.
  Returns:
  A copy (backup) of this Planner object.

- **`public void printAllCourses()`**
  Prints a neatly formatted table of each item in the list with its position number as shown in the sample output.
  Preconditions:
  This Planner has been instantiated.
  Postconditions:
  Displats a neatly formatted table of each course from the Planner.
  Hint: If your `toString()` method is implemented correctly as described below, you will simply need to call it and print the results to the user.

- **`public String toString()`**
  Gets the String representation of this Planner object, which is a neatly formatted table of each Course in the Planner on its own line with its position number as shown in the sample output.
  Returns:
  The String representation of this Planner object.

3. Write a fully-documented class named PlannerManager that is based on the following specification:

`public class PlannerManager`

`public static void main(String[] args)`
The main method runs a menu driven application which first creates an empty Planner object. The program prompts the user for a command to execute an operation. Once a command has been chosen, the program may ask the user for additional information if necessary, and performs the operation. The operations should be defined as follows:

- A - Add Course <Name> <Code> <Section> <Instructor> <Position>
  Adds a new course into the list.
- G - Get Course <Position>
  Displays information of a Course at the given position.
- R - Remove Course <Position>

Removes the Course at the given position.
- P - Print Courses in Planner
  Displays all courses in the list.
- F - Filter By Department Code <Code>
  Displays courses that match the given department code.
- L - Look For Course <Name> <Code> <Section> <Instructor>
  Determines whether the course with the given attributes is in the list.
- S - Size
  Determines the number of courses in the Planner.
- B - Backup
  Creates a copy of the given Planner. Changes to the copy will not affect the original and vice versa.
- PB - Print Courses in Backup
  Displays all the courses from the backed-up list.
- RB - Revert to Backup
  Reverts the current Planner to the backed up copy.
- Q - Quit
  Terminates the program.

4. You will need classes to handle the exceptions thrown (see class specifications above for exception classes you need).

Note: You may include additional methods in any class as necessary or as you find convenient.

INPUT FORMAT:

- **Each menu operation is entered on its own line and should be case insensitive (i.e. 'q' and 'Q' are the same).**
- **Check to make sure that the position, if required, is valid. If not, print and error message and return to the main menu.**
- **For the Add Course command, if the input information is valid, construct the object accordingly. Otherwise, print an error message and return to the main menu.**
- **You may assume Strings are at most 25 characters long.**

OUTPUT FORMAT:

- **Each command should output the result (as shown in sample IO below) after each operation is performed.**
- **All menu operations must be accompanied by a message indicating what operation was performed and whether or not it was successful.**
- **All lists must be printed in a nice and tabular form as shown in the sample output. You may use C style formatting as shown in the following example. The example below shows two different ways of displaying the name and address at pre-specified positions 21, 26, 19, and 6 spaces wide. If the '-' flag is given, then it will be left-justified (padding will be on the right), else the region is right-justified. The 's' identifier is for strings, the 'd' identifier is for integers. Giving the additional '0' flag pads an integer with additional zeroes in front.**

```
String name = "Doe Jane";
String address = "32 Bayview Dr.";
String city = "Fishers Island, NY";
```

```
        int zip = 6390;

    System.out.println(String.format("%-21s%-26s%19s%06d", name, address, city, zip));
    System.out.printf("%-21s%-26s%19s%06d", name, address, city, zip);

    Doe Jane              32 Bayview Dr.              Fishers Island, NY 06390
    Doe Jane              32 Bayview Dr.              Fishers Island, NY 06390
```

HINTS:

- **Remember that the position parameter to all of the methods in the Planner class refers to the position of a Course within the collection (starting at position 1) and not the position in the array (which starts at position 0). There are two ways that you can handle this issue:**
  - **Store the item one in position 0, item two in position 1, and so on and so forth. Inside each method, subtract one from the position given by the parameter to find the appropriate location in the array.**
  - **Define your array duch that it is the size of MAX_COURSES + 1 instead of MAX_COURSES. Store item one in position 1, item two in position 2, and so on and so forth. Position 0 of the array will not be used.**

SAMPLE INPUT/OUTPUT:

```
// Comment in green, input in red, output in black

(A) Add Course
(G) Get Course
(R) Remove Course
(P) Print Courses in Planner
(F) Filter by Department Code
(L) Look For Course
(S) Size
(B) Backup
(PB) Print Courses in Backup
(RB) Revert to Backup
(Q) Quit

Enter a selection: A

Enter course name: Data Structures
Enter department: CSE
Enter course code: 214
Enter course section: 1
Enter instructor: Ahmad Esmaili
Enter position: 1

CSE 214.01 successfully added to planner.

// Menu not shown in sample i/o
```

```
Enter a selection: A

Enter course name: Linear Algebra
Enter department: AMS
Enter course code: 210
Enter course section: 2
Enter instructor: Alan Tucker
Enter position: 1

AMS 210.02 successfully added to planner.

// Menu not shown in sample i/o

Enter a selection: A

Enter course name: System Fundamentals
Enter department: CSE
Enter course code: 220
Enter course section: 2
Enter instructor: Jennifer Wong
Enter position: 2

CSE 220.01 successfully added to planner.

// Menu not shown in sample i/o

Enter a selection: S

There are 3 courses in the planner.

// Menu not shown in sample i/o

Enter a selection: A

Enter course name: Elements of Music
Enter department: MUS
Enter course code: 119
Enter course section: 2
Enter instructor: Taylor Ackley
Enter position: 4

MUS 119.02 successfully added to planner.

// Menu not shown in sample i/o

Enter a selection: F

Enter department code: CSE
```

```
No. Course Name               Department Code Section Instructor
-------------------------------------------------------------------------------
  2 System Fundamentals       CSE         220      01 Jennifer Wong
  3 Data Structures           CSE         214      01 Ahmad Esmaili

// Menu not shown in sample i/o

Enter a selection: G

Enter position: 1

No. Course Name               Department Code Section Instructor
-------------------------------------------------------------------------------
  1 Linear Algebra            AMS         210      02 Alan Tucker

// Menu not shown in sample i/o

Enter a selection: B

Created a backup of the current planner.

// Menu not shown in sample i/o

Enter a selection: L

Enter course name: Elements of Music
Enter department: MUS
Enter course code: 119
Enter course section: 2
Enter instructor: Taylor Ackley

MUS 119.02 is found in the planner at position 4.

// Menu not shown in sample i/o

Enter a selection: R

Enter position: 2

CSE 220.01 has been successfully removed from the planner.

// Menu not shown in sample i/o

Enter a selection: P

Planner:
No. Course Name               Department Code Section Instructor
-------------------------------------------------------------------------------
  1 Linear Algebra            AMS         210      02 Alan Tucker
  2 Data Structures           CSE         214      01 Ahmad Esmaili
  3 Elements of Music         MUS         119      02 Taylor Ackley
```

```
// Menu not shown in sample i/o

Enter a selection: PB

Planner (Backup):
No. Course Name               Department Code Section Instructor
-------------------------------------------------------------------------
  1 Linear Algebra            AMS        210     02 Alan Tucker
  2 System Fundamentals       CSE        220     01 Jennifer Wong
  3 Data Structures           CSE        214     01 Ahmad Esmaili
  4 Elements of Music         MUS        119     02 Taylor Ackley

// Menu not shown in sample i/o

Enter a selection: RB

Planner successfully reverted to the backup copy.

// Menu not shown in sample i/o

Enter a selection: A

Enter course name: Classical Physics I
Enter department: PHY
Enter course code: 131
Enter course section: 1
Enter instructor: Thomas Hemmick
Enter position: 4

// Menu not shown in sample i/o

Enter a selection: P

Planner (Backup):
No. Course Name               Department Code Section Instructor
-------------------------------------------------------------------------
  1 Linear Algebra            AMS        210     02 Alan Tucker
  2 System Fundamentals       CSE        220     01 Jennifer Wong
  3 Data Structures           CSE        214     01 Ahmad Esmaili
  4 Classical Physics I       PHY        131     01 Thomas Hemmick
  5 Elements of Music         MUS        119     02 Taylor Ackley

// Menu not shown in sample i/o

Enter a selection: Q

Program terminating successfully...
```
