CSCI 313 / 613
Homeworks for
4-WEEK SUMMER PROJECT

Over the next 15 classes, you will be working on a comprehensive Data Structures program that will include:

- Creating and using Objects and Comparable types
- Storing Objects in a linked-list structure
- Compiling classes for Linked Lists, Stacks, Queues, Binary Search Trees
- Managing your data class using appropriate data structures
- Reading and writing text files (review)
- Using a Hash Function to access data quickly

Each day will have a short programming assignment for homework that will require that day's lecture. At the end of each week, you will submit the week's work as one program through BlackBoard. You will submit your files in .java or.txt format. There will be a "recap" document for each submitted part of the project that will consist of the "pre-conditions", "post-conditions", and runtimes for each "new" method/class in the project.

Homework #1:
Design a *class Employee implements Comparable*
The data will consist of the employee's first name, last name and ID number (as a String).
Methods will include constructors, sets and gets, equals, compareTo, and toString.
equals will test by ID number
compareTo will test by last name, then first name (only if last names are equal)

Homework #2:
Design a *sortedLinkedList class* using a "dummy node" for the head and a Comparable data type.
DO NOT USE THE JAVA LINKED LIST ADT!

Your *sortedLinkedList* will include methods:
isEmpty, insert (in order), delete(Comparable x), search(Comparable x), toString

Homework #3-4:
Write a program that will prompt the user to enter names and IDs of employees and store the information in a sorted linked list. Your program should give the user the option to continue adding names, insert each name into the list in sorted order, search for an employee, and display the current list. The user should also have an option to "quit".
The program is to be submitted on BlackBoard -> Assignments.

Homework #5:
Write and compile a stack class using an array implementation with a default size of 10, and a queue class using a circular-linked implementation. Test the classes with an Integer type.

Homework #6:
Add the stack and queue classes to *Homework #3-4*.

When a new employee is added to the HRList, the employee is also "pushed" to the LayOff (stack). If an employee is "laid-off", he/she will be removed from the LayOff and will be added to the ReHire (queue).

Homework #7-8:
Modify your current project as follows:

- The user will have the option to add employees, lay off employees, rehire employees, display the current active employees (linked list) or quit.
- A counter will be set for the number of current active employees.
- The employee linked list is to be maintained and displayed in sorted order.
- In the event an employee is laid-off, he/she will be removed from the active employee list and the LayOff, and will be place in the ReHire.
- If an employee is rehired, he/she will be removed from the ReHire and added to the active employee and LayOff
- The main program should consist of just the options (print statements are enough) and appropriate method calls. All of the "work" will be calls to the sortedLinkedList, stack and queue.

The project will be submitted through Blackboard and will be due on June 21 at 11:00 am.

Homework #9:
Write a short program to read and write a text file using simple strings. There is code on BlackBoard to help you if you do not remember how to use text files.

Compile a BST class using recursion as developed in lectures.

Homework #10:
Create a text file for your employee list.

- When the user opens the program, **read** the textfile *active.txt* into the active employee list as you would when adding new employees.
- When the user "quits" the program, **write** the active list back to the textfile, and the ReHire to a separate file *rehire.txt*.

Homework #11-12:
Replace the sortedLinkedList with a BST and make sure the text files are working.

Homework #13-14:
In your **main program**, create an array IDList with default size 100.

- As employees are placed into the active list by name, they are also placed into the IDList using a *hash function = IDNumber%100.*
- To search for an employee by name, use the BST. To search by ID, use IDList.