Faculty of Arts & Sciences
Department of Computer Science
CMPS 200—Introduction to Programming
Assignment 12 – Due Friday Dec 14, 2012

Announcements

- Reminder: Exam 3 will take place on Saturday Dec 15, 9:00-11:00am.
- As usual, the exam will be closed book. You may use your own laptop during the exam, but you have to make sure it has enough battery power to last you for the whole exam period.
- A reminder that you ***must*** use Eclipse for writing programs during the exam.
- We will have an extra "Programming Clinic" open to all on Friday Dec 14 2:00-5:00pm in Bliss 209.

Programs

1. PeriodicTable

Create a data type `Element` for representing elements in the periodic table of elements. Include private variables to store element name, atomic number, symbol, and atomic weight, and accessor methods for each of these values. Create appropriate tests to verify the correctness of the implementation.

Create a data type `PeriodicTable.` The data type should `have` a constructor that reads values from a file to create an array of `Element` objects; a method that allows a user to enter a molecule name, such as H2 O, and obtain its molecular weight; and a method that takes an array of molecule names (`Strings`) and returns an array of names sorted by molecular weight.

The file `elements.txt` on moodle contains the data pertaining to elements, one element per line. Learn how to use the instance method `split()` of the `String` class to help parse the individual lines.

2. Adder

A *stack* is a basic data structure, where insertion and deletion of items takes place at one end called top of the stack. A Stack supports three fundamental operations: push, pop and peek (described below). Write a data type `StackArray` that can support these operations on integers. A stack may be internally represented by an array of integers. The array is expanded as needed. An API for `StackArray` is:

```
public          StackArray()     // constructor
public          getSize()        // returns number of elements in stack
public boolean  isEmpty()        // returns true if stack is empty, false otherwise
public  void     push(int  val)   // places val at index 0 of the array, existing
                                  // elements are pushed by one
public int      pop()            // removes top element from the stack and returns it
public int      peek()           // returns top element without removing it
public String   toString()       // returns printed representation
```

Write a program Adder that initializes an integer variable currentValue to 10 and then does the following:

1- Prompts the user to enter a number or one of the following three actions: Undo, Redo, Exit
2- If the user entered a number, the number is added to currentValue
3- If the user entered Undo, the last action is undone
4- If the user entered Redo, currentValue takes the value it had just before the last Undo
5- If the user entered Exit, the execution stops.

Write appropriate classes or methods to test the operations mentioned above. A sample test file (TestFile.txt) is given to guide your testing.

Hint: Declare two `StackArray` objects, one for the Undo and one for the Redo.

3. CardDeck

In this program, you are to write a class `CardDeck` to represent a standard 52 card deck with operations to shuffle and draw cards from the deck. The class will use the `Card` class from last week's assignment. Here is a possible API with the private fields to store the information about the deck:

```
public class CardDeck {
      private Card[] cards;         // array that holds the 52 cards
      private int top;              // keeps track of the current top of deck,
                                    // initially zero, incremented as cards are drawn

      public CardDeck() {...}          // constructor, creates and fills cards array
      public boolean isEmpty() {...}   // is the deck empty or not?
      public int cardsLeft() {...}     // how many cards are left in deck
      public void shuffle() {...}      // shuffles the deck (see algorithm below)
      public Card draw() {...}         // returns the card on top of deck
}
```

Shuffling an array of N objects (N cards in this case) may be accomplished by looping through the array (using an index i), picking a random object located between i and the end of the array, and swapping it with the object in location i.

Test your implementation of `CardDeck` by writing a program that keeps drawing from a shuffled deck until an Ace is drawn. The program should report how many cards were drawn.

Notes

This assignment is a bit more involved than previous ones as the programs are now more substantial and require some careful planning *before* implementation, and systematic incremental debugging during implementation. Please start early and let us know if you have any questions.