Computer Sciences 302
Midterm Exam 2, 20%

Thursday 11/17, 2011

Print last name: ______________________________, first: ______________

Signature: ______________________________ CS login: ______________

Circle Your	**Lec 1**	**Lec 2**	**Lec 3**	**Lec 4**	**Lec 5**	**Lec 6**	**Lec 7**	**Lec 8**
Lecture	Skrentny	Skrentny	Dan	Peter	Dalibor	Rob	Alicia	Noah

Before you Begin:

(1) Turn in your UW student ID.
(2) **On this examination booklet:**
- Print and sign your name above.
- Write your CS login and circle your lecture above.

(3) Check that there is a total of 11 pages in this exam.
(4) You may not use any notes, books, calculators (or any other electronic devices), or neighbors on this exam. Turn off and put away your cell phone, pager, pda, etc. now.
(5) The exam is intended to take about 100 minutes, but **we will give you 2 hours to complete the exam**.
(6) We can't provide hints but if you need an exam question clarified or feel that there is an error, please bring this to our attention. If needed, **corrections will be written on the board**.

When you've Finished:

(7) Turn in this examination booklet and make sure we return your ID.

Taking the Exam

Write your answers to the questions in this examination booklet. Write neatly and format your Java code by indenting and spacing for readability. Comments are not required, *but if something isn't specified, do something reasonable and state your assumptions*.

Note a REFERENCE is provided at the END OF THE EXAM, which you should review when the exam begins.

Parts	Number of Questions	Topic	Possible Points	Score
I	1	Coding a Complete Program	20	
II	2	Using Pre-built Classes	16	
III	1	Coding an Instantiable Class	19	
IV	3	Using Arrays and ArrayLists	17	
		Total	72	

Part I Coding a Complete Program *[1 question, 20 points]*

[/ 20 points] Use good programming practices to write a complete Java program that allows the program user to repeatedly enter integer values until a negative integer is entered. Your program processes the integer values to determine the following:

- How many positive integers are entered (note positives do not include 0).
- What was the *smallest* positive value entered (note positives do not include 0).
- How many odd integers are entered.
- What was the *largest* odd value entered.
- How many times 0 is entered.
- How many times something other than an integer is entered (e.g., `12.345` or `hello`).

Note, if the user enters something other than an integer then your program should count it (as listed above), clear the input, display a message reminding the user to enter integers, and allow the user to continue to provide input.

After a negative integer is entered the program clearly displays the information in the order listed above and ends. If there is insufficient input to determine the smallest and/or largest values, your program should display "none" instead of a value.

Use this page if additional space is needed to answer Part I.

Part II Using Pre-built Classes *[2 questions, 16 total points]*

1.) [/ 8 points total, 4 each] Full credit is given for answers that make best use of the methods available for these Java pre-built classes. Consult the reference page as needed.

A. Write a code fragment that constructs a random number generator seeded with 11 and displays a (pseudo) random number in the range of 33 - 99.

B. Complete the method below that is passed two Strings and an integer. The method returns true if the first `n` letters of the words are the same, false otherwise.

```
public static boolean match(String word1, String word2, int n) {
```

2.) [/ 8 points] Consider the following classes that are much simpler versions of those used in program 3:

`Highway` class represents a multi-lane highway that has 1 or more lanes:

```
boolean hasLane(int n, boolean d) //returns true if there is another Lane
                            //in the specified direction d from the given
                            //lane number n
Lane    getLane(int n)      //returns a reference to Lane number n
```

`Lane` class represents a lane in the highway that has 0 or more cars:

```
void    addCar(Car c)       //adds Car c to this Lane
void removeCar(Car c)       //removes Car c from this Lane
```

`Car` class represent a car driving in a lane on a highway:

```
int     getLaneNumber()     //returns the number of the lane this car is in
```

Complete the method below that changes the lane for the given car `c` into the lane in the specified `direction` if one exists. Note, if direction is true, the lane change will be to the left --- the current lane number minus one; if false, it would be to the right --- the current lane number plus 1. The method returns true if the lane change occurs, false otherwise.

```
public static boolean changeLane(Highway h, Car c, boolean direction) {
```

Part III Coding an Instantiable Class *[1 question, 19 points]*

[/ 19 points] Assume a grocery store chain is trying to better track for their stores their produce inventory and orders and has chosen to develop an object-oriented program to do this. **Implement the instantiable class**, named `Produce`, used to represent a produce item. Use good object-oriented programming practices and *properly encapsulate numerical values by turning negatives into positive values (for example, -22.3 would be corrected to be 22.3) and for strings simply convert them to lowercase (see reference).*

- The **store** number (e.g., 42, 121)
- The **name** of the produce item (e.g., "banana", "broccoli")
- The **unit** of measurement (e.g., "pounds", "each")
- The **quantity** of measurement (e.g., 0.0, 6.0, 2.7)
- The **inventory status**, where true means the item is in the store's inventory, false means it's on order

- Two constructors; one that allows only a produce item's store, name and unit to be specified with the others being initialized by default, and another constructor that allows a produce item to be fully specified. *Ensure that the given numbers are not negative.*
- A means to safely access each piece of information above. *Note, if you use good object-oriented programming practices you can reduce your work for this part.*
- A method, named `sell`, that reduces a produce item's quantity by a given amount BUT only if the store has at least that amount to sell and it's in inventory. The method returns true if the sale is possible, false otherwise.
- A method, named `transfer`, that allows a produce item's store to be changed BUT only if the item is in inventory and its quantity is greater than 0. If the store is changed, this method also sets the produce item's inventory status to on order to indicate the transfer has been ordered.
- A `toString` method that returns a `String` representing the produce item. Your string will have in this order the store, name, quantity, unit, and inventory status of the produce.
- An `equals` method that is used to determine if this produce item is the same as another object, that is, they have the same name and unit of measurement.

Use this page if additional space is needed to answer Part III.

Part IV Using Arrays and ArrayLists *[3 questions, 17 total points]*

1.) **[/ 5 points] Write a complete method, named `exceedsLimit`,** that is passed an array filled with integers and an integer limit. The method returns true if the sum of the values in the array is greater than the limit, false otherwise.

2.) **[/ 6 points]** Assume `dictionary` is a properly constructed `ArrayList` of `String`s that has been filled with all of the words and word variations (e.g., plurals, different tenses, etc.) found in a dictionary, and `document` is a properly constructed `ArrayList` of `String`s that has been filled with all of the words in a document with each word stored as a separate string. **Complete the code fragment** that determines the indexes of all of the words in `document` that have been misspelled (i.e., not found in the dictionary) and stores those indexes in the `misspelled` array. *You may assume the `misspelled` array already exists and is big enough to store all of the indexes of the misspelled words.*

3.) [/ 6 points] In program 2, you learned that we could represent locations with integer numbers, a *route* of locations could be represented as a one-dimensional (1D) array of integers containing a list of locations in the route, and a *map* of locations could be represented as a two-dimensional (2D) array where each element stores the distance between two locations. For this problem, assume the elements of the 2D array store values corresponding to the cost of travelling between two locations. **Complete the method**, named `findRouteCost`, that is passed a 2D map array and a 1D route array. This method returns the total cost to travel the given route. *You may assume that the 2D array has been properly initialized and that the 1D array is filled with locations for a valid route.*

```
public static double findRouteCost(double[][] costMap, int[] givenRoute) {
```

Exam Reference Page

Methods from the `java.util.ArrayList` class (*REMEMBER 0-based indexing is used):

```
Note the E's below are replaced with the particular ArrayList's element type.
void add(E item)        // Adds the specified item to the end of this list.
void add(int index, E item)// Adds the specified item by inserting it into this
                           // list at the specified index.
boolean contains(E item)// Returns true iff the specified item is in this list,
                        // otherwise false.
E get(int index)        // Returns the item at the specified index in this list.
int indexOf(E item)     // Returns the index of the specified item if it is in
                        // this list, or -1 if the item is not in this list.
E remove(int index)     // Removes and returns the item from the specified index.
boolean remove(E item)  // Returns true iff the specified item was removed from
                        // this list, otherwise false.
int size()              // Returns the number of used elements in this list.
```

Methods from the `java.lang.Math` class:

```
double abs (double n)   //Returns the absolute value of n.
```

Methods from the `java.util.Random` class:

```
Random(int s)           //Creates a new random number generator seeded with s.
int nextInt(int n)      //Returns the next pseudo-random integer value
                        //between 0 (inclusive) and n (exclusive).
```

Methods from the `java.util.Scanner` class:

```
Scanner(System.in)      // Creates a Scanner object that reads from the keyboard.
boolean hasNext()       // Returns true if there's another token of input.
boolean hasNextDouble() // Returns true if the next input is a double value.
boolean hasNextInt()    // Returns true if the next input is an int value.
boolean hasNextLine()   // Returns true if there's another line of input.
String next()           // Returns the next token of input.
double nextDouble()     // Scans the next token of the input as a double.
int    nextInt()        // Scans the next token of the input as an int.
String nextLine()       // Returns the next line of input as a String.
```

Methods from the `java.lang.String` class (*REMEMBER 0-based indexing is used):

```
String(String str)      // Creates a String object given another String.
String toLowerCase()    // Returns a String that is the lower-case
                        // version of the String it is called with.
char charAt(int index)  // Returns the char value at the specified index.
boolean equals(String s)// Returns true iff the contents of this string
                        // is the same as that of the string s.
boolean equalsIgnoreCase(String s)// Returns true iff the contents of the
                                  // this string is the same as that of the
                                  // string s, ignoring differences in case.
int indexOf(int ch)     // Returns the index within this string of the first
                        // occurrence of the specified character.
int lastIndexOf(int ch) // Returns the index within this string of the last
                        // occurrence of the specified character.
int length()            // Returns the length of this string.
boolean startsWith(String prefix)
                        // Returns true if this string starts with the specified
                        // prefix, false otherwise.
boolean startsWith(String prefix, int offset)
                        // Returns true if this string start at the specified
                        // offset with the specified prefix, false otherwise.
String substring(int beginIndex)
                        // Returns a new string that is a substring of this string
                        // starting at beginIndex to the end of the this string.
String substring(int beginIndx, int endIndx)
                        // Returns a new string that is a substring of this string
                        // starting at beginIndx up to but not including endIndx.
```

Print last name: ____, first: __________________

Signature: ____ CS login: __________________

This sheet must be returned with your examination booklet. Please tuck it between the pages of your exam.