Test 2 : CompSci 6

Susan Rodger

November 10, 2005

Name (print): _____

Community Standard acknowledgment (signature): _____

	value	grade
Problem 1	15 pts.	
Problem 2	12 pts.	
Problem 3	12 pts.	
Problem 4	21 pts.	
Problem 5	25 pts.	
TOTAL:	85 pts.	

This test has 13 pages, be sure your test has them all. In addition to the test itself, you should have a list of common methods of the classes that we have studied in class so that you do not need to memorize such details.

Do **not** spend too much time on one question — remember that this class lasts only 75 minutes and there are 5 problems on the exam. That means you should spend no more than *15 minutes per problem.*

Don't panic. Just read all the questions carefully to begin with, and first try to answer those parts about which you feel most confident. Do not be alarmed if some of the answers are obvious.

If you think there is a syntax error or an ambiguity in a problem's specification, then please ask.

In writing code you do not need to worry about specifying the proper `import` statements. Assume that all libraries and packages we have discussed are imported in any code you write.

PROBLEM 1 : (*Short Answer:* (15 points))

Suppose the classes `Persian` and `Siamese` each extend the class `Cat`, and the classes `Cat` and `Dog` each extend the class `Pet`. Further, suppose that only the class `Pet` is abstract. Which of the following assignments are legal (i.e. do not result in a compilation error or throw an exception)?

Write *legal* or *illegal* next to each assignment statement and if it is illegal, briefly explain why (i.e., a compile or run time error because the statement is not valid). Statements that do not declare variables should be viewed independently, as if they do not affect later statements (i.e., the final three statements should be considered separately).

A. `Persian a = new Persian();`

B. `Pet b = new Siamese();`

C. `Dog c = new Siamese();`

D. `Pet d = new Dog();`

E. `Pet e = new Pet();`

F. `Object f = new Cat();`

G. `d = (Pet)f;`

H. `a = b;`

I. `a = (Persian)b;`

J. `b = a;`

PROBLEM 2 : (***I've got a URL Jones (12 points)***)

Each URL on the world-wide web has a corresponding Internet Protocol (IP) address. For example, the IP address of `www.duke.edu` is 152.3.233.20. The table below shows some URLs and the corresponding IP address.

URL	IP-address
www.parking.duke.edu	152.3.9.199
www.pastoralleadership.duke.edu	152.3.90.245
genomics.duke.edu	152.3.232.39
www.events.duke.edu	152.3.9.120
bigbang.phy.duke.edu	152.3.182.5

In this problem URLs and corresponding IP-addresses are stored in strings separated by one space as shown below.

```
"www.parking.duke.edu 152.3.9.199"
"www.pastoralleadership.duke.edu 152.3.90.245"
"genomics.duke.edu 152.3.232.39"
"www.events.duke.edu 152.3.9.120"
"bigbang.phy.duke.edu 152.3.182.5"
```

Part A (4 points)

Write method `getIPaddress` which returns the IP-address portion of a String representing a URL/IP-address pair. The URL and IP-address are separated by one space. For example, `getIP("www.cnn.com 64.236.16.52")` returns the string `"64.236.16.52"`.

```
/**
 * @param s is a URL/IP-address pair, URL separated from IP-address by one space
 * @return the IP-address portion of the String (after the space)
 */
public String getIP (String s)
{









}
```

Part B (8 points)

Complete the method `findIPAddress` which returns a String representing the IP address corresponding to the URL specified by parameter `url`. The array `addresses` stores Strings representing URL/IP-address pairs as described previously. If the URL specified by parameter `url` is not found in the array return `"0.0.0.0"` for the IP-address. For example, if `addresses` represents the data above the calls and values below help explain the method you will write.

call	return value
`findIPaddress(addresses, "www.parking.duke.edu")`	`"152.3.9.199"`
`findIPaddress(addresses, "genomics.duke.edu")`	`"152.3.232.39"`
`findIPaddress(addresses, "cs.duke.edu")`	`"0.0.0.0"`

Complete the method `findIPAddress` below. Your implementation **must** call the method `getIP` implemented in Part A at least once and use its result in determining the result of this method. In this part, you may assume that the method works correctly no matter what you wrote in Part A.

```
public String findIPAddress (String[] addresses, String url)
{
    final String NO_IPADDRESS = "0.0.0.0";









}
```

PROBLEM 3 : (*Out of order:* (12 points))

Write the method `indexOutofOrder` that, given a array of values that are supposed to be sorted in increasing order and returns the index of the first value that is out of order. If none of the values are out of order, it returns -1.

For example, given the list `values` below, then the call `indexOutOfOrder(values)` would return `3`, since `7`, which is in position `3` is the first value out of increasing order.

3	4	9	7	7	12	11	18	10

You will not receive full credit if you cast the objects in the given list to any type more specific than `Comparable` (i.e., there is no reason to call any other methods than `compareTo` on the objects stored in the list).

Complete the method `indexOutOfOrder` below.

```
/**
 * Returns index of the first object that is not greater than
 * the one before it in the given list, making no assumptions
 * about what types the list holds.
 * Note, there is guaranteed to be at least one object in the
 * list, i.e., list.size() >= 1
 */
public int indexOutOfOrder (List<Comparable> values)
{









}
```

What is the loop invariant for your implementation?

PROBLEM 4 : (***When it rains, it pours (21 pts)***)

Data about the montly amount of rainfall for each year is stored in a data file with each year on a separate line, as shown below for ten years. Note, each line is guaranteed to have twelve values.

```
12.0  3.2  2.5  0.4  2.5  1.2  0.3  0.0  0.5  1.4  5.2  2.3
 3.2  3.5 10.4  2.3  1.5  1.2  0.5  0.2  1.5  0.4  2.9  1.4
 3.1  3.5 10.4  2.3  1.5  1.2  0.5  0.2  1.5  0.4  2.9  2.3
 1.2  3.5  8.4  2.3  1.5  1.2  0.5  0.2  1.5  0.4  2.9  1.3
 4.2  3.5  4.2  2.3  1.5  1.2  0.5  0.2  1.5  2.3  2.9  2.3
 5.2  3.5  8.1  2.3  1.5  1.2  0.5  0.2  1.5  0.4  2.9  2.2
 2.2  3.5  9.4  2.3  1.5  1.2  0.5  0.2  1.5  0.4  2.9  2.3
 2.5  3.5 10.4  2.3  1.5  1.2  0.5  0.2  1.5  1.4  2.9  2.3
 2.9  3.5  7.7  2.3  1.5  1.2  0.5  0.2  1.5  4.4  2.9  2.3
 3.6  3.5  1.4  2.3  1.5  1.2  0.5  0.2  1.5  2.1  2.9  2.3
```

You are to complete several methods that operate on this data as a 2D matrix. Since Java does not provide this type of collection as part of its set of standard classes, each part will represent the data as a different collection: Part A as an `ArrayList` of `ArrayList`s of `Double`s; Part B as a 2D `array` of `double`s; and Part C as a simple `array` of `double`s (where each "row" starts on every twelfth index).

For example, to access the second item in the second row of a matrix, item `(1, 1)`, for each different representation, you would use each of the following pieces of code. In the first example, the result of the first call to `get` returns an `ArrayList`, which we then call `get` on to retrieve the second item in that list. In the second example, we are doing the same thing, but the syntax is much more compact. In last example below, note that it is assumed that each row includes twelve values (so items 0-11 represent the first row and 13 the second item in the second row).

matrix implementation	code to access element (1, 1)
`ArrayList<ArrayList<Double>> mat`	`mat.get(1).get(1)`
`double[][] mat`	`mat[1][1]`
`double[] mat`	`mat[13]`

(problem continued on next page ⇒)

Part A (9 points)

Write the function `readData` below that reads data from the file described above and stores it in a matrix stored as an `ArrayList` of `ArrayLists` of `Doubles`. This means that for each line of data, you must create a new `ArrayList<Double>`, add each number on the line to that list, and then `add` that list to the variable `results` declared at the top of the method. In this way, each item in `results` will itself be an `ArrayList`, rather than a simple number.

Complete the method `readData` below.

```
public ArrayList<ArrayList<Double>> readData (Scanner input)
{
    ArrayList<ArrayList<Double>> results = new ArrayList<ArrayList<Double>>();









    return results;
}
```

Part B (5 points)

Write the method `mostRain` below that, given a matrix of rain data as a 2D array, returns the maximum amount of rain that fell in one month over all the years of data.

Complete the method `mostRain` below.

```
public double mostRain (double[][] data)
{









}
```

Part C (7 points)

Write the method `totalForYear` below that, given a matrix implemented as a single array and a specific year, calculates the total rainfall for the given year. Note that in this implementation of a matrix that each "row" of data is twelve values in length, so the first year starts at index 0, the second year starts at index 12, the third at index 24, etc.

Complete the method `totalForYear` below.

```
/**
 * Returns the total amount of rainfall for each month in the given year.
 * Note, year is guaranteed to be valid, i.e., 1 <= year <= (data.length / 12)
 */
public double totalForYear (double[] data, int year)
{









}
```

PROBLEM 5 : (*Count on it:* (25 pts))

Implement a class that represents a set that keeps a count of the total number of copies of each item it contains. To do this, with each item it contains, it keeps a number that represents the number of times that item has been added to the set. For example, the following code:

```
CountedSet set = new CountedSet();
set.add("apple");
set.add("apple");
set.add("banana");
set.add("orange");
set.add("apple");
set.add("orange");
set.remove("orange");
int s = set.size();
int c = set.count();
```

results in the following values being stored:

word	count
apple	3
banana	1

and setting the value of `s` to `2` because there are two unique items in the set and the value of `c` to `4` because there were four total items added to the set. Note, that `remove` removes an item from the set, no matter how many times it has been added to the set.

You are to implement the class `CountedSet` in the following parts of this problem.

(problem continued on next page ⇒)

Part A: (5 points)

Complete the declaration of the class' instance variables below. You should add any private state (instance variables) you think your `CountedSet` class needs in order to implement the constructor and other methods of the class so that they function as shown in the example above.

```
public class CountedSet
{
    // Part A: declare instance variables here









    public CountedSet ()
    {
        // implemented in Part B
    }

    // returns the number of items in the set
    public int size ()
    {
        // implemented in Part B
    }

    // returns the total number of items added to the set
    public int count ()
    {
        // implemented in Part B
    }

    // adds the given item to the set
    public void add (Object toAdd)
    {
        // implemented in Part B
    }

    // removes the given item from the set (no matter how many times it has been added)
    public void remove (Object toRemove)
    {
        // implemented in Part B
    }
}
```

Part B: (20 points)

Write the implementation for the class methods previously declared below. Note, you may lose points on Part A for instance variables that are declared there, but not used in this part.

```
// create an empty set
public CountedSet ()
{









}

// returns the number of items in the set
public int size ()
{









}

// returns the total number of items added to the set
public int count ()
{









}
```

(problem continued on next page ⇒)

```
// adds the given item to the set
public void add (Object toAdd)
{









}

// removes the given item from the set (no matter how many times it has been added)
public void remove (Object toRemove)
{









}
```

Throughout this test, assume that the following classes and methods are available. These classes are taken directly from the material used in class. There should be no methods you have never seen before here. However, if you know of additional methods not listed here, you may use any of them in your solutions.

Integer

```
public class Integer
{
    // The smallest and largest values of type int
    public static final int MIN_VALUE
    public static final int MAX_VALUE

    // Returns the integer represented by the
    // argument as a decimal integer.
    public static int parseInt (String s)

    // Returns a new String object representing
    // the specified integer.
    public static String toString (int i)

    // Returns value of Integer object as an int
    public int intValue ()
}
```

Object

```
public class Object
{
    // Returns true iff o is the same as this object
    boolean equals (Object o)

    // Returns string representation of this object
    String toString ()
}
```

Iterator

```
public interface Iterator
{
    // Returns true if iteration has more elements.
    boolean hasNext ()

    // Returns the next element in this iteration.
    Object next ()
}
```

Comparable

```
public interface Comparable
{
    // Returns a positive value if this object is
    // greater than the given one, 0 if this object
    // is equal to the given one, and a negative
    // value if this object is less than the given one.
    int compareTo (Object other)
}
```

array

```
public class array
{
    // number of objects stored in the array
    public int length;
}
```

Scanner

```
public class Scanner implements Iterator
{
    // Create Scanner that reads data from a file.
    public Scanner (File file)

    // Create Scanner that reads data from a string.
    public Scanner (String str)

    // Change delimiters used to separate items
    public void useDelimiter (String characters)

    // Check if more items are available
    public boolean hasNext ()

    // Get next delimited item as a string
    public String next ()

    // Get next delimited item as an integer value
    public int nextInt ()

    // Get next delimited item as a real value
    public int nextDouble ()
}
```

TreeSet

```
public class TreeSet implements Set
{
    // Constructs a new, empty set
    public TreeSet ()

    // Constructs a new set, containing only the
    // unique elements in the given list
    public TreeSet (ArrayList list)

    // Returns an iterator over the elements in
    // this set.  The elements are returned in
    // ascending order.
    public Iterator iterator ()

    // Returns the number of elements in this set.
    public int size ()

    // Returns true if this set contains o
    public boolean contains (Object o)

    // Adds the specified element to this set
    // if it is not already present.
    public boolean add (Object o)

    // Adds the specified collection of objects to
    // this set if they are not already present.
    public boolean addAll (Collection c)

    // Retains only the elements in this set that
    // are also contained in the given collection
    public boolean retainAll (Collection c)
}
```

ArrayList

```
public class ArrayList implements List
{
    // Constructs an empty list
    public ArrayList ()

    // Returns the number of elements in this list.
    public int size ()

    // Searches for the first occurence of the given
    // argument, returns  -1 if not found
    public int indexOf (Object item)

    // Returns element at index in this list.
    public Object get (int index)

    // Removes element at index in this list; shifts
    // any subsequent elements to the left (i.e.,
    // subtracts one from their indices).
    public Object remove (int index)

    // Removes first instance of element o in this
    // list; shifts any subsequent elements to the
    // left. Returns false if o was the only instance
    // of that object in the list.
    public boolean remove (Object o)

    // Appends specified element to end of this list.
    public boolean add (Object o)

    // Appends all elements from c to end of this list.
    public boolean add (Collection c)

    // Replaces the element at the given position in this
    // list with the given element
    public boolean set (int index, Object o)

    // Returns an iterator over the elements in
    // this list.  The elements are returned in
    // the order they were added.
    public Iterator iterator ()
}
```

String

```
public class String
{
    // Returns the length of this string.
    public int length ()

    // Returns a new string that is a substring
    // of this string. The substring begins at
    // the specified beginIndex and extends to
    // the character at index endIndex - 1.
    public String substring (int beginIndex,
                             int endIndex)

    // Returns the index within this string of
    // the first occurrence of str
    public int indexOf (String str)

    // Returns the index within this string of
    // the last occurrence of str
    public int lastIndexOf(String str)
}
```