Test 2 Review 2

CompSci 6

Fall 2005

Name (print): _____

Honor Acknowledgment (signature): _____

DO NOT SPEND MORE THAN 15 MINUTES ON ANY OF THE QUESTIONS! If you do not see the solution to a problem right away, move on to another problem and come back to it later.

Before starting, make sure your test contains 10 pages.

If you think there is a syntax error, then ask.

	value	grade
Problem 1	10 pts.	
Problem 2	6 pts.	
Problem 3	18 pts.	
Problem 4	16 pts.	
TOTAL:	50 pts.	

PROBLEM 1 : (*No Credit for This* (10 points))

Credit card numbers can be validated using what's called a *checksum algorithm*. The algorithm basically works as follows, where the left-most digit has index 0 (as it does for strings) and the algorithm works from left to right examining each digit of the credit card.

Accumulate a sum based on adding a value obtained from each digit of the credit card as follows, where each k^{th} digit is examined starting with $k = 0$.

- If k is even, add the k^{th} digit to the sum.
- If k is odd, multiply the k^{th} digit by two. If the result is greater than or equal to 10, subtract 9. Add this computed value to the sum.

If the resulting sum is divisible by 10, the credit card number passes the checksum test, otherwise the card number is not valid. Credit card numbers are often 16-digits long, but they can be of any length.

Here's an example for the card number 7543210987654347 showing that it is a valid card number.

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
credit card number	7	5	4	3	2	1	0	9	8	7	6	5	4	3	4	7
value added	7	10-9	4	6	2	2	0	18-9	8	14-9	6	10-9	4	6	4	14-9
total	7	8	12	18	20	22	22	31	39	44	50	51	55	61	65	70

Since this is valid, the call `isValid("2543210987654321")` evaluates to true. However, the call `isValid("1234")` evaluates to false since the computed sum is $1 + 4 + 3 + 8 = 16$ which isn't divisible by 10.

Write the boolean-valued method *isValid* whose header is given on the next page.

In writing *isValid* you might want to use the method `total` below as a model — in particular this method shows how to use the method `Integer.parseInt` to convert a string to a number.

```
/**
 * Add up values of digits in s, return total.
 * This only works if s consists of digits 0..9,
 * e.g., total("12345") returns 15
 * @param s is the string whose digits are totalled
 * @return total of digits in s
 */
public int total (String s) {
    int total = 0;
    for(int k = 0; k < s.length(); k++) {
        total += Integer.parseInt(s.substring(k, k + 1));
    }
    return total;
}
```

(method on next page ⇒)

```
/**
 * Returns true if and only if credit card has a valid checksum.
 * @param ccard consists only of digits '0', ... '9'
 */
public boolean isValid (String ccard)
{









}
```

PROBLEM 2 : (*Empty Intersections* (6 points))

The method `intersection` given below is supposed to create a new `TreeSet` that contains the intersection of the collection of `TreeSet`s passed as an array parameter. However, when it is called, no matter how many sets are passed or what elements they contain, it always returns the empty set, i.e., a set with size of 0 that contains no elements — even though the code below calls the version of intersection you wrote in class, **which you can assume works correctly** to compute the intersection of two sets.

1. Analyze the code below and explain the flaw in logic that causes it not to work as intended.

```
/**
 * Returns a new set that contains only those elements common to
 * all sets in the given array.  Note, there is guaranteed to be
 * at least one set in the array, i.e., sets.length >= 1
 */
public TreeSet intersection (TreeSet[] sets)
{
    TreeSet results = new TreeSet();



    for (int k = 0; k < sets.length; k++)
    {

        results = intersection(results, sets[k]);

    }


    return results;
}
```

2. Concisely explain how to fix the logic error you noted above (the shortest possible change involves adding just a single line). You may note your fix by directly changing the code given above or write a description of your fix below. Your explanation does not need to be in the form of Java code.

PROBLEM 3 : (*This One's a Little Loopy:* (18 pts))

Part A: (8 points)

Write the method `isAcsending` that, given a list of `Comparable` objects, returns `true` if all of the objects are in ascending order (i.e., each object is less than the next one), and `false` otherwise.

You will not receive full credit if you cast the objects in the given list to any type more specific than `Comparable`.

Complete the method `isAcsending` below.

```
public boolean isAcsending (List<Comparable> data)
{




}
```

Part B: (10 points)

Write the function `minInARow` that given a list of numbers and a specific value returns the length of the shortest sequence of those values that appear consecutively.

For example, given the list below, the length of the shortest consecutive sequence of ones is 2, twos is 8, and fours is 1. If the value does not appear in the list, you should return 0.

```
1 1 2 2 2 2 2 2 2 2 1 1 1 1 4 4 1 1 1 4
```

You will not receive full credit if you cast the objects in the given list to any type more specific than `Object`.

Complete the method `minInARow` below.

```
public int minInARow (List<Object> values, Object toFind)
{









}
```

PROBLEM 4 : (*Duped Again:* (16 points))

Part A: (12 points)

Write the method `findDupe` that, given a list of integers each of which is within a range of 1 to one less than the size of the list, returns the first integer within the collection that appears twice. Since there is one more integer in the list than all possible integers in the range, it is guaranteed that at least one number within the collection of integers is repeated. Your function should return the repeated number as soon as it is found.

For example, given a list of 5 integers all within the range 1 to 4 shown first below, your method should return 1 — the only number repeated. Likewise, given a second list of 13 integers all within the range of 1 to 12, your method should return 3 — the first number repeated.

```
4  2  1  3  1
2  3  4  1  3  5  7  6  9  6  1 12 12
```

```
public int findDupe (int[] data)
{




}
```

Part B: (4 points)

If it is guaranteed that *one and only one* integer is repeated within the integers that appear in a list, then explain what is guaranteed about the sum of the integers in the collection (and a potentially faster way to determine the repeated number).

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.

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
    // first occurrence of str after index start
    public int indexOf (String str, int start)

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

Integer

```
public class Integer
{
    // The smallest value of type int
    public static final int MIN_VALUE

    // The largest value of type int
    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 ()
}
```

Iterator

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

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

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 ()
}
```

TreeSet

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

    // 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)
}
```

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)

    // Replaces the element at the specified position
    // in this list with the specified element.
    public Object set (int index, Object element)

    // 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 (Object c)

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

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 ()
}
```