Com S 227
Spring 2019
Exam 2x

DO NOT OPEN THIS EXAM UNTIL INSTRUCTED TO DO SO

Name: Shaunak Lahiri

ISU username: Sblahiri@iastate.edu

Lab section/TA name: **(please circle one)**
Your exam will be returned to you in the section you select.

	M	T	W	R	F
8:00 – 10:00			(14) Jose	(8) Alvin, Diego	
10:00 – 12:00		(12) June, Kamini	(4) Logan, Jeff K.	(5) Alvin, Oyendrila	
12:00 – 2:00			(7) Jacob, Shaiqur	(1) Oyendrila, June	(13) Ethan R., Zoe
2:00 – 4:00		(15) Ethan V.	(9) Sovann, Marcie	(6) Waqwoya, Long	
4:00 – 6:00		(3) Mailin, Jay	(2) Jack, Jeff Y.		
6:00 – 8:00			(11) Yonas, Alex	(10) Waqwoya, Evan	

Closed book/notes, no collaboration, no electronic devices, no headphones. Time limit 70 minutes. Partial credit may be given for partially correct solutions. *If you have questions, please ask!*

- There are some excerpts from the Java API documentation on the last 1.5 pages
- Use correct Java syntax for writing code.
- You do not need comments or import statements.
- It is ok to use literal numbers in your code
- You can write "SOP" to abbreviate "System.out.println"

Question	Points	Your Score
1	10	10
2	12	12
3	12	13
4	15	15
5	10	10
6	16	16
7	25	25
Total	100	100

100

1. (10 pts) Write a static method that, given an **ArrayList** of strings, returns **true** if all strings in the list are longer than 4 characters. *Assume that the list and all elements are non-null.*

```
public static boolean allStringsAreLong(ArrayList<String> list)
{   boolean condition = true;
    for(String e: list)
    {
      if(e.length() > 4)
      {
         condition = true;
      }
      else
      {  condition = false;
         return condition;
      }
    }
   return condition;
}
```

10/10

2. (12 pts) Write a static method that, given a string, returns a palindrome of it by *prepending* its characters in reverse. *For example*, given the string **"taco"**, the method returns **"ocattaco"**. If the given string has length 1 or less, it is returned without modification. Assume **s** is non-null.

0 1 2 3
t a c o Array length

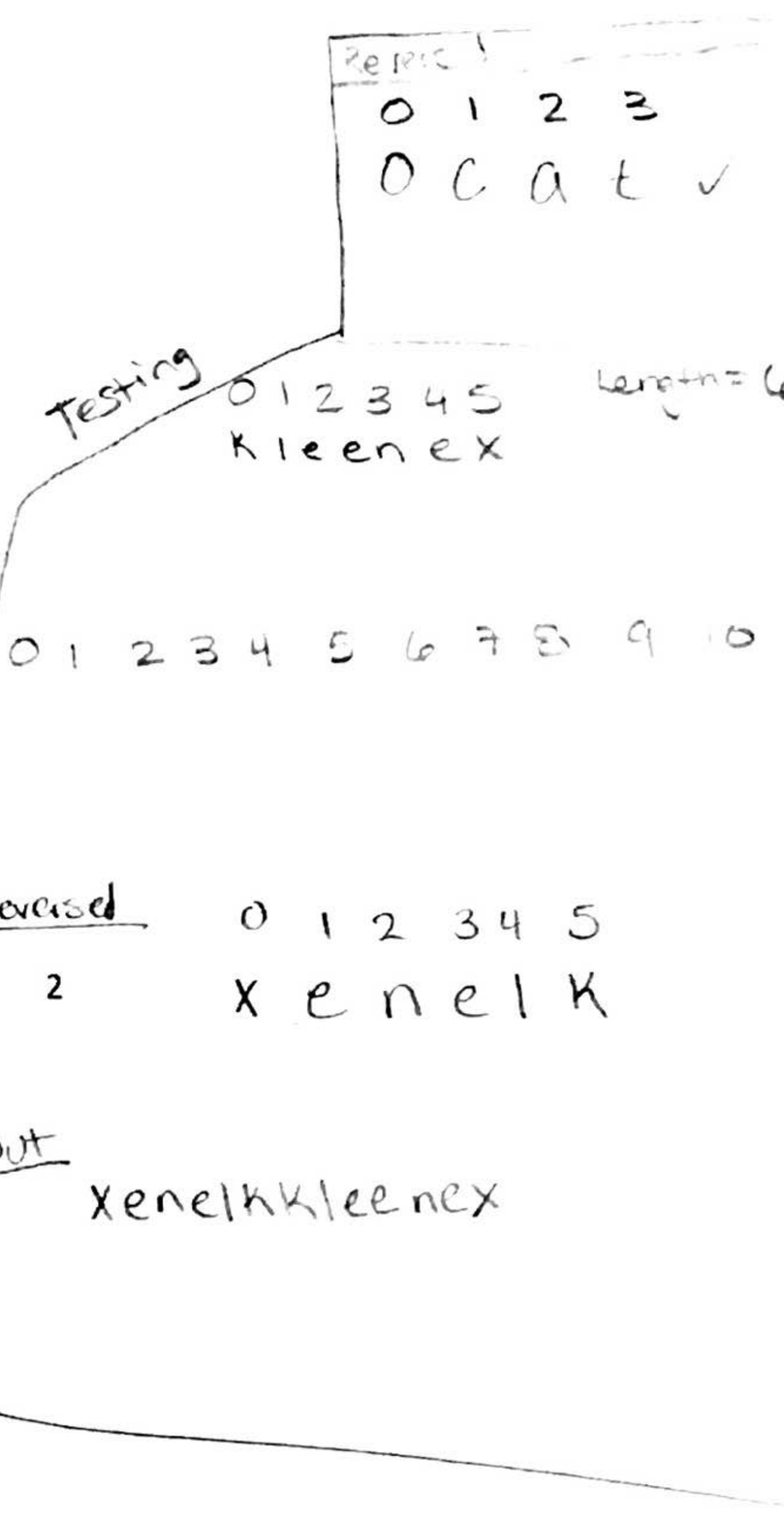
0 1 2 3 4 5 6 7
o c a t t a c o

12

```
public static String makePalindrome(String s)
{
   if(s.length() ≤ 1)
   { return s;
   }
  else
   {  char[] reversed = new char[s.length()];
      int j = s.length() - 1;

      for(int i = 0; i < reversed.length; i++)
      {
           reversed[i] = s.charAt(j);
            j -= 1;
      }
      String out = "";
      for(char e: reversed)
      {
        out += e+"";
      }
      out += s;
      return out;
   } //End if/else

} //End makePalindrome method
```



3. (12 pts) Write a static method that, given an array of **positive** integers, returns the largest value in the array that is not a multiple of 3. *For example, if the array contains [6, 5, 8, 7, 12], the method returns 8.* If the array is empty, or contains only multiples of 3, the method returns 0.

```
public static int largestNonMultipleOf3(int[] arr)
{ ArrayList<Integer> vals= new ArrayList<Integer>();
  if (arr.length<1)
  { return 0;
  }
  for(int K: arr )
  { vals.add(K);
  }
  int max = vals.get(0); int index=0;
  for (int i=0; i< vals.size(); i++) //Finding MAX
  { if(val.get(i).> max)
    { max = val.get(i); index= i;
  } }
  }
} //End method
```

```
if(max % 3 ==0) //Multiple of 3
{ vals.remove(index);
  int[]modify=new int[vals.size()];
  for(int i=0;i<modify.length;i++)
  {modify[i] = vals.get(i);
  }
  largestNonMultipleOf3(modify);
}
else
{
 return max;
}
```

2/12

4. (15 pts) Write a static method that shifts elements of an array to the right and puts the last element at the beginning. The method must **modify the given array** and return **void**. *(For example, given [5, 4, 6, 2], after executing this method the array contains [2, 5, 4, 6].)* You can assume the array is non-null, and that an empty or one-element array can be left unmodified.

15

```
public static void shiftRight(int[] arr)
{
  if(arr.length<2)
  { continue;
  }
 else
  {  int temp = arr[arr.length-1];
     for(int i= arr.length-1; i>0; i--)
     {
       arr[i] = arr[i-1];
     }
      arr[0] = temp;
  }
} //End method
```

5. (10 pts) Starting with **arr** equal to [2, 3, 8, 1, 4], trace execution of the call **mystery(arr)** by writing down, in the empty boxes below, the contents of the array each time the line labeled **(**)** is reached in the code. (There may be more boxes than you need.)

arr.length = 5

0	1	2	3	4
2	3	8	1	4
2	3	4	1	8
2	3	1	4	8
2	1	3	4	8
1	2	3	4	8

10

```
public static void mystery(int[] arr)
{
  for (int i = arr.length - 1; i > 0; i = i - 1)
  {
    int max = i;
    for (int j = i - 1; j >= 0; j = j - 1)
    {
      if (arr[j] > arr[max])
      {
        max = j;
      }
    }
    int temp = arr[i];
    arr[i] = arr[max];
    arr[max] = temp;
    // (**)
  }
}
```

6. (15 pts) a) Given the recursive method **topSecret** below, show all output printed for the method call **topSecret(4)**. **Write just the output in the box at right.**

```
public static int topSecret(int x)
{
  if (x <= 0)
  {
    System.out.println("boo");
    return 2;
  }
  else
  {
    int temp = x + topSecret(x - 1);
    System.out.println(temp);
    return temp;
  }
}
```

output

boo
3
5
8
12

Calls

temp = 4 + topsecret(3) → 4 + 8 = 12 (return)
temp = 3 + topsecret(2) → 3 + 5 = 8 (return)
temp = 2 + topsecret(1) → 2 + 3 = 5 (return)
temp = 1 + topsecret(0) → 1 + 2 = 3

b) On your birthday you get a present from a friend who is kind of a practical joker. When you unwrap it, you discover it's actually a *box*. Inside the box there are a few real items - e.g. there is a chocolate bar, and a small diary with a picture of a unicorn on the cover - but there are also *more boxes*. Well, when you start to open those boxes, you find the same thing: each one might contain some real items but possibly contains more boxes too. Suppose all this is represented by a **`Present`** class with the following methods:

```
// returns true if this present is actually a box
public boolean isBox()

// If this present is a real item (not a box), returns a description of the item (e.g. "unicorn diary").
// (If it's a box, returns an empty string.)
public String getDescription()

// If this present is a box, returns an array of all presents it contains.
// (If it's not a box, returns an empty array.)
public Present[] getContents()
```

Write a **recursive** method that, given a **`Present`** object, prints out the descriptions of all the real items that you actually receive (of course there may be just one, if the initially given **`Present`** is not a box).

```
public static void printAllDescriptions(Present p)
{
   if( P.isBox()!=true)
   {  System.out.println(P.getDescription);
    }
 else
   {  Present[] lot = P.getContents();
      for(Present e: lot)
      {  PrintAllDescriptions(e);
       }
   }
}//End method
```

16/16

7. (25 pts) This problem refers to the class **`Observation`**, with the constructor below, that is used for recording weather observations. (You can assume that in addition to the constructor, the class includes various accessor methods and so on, but they are not needed for this problem.)

Constructor Summary

Constructor and Description

`Observation(String date, int low, int high)`

Constructs a new Observation with the given date string (mm/dd/yyyy) and given low and high temperatures.

A **text file** consists of lines that have a date followed by two numbers, representing the low and high temperatures for that date. A short sample of such a text file is shown below.

```
03/09/2006 42 54
03/10/2006 -10 5
03/11/2006 53 112
```

Write a static method that, given the *name* of a file having the format above, returns an **`ArrayList`** of **`Observation`** objects, one for each line of the file.

Do not write the **`Observation`** ***class****, assume it is available for you to use. Do not worry about import statements.*

```
public static ArrayList<Observation> readWeatherFile(String filename)
    throws FileNotFoundException
{   ArrayList<Observation> results = new ArrayList<Observation>();
    File f = new File(filename);
    Scanner scan = new scanner(f);
    while(scan.hasNextLine())
    {   String line = scan.nextLine();
        if(line.equals(null) || line.equals(""))  //Possible to add characters which cause errors
        {  continue;                              // further down in the processing code
        }
        else
        {   scanner myscanner = new scanner(line);
            String date = line.next();
            int low =  Integer.ParseInt(line.next());
            int high =  Integer.ParseInt(line.next());
            Observation myObservation = new observation(date, low, high);
            results.add(myObservation);
        }
    } //End while loop
    return results;
} //End method
```

There is more space on the next page if you need it