Com S 227
Spring 2019
Exam 1x

DO NOT OPEN THIS EXAM UNTIL INSTRUCTED TO DO SO

Name: Shounak Lahiri

ISU username: Shlahiri@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, Marcia	(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 page
- 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	20	19
2	15	15
3	10	10
4	30	30
5	10	10
6	5	3
7	10	10
Total	100	97

97

1. (20 pts) a) Assume the declarations at right. For each expression in the left-hand column below, give its *value* and its *type* (int, double, boolean, String, etc.) in the second and third columns. If what's written is not a valid Java expression, just write "error" in both columns. *(The first one is done for you as an example.)*

```
int k = 17;
double d = 2.5;
String name = "george";
Rectangle r1 = new Rectangle(1,2,3,4);
Rectangle r2 = new Rectangle(1,2,3,4);
```

Expression	Value	Type
`d * k`	42.5	double
`k / 3`	5	integer
`name.length() != k`	true	Boolean
`name.length() = k`	Error	Error
`k % 3`	2	integer
`k.length() == 2`	Error	Error
`!(k > 0 && d > 5)` (k<0 && d<5)	true	Boolean
`r1 == r2;`	False	Boolean

-1

(b) - (d) Complete the right-hand side of each assignment by writing an expression satisfying the stated requirement. (In some cases there is more than one correct answer. *Do not write any additional lines of code, just write an expression.*) The first one is done for you.

(Example) A boolean value indicating whether the int variable `x` is positive

```
boolean isPositive = x > 0;
```

(b) A boolean value indicating whether the int variable `x` is an even number

```
boolean isEven =        x % 2 == 0;
```

(c) Given an `int` value `x`, a boolean indicating whether `x` is greater than 12 and less than 17

```
boolean isInSpecifiedRange =    (x > 12 && x < 17);
```

(d) Given a int value `h` of hours and `m` of minutes, a `double` that is the total amount of time in hours (e.g. 30 minutes is 0.5 hours)

```
double totalTime =  (h * 1.00) + (m / 60.0);
```

19

2. (15 pts) The Speedy Deluxe Pizza company sells large pizzas for 8.00 each, or you can get 5 for 30.00. Normally there is a delivery fee of .50 per pizza, but delivery is free to ISU students. Write an interactive main method that prompts the user to enter the number of pizzas and a "yes" or "no" indicating whether or not they are an ISU student, and prints the total cost of the order including delivery. *A sample interaction is shown below, with user's responses in* **bold** *Note that the price of 49.50 is correct for the inputs shown. It is ok to assume that anything other than the exact string "yes" can be interpreted as a "no".*

```
How many? 7
ISU student (yes/no)? nah
Price: 49.5
```

```
import java.util.Scanner;
public class SomeClass
{
    public static void main(String[] args)
    {
     Scanner in = new Scanner (System.in);
     System.out.Println ("How many? ");
     String Pizzas = in.nextInt();
     System.out.println("ISU Student (yes/no)?");
     String Student = (in.next()).toLowerCase();
     double Price = 0.00;
     if( (Student.equals("yes")) == false)
     {  Price += Pizzas*.50;
     }
     if( (Pizzas/5) >0)
     {  Price += (Pizzas/5) *30.00;
        Pizzas = Pizzas %5;
     }
     if (Pizzas >0)
     {  Price += Pizza * 8.00;
     }
    System.out.Println("Price: " + Price);
    } //End method
} //End Class
```

(15)

Problems 3 and 4 refer to a class named `GumballMachine`. See overview below. Detailed specification in Javadoc form is at the top of the next page.

In problems 3 and 4 we will refer to a class named **`GumballMachine`** that models an old-fashioned gumball machine. Its API documentation is shown below. A gumball machine contains a certain number of gumballs, up to a fixed maximum. It has a handle with a slot. A coin can be inserted into the slot. If the handle is turned when a coin is in the slot, the coin goes inside the machine and a gumball comes out, if possible. (That is, if the machine is empty, the coin still goes in, but the number of gumballs never goes below zero.) Turning the handle when there is no coin in the slot has no effect. More gumballs can be added to refill the machine. Assume that the machine can hold an unlimited number of coins.

(Before beginning problems 3 and 4, see the overview in the box on the previous page)

Constructor and Description
GumballMachine (int givenMax) Constructs a gumball machine that can hold the given maximum number of gumballs. Initially the machine is empty and contains no coins.

Modifier and Type	Method and Description
void	**turnHandle**() Simulates turning the handle on the gumball machine. If there is a coin in the slot, the coin goes into the machine, and the number of gumballs is reduced by 1 if possible. If there is no coin in the slot, this method does nothing.
void	**add**(int howMany) Adds the given number of gumballs to the machine, if possible (the number of gumballs in the machine cannot go over the maximum value specified in the constructor).
int	**getNumGumballs**() Returns the number of gumballs currently in the machine. This value is never negative.
int	**getTotalCoins**() Returns the total number of coins currently in the machine.
void	**insertCoin**() Inserts a coin into the slot. If there is already a coin in the slot, this method does nothing.

3. (10 pts) Assume that you have a class **GumballMachine** with the constructor and methods given in the Javadoc above. Write a short main method that will **test** whether the **GumballMachine** implementation is working correctly for the following three cases (for each case, your output should print the *actual* result and the *correct* result, no other output is required):

1. For a machine constructed with capacity 100 gumballs, after add(75) twice, it should contain 100 gumballs.
2. For a machine containing 100 gumballs, after insertCoin() and turnHandle(), it should contain 99 gumballs and one coin.
3. Turning the handle when no coin is present should not change the number of gumballs.

Assume you do not need any import statements.

```
public class GumballMachineTest
{
  public static void main(String[] args)
  {
    GumballMachine g = new GumballMachine(100);
     g.add(75);
     g.add(75);
    System.out.println("Expected: 100 ; Actual: " + g.getNumGumballs());
     g.insertCoin();
     g.turnHandle();
    System.out.println("Expected: 99 ; Actual: " + g.getNumGumballs());
    System.out.println("Expected: 1 ; Actual: " + g.getTotalCoins());
     g.turnHandle();
    System.out.println("Expected: 99 ; Actual: " + g.getNumGumballs());
    System.out.println("Expected: 1 ; Actual: " + g.getTotalCoins());
  } //End main
} //End class
```

(Before beginning problems 3 and 4, see the overview in the box on page 3!)

4. (30 pts.) Write the complete class **`GumballMachine`** according to the Javadoc on the previous page. Remember that all instance variables should be **`private`** and the constructor and methods should be **`public`**.

30

```
Public class GumballMachine
{
   Private int maxGum;    // Max gumballs
   Private int gum;       // Current number of gumballs
   Private int totCoins;  // Total number of coins
   Private boolean coin;  // Whether their is a coin in the slot

   Public GumballMachine (int givenMax)
   {  maxGum = givenMax;
      gum = 0;
      totCoins = 0;
      coin = false;
   }

   Public void turnHandle ()
   {  if (coin == true)
      {  totCoins += 1;
         coin = false;
         gum = Math.max ((gum - 1), 0);
      }
   }
   Public void add (int howMany)
   {
      gum = Math.min ( (gum + howMany), maxGum);
   }
   Public int getNumGumballs()
   {  return gum;
   }
   Public int getTotalCoins ()
   {
     return totCoins;
   }
   Public void insertCoin ()
   {
      coin = true;
   }
} //End class
```

. *(Additional space for Problem 4, if needed)*

5. (10 pts) Write a static method **guessThatPrice** that uses an instance of **Random** to guess the price of a prize in a quiz show. The price is a minimum of 3 dollars, and is a maximum of 24 dollars and 99 cents. The method should return a string of the form "*d* dollars and *c* cents" where *d* is a random value 3 through 24, and *c* is a random value from 0 through 99.

```
import java.util.Random;
public class SomeClass
{
  public static String guessThatPrice()
  {
   // TODO
   Random rand = new Random();
   int d = (rand.nextInt(21))+3;
   int c = rand.nextInt(100);
   return   d + " dollars and " + c + " cents";
  } //End method
}//End class
```

6. (5 pts) Rewrite the **foo** method at right so that it does exactly the same thing but does not use *any* conditional statements (no "**if**" or "**if/else**" statements, ternary expressions, **while** statements, etc.). *(Partial credit may be given if you do it with just one conditional statement.)*

```
if (x > 0 && x > y)
{
  return true;
}
else
{ return false;
}
```

(3) -2

```
public boolean foo(int x, int y){
  if (x > 0){
    if (y > x){
      return false;
    }
    else{
      return true;
    }
  }
  else {
    return false;
  }
}
```

max(x,y)

7. (10 pts) A class **Elvis** is shown at right. The class **Exam1** below has a main method:

```
public class Exam1
{
  public static void main(String[] args)
  {
    Elvis r1 = new Elvis(3);
    Elvis r2 = new Elvis(5);
    Elvis r3 = r2;
x=3 System.out.println(r1.rock(7));
x=3 System.out.println(r1.rock(-1));
x=100 System.out.println(r1.rock(7));
    r3.roll(42);
    System.out.println(r2.rock(7));
  }
}
```

In the box below, write down the **output printed on the console** when the **Exam1** main method is run.

```
6
-2
200
84
```

```
public class Elvis
{
  private int x;

  public Elvis (int givenX)
  {
    x = givenX;
  }

  public void roll(int givenX)
  {
    x = givenX;
  }

  public int rock(int y)
  {
    int z = 2 * y;
    if (z < 0)
    {
      x = 100;
    }
    else
    {
      z = 2 * x;
    }
    return z;
  }
}
```

r1	X	return
1st	3	6
2nd	3	-2
3rd	100	200

r2=r3	X	return
1st	5	N/A
2nd	42	84

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

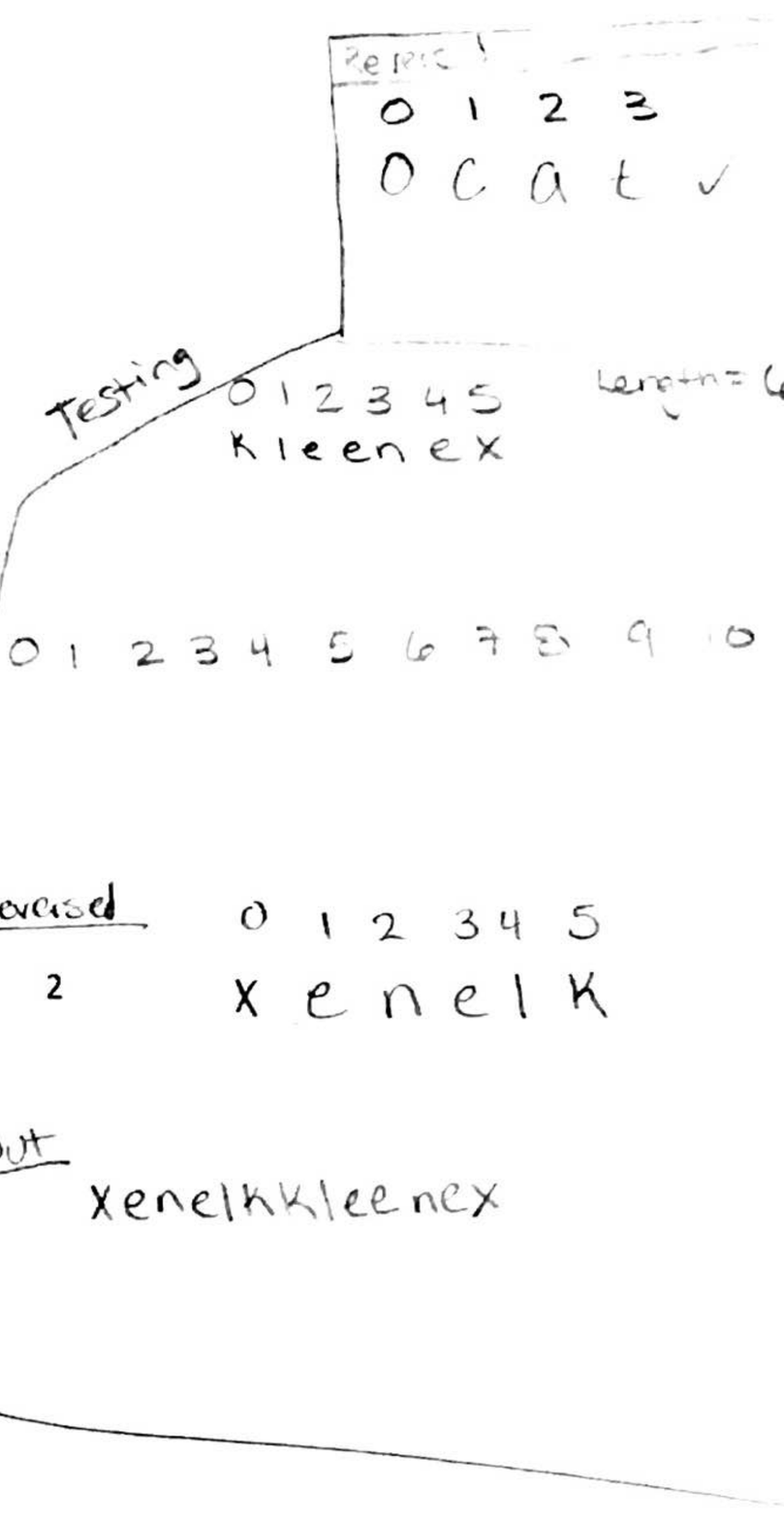
0 1 2 3 4 5 6 7
o c a t t a c o

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