Final Examination

CS 1324 Fall 2017

Name (printed legibly):

Student number:

Do not write on the back of any pages. Do not tear out or add pages.

Answer all programming questions in Java. Show your work to receive partial credit.

You do not need to include import statements, throw any FileNotFoundException or other exceptions, or use constants in any problem. Although comments can help me understand your solution, and should always be included in programs, they are not required on the examination.

Pay careful attention to whether the question asks for a signature, a code fragment, a method, or a complete program. Do not write a whole program when you are asked for only a few lines of code.

Tables may contain too many or too few lines. If there are too many, leave the extras blank. If there are too few, add additional lines to the table.

Things in solid boxes are required. Things shown in dotted boxes are not required, but may be used if you wish. Make sure that there are values in all solid boxes. ***For example, make sure that you've written your name and student number above and signed the integrity statement below.***

Pay attention to things that are in bold (especially if they are underlined and italicized). These are important statements that I think students might miss.

No problem on this examination requires more than fifteen to twenty lines of code. Most require ten or less. If you are writing pages of code, stop and think about whether there is an easier way to do this.

Remember that array and ArrayList are different.

Integrity Pledge

On my honor, I affirm that I have neither given nor received inappropriate aid in the completion of this exercise.

Signature:

1. (13 points; 3 points for a), 4 points for b), 4 points for c), 2 points for d))

a) Select the best type (int, double, boolean, String, char) for each data element listed below.

i) The number of pairs of shoes donated to Shoes For Kids on Monday. int

ii) The dollar value of the shoes donated to Shoes For Kids this month. double

iii) The name of a company that donates to Shoes For Kids. String

b) For the mathematical expression below, fill in the table below by indicating which operation is performed first, second, etc. and whether a promotion (changing an int to a double) has to be done to complete the operation. You do not have to perform the operations.

4.1 + 3 * 5 / 9 % 2

Order	Operation (+, -, *, /, %)	Promotion (Yes/No)
First	*	No
Second	/	No
Third	%	No
Fourth	+	Yes

c) What will the following code print out?

```
int number = 6;
if (number % 2 == 0 )
{
    System.out.println(“number is divisible by 2”);
}
else if (number % 3 == 0)
{
    System.out.println(“number is divisible by 3”);
}
else if (number % 2 == 0 && number % 3 == 0)
{
    System.out.println(“number is divisible by 2 and 3”);
}
```

number is divisible by 2

d) 27 / 4 is: 6

2. (6 points) Trace the code below in the table below.

```
int[] first = {1, 3, 6, 8};
int[] second = {2, 3, 5, 6};

int[] result = new int[first.length];
int resultSize = 0;

for (int firstIndex = 0; firstIndex < first.length; ++firstIndex) {
   boolean found = false;
   for (int secondIndex = 0; secondIndex < second.length && !found; ++secondIndex) {
      if (first[firstIndex] == second[secondIndex])
         found = true;
   }

   if (found) {
      result[resultSize] = first[firstIndex];
      ++resultSize;
   }
}
```

firstIndex	first[firstIndex]	second	second[secondIndex]	found	result array contents	resultSize
0	1	0	2	false		0
		1	3	false		
		2	5	false		
		3	6	false		
		4				
1	3	0	2	false		
		1	3	true	3	1
2	6	0	2	false		
		1	3	false		
		2	5	false		
		3	6	true	3, 6	2
3	8	0	2	false		
		1	3	false		
		2	5	false		
		3	6	false		
		4				
4						

3. (6 points} Find the values stored in data and average at the end of the program. You may use the memory diagram below if you wish to, but are not required to. You must fill out the solid box.

```
public class Test {

   public static void main(String[] args) {

      int[] data = {1, 3, 5, 7, 9};

      double average = 0.0;
      mean(data, average);
   }

   public static void mean(int[] data, double avg) {

      int[] copy = new int[data.length];
      copy[0] = data[0];
      for (int index = 0; index<data.length-1; ++index) {
         copy[index+1] = copy[index] + data[index+1];
      }

      avg = copy[copy.length-1]/ (double) copy.length;
   }
}
```

At the end of the program:

data contains: {1, 3, 5, 7, 9}

average contains: 0.0

OPTIONAL

main stack frame

Identifier	Address	Contents
	100	
	101	
	102	

mean stack frame

Identifier	Address	Contents
	200	
	201	
	202	
	203	
	204	

heap

Identifier	Address	Contents
	1000	
	1001	
	1002	
	1003	
	1004	
	1005	
	1006	
	1007	
	1008	
	1009	
	1010	
	1011	
	1012	
	1013	
	1014	
	1015	
	1016	

4. (15 points; 9 points for a) (3 points for algorithm name, 6 points for trace), 6 points for b))

a) Perform either selection sort or insertion sort on the array below.

Circle the name of the algorithm you are using below. The name of the algorithm matching the code is worth 3 points.

Selection sort Insertion sort

Each row should contain either one or two values. If the algorithm swaps data, the pair of swapped values should be in a single row. If the algorithm does not swap data, then just a single value should be in each row. Do not put values that are not changed in the table.

Selection sort

8	3	1	2	7	5	Auxiliary
1		8				
	2		3			
		3	8			
			5		8	
				7		

Insertion sort

8	3	1	2	7	5	Auxiliary
						3
	8					
3						
						1
		8				
	3					
1						
						2
			8			
		3				
	2					
						7
				8		
			7			
						5
					8	
				7		
			5			

b) Show the order in which data in the array below would be probed when the binary search algorithm is used to find the value 58.

1	14	19	27	33	42	49	51	58	66	73	79	82	94	95
							1	4	3		2			

Do not write anything on the back of any page

5. (10 points) Write a method called longestSequence as described in the documentation below.

public static int longestSequence(int[] array)

The method will return the length of the longest sequence of integers in the perfect size array. For example, if the array contained {4, 5, 1, 2, 3, 4, 5, 2, 3, 4, 5, 6, 7, 8, 4, 2}, the method would return 6 because the sequence from 2 to 7 is the longest and contains six elements. For the array {1, 3, 5, 2, 4, 6}, 1 would be returned since there are no sequences.

```
public static int longestSequence(int[] array)
{
    int max = 1;
    int sequence = 1;
    for (int index = 1; index < array.length; ++index)
    {
        if (array[index-1] + 1 == array[index])
        {
            ++sequence;
            if (sequence > max)
            {
                max = sequence;
            }
        else
            sequence = 1;
    }
    return max;
}
```

6. (10 points; 4 points for a) and 6 points for b))

The signature for the method in problem 5 is below:

int longestSequence(int[] array)

a) ***Rewrite the method signature*** (return type, method name, parameters) for problem #5 if the array data is oversized.

```
public static int longestSequence(int[] array, int arraySize)
```

b) Show the method from ***problem 5 (on the previous page) being called in code fragment.*** The original array should contain: 1, 3, 5, 6, 7, 8, 2, 4, 6. Your code fragment should call the method and print out the length of the longest sequence. You must use the variables below.

int[] arrayWithSequence = {1, 3, 5, 6, 7, 9, 2, 4, 6};

int result = longestSequence(arrayWithSequence);

System.out.println(result);

7. (10 points; 2 points for a) and b); 6 points for c))

The AtomicInteger class is another class in the Java API. It is similar to Integer, in that it stores a single integer value. The method names tell you what the operation does and in what order. For example, if an AtomicInteger object contained 3 and we called getAndIncrement(), it would return 3 (the get part) and then increment the stored value to 4. If instead we called IncrementAndGet(), it would increment 3 to 4, and then return the value 4. This is equivalent to the difference between x++ and ++x, if x is an int.

The UML below describes the AtomicInteger class in Java. The data in the class is not shown.

AtomicInteger
+get():int +addAndGet(delta: int): int +getAndAdd(delta: int): int +getAndIncrement(): int +getAndSet(newValue:int):int +incrementAndGet(): int +set(newValue: int): void +toString(): String +AtomicInteger(value: int) +AtomicInteger()

a) List one accessor method in the AtomicInteger class.

b) List one mutator method in the AtomicInteger class.

c) Use an AtomicInteger object to control a loop that adds 1 to every object in an ArrayList<Integer> with reference days.

```
ArrayList<Integer> days; // this is constructed and its contents set elsewhere

for(AtomicInteger index = new AtomicInteger(); index.get() < days.size(); index.IncrementAndGet())
{
    int value = days.get(index.get());
    days.set(index.get(), value+1);
}
```

8. (30 points; 10 points for a), 10 points for b), and 10 points for c)) Write three methods for the program as described below.

Santa Claus is a legendary philanthropist. He keeps a list of children who have been naughty and nice all year. On Christmas Eve, he rides around the world to every house and gives gifts to children who have been nice and lumps of coal to children who have been naughty.

I've been working with Santa to help him computerize his operation. Santa has a list of naughty acts. When a child's act is reported, Santa checks to see if the act is on his list of naughty acts. If it is, he adds the child's name to the naughty list. If it is not, he adds the child's name to the nice list. On Christmas Eve, he goes through one child at a time and compares the number of times each child's name appears on the naughty versus the nice list. If the child's name is on the nice list at least as many times as the naughty list, the child is nice, otherwise the child is naughty. Santa has guaranteed me that every child has a unique name.

Suppose the naughty acts are: tease sister, tease brother, tease dog, tease cat, tease pet, shout, scream

A program run is shown below: (the italicized comments on the side were added to explain the program). User input is in bold.

input for part a)

```
First we will collect information about how children behaved.
Enter the child's name or Quit to stop
Miranda
Enter the act the child did
feed dog                                    // Miranda is on nice list
Enter the child's name or Quit to stop
Horatio
Enter the act the child did
scream                                      // Horatio is on naughty list
Enter the child's name or Quit to stop
Miranda
Enter the act the child did
set table                                   // Miranda is on nice list 2x
Enter the child's name or Quit to stop
Miranda
Enter the act the child did
tease dog                                   // Miranda is on naughty list
Enter the child's name or Quit to stop
Horatio
Enter the act the child did
scream                                      // Horatio is on naughty list 2x
Enter the child's name or Quit to stop
quit
```

input for part b)

```
We can now tell you who is naughty and who is nice
Enter the name of a child or quit to stop
Miranda
Miranda is nice                             // On nice list 2x, naughty 1x
Enter the name of a child or quit to stop
Horatio
Horatio is naughty                          // Naughty list 2x, nice 0x
Enter the name of a child or quit to stop
quit
```

a) Write the method below.

public static void compileNaughtyAndNiceLists(ArrayList<String> naughtyChildren,
ArrayList<String> niceChildren, Scanner input)

This method reads in the list of naughty acts (stored in "naughtyActs.txt") using the method readFromFile(), with signature below, that was written by someone else. Do not write the readFromFile() method. The compileNaughtyAndNiceLists method then asks the user to repeatedly enter children's names and the acts they do until the sentinel quit is entered. If the act is on the list of naughty acts, the child's name is added to the ArrayList of naughtyChildren. If the act is not on the list of naughty acts, the child's name is added to the ArrayList of niceChildren. A child may be on both lists many times.

public static ArrayList<String> readFromFile(String fileName) // this method is written by someone else

public static void compileNaughtyAndNiceLists(ArrayList<String> naughtyChildren,
ArrayList<String niceChildren, Scanner input)

```
public static void compileNaughtyAndNiceLists(ArrayList<String>
naughtyChildren, ArrayList<String> niceChildren, Scanner input)
throws FileNotFoundException{
      ArrayList<String> naughtyActs = readFromFile("naughtyActs.txt");

      System.out.println("First we will collect information about how
children behaved.");
      System.out.println("Enter the child's name or Quit to stop");
      String name = input.nextLine();

      while (!name.equalsIgnoreCase("Quit"))
      {
         System.out.println("Enter the act the child did");
         String act = input.nextLine();

         if (naughtyActs.contains(act))
         {
            naughtyChildren.add(name);
         }
         else
         {
            niceChildren.add(name);
         }

         // Priming read
         System.out.println("Enter the child's name or Quit to stop");
         name = input.nextLine();
      }
   }
```

b) Write the method below.

public static void findFate(ArrayList<String> naughtyChildren, ArrayList<String> niceChildren, Scanner input)

This method takes the enormous lists of which children commit naughty and nice acts during the year and compares the number of times the child's name (entered by the user) is on the ArrayList of naughtyChildren versus the ArrayList of niceChildren. If the nice is on the nice list at least as many times as the naughty list, the child is declared nice. Otherwise the child is declared naughty.

public static void findFate(ArrayList<String> naughtyChildren, ArrayList<String> niceChildren, Scanner input)

```
public static void findFate(ArrayList<String> naughtyChildren,
ArrayList<String> niceChildren, Scanner input)
{
   // go through naughty list first

   System.out.println("We can now tell you who is naughty and "
      + "who is nice");
   System.out.println("Enter the name of a child or quit to stop");
   String name = input.nextLine();

   while(!name.equalsIgnoreCase("Quit"))
   {
      int naughty = Collections.frequency(naughtyChildren, name);
         // could also be a loop
      int nice = Collections.frequency(niceChildren, name);
         // could also be a loop

      if (naughty <= nice)
      {
         System.out.println(name + " is nice");
      }
      else
      {
         System.out.println(name + " is naughty");
      }

      System.out.println("Enter the name of a child or quit to stop");
      name = input.nextLine();

   }
}
```

c) Write the main program. The signature of the methods you wrote in a) and b) are listed below.

public static void findFate(ArrayList<String> naughtyChildren, ArrayList<String> niceChildren, Scanner input)

public static void compileNaughtyAndNiceLists(ArrayList<String> naughtyChildren, ArrayList<String niceChildren, Scanner input)

```
public static void main(String[] args) throws FileNotFoundException {

   ArrayList<String> naughtyChildren = new ArrayList<String>();
   ArrayList<String> niceChildren = new ArrayList<String>();

   Scanner input = new Scanner(System.in);

   compileNaughtyAndNiceLists(naughtyChildren, niceChildren, input);

   findFate(naughtyChildren, niceChildren, input);

}
```

Do not write anything on the back of any page