Midterm 3 Solution

CS 1323/1324, Fall 2018

Name (printed legibly): Key_____

Student number: _____

Integrity Pledge

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

Signature: _____

Do not write anything on the back of any page. If you need extra space use the blank page at the back of the examination. If you pull the test apart, please put all of the pages in order when you hand your exam in.

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

Pay attention to whether the program requires you to use an ArrayList or an array.

Pay careful attention to what is requested: a code fragment (a few lines of code), a method, or a program.

Since you do not have the whole API available during the examination, it is acceptable to guess at method names, parameters and return values. If your guesses are reasonable—even if not perfect—you will receive credit. If, however, you make up new methods that are not reasonable for the class or that magically solve problems the class cannot solve, you will not get credit.

You do not need to import packages or throw exceptions in any methods. You may use S.o.p as an abbreviation for System.out.println() and may abbreviate any user prompt as "prompt."

Relevant parts of the API are given in a separate document.

1. (15 points; 3 points each for a) and b); 9 points for c))

a) What does the String object initial contain after the following code is executed?

```
String initial = "DAT";
initial = initial.replace('A', 'a');  // API Page 1
```

DaT

b) What does the StringBuilder object initial contain after the following code is executed?

```
StringBuilder initial = new StringBuilder("DAT");
initial.replace(1,3, "MATZ"); // API Page 1
```

DMATZ

c) Write a method that returns the number of times that a given String appears in an oversize array of Strings, without considering case (i.e. 'A' and 'a' should be considered the same). If the oversize array contained {"Fee", "Fi", "Fo", "Fum", "fum"}, the array's size 5, and "FUM" were sought, the method would return 2. The same array with array size fee, and "fum" being sought would return 0.

The signature is below:

```
public static int frequencyWithoutCase (String[] array, int arraySize, String target)
{
        int count = 0;
        for (int index = 0; index < arraySize; ++index)
        {
                if (array[index].equalsIgnoreCase(target))
                {
                        ++count;
                }
        }
        return count;
}
```

2. (15 points; 5 points for algorithm, 10 for sorting values) Sort the values below using either insertion sort or selection sort. You must indicate which algorithm you used and show the process exactly as it was in class***, with each individual swap or move of data shown on a separate line***. **Only data that are moved or changed should be shown.**

I used (circle one): selection sort insertion sort

Selection sort

5	4	2	0	3	1	Aux
0			5			
	1				4	
		2				
			3	5		
				4	5	

Insertion sort

5	4	2	0	3	1	Aux
						4
	5					
4						
						2
		5				
	4					
2						
						0
			5			
		4				
	2					
0						
						3
				5		
			4			
		3				
						1
					5	
				4		
			3			
		2				
	1					

3. (15 points) Trace the code fragment below in the table below. Only show changes to the data.

```
int[] array  = {4, 6, 1, 7, 8, 2, 5};
int high = array.length;
int split = array[high-1];
int start = high-1;
int index;
int temp;
for (index = 0; index < high-1; ++index)
{
      if (array[index] > split)
      {
            if (high-1 != index)
            {
                  --high;
                  temp = array[high-1];
                  array[high-1] = array[index];
                  array[index] = temp;
            }
      }
}
--index;
temp = array[index];
array[index] = array[start];
array[start] = temp;
```

array [0]	array [1]	array [2]	array [3]	array [4]	array [5]	array [6]	high	split	start	index	temp
4	6	1	7	8	2	5	7	5	6	0	
	2				6		6			1	2
										2	
			8	7			5			3	8
										4	
			5			8				3	8

4. (15 points; 4 points each for a) and b), 3 points for c) 2 points each for d), and e)) Use the documentation for the AlphaComposite class (given in a separate document) to answer the following questions and perform the following tasks below. AlphaComposites are used to combine transparency (this is what an alpha value encodes) in a variety of ways (specified by the rules) to build images.

    a) Construct an AlphaComposite object with reference alphaComposite, using the rule which allows the destination to be left untouched, and alpha value of 0.5. Use the AlphaComposite constructor.

    AlphaComposite alphaComposite = new AlphaComposite(AlphaComposite.DST, 0.5);

    b) Construct an AlphaComposite object with reference alpha by creating an AlphaComposite object that is similar to the one stored at reference beta, but using the rule that the color and alpha of the destination should be cleared.

    AlphaComposite alpha = beta.derive(AlphaComposite.CLEAR);

    c) Construct a new AlphaComposite object that follows the rule that says that the part of the destination lying inside of the source replaces the destination.

    AlphaComposite obj = AlphaComposite.getInstance(AlphaComposite.SRC);

    d) Are AlphaComposite objects mutable or immutable?

    immutable

    e) Give the import statement that must be used to work with this class.

    import java.awt.AlphaComposite;

5. (40 points; 10 points for a) , 15 points for b), 15 points for c)) Write the program below as described. You may not change method signatures and must follow all constraints.

Finding a time when a group of people can meet is a challenging problem. If you send out email, some people will tell you when they can meet and others will tell you when they can't, which makes it hard to find common times. Also, the volume of possibilities can be overwhelming, especially for large groups of people with complicated schedules. One solution to this problem is to write a program that a group of people can use to perform scheduling. Lots and lots of apps are available for this, with doodle.com being one I frequently use. We will write a simplified version of this program.

We will only track dates and not times. We will also assume that users enter their data perfectly (except case problems with responses like "QUIT" and "Yes"). We will also assume that everyone must be available on the final chosen date, or a date cannot be chosen.

The program will first ask one person for the days that they can meet. These potential dates will be stored as an ArrayList. Then each other person will be asked in turn for the days they can meet. After each person's dates are collected, their available dates will be combined with previous people's available dates to determine dates when everyone who has participated so far can meet. If there are no times remaining when everyone can meet, the program should stop. When everyone has contributed, the list of possible dates will be shown. If there are no possible dates when everyone can meet, then a warning message must be shown.

The interaction of the program is shown below. There are two runs. In the first run, the users couldn't find an acceptable time. In the second run they could.

Here is the first run of the program. The underlined data was entered by the user.

```
Please provide a list of dates this person is available on a single line. Dates
should be written as month/day/year, separated by spaces, and end with QUIT
11/19/18 11/20/18 11/21/18 11/26/18 11/27/18 11/28/18 QUIT
Is another person attending the meeting? Yes/No
YES
Please provide a list of dates this person is available on a single line. Dates
should be written as month/day/year, separated by spaces, and end with QUIT
11/21/18 11/20/18 11/19/18 quit
Is another person attending the meeting? Yes/No
yes
Please provide a list of dates this person is available on a single line. Dates
should be written as month/day/year, separated by spaces, and end with QUIT
11/26/18 11/27/18 Quit
There are no dates when this meeting can be scheduled
```

Another run of the program is shown below.

```
Please provide a list of dates this person is available on a single line. Dates
should be written as month/day/year, separated by spaces, and end with QUIT
11/19/18 11/20/18 11/21/18 11/26/18 11/27/18 11/28/18 QUIT
Is another person attending the meeting? Yes/No
YES
```

```
Please provide a list of dates this person is available on a single line. Dates
should be written as month/day/year, separated by spaces, and end with QUIT
11/19/18 11/20/18 11/21/18 quit
Is another person attending the meeting? Yes/No
yes
Please provide a list of dates this person is available on a single line. Dates
should be written as month/day/year, separated by spaces, and end with QUIT
11/19/18 11/26/18 11/27/18 Quit
Is another person attending the meeting? Yes/No
No
The following dates are available [11/19/18]
```

The program will be written with three methods (the descriptions below are repeated on the next pages).

public static ArrayList<String> enterDates(Scanner keyboard) will allow the user to enter dates at the keyboard and store them in an ArrayList<String> object. The dates are entered on a single line, separated by spaces and end with QUIT.

public static ArrayList<String> findSharedTimes(ArrayList<String> shared, ArrayList<String> nextPerson) will take two ArrayList<String> objects as parameters: shared and nextPerson. The shared object will contain the dates that are available for everyone so far (not including the last person). The nextPerson object will contain the dates that are available for the last person to participate. Neither shared nor nextPerson should be modified in this method. The method will return an ArrayList<String> that contains the dates that are available for everyone who has participated so far. This method takes no input from the keyboard.

The third method is the main method. The main method takes input from the keyboard.

Do not write any code on this page.

a) Implement the method below. This method will allow the user to enter dates at the keyboard and store them in an ArrayList<String> object. The dates are entered on a single line, separated by spaces and end with QUIT. Example input is shown below.

11/19/18 11/20/18 11/21/18 11/26/18 11/27/18 11/28/18 QUIT

public static ArrayList<String> enterDates(Scanner keyboard)

```
public static ArrayList<String> enterInitialDates(Scanner keyboard)
{
	ArrayList<String> list = new ArrayList<String>();

	System.out.println("Please provide a list of dates this person is available on
a single line. Dates should be written as month/day/year, separated by spaces, and
end with QUIT");
	String input = keyboard.next();

	while (!input.equalsIgnoreCase("QUIT"))
	{
		list.add(input);
		input = keyboard.next();
	}

	keyboard.nextLine(); // remove end of line

	return list;
}
```

b) Implement the method below. This method will take two ArrayList<String> objects as parameters: shared and nextPerson. The shared object contains the dates that are available for everyone so far (not including the last person). The nextPerson object will contain the dates that are available for the last person to participate. Neither shared nor nextPerson should be modified in this method. The method will return an ArrayList<String> that contains the dates that are available for everyone who has participated so far. This method takes no input from the keyboard.

For example: If shared contained [11/19/18, 11/20/18, 11/21/18, 11/26/18, 11/27/18, 11/28/18] and nextPerson contained [11/30/18, 11/20/18, 11/19/18, 11/10/18], the returned ArrayList would contain [11/19/18, 11/20/18].

public static ArrayList<String> findSharedTimes(ArrayList<String> shared, ArrayList<String> nextPerson)

```
public static ArrayList<String> findSharedTimes (ArrayList<String> first,
ArrayList<String> second)
{
	ArrayList<String> result = new ArrayList<String>();
	for (int firstIndex = 0; firstIndex < first.size(); ++firstIndex)
	{
		boolean found = false;
		for (int secondIndex = 0; secondIndex < second.size(); ++secondIndex)
		{
			if (first.get(firstIndex).equals(second.get(secondIndex)))
			{
				found = true;
			}
		}

		if (found)
		{
			result.add(first.get(firstIndex));
		}
	}

	return result;
}
```

This method can be implemented very efficiently using the removeAll method in the ArrayList class.

c) Implement the main method. You must call the methods you wrote in a) and b) above. The signatures of these methods are below.

public static ArrayList<String> findSharedTimes(ArrayList<String> shared, ArrayList<String> nextPerson)

public static ArrayList<String> enterDates(Scanner keyboard)

```
public static void main(String[] args)
{
	Scanner keyboard = new Scanner(System.in);

	ArrayList<String> commonDates = enterInitialDates(keyboard);

	System.out.println("Is another person attending the meeting? Yes/No");
	String answer = keyboard.nextLine();

	while(answer.equalsIgnoreCase("Yes"))
	{
		ArrayList<String> secondDates = enterInitialDates(keyboard);

		commonDates = findSharedTimes(commonDates, secondDates);

		if (commonDates.isEmpty())
		{
			answer = "No"; // to break out of the loop
		}
		else
		{
			System.out.println("Is another person attending the meeting?
Yes/No");
			answer = keyboard.nextLine();
		}
	}

	if (commonDates.isEmpty())
	{
		System.out.println("There are no dates when this meeting can be
scheduled");
	}
	else
	{
		System.out.println("The following dates are available " + commonDates);
	}
}
```