Final Examination

CS 1324, Fall 2018

Name (printed legibly):

Student number:

<u>Do not write on the back of any pages. Do not tear out or add pages.</u>

Integrity Pledge

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

Signature:

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 and signed the integrity statement above.***

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.

You may abbreviate any prompt as "prompt" and use S.o.p to mean System.out.println.

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

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

i) The price of an entrée on a restaurant menu, stored in cents.

ii) The number of appetizers on a menu at a restaurant.

iii) The name of a restaurant.

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.

9 + 5 / 2.3 * (12 % 4)

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

c) What will the following code print out?

```
int value = 5;
if (value >= 7 || value <= 1)
{
   value = 3;
}
if (value > 5 && value < 12)
{
   value = 9;
}
else
{
   value = 2;
}
System.out.println(value);
```

d) 41 % 6 is:

2. (7 points; 5 points for the table and 2 points for the answer in the box) Trace the code below in the table below until the program finishes successfully, throws an Exception or enters an infinite loop. If the program enters an infinite loop, trace three iterations.

```
int[] first = {1, 5, 6};
int[] second = {2, 3, 4};

int[] result = new int[first.length + second.length];

int resultIndex = 0;
int firstIndex = 0;
int secondIndex = 0;
while (resultIndex < result.length)
{
   if (firstIndex < first.length && first[firstIndex] < second[secondIndex])
   {
      result[resultIndex] = first[firstIndex];
      ++firstIndex;
      ++resultIndex;
   }
   else if(secondIndex < second.length)
   {
      result[resultIndex] = second[secondIndex];
      ++resultIndex;
      ++secondIndex;
   }
}
```

firstIndex	**secondIndex**	**resultIndex**	**result**						
			[0]	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**

Circle ***one choice*** below to describe how the program ended (2 points):

The program terminated successfully

The program threw an exception

The program entered an infinite loop

3. (10 points} Find the values stored in first and second at the end of the program. You may use the memory diagram below if you wish to, but are not required to. ***<u>You must fill out the solid box.</u>***

```
public class Test {

   public static void main(String[] args) {

      String[] first = {"A", "S", "D"};
      String[] second = {"L", "K", "J"};

      swapValues(first, second);
   }

   public static void swapValues(String[] source1,
            String[] source2){
         String[] temp = source1;
         source1 = source2;
         source2 = temp;
   }
}
```

At the end of the main program:
The contents of array first are:
The contents of array second are:

At the end of the swapValues method:
The contents of array temp are:
The contents of array source1 are:
The contents of arrav source2 are:

<u>OPTIONAL</u>

main stack frame

Identifier	Address	Contents
	100	
	101	
	102	

swapValues 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. (10 points; 7 points for a) (2 points for algorithm name, 5 points for trace), 3 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

3	**1**	**5**	**2**	**0**	**4**	**Auxiliary**

b) Show the order in which data in the array below would be probed when the binary search algorithm is used to find that the value 35.

23	35	46	47	58	59	63	75	82	87	91	99	105	124	137

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

The method will return whether or not the perfect sized array of int first contains any of the values that are in the perfect size array of int second.

For example, if first contained {1, 3, 5, 7} and second contained {1, 6}, it would return true because the value 1 in second is also in first. If, however, second contained {2, 4, 6} it would return false because none of the values in second are in first.

public static boolean has Any(int[] first, int[] second)

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

The signature for the method in problem 5 is below:

public static boolean hasAny(int[] first, int[] second)

a) Show the method from ***problem 5 (on the previous page) being called in code fragment.*** Your code should print out "found" if the array {1, 3, 5, 7, 9} has any elements in common with the array {2, 4, 6, 9}, and nothing otherwise.

b) ***Write the signature*** (return type, method name, parameters) for a method that takes **two oversize arrays of int** and returns a new **perfect size array of int** that contains the elements that are in both arrays. So if the first array contains {1, 3, 5, 7} and has size 2 and the second contains {2, 3, 4, 5} and has size 4, the array {3} should be returned. 5 is not included in the returned array because the first array had size 2. ***Do not write the method.***

7. (10 points; 1 point for a) and b); 3 points for c), 5 points for d)) The Character class is a wrapper class in Java. Use the methods from the UML below (full descriptions in the API) to perform the following tasks.

Character
+CURRENCY_SYMBOL: char +LINE_SEPARATOR: char
+Character(value: char) +compare(x: char, y:char): int +compareTo(anotherCharacter: Character): int +isIdeographic(c: char) : boolean +isWhitespace(c: char) : boolean

a) Construct a Character object with reference myChar that represents the character 'A'.

b) Print the current symbol of currency to the console.

c) Use an instance method from the Character class to print out "Before" if 'A' comes before 'a', or "After" if it does not.

d) Write a method with the signature below that determines if all characters in the perfect sized array of characters are ideographic (this means being represented by pictures that represent whole words instead of parts of words). Ideographic characters come from the Chinese, Japanese, Korean and Vietnamese languages.
static boolean areAllIdeoGraphic(char[] ch)

8. (30 points; 10 points each part) Write three methods for the program as described below.

Habiticus produces an app that keeps track of things you need to do repeatedly and habitually to be healthy and happy. For example, you should drink 3 8oz glasses of water a day to be sufficiently hydrated (OK, so it's really 8, but I don't want the output to be too voluminous). Their app stores both the number of times a task should occur each day, and the name of the task in a file between daily runs. The format of the file is given below (times first and then the task).

8 Drink water
3 Walk
3 Eat
1 Sleep

Every day, the app lets you enter more habitual tasks or to perform tasks from the daily list. The interface is shown below. The menu is shown the first time and then abbreviated as Menu. User input is underlined and bolded. The method that wrote the input is given in a comment on the right hand side.

```
Choose from the menu:
1. Add a daily task.
2. Complete a task.
3. Show the list of remaining tasks.
4. Quit
3                    // main program
Tasks remaining: [Drink water, Drink water, Drink water, Walk, Walk, Walk,
Eat, Eat, Eat, Sleep]
Menu
2                    // main program
Enter the task
Drink water          // completeTask method
Menu
3                    // main program
Tasks remaining: [Drink water, Drink water, Walk, Walk, Walk, Eat, Eat, Eat,
Sleep]
Menu
1                    // main program
Enter the task
Walk dogs            // addTask method
Enter the number of times the task needs to be done daily
2                    // addTask method
Menu
3                    // main program
Tasks remaining: [Drink water, Drink water, Walk, Walk, Walk, Eat, Eat, Eat,
Sleep, Walk dogs, Walk dogs]
Menu
4                    // main program
```

After the program is complete, the file contains:

```
3 Drink water
3 Walk
3 Eat
1 Sleep
2 Walk dogs
```

The methods used are described below (ones with an asterisk are those you will implement):

```
public static void addToFile(String fileName, ArrayList<String> tasks)
```

This method appends the list of tasks to the existing file with that name. We do not know how to implement this method (yet), so it was written by someone else.

```
*public static ArrayList<String> readFile(String fileName)
```

This method opens the file with the given fileName, and reads in the input one line at a time (times first, then task) as long as there is still data left. It then puts times copies of task in an ArrayList<String> object that is returned.

```
*public static void addTask(ArrayList<String> dailyTasks, ArrayList<String>
newTasks, Scanner keyboard)
```

This method gets the task and the number of times the task needs to be performed from the user. If the task is already in dailyTasks, it prints out an error message to the user warning them that the task is already in the dailyTasks ArrayList. Otherwise, it adds the task to the dailyTasks ArrayList the given number of times. It also adds both times and the task to the newTasks ArrayList as a single String (e.g. "2 Walk dogs"), so this ArrayList can be used later by addToFile to store newly added tasks between daily program runs.

```
public static void completeTask(ArrayList<String> dailyTasks, Scanner
keyboard)
```

This method asks the user for the name of the completed task. If that task is not in the dailyTasks ArrayList, it returns an error message. If the task is in the list, it removes one copy of the task from the dailyTasks ArrayList. This method was also written by someone else.

```
*main program
```

This method reads the file that contains the list of daily tasks. It also creates an empty ArrayList of new tasks, in case the user enters new tasks that need to be added to the file later. It then repeatedly displays a menu and calls the other methods in the class, based on user responses. At the end, it writes a file containing the existing daily tasks (both the ones that were previously in the file, as well as any the user added).

a) (10 points) Implement the method below

```
public static ArrayList<String> readFile(String fileName)
```

This method opens the file with the given fileName, and reads in the input one line at a time (times first, then task) as long as there is still data left. It then puts times copies of task in an ArrayList<String> that is returned.

Hint: If you don't remember how to read from a file, just read from the console to get the most points.

b) (10 points) Implement the method below.

```
public static void addTask(ArrayList<String> dailyTasks,
   ArrayList<String> newTasks, Scanner keyboard)
```

This method gets the task and the number of times the task needs to be performed from the user. If the task is already in dailyTasks, it prints out an error message to the user warning them that the task is already in the dailyTasks ArrayList. Otherwise, it adds the task to the dailyTasks ArrayList the given number of times. It also adds both times and the task to the newTasks ArrayList as a single String (e.g. "2 Walk dogs"), so this ArrayList can be used later by addToFile to store newly added tasks between daily program runs.

c) (10 points) Implement the main program. The signature of the methods that you may call are given below (explanations of the methods were given in the problem statement).

```
public static void addToFile(String fileName, ArrayList<String> tasks)
public static ArrayList<String> readFile(String fileName)
public static void addTask(ArrayList<String> dailyTasks,
      ArrayList<String> newTasks, Scanner keyboard)
public static void completeTask(ArrayList<String> dailyTasks,
      Scanner keyboard)
```