Final Examination

CS 1323 Spring 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 name of the company that made my bicycle.

ii) Whether or not I have my panniers (side bags) on my bicycle.

iii) The number of miles I road my bicycle today, including partial miles.

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

3.2 * 4 + 6 % 9 / 5

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

c) What value does the variable value have at the end of execution of this code, you may use the table if you wish, but the answer in the box below will be graded.

```
int number = 9;
int bBound = 3;
int hBound = 9;
if (number % 2 == 1)
    number = number + 1;
if (number <= bBound || number >= hBound)
    number = bBound + hBound / 2;
else
    number = hBound;
```

number

number contains:

d) 35 / 12 is:

Do not write anything on the back of any page

2. (6 points) Trace the code below in the table below. ***<u>Show values only when they are changed.</u>***

```
int[] data = {9, 4, 2, 3};

for (int outer = 0; outer < data.length; ++outer)
{
   int count = 0;
   for (int inner = outer; inner < data.length; ++inner)
   {
      if (data[outer] >= data[inner])
         ++count;
   }
   data[outer] = count;
}
```

outer	data[outer]	count	inner	data[inner]	data array			
					[0]	[1]	[2]	[3]

Do not write anything on the back of any page

3. (6 points) The program below is supposed to find the median value in the array.

The median value is one that has the same number of array elements that are larger and smaller. For example, if an array contains: {5, 3, 1, 2, 4}, then 3 is the median because there are two elements that are larger (4 and 5) and two elements that are smaller (1 and 2).

The programmer even left you lots of comments! Unfortunately, the method is not correct.

Find what is printed out by this program either by reading the code, or by tracing it with the memory diagram below.

```
import java.util.Arrays;
public class Problem2 {
   public static void main(String[] args){
      int[] sizes = {6, 9, 2, 1, 4};
      int result = median(sizes);
      System.out.println(result);
   }

   public static int median(int[] data){
      int[] copy = Arrays.copyOf(data, data.length);  // create copy of array
      Arrays.sort(data);                              // sort the array
      int midpoint = data.length/2;                   // find the middle
      data = copy;                                    // restore the array
      return data[midpoint];                          // return middle
   }
}
```

This program prints out:

OPTIONAL

main stack frame

Identifier	Address	Contents
	100	
	101	
	102	

median 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.

5	4	6	2	3	1	Auxiliary

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 22 (which is in the array)

4	9	15	22	24	27	33	42	51	60	73	89	92	95	99

Do not write anything on the back of any page

5. (10 points) Write a method called removeCount as described below.

```
public static int removeCount(String[] data, int dataSize, String value,
                              int times)
```

Remove up to times occurrences of value from the oversized array data with dataSize elements.
Parameters:

`data` - An array. The relative order of elements in the array should not be changed.
`value` - A value to be removed from the array up to count times.
`times` – The maximum number of elements that should be removed.

Returns:

The number of elements that remain in array data after up to times occurrences of value have been removed.

For example: if the original array contained: [“a”, “e”, “f”, “a”, “b”, “g”], value was “c”, and count was 1 the array would be unchanged and 6 would be returned. If value was “a”, and count was 3, the array would contain [“e”, “f”, “b”, “g”] and 4 would be returned.

public static int removeCount(String[] data, int dataSize, String value, int times)

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

The signature for the method in problem 5 is below:

int removeCount(String[] data, int dataSize, String value, int times)

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

b) Show the method from ***problem 5 (on the previous page) being called in code fragment.*** The original array should be 100 elements long contain [“a”, “c”, “a”, ”d”] and the value “a” should be removed up to 3 times. You must use the variables below.

String[] original =

int originalSize =

String removeMe =

int repeats =

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

A StringIntMap object is used to keep track of key-value pairs of data, e.g. (“Raven”, 2), (“Jazz”, 1), (“Maggie”, 3). The first element (the String) is called the key. The second element (the int) is called the value. Repeating keys is not allowed. So if we try to add (“Raven”, 3) into the StringIntMap, the operation does nothing and returns false.

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

StringIntMap
+addPair(newKey: String, int value): boolean +clear(): void +equals(otherElements: StringIntMap): boolean +isEmpty(): boolean +getValue(String key): int +getKey(int value) : String +getAllKeys(): String[] +removePair(key: String): void +replace(key:String, int value): void +size(): int +StringIntMap()

a) List one accessor methods in the StringIntMap class.

b) List one mutator method in the StringIntMap class.

c) Suppose a StringIntMap called results has been declared, constructed, and contains an unknown number of key-value pairs. ***Print out the contents of the key-value pairs stored in the StringIntMap as (key, value), one to a line.*** There is no toString() method in this class.

8. (30 points; 5 points for a), 15 points for b), and 10 points for c)) Write three methods for the program as described below.

Red Apron is a meal delivery service that sends recipes and matching box of groceries to your home every week. Since this kind of delivery is quite expensive (express shipping, ice packs, insulation, etc.), the meals themselves have to be high quality. This means using fresh, seasonal produce to justify the cost. Unfortunately, the list of fresh, seasonal produce changes every week. So they need a program to help them identify what recipes they can offer.

Red Apron has a list of common ingredients they can get at any time (e.g. vinegar, butter, soy sauce, proteins, etc.), a list of ingredients that they expect customers to have (e.g. olive oil, salt, pepper, egg) and a list of produce that is available from their suppliers at this time (e.g. asparagus, arugula, onions). They also can order other ingredients. They also have a huge group of recipes, stored as a list of ingredients (no quantities are included).

The list of common ingredients is stored in a file called “common.txt”. The list of produce that is available this week is stored in a file called “produce.txt”. The list of ingredients their customers are expected to have called “customer.txt”. The recipes are stored in a file with the dish name, so the recipe for chicken with goat cheese is stored in “chickenWithGoatCheese.txt”. Each file contains ingredients, one to a line.

Your program should read in a recipe selected by the user, and provide a list of ingredients that is available from the common ingredients, a list of produce that is currently available, a list of ingredients that their customers are expected to have and a list of ingredients that would have to be found. Suppose the files contain the items below (the files actually contain one grocery item to a line and no commas, but showing them that way takes too much space).

produce.txt: onion, Shallot, arugula, asparagus, Bell pepper
common.txt: vinegar, butter, panko, Salmon, chicken breast, pork chop, Chicken thigh
customer.txt: olive oil, salt, Pepper, egg
chickenAndGoatCheese.txt: chicken breast, Arugula, asparagus, shallot, goat cheese, vinegar, SALT, pepper, olive oil

A program run is shown below:

```
Enter the file name for the recipe or QUIT to exit
chickenAndGoatCheese.txt
We can get the following produce
arugula, asparagus, shallot
We have the following common ingredients
chicken breast, vinegar
Customers have the following ingredients
olive oil, pepper, salt
We do not have the following ingredients
goat cheese
Enter the file name for the recipe or QUIT to exit
quit
```

a) Write the method below. This method takes the name of a file, and puts the contents of the file, one line at a time, in the ArrayList<String> object. The contents of the ArrayList<String> should be ***all lower case***, to avoid problems with case (capital and small letters) later. The contents of the ArrayList<String> should be left in ***sorted order***.

If the file contained:

```
ASPARAGUS
Arugula
shallot
Onion
```

The ArrayList<String> should contain {“arugula”, “asparagus”, “onion”, “shallot”}.

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

b) Write the method below. The method takes an ArrayList<String> that contains ingredients that are still needed for the recipe, and another ArrayList<String> that contains ingredients that Red Apron has. Both ArrayList<String> are all in lower case and in sorted order. Ingredients that are found are removed from need and put in an ArrayList<String> that is returned from the method. ***You must use binary search.***

For example: At the start of the method:
have contains {“egg”, “olive oil”, “pepper”, “salt”}
need contains: {“arugula”, “asparagus”, “olive oil”, “pepper”, “salt”, “shallot”, “vinegar”}
At the end of the method:
need contains: {“arugula”, “asparagus”, “shallot”, “vinegar”}
An ArrayList<String> containing {“olive oil”, “pepper”, “salt”} is returned.
have is not modified by this method.

public static ArrayList<String> ingredientsHaveAndNeed(ArrayList<String> need, ArrayList<String> have)

c) Write the main program. The methods below are available:

public static ArrayList<String> readFileAndSort (String fileName)
public static ArrayList<String> ingredientsHaveAndNeed(ArrayList<String> need, ArrayList<String> have)

File names are: “common.txt”, “produce.txt”, “customer.txt”. The recipe file name is provided by the user.

Do not write anything on the back of any page