Midterm 2 Solution

CS 1323, Fall 2015

Name (printed legibly):

Student number:

Do not write anything on the back of any page. Tests are scanned for grading. Work on the back of the page will be lost.

If you need additional space, use the blank pages at the end of the examination. If you pull your examination apart, put your examination pages in order when you turn them in.

Answer all programming questions in Java.

Pay careful attention to what is requested: a code fragment (a few lines of code), a method, or a program. Students who write a whole program when a code fragment is requested will waste so much time that they may not complete the examination.

Unless otherwise indicated, each part of a problem is worth the same number of points. Show your work to receive partial credit.

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. For example, if you forget that the String class has a length() method and call it size(), that is fine. 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.

Integrity Pledge

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

Signature (initial for Section 995):

1. (15 points; 3 points each) Perform the following operations. If the code is not legal, say so.

Find the logical value (true or false) of each of the statements below. Assume the variables are declared and initialized as follows:

int xPosition = 100;

int leftBoundary = 200;

int rightBoundary = 400;

a) leftBoundary < xPosition && xPosition < rightBoundary

200<100 && 100 < 400

false && true

false

b) leftBoundary < xPosition || rightBoundary > xPosition

200 < 100 || 400 > 100

true

c) leftBoundary > xPosition || xPosition > rightBoundary

200 > 100 || 100 > 400

true

d) ! (leftBoundary <= rightBoundary)

!(200 <= 400)

! true

false

e) leftBoundary >=0 && rightBoundary >= 0 || leftBoundary <=0 && rightBoundary <= 0

true && true || false && false

true || false

true

1. (20 points) Trace the loop below in the table at the right.

|  |  |  |
| --- | --- | --- |
| index | target | sum |
|  |  | 0 |
| 0 | 1 | 1 |
|  | 2 | 2 |
|  | 3 | 3 |
|  | 4 |  |
| 1 | 2 |  |
|  | 3 |  |
|  | 4 |  |
| 2 | 3 |  |
|  | 4 |  |
| 3 | 4 |  |
| 4 |  |  |

int[] data = {9, 2, 2, 2};

int sum = 0;

for (int index=0; index<data.length; ++index)

{

for (int target = index+1; target < data.length; ++target)

{

if (data[index] != data[target])

sum = sum + 1;

}

}

1. (10 points; 5 points each) The paragraphs below describe static methods. Determine the signature of the method **ONLY**. ***Do not write the method(s).***

The signature of a method is the return type, the method name, and the parameters.

Write the signature for a method that takes a dictionary of words, stored in an array of String objects and adds another word to the dictionary.

a) Assume the dictionary array is oversized.

int addWord(String[] dictionary, int dictionarySize, String word)

b) Assume the dictionary array is perfect sized.

String[] dictionary addWord(String[] dictionary, String word)

1. (10 points) Show the contents of the array in the main program when the program finishes execution. You may use a memory diagram to trace the code if you wish to, or may use your knowledge of passing by value and/or reference. Show or explain your work to get partial credit.

**public** **class** TraceMe {

**public** **static** **void** main(String[] args) {

**int**[] array = {5, 4, 3, 2, 1};

*method*(array, 9); // READ THIS CAREFULLY!!!!

}

**public** **static** **int**[] method(**int**[] data, **int** value){

**for** (**int** i=0; i<data.length; ++i)

data[i] = i\*i;

data = **new** **int**[5];

**for** (**int** j=0; j< data.length; ++j)

data[j] = value;

**return** data;

}

}

**At the end of the main method, array contains: 0, 1, 4, 9, 16**

**Explanation: The array is passed by sharing, so the initial changes made to the content of the array in method are reflected in the main program. The array is then reconstructed in method and the reference is returned. This returned reference is not stored anywhere, so the array ends up showing only the changes made at the start of the method.**

The tables below are for your use if you wish to trace the method to determine what is in array. You do not have to use these tables if you do not wish to, although it is one way to gain partial credit.

Main stack frame

|  |  |  |
| --- | --- | --- |
| Identifier | Address | Contents |
| array | 100 | 1000 |
|  | 101 |  |
|  | 102 |  |
|  | 103 |  |
|  | 104 |  |
|  | 105 |  |
|  | 106 |  |

Heap

|  |  |  |
| --- | --- | --- |
| Identifier | Address | Contents |
| 0 | 1000 | ~~5~~ 0 |
| 1 | 1001 | ~~4~~ 1 |
| 2 | 1002 | ~~3~~ 4 |
| 3 | 1003 | ~~2~~ 9 |
| 4 | 1004 | ~~1~~  16 |
| length | 1005 | 5 |
| 0 | 1006 | ~~0~~ 9 |
| 1 | 1007 | ~~0~~ 9 |
| 2 | 1008 | ~~0~~ 9 |
| 3 | 1009 | ~~0~~ 9 |
| 4 | 1010 | ~~0~~ 9 |
| length | 1011 | 5 |
|  | 1012 |  |
|  | 1013 |  |
|  | 1014 |  |
|  | 1015 |  |

Method stack frame

|  |  |  |
| --- | --- | --- |
| Identifier | Address | Contents |
| data | 200 | ~~1000~~ 1006 |
| value | 201 | 9 |
|  | 202 |  |
|  | 203 |  |
|  | 204 |  |
|  | 205 |  |
|  | 206 |  |

1. (15 points)  **Work either part a) or part b) of this problem.** You do not need to work both parts, and will not get extra credit for doing so. You must use the proper algorithm. If you fill in both parts and don’t indicate which you want to count, I will grade part a).

a) Trace ***selection sort*** on the array below. Show each swap in a separate line of the table and show only the data that are moved.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **4** | **5** | **1** | **2** | **3** |
| 1 |  | 4 |  |  |
|  | 2 |  | 5 |  |
|  |  | 3 |  | 4 |
|  |  |  | 4 | 5 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

b) Trace ***insertion sort*** on the array below. Show each assignment of data on a separate line of the table, in the order that the assignment statements are done. Show only the data that are moved.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **4** | **5** | **1** | **2** | **3** | **aux** |
|  |  |  |  |  | 4 |
|  |  |  |  |  | 5 |
|  |  |  |  |  | 1 |
|  |  | 5 |  |  |  |
|  | 4 |  |  |  |  |
| 1 |  |  |  |  |  |
|  |  |  |  |  | 2 |
|  |  |  | 5 |  |  |
|  |  | 4 |  |  |  |
|  | 2 |  |  |  |  |
|  |  |  |  |  | 3 |
|  |  |  |  | 5 |  |
|  |  |  | 4 |  |  |
|  |  | 3 |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

1. (30 points; 10 points each for a), b) and c)) Write a program that generates platitudes, by writing each of the methods on the following pages. Platitudes are “A trite and obvious observation, in particular one that's expressed as if it were fresh and significant,” at least according to <http://grammar.about.com/od/pq/g/Platitude.htm>.

The platitudes will be read in from a file of platitudes named “Quotes.txt”. This file is available in the Project directory. To select a platitude, you will find one that contains a word the user gave or, if no such platitude exists, to choose one randomly from the platitudes in the file.

The file contains no more than 1000 platitudes. The array that contains your quotes should be oversized.

Here is an example of the program running. Words in italics were entered by the user. The comments to the right explain the output and were not generated by the program.

Enter a word, or Quit to exit

*Honesty*

Honesty is the best policy // “Honesty” was found

Enter a word, or Quit to exit

*do*

Crime doesn't pay // “do” is part of “doesn’t”

Enter a word, or Quit to exit

*day*

All's well that ends well // No quotes included “day”

Enter a word, or Quit to exit

*quit*

You must write the methods exactly as they are given. You may not change the methods, parameters, or return type. You must use the methods you write in a) and b) in the main program in c).

Do not write code on this page.

1. public static int readFile(String[] platitudes, String filename)

**public** **static** **int** readFile(String[] array, String fileName)

**throws** FileNotFoundException

{

Scanner file = **new** Scanner(**new** File(fileName));

**int** index = 0;

**while** (file.hasNextLine())

{

array[index] = file.nextLine();

++index;

}

file.close();

**return** index;

}

1. This method should determine whether the String word is contained anywhere in the list of platitudes (given as an oversize array). If the word is in one of the platitudes, that one should be returned. If the word is not in a platitude, a random platitude from the array should be returned.

There is a useful method, called contains, in the String class. This method returns true if the argument is in the String used to call the method, and false otherwise. It is used in code below.

String example = “ABC”;

String test = “AB”;

if (example.contains(test)) System.out.println(test + “ is in “ + example);

The method signature is:

public static String findPlatitude(String[] platitude, int size, String word)

**public** **static** String findPlatitude(String[] array, **int** size, String word)

{

**for** (**int** i=0; i<size; ++i)

{

**if** (array[i].contains(word))

{

**return** array[i];

}

}

**int** rand = (**int**) (Math.*random*()\*size);

**return** array[rand];

}

1. Write the main program using the methods in a) and b).

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

{

**final** **int** MAX\_SIZE = 1000;

String[] platitudes = **new** String[MAX\_SIZE];

**int** numberPlatitudes = 0;

Scanner input = **new** Scanner(System.***in***);

// Read in a list of platitudes from a file

numberPlatitudes = *readFile*(platitudes, "Quotes.txt");

// Print out a platitude that contains a given word or a random

// platitude if no such word exists in the file

System.***out***.println("Enter a word, or Quit to exit");

String word = input.nextLine();

**while** (!word.equalsIgnoreCase("Quit"))

{

String quote = *findPlatitude*(platitudes,

numberPlatitudes, word);

System.***out***.println(quote);

System.***out***.println("Enter a word, or Quit to exit");

word = input.nextLine();

}

}