Name: Key UTA ID:

Instructions:

1. The test is worth 100 points. The point value of each question is given with the question. There are also extra credit questions at the end.
2. The test is open book and open notes for all printed and hand-written material. You may NOT bring an electronic book or any electronic device to use during the test (no computer, no smart phone, etc.) You may use as much printed or written material as desired including copies of code examples.
3. You will write your answers on the test pages. If additional space is needed, you may use the back of the pages. Please make a note on the test page whenever your answer continues onto another page and indicate where the answer is.
4. Please write legibly. Your writing should readable if the test is sitting on a desk in front of me. I am not looking for perfect handwriting but it does need to be legible. I will deduct points if your answers are much more difficult to read than those of the general student.
5. If you have a question during the test, please raise your hand. The TA and I will be available to come hear your question. Sometimes we may not be able to answer your question because it gives you too much information but you should always ask.
6. You have 1 hour and 20 minutes to complete the test.

1. Assume that the user has entered a value called *dayValue* that represents a day of the week, where 1 represents Monday, 2 is for Tuesday, … and 7 is for Sunday, and that you are given the pseudocode below:

String *wkTm*;

String *msg*;

boolean *valid* = true;

If *dayValue* is valid and before Saturday,

Then set *wkTm to* “work day”

Otherwise If *dayValue* is valid and less than or equal to Sunday

set *wkTm* to “weekend”

else set *wkTm* to “week” and *valid* to false

If *dayValue* is Monday or Tuesday,

Set msg to “beginning”

Otherwise if *dayValue* is Wednesday or Thursday,

Set msg to “middle”

Else if *dayValue* is Friday,

Set msg to “end”

If the *dayValue* is Saturday,

Set msg to “first day”

Otherwise if dayValue is Sunday,

set msg to “second day”

else set msg to “not a valid day” and *valid* to false

Finally print “Day “+*dayValue*+” is “+(valid?“the”:””) +*msg*+” of the “+*wkTm*

1.a) Write a set of if, if-else, and nested if-else Java statements to implement the pseudocode above. You only need to write the lines of code to implement from the first if statement above through the print statement. Use the back of the previous page or of this page if more space is needed. {8 points}

The answer below is not the only possible way to write this code. It is an example.

// dayValue is an integer entered by the user

if ((dayValue < 0)||(dayValue > 7))

{

valid = false; COMPOUND

wkTm = “week”;

}

else if (dayValue <= 5)

{

 wkTm = “work day”;

}

else

{

 wkTm = “weekend”;

}

if ((dayValue == 1) || (dayValue == 2))

{

 msg = “beginning”;

}

else if ((dayValue == 3) || (dayValue == 4))

{

 msg = “middle”;

}

else if (dayValue == 5)

{

 msg = “end”;

}

else if (dayValue == 6)

{

 msg = “first day”;

}

else if (dayValue == 7)

{

 msg = “second day”;

}

else

{

 msg = “not a valid day”;

 valid = false;

}

 SINGLE

System.out.println( “Day “+*dayValue*+” is “+(valid?“the”:””) +*msg*+” of the “+*wkTm*);

1.b) Rewrite your nested if-else statements above as a Java switch statement. Only use the nested statements in the switch. Do not rewrite any other part of the Java code. {7 points }

switch (dayValue)

{

 case 1:

 case 2:

 case 3:

 case 4:

 case 5: wkTm = “work day”;

 break;

 case 6:

 case 7: wkTm = “weekend”;

 break;

default: wkTm = “week”;

 valid = false;

 break;

}

OR

switch (dayValue)

{

 case 1:

 case 2: msg = “beginning”;

 break;

 case 3:

 case 4: msg = “middle”;

 break;

 case 5: msg = “end”;

 break;

 case 6: msg = “first day”;

 break;

 case 7: msg = “second day”;

 break;

default: msg = “not a valid day”;

 valid = false;

 break;

}

1.c) In the code you wrote in part 1.a) label any compound statement with the word “COMPOUND” with an arrow or other marking to connect the label with a compound statement in the code. Many possible choices here 1.c and in 1.d {4 points}

1.d) In the code you wrote in part 1.a) label any single simple statement with the word “SINGLE” with an arrow or other marking to connect the label with a single statement in the code. {3 pts}

2. Given the possible variable names below, choose a reasonable data type and write a declaration, with data type to the left, for each of the variable names. Use primitive types wherever possible. Indicate if any name cannot be used and if not, why not. {15 points total}

int *numberOfPeople* ; // no partial people

double *moneyAmount* ; // allows dollars and cents in normal notation $43.02

 *6PackVolume* illegal name because names cannot start with numbers in Java

String *petName* ; // for names

double *yearsInCollege* ; // allows partial years

boolean *isLegal* ; // asks a true/false question with the variable name

 *new Value* illegal name because ***new*** is a keyword in Java

3. We are going to implement a program that leads to the Gaussian summation equation. The program will add up the numbers from 0 to some *n* that is given by a user. The Gaussian summation equation recognizes a property of this problem and defines a short cut to solving this problem. Our program will not use the short cut equation but will use the pattern that Gauss recognized.

 The program you will develop will take in a value from a user and call it *n*. The program then should begin to sum up the numbers by adding 0, our first variable called *a*, and the user’s *n*, which we will save in variable *b*, and then saving that sum as the starting total sum, called *sum*. As long as the value of *a* is less than *b*, thenthe program then should increment *a* and decrement *b,* then add the new *a* and *b* together as an intermediate sum, called *abSum*, and then add *abSum* to the total *sum*. If *a* equals *b*, then add *a* to *sum* and stop. If *a* is greater than *b*, stop.

 The output of the program should show one line of output for each pair of *a* and *b* values. Each line should output *a*, *b*, *abSum*, and *sum*. On the first line of output, *a* should equal 0 and *b* should equal *n*.

3.a) Write the algorithm above in PSEUDOCODE. Lay out the algorithm in sentences with indentation. You should write a short sentence in words or phrases for each line that you think will be a Java statement. You do not need to write pseudocode for the class definition and the main method header but you should define any variables that you use that are not obvious from the program description above. {8 points}

Get a value of *n* from the user

Make sure *n* is greater than 0 // input validation / error checking

Declare integers *a, b, abSum,* and *sum*

Assign 0 to *a* and *n* to *b.* Set *sum* to 0.

Make a while loop that tests for *a* less than or equal to *b* (*a* <= *b*)

 Set *abSum* to 0

 In the loop body add *a* plus *b* and store in *abSum* // not efficient but required by algorithm

 Add *abSum* to *sum*

 Print *a, b, abSum, sum* in a nice format // format not specified

 Increment *a* (*a*++)

 Decrement *b* (*b* --)

// this is the end of the loop

if *a* equals *b* after the loop is done then

 add *a* to *sum*

Print *a, b, abSum, sum* in a nice format // final print not specified but needed to verify correctness

3.b) What data types are you using for your variables and why? {4 pts}

All integers. The input value from the user should be an integer, otherwise “add up the numbers from 0 to some *n*” would not be possible because you can’t add all the real numbers between any two decimal numbers.

3.c) Consider what variable(s) and tests you will use to help control the loop. State what test condition and which variable(s) you would use for this algorithm AND explain why you choose this test condition and variable(s). {6 pts}

The test condition is to test if *a* is less than *b*. These variables start as the endpoints of the range of integers to add and they are incremented and decremented at each step. There are two possible stopping conditions for this process: Either *a* and *b* cross each other, i.e. *a* becomes larger than *b,* or *a* and *b* are the same value.

Since these are the stopping conditions, then the condition to keep the loop going must be when *a* is less than the value of *b*

3.d) If *n* is defined to be 11, write the output that should be produced by the program. {9 pts}

n = 11; a = 0, b = 11; abSum = 0; sum = 0;

first iteration of loop : 0 < 11 is true; Add 0 + 11 = 11 🡺 abSum; abSum + sum = 11; a = 1, b = 10

second iteration : 1 < 10 is true; 1 + 10 = 11 🡺 abSum; abSum + sum = 22; a = 2, b = 9

3rd iteration : 2 < 9 is true; 2 + 9 = 11 🡺 abSum; abSum + sum = 33; a = 3, b = 8

4th iteration : 3 < 8 is true; 3 + 8 = 11 🡺 abSum; abSum + sum = 44; a = 4, b = 7

5th iteration : 4 < 7 is true; 4 + 7 = 11 🡺 abSum; abSum + sum = 55; a = 5, b = 6

6th iteration : 5 < 6 is true; 5 + 6 = 11 🡺 abSum; abSum + sum = 66; a = 6, b = 5

7th iteration : 6 < 5 is false;

sum = 66

3.e) What pattern do you see in the *abSum* column of values? {4 pts}

abSum is always equal to 11 and n is 11

3.f) Write the Java code to implement the algorithm using your pseudocode from answer 3.a. Include the variable declarations then just write the Java loop for the algorithm. Include a scanner to read in a value for *n* in your program and include the output statements needed. You do not need to write the class declaration or the main method header. {15 pts}

 //Declare integers a, b, abSum, and sum

 //Assign 0 to a and n to b. Set sum to 0.

 int a, b, abSum, sum, n;

 a = b = abSum = sum = 0;

 Scanner input = new Scanner(System.in);

 boolean isNotValid = true;

 //Get a value of n from the user

 //Make sure n is greater than 0 // input validation / error checking

 System.out.println("This program will add all the integers from 0 to some value called n ");

 while (isNotValid)

 { System.out.println("Please enter a positive integer for n: ");

 n = input.nextInt();

 if (n >= 0)

 { b = n;

 isNotValid = false;

 }

 else System.out.print("Your value was not valid. ");

 } /\* Make a while loop that tests for a less than or equal to b (a <= b)

 In the loop body add a & b and store in abSum // not efficient - required by algorithm

 Add abSum to sum

 Print a, b, abSum, sum in a nice format // format not specified

 Increment a (a++) and decrement b (b --) // this is the end of the loop \*/

 System.out.println();

 System.out.printf("%8s%8s%8s%8s\n","a","b","abSum", "sum");

 System.out.printf("%8s%8s%8s%8s\n","-","-","-----", "---");

 while (a < b)

 { abSum = a + b;

 sum += abSum;

 System.out.printf("%8d%8d%8d%8d\n",a++,b--,abSum, sum);

 } // if a equals b after the loop is done, then add a to sum

 // Print a, b, abSum, sum in a nice format // not specified but needed to verify correctness

 if (a == b)

 { sum += a;

 }

 System.out.println();

 System.out.printf("%5s%3s%8s%8s%8s\n","Final","a","b","abSum", "sum");

 System.out.printf("%8s%8s%8s%8s\n","-","-","-----", "---");

 System.out.printf("%8d%8d%8d%8d\n",a++,b--,abSum, sum);

4. Use the Java program Test1Spr16Q4.java to answer the following questions:

4.a) Is the variable *judgeCount* in the *scoring* method the same as the variable *judgeCount* in the main method? Explain why or why not. {3 points}

The variables are not the same. judgeCount in the scoring method is declared in the scope of the method and is not visible to the main method.

4.b) Why does the *if* statement in *scoring* test the value of *judgeCount- judgeNum +1*? In other words, explain the purpose of the entire *if* statement, when it will execute, and what it will do. {4 points}

The if statement checks to see if the current judge is the first judge and if so, it prints the message about the format of the input. This is inefficient because the same thing could be done by putting the message above the for loop.

4.c) Is there a more efficient way to do the task you just described in 4.b? Answer Yes or No and defend your answer. {3 points}

Yes, because the same thing could be done by putting the message above the for loop.

4.d) If there are 2 judges and the scores that are entered are 80 65 and 92 60, then what will *scoring* return? . {7 points}

80 +92 = 172; 172 / 2 = 86; 86 \* .6 = 51.6

65 + 60 = 125; 125 /2 = 62.5; 62.5 \* .4 = 25

 51.6 + 25 = 76.6

 86 62.5

 \* .6 \* .4

 ---- -----

 51.6 25.00

Extra Credit

XC1) Describe in your own words what the method *scoring* in problem 4 does. {2 points}

It calculates a score for some sort of project in a given area

XC2) How many times will the repetition structure execute inside *scoring* and how do you determine this? {3 points}

It executes judgeNum times because it counts down from judgeNum to 0

XC3) Give an example of an *import* statement that you have used in your programs and describe what importing something means for your Java program. {3 points}

import java.util.Scanner;

importing means finding the definition in a Java library and bringing it into your program.

XC4) Write a two-line poem about learning to program so far. {Any answer will receive 2 points}