Name: Key notes explain how 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. Read all of the instructions for each question and answer what is asked. Do not write down random stuff if you don’t know the answer.
3. All questions that have the same question number are related to each other. However, they are not all necessarily dependent on each other so you can skip around if needed.
4. 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.
5. 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.
6. Please write legibly and large enough to read easily. 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.
7. For multiple choice questions, circle the letter of your answer choice. Circle only one choice for your answer.
8. 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.
9. You have 1 hour and 20 minutes to complete the test.

1. Use the code Spr17Test1QA.java to answer the parts of question 1 below**. Read through the WHOLE program before starting to answer the questions**. Short versions of the questions are also embedded in the code to assist you but be sure to answer the question as it is written on this test not just what is in the code.

1.a) What would be better, more meaningful names for the variables below and why would you choose these types of meaningful names? (12 points)

|  |  |  |
| --- | --- | --- |
| Current name | Better name | Because |
| String moth; | month | This is name of month |
| String a; | dayOfWeek | This is the day of the week |
| int ear; | year | This is the year number |
| int aBer = 0; | dayNumber | This is the day of the month |
| int count = 0; | datesCount | The count of dates user entered |
| int mthct = 0; | monthCount | The number of the month 1 - 12 |

In the explanations below, I’m going to use the original names that are in the code but your answers could have your new variable names or the original names.

1.b) What does *count* tell you at the end of the program? (5 pts)

The number of dates entered by the user.

How do you know? : In the code, *count* is inside the control structure of \_\_\_( !\_\_\_\_). This structure has to be a selection (if, if-else…) or a repetition (while, for…). *count*  increments at the end of the structure and the first time it increments, it goes to 1 (because it starts at 0). If the structure was a selection, then *count* goes to 1, the structure is done and it ran 1 time. If the structure is a repetition, the *count* increments by 1 each time the loop runs and when it is done, it will have counted how many times the loop structure ran. [This is an acceptable answer.] The final question is “what is happening inside the structure?” and basically the structure takes in inputs and prints out messages based on the input month. Therefore, *count* is counting how many times the program took in inputs [also an acceptable answer] and, since the inputs are the dates the user entered, *count* is the number of dates the user entered.

1.c) Why is the output statement below repeated in the program? Explain in words why it is needed. (8 pts)

System.out.println("Give an input in the form of "

+ "\"Tuesday, February 21, 2017\".\n "

+ "If finished, enter \"Done\" : ");

It is first used before the loop to get the initial user input. Then is it used at the end of the loop to read each subsequent input from the user.

How do we figure this out? : We start by looking at the purpose of the statement itself. The statement tells the user the format of data to enter into the program. Therefore it most likely precedes an input statement which we can see with *a=input.next;* . The input statement uses *.next* which reads one word from the input. Since *a=input.next* is the first input statement after the message, then it reads the first word which will be either the day of the week OR the word *Done*. Given the way the output statement is worded, we can see that if the user enters *Done* we need to catch that word and end the program. In order to catch this word, we will need to test right after we read the word in. You can see that the next line of code is the control structure of \_\_\_( !\_\_\_\_). As mentioned before, this is a selection or a repetition, either of which will have a test condition. Since we have determined we need to test for the string *“Done”*, then this will need to be our test condition here. After we test, if the word was NOT *“Done”* then we read the remaining inputs.

When we get to the bottom of the structure, (assuming it is a repetition), then we need to let the user know that they should enter another input date or the word “Done”. Therefore, we need to repeat the output statement again so that the user knows what should be done when our program is ready for more input.

1.d) Which Java control statement would you choose to use at *d.* and why? (7 pts)

while – the user is entering input until they enter the word “Done” and we do not know how many times the loop will run. The while loop is the indefinite loop.

How do we decide and why? : At d. we clearly need either a selection or a repetition statement. If we choose a selection structure (if, if-else, switch,…) then the whole structure is only executed one time. If we are only going to execute this one time, then why do we ask the user to enter more data (as discussed in the last question)? Instead, asking for more data implies that we should repeat, i.e. have a loop using a repetition control structure.

So, if we need a repetition structure, then we should choose between while, do-while, and for. By looking at the code structure, we can eliminate do-while right away because the test is at the top not the bottom. To choose between a *while* loop and a *for* loop then we should think about what the code appears to do. It appears to take in input, look at the month value, then, based on the month value, print an output statement corresponding to the month value and save a value in *mthct*, then ask the user for another input, read the first word, and increment *count*.

If we want to use a *for* loop, then generally we are counting something and we have a specific stopping point for the count. As an example, if we have told the user to enter five dates, then we could count up to five dates and then stop the loop. However, we told the user to enter dates OR enter the word “Done”. This means that we are not counting, we are testing until a condition becomes true (i.e. the user has entered “Done”). If we are waiting for a condition to become true that is not a counted condition, then a *while* loop is the best choice. Remember that a *for* loop is called a definite loop because we are counting up (or down) to a definite value and we know how many times the loop is supposed to run. On the other hand, a *while* or *do-while* loop is an indefinite loop because we only know that the loop runs until the condition is true, i.e. in this case, until the input word is “Done”. This also is tied to the value of *count* because this counter is the only way to find out how many times the loop actually ran.

1.e) For the control statement at *e.*,

\_\_\_\_\_**(\_!\_\_\_\_\_\_\_\_)** // \_\_*d*.\_\_( !\_\_*e*.\_\_) after *a = input.next();*

what test condition do you need and why is that the correct test condition? Include the negation (!) as part of your test condition. Example: to write a test condition that is true when some variable b is greater than 10, you would write !*(b <= 10)*  (7 pts)

(a.equalsIgnoreCase(“Done”)) How do we determine this? : In the last problem we figured out that we were going to keep running until the user entered “Done” as the first word (or only word) of their input. Once we know that, then we just need to write the test.

The first impulse is to write *(a != “Done”)* or *( ! (a == “Done”)).* While this is intuitively correct for the logic, it won’t work. The *!=* and *==* commands only work on numbers. Therefore to test a string, we need to use string methods that test for equality. This means we need *(!a.equalsIgnoreCase(“Done”))* in order to most safely test to see if the user is ready to stop. We could use *(!a.equals(“Done”))* but this will fail if the user enters “done” or “DONE” or other combinations. Therefore the *equalsIgnoreCase* is the preferred method to test with.

1.f) Add the missing action inside the May case so that it is like the April and June cases. List the missing statement(s) here. Then describe what this action accomplishes in the program.

(7 pts)

mthct = 5;

This sets the month number variable to 5 based on the value of “May”

How do you figure it out? : You start by looking “inside” the cases for April and June and comparing that to the case for May. You are looking for something that is in the April and June cases that is not in the May case. So in April you see:

System.out.println…

mthct = …

break;

and in June you see:

System.out.println…

mthct = …

break;

but in May you only see:

System.out.println…

break;

Therefore, the missing line is *mthct =* something. The next thing is to compare the *mthct* lines from April and June:

mthct = 4; // from April and

mthct = 6; // from June

With April ~ 4 and June ~ 6 we now need to figure out what the numbers mean. Hopefully, since we were working with other numbers for the date, we can figure out that April is the 4th month and June is the 6th month. Therefore, May is the 5th month and the missing line is *mthct = 5;*

2. List and describe the three types of control structures in structured programming and give an example Java control statement for each type of structure. (12 pts)

Sequential control – statements follow one after another

Assignment

Method calls

Selection control – statements select from one or more options based on a test being true

If / if else/ nested if else

switch

Repetition control – group of statements are executed repeatedly until a test becomes false

while

do while

for

Many people did not DESCRIBE the types of control structures and therefore lost some points.

3. In addition to having information about the date, most computers also have a value for time. We are going to create our own time value. Our time value will include units as large as a day and as small as a minute. Let’s assume that we need to take two time values in the form of :

*dd hh mm* // Assume that *dd* is a day value between 1 and 31,

// *hh* is an hour value between 0 and 23, and

// *mm* is a minute value between 0 and 59

as our inputs and then calculate the amount of hours and fractions of hours between the two times. We’ll assume that our two days are in the same month and that we are given the two time values such that the earliest value is always given first. Our hours will be in military time so 8:20am will be 08 20 and 8:20pm will be 20 20. For this problem, you do not need to consider any error checking of the values because we will assume they are all valid.

3.a) Given this information, write an algorithm using pseudocode that will take in the two time values and correctly calculate the number of hours from the first time to the second time. The resulting value should be a floating point answer. Ex. the number of hours from 02 10 30 to 03 11 45 should be 25.25 because there are 25 and a quarter hours between the two times. Do not make any other assumptions than the ones given in the problem. (15 pts)

Assume values are dd1, hh1, mm1, dd2, hh2, and mm2

First, find the number of full days (days) by *dd2 – dd1 = days*

Next take *hh2 – hh1 = hours* to get the hours difference between the first and last day

Then take *mm2 – mm1 = minutes* to get minutes difference

Note that it doesn’t matter if days, hours, and minutes are negative or positive (with our assumptions above.) When you add them, the negatives get taken care of.

Now to convert the entire answer to hours you need to multiply the days by 24 and divide the minutes by 60 then you add these converted values with the hours value to get the total in hours.

The final answer is (days \* 24) + hours + (minutes/60)

03 11 45

02 10 30

01 01 15m = (1 \* 24) + 1+ (15/60) = 25 + (1/4) = 25.25

03 08 15

02 10 30

01 -2 -15m = (1 \* 24)+ -2+ (-15/60) = 22 – (1/4) = 22 - .25 = 21.75

The problem did not specifically say that data had to be read from a user. It just says the algorithm takes in the two time values. Therefore, pseudocode written for reading inputs was not a significant part of the problem.

Pseudocode generally means writing the information in such a way that the reader can understand all the steps of the algorithm without having to understand any specific kind of programming language or coding. However, it should also be at a detailed enough level that if you handed it to someone who knows how to write code, that they could interpret your algorithm clearly and in detail so that they could implement the algorithm in code.

3.b) Write the Java code needed in the *main* method to implement your algorithm. You do not need to write anything that would be before the *main* routine (no imports needed). (12 pts)

public static void main (String[] args)

{

// Assume values are dd1, hh1, mm1, dd2, hh2, and mm2

int dd1 = 0; int hh1 = 0; int mm1 = 0;

int dd2 = 0; int hh2 = 0; int mm2 = 0;

sout

“The first input day and time are Day “+dd1+”, Hour “+hh1+”, and Minute “+mm1

“The second input day and time are Day “+dd2+”, Hour “+hh2+”, and Minute “+mm2

int days = 0;

int hours = 0;

int minutes = 0;

double totalHours = 0;

// First, find the number of full days (days) by dd2 – dd1 = days

days = dd2 – dd1;

// Next take hh2 – hh1 = hours to get the hours difference between the first and last day

hours = hh2 – hh1;

// Then take mm2 – mm1 = minutes to get minutes difference

minutes = mm2 – mm1;

// The final answer is (days \* 24) + hours + (minutes/60)

totalHours = (days \* 24) + hours + (minutes/60.0);

sout

“The total hours between the two day and times are “+totalHours

3.c) If we were rewriting this code as a method separate from *main*, what values from your code would be needed as parameters to the method? Give the data types and the names. (4 pts)

int dd1;

int hh1;

int mm1;

int dd2;

int hh2;

int mm2;

Whatever values you are using for your days, hours, and minutes for the two times.

3.d) If we were rewriting this code from 3.b. as a method, what return type would you use and why? (4 pts)

The return type should be double so that the hours and fractions of hours can be returned.

4. Rewrite the code fragment below to use a for loop instead of a do-while loop (7 pts)

int c2 = count; //initialization // assume that count is defined previously in the program

do

{

System.out.println("Thank you for playing");

c2--; //increment (decrement)

}

while (c2 >= 1); // test

// assume that count is defined previously in the program

// for ( initialization ; test ; increment )

for ( int c2 = count; c2 >= 1 ; c2-- )

{

System.out.println("Thank you for playing");

}

Extra Credit

XC1) Using the code *Spr17Test1Q1.java* again, write the missing case statements for April and June below. List each case label and describe where that case should be put in the switch.

{4 points}

case "april":

case "APRIL": // both go before case "April":

case "june":

case "JUNE": // both go before case "June":

How do you figure this out? : What are the case statement labels for May? “May”, “MAY”, and “may”. Therefore if we have “April” currently, then we need “APRIL” and “april”. We don’t need “aPril” or those like that because we don’t have “mAy”.

XC2) Following the format for the switch statement in Spr17Test1QA.java and above, if all the missing cases and actions were filled in, how many “case” statements would there be in the switch and why?

{4 points}

3 cases per month, 12 months total plus one default case = 37 cases

Since the previous question filled in April and June, ALL must refer to all of the months in total.

XC3) List one way the Java has already made you a better person. Be amusing. ☺

{Any answer will receive 2 pts}