CSE143X Midterm, Fall 2020

This is a closed-book/closed-note exam. There is a "cheat sheet" at the end. You are not allowed to access the internet or other sources during the exam.

You are allowed to abbreviate "Always", "Never," and "Sometimes" as "A", "N", and "S" for the assertions question.

You are NOT to use any electronic devices while taking the test, including calculators.

Give yourself 75 minutes to complete the exam and then scan it (preferably as a pdf) and upload it to the course web page.

1. Expressions, 10 points. For each expression in the left-hand column, indicate its value in the right-hand column. Be sure to list a constant of appropriate type (e.g., 7.0 rather than 7 for a double, Strings in quotes).

Expression	Value
`3 * (5 - 2) - 3 - 2 * 2`	_____
`4 * 7 % 8 + 132 % 10 + 3 % 4`	_____
`27 / 5 / 2 + 3.4 * 2 - 1.1 * 2`	_____
`9 + 9 + "9 + 9" + 9 + 9`	_____
`19 / 2 / 2.0 + 2.5 * 6 / 2 + 0.5 * 4`	_____

2. Parameter Mystery, 12 points. Consider the following program.

```
public class ParameterMystery {
    public static void main(String[] args) {
        String literal = "8";
        String brace = "semi";
        String paren = brace;
        String semi = "brace";
        String java = "42";

        param(java, brace, semi);
        param(literal, paren, java);
        param(brace, semi, "literal");
        param("cse", literal + 4, "1");
    }

    public static void param(String semi, String java, String brace) {
        System.out.println(java + " missing " + brace + " and a " + semi);
    }
}
```

List below the output produced by this program.

3. If/Else Simulation, 12 points. Consider the following method.

```
public static void ifElseMystery(int a, int b) {
    if (a == b || b > 2) {
        a++;
        b = b + 3;
    } else {
        b++;
    }
    if (a < b && a % 2 == 1) {
        a++;
        b = a - 2;
    } else if (b % 2 == 0) {
        a = a - 2;
        b = a + 3;
    }
    System.out.println(a + " " + b);
}
```

For each call below, indicate what output is produced.

Method Call	Output Produced
ifElseMystery(2, 2);	_____
ifElseMystery(3, 1);	_____
ifElseMystery(4, 0);	_____
ifElseMystery(5, 3);	_____
ifElseMystery(1, 2);	_____
ifElseMystery(7, 4);	_____

4. While Loop Simulation, 12 points. Consider the following method:

```
public static void mystery(int z) {
    int x = 1;
    int y = 1;
    while (z > 1) {
        x++;
        y = y * 2;
        z = z / 2;
    }
    System.out.println(x + " " + y);
}
```

For each call below, indicate what output is produced.

Method Call	Output Produced
mystery(1);	_____
mystery(5);	_____
mystery(10);	_____
mystery(42);	_____

5. Assertions, 15 points. You will identify various assertions as being either always true, never true or sometimes true/sometimes false at various points in program execution. The comments in the method below indicate the points of interest.

```
public static void mystery(Scanner console) {
    int x = 3;
    int y = 0;
    // Point A
    while (x > 0) {
        // Point B
        x = console.nextInt();
        y++;
        if (x > 0) {
            x = -x;
            // Point C
        }
        // Point D
        x = -x;
    }
    // Point E
    System.out.println("y = " + y);
}
```

Fill in the table below with the words ALWAYS, NEVER or SOMETIMES.

	x > 0	x == 0	y == 0
Point A			
Point B			
Point C			
Point D			
Point E			

6. Programming, 10 points. Write a static method called spinWheel that takes a Random object and an integer n as parameters and that simulates the spinning of a wheel until the number 20 comes up n times in a row. On the wheel are the numbers 20, 30, 40, 50, and 60 and each number should be equally likely to come up when the wheel is spun. Your method should report the individual spins as well as indicating how many times it takes to get n occurrences of 20 in a row. For example, below are two sample calls:

```
Random r = new Random();
spinWheel(r, 2);
spinWheel(r, 3);
```

The first call should produce two lines of output like this:

```
spins: 40, 40, 50, 20, 50, 50, 40, 20, 30, 40, 50, 20, 20
2 in a row after 13 spins
```

The second call should produce two lines of output like this:

```
spins: 50, 50, 50, 20, 40, 20, 40, 20, 20, 20
3 in a row after 10 spins
```

Notice that the spin values are separated by commas and that the method stops when it has seen n occurrences of the value 20 in a row. You are to exactly reproduce the format of these logs, although the specific numbers will differ because of the use of a Random object. You may assume that the value n passed to your method is greater than or equal to 1.

7. File Processing, 10 points. Write a static method called printFigure that takes an input scanner as a parameter and that prints a figure using the data from the scanner. The data in the scanner is line-based. Each line has a sequence of count/text pairs. These pairs are used to produce a complete line of output. For example, this input line:

```
4 * 3 <> 5 (..)
```

indicates 4 occurrences of an asterisk followed by 3 occurrences of "<>" followed by 5 occurrences of "(..)", which would produce this output:

```
****<><><>(..)(..)(..)(..)(..)
```

The text to print for a given pair will always be a single token. This causes a problem for printing spaces because a space won't be recognized as a token. The special token "space" is used to indicate an actual space. For example, if a scanner called input contains the following data:

```
1 + 6 =* 1 +
1 | 2 space 1 /\ 4 . 1 /\ 2 space 1 |
1 | 1 space 2 /\ 2 . 2 /\ 1 space 1 |
1 | 6 /\ 1 |
1 + 6 =* 1 +
```

and we make the following call:

```
printFigure(input);
```

then the method would produce the following output:

```
+=*=*=*=*=*=*+
|  /\..../\  |
| /\/\../\/\ |
|/\/\/\/\/\/\|
+=*=*=*=*=*=*+
```

Notice that the token "space" indicates actual spaces in the output. A blank line in the input should produce a blank line of output. You may assume that the input is legal and that none of the counts is negative. You may not construct any extra data structures to solve this problem.

8. Arrays, 10 points. Write a static method called numUnique that takes a sorted array of integers as a parameter and that returns the number of unique values in the array. The array is guaranteed to be in sorted order, which means that duplicates will be grouped together. For example, if a variable called "list" stores the following values:

```
[5, 7, 7, 7, 8, 22, 22, 23, 31, 35, 35, 40, 40, 40, 41]
```

then the following call:

```
numUnique(list)
```

should return 9 because this list has 9 unique values (5, 7, 8, 22, 23, 31, 35, 40 and 41). It is possible that the list might not have any duplicates. For example if list instead stored this sequence of values:

```
[1, 2, 11, 17, 19, 20, 23, 24, 25, 26, 31, 34, 37, 40, 41]
```

then a call on the method would return 15 because this list contains 15 different values.

If passed an empty list, your method should return 0. Remember that you can assume that the values in the array appear in sorted (nondecreasing) order.

You may not construct any extra data structures to solve this problem and your method should not alter the array passed as a parameter.

9. Programming, 9 points. Write a static method called numWords that takes a String as a parameter and that returns the number of words in the String. By definition, words are separated by one or more spaces. The table below shows several sample calls and the value that should be returned.

Method Call	Value Returned
numWords("how many words here?")	4
numWords("to be or not to be, that is the question")	10
numWords(" how about merry-go-round ")	3
numWords(" !&$%--$$!!*() foo_bar_baz ")	2
numWords("x")	1
numWords(" ")	0
numWords("")	0

Notice that words can contain punctuation marks. Any non-empty sequence of non-space characters can be a word. Also notice that there might be spaces at the beginning or end of the String.

You may not construct any other objects to solve this problem (e.g., you can't use a Scanner or tokenizer) and you are limited to the String methods listed on the cheat sheet. You may assume that the string has no other whitespace characters such as tabs or newline characters. Your method can pay attention just to spaces to decide how many words there are.

CSE 143X Midterm Cheat Sheet

Syntax templates

```
for (initialization; test; update) {
    statement(s);
}

if (test) {
    statement(s);
} else {
    statement(s);
}

if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else {
    statement(s);
}

while (condition) {
    statement(s);
}
```

```
public static void name(parameters) {
    statement(s);
}

public static type name(parameters) {
    statement(s);
    ...
    return expression;
}

for (int i = 0; i < array.length; i++) {
    do something with array[i];
    ...
}

for (int i = 0; i < string.length(); i++)
{
    do something with string.charAt(i);
    ...
}
```

Math Method	Description
`Math.abs(`*value*`)`	absolute value
`Math.min(`*v1*`, `*v2*`)`	smaller of two values
`Math.max(`*v1*`, `*v2*`)`	larger of two values
`Math.round(`*value*`)`	nearest whole number
`Math.pow(`*b*`, `*e*`)`	b to the e power

Random Method	Description
`nextInt(`*max*`)`	random integer from 0 to *max*-1

Construction Examples

```
int[] data = new int[10];
Random r = new Random();
Scanner console = new Scanner(System.in);
```

String Method	Description
`contains(`***str***`)`	`true` if this string contains the other's characters inside it
`endsWith(`***str***`)`, `startsWith(`***str***`)`	`true` if this string starts/ends with the other's characters
`equals(`***str***`)`	`true` if this string is the same as *str*
`equalsIgnoreCase(`***str***`)`	`true` if this string is the same as *str*, ignoring capitalization
`indexOf(`***str***`)`	index in this string where given string begins (-1 if not found)
`length()`	number of characters in this string
`substring(`***i***`, `***j***`)`	characters in this string from index *i* (inclusive) to *j* (exclusive)
`toLowerCase()`, `toUpperCase()`	a new string with all lowercase or uppercase letters
`charAt(`***i***`)`	returns char at index *i*

Scanner Method	Description
`nextInt(), hasNextInt()`	read/return token as `int` and test if reading will succeed
`next(), hasNext()`	read/return token as `String` and test if reading will succeed
`nextDouble(), hasNextDouble()`	read/return token as `double` and test if reading will succeed
`nextLine(), hasNextLine()`	read/return line as `String` and test if reading will succeed

Operator	Description
<	less than
<=	less than or equal
>	greater than
>=	greater or equal
==	equal
!=	not equal

Operator	Description
&&	and
||	or
!	not