CSE 142 Sample Midterm #5

1. Expressions

For each expression at left, indicate its value in the right column. List a value of appropriate type and capitalization. e.g., `7` for an `int`, `7.0` for a `double`, `"hello"` for a `String`, `true` or `false` for a `boolean`.

Expression	Value
`3 * -1 + 7 - 5 / 2`	_____
`2 + 2 + "(2 + 2)" + 2 + (2 + 2)`	_____
`13 / 3 - 27 / 5 * 0.5 + (7.5 - 6)`	_____
`2 % 11 % 2 + 11 % 2 + 2`	_____
`(5 / 3 == 1 && 10 < 4 + 5) != false`	_____

2. Parameter Mystery

At the bottom of the page, write the output produced by the following program, as it would appear on the console.

```
public class ParameterMystery {
   public static void main(String[] args) {
      String eggman = "googoo";
      String walrus = "eggman";
      String googoo = "me";
      String me = "he";
      String he = "walrus";
      String lucy = "in the sky";

      magicalMystery(he, me, lucy);
      magicalMystery(walrus, eggman, googoo);
      magicalMystery("eggman", eggman, walrus);
      magicalMystery(lucy, "we " + he, "googoo");
   }

   public static void magicalMystery(String we, String he, String me) {
      System.out.println("I am " + he + " as you are " + me + " as you are " + we);
   }
}
```

3. If/Else Simulation

For each call below to the following method, write the output that is produced, as it would appear on the console:

```
public static void ifElseMystery(int a, int b) {
    int c = 2;
    if (a + c < b) {
        c = c + 8;
    } else {
        b = b + 10;
    }
    if (a + c < b) {
        c = c + 8;
    } else {
        b = b + 10;
    }
    System.out.println(b + " " + c);
}
```

Method Call	Output
`ifElseMystery(7, 17);`	_____
`ifElseMystery(12, 5);`	_____
`ifElseMystery(16, 8);`	_____

4. While Loop Simulation

For each call below to the following method, write the output that is produced, as it would appear on the console:

```
public static void whileMystery(int n) {
    int x = 1;
    int y = 1;
    while (n > y) {
        x++;
        y = 10 * y + x;
    }
    System.out.println(x + " " + y);
}
```

Method Call	Output
`whileMystery(0);`	_____
`whileMystery(7);`	_____
`whileMystery(32);`	_____
`whileMystery(256);`	_____

5. Assertions

For each of the five points labeled by comments, identify each of the assertions in the table below as either being *always* true, *never* true, or *sometimes* true / sometimes false. (You may abbreviate them as A, N, or S.)

```
public static void mystery(int x, int y) {
    int z = 0;

    // Point A
    while (x < y) {
        // Point B
        z++;
        if (z % 2 == 0) {
            x = x * 2;
            // Point C
        } else {
            y--;
            // Point D
        }
    }

    // Point E
    System.out.println(z);
}
```

	x < y	z == 0	z % 2 == 0
Point A			
Point B			
Point C			
Point D			
Point E			

6. Programming

Write a static method named `yardSale` that accepts two parameters, a console Scanner and an initial amount of money, and returns the amount of money the user ends up with after a series of purchases. Your method should prompt the user for a series of potential purchases specified by a unit price per single item and the quantity of that item until the user's available amount of money is less than $5. The user buys everything that doesn't exceed their current amount of money and that is considered a "good deal" (unit price is under $10). A message should be printed when such a purchase is made. In addition, your method should print the total quantity of items purchased and the total cost of the most expensive item purchased. Results do not need to be rounded after the decimal place. You may assume the user will always type valid input.

For example, the following call

```
Scanner console = new Scanner(System.in);
double remainingBudget = yardSale(console, 50.75);
```

would generate an interaction like this (user input underlined and bold)

```
Price? 8.75
Quantity? 2
What a deal! I'll buy it.
Remaining money: $33.25

Price? 11.50
Quantity? 3
Remaining money: $33.25

Price? 5.10
Quantity? 5
What a deal! I'll buy it.
Remaining money: $7.75

Price? 6.95
Quantity? 2
Remaining money: $7.75

Price? 0.75
Quantity? 8
What a deal! I'll buy it.
Remaining money: $1.75

Total quantities purchased: 15
Most expensive purchase: $25.5
```

You must exactly reproduce the format of this sample execution. Notice that the second item is not purchased because it is not a good deal (unit price is not under $10) and the fourth item is not purchased because the total cost of $13.90 exceeds the amount of money the user has left ($7.75). The most expensive purchase is based on the total cost (5 items at $5.10) rather than the unit cost (2 items at $8.75).

This sample call would return 1.75 as the remaining money because the user spent a total of $49.00 (17.50 + 25.50 + 6.00) and started with $50.75.

6. Programming (Writing Space)

7. Programming

Write a static method named `randomPattern` that takes an integer `size` as a parameter and that prints a square where each line has a series of 1 or more "\" characters followed by 0 or more "/" characters, with each slash separated by a "-" character. The square should contain `size` lines and `size` total backslash/slash characters on each line. Your method should construct a `Random` object that it uses to choose the number of "\"s to print on each line (a number between 1 and size inclusive). You may assume that the parameter passed to your method is at least 2.

The following table lists some calls to your method and one of their possible expected outputs. Keep in mind that since the number of "\" characters on each line is random, your output may not match exactly.

Call	`randomPattern(5);`	`randomPattern(3);`	`randomPattern(2);`	`randomPattern(10);`
Example Output	\-\-\-\-/ \-\-\-\-\ \-\-/-/-/ \-\-/-/-/ \-\-\-/-/	\-/-/ \-/-/ \-\-/	\-/ \-\	\-\-\-/-/-/-/-/-/ \-/-/-/-/-/-/-/-/-/ \-\-\-\-/-/-/-/-/-/ \-\-\-\-\-/-/-/-/-/ \-\-\-\-\-\-/-/-/-/ \-\-\-/-/-/-/-/-/ \-\-\-/-/-/-/-/-/ \-\-\-\-\-\-\-/-/-/ \-\-\-\-\-/-/-/-/-/ \-\-\-\-\-\-\-\-/-/-/

8. Programming

Write a static method called `weave` that that takes two integers as parameters and that returns the result of weaving their digits together to form a single integer. Two numbers x and y are weaved together as follows. The last pair of digits in the result should be the last digit of x followed by the last digit of y. The second-to-the-last pair of digits in the result should be the second-to-the-last digit of x followed by the second-to-the-last digit of y. And so on.

For example, consider weaving 128 with 394. The last pair of digits in the result should be 84 (because the original numbers end in 8 and 4). The second-to-the-last pair of digits in the result should be 29 (because the second-to-the-last digits of the original numbers are 2 and 9). The third-to-the-last pair of digits in the result should be 13 (because the third-to-the-last digits of the original numbers are 1 and 3). Thus:

```
weave(128, 394)
```

should return 132984. Notice that the order of the arguments is important. The call `weave(394, 128)` would return 319248.

If one of the numbers has more digits than the other, you should imagine that leading zeros are used to make the numbers have equal length. For example, `weave(2384, 12)` should return 20308142 (as if it were a call on `weave(2384, 0012)`). Similarly, `weave(9, 318)` should return 30198 (as if it were a call on `weave(009, 318)`).

You may assume that the numbers passed to weave are non-negative. You may not use Strings to solve this problem; you must solve it using integer arithmetic.