****

PAPER COMPILING EXERCISES: Solution key with explanation

****

DIRECTIONS:

1) Compute by HAND the output of each program.
2) Compare your work with a peer BEFORE verifying with a compiler
3) Review your work and learn from any mistakes

****

WhatsMyValue1

****

```
public class WhatsMyValue1 {

    public static void main(String[] args) {
        int saturn = 100;
        int mars = 20;
        int comet = saturn / mars;
        mars = comet * 10;
        saturn = mars % 10;
        System.out.println("Saturn: " + saturn);
        System.out.println("Mars: " + mars);
        System.out.println("Comet: " + comet);
    } // close main
} // close class
```

Output:

```
run:
Saturn: 0
Mars: 50
Comet: 5
BUILD SUCCESSFUL (total time: 0
```

Solution Notes:

The % is called the modulous opeator. You can think of it like the “remainder” operator: it divides the left operand with the right operand and returns the integer value of the reminder after integer division. Examples:

4%2 = 0 [2 divides evenly into 4 with zero remainder. 4/2 = 2 with nothing left over.]

4%3 = 1 [3 does not divide evenly into 4. 1 is left over after 4/3]

It’s really handy to use the % modulus operator to determine if a number is even or odd. If the number is X, and we take the modulo 2 of it like X%2, we’ll get 0 if X is even, and 1 if X is odd. This is very handy for implementing many data-related algorithms.

```
*********************************************************************************
                        WhatsMyValue2
*********************************************************************************
public class WhatsMyValue2 {
    public static void main(String[] args) {
        boolean tomato = true;
        boolean ginger = false;
        int oregeno = 50;
        int fenugreek = 2;
        if(tomato){
            oregeno = oregeno / fenugreek;
        } else {
            oregeno = fenugreek;
        }
        if(ginger && tomato){
            oregeno = oregeno * fenugreek;
        }
        ginger = !ginger;
        System.out.println("Tomato: " + tomato);
        System.out.println("Ginger: " + ginger);
        System.out.println("Oregeno: " + oregeno);
        System.out.println("Fenugreek: " + fenugreek);

    } // close main
} // close class
```

Output:

```
run:
Tomato: true
Ginger: true
Oregeno: 25
Fenugreek: 2
BUILD SUCCESSFUL (total time: 0
```

Solution Notes:

If Control logic

The line `if(tomato) { }` Uses the value of a plain old boolean variable to control the if() block. Any code inside the { and } of the if block is executed if the value inside the if's() is true. At its most simple form, a boolean type variable can be used to control this logic.

Compound boolean operators: && (and) and || (or)

The && and || operators take two operands (left and right). Each is evaluated to its `true` or `false` value. And expression with && evaluates to `true` if and only if both the left and the right operand evaluate to `true`.

The || (or) operator takes to operands (left and right). Each side is evaluated to its `true`/`false` value. If either one of the two operands OR both are `true`, the entire expression evaluates to `true`.

Negation Operator!

We can negate a boolean type variable with the exclamation point. This is a unary operator meaning it only acts on a single value. It "flips" `true` values to `false`, and `false` values to true.Examples:

```
!true evaluates to false
!false evaluates to true
boolean t = true;
System.out.println(!t)  // this will spit out false

*********************************************************************************
                        WhatsMyValue3
*********************************************************************************
public class WhatsMyValue3 {
    public static void main(String[] args) {
        int turtle = 4;
        int marmot = 2;
        boolean sloth = false;

        while(marmot <= turtle){
            sloth = !sloth;
            marmot = marmot + 1;
        } // end while
        if(sloth){
            marmot = marmot * marmot;
        }
        System.out.println("Turtle: " + turtle);
        System.out.println("Marmot: " + marmot);
        System.out.println("Sloth: " + sloth);
    } // close main
} // close class
```

Output:

```
run:
Turtle: 4
Marmot: 25
Sloth: true
BUILD SUCCESSFUL (total time: 0
```

Solution Notes:

Negation Operator!

We can negate a `boolean` type variable with the exclamation point. This is a unary operator meaning it only acts on a single value. It "flips" `true` values to `false`, and `false` values to `true`.

Examples:

```
!true evaluates to false
!false evaluates to true
boolean t = true;
System.out.println(!t)  // this will spit out false
```

While Loops:

Note that the while loop will run as long as the expression `marmot <= turtle` evaluates to `true`. In this case, when the while is first encountered, `marmot` is 2 and `turtle` is 4. The `while` loop will run three times exactly since 1 is added to `marmot` each cycle. **When we jump after the while once** `marmot <= turtle` **becomes** `false`**,** `marmot` **is 5 (not 4!!). Work it out by hand, carefully!**

Remember, `marmot` will be incremented by one during the last pass through the `while` loop EVEN THOUGH the control expression evaluates to false after `marmot` becomes 5. And after the while loop, we execute `marmot = marmot * marmot`, which comes out to: 5 *5 = 25.

25 which is the value of `marmot` at the `println()`call.

```
**********************************************************************************
                        WhatsMyValue4
**********************************************************************************
public class WhatsMyValue4 {
    public static void main(String[] args) {
        int australia = 10;
        int chile = 5;
        boolean ecuador = false;
        chile = australia + chile;
        System.out.println("Australia before call: " + australia);
        System.out.println("Ecuador before if: " + ecuador);
        if(!ecuador){
            doSimpleMath(australia);
        } // close if
        System.out.println("Chile: " + chile);
        System.out.println("Ecuador after if: " + ecuador);
    } // close main

    public static void doSimpleMath(int numToFlip){
        int result = (numToFlip * -1) + 15;
        System.out.println("Result: " + result);

    } // close doSimpleMath
} // close class
```

Output:

```
Australia before call: 10
Ecuador before if: false
Result: 5
Chile: 15
Ecuador after if: false
BUILD SUCCESSFUL (total time: 0
```

Solution Notes:

Remember the negation ! unary operator just “toggles” true values to false, and false values to true.

When we use `!ecuador` to control the if() block, we are only using the variable in evaluation: we are NOT changing the value of `ecador` with `!ecuador`. So, the value of `ecuador` before and after if is `false`. BUT when we put `!ecuador` into the `if()` block, we get `!false` which evaluates to `true`.

Method calls with parameter inputs

This means our call to `doSimpleMath()` is executed and whatever value is inside the () for this method is passed down into `doSimpleMath's` local variable called `numToFlip`.

When `doSimpleMath` is executed, `australia` was passed in as a parameter. In our case, `australia` holds the value of 10 at the time of the call to `doSimpleMath`. The 10 is transferred by the compoiler to `numToFlip` so when the right side of the assignment operator = on the first line of `doSimpleMath` is executed, we have

(10 * -1) + 15 = -10 + 15 = 5 // brush up on your math if needs be! Gotta have the basics down.

This is stored in `result` and printed to the console.

Multiplying a positive value by a negative value gives us a negative value so `numToFlip` * -1 will just flip the sign on whatever value is in `numToFlip`.