Test-2-Review

1

Since there are only two boolean values, you could test each case, because there are only four.

`p` is true and `b` is true = false

```
!(true && true) && (!true || true)
!(true) && (false || true)
false && true
false
```

`p` is false and `b` is true = true

```
!(false && true) && (!false || true)
!(false) && (true || true)
true && true
true
```

`p` is true and `b` is false = false

```
!(true && false) && (!true || false)
!(false) && (false || false)
true && false
false
```

`p` is false and `b` is false = true

```
!(false && false) && (!false || false)
!(false) && (true || false)
true && true
true
```

When `p` is true, the expression is false. When `p` is false, the expression is true.

The answer is b).

2

Answer Choice A

Answer Choice A prints `9 + 0.95`, a float plus an int, which returns **9.95**.

Answer Choice B

Answer Choice B prints `995/100.0`, an int divided by a float, which returns a **9.95**.

Answer Choice C

Answer Choice C prints `9. + 95/100`, a float plus an int divided by an int. The integer division returns 0, because division of an integer by an integer rounds down to an integer. Therefore the sum is 9.0 + 0, which is **9.0**.

Answer Choice D

Answer Choice D prints `9 + 95.0/100`, an int plus a float divided by an int. The division returns a float, so the float plus the int returns **9.95**.

Answer Choice E

Answer Choice E prints `9 + "." + 95`, an int concatenated with a string concatenated with an int. This forces all the ints into string form, so it's the same as printing `"9" + "." + "95"`, or **9.95**.

The answer is c).

3

It's helpful to look at each step of the loop. Here's what I did:

1. `i = 12, output = ""`
   Is `12 > 0`? Yes. Print i and run decrement `i` by 3.
2. `i = 9, output = "12"`
   Is `9 > 0`? Yes. Print i and run decrement `i` by 3.
3. `i = 6, output = "129"`
   Is `6 > 0`? Yes. Print i and run decrement `i` by 3.
4. `i = 3, output = "1296"`
   Is `3 > 0`? Yes. Print i and run decrement `i` by 3.
5. `i = 0, output = "12963"`
   Is `0 > 0`? No. Loop stops and code is done executing.

The output is `"12963"`.

The answer is b).

4

Again, it helps to visualize each step of the loop.

1. `i = 0`
   Is `i < 8` ? Yes, print i and continue loop.
2. `i = 1`
   Is `i < 8` ? Yes, print i and continue loop.
3. `i = 2`
   Is `i < 8` ? Yes, print i and continue loop.
4. `i = 3`
   Is `i < 8` ? Yes, print i and continue loop.
5. `i = 4`
   Is `i < 8` ? Yes, print i and continue loop.
6. `i = 5`
   Is `i < 8` ? Yes, print i and continue loop.
7. `i = 6`
   Is `i < 8` ? Yes, print i and continue loop.
8. `i = 7`
   Is `i < 8` ? Yes, print i and continue loop.
9. `i = 8`
   Is `i < 8` ? No, stop loop, code block has finished execution.

The final value of `i` is 8.

The answer is c)

5

The question asks you to only comment out violations of the rules for **class members**. The below code block displays the two lines that the compiler identifies as errors, and why.

```
public class GrokStatic {
        public static int classField;
```

```
        public /* non-static */ int instanceField;

        public static void classMethod(int x) {
                classField = 13;
                // instanceField = 17;
                // referring to an instanceField from a static method. but which instar
                classMethod(19);
                // instanceMethod(23);
                // referring to an instanceMethod from a static method. but which insta
        }
        public /* non-static */ void instanceMethod(int y) {
                classField = 29;
                instanceField = 31;
                classMethod(37);
                instanceMethod(41);
        }
}
```

There are no errors in one method, and two errors in another method.

The answer is e).

6

The factorial, not necessarily in the programming form, is a good example of a recursive problem. Here is a code sample:

```
public static int factorial (int number) {
        if (n < 1) {
                return 1;
        }
        else {
                return factorial(number - 1);
        }
}
```

One can see here the the factorial function takes a number, and calls *itself* on a *smaller* number (number - 1).

The answer is c).

7

According to Wikipedia,

> In computing, a namespace is a set of signs (names) that are used to identify and refer to objects of various kinds.

The definition provided is a paraphrased version of this.

The rest of the definitions are:

(dictionary)

> In computer science, an associative array, map, symbol table, or dictionary is an abstract data type composed of a collection of (key, value) pairs, such that each possible key appears at most once in the collection.

(scope)

> In computer programming, the scope of a name binding—an association of a name to an entity, such as a variable—is the part of a program where the name binding is valid, that is where the name can be used to refer to the entity.

(glossary)

> A glossary is an alphabetical list of terms in a particular domain of knowledge with the definitions for those terms.

(lookup table)

> In computer science, a lookup table (LUT) is an array that replaces runtime computation with a simpler array indexing operation.

The answer is a).

8

Answer choice d) contains two floats, which, when divided, will return a float. This will return an error, since the compiler wants to avoid lossy conversion and will only do so when explicit typecasting occurs.

The rest of the answer choices return ints and therefore are valid.

The answer is d).

9

Look at the `func` method in the `recursion` class. The function seems to take in `n`, but doesn't do anything with it. Therefore, the function keeps calling itself without any way to stop. This causes the lovely `java.lang.StackOverflowError` error.

This is a runtime error as the compiler lets the code compile.

The answer is e).

10

This is a relatively simple question. We can work through it like this:

1. Is `t > h`?
   No, therefore we go to the else statement.
2. Is `h > w`? Yes, therefore `s` is now 4.
3. Is `h > t`? Yes, therefore `s = s + 1` and therefore `s = 5`.

The answer is a).

11

In Java, the compiler forces the program to obey the PEMDAS system and the operates on same operators from left to right. With this in mind, this is the calculation.

```
(int)(5.0 + 2.0 + 5.0 / 2.0 - 5.0 * 2.0 - 5.0 / (10 * 2.0))
(int)(5.0 + 2.0 + 2.5 - 10.0 - 0.25)
(int)(-0.75)
0
```

The answer is a).

12

As per what we learned in class, these are the sizes of each type:

- `int` : 32 bits
- `byte` : 8 bits

- `short` : 16 bits
- `long` : 64 bits
- `double` : 64 bits

Each bit means that the data type can hold two times as much data. Therefore, each data type holds 2^number of bits that it can hold.

We start with `byte` .

A byte can hold numbers with a range of 256 (2^8). This starts from -128 and ends at 127.

Since `byte` can clearly hold this number, all other datatypes mentioned here can, so the answer is all of them.

The answer is d).

13

I honestly have got no idea. I think I heard e) in class once.

14

The answer is the one that uses a different datatype than `double` because it will either return an error, or within the function, change the value from the true value.

In a), the value in parentheses evaluate to 6. Typecasting it to double simply changes the data type, not the value. Therefore, the end value is 6.0.

The answer is a).

15

We can use a process of elimination here.

a)

`Math.random()` returns a random number between 0 and 1.
Therefore, `Math.random() * 15` would return a random number between 0 and 15.

b)

`(int)Math.random() + 15` merely truncates the decimal off a wrong answer, therefore it's wrong too.

c)

`(int)Math.random() * 5 + 10` gets the integer part of a random number from 0 to 1, multiplies it by 5, and then adds 10. The problem here is that `(int)Math.random()` ALWAYS returns 0, so the "random" number will always be 10.

d)

`(int)(Math.random() * 5) + 10` takes a random number from 0 to 1, multiplies it by 5 (0 to 5, excluding 5), and gets the integer part, then adds 10. This is the correct algorithm.

The correct answer is d).

16

The code tries to print a string, namely, the concatenation of a string saying `"1"`, a string saying `"2"`, and a string saying `"3"`. This is valid Java code and will print `"123"`.

The answer is b).

17

Comparing two instances of a class we defined ourselves directly will never return true as they are in different memory locations and therefore are not equal.

However, if we define an `equals()` method, and define the coin's default constructor, the function should return true when comparing two coins with null parameters.

The answer is d).

18

As per what we learned in class, these are the sizes of each type:

- `int` : 32 bits
- `byte` : 8 bits

- `short` : 16 bits
- `long` : 64 bits
- `double` : 64 bits

Each bit means that the data type can hold two times as much data. Therefore, each data type holds 2^number of bits that it can hold.

We start with `byte` .

A byte can hold numbers with a range of 256 (2^8). This starts from -128 and ends at 127.

`byte` cannot hold this number as 160 > 127. Therefore, we move onto the next-smallest datatype.

A short can hold numbers with a range of 65536 (2^16). This starts from -32768 and ends at 32767.

Since 160 < 32767, the `short` datatype can hold 160. All datatypes, above that, can hold 160.

The answer is b).

19

As always, it's helpful to look at each iteration the loop takes on here.

1. `x = 123` , `y = 0`
   Is `x > 0` ? Yes.
2. `x = 12` , `y = 3`
   Is `x > 0` ? Yes.
3. `x = 1` , `y = 32`
   Is `x > 0` ? Yes.
4. `x = 0` , `y = 321`
   Is `x > 0` ? No. Stop loop and report y value.

The final value of y is 321.

The answer is d).

20

Java evaluates the boolean expression such that it terminates calculation when it can ascertain that a value is guaranteed.

`meMaybe()` would be called if `((a == b) || !(c <= b))` is false.

Therefore, both `(a == b)` and `!(c <= b)` would be false.

`c <= b` would be true, and `a != b` would also be true.

d) is the only answer choice that fits here.

The answer is d).

21

I'll go through the loop here.

1. `p = 3, q = 1, sum = 0`
   Is `p <= 10`? Yes.
2. `p = 4, q = 2, sum = 0`
   Is `p <= 10`? Yes.
3. `p = 5, q = 3, sum = 0`
   Is `p <= 10`? Yes.
4. `p = 6, q = 4, sum = 2`
   Is `p <= 10`? Yes.
5. `p = 7, q = 5, sum = 4`
   Is `p <= 10`? Yes.
6. `p = 8, q = 6, sum = 6`
   Is `p <= 10`? Yes.
7. `p = 9, q = 7, sum = 8`
   Is `p <= 10`? Yes.
8. `p = 10, q = 8, sum = 10`
   Is `p <= 10`? Yes.
9. `p = 11, q = 9, sum = 12`
   Is `p <= 10`? No. Stop loop and report `sum`.

`sum` is 12 at the end of the loop.

The answer is c).

22

There isn't a better way to do this so here goes a process of elimination:

a)

A is wrong because if we were returning factors, the values of `m` would be divided, not multiplied. Plus, why is the variable name `m` in the first place?

b)

B is wrong because Crisco is already vegan and therefore cannot have a vegan alternative.

c)

There are no visible problems with this explanation.

d)

D is wrong because the least common factor is 1 and this can be obtained in a much easier way.

e)

E is wrong because `y * x` is much easier to obtain than showed in the function.

The answer is c).

23

`temp` stores the first letter of the first string. `w[0]` then stores the first letter of the second string followed by the rest of the first string. `w[1]` holds `temp` followed by the rest of the second string.

Following this protocol, we end up with `NOW HEAT`.

The answer is c).

24

We can go through the loop.

1. `n = 253`, `i = 0`
   Is `i < 50`? Yes, iterate.

2. `n = 128, i = 1`
   Is `i < 50` ? Yes, iterate.
3. `n = 65, i = 2`
   Is `i < 50` ? Yes, iterate.
4. `n = 34, i = 3`
   Is `i < 50` ? Yes, iterate.
5. `n = 18, i = 4`
   Is `i < 50` ? Yes, iterate.
6. `n = 10, i = 5`
   Is `i < 50` ? Yes, iterate.
7. `n = 6, i = 6`
   Is `i < 50` ? Yes, iterate.
8. `n = 4, i = 7`
   Is `i < 50` ? Yes, iterate.
9. `n = 3, i = 8`
   Is `i < 50` ? Yes, iterate.
10. `n = 3, i = 9`
    Is `i < 50` ? Yes, iterate.
11. `n = 3, i = 10`
    Is `i < 50` ? Yes, iterate. …

As we can see, around 8/50 iterations, n stabilizes at 3. Since n is not moving, it is safe to assume that the final value of n is 3.

The answer is c).