Team Walrus HW33 Response

Gabriel Thompson, Joshua Yagupsky, Nora Miller

November 12, 2021

1 Questions

1.1 Q0

GT's answer: Purple Hat by Sofi Tukker

NM's answer: Black Hole Sun by Soundgarden

YG's answer: fill this in, joshua

1.2 Q1

Our answer: **A**
Reasoning: If you make a truth table like the following:

p	b	!(p && b) && (!p |||b)
true	true	false
true	false	false
false	true	true
false	false	true

This demonstrates that p must be true in order for the result to evaluate to true

1.3 Q2

Our answer: **C**
Reasoning: Choice A would result in 9 + 0.95 evaluating to 9.95, choice B, 995/100.0, would also evaluate to 9.95 because there is a decimal point in 100.0. Choice C would evaluate, however, to 9.0, because 95/100 will be processed as int division and will be rounded down to 0.

1.4 Q3

Our answer: **B**
Reasoning: The code tells us several things: Firstly,numbers are being printed out in a line, starting with the number ten. Secondly, the number being printed reduces by 3 each time. This eliminates choices c, d, and e. Thirdly, the boolean expression for the loop is i¿0, meaning that when i becomes less than 0, the loop will end. This means that answer D is incorrect, leaving us with B.

1.5 Q4

Our answer: **C**
Reasoning: The while loop is essentially running through all values of i from 0 to 7 (inclusive), and adding 1 each time. In the last iteration, i will have a value of 7, so when it is incremented, the final value will be 8.(i cannot be more than 8, because once i is 8, the initial condition for the loop is not satisfied).

1.6 Q5

Our answer: **B**
Reasoning: There are two errors present in this code: `instanceField` and `instanceMethod()` can only be used in reference to a specific instance of the class, and each method uses both `instanceField` and `instanceMethod()`. Thus, 2 lines of code would need to be commented out from each method.

1.7 Q6

Our answer: **C**
Reasoning: Recursion, such as in the case of the factorial function, always relies on using the smallest possible cases of the same function to solve larger cases of that function. Therefore, recursion is a method in which the solution of a problem depends on smaller instances of the same problem.

1.8 Q7

Our answer: **A**

Reasoning: According to its definition on Wikipedia: "a namespace is a set of signs (names) that are used to identify and refer to objects of various kinds." In general, a namespace is an identifier for a piece of data.

1.9 Q8

Our answer: **D**
Reasoning: `3.9 / 1.3` will by default return the `double` value of `3.0`. This won't be automatically converted to `int`, because of concern of "lossy conversion". (**Note:** The oher way around, where g is a `double` and the value being assigned to g is an `int` would be treated differently. See questions 12 and 18.)

1.10 Q9

Our answer: **D**
Reasoning: The code will not be able to compile, because the prefix for the classes are `class` rather than `public class`. This means that they will not be able to access eachother. Also, the file won't be able to run, because Java requires a `public class` method in the file for it to be executable.

1.11 Q10

Our answer: **A**
Reasoning: The first `if` statement checks if `t` is larger than `h`, which it is not, so Java proceeds to the next `if` statement without changing anything. This statement checks if `h` is larger than `w`, which it is, so the value of `s` is set to 4. Then, the program checks if `h` (4) is greater than `t` (-4), which it is, so it increments `s` by 1, making it equal to 5.

1.12 Q11

Our answer: **A**
Reasoning: The program starts with the following:

```
(int)(x + y + x / y - x * y - x / (10*y) );
```

This is equivalent to,

```
(int)((5.0 + 2.0) + (5.0 / 2.0) - (5.0 * 2.0) - (5.0 / (10*2.0)) );
```

This then simplifies to,

```
(int)(7.0 + 2.5 - 10.0 - 0.25);
```

This then turns into `(int)(-0.75)`, which then gets turned into 0.(**Note:** Upon further investigation, typecasting from a `double` to an `int` returns the floor of a number if it is positive and the ceiling of a number if it is negative.)

1.13 Q12

Our answer: **D**
Reasoning: `int`, `short`, `long`, and `byte` all contain integers, so they can contain a number such as 89. `double` contains a floating point number, but since storing an integer as a floating point number does not risk "lossy conversion," this is fine, too. The range of `int` is from $-2^{31}+1$ to $2^{31}-1$, the range of `byte` is from -255 to $+255$, the range of `short` is from -32767 to $+32767$, the range of `long` is from $-2^{63}+1$ to $2^{63}-1$, and the range of `double` is from $-2^{31}+1$ to $2^{31}-1$, and 89 falls within all of those ranges.

1.14 Q13

Our answer: **C**
Reasoning: It's always a good idea to review code with others. The only other good moral is option E, although it's not really the moral of the class and we barely touched upon cognitive dissonance except during egoless programming. Also, pineapples have sleeves.

1.15 Q14

Our answer: **A**
Reasoning: Option A would evaluate `(double)(25/4)` to `(double)(6)`, because neither `25` nor `4` has a decimal place to indicate floating point accuracy. `(double)(6)` then evaluates to `6.0`, which is not `6.25`, so it is the correct answer.

1.16 Q15

Our answer: **D**
Reasoning: Essentially, we want to write some code that chooses a number at least 10 and less than 15, using `Math.random()`, which generates a random value from 0 to 1. Since the range our number can be in is 5, we multiply `Math.random()` by 5. Flooring the result using typecasting yields the integers 0, 1, 2, 3, and 4 with equal probability. Finally, we can add ten to the result of `(int)(Math.random()*5)`, to get a random number in range [10,15). Codified, this would be: `(int)(Math.random()*5)+10`, which does the aforementioned operations.

1.17 Q16

Our answer: **B**
Reasoning: The code `System.out.println("1" + new String("2") + 3);` first creates a new `String` object using its constructor, which returns the class instance, and then concatenates that with "1" and "3" by using the `String` object's `toString()` method. Note that Java treats `+` as a concatenation operator as soon as it finds a `String` in whatever it is running. Since the first thing before the `+`is a `String`, Java concatenates instead of adding.

1.18 Q17

Our answer: **D**
Reasoning: The code first creates three identical instances of the class `Coin()`. Then, it is testing their equality with `==`, which compares their memory addresses. This yield `false` for the first two invocations, as the three instances are stored at separate memory addresses. The next two, however, will yield `true`, as the `.equals()` method verifies that the instances themselves are the same.

1.19 Q18

Our answer: **B**
Reasoning: As explained in question 12, The value of 160 falls within all of the ranges listed in Q12, except for the range of `byte`, which can only handle values up to 127. Therefore, the code will be valid for datatype `int`, `short`, `long`, and `double` (I, III, IV, and V), just not `byte` (IV).

1.20 Q19

Our answer: **D**
Reasoning: The code will do the following: First, `x` will be 123 and `y` will be 0. Then, `y` will increase by 3 to become 3, and `x` will become 12. Then, `y` will multiply by 10 to become 30, `y` will increase by 2 to become 32, and `x` will become 1. Then, `y` will multiply by 10 to become 320, `y` will increase by 1 to become 321, and `x` will again divide by 10 to become 0. This will make the final value of `y` 321. This code is essentially just setting `y` to the reversed value of `x`, and setting `x` to 0.

1.21 Q20

Our answer: **D**
Reasoning: In order for `meMaybe()` to fire, the first part of the statement, must be `true`, so that Java gets to that part of the OR gate. If $a > b > c$, then both parts of $(a == b)||!(c <= b)$ will be false, as a will not be equal to b, and c won't not be less than or equal to c, as it *will* be less than b. This means that if $a < b < c$, the code will get to `meMaybe()`, so it will be called.

1.22 Q21

Our answer: **C**
Reasoning: This code is essentially incrementing `sum`, `p`, and `q` until `p` reaches 10, and in each iteration, incrementing `sum` by $p\%q$. First, $p\%q$ will be incremented by 0 (`3 % 1`), then incremented by 0 again (`4 % 2`), then incremented by 2 (`5 % 3`), then 0, then 2, then 2, then 2, then 2, and then 2 again. This will make for a total of 12, which will be the final value of `sum`.

1.23 Q22

Our answer: **C**
Reasoning: This code is basically running through every multiple of `x` until it reaches one which is divisible by `y`, which is how you find the least common multiple of two integers `x` and `y`. The output, `m`, is going to be the LCM of `x` and `y`.

1.24 Q23

Our answer: **C**
Reasoning: The code given essentially takes an array of `String`s, and returns the same array but with the first characters of the first two items swapped. It does this by creating a `temp` variable to store the first character of the first item. Then, it sets the first item to the concatenation of the first char of the second item and the rest of the first item. It then does the reverse for the second item, setting it to the first item of the first character (stored in `temp`) followed by the rest of the second item.

1.25 Q24

Our answer: **C**
Reasoning: This code starts with $n = 253$, and then adds 3 to it and halves it with integer division, 50 times. In the first iteration, `n` becomes 128, becomes 65 in the second iteration, and follows the following sequence: $253 \to 128 \to 65 \to 34 \to 17 \to 10 \to 6 \to 4 \to 3. Once it reaches 3, it stays at 3, as (3 + 3)/2 = 3$. The final value of `n` is 3.