Java Academy 2023 - Exam Week 1

Víctor Manuel Lavalle Cantón

1. **Which of the following Java operators can be used with boolean variables? (Choose all that apply.)**

   Options:

   - ==
   - +
   - -
   - !
   - %
   - <=
   - Cast with (boolean)

Answer: `==`, `!`, `Cast with (boolean)` -

Explanation : *The `==` can be used in both primitive operations and object references. References to objects, the `!` can only be used for booleans and can be applied a cast so that the returned value is a boolean.*

---

2. **What data type (or types) will allow the following code snippet to compile? (Choose all that apply.)**

```
byte apples = 5;
short oranges = 10;
_______ bananas = apples + oranges;
```

Options:

- int
- long
- boolean
- double
- short
- Byte

Answer: `int`, `long`, `double`

Explanation: `int(32)`, `long(64)` *y* `double(64)` *are primitives that have a larger value than* `short(16)` *and* `byte(8)`*, so they can be used as primitives. In the other hand can be casted but it does not apply for this question.*

---

3. **What change, when applied independently, would allow the following code snippet to compile? (Choose all that apply.)**

```
long ear = 10;
int hearing = 2 * ear;
```

   Options:

   - No change; it compiles as is.
   - Cast ear on line 4 to int.
   - Change the data type of ear on line 3 to short.
   - Cast 2 * ear on line 4 to int.

- Change the data type of hearing on line 4 to short.
- Change the data type of hearing on line 4 to long.

Answers:

- `Cast 2 * ear on line 4 to int`
- `Change the data type of ear on line 3 to short.`
- `Change the data type of hearing on line 4 to long`
- `Change the data type of hearing on line 4 to long.`

---

4. What is the output of the following program?

```
public class CandyCounter {
    static long addCandy(double fruit, float vegetables) {
        return (int) fruit + vegetables;
    }

    public static void main(String[] args) {
        System.out.print(addCandy(1.4, 2.4f) + "- ");
        System.out.print(addCandy(1.9, (float) 4) + "-");
        System.out.print(addCandy((long) (int) (short) 2, (float) 4));
    }
}
```

Options:

- 4-6-6.0
- 3-5-6
- 3-6-6
- 4-5-6
- The code does not compile because of line 9
- Non of the above

Answer: `None of the above`

Explanation: *The code would not compile because, although in line 3 we have a cast to int, vegetables is still a float, so the data type that remains is float.*

---

5. What are the unique outputs of the following code snippet? (Choose all that apply.)

```
int a = 2, b = 4, c = 2;
System.out.println(a > 2 ? --c : b++);
System.out.println(b = (a!=c ? a : b++));
System.out.println(a > b ? b < c ? b : 2 : 1);
```

Options:

- 1
- 2
- 3
- 4
- 5
- 6
- The code does not compile

Answer: `1`, `4` and `5`

Explanation: *First it prints* `4` *since* `a > 2` *is false, but, although* `b` *increases it will not increase until the next line since I use a* `b++.` *Then it prints* `5` *since* `a` *and* `c` *are equal. Equal will give us a* `false` *so* `b++`*, and as* `b` *increased to* `5` *on the last line,* `5` *will be assigned and the* `b++` *on that line will be cancelled. Finally,* `a=2`*,* `c=2` *and* `b=5` *so it will therefore prints* `1` *since* `b` *is not less than* `c` *and a is not greater than* `b`*.*

---

6. Given the following code snippet, what is the value of the variables after it is executed? (Choose all that apply.)

```
int ticketsTaken = 1;
int ticketsSold = 3;
ticketsSold += 1 + ticketsTaken++;
ticketsTaken *= 2;
ticketsSold += (long)1;
```

Options:

- `ticketsSold` is 8
- `ticketsTaken` is 2
- `ticketsSold` is 6
- `ticketsTaken` is 6
- `ticketsSold` is 7
- `ticketsTaken` is 4
- The code does not compile

Answers: `ticketsSold is 6 and ticketsTaken is 4`

Explanation: *In line 3* `ticketsTacken` *increases to 2, so* `ticketsSold` *is multiplied by 3, so its value is 6.*
3 is added to `ticketsSold` *so its value is 6, in line 4* `ticketsTaken` *is multiplied by 2*
*so (22) = 4.**

---

7. What is the output of the following code snippet? (Choose all that apply.)

```
3:      int temperature = 4;
4:      long humidity = -temperature + temperature * 3;
5:      if (temperature >= 4)
6:            if (humidity < 6) System.out.println("Too Low");
7:            else System.out.println("Just Right");
8:        else System.out.println("Too High");
```

Options:

- Too low
- Just Right
- To High
- A NullPointerException is thrown at runtime.
- The code will not compile because of line 7.
- The code will not compile because of line 8.

Answer: `Just Right`

Explanation: *In the first two lines the value of temperature is 4 while humidity is 8 by precedence of operations, therefore, it complies with the first if while the second one does not, since 6 is not less than 8, so it prints* `Just Right.`

---

8. Which statements, when inserted independently into the following blank, will cause the code to print 2 at runtime? (Choose all that apply.)

```
int count = 0;
BUNNY:
for (int row = 1; row <= 3; row++)
    RABBIT:for (int col = 0; col < 3; col++) {
        if ((col + row) % 2 == 0)
            _____________________;
        count++;
    }
System.out.println(count);
```

Options:

- break BUNNY
- break RABBIT
- continue BUNNY
- continue RABBIT
- break
- continue
- None of the above, as the code contains a compiler error

Answers: `break RABBIT`, `continue RABBIT`, `break`

Explanation: *The option that I would use in this case, is to use the break since inside* `count` *would increase to 1 and would re-enter since the value of* `row=2` *and* `col=0`*, so count would increase to 2, so* `row=3` *and* `col=0`*, so it would enter the for therefore count would increase to 2, so* `row= 3` *and* `col=0`*, so it would enter in the for again since it still complies with that it is* `<= 3`*, since count is already worth 2 the best would be to use the break so that it conserves the value we consider that break rabbit is equivalent reason why also it is a correct answer and at the end continue bunny it would not affect us since in that cycle for does not increase count.*

---

9. What is the output of the following code snippet?

```
2: boolean keepGoing = true;
3: int result = 15, meters = 10;
4: do {
5: meters--;
6: if(meters==8) keepGoing = false;
7: result -= 2;
8: } while keepGoing;
9: System.out.println(result);
```

Options:

- 7
- 9
- 10
- 11
- 15
- The code will not compile because of line 6.
- The code does not compile for a different reason.

Answer: `The code does not compile for a different reason.`

Explanation: *on line 8* `keepGoing` *does not have the parentheses so it does not make the comparison of a boolean.*

---

10. What is the output of the following code snippet? (Choose all that apply.)

```
 9: int w = 0, r = 1;
10: String name = "";
11: while(w < 2) {
12:     name += "A";
13:     do {
14:         name += "B";
15:         if(name.length()>0) name += "C";
16:         else break;
17:     } while (r <=1);
18: r++; w++; }
19: System.out.println(name);
```

Options:

- ABC
- ABCABC
- ABCABCABC
- Line 15 contains a compilation error.
- Line 18 contains a compilation error.
- The code compiles but never terminates at runtime.
- The code compiles but throws a NullPointerException at runtime.

Answer: `The code compiles but never terminates at runtime.`

Explanation: *In line 18 we can notice that* `r++`*, is incremented outside of the* `do while` *cycle so the cycle will continue to execute since r will not increment.*

---

11. What is output by the following code? (Choose all that apply.)

```
1: public class Fish {
2:  public static void main(String[] args) {
3:      int numFish = 4;
4:      String fishType = "tuna";
5:      String anotherFish = numFish + 1;
6:      System.out.println(anotherFish + " " + fishType);
7:      System.out.println(numFish + " " + 1);
8:  }
9: }
```

Options:

- 4 1
- 5
- 5 tuna
- 5tuna
- 51tuna
- The code does not compile.

Answer: `The code does not compile`

Explanation: *In line 5 there's this piece of code* `String anotherFish = numFish + 1;`*, we can notice that it wants to assign an int to a String and this cannot be done. It can be noticed that they are trying to assign an* `int` *to a* `String` *and this is not possible, therefore, the code will not compile.*

---

12. What is the result of the following code?

```
7: StringBuilder sb = new StringBuilder();
8: sb.append("aaa").insert(1, "bb").insert(4, "ccc");
9: System.out.println(sb);
```

Options:

- abbaaccc
- abbaccca
- bbaaaccc
- bbaaccca
- An empty line.
- The code does not compile.

Answer: `abbaccca`

Explanation: *First "aaa" is added to* `sb`*, then in position 1 after the first "a" "bb" is added, therefore, it would be "abbaa", finally in position 4 after the second "a" " we add "ccc" so that at the end it would print "*`abbaccca`*".*

---

13. What is the result of the following code

```
12: int count = 0;
13: String s1 = "java";
14: String s2 = "java";
15: StringBuilder s3 = new StringBuilder("java");
16: if (s1 == s2) count++;
17: if (s1.equals(s2)) count++;
18: if (s1 == s3) count++;
19: if (s1.equals(s3)) count++;
20: System.out.println(count);
```

Options:

- 0
- 1
- 2
- 3
- 4
- An exception is thrown.
- The code does not compile

Answer: `An exception is thrown`

Explanation: *Everything goes well until line 18, in this line we can see the use of a comparison between* `s1` *which is a* `String` *and* `s3` *which is a* `StringBuilder`*, something that is not allowed so the code will not compile.*

---

14. What is the result of the following code?

```
public class Lion {
    public void roar(String roar1, StringBuilder roar2) {
        roar1.concat("!!!");
        roar2.append("!!!");
    }

    public static void main(String[] args) {
        String roar1 = "roar";
        StringBuilder roar2 = new StringBuilder("roar");
        new Lion().roar(roar1, roar2);
        System.out.println(roar1 + " " + roar2);
    }
}
```

Options:

- roar roar
- roar roar!!!
- roar!!! roar
- roar!!! roar!!!
- An exception is thrown.
- The code does not compile

Answer: `roar roar!!!`

Explanation: *The problem with this block of code is that when calling the method* `roar`*, as* `roar1` *is a* `String` *it does not concatenate it since* `String` *is immutable and therefore is not modified.*

---

15. **Which of the following can replace line 4 to print "avaJ"? (Choose all that apply.)**

```
3: var puzzle = new StringBuilder("Java");
4: // INSERT CODE HERE
5: System.out.println(puzzle);
```

Options:

- puzzle.reverse();
- puzzle.append("vaJ$").substring(0, 4);
- puzzle.append("vaJ$").delete(0, 3).deleteCharAt(puzzle.length() - 1);
- puzzle.append("vaJ$").delete(0, 3).deleteCharAt(puzzle.length());
- None of the above

Answers: `puzzle.reverse();` and `puzzle.append("vaJ$").delete(0, 3).deleteCharAt(puzzle.length() - 1);`

Explanation: *When I saw the code I thought that only option 1 would be correct, but in the option 3* `puzzle.append("vaJ$").delete(0, 3).deleteCharAt(puzzle.length() - 1)` *it says that it will add "vaJS" and delete the first 3, so we would have "avaJ$" and then it deletes the last character so at the end we would also have "avaJ".*