

CAPITULO 2

# ACADEMIA JAVA

Esthefanny De La Cruz Carbajal

14-1-2025

# Review Questions

The answers to the chapter review questions can be found in the Appendix.

1. Which of the following Java operators can be used with boolean variables? (Choose all that apply.)
- A. ==
  - B. +
  - C. --
  - D. !
  - E. %
  - F. ~
  - G. Cast with (boolean)

A,D.

Los operadores que se pueden usar con booleanos son:

- ==
- !=
- !
- &&
- ||

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;
```

- A. int
- B. long
- C. boolean
- D. double
- E. short
- F. byte

A. int:

✓ Correcto. El resultado de apples + oranges es de tipo int, por lo que se puede asignar directamente a bananas si esta es de tipo int.

B. long:

✓ Correcto. Un resultado de tipo int se puede asignar a una variable de tipo long, ya que long tiene un rango mayor que int.

C. boolean:

✗ Incorrecto. El tipo boolean no es compatible con operaciones numéricas.

D. double:

✓ Correcto. Un resultado de tipo int se puede asignar a una variable de tipo double, ya que double puede almacenar números enteros y decimales.

E. short:

✗ Incorrecto. Un resultado de tipo int no se puede asignar directamente a una variable de tipo short.

F. byte:

✗ Incorrecto. Un resultado de tipo int no se puede asignar directamente a una variable de tipo byte .

**Respuesta: A,B,D**

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

```
3: long ear = 10;  
4: int hearing = 2 * ear;
```

- A. No change; it compiles as is.
- B. Cast ear on line 4 to int.
- C. Change the data type of ear on line 3 to short.
- D. Cast 2 \* ear on line 4 to int.
- E. Change the data type of hearing on line 4 to short.
- F. Change the data type of hearing on line 4 to long.

**B,C, D, F**

A. No change; it compiles as is.

✗ Incorrecto. El código no compila en su estado actual debido al conflicto de tipos de variables en la asignación de un long a una variable int.

B. Cast ear on line 4 to int.

✓ Correcto. convertir ear de long a int, permite que el código compile.

C. Change the data type of ear on line 3 to short.

✗ Incorrecto. Cambiar ear a short no soluciona el problema porque el resultado de la operación  $2 * ear$  aún se promueve a int, y asignar un int a una variable int funciona, pero el problema de tipos inicial persiste con valores más grandes.

D. Cast  $2 * ear$  on line 4 to int.

✓ Correcto. Realizar una conversión de tipo de variable en toda la operación garantiza que el resultado sea tratado como int y se pueda asignar a hearing

```
int hearing = (int) (2 * ear);
```

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

✗ Incorrecto. Cambiar el tipo de hearing a short no resuelve el problema, ya que el resultado de  $2 * ear$  sigue siendo long, y no se puede asignar directamente a una variable más pequeña sin usar algún método de conversión de tipo de variable

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

✓ Correcto. Cambiar el tipo de hearing a long permite que el resultado de  $2 * ear$  se asigne directamente sin conflicto de tipos.

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

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

```
3: boolean canine = true, wolf = true;
4: int teeth = 20;
5: canine = (teeth != 10) ^ (wolf=false);
6: System.out.println(canine+", "+teeth+", "+wolf);
```

- A. true, 20, true
- B. true, 20, false
- C. false, 10, true
- D. false, 20, false
- E. The code will not compile because of line 5.
- F. None of the above.

**B. true,20, false**

La línea 5 significa que si teeth diferente de 10 es true, a su vez esa línea asigna a wolf falso, finalmente a canine se le asigna el valor de esta expresión con el operador XOR, resultando canine en true.

5. Which of the following operators are ranked in increasing or the same order of precedence? Assume the + operator is binary addition, not the unary form. (Choose all that apply.)

- A. +, \*, %, --
- B. ++, (int), \*
- C. =, ==, !
- D. (short), =, !, \*
- E. \*, /, %, +, ==
- F. !, |, |, &
- G. ^, +, =, +=

C. =, ==, !

Precedencia:

== tiene mayor precedencia que ! (negación lógica), y ambos son de menor precedencia que =.

Conclusión: Correcto. La precedencia sigue un orden creciente o el mismo.

E. \*, /, %, +, ==

Precedencia:

\*, /, y % tienen la misma precedencia y son mayores que +.

== tiene la menor precedencia de todos.

Conclusión: Correcto. La precedencia de los operadores sigue un orden creciente o el mismo.

F. !, |, |, &

Precedencia:

! (negación lógica) tiene la mayor precedencia.

& (AND lógico) tiene mayor precedencia que | | (OR lógico).

Conclusión: Correcto. Los operadores siguen un orden de precedencia creciente o el mismo.

G. ^, +, =, +=

Precedencia:

^ (XOR) tiene mayor precedencia que + (suma), pero menor que =.

+= tiene la misma precedencia que =.

Conclusión: Correcto. La precedencia sigue un orden creciente o el mismo.

**Respuestas correctas:**

C. =, ==, !

E. \*, /, %, +, ==

F. !, ||, &

G. ^, +, =, +=

6. What is the output of the following program?

```
1: public class CandyCounter {
2:     static long addCandy(double fruit, float vegetables) {
3:         return (int)fruit+vegetables;
4:     }
5:
6:     public static void main(String[] args) {
7:         System.out.print(addCandy(1.4, 2.4f) + ", ");
8:         System.out.print(addCandy(1.9, (float)4) + ", ");
9:         System.out.print(addCandy((long)(int)(short)2, (float)4)); } }
```

A. 4, 6, 6.0

B. 3, 5, 6

C. 3, 6, 6

D. 4, 5, 6

E. The code does not compile because of line 9.

F. None of the above.

E

Línea 9: addCandy((long)(int)(short)2, (float)4)

Aquí, el valor 2 es convertido primero a short, luego a int, y finalmente a long, aunque el tipo long no es necesario para esta operación, ya que int es suficiente. El valor de fruit será 2 después de la conversión. La suma de 2 + 4 dará 6.

PERO....

El código no compilará porque la operación en el método addCandy intenta retornar un float como un long sin realizar un casting explícito.

`return (int)fruit+vegetables;` si fruit y vegetables tuvieran un paréntesis regresaría un int.

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

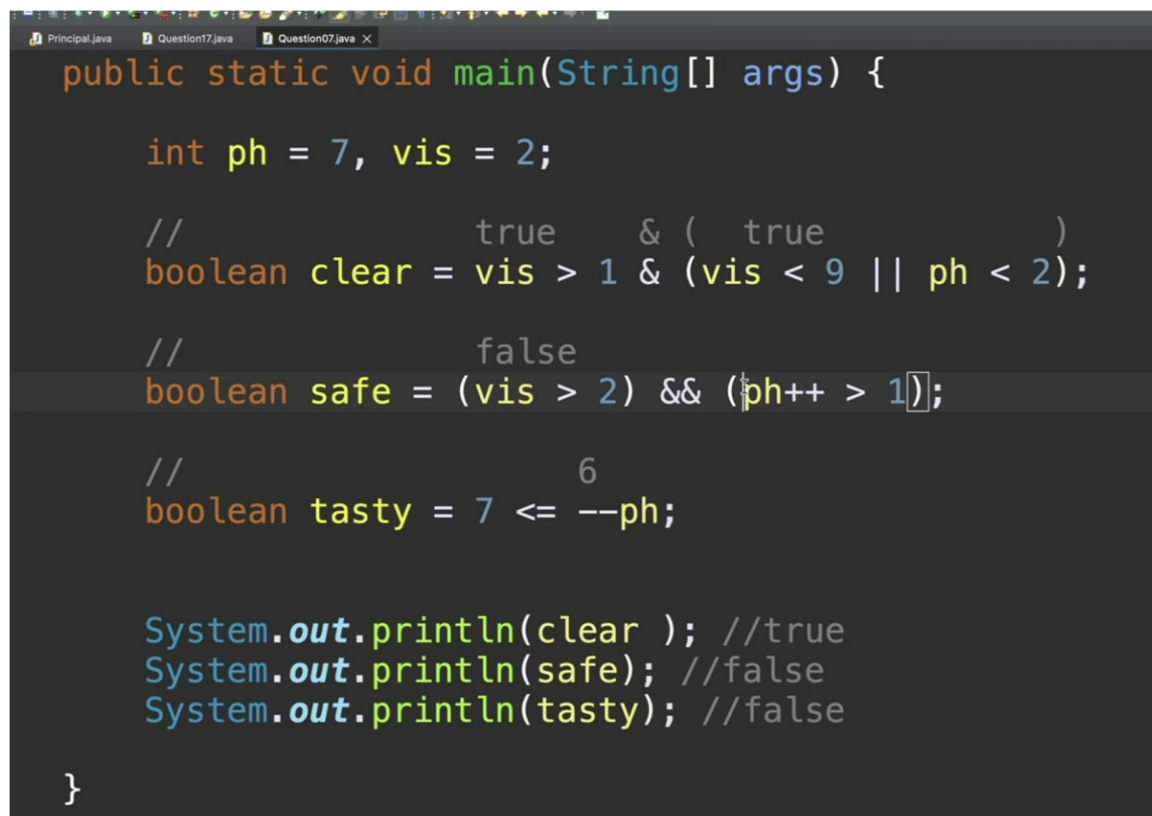
```
int ph = 7, vis = 2;
boolean clear = vis > 1 & (vis < 9 || ph < 2);
boolean safe = (vis > 2) && (ph++ > 1);
boolean tasty = 7 <= --ph;
System.out.println(clear + "-" + safe + "-" + tasty);
```

- A. true-true-true
- B. true-true-false
- C. true-false-true
- D. true-false-false
- E. false-true-true
- F. false-true-false
- G. false-false-true
- H. false-false-false

Clear :  $2 > 1 \ \& \ (2 < 9 \ \vee \ 7 < 2) \rightarrow \text{true}$

Safe:  $2 > 2$  si y solo si  $6 > 1 \rightarrow \text{false}$  (recordar que la segunda condición ya no se evalúa porque se usa  $\&\&$ )

Tasty:  $7 \leq 6 \rightarrow \text{false}$



```
Principal.java  Question17.java  Question07.java x
public static void main(String[] args) {

    int ph = 7, vis = 2;

    //           true    & (   true
    boolean clear = vis > 1 & (vis < 9 || ph < 2);

    //           false
    boolean safe = (vis > 2) && (ph++ > 1);

    //           6
    boolean tasty = 7 <= --ph;

    System.out.println(clear ); //true
    System.out.println(safe); //false
    System.out.println(tasty); //false

}
```

**Respuesta: D**

**8.** What is the output of the following code snippet?

```
4: int pig = (short)4;  
5: pig = pig++;  
6: long goat = (int)2;  
7: goat -= 1.0;  
8: System.out.print(pig + " - " + goat);
```

- A.** 4 - 1
- B.** 4 - 2
- C.** 5 - 1
- D.** 5 - 2
- E.** The code does not compile due to line 7.
- F.** None of the above.

**Respuesta: A**

Pig queda igual ya que la línea 5 se refiere a un postdecremento , es decir la variable se aumenta después de usarla.

En la línea 7 a Goat se le resta 1 , por lo cual su nuevo valor es 1.

**9.** 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);
```

- A.** 1
- B.** 2
- C.** 3
- D.** 4
- E.** 5
- F.** 6
- G.** The code does not compile.

Estructura general:

(condición ? si true : si false)



$a > 2 \rightarrow \text{false}$ ,  $b = 4$  y será 5 después de imprimir el valor. Se imprime  $b=4$

$2 \neq 2 \rightarrow \text{false}$ ,

Se ejecuta la parte después de los dos puntos ( $b++$ ), pero como es una asignación, el valor de  $b$  se usa antes de incrementar, por lo que  $b$  sigue siendo 5. Se imprime  $b=5$

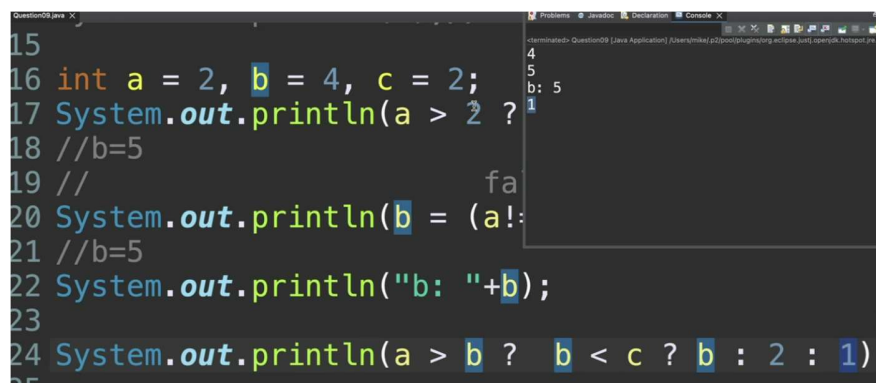
La asignación es  $b = b++$ , pero debido al operador de post-incremento, el valor de  $b$  antes de incrementar sigue siendo 5. Después de esta línea,  $b$  se convierte en 6.

Para la tercera línea de código :

Podemos interpretarlo con una estructura if, else:

```
int result;  
  
if (a > b) {  
    if (b < c) {  
        result = b;  
    } else {  
        result = 2;  
    }  
} else {  
    result = 1;  
}  
  
System.out.println(result);  
  
 $2 > 6 \rightarrow \text{false}$  y se imprime 1
```

Respuestas A,D,E



The screenshot shows a Java IDE with a code editor on the left and a console on the right. The code in the editor is as follows:

```
15  
16 int a = 2, b = 4, c = 2;  
17 System.out.println(a > 2 ? 1 : 2);  
18 //b=5  
19 //fa  
20 System.out.println(b = (a != 2) ? b : 2);  
21 //b=5  
22 System.out.println("b: " + b);  
23  
24 System.out.println(a > b ? b : b < c ? b : 2 : 1);  
25
```

The console on the right shows the output of the program:

```
4  
5  
b: 5  
1
```

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

```
short height = 1, weight = 3;
short zebra = (byte) weight * (byte) height;
double ox = 1 + height * 2 + weight;
long giraffe = 1 + 9 % height + 1;
System.out.println(zebra);
System.out.println(ox);
System.out.println(giraffe);
```

- A. 1
- B. 2
- C. 3
- D. 4
- E. 5
- F. 6
- G. The code does not compile.

El código no compila debido a la línea dos, ya que es necesario realizar un casting explícito short, ya que `byte * byte` da `int` como resultado e `int` es un tipo de dato más grande que `short`, por lo cual no es posible asignarlo a un `short` sin hacer un casting explícito.

La respuesta es G

11. What is the output of the following code?

```
11: int sample1 = (2 * 4) % 3;
12: int sample2 = 3 * 2 % 3;
13: int sample3 = 5 * (1 % 2);
14: System.out.println(sample1 + ", " + sample2 + ", " + sample3);
```

- A. 0, 0, 5
- B. 1, 2, 10
- C. 2, 1, 5
- D. 2, 0, 5
- E. 3, 1, 10
- F. 3, 2, 6
- G. The code does not compile.

Sample1= 2

Sample2= 6 % 3 = 0 (residuo)

Sample3 = 5 \* (1 % 2) = 5 \* 1 = 5

La respuesta es D

12. The \_\_\_\_\_ operator increases a value and returns the original value, while the \_\_\_\_\_ operator decreases a value and returns the new value.
- A. post-increment, post-increment
  - B. pre-decrement, post-decrement
  - C. post-increment, post-decrement
  - D. post-increment, pre-decrement
  - E. pre-increment, pre-decrement
  - F. pre-increment, post-decrement

Respuesta correcto D.

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

```
boolean sunny = true, raining = false, sunday = true;  
boolean goingToTheStore = sunny & raining ^ sunday;  
boolean goingToTheZoo = sunday && !raining;  
boolean stayingHome = !(goingToTheStore && goingToTheZoo);  
System.out.println(goingToTheStore + "-" + goingToTheZoo  
    + "-" +stayingHome);
```

- A. true-false-false
- B. false-true-false
- C. true-true-true
- D. false-true-true
- E. false-false-false
- F. true-true-false
- G. None of the above

GoingToTheStore = (false) XOR true = true

GoingToZoo = true && true = true

StayingHome = NOT(true && true) = false

Respuesta F

14. Which of the following statements are correct? (Choose all that apply.)
- A. The return value of an assignment operation expression can be `void`.
  - B. The inequality operator (`!=`) can be used to compare objects.
  - C. The equality operator (`==`) can be used to compare a `boolean` value with a numeric value.
  - D. During runtime, the `&` and `|` operators may cause only the left side of the expression to be evaluated.
  - E. The return value of an assignment operation expression is the value of the newly assigned variable.
  - F. In Java, `0` and `false` may be used interchangeably.
  - G. The logical complement operator (`!`) cannot be used to flip numeric values.

#### Respuestas B,E,G

A. The return value of an assignment operation expression can be `void`.

Incorrecto. En Java, las expresiones de asignación (`=`) devuelven el valor asignado, no `void`.

B. The inequality operator (`!=`) can be used to compare objects.

Correcto. El operador de desigualdad (`!=`) puede comparar referencias de objetos

Análisis de cada opción:

C. The equality operator (`==`) can be used to compare a `boolean` value with a numeric value.

Incorrecto. En Java, no puedes comparar valores booleanos con valores numéricos directamente. Esto provocaría un error de compilación.

D. During runtime, the `&` and `|` operators may cause only the left side of the expression to be evaluated.

Incorrecto. Los operadores `&` y `|` siempre evalúan ambos lados de la expresión. Si quieres evitar la evaluación del lado derecho cuando no es necesario, deberías usar los operadores lógicos de corto circuito (`&&` y `||`).

E. The return value of an assignment operation expression is the value of the newly assigned variable.

Correcto. El valor devuelto por una expresión de asignación es el valor asignado

F. In Java, `0` and `false` may be used interchangeably.

Incorrecto. En Java, no puedes usar `0` como sinónimo de `false`. Esto es diferente de lenguajes como C/C++.

G. The logical complement operator (`!`) cannot be used to flip numeric values.

Correcto. El operador lógico de complemento (`!`) solo se puede usar con valores booleanos y no con números.

15. Which operators take three operands or values? (Choose all that apply.)

- A. =
- B. &&
- C. \*=
- D. ? :
- E. &
- F. ++
- G. /

únicamente **la opción D** porque es el operador ternario

16. How many lines of the following code contain compiler errors?

```
int note = 1 * 2 + (long)3;  
short melody = (byte)(double)(note *= 2);  
double song = melody;  
float symphony = (float)((song == 1_000f) ? song * 2L : song);
```

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4

**Respuesta D**

Línea 1: error, no se puede asignar a un int un long.

Línea 2: error, no se puede asignar a un short un double

Línea 3: no hay error, se podría asignar song el valor de melody, ( en caso de que melody tuviera un valor válido)

Línea 4: La expresión ternaria (song == 1\_000f) ? song \* 2L : song genera un valor de tipo double, porque song \* 2L eleva el tipo al de mayor rango (double). Esto no puede asignarse a float sin un casting explícito.

17. Given the following code snippet, what are the values 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;
```

- A. ticketsSold is 8.
- B. ticketsTaken is 2.
- C. ticketsSold is 6.
- D. ticketsTaken is 6.
- E. ticketsSold is 7.
- F. ticketsTaken is 4.
- G. The code does not compile.

Respuesta C,F

En las operaciones compuestas se hace el cast de manera automática.

Línea 3: Primero se resuelve la suma y luego la operación compuesta , en esta línea se convierte ticketsTaken en 2

Ticketsold es 5 despues de la tercera línea

Ticketstaken es 4 despues en la cuarta línea

Ticketsold +1 = 6

18. Which of the following can be used to change the order of operation in an expression?  
(Choose all that apply.)

- A. [ ]
- B. < >
- C. ( )
- D. \ /
- E. { }
- F. " "

C

La única forma válida en Java para cambiar el orden de las operaciones en una expresión es usar paréntesis ( ), ya que estos permiten agrupar expresiones y forzar que ciertas partes de la expresión sean evaluadas primero, independientemente de la precedencia de los operadores.

**19.** What is the result of executing the following code snippet? (Choose all that apply.)

```
3: int start = 7;  
4: int end = 4;  
5: end += ++start;  
6: start = (byte)(Byte.MAX_VALUE + 1);
```

- A.** start is 0.
- B.** start is -128.
- C.** start is 127.
- D.** end is 8.
- E.** end is 11.
- F.** end is 12.
- G.** The code does not compile.
- H.** The code compiles but throws an exception at runtime.

**Respuesta: B,F**

Línea 5: end += ++start;

La operación ++start incrementa start en 1 antes de usarlo, por lo que start ahora es 8.

Luego, end += ++start equivale a end = end + start, es decir, end = 4 + 8.

Resultado: end = 12.

Línea 6: start = (byte)(Byte.MAX\_VALUE + 1);

Byte.MAX\_VALUE es 127.

Al sumar 1 a 127, el resultado es 128.

Sin embargo, al convertirlo a un byte, el valor se desborda debido al rango de byte (-128 a 127).

**20.** Which of the following statements about unary operators are true? (Choose all that apply.)

- A.** Unary operators are always executed before any surrounding numeric binary or ternary operators.
- B.** The - operator can be used to flip a boolean value.
- C.** The pre-increment operator (++) returns the value of the variable before the increment is applied.
- D.** The post-decrement operator (--) returns the value of the variable before the decrement is applied.
- E.** The ! operator cannot be used on numeric values.
- F.** None of the above

A. Unary operators are always executed before any surrounding numeric binary or ternary operators.

Correcto. Los operadores unarios (+, -, !, ++, --) tienen una mayor precedencia que los operadores binarios y ternarios. Por lo tanto, siempre se ejecutan primero.

D. The post-decrement operator (--) returns the value of the variable before the decrement is applied.

Correcto. El operador de post-decremento (--) primero devuelve el valor actual de la variable y luego aplica el decremento.

E. The ! operator cannot be used on numeric values.

Correcto. El operador lógico de negación (!) solo puede aplicarse a valores booleanos, no a valores numéricos. Intentar usarlo con un valor numérico generará un error de compilación.

A,E,D

**21.** What is the result of executing the following code snippet?

```
int myFavoriteNumber = 8;
int bird = ~myFavoriteNumber;
int plane = -myFavoriteNumber;
var superman = bird == plane ? 5 : 10;
System.out.println(bird + "," + plane + "," + --superman);
```

- A.** -7,-8,9
- B.** -7,-8,10
- C.** -8,-8,4
- D.** -8,-8,5
- E.** -9,-8,9
- F.** -9,-8,10
- G.** None of the above

No resolver