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, G

La opción A es el operador de igualdad y se puede usar en primitivas y referencias de objetos. Las opciones B y C son operadores aritméticos y no se pueden aplicar a un valor booleano. La opción D es el operador de complemento lógico y se usa exclusivamente con valores booleanos. La opción E es el operador de módulo, que solo se puede usar con primitivas numéricas. La opción F es un operador de complemento a nivel de bits y solo se puede aplicar a valores enteros. Finalmente, la opción G es correcta, ya que se puede convertir una variable booleana ya que boolean es un tipo.

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, B, D

La expresión apples + oranges se promueve automáticamente a int, por lo que int y los tipos de datos que se pueden promover automáticamente desde int funcionarán. Las opciones A, B y D son dichos tipos de datos. La opción C no funcionará porque boolean no es un tipo de datos numérico. Las opciones E y F no funcionarán sin una conversión explícita a un tipo de datos más pequeño.

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

El código no compilará tal como está, por lo que la opción A no es correcta. El valor 2 * ear se promueve automáticamente a long y no se puede almacenar automáticamente en hearing, que es un valor int. Las opciones B, C y D resuelven este problema reduciendo el valor long a int. La opción E no resuelve el problema y en realidad lo empeora al intentar colocar el valor en un tipo de datos más pequeño. La opción F resuelve el problema aumentando el tipo de datos de la asignación para que se permita long.

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

El código compila y se ejecuta sin problemas, por lo que la opción E no es correcta. Este ejemplo es complicado debido al segundo operador de asignación incrustado en la línea 5. La expresión (wolf=false) asigna el valor false a wolf y devuelve false para toda la expresión. Como teeth no es igual a 10, el lado izquierdo devuelve true; por lo tanto, la o exclusiva (^) de toda la expresión asignada a canine es true. La salida refleja estas asignaciones, sin cambios en teeth, por lo que la opción B es la única respuesta correcta.

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. ^, +, =, +=

A, C

Las opciones A y C muestran operadores en orden creciente o en el mismo orden de precedencia. Las opciones B y E están en orden decreciente o en el mismo orden de precedencia. Las opciones D, F y G no están ni en orden creciente ni decreciente de precedencia. En la opción D, el operador de asignación (=) está entre dos operadores unarios, con el operador de multiplicación (*) incorrectamente en lugar de la precedencia más alta. En la opción F, el operador de complemento lógico (!) tiene el orden de precedencia más alto, por lo que debería ser el último. En la opción G, los operadores de asignación tienen el orden de precedencia más bajo, no el más alto, por lo que los dos últimos operadores deberían ser los primeros.

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.

F

El código no compila porque la línea 3 contiene un error de compilación. La conversión (int) se aplica a fruit, no a la expresión fruit+vegetables. Dado que el operador de conversión tiene una precedencia de operador más alta que el operador de suma, se aplica a fruit, pero la expresión se promueve a float, debido a que vegetables es float. El resultado no se puede devolver como long en el método addCandy() sin una conversión. Por esta razón, la opción F es correcta. Si se agregaran paréntesis alrededor de fruit+vegetables, entonces la salida sería 3, 5, 6, y la opción B sería correcta. Recuerde que la conversión de números de punto flotante a valores integrales da como resultado un truncamiento, no un redondeo.

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
	D
	
	En la primera expresión booleana, vis es 2 y ph es 7, por lo que esta expresión se evalúa como true & (true || false), que se reduce a true. La segunda expresión booleana usa el operador condicional, y como (vis > 2) es falso, el lado derecho no se evalúa, dejando ph en 7. En la última asignación, ph es 7, y el operador de predecremento se aplica primero, reduciendo la expresión a 7 <= 6 y resultando en una asignación de false. Por estas razones, la opción D es la respuesta correcta.

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.	A El código compila y se ejecuta sin problemas, por lo que la opción E es incorrecta. La línea 7 no produce un error de compilación ya que el operador compuesto aplica la conversión automáticamente. La línea 5 incrementa pig en 1, pero devuelve el valor original de 4 ya que está utilizando el operador de postincremento. La variable pig luego se asigna a este valor, y la operación de incremento se descarta. La línea 7 simplemente reduce el valor de goat en 1, lo que resulta en una salida de 4 - 1 y hace que la opción A sea la respuesta correcta.

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.

A, D, E

El código compila sin problemas, por lo que la opción G es incorrecta. En la primera expresión, a > 2 es falso, por lo que b se incrementa a 5; pero como se usa el operador de postincremento, se imprime 4, lo que hace que la opción D sea correcta. El --c no se aplicó, porque solo se evaluó una de las expresiones del lado derecho. En la segunda expresión, a!=c es falso ya que c nunca se modificó. Dado que b es 5 debido a la línea anterior y se usa el operador de postincremento, b++ devuelve 5. El resultado luego se asigna a b usando el operador de asignación, anulando el valor incrementado para b e imprimiendo 5, lo que hace que la opción E sea correcta. En la última expresión, no se requieren paréntesis, pero la falta de paréntesis puede dificultar la lectura de las expresiones ternarias. De las líneas anteriores, a es 2, b es 5 y c es 2. Podemos reescribir esta expresión con paréntesis como (2 > 5 ? (5 < 2 ? 5 : 2) : 1). La segunda expresión ternaria nunca se evalúa ya que 2 > 5 es falso, y la expresión devuelve 1, lo que hace que la opción A sea correcta.

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.

G

El código no compila debido a un error en la segunda línea. Aunque tanto height como weight se convierten a byte, el operador de multiplicación los promueve automáticamente a int, lo que resulta en un intento de almacenar un int en una variable short. Por esta razón, el código no compila, y la opción G es la única respuesta correcta. Esta línea contiene el único error de compilación.

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.

D

Primero, * y % tienen la misma precedencia de operador, por lo que la expresión se evalúa de izquierda a derecha a menos que haya paréntesis presentes. La primera expresión se evalúa como 8 % 3, lo que deja un resto de 2. La segunda expresión se evalúa de izquierda a derecha ya que * y % tienen la misma precedencia de operador, y se reduce a 6 % 3, que es 0. La última expresión se reduce a 5 * 1, que es 5. Por lo tanto, la salida en la línea 14 es 2, 0, 5, lo que hace que la opción D sea la respuesta correcta.

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

D

El prefijo pre- indica que la operación se aplica primero y se devuelve el nuevo valor, mientras que el prefijo post- indica que el valor original se devuelve antes de la operación. Luego, el incremento aumenta el valor, mientras que el decremento disminuye el valor. Por estas razones, la opción D es la respuesta correcta.

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

F

La primera expresión se evalúa de izquierda a derecha ya que la precedencia del operador de & y ^ es la misma, lo que nos permite reducirla a false ^ sunday, que es true, porque sunday es true. En la segunda expresión, aplicamos el operador de negación (!) primero, reduciendo la expresión a sunday && true, que se evalúa como true. En la última expresión, ambas variables son true, por lo que se reducen a !(true & true), que se reduce aún más a !true, también conocido como false. Por estas razones, la opción F es la respuesta correcta.

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.

B, E, G

El valor de retorno de una operación de asignación en la expresión es el mismo que el valor de la variable recién asignada. Por esta razón, la opción A es incorrecta y la opción E es correcta. La opción B es correcta, ya que los operadores de igualdad (==) y desigualdad (!=) se pueden usar con objetos. La opción C es incorrecta, ya que los tipos booleano y numérico no son comparables. Por ejemplo, no se puede decir true == 3 sin un error de compilación. La opción D es incorrecta, ya que los operadores lógicos evalúan ambos lados de la expresión. El operador (!) hará que se evalúen ambos lados. La opción F es incorrecta, ya que Java no acepta números para valores booleanos. Finalmente, la opción G es correcta, ya que se necesita usar el operador de negación (-) para invertir o negar valores numéricos, no el operador de complemento lógico (!).

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

A. =

B. &&

C. *=

D. ? :

E. &

F. ++

G. /

D

El operador ternario es el único operador que toma tres valores, lo que hace que la opción D sea la única opción correcta. Las opciones A, B, C, E y G son todos operadores binarios. Si bien se pueden encadenar en expresiones más largas, cada operación usa solo dos valores a la vez. La opción F es un operador unario y toma solo un valor.

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

B

La primera línea contiene un error de compilación. El valor 3 se convierte a long. El valor 1 * 2 se evalúa como int pero se promueve a long cuando se suma al 3. Intentar almacenar un valor long en una variable int desencadena un error del compilador. Las otras líneas no contienen ningún error de compilación, ya que almacenan valores más pequeños en tipos de datos más grandes o del mismo tamaño, con las líneas 2 y 4 usando conversión para hacerlo. Dado que solo una línea no compila, la opción B es correcta.

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.

C, F

Los valores iniciales de ticketsTaken y ticketsSold son 1 y 3, respectivamente. Después de la primera asignación compuesta, ticketsTaken se incrementa a 2. El valor de ticketsSold se incrementa de 3 a 5; dado que se utilizó el operador de postincremento, el valor de ticketsTaken++ devuelve 1. En la siguiente línea, ticketsTaken se duplica a 4. En la línea final, ticketsSold se incrementa en 1 a 6. Los valores finales de las variables son 4 y 6, para ticketsTaken y ticketsSold, respectivamente, lo que hace que las opciones C y F sean las respuestas correctas. Tenga en cuenta que la última línea no desencadena un error de compilación ya que el operador compuesto convierte automáticamente el operando de la derecha.

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

Solo los paréntesis, (), se pueden usar para cambiar el orden de operación en una expresión, lo que hace que la opción C sea correcta. Los otros operadores, como [], <> y {}, no se pueden usar como paréntesis en Java.

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.

B, F

El código compila y se ejecuta correctamente, por lo que las opciones G y H son incorrectas. En la línea 5, el operador de preincremento se ejecuta primero, por lo que start se incrementa a 8, y el nuevo valor se devuelve como el lado derecho de la expresión. El valor de end se calcula sumando 8 al valor original de 4, dejando un nuevo valor de 12 para end y haciendo que la opción F sea una respuesta correcta. En la línea 6, estamos incrementando uno más allá del valor máximo de byte. Debido al desbordamiento, esto dará como resultado un número negativo, lo que hace que la opción B sea la respuesta correcta. Incluso si no conocía el valor máximo de byte, debería haber sabido que el código compila y se ejecuta y haber buscado la respuesta para start con un número negativo.

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, D, E

Los operadores unarios tienen el orden de precedencia más alto, lo que hace que la opción A sea correcta. El operador de negación (-) se usa solo para valores numéricos, mientras que el operador de complemento lógico (!) se usa exclusivamente para valores booleanos. Por estas razones, la opción B es incorrecta y la opción E es correcta. Finalmente, los operadores de preincremento/predecremento devuelven el nuevo valor de la variable, mientras que los operadores de postincremento/postdecremento devuelven la variable original. Por estas razones, la opción C es incorrecta y la opción D es correcta.

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

E

El complemento a nivel de bits de 8 se puede encontrar multiplicando el número por menos uno y restando uno, lo que hace que -9 sea el valor de bird. Por el contrario, plane es -8 porque niega myFavoriteNumber. Dado que bird y plane no son iguales, a superman se le asigna un valor de 10. El operador de predecremento toma superman, resta 1 y devuelve el nuevo valor, imprimiendo 9. Por esta razón, la opción E es correcta.