OCP

Oracle® Certified Professional Java SE 17 Developer

STUDY GUIDE

EXAM 1Z0-829

Includes one year of FREE access after activation to the interactive online learning environment and study tools:

3 custom practice exams

More than 500 electronic flashcards

Searchable key term glossary

SCOTT SELIKOFF
JEANNE BOYARSKY

SYBEX®
A Wiley Brand

CAPITULO 1:

1. Which of the following are legal entry point methods that can be run from the commandline? (Choose all that apply.)

   A. `private static void main(String[] args)`
   B. `public static final main(String[] args)`
   C. `public void main(String[] args)`
   D. `public static final void main(String[] args)`
   E. `public static void main(String[] args)`
   F. `public static main(String[] args)`

EXPLICACION: Para considerar la respuesta correcta debemos tener en cuenta los **Modificadores de acceso** public y static y la **Firma del método** void main(String[] args.

   A. La opción E es el método estándar de entrada con la firma correcta y sus modificadores necesarios.
   B. La opción D es correcta aún con el modificador final, porque final no impide que el método main sea ejecutado, solo asegura que no puede ser sobrescrito en subclases.

2. Which answer options represent the order in which the following statements can be assem-bled into a program that will compile successfully? (Choose all that apply.)

```
X: class Rabbit {}
Y: import
java.util.*;Z:
package animals;
```

   A. `X,Y,Z`
   B. `Y,Z,X`
   C. `Z,Y,X`
   D. `Y,X`
   E. `Z,X`
   F. `X,Z`
   G. None of the above

EXPLICACION: Un archivo Java puede contener solo una declaración de **package**, que está debe ser la primera línea del archivo (Si está presente). Después pueden ir las declaraciones **import** y finalmente se puede declarar la **clase**.

3. Which of the following are true? (Choose all that apply.)

```
public class Bunny {
   public static void main(String[] x)
      {Bunny bun = new Bunny();
} }
```

A. Bunny is a class.
B. bun is a class.
C. main is a class.
D. Bunny is a reference to an object.
E. bun is a reference to an object.
F. main is a reference to an object.
G. The main() method doesn't run because the parameter name is incorrect.

EXPLICACION:

C. A es correcto porque Bunny esta definido como una clase

D. E es correcto porque en la línea Bunny bun = new Bunny(); bun es una variable que se utiliza para referirse a una instancia objeto de la clase Bunny.

4. Which of the following are valid Java identifiers? (Choose all that apply.)

A. `_`

B. `_helloWorld$`

C. `true`

D. `java.lang`

E. `Public`

F. `1980_s`

G. `_Q2_`

EXPLICACION: Para esta pregunta se tomo en cuenta la regla básica de los identificadores en Java.

E. Los identificadores pueden comenzar con: Una letra (a-z, A-Z), un carácter monetario ($) o un guion bajo (_)

F. No pueden incluir espacios en blancos.

G. No se pueden utilizar palabras reservadas como identificadores (true, false, null, class, public, etc.)

H. Los identificadores pueden incluir dígitos, pero no pueden comenzar con ellos.

Por lo tanto las opciones B, E y G cumplen con la regla básica, la opción E la hace correcta por empezar con P mayúscula.

5. Which statements about the following program are correct? (Choose all that apply.)

```
2:  public class Bear {
3:     private Bear pandaBear;
4:     private void roar(Bear b) {
5:        System.out.println("Roar!");
6:        pandaBear =
b;7:   }
8:     public static void main(String[] args)
{9:       Bear brownBear = new Bear();
10:       Bear polarBear = new
Bear();11:
             brownBear.roar(polarBear);
```

```
12:       polarBear = null;
13:       brownBear = null;
14:       System.gc(); } }
```

A. The object created on line 9 is eligible for garbage collection after line 13.
B. The object created on line 9 is eligible for garbage collection after line 14.
C. The object created on line 10 is eligible for garbage collection after line 12.
D. The object created on line 10 is eligible for garbage collection after line 13.
E. Garbage collection is guaranteed to run.
F. Garbage collection might or might not run.
G. The code does not compile.

EXPLICACION:

I. La opción A es correcta ya que en la línea 9 efectivamente se crea un objeto Bear y en la línea 13 se establece en null por lo que ya no hay referencias a ese objeto y es elegible para el garbage collection.

J. Opción D, tanto brownBear como polarBear son null después de loa línea 13, por lo tanto el objeto creado en la línea 10 es elegible para el garbage collection.

K. F, Si bien se utilizo System.gc() no garantiza que sea llamado el garbageCollector, realmente no se sabe con exactitud cuando el garbage collection hace acción en el código pero sí se sabe que mientras más nueva la versión de Java más merojado y optimizado es.

6. Assuming the following class compiles, how many variables defined in the class or method are in scope on the line marked on line 14?

```
1: public class Camel {
2:          { int hairs = 3_000_0; }
3:          long water, air=2;
4:          boolean twoHumps = true;
5:          public void spit(float distance) {
6:                    var path = "";
7:                    { double teeth = 32 + distance++; }
8:                    while(water > 0) {
9:                    int age = twoHumps ? 1 : 2;
10:                   short i=-1;
11:                   for(i=0; i<10; i++) {
12:                   var Private = 2;
13:                   }
14:                   // SCOPE
15:                   }
16:         }
17: }
```

A. 2
B. 3
C. 4
D. 5
E. 6
F. 7
G. None of the above

EXPLICACION:

L. Variables que ESTAN en alcance en la línea 14:

- **Wáter –** Variable de instancia de la clase Camel
- **air –** Variable de instancia de la clase Camel
- **twoHumps –** Variable de instancia de la clase Camel
- **path -** Variable local en el método spit
- **age –** Variable local en el bucle while
- **i –** Variable local en el bucle while
- **Private –** Variable local en el bucle for

7. Which are true about this code? (Choose all that apply.)

```
public class KitchenSink
    {private int
    numForks;

    public static void main(String[] args)
       {int numKnives;
       System.out.print("""
          "# forks = " + numForks +
           " # knives = " + numKnives
          +# cups = 0""");
    }
}
```

a. The output includes: `# forks = 0`.

b. The output includes: `# knives = 0`.

c. The output includes: `# cups = 0`.

d. The output includes a blank line.

e. The output includes one or more lines that begin with whitespace.

The code does not compile.

EXPLICACION:

A. FALSO, no se puede acceder a numForks desde main, por lo que no se puede imprimir

B. FALSO, numKnives no se ha inicializado, por lo que no se puede imprimir

C. VERDADERO, aunque no se puede imprimir coo parte de la cadena, el texto está presente en la línea, pero no se ejecutara correctamente.

D. FALSO, No se genera una salida en blanco debido a los errores de compilación.

E. CIERTO, La cadena de texto incluye espacios en blanco al principio.

8. Which of the following code snippets about `var` compile without issue when used in amethod? (Choose all that apply.)

A. `var spring = null;`

B. `var fall = "leaves";`

C. `var evening = 2; evening = null;`

D. `var night = Integer.valueOf(3);`

E. `var day = 1/0;`

F. `var winter = 12, cold;`

G. `var fall = 2, autumn = 2;`

H. `var morning = ""; morning = null;`

EXPLICACION:

A. FALSO, var no puede inferir un tipo de null, porque causará un error de compilación

B. VERDADERO, var puede inferir el tipo String correctamente

C. FALSO, evening infiere como int y al asignarle null causará un error de compilación

D. VERDADERO, var puede inferir el tipo Integer correctamente

E. VERDADERO, aunque 1/10 causará una excepción de división por cero en tiempo de ejecución, el cpodigo compila sin problemas.

F. FALSO, no se puede declarar múltiples variables con var en una sola declaración.

G. FALSO, el caso es similar al anterior

H. VERDADERO, morning se infiere como String, y luego se le puede asignar null.

9. Which of the following are correct? (Choose all that apply.)

a. An instance variable of type `float` defaults to `0`.

b. An instance variable of type `char` defaults to `null`.

c. A local variable of type `double` defaults to `0.0`.

d. A local variable of type `int` defaults to `null`.

e. A class variable of type `String` defaults to `null`.

f. A class variable of type `String` defaults to the empty string `""`.

g. None of the above.

EXPLICACION:

A. FALSO, para un float el valor predeterminado es 0.0f

B. FALSO, las variables de instancias de tipo char se inicializan a '\u0000' que es el valor nulo para caracteres.

C. FALSO, Las variables locales no tienen un valor predeterminado y deben ser inicializadas

D. FALSO, las variables locales de tipo int no tienen un valor predeterminado

E. VERDADERO, las variables de clase (también llamadas variables estáticas o de instancias) de tipo String se inicializan a null si no se le asignan un valor.

F. FALSO, las variables de clase no se puede inicializar en una cadena vacía ""

10. Which of the following expressions, when inserted independently into the blank line, allow the code to compile? (Choose all that apply.)

```
public void printMagicData()
   {    var     magic     =
        _____________________
        ;
```

```
        System.out.println(magic);
    }
```

A. `3_1`

B. `1_329_.0`

C. `3_13.0_`

D. `5_291._2`

E. `2_234.0_0`

F. `9___6`

G. `_1_3_5_0`

EXPLICACION DE LOS GUIONES BAJOS: En java, se puede usar guiones bajos para hacer que los números sean más legibles.

M. No se puede usar guion bajo al final de un número o antes de un punto decimal

N. No se puede comenzar un número con guion bajo

Por lo tanto, las salidas serian:

A. 31

E. 2234.00

F. 96

11. Given the following two class files, what is the maximum number of imports that can be removed and have the code still compile?

```
// Water.java
package aquarium;
public class Water { }

// Tank.java
package aquarium;
import java.lang.*;
import java.lang.System;
import aquarium.Water;
import aquarium.*;
public class Tank {
   public void print(Water water) {
  System.out.println(water); } }
```

A. 0
B. 1
C. 2
D. 3
E. 4
F. Does not compile

EXPLICAION:

O. El paquete java.lang se importa automáticamente, por lo que no necesitas especificarlo.

P. import java.lang.System; parte del paquete java.lang. Sin embargo, dado que java.lang ya se importa automáticamente, esta línea también es redundante y se puede eliminar.

Q. Cuando dos clases están en el mismo paquete, puedes acceder a ellas directamente sin necesidad de importaciones explícitas.

12. Which statements about the following class are correct? (Choose all that apply.)

```
1: public class ClownFish {
2:    int gills = 0, double
weight=2;
3:    { int fins = gills; }
4:    void print(int length = 3) {
5:
      System.out.println(gills);
6:       System.out.println(weight);
7:       System.out.println(fins);
8:       System.out.println(length);
9: } }
```

A. Line 2 generates a compiler error.
B. Line 3 generates a compiler error.
C. Line 4 generates a compiler error.
D. Line 7 generates a compiler error.
E. The code prints `0`.
F. The code prints `2.0`.
G. The code prints `2`.
H. The code prints `3`.

EXPLICACION:
A. Línea 2: ERROR DE COMPILACION, en Java, no se puede declarar múltiples variables de diferentes tipos en una sola declaración.
C. Línea 4: ERROR DE COMPILACIÓN, no se permite proporcionar valores predeterminados para los parámetros de los métodos.
D. Línea 7: ERROR DE COMPILACIÓN, la variable fins fue declarada en un bloque de inicialización y no está accesible dentro del método print. Por lo tanto, intentar acceder a fins aquí generará un error.

13. Given the following classes, which of the following snippets can independently be inserted in place of `INSERT IMPORTS HERE` and have the code compile? (Choose all that apply.)

```
package aquarium;
public class Water
{
   boolean salty = false;
}

package
aquarium.jellies;public
class Water {
   boolean salty = true;
}

package employee;
```

```
INSERT IMPORTS
HERE
public class WaterFiller
   {Water water;
}
```

a. `import aquarium.*;`

b. `import aquarium.Water;`
`import aquarium.jellies.*;`

c. `import aquarium.*;`
`import aquarium.jellies.Water;`

d. `import aquarium.*;`
`import aquarium.jellies.*;`

e. `import aquarium.Water;`
`import aquarium.jellies.Water;`

f. None of these imports can make the code compile.

EXPLICACION: Las opciones correctas que permiten que el código compile son A, B y C

14. Which of the following statements about the code snippet are true? (Choose all that apply.)

```
3: short numPets = 5L;
4: int numGrains = 2.0;
5: String name = "Scruffy";
6: int d = numPets.length();
7: int e = numGrains.length;
8: int f = name.length();
```

A. Line 3 generates a compiler error.
B. Line 4 generates a compiler error.
C. Line 5 generates a compiler error.
D. Line 6 generates a compiler error.
E. Line 7 generates a compiler error.
F. Line 8 generates a compiler error.

EXPLICACION:

A. El sufijo L indica que el número es un long. No puedes asignar un long directamente a un short sin una conversión explícita, ya que puede haber pérdida de datos. Por lo tanto, esta línea genera un error de compilación.

B. El valor 2.0 es un número de punto flotante (tipo double), y no se puede asignar directamente a una variable de tipo int sin una conversión explícita.

D. La variable numPets es de tipo short, y los tipos primitivos como short no tienen un método length(). Este método se utiliza para cadenas (Strings) y arreglos,

E. numGrains es un tipo primitivo int, y los tipos primitivos no tienen una propiedad length.

15. Which of the following statements about garbage collection are correct? (Choose allthat apply.)

A. Calling `System.gc()` is guaranteed to free up memory by destroying objects eligiblefor garbage collection.

B. Garbage collection runs on a set schedule.

C. Garbage collection allows the JVM to reclaim memory for other objects.

D. Garbage collection runs when your program has used up half the available memory.

E. An object may be eligible for garbage collection but never removed from the heap.

F. An object is eligible for garbage collection once no references to it are accessible in theprogram.

G. Marking a variable `final` means its associated object will never be garbage collected.

EXPLICACION:

R. Llamar a System.gc() es una sugerencia para el recolector de basura, pero no garantiza que se libere memoria ni que se destruyan objetos. El recolector de basura decide cuándo y qué objetos recolectar.

S. La recolección de basura no sigue un horario fijo y se activa en función de las necesidades de memoria de la aplicación.

T. La recolección de basura permite que la JVM recupere memoria ocupada por objetos que ya no son accesibles, permitiendo así que esa memoria sea reutilizada para otros objetos.

U. No hay un umbral específico para la recolección de basura

V. Un objeto puede ser elegible para la recolección de basura si no tiene referencias accesibles, pero puede no ser recolectado inmediatamente. Por lo tanto, puede permanecer en el heap hasta que el recolector decida liberar la memoria.

W. El modificador final significa que la referencia a esa variable no puede cambiar, pero no impide que el objeto al que apunta sea recolectado si no hay otras referencias a él.

16. Which are true about this code? (Choose all that apply.)

```
var blocky =
   """squirrel
   \s pigeon
            \
   termite""";
System.out.print(blocky);
```

a. It outputs two lines.

b. It outputs three lines.

c. It outputs four lines.

d. There is one line with trailing whitespace.

e. There are two lines with trailing whitespace.

f. If we indented each line five characters, it would change the output.

EXPLICACION:

A. La cadena blocky contiene caracteres de nueva línea implícitos debido a la forma en que se define. En este caso, \s se considera un espacio, y la cadena se interpreta como dos líneas: una línea para "squirrel" y otra línea para "pigeon \ termite".

D. La primera línea ("squirrel ") tiene un espacio al final (debido a la forma en que se define la cadena), lo que significa que hay un espacio en blanco al final de esa línea.

17. What lines are printed by the following program? (Choose all that apply.)

```
1: public class WaterBottle {
2:         private String brand;
3:         private boolean empty;
4:         public static float code;
5:         public static void main(String[] args) {
6:                 WaterBottle wb = new WaterBottle();
7:  System.out.println("Empty = " + wb.empty);
8:  System.out.println("Brand = " + wb.brand);
9: System.out.println("Code = " + code);
10:  } }
```

A. Line 8 generates a compiler error.
B. Line 9 generates a compiler error.
C. Empty =
D. Empty = false
E. Brand =
F. Brand = null
G. Code = 0.0
H. Code = 0f

EXPLICACION:

D. El campo empty es de tipo booleano y, por defecto, se inicializa a false.

F. El campo brand es de tipo String y, por defecto, se inicializa a null.

G. El campo brand es de tipo String y, por defecto, se inicializa a null.

18. Which of the following statements about `var` are true? (Choose all that apply.)

A. A `var` can be used as a constructor parameter.
B. The type of a `var` is known at compile time.
C. A `var` cannot be used as an instance variable.
D. A `var` can be used in a multiple variable assignment statement.
E. The value of a `var` cannot change at runtime.
F. The type of a `var` cannot change at runtime.
G. The word `var` is a reserved word in Java.

EXPLICACION:

- var se utiliza para inferir tipos en declaraciones locales, pero no se puede usar como tipo en parámetros de métodos o constructores. Los parámetros deben tener un tipo explícito.
- Cuando se usa var, el compilador infiere el tipo de la variable en base al valor que se le asigna. Esto significa que el tipo se determina en tiempo de compilación, lo que permite al compilador realizar verificaciones de tipo y evitar errores.
- var solo se puede usar para variables locales dentro de un método o bloque de código. No se puede usar para declarar variables de instancia (atributos de una clase)
- No se puede usar var para declarar múltiples variables en una sola declaración. Cada variable debe ser declarada individualmente con var o con su tipo explícito.
- Al usar var, la variable puede ser reasignada a otros valores del mismo tipo.
- podemos cambiar el valor de una variable declarada con var, el tipo de esa variable (inferido en el momento de la declaración) no puede cambiar.

- var se utiliza para la inferencia de tipos, no es una palabra reservada en el sentido tradicional.

19. Which are true about the following code? (Choose all that apply.)

```
var num1 =Long.parseLong("100");

var num2 = Long.valueOf("100");
System.out.println(Long.max(num1, num2));
```

a. The output is `100`.
b. The output is `200`.
c. The code does not compile.
d. `num1` is a primitive.
e. `num2` is a primitive.

A. Este código convierte la cadena "100" a valores de tipo Long, tanto con parseLong como valueOf. Ambos métodos convierten la cadena "100" a un número 100 (parseLong es un método estático en la clase Long, que convierte un String en un tipo de dato primitivo, valueOf es otro método de la clase Long, que convierte una String en un objeto Long, donde, en este caso retorna una instancia de Long, que es un tipo de dato envolvente (wrapper) y no un primitivo.

D. Esta opción es correcta porque parseLong("100") convierte la cadena a un valor de tipo long primitivo y var se adapta al tipo de retorno del método.

20. Which statements about the following class are correct? (Choose all that apply.)

```
1: public class PoliceBox {
2:         String color;
3:         long age;
4:         public void PoliceBox() {
5:                 color = "blue";
6:                 age = 1200;
7:         }
8:         public static void main(String []time) {
9:                 var p = new PoliceBox();
10:        var q = new PoliceBox();
11:        p.color = "green";
12:        p.age = 1400;
13:        p = q;
14:        System.out.println("Q1="+q.color);
15:        System.out.println("Q2="+q.age);
16:        System.out.println("P1="+p.color);
17:        System.out.println("P2="+p.age);
18: } }
```

A. It prints Q1=blue.
B. It prints Q2=1200.
C. It prints P1=null.
D. It prints P2=1400.
E. Line 4 does not compile.
F. Line 12 does not compile.
G. Line 13 does not compile.

H. None of the above.

EXPLICACION:

C. En la línea 4 se define un método y no un constructor, ya que que tiene el tipo de retorno void y los constructores no deben tener tipo de retorno. Así que este método no es llamado cuando se crean instancias de PoliceBox

Por lo tanto, al no tener un constructor definido explícitamente, el compilador utiliza el constructor predeterminado que no inicializa los campos color y age así que estos valores también quedan con sus valores predeterminados. Color = null y age = 0.

21. What is the output of executing the following class?

```
1: public class Salmon {
2:         int count;
3:         { System.out.print(count+"-"); }
4:         { count++; }
5:         public Salmon() {
6:                 count = 4;
7:         System.out.print(2+"-");
8:         }
9:         public static void main(String[] args) {
10:                System.out.print(7+"-");
11:                var s = new Salmon();
12:                System.out.print(s.count+"-"); } }
```

A. 7-0-2-1-
B. 7-0-1-
C. 0-7-2-1-
D. 7-0-2-4-
E. 0-7-1-
F. The class does not compile because of line 3.
G. The class does not compile because of line 4.
H. None of the above.

EXPLICACION:

- La ejecución comienza en el método main (línea9), que imprime **7-** en la consola como primer paso.
- Luego se crea una nueva instancia de Salmon (línea 11) que esto activa un bloque de inicialización y el constructor ejecutándose en el siguiente orden:
  1. Línea 3: Se imprimme count sin haber sido inicializado, por lo que el valor predeterminado de count es o, imprime **o-**
  2. Línea 4: count se incrementa en 1, por lo que ahora count =1
- Línea 5 – 8: count se establece en 4 (línea 6) y en la línea 7 se imprime **2-**
- Por ultimo, después de ejecutar el constructor, el valor actual de s.count es 4. En la línea 12, se imprime s.count + "-", dando el resultado **4-**

22. Given the following class, which of the following lines of code can independently replace `INSERT CODE HERE` to make the code compile? (Choose all that apply.)

```
public class Price {
   public void admission()
      {INSERT CODE HERE
      System.out.print(amount);
```

```
} }
```

A. `int Amount = 0b11;`

B. `int amount = 9L;`

C. `int amount = 0xE;`

D. `int amount = 1 2.0;`

E. `double amount = 1 0 .0;`

F. `int amount = 0b101;`

G. `double amount = 9 2.1 2;`

H. `double amount = 1 2 .0 0;`

EXPLICACION:

Opción C: int amount = 0xE; es correcta porque 0xE es una notación hexadecimal válida para un valor entero.

Opción F: int amount = 0b101; es válida porque 0b101 es una notación binaria correcta para un valor entero, y es compatible con el tipo int.

Opción G: double amount = 9_2.1_2; también es correcta, ya que 9_2.1_2 es una notación válida para un valor double en Java.

23. Which statements about the following class are true? (Choose all that apply.)

```
1: public class River {
2:     int Depth = 1;
3:     float temp = 50.0;
4:     public void flow() {
5:     for (int i = 0; i < 1; i++) {
6:         int depth = 2;
7:     depth++;
8:     temp--;
9:     }
10:    System.out.println(depth);
11:    System.out.println(temp); }
12:    public static void main(String... s) {
13:        new River().flow();
14: } }
```

A. Line 3 generates a compiler error.

B. Line 6 generates a compiler error.

C. Line 7 generates a compiler error.

D. Line 10 generates a compiler error.

E. The program prints 3 on line 10.

F. The program prints 4 on line 10.

G. The program prints 50.0 on line 11.

H. The program prints 49.0 on line 11.

EXPLICACION:

A es correcta porque la declaración float temp = 50.0; en la línea 3 genera un error de compilación. En Java, un valor decimal sin sufijo (f o F) se interpreta como double, no como float. Para que esta línea compile correctamente, el valor debería declararse como 50.0f o 50.0F.

D es correcta porque la variable depth declarada en la línea 6 dentro del bucle for tiene un alcance limitado a ese bloque. Al intentar acceder a depth en la línea 10, fuera del for, se genera un error de compilación ya que depth no está definida en ese ámbito

Capítulo 2

1. Which of the following Java operators can be used with `boolean` variables? (Choose allthat apply.)

A. `==`

B. `+`

C. `--`

D. `!`

E. `%`

F. `~`

G. Cast with `(boolean)`

EXPLICACION:

Operadores en Java:
==: Compara dos valores y devuelve true si son iguales, false si son diferentes.
+: Operador de suma para números o concatenación para cadenas.
--: Decrementa el valor de una variable numérica en 1.
!: Operador lógico de negación que invierte el valor booleano.
%: Operador de módulo, que devuelve el residuo de una división.
~: Operador bit a bit de negación, solo válido para tipos enteros.
Casting (boolean): Java permite convertir valores de tipo primitivo o de objetos a boolean usando métodos específicos como Boolean.valueOf(), pero no se puede hacer un casting directo desde otros tipos.

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

```
byte apples = 5;

short oranges = 10;
_____ bananas = apples + oranges;
```

A. `int`

B. `long`

C. `boolean`

D. `double`

E. `short`

F. `byte`

EXPLICACION:

- A. int: Esta es la respuesta correcta. Cuando sumas un byte y un short en Java, el resultado de la operación es automáticamente promocionado a int
- B. long: Esta también es correcta. Aunque el tipo por defecto de la operación es int, puedes asignar ese valor a un tipo long sin necesidad de hacer un casting explícito
- D. double: Esta opción también es correcta. El tipo int puede ser asignado a un tipo double de forma implícita, ya que double es más grande y puede almacenar el valor sin pérdida de precisión.

3. What change, when applied independently, would allow the following code snippet tocompile? (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`.

EXPLICACION:

- B es correcto porque al hacer un casting de ear a int, se puede realizar la multiplicación y la asignación sin error.
- C es correcto porque cambiar el tipo de ear a short evita la incompatibilidad de tipos al multiplicar por 2.
- D es correcto porque hacer un casting de todo el resultado de 2 * ear a int permite asignarlo correctamente a la variable hearing de tipo int.
- F es correcto porque cambiar el tipo de hearing a long soluciona el problema sin necesidad de casting.

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.

EXPLICACION: La opción B es correcta porque el operador XOR (^) entre true y false da como resultado true, lo que hace que canine se iguale a true. La asignación de wolf = false cambia el valor de wolf a false, pero el resultado de la operación XOR no se ve afectado. Las otras opciones son incorrectas porque no reflejan correctamente el comportamiento de la operación XOR y la asignación de valores.

Las asignaciones devuelven el valor asignado, lo que puede afectar otras operaciones. El operador XOR devuelve true cuando los operandos son diferentes. Además, es crucial entender cómo las operaciones booleanas y las asignaciones interactúan para evitar errores lógicos.

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. `^, +, =, +=`

EXPLICACION: La opción A es correcta porque los operadores +, *, % y -- están clasificados de acuerdo con las reglas de precedencia de Java en un orden creciente o similar. En C, los operadores =, ==, y ! también siguen un orden lógico

con el operador ! teniendo mayor precedencia que == y == mayor que =. Las otras opciones no están en el orden adecuado según las reglas de precedencia de Java.

- Los operadores *, %, y + tienen precedencia similar, con los operadores de multiplicación y módulo teniendo más alta precedencia que la suma.
- Los operadores de asignación = tienen la menor precedencia
- Los operadores de comparación == y ! siguen un orden específico con ! teniendo mayor precedencia que ==.

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.

EXPLICACION: La respuesta correcta es F porque el código no compila debido a un error de incompatibilidad de tipos de retorno en la línea 9. El método addCandy intenta devolver un valor float, pero está definido para devolver un long, lo que causa el error. Las otras opciones son incorrectas porque se basan en resultados que no se alcanzan debido al error de compilación.

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`

EXPLICACION:

Operadores a nivel de bits y lógicos en Java:

- El operador & se utiliza para una evaluación completa de ambos operandos. En comparación con el operador && (que corta la evaluación cuando la primera parte es false), el & evalúa ambos lados.
- El operador && es un operador lógico corto, que solo evalúa el segundo operando si es necesario (si el primero es true).

Operadores de decremento (--):

- --ph es el operador de decremento previo, que disminuye el valor de ph antes de usarlo en la comparación.

Precedencia de operadores:

- El operador && tiene mayor precedencia que el operador &.
- Los operadores de comparación como <= tienen mayor precedencia que los operadores lógicos.

Evaluación de expresiones booleanas:

- En Java, las expresiones booleanas se evalúan de acuerdo con las reglas de los operadores lógicos. Si la primera parte de una expresión && es false, el resultado será false sin evaluar el resto de la expresión.

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

```
4: int pig = (short)4;

5: pig = pig++; //5 - 2
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.

EXPLICACION: El operador ++ postfijo incrementa el valor de la variable pero devuelve el valor original antes de la operación. Las operaciones entre tipos incompatibles, como long y double, requieren conversión explícita para evitar errores de compilación. El operador de asignación -= restará un valor del operando y lo asignará a la variable, pero solo si los tipos son compatibles.

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

The code does not compile

EXPLICACION: Las respuestas correctas son A (1), D (4), E (5). El operador ternario se usa para decidir qué valor asignar a las variables, y el operador de post-incremento se usa para imprimir el valor de b antes de incrementarlo. Las otras

opciones son incorrectas porque los valores de las variables en el código no coinciden con las respuestas propuestas debido al uso de la asignación y el incremento.

El operador ternario tiene la forma condición ? valor_si_true : valor_si_false. Si la condición es verdadera, se selecciona el valor antes de los dos puntos (:), y si es falsa, se selecciona el valor después de los dos puntos.

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.

EXPLICACION: La respuesta correcta es G (El código no compila) debido a que en la segunda línea se intenta asignar un valor de tipo int a una variable de tipo short, lo que genera un error de compilación. Las demás opciones no son correctas porque, aunque se realicen los cálculos correctamente en las demás líneas, el código no compilará debido al problema de tipo en la segunda línea.

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.

EXPLICACION: La opción correcta es D (2, 0, 5) porque, al evaluar las tres expresiones, obtenemos los resultados de 2, 0 y 5 respectivamente.

Las operaciones aritméticas se siguen según las reglas de precedencia de operadores. Las multiplicaciones se realizan antes que las operaciones de módulo, y el operador % devuelve el residuo de una división. Además, el orden de

evaluación en las expresiones se respeta de izquierda a derecha, excepto cuando se usan paréntesis, los cuales tienen mayor prioridad.

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

EXPLICACION: La opción correcta es D (post-incremento, pre-decremento) porque el operador de incremento que devuelve el valor original es el post-incremento (i++), y el operador de decremento que devuelve el nuevo valor es el pre-decremento (--i). Las otras opciones son incorrectas porque no cumplen con la descripción de los operadores.

Los operadores de incremento y decremento pueden ser post o pre. El post-incremento incrementa la variable pero devuelve su valor original, mientras que el pre-decremento decrementa la variable y devuelve el valor actualizado.

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

EXPLICACION: La opción correcta es F (verdadero-verdadero-falso) porque las operaciones lógicas y bit a bit devuelven true para goingToTheStore y goingToTheZoo, mientras que stayingHome es false debido a la negación de la operación AND. Las otras opciones son incorrectas porque no concuerdan con las evaluaciones lógicas de las variables.

& (AND bit a bit): Compara dos valores bit por bit. Devuelve true si ambos bits son 1, y false en otros casos.
^ (XOR bit a bit): Devuelve true si los bits son diferentes, y false si son iguales.
&& (AND lógico): Devuelve true solo si ambas expresiones son true.
! (Negación): Cambia el valor lógico de una expresión: de true a false, y de false a true.

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 numericvalue.

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

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

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

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

EXPLICACION: Las opciones correctas son **B, E, G** porque el operador != puede usarse para comparar objetos (aunque compara referencias), la operación de asignación devuelve el valor asignado, y el operador ! solo puede usarse con valores booleanos. Las otras opciones son incorrectas debido a que el operador == no se puede usar para comparar un booleano con un valor numérico, los operadores & y | siempre evalúan ambos lados, y 0 y false no son intercambiables en Java.

- Operador !=: Se utiliza para comprobar si dos objetos o valores no son iguales. En el caso de objetos, compara sus referencias.
- Operador ==: Se utiliza para comparar dos valores primitivos o las referencias de dos objetos. No debe usarse para comparar valores booleanos y numéricos.
- Operadores lógicos (&&, &, !, ||): Se usan para combinar condiciones booleanas y realizar operaciones lógicas.
- Operación de asignación: En Java, las operaciones de asignación devuelven el valor asignado, lo que permite usar la asignación dentro de expresiones.
- Tipos de datos primitivos y de referencia: 0 es un valor numérico, y false es un valor booleano, por lo que no pueden ser usados de forma intercambiable.

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

A. `=`

B. `&&`

C. `*=`

D. `? :`

E. `&`

F. `++`

G. `/`

EXPLICACION: La respuesta correcta es D (?:), porque este es el único operador que toma tres operandos. El operador ternario ?: evalúa una condición y devuelve uno de los dos valores, dependiendo de si la condición es verdadera o falsa.

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

EXPLICACION: Línea 4: float symphony = (float)((song == 1_000f) ? song * 2L : song);
Error de compilación aquí. La expresión song * 2L intenta multiplicar un double (song) por un long (2L), lo cual produce un double. Sin embargo, la condición ternaria intenta asignar el resultado a un float. En Java, asignar double a float sin conversión explícita provoca un error de compilación. Esto hace que esta línea sea incorrecta.

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.

EXPLICACION:

1. Inicialización de Variables: ticketsTaken empieza en 1, y ticketsSold comienza en 3
2. ticketsSold += 1 + ticketsTaken++; → ticketsTaken++ usa el valor actual de ticketsTaken (1) en la operación antes de incrementarlo. Entonces, 1 + ticketsTaken++ se evalúa como 1 + 1, que da 2. Luego, ticketsSold += 2 se convierte en ticketsSold = 3 + 2, lo que da ticketsSold = 5. Después de esta línea, ticketsTaken se incrementa en 1, por lo que su nuevo valor es 2.
3. ticketsTaken *= 2; → Esta línea multiplica el valor actual de ticketsTaken (que ahora es 2) por 2, dando como resultado ticketsTaken = 4.
4. ticketsSold += (long)1; → Aquí, se añade 1 a ticketsSold, convirtiéndolo a long, pero debido a que ticketsSold es int, el tipo final sigue siendo int. Esto da ticketsSold = 5 + 1, resultando en ticketsSold = 6.

Por lo tanto, al final de estas operaciones:
ticketsSold es 6.
ticketsTaken es 4.

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. `" "`

EXPLICACION: Como regla fundamental en Java, el orden de evaluación sigue una jerarquía de operadores, pero los paréntesis permiten sobrescribir esta jerarquía, haciendo que las operaciones dentro de ellos se ejecuten antes. Esto es esencial para escribir expresiones que necesiten una evaluación específica, controlando así los resultados de manera precisa.

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; //12
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.

EXPLICACION: La respuesta correcta es **B. start es -128** y **F. end es 12**.

- En la línea 3, se inicializa start con el valor 7.
- En la línea 4, end se inicializa con el valor 4.
- En la línea 5, se realiza la operación end += ++start;. Aquí, ++start incrementa start antes de utilizarlo, por lo que start pasa de 7 a 8. Luego, end se incrementa en 8 (end += 8), lo que hace que end tenga ahora el valor 12.
- En la línea 6, start se establece en (byte)(Byte.MAX_VALUE + 1). El valor de Byte.MAX_VALUE es 127, por lo que Byte.MAX_VALUE + 1 da 128. En el contexto de Java, al convertir 128 a byte, el valor se convierte en -128 debido al desbordamiento del tipo byte.

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 ternaryoperators.
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 isapplied.
D. The post-decrement operator (––) returns the value of the variable before the decrementis applied.
E. The `!` operator cannot be used on numeric values.
F. None of the above

EXPLICACION: Las opciones A, D y E son correctas. Los operadores unarios en Java, como se menciona en A, tienen precedencia sobre los operadores binarios y ternarios. En D, el operador postdecremento (--) devuelve el valor actual antes de aplicar el decremento. En E, el operador ! no puede aplicarse a valores numéricos, solo a booleanos. Las opciones B y C son incorrectas debido al mal uso de los operadores - y ++ con booleanos y al retorno del valor incrementado.

Los operadores unarios tienen una precedencia más alta que los operadores binarios y son utilizados para manipular variables y expresiones booleanas. Por ejemplo, ++ y -- cambian los valores numéricos, mientras que ! se usa solo en valores booleanos para invertirlos. Esto es importante al comprender cómo se evalúan las expresiones en Java.

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

EXPLICACION:
Evaluación de bird: La variable bird se calcula usando el operador ~, que es el operador de complemento bit a bit. El complemento bit a bit de 8 da como resultado -9. Esto se debe a que ~x equivale a -(x+1), y para 8, esto sería -(8 + 1) = -9.

- Evaluación de plane: La variable plane se calcula como -myFavoriteNumber, que es simplemente -8 porque myFavoriteNumber es 8.
- Evaluación de superman: El valor de superman se determina con el operador ternario. La condición es bird == plane, es decir, -9 == -8, lo cual es falso. Por lo tanto, superman toma el valor 10.
- Impresión de superman: La expresión --superman reduce el valor de superman en 1 antes de ser impreso, lo que resulta en 9.-
- Salida final: System.out.println(bird + "," + plane + "," + --superman); imprimirá -9,-8,9.

Capítulo 3

1. Which of the following data types can be used in a `switch` expression? (Choose all that apply.)

   A. `enum`
   B. `int`
   C. `Byte`
   D. `long`
   E. `String`
   F. `char`
   G. `var`
   H. `double`

EXPLICACION:

- Los valores de tipo enum son permitidos en switch desde Java 5, ya que el compilador puede evaluar su valor constante
- int es un tipo primitivo que se permite en switch porque es un tipo de datos de valor constante.
- Byte es una clase envolvente (wrapper) para el tipo primitivo byte, y también es válida en un switch.
- String: Desde Java 7, String es un tipo permitido en switch, lo que permite hacer comparaciones directas con valores de texto.
- char es un tipo primitivo que puede usarse en switch ya que representa valores constantes y enteros.
- var: Aunque var es una palabra clave introducida en Java 10 para inferencia de tipos, se refiere a tipos válidos que pueden inferirse de su valor en el switch, como int o String. Si var representa un tipo compatible, es una opción válida indirectamente.

long y double no son válidos en un switch porque Java solo permite tipos de datos que se pueden comparar de manera exacta y eficiente, lo que no se garantiza con long o double debido a su mayor rango y posibilidad de error de precisión en double.

2. 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");
```

   A. `Too Low`
   B. `Just Right`
   C. `Too High`
   D. A `NullPointerException` is thrown at runtime.
   E. The code will not compile because of line 7.
   F. The code will not compile because of line 8.

EXPLICACION:

Se inicializa temperature con el valor de 4. Luego, en la línea 4, se calcula humidity como -temperature + temperature * 3, lo cual es equivalente a -4 + (4 * 3) = -4 + 12 = 8. Así que humidity tendrá el valor 8. En la línea 5, el if verifica si temperature >= 4, lo cual es verdadero ya que temperature es 4. En la línea 6, otro if se evalúa con la condición humidity < 6. Dado que humidity es 8, la condición es falsa, por lo tanto, el control de flujo se va al bloque else en la línea 9 y se imprime "Just Right".

- Cuando un if no tiene llaves quiere decir que solo tiene una línea de código.

3. Which of the following data types are permitted on the right side of a for-each expression? (Choose all that apply.)

A. `Double[][]`
B. `Object`
C. `Map`
D. `List`
E. `String`
F. `char[]`
G. `Exception`
H. `Set`

EXPLICACION:
En una estructura for-each se puede iterar sobre tipos de datos que sean arreglos o colecciones que implementen la interfaz Iterable.

- Double[][]: Es un arreglo bidimensional, lo cual es válido para una expresión for-each ya que es un arreglo.
- List: Es una interfaz que extiende Collection e implementa Iterable, lo cual lo hace compatible con for-each.
- char[]: Es un arreglo de caracteres, y todos los arreglos en Java son compatibles con for-each.
- Set: Es una colección que implementa la interfaz Iterable, permitiendo que se use en una expresión for-each.
- Map: Aunque Map es una colección, no implementa directamente Iterable. Para iterar sobre un Map en un for-each, se deben obtener las entradas, claves o valores mediante métodos como entrySet(), keySet() o values().

4. What is the output of calling `printReptile(6)`? `void`

```
printReptile(int category) {
  var type = switch(category) { case
      1,2 -> "Snake";
      case 3,4 -> "Lizard"; case 5,6
      -> "Turtle"; case 7,8 ->
      "Alligator";
   };
   System.out.print(type);
}
```

A. `Snake`
B. `Lizard`
C. `Turtle`
D. `Alligator`
E. `TurtleAlligator`
F. None of the above

EXPLICACION: A. Snake, B. Lizard, C. Turtle, D. Alligator, y E. TurtleAlligator son incorrectas, ya que la estructura switch en el código dado no es capaz de devolver ninguna salida debido a la falta de un caso default, y no maneja casos fuera de los rangos establecidos, provocando así una excepción.

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

```
List<Integer> myFavoriteNumbers = new ArrayList<>();
myFavoriteNumbers.add(10); myFavoriteNumbers.add(14);
```

```
for (var a : myFavoriteNumbers) {
   System.out.print(a + ", "); break;
}

for (int b : myFavoriteNumbers) {
   continue;
   System.out.print(b + ", ");
}

for (Object c : myFavoriteNumbers) System.out.print(c + ",
   ");
```

A. It compiles and runs without issue but does not produce any output.
B. `10, 14,`
C. `10, 10, 14,`
D. `10, 10, 14, 10, 14,`
E. Exactly one line of code does not compile.
F. Exactly two lines of code do not compile.
G. Three or more lines of code do not compile.
H. The code contains an infinite loop and does not terminate.

EXPLICACION: La línea que causa el error de compilación es la segunda línea del código, donde se utiliza var dentro del bucle for-each. El uso de var en un bucle for-each es válido solo a partir de **Java 10**. En versiones anteriores de Java, no se puede usar var como tipo implícito.

6. Which statements about decision structures are true? (Choose all that apply.)
   A. A for-each loop can be executed on any Collections Framework object.
B. The body of a `while` loop is guaranteed to be executed at least once.
C. The conditional expression of a `for` loop is evaluated before the first execution of the loop body.
D. A `switch` expression that takes a `String` and assigns the result to a variable requires a `default` branch.
E. The body of a `do/while` loop is guaranteed to be executed at least once.
F. An `if` statement can have multiple corresponding `else` statements.

EXPLICACION:
- En un bucle for, la condición es evaluada antes de cada iteración, incluidas las primeras ejecuciones del cuerpo del bucle. Si la condición es falsa desde el principio, el cuerpo del bucle no se ejecutará ni una vez.
- En un bucle do/while, el cuerpo del bucle se ejecuta primero, luego se evalúa la condición. Esto asegura que el cuerpo se ejecute al menos una vez, independientemente de si la condición es verdadera o falsa al principio.

7. Assuming `weather` is a well-formed nonempty array, which code snippet, when inserted independently into the blank in the following code, prints all of the elements of `weather`? (Choose all that apply.)

```
private void print(int[] weather) { for(
   _____) {
      System.out.println(weather[i]);
   }
```

```
}
```

A. `int i=weather.length; i>0; i--`
B. `int i=0; i<=weather.length-1; ++i`
C. `var w : weather`
D. `int i=weather.length-1; i>=0; i--`
E. `int i=0, int j=3; i<weather.length; ++i`
F. `int i=0; ++i<10 && i<weather.length;`
G. None of the above

EXPLICACION:
- B. Esta es la forma típica de recorrer un arreglo. La variable i comienza en 0 (primer índice del arreglo), y el bucle continuará mientras i sea menor o igual al último índice, que es weather.length - 1. Después de cada iteración, i se incrementa con ++i.
- D. Esta opción también es válida. El bucle comienza desde el último índice del arreglo (weather.length - 1) y recorre el arreglo de atrás hacia adelante, decrementando i en cada iteración. Esto sigue siendo una forma válida de imprimir todos los elementos, pero en orden inverso.

8. What is the output of calling printType(11)?

```
31: void printType(Object o) {
32: if(o instanceof Integer bat) {
33: System.out.print("int");
34: } else if(o instanceof Integer bat && bat < 10) {
35: System.out.print("small int");
36: } else if(o instanceof Long bat || bat <= 20) { 37: System.out.print("long");
38: } default {
39: System.out.print("unknown");
40: }
41: }
```

A. int
B. small int
C. long
D. unknown
E. Nothing is printed.
F. The code contains one line that does not compile.
G. The code contains two lines that do not compile.
H. None of the above

EXPLICACION: La opción correcta es **G** porque el código contiene **dos errores de compilació,** uno al intentar declarar nuevamente la variable bat en otro else if con instanceof, y otro por usar || con instanceof, lo que causa un error de acceso a bat en el contexto actual.

9. 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);
```

A. `break BUNNY`
B. `break RABBIT`
C. `continue BUNNY`
D. `continue RABBIT`
E. `break`
F. `continue`
G. None of the above, as the code contains a compiler error.

EXPLICACION: La respuesta correcta es **B, C y E** porque estas opciones interrumpen el flujo normal de los bucles de manera que count se incrementa exactamente dos veces. break RABBIT y break salen del bucle interno prematuramente, mientras que continue BUNNY fuerza el ciclo externo a la siguiente iteración. Las demás opciones permiten que los bucles se ejecuten más veces, lo cual resulta en un valor de count mayor a 2.
El uso de etiquetas en bucles anidados, junto con break y continue, es clave para controlar el flujo en ciclos complejos.

10. Given the following method, how many lines contain compilation errors? (Choose all that apply.)

10: private DayOfWeek getWeekDay(int day, final int thursday) { 11: int otherDay = day;
12: int Sunday = 0;
13: switch(otherDay) {
14: default:
15: case 1: continue;
16: case thursday: return DayOfWeek.THURSDAY; 17: case 2,10: break;
18: case Sunday: return DayOfWeek.SUNDAY;
19: case DayOfWeek.MONDAY: return DayOfWeek.MONDAY; 20: }
21: return DayOfWeek.FRIDAY;
22: }

A. None, the code compiles without issue.
B. 1
C. 2
D. 3
E. 4
F. 5
G. 6
H. The code compiles but may produce an error at runtime.

EXPLICACION: La opción correcta es **E (4)** porque hay cuatro líneas en el código que contienen errores de compilación. Los errores se deben a que continue no se puede usar en un switch, se intenta usar un parámetro en case en lugar de una constante, y los tipos enum no se pueden mezclar con tipos int en un switch.

Los case deben usar valores constantes (final static) del mismo tipo que la variable de switch, y continue solo funciona dentro de bucles, no en switch.

11. What is the output of calling `printLocation(Animal.MAMMAL)`? `10:`

```
   class Zoo {
11:    enum Animal {BIRD, FISH, MAMMAL}
12:    void printLocation(Animal a) {
13:       long type = switch(a) {
14:          case BIRD -> 1;
15:          case FISH -> 2;
16:          case MAMMAL -> 3;
```

```
17:          default -> 4;
18:       };
19:       System.out.print(type);
20:    } }
```

A. 3
B. 4
C. 34
D. The code does not compile because of line 13.
E. The code does not compile because of line 17.
F. None of the above

EXPLICACION: La opción correcta es **A (3)**, ya que al pasar Animal.MAMMAL como argumento, el switch evalúa el caso MAMMAL y asigna el valor 3 a la variable type, que luego se imprime. Las demás opciones son incorrectas porque el switch solo ejecuta default si ningún caso coincide, y el código no tiene problemas de compilación en las líneas indicadas.

El switch con expresiones en Java 17 permite asignar valores a una variable directamente, usando flechas (->) para cada caso, lo cual es correcto y eficiente en esta implementación.

12. What is the result of the following code snippet?

```
3: int sing = 8, squawk = 2, notes = 0;
4: while(sing > squawk) {
5:         sing--;
6:         squawk += 2;
7:         notes += sing + squawk;
8: }
9: System.out.println(notes);
```

A. 11
B. 13
C. 23
D. 33
E. 50
F. The code will not compile because of line 7.

EXPLICACION: La opción **C (23)** es correcta porque el bucle suma los valores de sing y squawk en cada iteración hasta que sing ya no sea mayor que squawk. Al final, notes contiene el valor 23. Las demás opciones son incorrectas, ya que no reflejan el valor exacto acumulado al final del bucle.

Las reglas fundamentales aquí son que, en Java, el bucle while evalúa una condición antes de cada iteración. Los operadores -- y += se utilizan para ajustar valores en cada ciclo, y la condición sing > squawk determina cuándo el bucle debe detenerse.

13. 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);
```

a. 7
b. 9
c. 10
d. 11
e. 15
f. The code will not compile because of line 6.
g. The code does not compile for a different reason.

EXPLICACION: La opción **G** es correcta porque el código no compila debido a la falta de paréntesis en while en la línea 8. Las demás opciones son incorrectas, ya que, sin esta corrección, el programa no ejecuta el bucle correctamente ni imprime una salida. En un bucle do-while en Java, es obligatorio poner la condición while entre paréntesis al final de la estructura y finalizar con un punto y coma (;), de lo contrario, el compilador generará un error.

14. Which statements about the following code snippet are correct? (Choose all that apply.)

```
for(var penguin : new int[2]) System.out.println(penguin);
var ostrich = new Character[3]; for(var emu : ostrich)
System.out.println(emu);
List<Integer> parrots = new ArrayList<Integer>(); for(var macaw : parrots)
System.out.println(macaw);
```

A. The data type of penguin is Integer.
B. The data type of penguin is int.
C. The data type of emu is undefined.
D. The data type of emu is Character.
E. The data type of macaw is List.
F. The data type of macaw is Integer.
G. None of the above, as the code does not compile.

EXPLICACION:
La opción correcta es B, C, E porque penguin es de tipo int (de acuerdo con el arreglo de enteros), emu tiene el tipo inferido como Character (basado en el tipo del arreglo Character), y macaw es de tipo Integer (porque la lista contiene elementos de tipo Integer). Las otras opciones son incorrectas porque malinterpretan los tipos de las variables.

Las reglas fundamentales incluyen el uso de la palabra clave var para inferir tipos de manera implícita, la diferencia entre tipos primitivos y objetos, y la manera en que Java maneja la inferencia de tipos dentro de bucles for-each.

15. What is the result of the following code snippet?

```
final char a = 'A', e = 'E'; char
grade = 'B';
switch (grade) { default:
   case a:
   case 'B': 'C': System.out.print("great "); case
   'D': System.out.print("good "); break; case e:
   case 'F': System.out.print("not good ");
}
```

A. great
B. great good
C. good
D. not good
E. The code does not compile because the data type of one or more `case` statements does not

match the data type of the `switch` variable.

F. None of the above

EXPLICACION: La respuesta correcta es F. Ninguna de las anteriores debido a un error en la sintaxis del código, el case 'C': no está asociado a ninguna acción, lo que provoca un flujo de ejecución erróneo. Esto impide que el código funcione como se espera y causa que no se genere ninguna salida válida. El resto de las opciones son incorrectas porque el código no compila correctamente.

Las reglas fundamentales incluyen que un switch debe estar correctamente estructurado, con case que contengan bloques de código válidos, y los valores de los casos deben coincidir con el tipo de la variable que se evalúa. En este caso, el error de sintaxis impide la correcta ejecución del código.

16. Given the following array, which code snippets print the elements in reverse order from how they are declared? (Choose all that apply.)

```
char[] wolf = {'W', 'e', 'b', 'b', 'y'};
```

A.
```
int q = wolf.length; for( ; ; ) {
System.out.print(wolf[--q]); if(q==0) break;
}
```

B.
```
for(int m=wolf.length-1; m>=0; --m) System.out.print(wolf[m]);
```

C.
```
for(int z=0; z<wolf.length; z++) System.out.print(wolf[wolf.length-z]);
```

D.
```
int x = wolf.length-1;
for(int j=0; x>=0 && j==0; x--) System.out.print(wolf[x]);
```

E.
```
final int r = wolf.length; for(int w = r-1; r>-1; w = r-1)
System.out.print(wolf[w]);
```

F.
```
for(int i=wolf.length; i>0; --i) System.out.print(wolf[i]);
```

G. None of the above

EXPLICACION:

- Las opciones correctas son A, B, D porque todas implementan correctamente un ciclo que recorre el arreglo wolf en orden inverso. La opción A decremente el índice antes de acceder al arreglo, la opción B recorre el arreglo de atrás hacia adelante y la opción D hace lo mismo utilizando un ciclo for con la condición adecuada.
- Las opciones C, E y F son incorrectas debido a errores en el manejo de índices o la estructura del ciclo.

Un ciclo for que recorre un arreglo en orden inverso debe comenzar en el índice arr.length - 1 y decrementar el índice en cada iteración. Es crucial asegurar que los índices estén dentro de los límites del arreglo para evitar errores de ejecución.

17. What distinct numbers are printed when the following method is executed? (Choose all that apply.)

```
private void countAttendees() {
   int participants = 4, animals = 2, performers = -1;
   while((participants = participants+1) < 10) {}
   do {} while (animals++ <= 1);
   for( ; performers<2; performers+=2) {}
```

```
        System.out.println(participants);
        System.out.println(animals);
        System.out.println(performers);
    }
```

A. 6
B. 3
C. 4
D. 5
E. 10
F. 9
G. The code does not compile.
H. None of the above

EXPLICACION: Las respuestas correctas son B (3) y E (10), ya que después de ejecutar el código, los valores finales de las variables son: participants = 10, animals = 3 y performers = 1. Las otras opciones son incorrectas porque no reflejan los valores correctos de las variables tras los ciclos ejecutados.

El ciclo while incrementa participants hasta 10, el ciclo do-while incrementa animals a 3, y el ciclo for incrementa performers a 1. Estos son los valores que se imprimen.

18. Which statements about pattern matching and flow scoping are correct? (Choose all that apply.)

a. Pattern matching with an `if` statement is implemented using the `instance` operator.
b. Pattern matching with an `if` statement is implemented using the `instanceon` operator.
c. Pattern matching with an `if` statement is implemented using the `instanceof` operator.
d. The pattern variable cannot be accessed after the `if` statement in which it is declared.
e. Flow scoping means a pattern variable is only accessible if the compiler can discern its type.
f. Pattern matching can be used to declare a variable with an `else` statement.

EXPLICACION:
La respuesta correcta es C (La coincidencia de patrones con una sentencia if se implementa utilizando el operador instanceof) y E (El ámbito de flujo significa que una variable de patrón solo es accesible si el compilador puede discernir su tipo), ya que ambos conceptos son fundamentales en la coincidencia de patrones y el ámbito de flujo en Java.

Las otras opciones son incorrectas porque mencionan operadores o conceptos inexistentes o erróneos, como instance o instanceon, y el uso incorrecto de la coincidencia de patrones en else.

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

```
2: double iguana = 0;
3: do {
4:     int snake = 1;
5:     System.out.print(snake++ + " "); 6:
       iguana--;
7: } while (snake <= 5);
8: System.out.println(iguana);
```

A.

```
1    2  3  4  -4.0
```

B.

```
1    2  3  4  -5.0
```

C.

```
1    2  3  4  5 -4.0
```

D.

```
0    1  2  3  4 5
                -5.0
```

E. The code does not compile.

F. The code compiles but produces an infinite loop at runtime.

G. None of the above

EXPLICACION:
La respuesta correcta es E (El código no compila) porque el ciclo do-while está utilizando una variable snake declarada dentro del bloque del bucle, lo que hace que la condición nunca cambie, causando que el código no se ejecute correctamente.

Las otras opciones son incorrectas porque el código no produce las salidas esperadas debido a la forma en que la variable snake es tratada en cada iteración. Este es un error de lógica en la declaración de la variable, no un error de sintaxis.

20. Which statements, when inserted into the following blanks, allow the code to compile and run without entering an infinite loop? (Choose all that apply.)

```
4: int height = 1;
5: L1: while(height++ <10) {
6: long humidity = 12; 7: L2: do {
8: if(humidity-- % 12 == 0) ;
9: int temperature = 30;
10: L3: for( ; ; ) {
11: temperature++;
12: if(temperature>50) ;
13: }
14: } while (humidity > 4);
15: }
```

A. break L2 on line 8; continue L2 on line 12
B. continue on line 8; continue on line 12
C. break L3 on line 8; break L1 on line 12
D. continue L2 on line 8; continue L3 on line 12
E. continue L2 on line 8; continue L2 on line 12
F. None of the above, as the code contains a compiler error

EXPLICACION: El uso de continue y break permite controlar los bucles de manera efectiva. continue salta a la siguiente iteración de un bucle, mientras que break sale completamente del bucle. El bucle do-while se ejecuta al menos una vez y luego evalúa su condición, mientras que el bucle while verifica la condición antes de cada iteración.

El uso de estas sentencias en los puntos correctos asegura que no haya bucles infinitos.

21. A minimum of how many lines need to be corrected before the following method will compile?

```
21: void findZookeeper(Long id) { 22:
      System.out.print(switch(id) {
23:       case 10 -> {"Jane"}
24:       case 20 -> {yield "Lisa";};
25:       case 30 -> "Kelly";
26:       case 30 -> "Sarah";
27:       default -> "Unassigned"; 28:
      });
29: }
```

A. Zero
B. One
C. Two
D. Three
E. Four
F. Five

EXPLICACION: La opción E (Cuatro líneas) es la correcta porque hay un total de cuatro líneas que necesitan corrección para que el código compile sin errores:

- la línea 23 (eliminando el bloque de código),
- la línea 24 (cambiando el uso de yield por un valor directo),
- la línea 26 (eliminando el case duplicado)
- y la línea 28 (debido a que el código no tiene errores de compilación si las correcciones se hacen).

El switch en Java 17 permite usar una expresión con la sintaxis case -> value;. Los bloques {} no son necesarios a menos que se utilicen con yield. Además, los valores de los casos deben ser únicos y no pueden repetirse en un switch.

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

```
2: var tailFeathers = 3;
3: final var one = 1;
4: switch (tailFeathers) {
5:    case one: System.out.print(3 + " ");
6:    default: case 3: System.out.print(5 + " "); 7: }
8: while (tailFeathers > 1) {
9:    System.out.print(--tailFeathers + " "); }
```

a. 3
b. 5 1
c. 5 2
d. 3 5 1
e. 5 2 1
f. The code will not compile because of lines 3–5.
g. The code will not compile because of line 6.

EXPLICACION:
La opción E (5 2 1) es la correcta, ya que el switch imprime 5 en el case 3, luego el ciclo while imprime los valores 2 y 1 al decrementar tailFeathers. Las otras opciones son incorrectas debido a que no reflejan correctamente el orden y los valores que el código imprime.

El switch en Java permite evaluar valores y ejecutar bloques específicos de código basados en coincidencias exactas. El ciclo while sigue ejecutándose hasta que la condición es falsa. El uso de final garantiza que una variable no se pueda modificar después de su inicialización.

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

```
15: int penguin = 50, turtle = 75;
16: boolean older = penguin >= turtle;
17: if (older = true) System.out.println("Success"); 18: else
System.out.println("Failure");
19: else if(penguin != 50) System.out.println("Other");
```

a. `Success`
b. `Failure`
c. `Other`
d. The code will not compile because of line 17.
e. The code compiles but throws an exception at runtime.
f. None of the above

EXPLICACION: La opción F (Ninguna de las anteriores) es la correcta porque el código no compilará debido a un error en la línea 19, donde un else if está mal estructurado después de un else, lo que provoca que el compilador lo marque como un error de sintaxis.

Es importante usar correctamente la asignación (=) y la comparación (==). Además, la estructura de condicionales debe ser correctamente organizada, donde un else if debe seguir a un if o else if previo, no a un else.

24. Which of the following are possible data types for `friends` that would allow the code to compile? (Choose all that apply.)

```
for(var friend in friends) { System.out.println(friend);
}
```

a. `Set`
b. `Map`
c. `String`
d. `int[]`
e. `Collection`
f. `StringBuilder`
g. None of the above

EXPLICACION: La respuesta correcta es G (Ninguna de las anteriores), ya que la sintaxis for(var friend in friends) no es válida en Java. Para que el código funcione correctamente, se debe utilizar la sintaxis de un ciclo for-each estándar y aplicar correctamente los tipos Iterable, como Set o Collection, pero no Map ni StringBuilder. Las opciones a y e son incorrectas porque la sintaxis proporcionada no es válida para un for-each en Java.

Los ciclos for-each solo pueden ser utilizados con objetos que implementen la interfaz Iterable, como colecciones (Set, Collection, etc.) y arreglos. El uso incorrecto de la sintaxis for(var friend in friends) hace que el código no compile.

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

```
6: String instrument = "violin";
7: final String CELLO = "cello";
```

```
8: String viola = "viola";
9: int p = -1;
10: switch(instrument) {
11:         case "bass" : break;
12:         case CELLO : p++;
13:         default: p++;
14:         case "VIOLIN": p++;
15:         case "viola" : ++p; break;
16: }
17: System.out.print(p);
```

A. -1
B. 0
C. 1
D. 2
E. 3
F. The code does not compile.

EXPLICACION: La respuesta correcta es D (2).

- El código ejecuta el default porque no hay coincidencia con los primeros dos case (que son "bass" y "cello").
- Luego, el flujo continúa debido a la falta de break en los case de "VIOLIN" y "viola", incrementando el valor de p en 1 en cada paso. Al final, el valor de p es 2.

En un switch, el flujo de ejecución sigue el principio de "fall-through", lo que significa que, si no se encuentra un break, la ejecución continuará en los siguientes case. El valor de la variable p es incrementado sucesivamente hasta que se encuentra un break, terminando el switch.

26. 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);
```

A. `ABC`
B. `ABCABC`
C. `ABCABCABC`
D. Line 15 contains a compilation error.
E. Line 18 contains a compilation error.
F. The code compiles but never terminates at runtime.
G. The code compiles but throws a `NullPointerException` at runtime.

EXPLICACION:
La respuesta correcta es F (El código se compila pero nunca termina en tiempo de ejecución).

- El ciclo do-while entra en un ciclo infinito debido a la condición r <= 1
- Como r no se incrementa hasta después de salir del ciclo do-while, el ciclo nunca se interrumpe y el código nunca termina.

El ciclo do-while debe tener una condición de salida que cambie adecuadamente, pero aquí la condición r <= 1 siempre se cumple mientras r sea 1, lo que provoca un ciclo infinito.

27. What is printed by the following code snippet?

```
23: byte amphibian = 1;
24: String name = "Frog";
25: String color = switch(amphibian) {
26:         case 1 -> { yield "Red"; }
27:         case 2 -> { if(name.equals("Frog")) yield "Green"; }
28:         case 3 -> { yield "Purple"; }
29:         default -> throw new RuntimeException();
30: };
31: System.out.print(color);
```

A. Red
B. Green
C. Purple
D. RedPurple
E. An exception is thrown at runtime.
F. The code does not compile.

EXPLICACION: La respuesta correcta es F (El código no compila) porque en un switch conciso (case x -> {...}), no se puede usar yield. Para que el código compile, se debe usar la sintaxis completa del switch (case x: {...}), lo que permite usar yield dentro del bloque. Las otras opciones son incorrectas porque el código no llega a ejecutarse debido al error de compilación.

En Java, los switch modernos permiten usar yield solo dentro de bloques case completos (no concisos). En este caso, el código no compila porque el switch está utilizando una forma concisa incompatible con yield.

28. What is the output of calling `getFish("goldie")`? `40:`

```
   void getFish(Object fish) {
41:    if (!(fish instanceof String guppy)) 42:
          System.out.print("Eat!");
43:    else if (!(fish instanceof String guppy)) { 44:
          throw new RuntimeException();
45:    }
46:    System.out.print("Swim!");
47: }
```

A. `Eat!`
B. `Swim!`
C. `Eat!` followed by an exception.
D. `Eat!Swim!`
E. An exception is printed.
F. None of the above

EXPLICACION: La respuesta correcta es F (Ninguna de las anteriores) porque el código contiene un error lógico en la evaluación de las condiciones if y else if, lo que lleva a lanzar una excepción RuntimeException. Esto interrumpe la ejecución antes de llegar al System.out.print("Swim!").

El operador instanceof se usa para verificar el tipo de una variable, pero el flujo del código es incorrecto porque se intenta evaluar dos veces lo mismo, lo que genera una excepción que no es necesaria.

29. What is the result of the following code?

```
1: public class PrintIntegers {
2:    public static void main(String[] args) { 3:
      int y = -2;
4:       do System.out.print(++y + " "); 5:
         while(y <= 5);
6: } }
```

a. -2 -1 0 1 2 3 4 5
b. -2 -1 0 1 2 3 4
c. -1 0 1 2 3 4 5 6
d. -1 0 1 2 3 4 5
e. The code will not compile because of line 5.
f. The code contains an infinite loop and does not terminate.

EXPLICACION:
Paso a paso de las iteraciones:

1. Primera iteración: y = -2. Se incrementa a -1, y se imprime -1.
2. Segunda iteración: y = -1. Se incrementa a 0, y se imprime 0.
3. Tercera iteración: y = 0. Se incrementa a 1, y se imprime 1.
4. Cuarta iteración: y = 1. Se incrementa a 2, y se imprime 2.
5. Quinta iteración: y = 2. Se incrementa a 3, y se imprime 3.
6. Sexta iteración: y = 3. Se incrementa a 4, y se imprime 4.
7. Séptima iteración: y = 4. Se incrementa a 5, y se imprime 5.
8. Octava iteración: y = 5. Se incrementa a 6, y se imprime 6. En este punto, y es mayor que 5, por lo que la condición y <= 5 se evalúa como falsa, y el ciclo termina.

- Operador ++ (preincremento): El operador de preincremento ++y incrementa primero el valor de la variable antes de usarla. En este caso, al comienzo del ciclo do, se incrementa el valor de y antes de que se imprima.
- Un ciclo do-while garantiza que el código dentro del ciclo se ejecute al menos una vez, ya que la condición se evalúa después de cada ejecución. La condición del ciclo do-while se evalúa al final de cada iteración. Si la condición es verdadera, el ciclo continúa; si es falsa, el ciclo termina.
- El incremento se realiza antes de la impresión de la variable, lo que significa que el valor de y se incrementa y luego se imprime.