

**Nombre:** Juan Ivan Velazquez Cabello

**Semana:** Tarea semana 1

## Indice

<b>Indice.....</b>	<b>1</b>
<b>Chapter 1 ■ Building Blocks.....</b>	<b>2</b>
<b>Chapter 2 ■ Operators.....</b>	<b>21</b>
<b>Chapter 3 ■ Making Decisions.....</b>	<b>33</b>

# Chapter 1 ■ Building Blocks

1. Which of the following are legal entry point methods that can be run from the command line? (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)

**Opción D, E:** El método principal debe ser public para que la JVM pueda acceder a él desde fuera de la clase, además debe ser static ya que main se ejecuta sin instanciar la clase que lo contiene, el tipo de retorno debe ser void porque no retorna ningún valor, además el nombre debe de llamarse main exactamente con todas en minúsculas porque la JVM busca el nombre en específico para iniciar el programa. El parámetro debe aceptar un array de String y la declaración típica es String[] args aunque el nombre del parámetro puede ser cualquiera nombre válido, también es válido declararlo como String... args main(String... args) exactamente con 3 puntos además el orden de static y public pueden ir como sea ya sea primero static y luego public o primero public y luego static pero el void siempre debe ir al lado del main.

2. Which answer options represent the order in which the following statements can be assembled 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

**Opción C, D, E:** Las respuestas son correctas porque siempre debe ir al principio el paquete después el import y al final la clase, si no se lleva esta estructura no se puede compilar el programa, aunque pueden venir solo el paquete y la clase o solo la importación y 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.

**Opción A, E:** Las respuestas son correctas porque Bunny es una clase y se ve claramente en la primera línea, además esta crea una variable de referencia llamada bun, la variable de referencia se identifica porque apunta hacia un objeto que en este caso es 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\_

**Opción B, E, G:** Las variables pueden iniciar con un guión bajo y terminar con el mismo, también pueden iniciar Public porque al tener la mayúscula no es una palabra reservada ya que las palabras reservadas no se pueden usar para nombres de variables.

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.

**Opción A, D, F:** Garbage collector solo se puede llevar a los objetos cuando no están siendo referenciados, los que ya perdieron una referencia o cuando el objeto está inaccesible.

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 = "";
```

---

56 Chapter 1 • Building Blocks

```
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

**Opción F:** Es la respuesta porque todas estas variables están al alcance de la línea 14, ya que las variables de instancia de clase pueden ser usadas donde sea y las variables declaradas dentro del método también pueden ser usadas dentro del mismo método

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.
- F. The code does not compile.

**Opción C, E:** Ya que una salida con triple comilla no puedes concatenar valores de variables, además si se deja un espacio en blanco de una línea no la respeta a menos que haya más líneas arriba y la otra sea la que está más adentro.

8. Which of the following code snippets about var compile without issue when used in a method? (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;

**Opción B, D, E, H:** Var no se puede declarar como null porque es una inferencia de tipo y necesita un valor para poder ser creado, también no se pueden declarar varias var en una sola línea, tampoco se puede declarar un var sin inicializar, pero si se puede declarar pero si puedes declarar una línea y luego reasignarle su valor.

- 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.

**Opción E:** Las variables locales no tienen un valor por default y las variables tipo float deben de tener un 0.0 no un 0, además los primitivos no tienen valores nulos predeterminados a diferencia de las variables de clase estas solo tienen valores nulos.

- 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

**Opción A, E, F:** Se puede colocar un guión bajo (\_) en cualquier literal numérico, siempre que no esté al principio, al final o al lado de un punto decimal. Los guiones bajos pueden incluso colocarse uno al lado del otro. Por estos motivos, las opciones A, E y F son correctas. Las opciones B y D son incorrectas ya que el guión bajo (\_) está al lado de un punto decimal (.). Las opciones C y G son incorrectas porque no se puede colocar un guión bajo (\_) al principio o al final del literal.

- 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 { }
```

58 Chapter 1 • Building Blocks

```
// 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

**Opción E:** Las dos primeras importaciones se pueden eliminar porque java.lang se importa automáticamente. Las dos importaciones siguientes se pueden eliminar porque Tank y Water están en el mismo paquete, lo que hace que la opción correcta sea E. Si Tank y Water estuvieran en paquetes diferentes, se podría eliminar exactamente una de estas dos importaciones. En ese caso, la respuesta sería la opción D.

- 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.

**Opción A, C, D:** La línea 2 no se compila porque solo se debe especificar un tipo, lo que hace que la opción A sea correcta. La línea 3 se compila sin problemas porque declara una variable local dentro de un inicializador de instancia que nunca se usa. La línea 4 no se compila porque Java no admite la configuración de valores de parámetros de método predeterminados, lo que hace que la opción C sea correcta. Finalmente, la línea 7 no se compila porque fins está dentro del alcance y es accesible sólo dentro del inicializador de instancia en la línea 3, lo que hace que la opción D sea correcta.

- 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;`  
C. `import aquarium.*;`  
D. `import aquarium.*;`  
E. `import aquarium.Water;`  
F. None of these imports can make the code compile.

**Opción A, B, C:** La opción A es correcta porque importa todas las clases del paquete `aquarium` incluido `aquarium.Water`. Las opciones B y C son correctas porque importan `Water` por nombre de clase. Dado que la importación por nombre de clase tiene prioridad sobre una importación `*` general, estas se compilan.

La opción D es incorrecta porque Java no sabe cuál de las dos clases de comodín `Water` debe utilizar. La opción E es incorrecta porque no se puede especificar el mismo nombre de clase en dos importaciones.

- 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();
```

---

60 Chapter 1 • Building Blocks

- 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.

**Opción A, B, D, E:** La línea 3 no se compila porque el sufijo L hace que el valor literal sea un long, que no se puede almacenar directamente dentro de un short, lo que hace que la opción A sea correcta. La línea 4 no se compila porque int es un tipo integral, pero 2.0 es un valor literal doble, lo que hace que la opción B sea correcta. La línea 5 se compila sin problemas. Las líneas 6 y 7 no se compilan porque numPets y numGrains son primitivos, y solo se pueden llamar métodos en tipos de referencia, no en valores primitivos, lo que hace que las opciones D y E sean correctas, respectivamente. Finalmente, la línea 8 se compila porque hay un método length() definido en String.

- 15.** Which of the following statements about garbage collection are correct? (Choose all that apply.)
- A. Calling `System.gc()` is guaranteed to free up memory by destroying objects eligible for 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 the program.
  - G. Marking a variable `final` means its associated object will never be garbage collected.

**Opción C, E, F:** En Java, no hay garantías sobre cuándo se ejecutará la recolección de basura por lo cual la opción A es falsa ya que la JVM es libre de ignorar las llamadas a `System.gc()` porque llamarla no significa que eliminará los objetos sin referencia. Por este motivo, las opciones A, B y D son incorrectas.

La opción C es correcta, ya que el propósito del garbage collector es recuperar la memoria usada. La opción E también es correcta porque un objeto puede ser elegible para ser eliminado por el garbage collector pero puede que nunca sea eliminado del heap que es la memoria donde se guardan los objetos.

La opción G es incorrecta, ya que marcar una variable como `final` significa que es constante dentro de su propio alcance. Por ejemplo, una variable local marcada como `final` será elegible para la recolección de elementos no utilizados después de que finalice el método, suponiendo que no existan otras referencias al objeto que existan fuera del método.

- 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.

**Opción A, C:** La opción A es correcta. Hay dos líneas. Una comienza con `squirrel` y la otra comienza con `pigeon`. Porque una barra invertida por sí sola `\` significa omitir el salto de línea. La opción D también es correcta, ya que `\s` significa mantener el espacio en blanco. En un bloque de texto, la sangría incidental se ignora, lo que hace que la opción F sea incorrecta.

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

**Opción D, F, G:** El código se compila y se ejecuta sin problemas, por lo que las opciones A y B son incorrectas. Un campo booleano se inicializa como falso, lo que hace que la opción D sea correcta y se imprima Empty = false. Las referencias a objetos se inicializan como nulas, no como la cadena vacía, por lo que la opción F es correcta y se imprime Brand = null. Por último, el valor predeterminado de los números de punto flotante es 0.0. Aunque los valores flotantes se pueden declarar con un sufijo f, no se imprimen con un sufijo f. Por estos motivos, la opción G es correcta y se imprime Code = 0.0.

**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.

**Opción D, F, G:** Una var no se puede utilizar como parámetro de método o de constructor ni como variable de clase o instancia, lo que hace que la opción A sea incorrecta y la opción C sea correcta. El tipo de una var se conoce en tiempo de compilación y no se puede cambiar en tiempo de ejecución, aunque su valor sí puede cambiar en tiempo de ejecución. Por estos motivos, las opciones B y F son correctas y la opción E es incorrecta. La opción D es incorrecta, ya que var no está permitida en declaraciones de múltiples variables. Por último, la opción G es incorrecta, ya que var no es una palabra reservada en Java.

**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.**

**Opción A, D:** Las dos primeras líneas proporcionan una forma de convertir una cadena en un número, ya que el método parseLong espera un string se envía “100” pero este convierte a número primitivo a diferencia del método valueOf este convierte a un objeto de tipo Long y también espera recibir un string. La primera es una primitiva larga y la segunda es un objeto de referencia largo, lo que hace que la opción D sea una de las respuestas. El código es correcto y el máximo es 100, que es la opción A

- 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;
```

**62** Chapter 1 • Building Blocks

```
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.

**Opción C:** Lo más importante que hay que tener en cuenta es que la línea 4 no define un constructor, sino un método llamado PoliceBox(), ya que tiene un tipo de retorno void. Este método nunca se ejecuta durante la ejecución del programa porque nunca es llamado, y a color y age se les asignan los valores predeterminados null y 0L,

Las líneas 11 y 12 cambian los valores de un objeto asociado con p, pero luego, en la línea 13, la variable p se cambia para que apunte al objeto asociado con q, que todavía tiene los valores predeterminados. Por este motivo, el programa imprime Q1=null, Q2=0, P1=null y P2=0, lo que hace que la opción C sea la única respuesta correcta.

- 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.

**Opción D:** Comenzamos con el método main(), que imprime 7- en la línea 10. A continuación, se crea una nueva instancia de Salmon en la línea 11. Esto llama a los dos bloques de inicialización de instancia en las líneas 3 y 4 para que se ejecuten en orden, estos siempre se

ejecutan cada vez que se crea una instancia de clase. El valor predeterminado de una variable de instancia de tipo int es 0, por lo que se imprime 0- a continuación y se le asigna a count un valor de 1. A continuación, se llama al constructor. Este asigna un valor de 4 a count e imprime 2-. Finalmente, la línea 12 imprime 4-, ya que ese es el valor de count. Al poner todo junto, tenemos 7-0-2-4-, lo que hace que la opción D sea la respuesta correcta.

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;

**Opción C, F, G:** Primero, 0b es el prefijo para un valor binario y 0x es el prefijo para un valor hexadecimal. Estos valores se pueden asignar a muchos tipos primitivos, incluidos int y double, lo que hace que las opciones C y F sean correctas. La opción A es incorrecta porque nombrar la variable Amount hará que la llamada System.out.print(amount) en la siguiente línea no se compile. La opción B está incorrecta porque 9L es un valor largo. Si el tipo se cambiará a long amount = 9L, entonces se complicaría. La opción D está incorrecta porque 1\_2.0 es un valor doble. Si el tipo se cambiará a double amount = 1\_2.0, entonces se complicaría. Las opciones E y H son incorrectas porque el guión bajo (\_) aparece junto al punto decimal (.), lo cual no está permitido. Finalmente, la opción G es correcta y el uso del guión bajo y la asignación son válidos.

**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:    } }
```

---

64 Chapter 1 • Building Blocks

```
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.

**Opción A, D:** El primer error del compilador está en la línea 3. La variable temp se declara como un valor flotante, pero el valor asignado es 50.0, qué es un valor doble sin el sufijo F/f. Como un valor doble no cabe dentro de un valor flotante, la línea 3 no se compila. A continuación, se declara depth dentro del bucle for y solo tiene alcance dentro de este bucle.

Por lo tanto, leer el valor en la línea 10 activa un error del compilador. Por estos motivos, las opciones A y D son las respuestas correctas.

## Chapter 2 ■ Operators

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)

**Opción A, D, G:** La opción A es el operador de igualdad y se puede utilizar en primitivas y referencias a 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 utiliza exclusivamente con valores booleanos. La opción E es el operador de módulo, que se puede utilizar solo con primitivas numéricas. La opción F es un operador de complemento bit a bit y solo se puede aplicar a valores enteros. Por último, 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

**Opción A, B, D:** La expresión apples + oranges se convierte automáticamente en int, por lo que int y los tipos de datos que se pueden convertir automáticamente de int funcionarán. Las opciones A, B y D son esos 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.**

**Opción B, C, D, F:** El código no se 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`, qué es un valor int. Las opciones B, C y D resuelven este problema al reducir 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 al aumentar 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.

**Opción B:** El código se 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 incorporado en la línea 5. La expresión `(wolf=false)` asigna el valor false a `wolf` y devuelve false para toda la expresión. Cómo `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 verdadera. 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. ^, +, =, +=

**Opción A, B:** 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 en orden de precedencia creciente ni decreciente. En la opción D, el operador de asignación (=) está entre dos operadores unarios, y el operador de multiplicación (\*) está incorrectamente en lugar del operador de mayor precedencia. 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.

**Opción B:** El código no se compila porque la línea 3 contiene un error de compilación. La conversión (int) se aplica a la fruta, no a la expresión fruta+verduras. Dado que el operador de conversión tiene una precedencia de operador más alta que el operador de suma, se aplica a la fruta, pero la expresión se promueve a un valor flotante, debido a que las verduras son flotantes. El resultado no se puede devolver como long en el método addCandy() sin una conversión. Por este motivo, la opción F es correcta. Si se agregaran paréntesis alrededor de

fruta+verduras, entonces el resultado sería 3, 5, 6 y la opción B sería correcta. Recuerde que la conversión de números de punto flotante a valores

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

**Opción D:** En la primera expresión booleana, vis es 2 y ph es 7, por lo que esta expresión se evalúa como verdadero y (verdadero || falso), lo que se reduce a verdadero. La segunda expresión booleana usa el operador condicional y, dado que (vis > 2) es falso, no se evalúa el lado derecho, dejando ph en 7. En la última asignación, ph es 7 y se aplica primero el operador de predecremento para reducir la expresión a 7 <= 6 y dar como resultado una asignación de falso. Por estos motivos, 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.

**Opción A:** Es correcta porque en la línea 5 se hace un postincremento lo que hace que primero se guarde el valor de pig que es 4 en la variable pig y luego incremente pig en 5 pero el valor 4 ya se quedó en pig.

El código se 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 post-incremento. A continuación, se le asigna este valor a la variable pig y se descarta la operación de incremento. La línea 7 solo reduce el valor de goat en 1, lo que da como resultado un resultado 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.

**Opción A, D, E:** El código se 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 post-incremento, se imprime 4, lo que hace que la opción D sea correcta. No se aplicó  $--c$ , porque solo se evaluó una de las expresiones de la derecha. En la segunda expresión,  $a!=c$  es falso ya que  $c$  nunca se modificó. Como  $b$  es 5 debido a la línea anterior y se usa el operador de post-incremento,  $b++$  devuelve 5. Luego, el resultado 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 hacer que las expresiones ternarias sean difíciles de leer. 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.

**Opción G:** El código no se compila debido a un error en la segunda línea. Aunque tanto la altura como el peso se convierten a byte, el operador de multiplicación los convierte automáticamente a int, lo que da como resultado un intento de almacenar un int en una variable short. Por este motivo, el código no se compila y la opción G es la única respuesta correcta. Esta línea contiene el único error de compilación. Un int es de 32 bits y un short es de 16 bits porque es de 8.

Aquí un ejemplo:

# Java's primitive types

Ordered by descending upper range limit

CATEGORIZATION	NAME	UPPER RANGE	LOWER RANGE	BITS
Floating point	double	Really huge positive	Really huge negative	64
	float	Huge positive	Huge negative	32
Integral	long	9,223,372,036,854,775,807	-9,223,372,036,854,775,808	64
	int	2,147,483,647	-2,147,483,648	32
	char	65,535	0	16
	short	32,767	-32,768	16
	byte	127	-128	8
Boolean	boolean	No numeric equivalent	true or false	1

SOURCE: CAMERON MCKENZIE

©2020 TECHTARGET. ALL RIGHTS RESERVED. 

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.

**Opción D:** En la línea 11 tenemos  $(2 * 4) \% 3$  pero primero se hace lo que esta entre parentesis en este caso si no hubiera paréntesis también se hace primero la multiplicación porque tiene mayor jerarquía y luego el módulo de 3 da un resultado de 2, este orden se sigue para la línea 12 primero la multiplicación y luego el módulo  $3 * 2 \% 3$ , en la línea 13 ocurre lo mismo pero primero la operación entre paréntesis que es el módulo de  $(1 \% 2)$  que da un 1 y luego se hace la multiplicación dando como resultado un 5.

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

**Opción D:** Es la única respuesta correcta porque el valor post incremento siempre incrementa después de pasar su ejecución mientras que el pre decremento decremente y asigna el valor inmediatamente pudiendo ser ocupado el nuevo valor en una misma operación donde se está ejecutando mientras que el otro post incremento necesita saltarse a la siguiente línea de ejecución para que su valor cambie.

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

**Opción F:** Aquí se está usando la precedencia de operadores que se evalúan las expresiones en un cierto orden, esto aplica a los operadores lógicos AND OR, el operador  $\wedge$ (XOR) tiene la misma precedencia que el  $\&$ (AND) y ambos son evaluados de izquierda a derecha, en este caso `boolean goingToTheStore = true & false ^ true;` aquí como ambos tienen la misma precedencia y comienza de izquierda a derecha comienza ser evaluado el `true & false` lo que da un `false` luego se evalúa el resultado con `false ^ true`, este  $\wedge$  operador devuelve un `true` si exactamente uno de los operandos es `true` y devuelve `false` si ambos operandos son `true` o ambos son `false`, se puede decir que es la inversa del AND, el resultado de `false ^ true` dará un `true` porque solo un operador es `true`.

true                  false

En `boolean goingToTheZoo = sunday && !raining;` como `raining` se está negando quiere decir que es `true` y el resultado de `true && true` es `true`

true                      true

En **boolean stayingHome = !(goingToTheStore && goingToTheZoo);** el resultado de la operación es true pero como todo esto se niega entonces sale un false.

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.

**Opción B, E, G:** El valor de retorno de una operación de asignación es el mismo que el valor de la variable recién asignada. Por este motivo, la opción A es incorrecta y la opción E es correcta.

La opción B es correcta, ya que los operadores de igualdad (`==`) y desigualdad (`!=`) pueden utilizarse con objetos. La opción C es incorrecta, ya que los tipos booleanos y numéricos 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 a menos que se use en un `instanceof` ya que si usamos `||` no se evalúa lo siguiente si no se cumple la condición. El operador `()` hará que se evalúen ambos lados.

La opción F es incorrecta, ya que Java no acepta números para valores booleanos.

Por último, la opción G es correcta, ya que se necesita utilizar 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. `/`

**Opción D:** El operador ternario es el único que puede tomar 3 valores

- 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

**Opción D:** 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 agrega al 3. Intentar almacenar un valor long en una variable int activa un error de compilación. 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, y las líneas 2 y 4 usan la conversión para hacerlo. Dado que solo una línea no se 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.

**Opción D:** 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 post-incremento, 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. La última línea de código no da un error de compilación aunque se creería que sí porque no se puede guardar un long que tiene 64 bits en un int de 32 bits, pero java permite que al usar operadores compuestos como, `+=`, `*=`, `-=` porque java realiza una conversión implícita que garantiza que el tipo de dato resultante se adapte al tipo de la variable donde se está guardando el valor, por ejemplo int value `+=` siempre el resultado se convierte a un int.

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. " "

**Opción C:** Sólo los paréntesis () pueden cambiar el orden de operación en una expresión

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.

**Opción B, F:** El código se compila y se ejecuta correctamente, por lo que las opciones G y H son incorrectas. En la línea 5, se ejecuta primero el operador de incremento previo, por lo que start se incrementa a 8 y el nuevo valor se devuelve de start es 8 y como viene con `+=` entonces quiere decir que a end le vamos a sumar su valor de end y el de start y asignarlo a end. El valor de end se calcula sumando 8 al valor original de 4, lo que deja un nuevo valor de 12 para end y hace 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 sabía el valor máximo de byte, debería haber sabido que el código se compila y se ejecuta y 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

**Opción 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 utiliza solo para valores numéricos, mientras que el operador de complemento lógico `(!)` se utiliza exclusivamente para valores booleanos. Por estos motivos, la opción B es incorrecta y la opción E es correcta. Por último, los operadores de pre-incremento/pre-decremento devuelven el nuevo valor de la variable, mientras que los operadores de post-incremento/post-decremento devuelven la variable original. Por estos motivos, 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

**Opción E:** Los operadores unarios tienen el orden de precedencia más alto, lo que hace que el complemento bit a bit 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`. Como `bird` y `plane` no son lo mismo, 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 este motivo, la opción E es correcta. Si se puede usar el operador `==` para comparar números.

# Chapter 3 ■ Making Decisions

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

**Opción A, B, C, E, F, G:** Una expresión `switch` solo admite las primitivas `int`, `byte`, `short` y `char`, junto con sus clases wrappers asociadas `Integer`, `Byte`, `Short` y `Character`, respectivamente, lo que hace que las opciones B, C y F sean correctas y descarta las opciones D y H. También admite `enum` y `String`, lo que hace que las opciones A y E sean correctas. Finalmente, `switch` admite `var` si el tipo se puede resolver en un tipo de datos `switch` compatible, lo que hace que la opción G sea correcta.

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.

**Opción B:** Es correcto porque el código compila bien además de que al hacer la operación de la línea 4 primero se hace la multiplicación seguido de la suma pero como el número a la izquierda es negativo se hace  $-4 + 12$  da un 8 o que hace que `humidity` valga 8 y al hacer el if de la línea 5 este se cumple entrando al otro if de la línea 6, en este `humidity` que vale 8 es mayor a 6 por lo que se va al else imprimiendo "Just Right"

**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**

**Opción A, C, F, H:** Es correcto porque el for each acepta todos los tipos que son iterables y los arrays todo lo que se implemente de la clase java.lang.Iterable

**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**

**Opción F:** El código no se compila porque la expresión switch requiere que se gestionen todos los valores de caso posibles, lo que hace que la opción F sea correcta. Si se agregó una declaración predeterminada válida, entonces el código se compilaría e imprimiría Turtle en tiempo de ejecución. A diferencia de las declaraciones switch tradicionales, las expresiones switch ejecutan exactamente una rama y no usan declaraciones break entre las declaraciones case. Esto quiere decir que en una expresión switch el uso de un default es obligatorio.

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.

**Opción E:** El segundo bucle for-each contiene un continue seguido de una sentencia print() y esto funciona de la manera que cuando se ejecuta continue dentro de un ciclo el flujo de ejecución salta inmediatamente al siguiente ciclo de la iteración, omitiendo cualquier código que esté después de continue, esto provocará un error de compilación llamado unreachable code.

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.

**Opción C, D, E:** Un bucle for-each se puede ejecutar en cualquier objeto Collections que implemente java.lang.Iterable, como List o Set, pero no en todas las clases Collections, como Map, por lo que la opción A es incorrecta. El cuerpo de un bucle do/while se ejecuta por lo menos una vez porque la evaluación está al final, mientras que el cuerpo de un bucle while se

ejecuta cero o más veces porque se evalúa primero la condición , lo que hace que la opción E sea correcta y la opción B incorrecta.

La expresión condicional de los bucles for se evalúa al comienzo de la ejecución del bucle, lo que significa que el bucle for puede ejecutarse cero o más veces, lo que hace que la opción C sea correcta. Una expresión switch que toma una String requiere una default predeterminado si el resultado se asigna a una variable, lo que hace que la opción D sea correcta. Finalmente, cada declaración if tiene como máximo una declaración else coincidente, lo que hace que la opción F sea incorrecta.

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

**Opción B, D:** La opción A es incorrecta porque en la primera iteración, intenta acceder a weather[weather.length] de la matriz no vacía, lo que hace que se lance una ArrayIndexOutOfBoundsException. La opción B es correcta e imprimirá los elementos en orden. La opción C no se compila porque i no está definido en weather[i]. Para que esto funcione, el cuerpo del bucle for-each también debería actualizarse. La opción D también es correcta y es una forma común de imprimir los elementos de una matriz en orden inverso. La opción E no se compila y, por lo tanto, es incorrecta porque no se pueden declarar varias variables en esa parte del ciclo. Se pueden declarar varios elementos en un bucle for, pero el tipo de datos debe enumerarse solo una vez por ejemplo for(int i =0, j= 0;), como en for(int i=0, j=3; i < 10, i++, j++) La opción F es incorrecta porque se omite el primer elemento de la matriz. Dado que la expresión condicional se comprueba antes de que se ejecute el bucle por primera vez, el primer valor de i utilizado dentro del cuerpo del bucle será 1, esto porque se usa ++i provoca que i se incrementa en 1 accediendo al segundo elemento de arreglo.

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: }
```

Review Questions      145

- 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.

**Opción F:** El código no compila porque en un if se está usando el default y eso no se puede debería ser un else.

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.

**Opción B, C, E:** El código contiene un bucle anidado y una expresión condicional que se ejecuta si la suma de col + row es un número par; de lo contrario, se incrementa el count la primera vez que la condición es verdadera es en la segunda iteración del bucle interno, cuando row es 1 y col es 1. La opción A es incorrecta porque esto hace que el bucle salga inmediatamente y que el recuento sólo se establezca en 1. Las opciones B, C y E siguen la misma ruta. Primero, el recuento se incrementa a 1 en el primer bucle interno y luego se sale del bucle interno. En la siguiente iteración del bucle externo, row es 2 y col es 0, por lo que la ejecución sale del bucle interno inmediatamente. En la tercera iteración del bucle externo, la fila es 3 y la columna es 0, por lo que el recuento se incrementa a 2. En la siguiente iteración del bucle interno, la suma es par, por lo que salimos y nuestro programa está completo, lo que hace que las opciones B, C y E sean correctas.

Las opciones D y F son incorrectas, ya que hacen que los bucles interno y externo se ejecuten múltiples veces, y que el recuento tenga un valor de 5 cuando finalice. No es necesario que realice un seguimiento de todas las iteraciones; simplemente deténgase cuando el valor de recuento supere 2.

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 continue;  
16:                 thursday: return DayOfWeek.THURSDAY;  
17:             case 2,10: break;
```

---

146 Chapter 3 • Making Decisions

```
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.

**Opción E:** El código contiene varios errores, primero la línea 15 no se puede usar un continue en un switch, línea 16 tampoco se puede usar un valor no conocido por el compilador y además si se usara en esta posición donde está el thursday debería ser final aunque en este caso si es final pero su valor no debe recibirse por parámetro porque de esta manera no se conoce en tiempo de compilación pero si se usará un valor recibido por parámetro en el switch(thursday) si

funcionará, línea 18 como se mencionó no se puede usar un valor que no sea final en este caso si se conoce el valor de la variable `sunday` pero no es final lo cual causa un error, línea 19 no se puede tener un `case` con un `enum`.

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

**Opción A:** El código se compila y se ejecuta sin problemas, imprime 3 en tiempo de ejecución y hace que la opción A sea correcta. La declaración predeterminada en la línea 17 es opcional, ya que todos los valores de enumeración se tienen en cuenta y se pueden eliminar sin cambiar porque todos los casos se tienen cubiertos, en este caso el `switch` si puede recibir un valor por medio de un parámetro del método porque se está usando para ponerlo en el `switch` `switch(a)`, pero si se intentaría usar en un `case` por ejemplo en el lugar de `case -> BIRD` de esta manera `case .> a` esto provocaría un error porque no se puede usar como un valor de `case`.

**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.

**Opción C:** Es correcto porque si se sigue el orden de ejecución en la primera iteración

1. Sing se decrementa en `sing--`; lo que da un 7 después squawk se suma a sí mismo y se le suma 2 + 2 quedando en 4, notes se suma a sí mismo que valor 0 mas lo que vale sing y squawk 7+4 quedando en 11
2. Sing decrementa a 6 y squawk se suma a sí mismo más 4 + 2 = 6 quedando en 6, después notes se suma a sí mismo más sing y squawk notes = 11 + 6+6 = 23
3. En esta iteración `sing > squawk` ya no se cumple por lo que sale del `while` quedando el valor de notes en 23.

- 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.

**Opción G:** Es correcta porque el do while en la línea 8 no tiene los paréntesis y esto no puede ir así causaría un error de compilación.

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);
```

---

48 Chapter 3 ▪ Making Decisions

- 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.

**Opción B, D, F:** El código se compila ya que si se pueden crear arrays de primitivos (`int`, `char`, `double`) y objetos también (`String`, `List`) la única diferencia es que los primitivos guardan el valor directamente no en memoria y los objetos almacenan la referencia en memoria.

- 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

**Opción F:** El código no se compila ya que el segundo `case` contiene una sintaxis que no es válida, no se puede poner un `case` seguido de otro `case` en la misma línea de esta manera `case 'B': 'C':`, para ponerlo de forma correcta debería ser `case 'B', 'C':` de esta forma si es sintaxis válida.

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

**Opción A, B, D:** Son correctas porque la A recorre el arreglo guardando la longitud del arreglo entonces guarda un 5, cuando el arreglo tiene 5 posiciones es incorrecto, pero en el print del for

se decrementa con un predecremento lo que hace que no acceda a la 5 que no existe si no a la 4 que es el ultimo elemento y asi se va recorriendo hasta llegar a 0.

La opción C no es correcta porque se está imprimiendo la longitud del arreglo - z que valor 0 esto es 5 - 0 da un 5 lo que provocaría un error.

La opción B es correcta porque se imprime la posición del arreglo en m que vale 4 porque al momento de declararlo se le asignó a m la longitud del arreglo que es 5 - 1 que sería 4 la última posición del arreglo y esta el for lo va decrementando hasta llegar a 0.

La opción D es correcta porque ocurre lo mismo que la B en la variable x se guarda  $5-1 = 4$  y esta se va imprimiendo y decrementando hasta llegar a 0 aunque el for tenga  $j==0$  como condición && siempre se cumple porque j es igual a 0 y nunca se incrementa haciendolo correcto.

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

**Opción B, E:** El código se compila sin problemas e imprime dos números distintos en tiempo de ejecución, por lo que las opciones G y H son incorrectas. El primer bucle se ejecuta un total de cinco veces y el bucle finaliza cuando los participantes tienen un valor de 10, porque la primera vez es  $4 + 1$  son 5 y la segunda solo se le suma 1 al valor obtenido por ejemplo  $5 + 1$  son 6. Por este motivo, la opción E es correcta. En el segundo bucle, los animales comienzan con un valor no menor o igual a 1, pero como es un bucle do/while, se ejecuta al menos una vez. De esta manera, los animales toman un valor de 3 y el bucle termina, lo que hace que la opción B sea correcta. Finalmente, el último bucle se ejecuta un total de dos veces, con los

artistas comenzando con -1, pasando a 1 al final del primer bucle y luego terminando con un valor de 3 después del segundo bucle, lo que interrumpe el bucle. Esto hace que la opción B sea una respuesta correcta dos veces.

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.

**Opción C, E:** El patrón de coincidencia se usa con el `instanceof` más el operador que puede ser el tipo de objeto que queremos comparar por ejemplo `instanceof Integer`.

La opción D es incorrecta ya que es posible acceder a una variable de patrón fuera de la declaración `if` en la que está definida. La opción E es una declaración correcta sobre el alcance del flujo. La opción F es incorrecta. La coincidencia de patrones no admite la declaración de variables en declaraciones `else` ya que las declaraciones `else` no tienen una expresión booleana.

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

**Opción E:** El código no compila porque la variable `snake` que está en la comparativa del `while(snake <= 5)` no está al alcance porque esta es declarada en el `do` y solo puede ser usada en ese fragmento de código.

- 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

**Opción A, E:** Lo más importante que hay que tener en cuenta al leer este código es que el bucle más interno es un bucle infinito. Por lo tanto, se buscan soluciones que salten el bucle más interno por completo o que salgan de ese bucle. La opción A es correcta, ya que break L2 en la línea 8 hace que el segundo bucle interno salga cada vez que se ingresa, saltando el bucle más interno por completo. Para la opción B, el primer continue en la línea 8 hace que la ejecución salte el bucle más interno en la primera iteración del segundo bucle, pero no en la segunda iteración del segundo bucle. El bucle más interno se ejecuta y, con continue en la línea 12, produce un bucle infinito en tiempo de ejecución, lo que hace que la opción B sea incorrecta. La opción C es incorrecta porque contiene un error del compilador. La etiqueta L3 no es visible fuera de su bucle. La opción D es incorrecta, ya que es equivalente a la opción B, ya que los break y continue sin etiquetar se aplican al bucle más cercano y, por lo tanto, producen un bucle infinito en tiempo de ejecución. Al igual que la opción A, la continuación L2 en la línea 8 permite que el bucle más interno se ejecute la segunda vez que se llama al segundo bucle. Sin embargo, la continuación L2 en la línea 12 sale del bucle infinito,

lo que hace que el control regrese al segundo bucle. Dado que el primer y el segundo bucle terminan, el código termina y la opción E es una respuesta correcta.

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

**Opción E:** La línea 22 no se compila porque Long no es un tipo compatible para una declaración switch o expresión. La línea 23 no se compila porque le falta un punto y coma después de "Jane" y una declaración yield debería haber quedado así {yield "Jane";}. La línea 24 no se compila porque contiene un punto y coma adicional al final. Por último, las líneas 25 y 26 no se compilan porque utilizan el mismo valor de caso. Al menos una de ellas debería modificarse para que el código se compile. Dado que se deben corregir cuatro líneas, la opción E es correcta, en un switch normal o tradicional no lleva al final de todo un punto y como después de las llaves }; este solo lo lleva una expresión 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.

**Opción E:** El código se compila sin problemas, lo que hace que las opciones F y G sean incorrectas, se podría pensar que en un default no se puede agregar así default: case 3: pero esto es correcto además, var es compatible con los bucles switch y while, siempre que el compilador determine que el tipo es compatible con estas instrucciones. Además, la variable one está permitida en una instrucción case porque es una variable local final, lo que la convierte en una constante de tiempo de compilación. El valor de tailFeathers es 3, que coincide con la segunda instrucción case, lo que hace que 5 sea la primera salida. El bucle while se ejecuta dos veces, con el operador de preincremento (--) modificando el valor de tailFeathers de 3 a 2 y luego a 1 en el segundo bucle. Por este motivo, la salida final es 5 2 1, lo que hace que la opción E sea la respuesta correcta.

- 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

**Opción F:** El código no compila porque en la línea 19 hay un else sin previamente un if, no puede haber un else sin un if porque la estructura va if else si no se cumple algo entonces hace lo demás, aunque sí puede haber un if sin else pero solo la primera vez al poner lo que haría en el if y luego en otro línea din indent poner lo que no haría si no se cumple la condición pero si viene un if con else y luego otro else esto si causa un error de compilación.

- 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

**Opción G:** No se puede usar un for each de esa manera con el in para iterar en java 17 solo se permite usar : en lugar del in quedando así for(var friend : friends) en vez de for(var friend in friends), si no tuviera este error si aceptaría Set, int y Collection. Porque para usar String no se usa esa estructura si no esta for(var friend : text.toCharArray()) y para usar y StringBuilder no se puede porque no implementa un iterable para iterar sobre él se debe convertir el contenido en un String y luego iterar sobre los caracteres así : for(var c: variable.toString().toCharArray())

1. Se puede usar for each con String si se convierte en un array de caracteres
2. No se puede iterar directamente sobre un StringBuilder sin convertilo a un String
3. Se usa siempre : en lugar del in en el for each

**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.

**Opción D:** El código se compila sin problemas, por lo que la opción F está incorrecta. La variable viola creada en la línea 8 nunca se utiliza y se puede ignorar. Si se hubiera utilizado como valor de caso en la línea 15, habría causado un error de compilación ya que no está marcada como final. Ahora como "violin" y "VIOLIN" no son una coincidencia exacta porque una es con mayúsculas y la otra con minúsculas esto hace que no entre al case "VIOLIN", así que el default predeterminado de la instrucción switch se ejecuta en tiempo de ejecución. Esta ruta de ejecución incrementa p un total de tres veces, lo que lleva el valor final de p a 2 y hace que la opción D sea la respuesta correcta.

**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.

**Opción F:** El código compila y ejecuta el while asignado a name una A y luego entra al do while aqui esta el problema porque el do while tiene esta condición while( $r \leq 1$ ) y r nunca se incrementa dentro del do while lo que provoca un ciclo infinito al no salir de este ciclo.

**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);
```

154 Chapter 3 ▪ Making Decisions

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

**Opción F:** La línea 27 no se compila porque el bloque case no genera un valor si el nombre no es igual a Frog. Por este motivo, la opción F es correcta. Cada ruta dentro de un bloque case debe generar un valor si se espera que la expresión switch devuelva un valor.

**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

**Opción F:** Según el alcance del flujo, `guppy` está dentro del alcance después de las líneas 41 y 42 si el tipo no es una cadena. En este caso, la línea 43 declara una variable `guppy` que es un duplicado de la variable local definida previamente en la línea 41. Por este motivo, el código no se compila y la opción F es correcta. Si se utilizó un nombre de variable diferente en la línea 43, entonces el código se compilaría e imprimiría `Swim!` en tiempo de ejecución con la entrada especificada.

**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.

**Opción F:** Es correcto porque en la línea 4 se está imprimiendo con el preincremento lo que provoca que siempre se imprima un valor después del anterior haciendo que en lugar de llegar hasta 5 por el `while(z <= 5)` aun 5 es verdadero por lo que se repite e imprime un 6.