Capitulo 3
Certificación de Java 17

Tomando decisiones

1.	A,B,C,E,F,G. Los tipos de datos que pueden ser utilizados en una expresión switch, son: • los enum. • los primitivos: byte, short, char, int (y sus versiones wrappers) • String • var, sólo si puede inferir el tipo No acepta long, doble, float, ni boolean.
2.	B. int temperature = 4; / / humidity = (-4) + (4*3) / / humidity = 8 long humidity = -temperature + temperature * 3; / / true, sí continua y entra al siguiente if if (temperature>=4) / / false, entonces se va al else **“Just Right”** if (humidity < 6) System.out.println("Too Low"); else System.out.println("Just Right"); else System.out.println("Too High");
3.	A, D, F, H. Los tipos de datos permitidos en un **for-each** son: • todos los arreglos, en este caso, **Double [ ][ ]** y **char [ ]** • cualquier collection que implemente iterable, como **List** o **Set.** • Map, Object, String, Exeption no, porque no implementan iterable
4.	NO ES NECESARIO CONTESTAR.

5.

E.

ArrayList sí implementa iterable, por lo que puede usarse en un for .

```
List<Integer> myFavoriteNumbers = new ArrayList<>();
myFavoriteNumbers.add(10);
myFavoriteNumbers.add(14);
for (var a : myFavoriteNumbers) { // var es un Integer
System.out.print(a + ", ");
 break;
}
for (int b : myFavoriteNumbers) {
 continue; / /error. Este bucle tiene el error. El continue hace que el bucle salte inmediatamente a la siguiente iteración, por lo que la línea del print nunca se ejecutará.
 System.out.print(b + ", ");
}
for (Object c : myFavoriteNumbers)
 System.out.print(c + ", ");
```

6.

C,D, E.

A. Falso. Un for-each o puede ser ejecutado por cualquier colección, únicamente por las que implementan iterable.

B. Falso. Un while puede no ejecutarse si la condición es false.

C. Verdadero. Antes de entrar a un for, se evalúa la condición, si la condición es for, no entraría es éste.

D. Verdadero, ya que resultaría imposible tener como cases todos los Strings, por lo que necesitaremos un default.

E. Verdadero. A diferencia del while, el do/while siempre se ejecutará al menos una vez.

F. Falso. Un if, tendrá como máximo un else.

7.

B, D.

Para imprimir todos los elementos del array, podríamos usar:

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

A. int i=weather.length; i>0; i-- / / weather.lenght me daría el tamaño del arreglo, pero necesitaría al menos colocarle un -1 para que entrara al último índice existente.

B. int i=0; i<=weather.length-1; ++i / / Es correcto

C. var w : weather / / éste es un for-each, es correcto, pero en el sysout, tendría que modificar lo que se imprimirá, ya que no existe i, sino w.

D. int i=weather.length-1; i>=0; i--/ / corrige lo que faltaba en la opción A, y se imprimen todos los elementos de forma descendente

E. **int i=0, int j=3**; i<weather.length; ++i / / sí podría definir dos variable al inicio, pero sin volverlo a definir, si quitara el segundo int, estaría correcto.

F. int i=0; **++i**<10 && i<weather.length; / / Como se incrementa, nunca alcazaría el indice cero

8.

NO ES NECESARIO CONTESTAR.

9.

B, C, E.

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) / / como este if no tiene llaves{}, en caso de cumpliste la condición, la acción que realiza únicamente es la siguiente y el count++ funciona como else cuando la condición no se cumple.
 _____ ; / / Opción
 count++;
 }
```

/ / breakBUNNY (opción A)

/ / count. row. col

/ / 0. 1. 0. al inicio

	/ / 1. 1. solo incrementa el for interno RABBIT porque éste aún no ha llegado a su break. 2 % 2 = 0, y aquí sí hace el break de el for mayor, rompe y ya no suma a count, count se queda en 1. **/ / breakRABBIT (opción B)** / / count. row. col / / 0. 1. 0. al inicio / / 1. 1. solo incrementa el for interno RABBIT porque éste aún no ha llegado a su break. 2 % 2 == 0, y aquí sí hace el break pero del for interno y sigue con el mayor. / / 2. 0. se vuelve a iniciar con el for interno su valor., 2 % 2 == 0 ,se cumple y vuelve a hacer break en RABBIT. / / 3. 0. 3 % 2 == 0 false e incrementa a count y sigue en el for interno porque no se rompió / / 2 1 / / 4. count se queda con valor 2 System.out.println(count); Si usara solo un break, funcionaría igual que breakRABBIT porque rompería el mismo for. continueBUNNY hará que igual regrese al for externo y corte de cierta manera con el for interno.
10.	E. 1. no puedo usar continue en un switch PRIMER ERROR 2. para usar la variable thursday en los casos, tendría que ser final y necesitaría conocer su valor. SEGUNDO ERROR 3. case 2, 10: sí es posible. Los cases pueden ser combinados con coms a partir de Java 14 4. Sunday no se puede usar porque no es final. TERCER ERROR 5. caseDayOfWeek. necesitaría evaluar un entero y éste es un Enum CUARTO ERROR.
11.	NO ES NECESARO CONITESTAR.
12.	C. 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); / / sing. squawk. notes / / 8. 2. 0 / / 7. 4 11 / / 6 6 23
13.	G. En la línea 8 **} while keepGoing;** la condición, es decir el keepGoing debería venir entra paréntesis, como no lo está, no compilaría.
14.	B, D, F, B. var en penguin tomaría como valor un int. D. var en emu tomaría como valor un Character F. var en macaw tomaría valor de Integer
15.	**E.** **case 'B': 'C': System.out.print("great ");** / / Aquí marcaría error, ya que necesitamos estableces case en cada caso, si agregáramos case o una coma para separar los casos, compilaría: **case 'B': case 'C': System.out.print("great ");** **case 'B', 'C': System.out.print("great ");** (a partir de Java 14)
16.	A,B,D. A. int q = wolf.length; for(; ;) { System.out.print(wolf[--q]); if(q==0) break; } Cuando el for está vacío (; ;) toma por default un true en su condición y entra, ya que marcamos la longitud de q, y al iniciar se imprime como --, comienza imprimiendo el último indice y continúa la iteración hasta que el índice sea cero, es decir, el primer valor. B. for(int m=wolf.length-1; m>=0; --m)

	System.out.print(wolf[m]); Funciona de forma similar al anterior, y que agarra la longitud del array menos uno, para iniciar en el último índice y continuará hasta que m sea mayor o igual a 0, es decir, hasta llegar al índice cero del primer valor. D. int x = wolf.length-1; for(int j=0; x>=0 && j==0; x--) System.out.print(wolf[x]); En este caso, j sobra ya que nunca se modifica y siempre sería cero, pero tampoco afecta ni provoca ningún error de compilación, por lo que funcionaría.
17.	B,E. private void countAttendees() { int participants = 4, animals = 2, performers = -1; while((participants = participants+1) < 10) {} / / continuará incrementando uno cada vez que itere hasta llegar a a 10 y sea false do {} while (animals++ <= 1); / / la condición será falsa pero sí incrementa una vez a animals, por lo que animals es 3 for(; performers<2; performers+=2) {} / / -1 / / 1 / / 3 System.out.println(participants); / / 4 System.out.println(animals); / / 3 System.out.println(performers); / / 3
18.	NO ES NECESARIO CONTESTAR.
19.	E snake vive dentro del bloque de código en el que fue declarado, es decir, sólo en el bloque de do { }, por lo que cuando salimos del bloque, while no la podría usar, por tanto, no compilaría.
20.	A,E. 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** **E. continue L2 on line 8; continue L2 on line 12**
21.	NO ES NECESARIO CONTESTAR.
22.	E. var tailFeathers = 3; final var one = 1; switch (tailFeathers) { case one: System.out.print(3 + " "); default: case 3: System.out.print(5 + " "); / / el valor de tailFeathers es 3, por lo que se imprimiría 5 7: } while (tailFeathers > 1) { System.out.print(--tailFeathers + " "); } ?????????????
23.	D. No compila porque no sigue el orden correcto que sería if {} else if{} else {} .
24.	G. for(var friend in friends) { / / No compila porque no sigue la sintaxis del for-each, necesitaría dos puntos en lugar del in.

	System.out.println(friend); }
25.	D Como instrument refiere a “violin” y no es ninguno de los casos establecidos, se va a default, y aumenta en 1, pero como no hay break, sigue y entra en los que tiene delante, así que incrementa dos veces más, dando como resultado 2.
26.	F Como r es menor que 1, el do while se va a volver a realizar infinitamente, nunca incrementaría r, por lo que se queda en loop infinito.
27.	NO ES NECESARIO CONTESTAR.
28.	NO ES NECESARIO CONTESTAR.
29.	C. public class PrintIntegers { public static void main(String[] args) { 3: int y = -2; / / y comienza en -2, por lo que al entrar al primer loop, tomará valor de -1, y así seguirá incrementando, hasta que se cumpla el while, en que y=5, entrará al último loop y sumará uno, quedando en 6. do System.out.print(++y + " "); while(y <= 5); } } **/ / resultado : C. -1 0 1 2 3 4 5 6**

Notas:

```
for ( ; ; ) {
    System.out.println(“Hello”);
}
```

Cuando tenemos un for, cuyos valores no fueron definidos, éste no lanzará error, por el contrario, la condición se tomará por default como true y entrará al ciclo, iterando

infinitamente. Sin embargo, ésta sería una mala práctica.