

# Capítulo 4

## Certificación de Java 17

Core APIs



1.

F.

No compila.

```
1: public class Fish {
```

```
2: public static void main(String[] args) {
```

```
3: int numFish = 4;
```

```
4: String fishType = "tuna";
```

```
5: String anotherFish = numFish + 1;
```

```
6: System.out.println(anotherFish + " " + fishType); 7: System.out.println(numFish +  
" " + 1);
```

```
8: } }
```

En la línea 5, no podríamos asignar un int a un String, lanzaría error de compilación. Si concatenamos "" podríamos tener los siguientes resultados.

String anotherFish = numFish + 1 + ""; //primero haría la suma y luego lo volvería String. Resultado = 5

String anotherFish = "" + numFish + 1; //ya no haría la suma, concatena el 41. Resultado = 41

2.

C,E,F.

Los arreglos en Java son de tamaño fijo, por lo que al inicializarlos es importante definir su tamaño.

Es indispensable establecer la posición del primer elemento, el segundo podría ser asignado después.

**A. int[][] scores = new int[5][]; //sí es válido**

[][] indica que es un arreglo bidimensional

int[5][] es necesario definir la posición del primer elemento, de esta manera, establezco que tendré 5 filas, (0,1,2,3,4). Si no tuviese el 5, sería inválido.

Sin embargo, en la opción A, como está establecida, únicamente estamos creando el objeto de filas, pero no el objeto de columnas, por lo que el siguiente código

generaría NullPointerException.

```
public class Question02 {  
    public static void main(String[] args) {  
        int[][] scores = new int[5][];  
  
        scores[0][0] = 1;  
        scores[0][1] = 2;  
        scores[0][2] = 3;  
        scores[0][3] = 4;  
  
        scores[1][0] = 5;  
        scores[1][1] = 6;  
  
        scores[2][0] = 7;  
  
        scores[3][0] = 8;  
        scores[3][1] = 9;  
        scores[3][1] = 10;  
  
        System.out.println(scores);  
    }  
}
```

Necesitaría definir los valores de cuántas columnas habrá en cada fila:

```
int[][] scores = new int[5][];
```

```
scores[0] = new int[4];  
scores[0][0] = 1;  
scores[0][1] = 2;  
scores[0][2] = 3;  
scores[0][3] = 4;
```

```
scores[1] = new int[2];  
scores[1][0] = 5;  
scores[1][1] = 6;
```

```
scores[2] = new int[1];  
scores[2][0] = 7;
```

```
scores[3] = new int[3];  
scores[3][0] = 8;  
scores[3][1] = 9;  
scores[3][2] = 10;
```

```
scores[4] = new int[20];
```

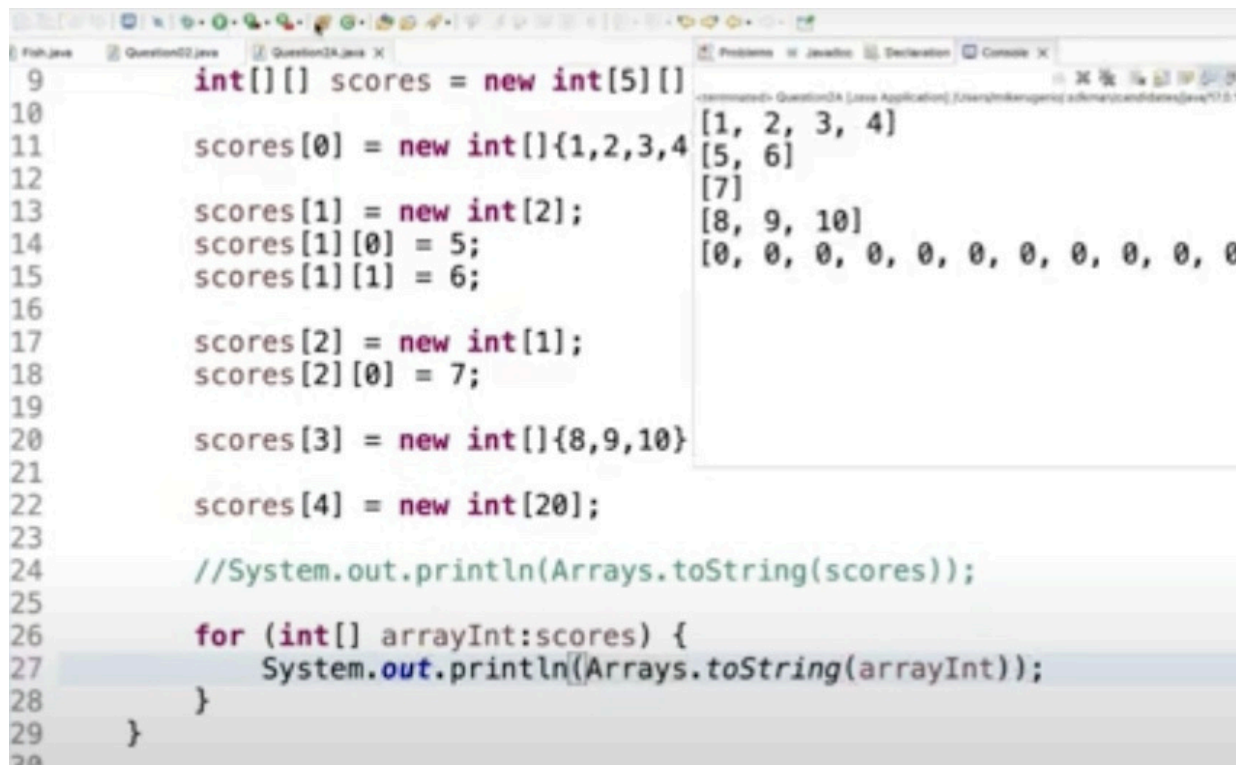
Otra manera de declarar las variables sería la siguiente:

```
scores[0] = new int[]{1,2,3,4};
```

Sin embargo, en ambos casos, **System.out.println(scores);** imprimiría el espacio en memoria al que apunta el array.

Para imprimir los valores podríamos usar un for-each. Para que itere scores y me regrese un arreglo de enteros.

```
9      int[][] scores = new int[5][]
10
11      scores[0] = new int[]{1,2,3,4}
12
13      scores[1] = new int[2];
14      scores[1][0] = 5;
15      scores[1][1] = 6;
16
17      scores[2] = new int[1];
18      scores[2][0] = 7;
19
20      scores[3] = new int[]{8,9,10}
21
22      scores[4] = new int[20];
23
24      //System.out.println(Arrays.toString(scores));
25
26      for (int[] arrayInt:scores) {
27          System.out.println(Arrays.toString(arrayInt));
28      }
29  }
```

The screenshot shows a Java IDE with a code editor on the left and a console on the right. The code in the editor defines a 2D array 'scores' of type 'int[]'. It initializes five rows: scores[0] with 4 elements, scores[1] with 2 elements, scores[2] with 1 element, scores[3] with 3 elements, and scores[4] with 20 elements. A loop prints each row. The console on the right shows the output of the program, displaying each row's contents: [1, 2, 3, 4], [5, 6], [7], [8, 9, 10], and [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0].

**B. Object[][][] cubbies = new Object[3][0][5];**

Declara un arreglo tridimensional.

Hay que tomar en cuenta que, cuando se define que habrá cero elementos, no podría existir ni siquiera el índice cero, pues éste ya contaría como el primer elemento, por lo que arrojaría un `NullPointerException`.

```
int[] arrayx = new int[0];
```

The screenshot shows a single line of Java code: 'int[] arrayx = new int[0];'. The code is highlighted in a light blue background.

**3.**

A,C,D.

**A. LocalDate.of(2022, 3, 13)**

Es correcta (año, mes, día).

**B. LocalDate.of(2022, 3, 40)**

Es incorrecta porque el día 40 no existe y está fuera del rango existente. Muestra un `nullException`

**C. LocalDate.of(2022, 11, 6)**

Es correcta.

**D. LocalDate.of(2022, 11, 7)**

Es correcta.

**E. `LocalDate.of(2023, 2, 29)`**

Es incorrecta porque no es año bisiesto.

**F. `LocalDate.of(2022, MonthEnum.MARCH, 13);`**

Es correcto, sólo estoy obteniendo el mes de un enum.

**4.**

A,C,D.

Which of the following are output by this code? (Choose all that apply.)

3: `var s = "Hello";`

4: `var t = new String(s);`

5: `if ("Hello".equals(s)) System.out.println("one");` // es verdadero porque como son Strings, el equals hace override y revisa si tienen el mismo contenido.

6: `if (t == s) System.out.println("two");` // falso, no apuntan al mismo objeto porque s está dentro del pool de Strings y t está fuera

7: `if (t.intern() == s) System.out.println("three");` // es verdadero porque con intern lo metemos al pool de Strings, pero esto sólo es temporalmente.

8: `if ("Hello" == s) System.out.println("four");` // es verdadero porque ambos están dentro del pool de Strings y apuntan al mismo,

9: `if ("Hello".intern() == t) System.out.println("five");` // es falso porque "Hello" ya había nacido dentro del pool de Strings, no era necesario el .intern(), pero t sigue fuera del pool.

**5.**

B.

Cuando tengo tantos insert seguidos, primero resuelvo el primero antes de continuar con el siguiente. //abbacca

`sb = sb.append` no sería necesario asignar porque `StringBuilder` es mutable, igual se asigna sin el `sb =`

**7: `var sb = new StringBuilder();`**

**8: `sb.append("aaa").insert(1, "bb").insert(4, "ccc");` 9: `System.out.println(sb);`**

// aaa

// abbaa inserto en el índice 1

// abbaccca

C.

<p><b>6.</b></p>	<p><b>6. How many of these lines contain a compiler error? (Choose all that apply.)</b></p> <p><b>23:</b> <code>double one = Math.pow(1, 2);</code> // Es correcta. Al método pow se le pasan dos int y regresa un double.</p> <p><b>24:</b> <code>int two = Math.round(1.0);</code> // Error de compilación. Math.round regresa un long, por lo que no puedo poner al inicio que me regresará un int.</p> <p><b>25:</b> <code>float three = Math.random();</code> // Error de compilación. Math.random regresa un double, por lo que no puedo poner al inicio que me regresará un float.</p> <p><b>26:</b> <code>var doubles = new double[] {one, two, three};</code> // Es correcta. Es un array de doubles. Incluso si two es long, cabría en un double.</p>
<p><b>7.</b></p>	<p><b>A,E.</b></p> <p><b>7. Which of these statements is true of the two values? (Choose all that apply.)</b></p> <p><b>2022-08-28T05:00 GMT-04:00</b> // le sumo 4 horas a la hora, es decir que son las 9:00. (cuando trae un signo negativo se suma y cuando trae un signo positivo, se resta)</p> <p><b>2022-08-28T09:00 GMT-06:00</b> // le sumo 6 horas a la hora, es decir que son las 15:00.</p>
<p><b>8.</b></p>	<p><b>A,B,F.</b></p> <p>Which of the following return 5 when run independently? (Choose all that apply.)</p> <p><b>var string = "12345";</b></p> <p><b>var builder = new StringBuilder("12345");</b></p> <p><b>A. builder.charAt(4)</b> // se va al caracter del índice 4, que en efecto es 5.</p> <p><b>B. builder.replace(2, 4, "6").charAt(3)</b> // toma builder y hace un replace de la posición del índice 2 hasta antes del índice 4 (es decir, que éste es exclusive y no lo reemplaza) y reemplaza con "6". Después se va al índice 3 que es 5.</p> <p>// 12345</p> <p>// 1265</p> <p><b>C. builder.replace(2, 5, "6").charAt(2)</b> // establece que vaya del índice 2, hasta antes del índice 5. Incluso si el segundo número fuera mayor, en este caso específico, no lanzaría nullException, simplemente tomaría hasta el último valor.</p>

// 12345

// 126

Tomaría el valor 6.

**D. string.charAt(5)** // Error, porque no existe el índice 5. NullPointerException.

**E. string.length** // Error de compilación. length funciona como atributo (funcionaría con .length) para los arreglos, pero para String funciona como método, (necesitaría .length() para que funcionara).

**F. string.replace("123", "1").charAt(2)**

// 12345

// 145

// toma el índice 2, es decir 5.

**G. None of the above**

**9.**

A,C,F.

**A. The first element is index 0.** //true

**B. The first element is index 1.** //false

**C. Arrays are fixed size.** //true

**D. Arrays are immutable.** // false, los valores de un arreglo sí se pueden modificar

**E. Calling equals() on two different arrays containing the same primitive values always returns true.** //false porque array no sobrescribe equals, toma el comportamiento de Object porque los array son objetos, y no son el mismo objeto.

**F. Calling equals() on two different arrays containing the same primitive values always returns false.** //true

**G. Calling equals() on two different arrays containing the same primitive values can return true or false.** //false

**10.**

A.

No hay errores de compilación.



How many of these lines contain a compiler error? (Choose all that apply.)

**23: int one = Math.min(5, 3); //3**

**24: long two = Math.round(5.5); //6**

**25: double three = Math.floor(6.6); //6.0 Math.floor corta el decimal**

**26: var doubles = new double[] {one, two, three};**

Ninguna línea genera error de compilación,

- min regresa int porque los valores comparados son int
- round regresa long
- floor regresa double
- var toma el valor de double en el que cabe el int, long y double.

**11.**

E.

**var date = LocalDate.of(2022, 4, 3);**

**date.plusDays(2);**

**date.plusHours(3); // nunca definí horas, es imposible agregarle, por lo que no compila**

**System.out.println(date.getYear() + " " + date.getMonth()**

**+ " " + date.getDayOfMonth());**

**12.**

A,D,E.

What is output by the following code? (Choose all that apply.)

**var numbers = "012345678".indent(1); // var lo toma como String. .indent(1) genera espacios al inicio, en este caso un espacio en blanco.**

**numbers = numbers.stripLeading(); // vuelve a modificar numbers. .stringLeading() remueve todos los espacios que haya al inicio, le quita el espacio que agregamos antes.**

strip() o trim() son métodos que remueven espacios del inicio y el final.

**System.out.println(numbers.substring(1, 3)); // da la cadena desde el índice 1 hasta antes del índice 2 resultado = 12**

**System.out.println(numbers.substring(7, 7)); // Imprime una línea en blanco porque es exclusive, necesitaría ser 7,8 para imprimir 7.**

`System.out.print(numbers.substring(7));` // empieza en el índice 7 hasta el final.  
resultado = 78

13.

B.

```
public class Lion {  
  
    public void roar(String roar1, StringBuilder roar2) {  
  
        roar1.concat("!!!"); // no se concatena porque como es String, necesitaría ser  
        asignado porque String es inmutable.  
  
        roar2.append("!!!"); // sí se concatena porque StringBuilder es mutable  
    }  
  
    public static void main(String[] args) {  
  
        var roar1 = "roar";  
  
        var roar2 = new StringBuilder("roar"); new Lion().roar(roar1, roar2);  
  
        System.out.println(roar1 + " " + roar2); // roar roar!!!  
    }  
}
```

14.

A,F.

Given the following, which can correctly fill in the blank? (Choose all that apply.)

```
var date = LocalDate.now();  
  
var time = LocalTime.now();  
  
var dateTime = LocalDateTime.now();  
  
var zoneId = ZoneId.systemDefault();  
  
var zonedDateTime = ZonedDateTime.of(dateTime, zoneId);  
  
Instant instant = ;
```

**A. `Instant.now()`** // Nos da la hora cero

**B. `new Instant()`** // No se puede usar new porque los constructores del paquete time son privados.

**C. `date.toInstant()`** // Siempre necesita saber la zona horaria para poder ejecutarse, como en este ejemplo no se cuenta con ella, no compila. No le estoy diciendo con

respecto a qué.

**D. `dateTime.toInstant()`** // Lo mismo que la opción C. Siempre necesita saber la zona horaria para poder ejecutarse, como en este ejemplo no se cuenta con ella, no compila.

**E. `time.toInstant()`** // No le estoy diciendo con respecto a qué.

**F. `zonedDateTime.toInstant()`** // Es correcto. Nos dice la fecha y hora de la zona cero con respecto a la zona que estoy definiendo primero.

Nota: `instant` tiene que ver con la hora cero (Greenwich), si nosotros le indicamos nuestra zona horaria (con fecha y hora), nos dirá con respecto a ello, la fecha y hora que es en ese mismo momento pero en la zona cero.

15.

C,E.

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

```
var arr = new String[] { "PIG", "pig", "123"};
```

**`Arrays.sort(arr)`**; // se ordena 123, PIG, pig Ordena primero números, luego mayúsculas y luego minúsculas porque toma criterios del código ASCII. Si hubiese un null mandaría `nullException`. Si no se ordenara, el `.binarySearch()` no funcionaría correctamente.

```
System.out.println(Arrays.toString(arr));
```

**`System.out.println(Arrays.binarySearch(arr, "Pippa"))`**; // Como no existe, lo que regresa es **-3**, porque chequea dónde sería colocado si existiera (entre las mayúsculas y las minúsculas, es decir, índice 2, conviértelo a negativo y réstale 1)

Si existiera, lo colocaría después de mayúsculas y antes de minúsculas.

`binarySearch()`; // regresa el índice del elemento buscado dentro de la lista ordenada, siempre debe estar primero ordenada.

16.

A,B,G.

What is included in the output of the following code? (Choose all that apply.)

```
var base = "ewe\nsheep\t";
```

// \n es un salto de línea y \t es un tabulador pero como tiene un carácter de escape \ no lo toma como tab y lo imprime. Es decir que imprime:

ewe

sheep\t

`int length = base.length();` // la **longitud sería 11** porque `\n` se toma como 1 y `\el` caracter de escape no suma.

`int indent = base.indent(2).length();` // `.indent()` por cada línea agrega dos espacios en línea y luego hace un salto de línea. Quedaría así:

`__ewe`

`__sheep\t`

`(_)`

\*En estas notas, `_` representa un espacio vacío y `(_)` un salto de línea.

`length()` ya sería **16**

`int translate = base.translateEscapes().length();` // `translateEscapes` quita los escapes, es decir que ahora sí leería el tabulador. Si teníamos al inicio una longitud de 11, baja a **10** porque el `\t` ya solo es un tab.

`var formatted = "%s %s %s".formatted(length, indent, translate);`

`System.out.format(formatted);` // 11 16 10

**TABLE 4.2** Common formatting symbols

Symbol	Description
<code>%s</code>	Applies to any type, commonly String values
<code>%d</code>	Applies to integer values like int and long
<code>%f</code>	Applies to floating-point values like float and double
<code>%n</code>	Inserts a line break using the system-dependent line separator

The following example uses all four symbols from Table 4.2:

```
var name = "James";
var score = 98.25;
var total = 100;
System.out.println("%s:\n    Score: %f
out of %d"
    .formatted(name, score, total));
```

This prints the following:

```
James:
    Score: 98.250000 out of 100
```

Mixing data types may cause exceptions at runtime. For example, the following throws an exception because a floating-point number is used when an integer value is expected:

```
var str = "Food: %d
tons".formatted(2.0); //
IllegalFormatConversionException
```

17.

A,G.

Which of these statements are true? (Choose all that apply.)

`var letters = new StringBuilder("abcdefg");`

A. `letters.substring(1, 2)` returns a single-character String. // true, **regresa : b**

B. `letters.substring(2, 2)` returns a single-character String. // false, **no regresa nada**

- C. `letters.substring(6, 5)` returns a single-character **String**. // genera Exception
- D. `letters.substring(6, 6)` returns a single-character **String**. / false, no regresa nada
- E. `letters.substring(1, 2)` throws an exception. // false, regresa: b
- F. `letters.substring(2, 2)` throws an exception. // false, no regresa Exception, simplemente no regresa nada
- G. `letters.substring(6, 5)` throws an exception. // true, no encuentra el índice.
- H. `letters.substring(6, 6)` throws an exception. // false, sólo no regresa nada.

18.

C,F.

What is the result of the following code? (Choose all that apply.)

```
String s1 = ""
```

```
    purr"";
```

```
String s2 = "";
```

```
s1.toUpperCase();
```

```
s1.trim();
```

```
s1.substring(1, 3);
```

// hasta aquí, se han creado los objetos pero no se ha apuntado a ellos con un =, recordemos que String es immutable. Si sí hubiese sido asignado tomaría el elemento del índice 1 hasta antes del 3.

```
s1 += "two";
```

// Como s1 sigue siendo purr, traducimos esto como s1 = s1 + "two"

```
// purrtwo
```

```
s2 += 2;
```

// 2

```
s2 += 'c';
```

// 2c

```
s2 += false;
```

// 2cfalse Cuando se asigna con un operador compuesto como += NO se almacena en el pool de Strings, sería como hacer un new String()

```
if ( s2 == "2cfalse") System.out.println("==");
```

// s2 == "2cfalse" pregunta si apuntan al mismo objeto, false porque s2 no se almacenó en el pool de Strings, necesitaríamos agregar .intern()

```
if ( s2.equals("2cfalse")) System.out.println("equals");
```

// Podría comparar con el equals que reescribe el método cuando tiene Strings y sólo compara el contenido, sería true e imprimiría equals.

```
System.out.println(s1.length());
```

// 7 porque s1 = purrtwo

19.

A,B,D.

Which of the following fill in the blank to print a positive integer? (Choose all that apply.)

```
String[] s1 = { "Camel", "Peacock", "Llama"};
```

```
String[] s2 = { "Camel", "Llama", "Peacock"};
```

```
String[] s3 = { "Camel"};
```

```
String[] s4 = { "Camel", null};
```

```
System.out.println(Arrays. _____);
```

**A. compare(s1, s2)**

El primer valor es igual, pero al entrar a Peacock y Llama, la P tiene un valor más grande que la L porque viene después. El primer array es mayor que el segundo por tanto, regresa un número positivo. resultado = 4

**compare() regresa:**

- A negative number means the first array is smaller than the second.
- A zero means the arrays are equal.
- A positive number means the first array is larger than the second.

Here's an example:

```
System.out.println(Arrays.compare(new  
int[] {1}, new int[] {2}));
```

This code prints a negative number. It should be pretty intuitive that 1 is smaller than 2, making the first array smaller.

Now that you know how to compare a single value, let's look at how to compare arrays of different lengths:

- If both arrays are the same length and have the same values in each spot in the same order, return zero.
- If all the elements are the same but

the second array has extra elements at the end, return a negative number.

- If all the elements are the same, but the first array has extra elements at the end, return a positive number.
- If the first element that differs is smaller in the first array, return a negative number.
- If the first element that differs is larger in the first array, return a positive number.

Finally, what does smaller mean? Here are some more rules that apply here and

to `compareTo()`, which you see in [Chapter 8](#), "Lambdas and Functional Interfaces":

- `null` is smaller than any other value.
- For numbers, normal numeric order applies.
- For strings, one is smaller if it is a prefix of another.
- For strings/characters, numbers are smaller than letters.
- For strings/characters, uppercase is smaller than lowercase.

[Table 4.4](#) shows examples of these rules in action.

**TABLE 4.4** `Arrays.compare()` examples

First array	Second array	Result	Reason
<code>new int[] {1, 2}</code>	<code>new int[] {1}</code>	Positive number	The first element is the same, but the first array is longer.
<code>new int[] {1, 2}</code>	<code>new int[] {1, 2}</code>	Zero	Exact match

## B. `mismatch(s1, s2)`

## Using *mismatch()*

Now that you are familiar with `compare()`, it is time to learn about `mismatch()`. If the arrays are equal, `mismatch()` returns `-1`. Otherwise, it returns the first index where they differ.

20.

A,D.

Note that March 13, 2022 is the weekend that clocks spring ahead for daylight saving time. What is the output of the following? (Choose all that apply.)

```
var date = LocalDate.of(2022, Month.MARCH, 13);
```

```
var time = LocalTime.of(1, 30);
```

```
var zone = ZoneId.of("US/Eastern"); // Estamos a menos 5 horas de la zona cero
```

```
var dateTime1 = ZonedDateTime.of(date, time, zone);
```

```
var dateTime2 = dateTime1.plus(1, ChronoUnit.HOURS); // le suma una hora pero volverá a ser la 1:30 porque es horario de verano y a las 2 regresa una hora. Lo que cambiará es la diferencia de horas con respecto a la zona cero, ya sólo estará a -4 horas de diferencia.
```

```
long diff = ChronoUnit.HOURS.between(dateTime1, dateTime2); int hour =  
dateTime2.getHour(); // la diferencia entre -5 horas a -4 horas daría 1 hora de  
diferencia
```

```
boolean offset = dateTime1.getOffset()
```

```
== dateTime2.getOffset(); // getOffset() da a cuántas horas de diferencia se está  
con respecto a la zona cero, y no son iguales porque uno es -5 y otro es -4 horas por  
lo que saldría false.
```

```
System.out.println("diff = " + diff);
```

```
System.out.println("hour = " + hour);
```

```
System.out.println("offset = " + offset);
```

A. diff = 1

D. hour = 3



```
2
3 import java.time.*;
4 import java.time.temporal.ChronoUn
5
6 public class Question20 {
7
8     public static void main(String
9
10         var date = LocalDate.of(20
11         var time = LocalTime.of(1,
12         var zone = ZoneId.of("US/E
13         var dateTime1 = ZonedDateTime.of(date, time, zone,
14
15         System.out.println(dateTime1);
16         //2022-03-13T01:30-05:00[US/Eastern]
```

2022-03-13T01:30-05:00[US/Eastern]  
2022-03-13T03:30-04:00[US/Eastern]

21.

A.C.

Which of the following can fill in the blank to print avaJ? (Choose all that apply.)

3: var puzzle = new StringBuilder("Java");

4: puzzle.\_\_\_\_\_;

5: System.out.println(puzzle);

**A. reverse()** // *StringBuilder sí tiene el método reverse(), String no.*

**B. append("vaJ\$").substring(0, 4)**

*//JavavaJ\$*

*// Java*

*Si hubiese sido del (3,7) sí lo hubiese impreso al revés*

*//Nota: los substrings se generan fuera del pool de Strings*

**C. append("vaJ\$").delete(0, 3).deleteCharAt(puzzle.length() - 1)**

*//JavavaJ\$*

*//avaJ\$*

*// avaJ*

**D. append("vaJ\$").delete(0, 3).deleteCharAt(puzzle.length())**

*Como no existe el índice 5 generaría Exception*

**E. None of the above**

22.

A.

```
var date = LocalDate.of(2022, Month.APRIL, 30); // como los meses son enum,  
podría también poner 4 y pondría el 4to mes
```

```
date.plusDays(2); //agregamos 2 días
```

```
date.plusYears(3); //agregamos 2 años
```

Pero como son inmutables, saldría la misma fecha porque no ha sido asignado.

```
System.out.println(date.getYear() + " " + date.getMonth()  
+ " " + date.getDayOfMonth());
```

