Nombre: Juan Ivan Velazquez Cabello
Semana: Tarea semana 2

Chapter 4 ■ Core APIs

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

```
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: } }
```

   A. `4 1`
   B. `5`
   C. `5 tuna`
   D. `5tuna`
   E. `51tuna`
   F. The code does not compile.

Opción F: El código no compila porque la línea 5 quiere guardar un int en un String y esto provoca un error de compilación, si se hubiera hecho de esta manera: String anotherFish = numFish + 1 + “”; esto si funcionaria porque se está concatenando un string y esto lo convierte en un String pero si se hace la primera operación, es decir si suma el valor de numFish = 4 + 1 = 5 pero el cinco se convierte en String por concatenarlo al final, al principio o en medio puede venir en cualquier lugar el + “”.

2. Which of these array declarations are not legal? (Choose all that apply.)
   A. `int[][] scores = new int[5][];`
   B. `Object[][][] cubbies = new Object[3][0][5];`
   C. `String beans[] = new beans[6];`
   D. `java.util.Date[] dates[] = new java.util.Date[2][];`
   E. `int[][] types = new int[];`
   F. `int[][] java = new int[][];`

Opción C, E, F:
A. `int[][] scores = new int[5][];` ✅ LEGAL

- Es una declaración válida de un array 2D "ragged" (irregular)

- Solo se especifica la primera dimensión (5)
- Las sub-arrays pueden ser inicializadas después con diferentes tamaños, pero forzosamente deben ser inicializados si no provocara un error

B. `Object[][][] cubbies = new Object[3][0][5];` ✅ LEGAL

- Es una declaración válida de un array 3D
- Todas las dimensiones están especificadas
- Es válido tener 0 como dimensión

C. `String beans[] = new beans[6];` ❌ ILEGAL

- Error de sintaxis
- Debería ser `new String[6]`
- No se puede usar `beans` como tipo de array

D. `java.util.Date[] dates[] = new java.util.Date[2][];` ✅ LEGAL

- Es una declaración válida de un array 2D de Date
- La sintaxis alternativa con `[]` es válida
- Similar al caso A, es un array "ragged"

E. `int[][] types = new int[];` ❌ ILEGAL

- Falta especificar al menos la primera dimensión
- Debería ser algo como `new int[5][]`

F. `int[][] java = new int[][];` ❌ ILEGAL

- Igual que E, falta especificar dimensiones
- No se pueden dejar ambas dimensiones sin especificar en la inicialización

3. Note that March 13, 2022 is the weekend when we spring forward, and November 6, 2022 is when we fall back for daylight saving time. Which of the following can fill in the blank without the code throwing an exception? (Choose all that apply.)

```
var zone = ZoneId.of("US/Eastern");
var date = _____________________________;
var time = LocalTime.of(2, 15);
var z = ZonedDateTime.of(date, time, zone);
```

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

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

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

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

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

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

Opción A, C, D: Las opciónes A y C son correctas porque aunque `LocalTime.of(2, 15)` se encuentra en la hora que salta (no existe en esa zona horaria), el código **no lanzará una excepción**, pero `ZonedDateTime` ajustará la hora a las 3:15 a.m.

La opción B está incorrecta porque el 40 de marzo no es una fecha válida por lo que se lanzará una excepción `DateTimeException` porque la fecha no existe.

La opción D es correcta porque el 7 de noviembre de 2022 es una fecha válida y no está relacionada con cambios de horario de verano así que no se lanzará una excepción.

La opción es incorrecta porque el dia 29 de febrero del año 2023 no existe no es un año bisiesto por lo que se provocara una `DateTimeException` porque la fecha no existe.

La opción F no es correcta (`LocalDate.of(2022, MonthEnum.MARCH, 13)`) porque `MonthEnum` no existe en la API de Java. Debería ser `Month`.

4. 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");
6: if (t == s) System.out.println("two");
7: if (t.intern() == s) System.out.println("three");
8: if ("Hello" == s) System.out.println("four");
9: if ("Hello".intern() == t) System.out.println("five");
```

A. `one`
B. `two`
C. `three`
D. `four`
E. `five`
F. The code does not compile.
G. None of the above

Opción A, C, D:
La opción A es correcta porque en la línea 5 está comparando un String al ponerlo “Hello” estamos usando la clase String por lo que al usar el equals la clase String lo sobrescribe para poder comparar el contenido y no la referencia de los objetos por lo que compara el contenido de “Hello” y s que contiene “Hello” esto es verdadero e imprime one.

La opción B es incorrecta porque en la línea 6 al usar el operador == está comparando la referencia de los objetos si ambos apuntan al mismo objeto y como en la línea 3 fue creado un String y en la línea 4 fue creado otro String usando new String explícitamente se creó otro objeto por lo que al usar el operador == nos da falso porque ambos apuntan a diferentes objetos, ya que se crea una nueva instancia de `String` usando el constructor `new String(s)`. Aunque el valor de `t` es `"Hello"`, no apunta al mismo objeto en el pool de strings, sino que es un objeto nuevo..

La opción C es correcta porque el método `intern()` busca en el **pool de strings** una cadena con el mismo contenido que `t`. Si la encuentra, devuelve la referencia de la cadena en el pool; si no la encuentra, agrega la cadena al pool y devuelve la referencia a esa nueva cadena en el pool. En este caso, `"Hello"` ya existe en el pool (asignado a `s`), por lo que `t.intern()` devuelve la referencia a `"Hello"` en el pool de strings. Como `s` también apunta a la cadena `"Hello"` en el pool, `t.intern() == s` evalúa como `true`, por lo que se imprime `three`.

La opción D es correcta porque la variable `s` se inicializa con `var s = "Hello";`, lo que hace que `s` apunte al objeto de cadena `"Hello"` en el **pool de strings**. El literal `"Hello"` usado en el `if ("Hello" == s)` también apunta al mismo objeto en el **pool de strings**, porque las cadenas literales idénticas en Java comparten la misma referencia en el pool. Dado que `s` y el

literal `"Hello"` tienen la **misma referencia** en la memoria (el mismo objeto en el pool de strings), la comparación con `==` da `true`.

La opción E es incorrecta porque **`"Hello".intern()`**: Esta llamada devuelve la referencia al objeto de la cadena `"Hello"` que está en el **pool de strings**. En este caso, `"Hello"` ya existe en el pool, por lo que `intern()` devuelve la referencia a ese objeto, aquí va todo bien pero la variable `t` se creó con `new String(s)`, lo que significa que `t` es un nuevo objeto `String` en la memoria, **no en el pool de strings**. Aunque `t` contiene el mismo contenido que `"Hello"`, no es el mismo objeto que la cadena internada en el pool. Aunque `"Hello".intern()` y `t` tienen el mismo contenido de cadena (`"Hello"`), no son la misma referencia en la memoria. `"Hello".intern()` apunta al objeto en el **pool de strings**, mientras que `t` apunta a un objeto diferente en la memoria **heap** al usar el operador **==**.

5. What is the result of the following code?

```
7: var sb = new StringBuilder();
8: sb.append("aaa").insert(1, "bb").insert(4, "ccc");
9: System.out.println(sb);
```

- **A.** abbaaccc
- **B.** abbaccca
- **C.** bbaaaccc
- **D.** bbaaccca
- **E.** An empty line
- **F.** The code does not compile.

Opción B:
La opción B es la única correcta porque el ejercicio crea un StringBuilder llamado sb y luego le agrega “aaa” además de insertar en el carácter 1 “bb” el carácter 1 inicia desde el primer carácter pero se agrega al final del caracter 1 por lo que se agrega “abbaa” pero después se vuelve a insertar en el carácter 4 “ccc” por lo que al final de la cuarta letra se agregan las 3 c quedando “abbaccca” haciendo la única respuesta correcta la B.

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

```
23: double one = Math.pow(1, 2);
24: int two = Math.round(1.0);
25: float three = Math.random();
26: var doubles = new double[] {one, two, three};
```

A. 0

B. 1

C. 2

D. 3

E. 4

Opción C:

La opción c es la única correcta ya que en la línea 24 `Math.round(1.0)` toma un argumento de tipo `double` y devuelve un valor de tipo `long`, no un `int`. Sin embargo, se está tratando de asignarlo a una variable `int`. Esto causará un error de compilación debido a la incompatibilidad de tipos. La solución sería hacer un cast explícito:

```
int two = (int) Math.round(1.0);
```

La razón por lo cual Math.random devuelve un double es porque por defecto regresa un double el error al querer guardar un long de 64 bits en un int de 32 bits.
Además Math.random

La línea 25 también está mal `float three = Math.random();` **ya que** : `Math.random()` devuelve un valor de tipo `double`, pero se está tratando de asignarlo a una variable `float` sin un cast. Esto también causará un error de compilación porque no se puede asignar un `double` a un `float` sin un cast explícito. El método `Math.random()` devuelve un **`double`**, no un

`float`. En Java, el valor `double` tiene mayor precisión que `float`, por lo que no se puede asignar directamente un `double` a una variable `float` sin hacer un **cast explícito**. La solución sería:

```
float three = (float) Math.random();
```

7. Which of these statements is true of the two values? (Choose all that apply.)

```
2022-08-28T05:00 GMT-04:00
2022-08-28T09:00 GMT-06:00
```

A. The first date/time is earlier.
B. The second date/time is earlier.
C. Both date/times are the same.
D. The date/times are two hours apart.
E. The date/times are six hours apart.
F. The date/times are 10 hours apart.

Opción A, E:
La opción A y C son correctas porque cuando se trabaja con zonas horarias, es mejor convertir a GMT primero restando la zona horaria. Primero es restar un valor negativo es como sumar. La primera fecha/hora es 9:00 ya que se le suma a las 05:00 el 04:00 dando un 9:00 GMT y la segunda es 15:00 GMT ya que se realiza lo mismo se le suma a 09:00 un 06:00 dando un 15:00. Por lo tanto, la primera es seis horas más temprana.

8. Which of the following return 5 when run independently? (Choose all that apply.)

```
var string = "12345";
var builder = new StringBuilder("12345");
```

A. `builder.charAt(4)`
B. `builder.replace(2, 4, "6").charAt(3)`
C. `builder.replace(2, 5, "6").charAt(2)`
D. `string.charAt(5)`
E. `string.length`
F. `string.replace("123", "1").charAt(2)`
G. None of the above

Opción A, B, F:
La opción A es correcta porque al usar .charAt(4) se está accediendo al elemento número 4 del builder = “12345” iniciando a contar del 1 con el índice 0 y el índice 4 es el número 5 por lo que es correcta.

La opción B es correcta porque porque está usando .replace(2, 4, “6”).charAt(3) al replace dice que accedemos a la posición 2 de la palabra llegando número 3, después le dice que a la posición 4 se reemplace con un “6” entonces elimina el 4 y el 5 no porque la última posición

siempre no se considera no es afectada y pone un 6 quedando un “1265” y luego charAt(3) pide que se regrese la posición 3 escogiendo el número 5.

vieja

1	2	3	4	5
0	1	2	3	4

nueva

1	2	6	5	
0	1	2	3	

La opción F también es correcta porque el replace(“123”, “1”) ahora está reemplazando una cadena completa no está posicionándose por índice lo que sucede es que “123” se va a eliminar y se colocará un “1” en su lugar quedando “145” y después está pidiendo con charAt(2) accede al índice 2 y extraerlo por lo que el elemento en el índice 2 es el número 5.

vieja

1	2	3	4	5
0	1	2	3	4

nueva

1	4	5		
0	1	2		

9. Which of the following are true about arrays? (Choose all that apply.)

A. The first element is index 0.

B. The first element is index 1.

C. Arrays are fixed size.

D. Arrays are immutable.

E. Calling `equals()` on two different arrays containing the same primitive values always returns `true`.

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

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

Opción A, C, F:

La opción A es correcta porque un arreglo siempre su primer elemento tiene el índice 0, también la C es correcta porque los arreglos tienen un tamaño fijo estos no pueden cambiar de tamaño durante la ejecución del código.

La opción F es correcta porque los arrays no sobrescriben el método equals por lo que equals se comporta como objet comparando si apuntan a la misma referencia.

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

```
23: int one = Math.min(5, 3);
24: long two = Math.round(5.5);
25: double three = Math.floor(6.6);
26: var doubles = new double[] {one, two, three};
```

A. 0
B. 1
C. 2
D. 3
E. 4

Opción A:
La opción A es correcta porque Math.min devuelve el mismo tipo de dato si ambos elementos que se le enviaron son del mismo tipo, en este caso la línea 23 ambos son int por lo que devuelve un int y no hay problema, aunque si hubiera un int y un long java regresaría un long para mantener la exactitud de los datos.

Si se enviara un int y un double java regresa un double, si se envía un long y un double java regresa un double y ya dependería si la variable donde se va almacenar puede contener lo que java regrese si no hay que hacer un casting.

Math.min regresa un int, Math.round regresa un long y Math.floor regresa un double

11. What is the output of the following code?

```
var date = LocalDate.of(2022, 4, 3);
date.plusDays(2);
date.plusHours(3);
System.out.println(date.getYear() + " " + date.getMonth()
    + " " + date.getDayOfMonth());
```

A. `2022 MARCH 4`
B. `2022 MARCH 6`
C. `2022 APRIL 3`
D. `2022 APRIL 5`
E. The code does not compile.
F. A runtime exception is thrown.

Opción E:
La opción E es correcta porque al haber creado un LocalDate.of(2022, 4, 3); solo podemos trabajar con fechas, como agregar dias,

- date.getYear();
- Month month = date.getMonth(); // Devuelve el mes como un enum Month
- int dayOfMonth = date.getDayOfMonth(); //devuelve el dia del mes
- DayOfWeek dayOfWeek = date.getDayOfWeek(); //devuelve el día de la semana
- int monthValue = date.getMonthValue(); //devuelve el valor numero del mes (1-12)
- boolean isLeap = date.isLeapYear(); //verifica si el año de la fecha es bisiesto
- LocalDate newDate = date.plusDays(10); // 10 días después de la fecha original
- LocalDate newDate = date.minusDays(5); // 5 días antes de la fecha original
- LocalDate newDate = date.plusMonths(2); // 2 meses después de la fecha original
- LocalDate newDate = date.minusMonths(3); // 3 meses antes de la fecha original
- LocalDate newDate = date.plusYears(1); // 1 año después de la fecha original

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

```
var numbers = "012345678".indent(1);
numbers = numbers.stripLeading();
System.out.println(numbers.substring(1, 3));
System.out.println(numbers.substring(7, 7));
System.out.print(numbers.substring(7));
```

A. `12`
B. `123`
C. `7`
D. `78`
E. A blank line
F. The code does not compile.
G. An exception is thrown.

Opción A, D, E:
La opción E es correcta porque la línea var numbers = "012345678".indent(1); deja un espacio en blanco dependiendo donde se le especifique en este caso al principio sin espacio “012345678” con espacio “ 012345678”.

La línea numbers = numbers.stripLeading(); lo que hace es quitarle los espacios en blanco pero solamente del inicio en lugar de verse así “ 012345678” ahora le quitara los espacio dejandolo asi “012345678”.

La opción A es correcta porque en System.out.println(numbers.substring(1, 3)); “012345678” lo que hace es sustraer desde el incide 1 al 3 dejando solo “12” el número 3 ex exclusive por lo que no se toma en cuenta dejando solo el 12

La opción D es correcta porque el System.out.println(numbers.substring(7)); dice que vamos a sustraer del indice 7 al 7 “012345678” dejando solo el 78 porque el 7 último no se toma en cuenta.

La opción de E es correcta porque System.out.println(numbers.substring(7, 7)); El método **`substring(int start, int end)`** toma dos índices: **`start`** (inclusive) y **`end`** (exclusivo). Esto significa que la subcadena extraída comienza en el índice `start` y termina en el índice `end - 1`. En este caso, se está llamando a **`substring(7, 7)`**. Esto significa que se está tratando de extraer una subcadena desde el índice **7** hasta el índice **7** (sin incluir el carácter en el índice 7). Debido a que **`start`** y **`end`** son iguales, **no se extrae ningún carácter**, por lo que el resultado es una **cadena vacía** (`""`).

13. What is the result of the following code?

```
public class Lion {
   public void roar(String roar1, StringBuilder roar2) {
      roar1.concat("!!!");
      roar2.append("!!!");
   }
   public static void main(String[] args) {
      var roar1 = "roar";
      var roar2 = new StringBuilder("roar");
      new Lion().roar(roar1, roar2);
      System.out.println(roar1 + " " + roar2);
} }
```

A. `roar roar`
B. `roar roar!!!`
C. `roar!!! roar`
D. `roar!!! roar!!!`
E. An exception is thrown.
F. The code does not compile.

Opción B:
La opción B es correcta porque en esta parte roar1.concat("!!!"); se está concatenando “!!!” Pero no se está guardando o apuntando a esa referencia por lo que se pierde y solo guarda roar que se creó en el método main, esto pasa porque es String y este no es mutable a diferencia de StringBuilder que si es mutable y no es necesario agregar o asignar el cambio a una variable por lo que al imprimir la variable imprimirá la concatenación agregada roar !!!

14. 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()`
B. `new Instant()`
C. `date.toInstant()`
D. `dateTime.toInstant()`
E. `time.toInstant()`
F. `zonedDateTime.toInstant()`

Opción A, F:
La opción **A** es correcta porque para realizar ese proceso se ocupa Instant.now() además para poder usarlo debe tener antes una zona horaria por lo que hace la opción C incorrecta ya que se quiere hacer un date.toInstant cuando date solo tiene la fecha y no una zona horaria lo mismo aplica para la opción D es incorrecta, también la opción B es incorrecta porque nunca se puede instanciar ya que es un método estático y estos en java se llaman directamente desde la clase como se realizó en la opción A y la opción **F** también es correcta porque la variable zonedDateTime.toInstant() ya trae una zona horaria demás de la fecha y la hora.

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

```
var arr = new String[] { "PIG", "pig", "123"};
Arrays.sort(arr);
System.out.println(Arrays.toString(arr));
System.out.println(Arrays.binarySearch(arr, "Pippa"));
```


A. `[pig, PIG, 123]`

B. `[PIG, pig, 123]`

C. `[123, PIG, pig]`

D. `[123, pig, PIG]`

E. `-3`

F. `-2`

G. The results of `binarySearch()` are undefined in this example.

Opción C, E:
La opción **C** Los arrays al ordenarlos y contener números, Palabras en minúsculas y palabras en mayúsculas primero ordenará los números luego las mayúsculas y al final las minúsculas y esto pasa así por el código ASCII porque primero vienen los números luego Mayúsculas y después las minúsculas, si el array llega a traer un null este no se podrá ordenar y saldrá **error de compilación**, el método **binarySearch** no puede trabajar si el array no está ordenado por lo que si no está ordenado y se intenta buscar algo daría un -1 además este método cuando se usa correctamente da el índice del elemento que se está buscando pero si se busca Un valor que no existe como en este caso “Pippa” lo que pasa es que el arreglo lo ordena y lo pone en su posición con su índice y lo vuelve negativo al índice y se le resta -1 en este caso si se ordena este elemento quedaría en el índice 2 quedando negativo en -2 por lo que al restarle 1 el resultado es -3 siendo la opción correcta la **E**.

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

```
var base = "ewe\nsheep\\t";
int length = base.length();
int indent = base.indent(2).length();
int translate = base.translateEscapes().length();

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

A. 10
B. 11
C. 12
D. 13
E. 14
F. 15
G. 16

Opción A, B, G:
La opción **A** es correcta porque en la primera línea **var base = "ewe\nsheep\\t";** el **\n** está haciendo un salto de línea.
\t es una secuencia de escape que se traduce a un carácter de tabulación cuando se imprime o se muestra. Cuando Java encuentra **\t** dentro de una cadena, lo interpreta como un espacio de tabulación y no como los caracteres \ y t por separado. El tabulador inserta un espacio equivalente a varios espacios en blanco (la cantidad exacta depende del entorno o la configuración del editor/terminal). **\\t** (doble barra invertida) se usaría en un contexto donde necesitas representar el carácter \t como una cadena literal, es decir, no como la secuencia de escape, sino como los caracteres \ y t juntos. Por lo que se imprime un
“ewe
sheep\t”
Si se imprimiera la longitud saldría un 11 porque estaría conformado así porque el **\n** se toma como un solo espacio además el **\\t** este se toma como dos espacio porque escapa con la doble diagonal invertida y muestra la cadena literal **\t** lo cual ocupa dos espacios. Por lo que la opción **B** es la correcta.

e	w	e	\n	s	h	e	e	p	\	t
1	2	3	4	5	6	7	8	9	10	11

La opción **G** es correcta porque esta línea **int indent = base.indent(2).length();** la **indent** está diciendo que se le agregue 2 espacios en la indentación al principio de cada línea y como tenemos dos líneas hechas entonces si contamos la longitud total ahora incrementa de 11 a 15 pero hay uno más porque el **indent** genera un salto de línea al final por lo que en total son 5 espacios en total y si le sumamos 11 que eran de la anterior más 5 son 16.

La opción **A** también es correcta porque en esta línea **int translate = base.translateEscapes().length();** el **translateEscapes()** está elimina los escape que son las dobles diagonales invertidas \\ entonces como \\t tiene dos diagonales invertidas esta escapa y ahora si pone un tabulador quedando así:
“ewe
sheep ”
Ahora el pasa de tener 11 a solo tener una longitud de 10 por haber quitado el escape y dejando el tabulador

e	w	e	**\n**	s	h	e	e	p	**\t**
0	1	2	3	4	5	6	7	8	9

`"ewe"` tiene 3 caracteres.

`\n` cuenta como 1 carácter.

`"sheep"` tiene 5 caracteres.

`\t` (tabulador) cuenta como 1 carácter.

17. 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`.

B. `letters.substring(2, 2)` returns a single-character `String`.

C. `letters.substring(6, 5)` returns a single-character `String`.

D. `letters.substring(6, 6)` returns a single-character `String`.

E. `letters.substring(1, 2)` throws an exception.

F. `letters.substring(2, 2)` throws an exception.

G. `letters.substring(6, 5)` throws an exception.

H. `letters.substring(6, 6)` throws an exception.

Opción A, G:
La opción **A** es correcta porque el susbtring(1, 2) está diciendo que toma el índice 1 y luego el índice 2 pero este siempre es exclusive por lo que se le resta - 1 quedando en índice 1 eligiendo sólo el elemento en esa posición.

a	b	c	d	e	f	g
0	1	2	3	4	5	6

La opción **G** también es correcta porque dice que agarra del índice 6 al 5 pero no se puede comenzar por un número mayor y luego intentar uno menor porque provoca una excepción.

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

```
13: String s1 = """
14:    purr""";
```



```
15: String s2 = "";
16:
17: s1.toUpperCase();
18: s1.trim();
19: s1.substring(1, 3);
20: s1 += "two";
21:
22: s2 += 2;
23: s2 += 'c';
24: s2 += false;
25:
26: if ( s2 == "2cfalse") System.out.println("==");
27: if ( s2.equals("2cfalse")) System.out.println("equals");
28: System.out.println(s1.length());
```

A. 2
B. 4
C. 7
D. 10
E. ==
F. equals
G. An exception is thrown.
H. The code does not compile.

Opción C, F:
La opción **C** es correcta porque en la línea 20 se le está asignando a s1 += “two” la palabra two y luego saca la longitud y el total es de 7 por lo que la hace correcta.

La opción **F** también es correcta porque primero se le concatena s2 += 2 el número 2 que lo convierte a un String porque se está usando el operador + para asignarlo por lo que convierte el 2 en un String y no hay error pero si se pusiera solo por ejemplo s2 = 2; esto si causaría un error porque intenta guardar un int en un String, entonces después concatena una s2 += ‘c’ que igual si se pone sólo s2 = ‘c’ daría error y luego un s2 += false que este actúa igual que guardar un int si estuviera solo s2 = false daría un error. Luego en el primer if s2 == “2cfalse” da un false porque el operador == está comparando si son la misma referencia y no lo son, después el segundo if s2.equals(“2cfalse”) esta comparando ahora si el contenido porque el s2 es un string y esta clase sobrescribe el comportamiento de equals por lo que ahora sí da un true porque ambos contenidos son iguales.

19. 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)`
B. `mismatch(s1, s2)`
C. `compare(s3, s4)`
D. `mismatch (s3, s4)`
E. `compare(s4, s4)`
F. `mismatch (s4, s4)`

Opción A, B, D:
La opción **A** es correcta porque **`Arrays.compare()`**: Compara dos arreglos lexicográficamente y devuelve un valor:

- Un **número negativo** si el primer arreglo es "menor" que el segundo.
- **Cero** si los arreglos son iguales.
- Un **número positivo** si el primer arreglo es "mayor" que el segundo.

`Arrays.mismatch()`: Devuelve el índice de la primera posición en la que los dos arreglos difieren. Si no hay diferencias, devuelve **-1**.

A. `Arrays.compare(s1, s2)`

- Compara los dos arreglos `s1` y `s2` lexicográficamente.
- `s1 = { "Camel", "Peacock", "Llama" }`
- `s2 = { "Camel", "Llama", "Peacock" }`
- Comienza comparando los elementos en el mismo índice de ambos arreglos:
    - El primer elemento es igual (`"Camel"`).
    - El segundo elemento difiere: `"Peacock"` en `s1` y `"Llama"` en `s2`.

- `"Peacock"` es lexicográficamente mayor que `"Llama"`, por lo que `Arrays.compare(s1, s2)` devuelve un **número positivo**.

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

- Busca el primer índice en el que `s1` y `s2` difieren.
- La diferencia ocurre en el índice **1** (segundo elemento), por lo que devuelve **1**.

C. `Arrays.compare(s3, s4)`

- `s3 = { "Camel" }`
- `s4 = { "Camel", null }`
- Compara los elementos:
  - El primer elemento es igual (`"Camel"`).
  - `s3` se queda sin elementos, pero `s4` tiene un `null`, lo que significa que `s3` es "menor" que `s4`.
- Devuelve un **número negativo**.

D. `Arrays.mismatch(s3, s4)`

- Encuentra una diferencia en la longitud del arreglo.
- Devuelve el índice donde termina `s3`, que es **1**.

E. `Arrays.compare(s4, s4)`

- Compara el arreglo consigo mismo.
- Todos los elementos son iguales, por lo que devuelve **0**.

F. `Arrays.mismatch(s4, s4)`

- No hay diferencias, por lo que devuelve **-1**.

20. 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");
var dateTime1 = ZonedDateTime.of(date, time, zone);
var dateTime2 = dateTime1.plus(1, ChronoUnit.HOURS);

long diff = ChronoUnit.HOURS.between(dateTime1, dateTime2);
int hour = dateTime2.getHour();
boolean offset = dateTime1.getOffset()
   == dateTime2.getOffset();
System.out.println("diff = " + diff);
System.out.println("hour = " + hour);
System.out.println("offset = " + offset);
```

A. `diff = 1`
B. `diff = 2`
C. `hour = 2`
D. `hour = 3`
E. `offset = true`
F. The code does not compile.
G. A runtime exception is thrown.

Opción A, D:
La opción **A** es correcta porque **long diff = ChronoUnit.HOURS.between(dateTime1, dateTime2);** Esta línea solo da la diferencia real entre las horas que es solo 1

La opción **D int hour = dateTime2.getHour();** Esta línea solo obtiene las horas que son 3 por lo que también es correcta.

boolean offset = dateTime1.getOffset() == dateTime2.getOffset(); Esta línea verifica si las dos zonas horarias están a la misma zona horaria si es cierto da un true si no un false, en este caso da un false porque están a diferentes zonas horarias por haber incrementado una hora en un horaria de cambio de horario.

21. 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()`
B. `append("vaJ$").substring(0, 4)`
C. `append("vaJ$").delete(0, 3).deleteCharAt(puzzle.length() - 1)`
D. `append("vaJ$").delete(0, 3).deleteCharAt(puzzle.length())`
E. None of the above

Opción A, C:
La opción **A** es correcta porque al usar el método reverse voltea en reversa una cadena pero este método solo está disponible en la clase StringBuilder y la clase StringBuffer en la clase String no está disponible.

La opción **C** es correcta porque **append("vaJ$").delete(0, 3).deleteCharAt(puzzle.length())** la línea hace lo siguiente:

append("vaJ$").delete(0, 3).deleteCharAt(puzzle.length() - 1)
“JavavaJ$” “avaJ$” “avaJ”
Esto imprime la palabra “avaJ” correctamente.

La opción **B** no es correcta porque cuando aplicamos un substring a un StringBuilder este deja de ser de este tipo y se convierte a String por lo que esta línea en realidad daría un “Java” al no ser mutable. Aparte de que al hacer esa operación también se deja fuera del pool de String por lo que se crean dos objetos diferentes. **var puzzle = new StringBuilder("Java");** y **String s1 = append("vaJ$").substring(0, 4);** y luego s == “Java” da un false, pero si lo metemos con un inter s.intern() == “Java” aquí daría un true.

La opción **D** no es correcta porque la longitud de la palabra “avaj$” 5 por lo que se coloca en el índice 5 y ahí no hay nada por lo que se provocaría un error.
append("vaJ$").delete(0, 3).deleteCharAt(puzzle.length())
“Javavaj$” “avaj$” “avaj$”

22. What is the output of the following code?

```
var date = LocalDate.of(2022, Month.APRIL, 30);
date.plusDays(2);
date.plusYears(3);
System.out.println(date.getYear() + " " + date.getMonth()
   + " " + date.getDayOfMonth());
```


A. `2022 APRIL 30`

B. `2022 MAY 2`

C. `2025 APRIL 2`

D. `2025 APRIL 30`

E. `2025 MAY 2`

F. The code does not compile.

G. A runtime exception is thrown.

Opción A:
La opción **A** es correcta porque el tipo de dato LocalDate no es mutable pasa lo mismo que la clase String, está una vez que es creada no se puede modificar para poder modificarla se crean nuevos objetos y se tiene que apuntar a él si no el valor se pierde por ejemplo al hacer estas dos líneas
date.plusDays(2);
date.plusYears(3);
Lo que pasa es que el valor se pierde porque no se apunta a la misma referencia para no perderlo se debería asignar el nuevo valor a la misma variable
date = date.plusDays(2);

```
date = date.plusYears(3);
```