Temario ASO - Parte 2

Código de colores:

- Azul: la respuesta que encontré.
- Verde: es la opción más probable.
- Amarillo: puede ser.
- Rojo: no es una opción válida.

Sólo en algunas preguntas pondré todos los incisos, aquellas que requieran una mayor justificación.

56. Son patrones de diseño de microservicios.
    - Retry: Es un proceso que consiste en repetir una petición automáticamente en caso de que algún fallo sea detectado.
    - Circuit Breaker: previene que una aplicación intente repetidamente ejecutar una operación que es muy probable que falle.
    - Adaptive Lifo: la última solicitud que aparezca es a la que se le da salida, ya que se asume que las peticiones que se realizaron antes ya no son necesarias.
    - Bulkhead: aísla los componentes en grupos para garantizar que, en caso de que alguno falle, los demás puedan seguir funcionando.

    a.

    b.

    c. Retry, Circuit Breaker, Adaptive Lifo y Bulkhead.

57. ¿Qué afirmaciones son verdaderas tanto para las clases abstractas como para las correctas? Elige todas las correctas.

    a. Ambos pueden contener métodos estáticos.

    b. Ambos se pueden ampliar con la clave `extends`. ← Una interfaz puede extender otra interfaz, y una clase abstracta puede implementar varias interfaces.

    c. Ambos pueden contener métodos predeterminados. ← Suponiendo que se refieren a la palabra reservada `default`, ambos pueden contener esos métodos.

    d. Ambos heredan de `java.lang.Object`. ← Las interfaces no heredan de Object.

    e. Ninguno de los dos puede ser instanciado directamente.

    f. Ambos pueden contener variables finales estáticas públicas.

    g. Supone que todos los métodos dentro de ellos son abstractos. ← No, ya que las interfaces pueden tener métodos estáticos y `default`; mientras que las clases abstractas pueden tener métodos con comportamiento.

58. ¿Cuál no es un objetivo de Maven?

a. Clean ← Comando Maven

b. Package ← Comando Maven

c. Debug ← Eso lo suelen hacer las IDEs

d. Install ← Comando Maven

59. Si deseas obtener una copia de un repositorio Git existente en un servidor, ¿qué comando utilizarías?

`git clone` es un comando que se utiliza para crear una copia local de un repositorio Git existente en un servidor. Este comando copia todo el historial del repositorio, incluyendo todas las versiones de cada archivo y directorio del proyecto.

a.

b.

c. `git clone`

60. ¿Qué es un repositorio remoto en Git?

Un repositorio remoto en Git es una versión del proyecto que se encuentra alojada en la red. Permite a múltiples usuarios colaborar en un mismo proyecto.

a.

b.

c. Un servidor Git que almacena una copia central del repositorio ← Es la más acertada.

61. Dada la siguiente clase:

```
01. public class Helper {
02.   public static <U extends Exception> void
03.     printException(U u) {
04.           System.out.println(u.getMessage());
05.   }
06.
07.
08.   public static void main(String[] args) {
09.     //TODO
10.   }
11. }
```

¿Cuál de las siguientes instrucciones puede colocarse en la línea 9 para que la clase `Helper` compile?

a. `Helper.printException(new Exception("B"));`

b. `Helper.printException(new FileNotFoundException("A"));`

c.

d. `Helper.<NullPointerException>printException(new NullPointerException(”D”));`

32. ¿Cuál es la salida al ejecutar el siguiente código?

```
01. public class Fish {
02.   public static void main(String[] args) {
03.       int numFish = 4;
04.       String fishType = "tuna";
05.       String anotherFish = numFish + 1;
06.       System.out.println(anotherFish + " " + fishType);
07.       System.out.println(numFish + " " + 1);
08.   }
09. }
```

El código no compila en la línea 5 porque se quiere guardar una suma de dos enteros en una variable de tipo `String` sin haber utilizado el método `toString()`.

g. El código no compila

33. ¿Cuál de las siguientes opciones son correctas? Elige todas las correctas.

```
public class StringBuilders {

    public static StringBuilder work(StringBuilder a, StringBuilder b) {
        a = new StringBuilder("a");
        b.append("b");
        return a;
    }

    public static void main(String[] args) {
        StringBuilder s1 = new StringBuilder("s1");
        StringBuilder s2 = new StringBuilder("s2");
        StringBuilder s3 = work(s1, s2);
        System,out.println("s1 = " + s1);
        System,out.println("s2 = " + s2);
        System,out.println("s3 = " + s3);
    }
}
```

a. s2 = s2 ← Falso, `s1 = "s1"` y `s2 = s2b`.

b. s3 = null ← Falso, `s1 = "a"`.

c. s1 = s1

d. s3 = a

e. El código no compila

f. s2 = s2b

g. s1 = a ← Falso, `s1 = "s1"`.

34. ¿A qué hace referencia el principio de Liskov? (Principios SOLID)

    Hace referencia a cómo usar la referencia de forma adecuada. En pocas palabras, dice: *"Si S es de un subtipo de T, T puede ser reemplazado con objetos de tipo S sin alterar el comportamiento esperado en el programa."*.

    a.

    b. Este principio nos indica que dentro del programa una clase puede ser sustituida por cualquier clase que se extienda de ella sin alterar el comportamiento del programa.

35. ¿Qué es un *code smell*?

    Son una especie de rastros en el código que indican un problema mucho más profundo en la aplicación. No son como tal errores o bugs pero sí rompen fundamentos de desarrollo de código que eventualmente conducen a una calidad baja del código.

    a.

    b.

    c. Un indicador de que pude haber un problema en el código que puede ser difícil de detectar o que podría ser una fuente potencial de errores o problemas de mantenimiento en el futuro.

36. ¿Qué significa el acrónimo CRUD en una API REST?

    b. *Create*, *Read*, *Update* y *Delete*.

37. ¿Para qué nos sirve utilizar un *profile* dentro del archivo `pom.xml`?

    Un *profile* en Maven es una serie de configuraciones específicas que pueden ser activadas para personalizar el proceso de construcción (build) del proyecto. Esto permite adaptar el procedimiento de construcción a diferentes entornos, por ejemplo, para desarrollo, pruebas o producción.

    a. a

    b. Es la etiqueta por la cual podemos definir varias características que tendrá nuestro proyecto al ser compiladas. ← Es la más acertada.

38. Dadas las siguientes clases `Vehicle` y `Car`:

```
package my.vehicles;

public class Vehicle {
    public String make;
    protected String model;
    private int year;
```

```
    int mileage
}
```

```
01. package my.vehicles.cars;
02.
03. import my.vehicles.*;
04.
05. public class Car extends Vehicle {
06.     public Car(){
07.         //TODO
08.     }
09. }
```

¿Cuál de las siguientes instrucciones pueden colocarse en la línea 7 para que la clase `Car` compile correctamente? Seleccione las que apliquen.

a. `mileage = 15285;` ← No compilaría porque el acceso a `milage` es *default*, es decir, sólo quien está dentro del mismo paquete puede acceder a ésta.

b. Ninguna de las anteriores. ← ...

c. `make = “Honda”;`

d. `year = 2009;` ← No se puede acceder a `year` ya que la variable es privada.

e. `model = “Pilot”;`

69. Enumere cuatro interfaces de la API colecciones.

a. `List`, `Map`, `Set`, `Queue`.

70. Selecciona la respuesta correcta con respecto al resultado del bloque de código.

```
public class Tests {
    public static void main(String args[]) {
        Side primerIntento = new Head();
        Tail segundoIntento = new Tail();
        Coin.overload(primerIntento);
        Coin.overload((Object)segundoIntento);
        Coin.overload(segundoIntento);
        Coin.overload((Side)primerIntento);
    }
}

interface Side { String getSide(); }

class Head implements Side {
    public String getSide() { return "Head "; }
}
```

```
class Tail implements Side {
    public String getSide() { return "Tail "; }
}

class Coin {
    public static void overload(Head side) {
        System.out.println(side.getSide());
    }
    public static void overload(Tail side) {
        System.out.println(side.getSide());
    }
    public static void overload(Side side) {
        System.out.println("Side ");
    }
    public static void overload(Object side) {
        System.out.println("Object ");
    }
}
```

El siguiente código imprime lo siguiente:

```
"Side "
"Object "
"Tail "
"Side "
```

Respuesta correcta: inciso c.

71. ¿Cuál es la salida al ejecutar el siguiente código?

```
public class Lion {
    public void roar(String roar1, StringBuilder roar2) {
        roar1.concat("!!!");
        roar2.append("!!!");
    }

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

El código imprime `roar roar!!!`

a. `roar roar!!!`.

72. ¿Cuál de los siguientes es cierto acerca de una subclase concreta?

    a. Una subclase concreta no se puede marcar como final. ← Sí puede marcarse, un ejemplo es `String`.

    b. Una subclase concreta debe implementar todos los métodos definidos en una interfaz heredada. ← No necesariamente, los métodos estáticos pueden estar en las interfaces y no pueden ser sobrescritos.

    c. Una subclase concreta debe implementar todos los métodos abstractos heredados.

    d. Una subclase concreta puede declararse como abstracta. ← No, eso implicaría que no se podrían instanciar objetos de esta clase, por lo cual perdería el calificativo de concreta.

    e. Los métodos abstractos no pueden ser anulados por una subclase concreta. ← No entiendo a qué se refiere con anulados.

73. ¿Cuál es la salida del siguiente código?

```
public abstract class Catchable {
    protected abstract void catchAnObject(Object x);

    public static void main(String [] args) {
        java.util.Date now = n new java.util.Date();
        Catchable target = new MyStringCatcher();
        target.catchAnObject(now);
    }
}

class MyStringCatcher extends Catchable {
    public void catchAnObject(Object x) {
        System.out.println("Caught object");
    }

    public void catchAnObject(String s) {
        System.out.println("Caught string");
    }
}
```

    El código da como salida `"Caught Object"`, ya que todas las clases heredan de `Object` y en cuanto a la firma de métodos, la variable de referencia agarra aquel cuyo parámetro embone mejor y, como no tiene nada que ver `Date` con `String`, se va con `Object`.

    e. `"Caught Object"`.

74. Seleccione la respuesta que considere correcta, dado el siguiente bloque de código.

```
import java.util.Arrays;
import java.util.List;
```

```
public class Example {

    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

        int result = numbers.stream()
                                    .filter(n -> n % 2 == 0)
                                    .reduce(0, (a, b) -> a + b);

        system.out,println(result);
    }
}
```

El código toma los números pares con la función `filter` y después los suma. lo cual da como resultado 6.

d. 6

75. ¿Qué declaración representa una declaración válida que permitirá la inclusión de clases del paquete `java.util`?

c. `import java.util.*;`

76. ¿Qué es la cobertura de código?

Es un parámetro con el que identificamos qué tanto porcentaje del código ha sido sometido a una prueba unitaria que comprueba el correcto funcionamiento del programa.

d. La cantidad de código que se ejecuta durante una prueba.

77. ¿Cuál es el formato correcto para hacer un commit en Git?

Por buena práctica, un *commit* debe contener qué cambio de está haciendo (añadiendo archivos, modificando o eliminando), la descripción del cambio que se está haciendo, detallar más si el cambio es muy grande y una referencia en caso de ser necesaria. La respuesta la considero muy abierta ya que cada proyecto tendrá su propio formato para los commit.

a.

b. Tipo de cambio, descripción breve, cuerpo opcional y notas de pie de página. ← Me parece la más acertada.

78. ¿Qué es el patrón de diseño Singleton y cómo se implementa en Java 8?

Es un patrón de diseño que garantiza que una clase solo tenga una instancia y proporciona un punto de acceso global a ésta. Se utiliza cuando solo se necesita un objeto para coordinar acciones en todo el sistema.
En Java 8, podemos implementarlo utilizando una clase con un sólo elemento privado estático del tipo de la clase, con un constructor privado y un método que nos devuelva este elemento.

a. El patrón de diseño Singleton es un patrón que se utiliza para garantizar que una clase tenga una única instancia en todo el sistema. Se implementa utilizando una variable estática y un constructor privado.

79. En los verbos REST, ¿cuál es la diferencia en el uso de PATCH y PUT?

- PUT: Actualiza completamente un recurso en el servidor, donde se espera que el cuerpo de la solicitud tenga la representación completa del recurso.
- PATCH: Actualiza parcialmente un recurso ya existente. Sólo contendrá los cambios concretos que se desean actualizar.

a.

b.

c. PUT requiere que se le envíe la entidad completa mientras que PATCH solo los atributos a modificar.

80. ¿Cuál es la diferencia entre las anotaciones `@RestController`, `@Component`, `@Service` y `@Repository`?

`@RestController`

- Es una especialización de `@Controller`.
- Combina `@Controller` y `@ResponseBody`, lo que significa que los métodos dentro de una clase anotada con `@RestController` devuelven datos directamente en formato JSON o XML.
- Se utiliza para construir controladores que manejan solicitudes HTTP REST.

`@Controller`

- Anotación de Spring que indica que una clase es un controlador web.
- Se utiliza principalmente en aplicaciones web basadas en MVC.
- Los métodos dentro de una clase anotada con `@Controller` pueden devolver vistas que se renderizan como HTML.

`@Component`

- Es una anotación genérica para cualquier componente gestionado por Spring.
- Indica que una clase es un componente y debe ser gestionada por el contenedor de Spring.
- Puede ser usada en cualquier clase para marcarla como un bean de Spring.

`@Service`

- Es una especialización de `@Component`.
- Se usa para marcar una clase como un servicio, típicamente contiene lógica de negocio.
- Es una indicación semántica que la clase contiene lógica de negocio, aunque funcionalmente es similar a `@Component`.

@ `Repository`

- Es una especialización de `@Component`.

- Se usa para marcar una clase como un repositorio, típicamente interactúa con la base de datos.
- Además de ser gestionada como un bean de Spring, `@Repository` proporciona una capa de abstracción adicional para el manejo de excepciones específicas de la base de datos.

En resumen, todas estas anotaciones indican que las clases deben ser gestionadas por el contenedor de Spring (se convierten en beans de Spring), pero tienen propósitos semánticos diferentes que ayudan a organizar el código y a proporcionar claridad sobre el rol de cada componente dentro de la aplicación.

a.

b.

c. No existe diferencia funcional entre ellas sino semántica, las 4 son anotaciones de Spring que crean un bean y lo agregan al contexto de Spring.

31. ¿Cuál es una buena práctica al escribir pruebas unitarias?

a. Ejecutar pruebas con poca frecuencia.

b. Asegurarse de que las pruebas sean claras y concisas.

c. Probar solo una pequeña parte de una función. ← Debe probar toda la función a la que se quiere hacer la prueba.

d. Hacer que las pruebas dependan de otras pruebas. ← Se perdería lo unitario de las pruebas.

32. ¿Cuál es la ventaja de usar APIs REST sobre otros tipos de servicios web?

Las APIs REST pueden implementarse y modificarse fácilmente, lo que brinda visibilidad y portabilidad entre plataformas a cualquier sistema de API.

a. Mayor seguridad. ← Se dice popularmente que SOAP es más seguro.

b. Mayor facilidad de implementación. ← Es la característica que más destaca.

c. Mayor velocidad de transferencia de datos. ← No depende del servicio, depende del diseño.

d. Mayor compatibilidad con diferentes plataformas. ← En teoría, todos los servicios web son compatibles, pero también destaca por otras fuentes como una característica de REST.

33. Selecciona la respuesta correcta con respecto al resultado del bloque de código.

```
public class Test4 {
    public static void main(String[] args) {
        List list = Arrays.asList(25, 7, 25, 67);
        System.out.println(list);
        System.out.println(new HashSet(list));
        System.out.println(new TreeSet(list));
        System.out.println(new HashSet(list));
```

```
        System.out.println(new ConcurrentSkipListSet(list));
    }
}
```

El resultado del código es el siguiente:

```
"[25, 7, 25, 67]" // Implementación del toString
"[25, 7, 67]" // El orden puede variar
"[7, 25, 67]" // Ordenado
"[25, 7, 67]" // El orden puede variar
"[7, 25, 67]" // Revisando la doc, ordena la lista.
```

b. Es la opción más acertada.

34. ¿Cuáles son los cuatro pilares de la programación orientada a objetos?

d. Polimorfismo, Abstracción, Herencia y Encapsulamiento.

35. ¿Qué utilidad de línea de comandos basada en MS Windows le permitirá ejecutar el intérprete de Java sin abrir la ventana de la consola?

b. `javaw` ← Correcta

36. ¿Qué es un endpoint en una API REST? (Repetida)

d. Es la URL que se utiliza para acceder a una API REST.

37. ¿Cuál es el valor de `x` e `y` al final del programa?

```
int x = 0;

do {
    System.out.println(x);
    x++;
} while(x < 10);

int y = 0;

while(y < 10) {
    System.out.println(y);
    y++;
}
```

El valor de `x` será 10 ya que se hacen los incrementos necesarios para que llegue a ese valor y es cuando ya se sale del ciclo. Por otro lado, `y` será 10 ya que al llegar a dicho valor, es cuando ya no entrará al ciclo.

c. `x=10` y `y=10`.

38. Dado el siguiente `enum` y clase, ¿cuál es la opción que puede ir en el espacio en blanco para que el código compile?

```
enum Season { SPRING, SUMMER, WINTER }

public class Weather {
    public int getAverageTemperate(Season s) {
        switch(s) {
        default:
            __________ return 30;
        }
    }
}
```

a. Ninguno de los anteriores ← Todas las opciones proponen usar un `case` cuando no pueden ir dentro de un `defau`

39. ¿Cuál es el resultado de compilar y ejecutar el siguiente programa?

```
public static void main(String[] args) {
    boolean stmt1 = "champ" == "champ";
    boolean stmt2 = new String("champ") == "champ";
    boolean stmt3 = new String("champ") == new String("champ");
    System.out.println(stmt1 && stmt2 || stmt3);
}
```

Tomando en cuenta que `stmt1 = true` y los otros dos son falsos, entonces `stmt1 && stmt2` es falso, y `false || stmt3` es falso. Por lo tanto, se imprime `false`.

a. `false`

90. ¿Cómo se manejan las excepciones en Java?

Utilizamos `try` cuando cierto fragmento del código puede lanzar una excepción; con `catch` atrapamos una excepción y manejamos el error. Optativamente (o si no utilizamos el `catch`), usamos `finally`, que es un bloque que siempre va a ejecutarse independientemente si se lanzó un error o no y de manera general sirve para cerrar los recursos del bloque `try`.

a.

b.

c.

d. Se manejan con bloques try-catch-finally en Java. La excepción *try with Resources* es una forma de cerrar automáticamente los recursos abiertos en un bloque *try*.

91. ¿Qué clase del paquete `java.io` permite leer y escribir archivos en ubicaciones específicas dentro de un archivo?

a. File ← Clase abstracta

b. filename filter

c. file descriptor

d. RandomAccessFile

92. Todas las siguientes definiciones de clases `my.school.Classroom` y `my.city.School`, ¿qué números de línea en el método `main` generan un error de compilación? Elige todas las opciones correctas.

```
1. package my.school;
2. public class Classroom {
3.  private int roomNumber;
4.  protected String teacherName;
5.  static int globalKey = 54321;
6.  public int floor = 3;
7.  Classroom(int r, String t) {
8.      roomNUmber = r;
9.      teacherName = t } }
```

```
1. package my.city;
2. import my.school.*;
3. public class School {
4.  public static void main(String[] args) {
5.  System.out.println(Classroom.globalKey);
6.  Classroom room = new Classroom(101, "Mrs. Anderson");
7.  System.out.println(room.roomNumber);
8.  System.out.println(room.floor);
9.  System.out.println(room.teacherName); } }
```

a. Ninguno, el código compila bien. ← No compila.

b. 6 ← El constructor al que quiere acceder `School` tiene modificador de acceso *default* y se encuentra en otro paquete.

c. 9 ← `teacherName` es `protected` y `School` no hereda de `Classroom`.

d. 7 ← `roomNumber` es privado.

e. 8 ← `floor` es público.

f. 5 ← `globalKey` es *default*.

93. ¿Qué es una expresión lambda en Java? (Repetida)

a.

b.

c. Una forma concisa de representar una función anónima que se puede pasar como argumento.

94. ¿Qué hace el siguiente código fuente?

```
int x = 0;
boolean flag = false;
while((x < 10) || !flag) {
    System.out.println(x);
    x++;
}
```

Ya que en la condición del *while* se encuentra un operador *or* y éste contiene `!flag` (que siempre será verdadero), el código imprimirá el valor de `x` que será una unidad mayor en cada iteración indefinidamente.

c. Se queda en un bucle infinito.

95. ¿Qué son las anotaciones en Java?

Proveen información adicional al programa y ayudan a asociar metadatos a los elementos del programa.

a.

b.

c. Un mecanismo para etiquetar y procesar código de forma especial. ← Es la que me hace más sentido.

96. ¿Qué son las expresiones regulares en Java?

Una expresión regular es una secuencia de caracteres que forman un patrón de búsqueda. En Java, podemos importar el paquete `java.util.regex` para utilizarlas e incluye las clases `Pattern`, `Matchet` y `PatternSyntaxException`.

a.

b. Un mecanismo para validar y manipular cadenas de caracteres. ← Es la que mejor suena.

97. ¿Qué es una anotación en Spring?

Es una forma de agregar metadatos a los programas en Java. Se utilizan como etiquetas para configurar aspectos de la aplicación sin la necesidad de escribir código explícito. Algunas anotaciones comunes en Spring incluyen `@Component`, `@Repository`, `@Service`, y `@Controller`, cada una de las cuales se utiliza para definir el papel de una clase en la estructura de la aplicación.

a.

b. Una etiqueta que se utiliza para anotar una clase o un método y proporcionar información adicional al contenedor de Spring. ← La que mejor suena.

98. ¿Cuál es la salida? Nota: el `var` no es cosa de Java 8.

```
import java.utils.ArrayList;

public class OtherExample {
    public static void main(String[] args) {
        var list = new ArrayList<String>();
        list.add("Austin");
        list.add("Boston");
        list.add("San Francisco");
        var c = list.stream()
                    .filter(a -> a.length() > 10)
                    .count();
        System.out.println(c + " " + list.size());
    }
}
```

El código se encarga de filtrar la lista para quedarse con los elementos que tengan más de 10 caracteres, lo cual sólo lo cumple la cadena `"San Francisco"`. Luego, contará los elementos filtrados. Entonces, el valor de `c` es 1 porque sólo tiene un elemento pero el tamaño de la lista seguirá siendo 3 porque nunca se le hizo ninguna operación a ésta, sólo al *stream*.
En cuando
`var`, es una palabra reservada utilizada para las variables locales, cuyo tipo depende del objeto al que esté apuntando.

b. 1 3

99. ¿Cuál es la salida de la siguiente aplicación?

```
interface Speak {
    default int talk(){
        return 7;
    }
}
interface Sing {
    default int talk(){
        return 5;
    }
}

public class Performance implements Speak, Sing {
    public int talk(String... x) {
        return x.length;
    }

    public static void main(String[] notes) {
        System.out.println(new Performance().talk());
```

```
    }
}
```

El código no compilará ya que el método `main` toma el método `default` que no recibe parámetros, pero es tan difícil saber cuál de las dos interfaces tomará tanto para nosotros como para el compilador.

e. El código no compila

30. ¿Qué conjunto de modificadores, cuando son agregados a un método *default* dentro de una interfaz, evitan que sea sobrescrito por la clase que lo implementa?

De manera general, podemos evitar la sobrescritura de métodos con `final`.

c. `final`

31. ¿Qué tipo de excepción se produce cuando se intenta realizar una operación incompatible con el tipo de datos en Java?

d. `ClassCastException`.

32. ¿Qué es un *starter*?

a. Es una herramienta que nos facilita la creación y configurar un proyecto cargando las dependencias necesarias para un objetivo.

33. Selecciona la respuesta correcta con respecto al resultado del siguiente bloque de código.

```
public class Test2 extends Thread {
    public static void main(String[] args) {
        protected Thread t = new Thread(new Test2());
        Thread t2 = new Thread(new Test2());

        t.start();
        t2.start();
    }

    public void main() {
        for(int i = 0; i < 2; i++)
            System.out.println(Thread.currentThread().getName()+" ");
    }
}
```

El código no compila ya que no es posible colocar modificadores de acceso a las variables locales (las que se encuentran dentro de los métodos).

34. Comando Docker que se utiliza para construir la imagen a partir del Dockerfile.

`docker build`

05. Seleccione la respuesta que considere correcta dado el siguiente bloque de código.

```
import java.util.Arrays;

public class Example {
    public static void main(String[] args ) {
        int[][][] matrix= {{{ 1, 2}, {3, 4}}, {{5, 6}, {7, 8}}};
        int[][] flattened= Arrays.stream(matrix)
                        .flatMapoIn(layer)Arrays.string(layer)
                        .flatMapToIn(Arrays::stream))
                        .boxed().map(n -> new int[](n)).toArray(int[] []::
        System.out.println(Arrays.deepToString(flattened));
    }
}
```

El código está mal escrito pero la intención es de quitarle una dimensión a `matrix`, por lo cual la respuesta correcta es `[[1, 2], [3,4], [5,6], [7,8]]`.