

56. Son patrones de diseño de microservicios

Investigación:

Estos patrones abordan diferentes aspectos y preocupaciones en el diseño de sistemas distribuidos y desacoplados. Aquí hay una breve explicación de cada uno:

Retry: Este patrón se utiliza para manejar fallas temporales o intermitentes en los servicios. Cuando una operación falla, el cliente (otro servicio) intenta volver a ejecutar la operación después de un retraso, con la esperanza de que la falla sea transitoria. Este patrón ayuda a mejorar la resiliencia del sistema.

Circuit Breaker: El patrón Circuit Breaker previene que un servicio siga intentando operaciones fallidas contra un servicio remoto que podría estar inactivo.

Básicamente, después de un número determinado de fallas, el "circuito" se abre y todas las llamadas posteriores fallan rápidamente en lugar de intentar una operación que probablemente falle. Esto protege al servicio cliente de saturarse con solicitudes fallidas.

Adaptative Lifo (Adaptative Life Cycle Oriented): Este patrón propone dividir los microservicios en dos partes: una parte sin estado (stateless) que maneja las solicitudes entrantes y una parte con estado (stateful) que contiene la lógica de negocio. Esto permite escalar y replicar la parte sin estado de forma independiente de la parte con estado, lo que puede mejorar el rendimiento y la escalabilidad.

Bulkhead: Similar al patrón Circuit Breaker, pero aplicado a recursos en lugar de servicios remotos. Este patrón aísla los recursos (como conexiones de base de datos, hilos, etc.) en grupos o "compartimentos" separados. Si uno de los compartimentos falla o se satura, los otros compartimentos no se ven afectados, lo que evita que una falla se propague por todo el sistema.

Estos patrones se utilizan comúnmente en conjunto para construir sistemas de microservicios altamente resilientes, escalables y tolerantes a fallas. Abordan problemas como la gestión de fallas, el aislamiento de recursos, la escalabilidad y el rendimiento en entornos distribuidos.

- a) Circuit Breaker, Adaptative Lifo, MQ Strategy
- b) System, Process y Client
- c) **Retry, Circuit Breaker, Adaptative Lifo y Bulkhead.**
- d) Ninguna de las anteriores

57. ¿Qué afirmaciones son verdaderas tanto para las clases abstractas como para las interfaces? (Elije todas las correctas)

- a) **Ambos pueden contener métodos estáticos.**
- b) **Ambos se pueden ampliar con la clave extend. ← Se puede**
- c) Ambos pueden contener métodos predeterminados. ← No estamos seguros
- d) Ambos heredan de java.lang.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.

58. ¿Cuál no es un objetivo de Maven?

Investigación:

Los objetivos principales de Maven son:

- a) **Clean:** Este objetivo se utiliza para limpiar los archivos generados durante la compilación y empaquetado del proyecto.
- b) **Package:** Este objetivo se utiliza para empaquetar el código compilado y los recursos del proyecto en un formato de distribución, como un archivo JAR, WAR o EAR, dependiendo del tipo de proyecto.
- d) **Install:** Este objetivo se utiliza para instalar el paquete generado en el repositorio local de Maven, para que pueda ser utilizado como dependencia por otros proyectos.

Además de estos, otros objetivos comunes de Maven son:

- **Compile:** Compila el código fuente del proyecto.
- **Test:** Ejecuta los casos de prueba unitarios del proyecto.
- **Deploy:** Copia el paquete final en un repositorio remoto para compartirlo con otros desarrolladores y proyectos.

Sin embargo, Maven no tiene un objetivo específico llamado "Debug". El depurado (debugging) se realiza normalmente a través de una herramienta de desarrollo integrada (IDE) o utilizando herramientas de depuración externas, como un depurador remoto.

- a) Clean
- b) Package
- c) Debug
- d) Install

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

Investigación:

El comando git clone se utiliza para obtener una copia completa de un repositorio Git existente en un servidor remoto.

Explicación:

git clone crea una copia local de un repositorio remoto en tu máquina local. Este comando descarga todo el historial de commits, ramas y archivos del repositorio especificado.

La sintaxis básica del comando es: git clone [url-del-repositorio]. Donde [url-del-repositorio] es la dirección URL del repositorio remoto que deseas clonar.

Después de ejecutar git clone, tendrás una copia completa del repositorio en tu máquina local y podrás trabajar en ella, realizar cambios, crear ramas, etc.

- a) git commit: se utiliza para confirmar los cambios realizados en el repositorio local y crear un nuevo commit.
- b) git log: muestra el historial de commits del repositorio.
- d) git add: agrega archivos al área de preparación (staging area) para ser incluidos en el próximo commit.

- a) git commit
- b) git log
- c) git clone
- d) git add

60. ¿Qué es un repositorio remoto en Git?

Investigación:

Un repositorio remoto en Git es un servidor Git que actúa como un repositorio central y almacena una copia del código fuente de un proyecto. Este repositorio remoto se utiliza para compartir y colaborar en el código fuente entre múltiples desarrolladores o equipos.

- a) Una herramienta que se utiliza para compartir y fusionar cambios entre diferentes ramas de un repositorio.
- b) Una copia local de un repositorio que se utiliza para hacer cambios en el código fuente.
- c) Un servidor Git que almacena una copia central del repositorio.
- d) Un archivo que contiene una instantánea del código fuente en un momento determinado.

61. Dada la siguiente clase

```
public class Helper {  
    public static < U extends Exception > void  
        printException(U u){  
        System.out.println(u.getMessage());  
    }  
  
    public static void main(String[] args) {  
        //línea 9  
    }  
}
```

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

- a) `Helper.printStackTrace(new Exception("B"));`
- b) `Helper.printStackTrace(new FileNotFoundException("A"));`
- c) `Helper.<Throwable>printException(new Exception("C"));`
- d) `Helper.<NullPointerException>printException(new NullPointerException("D"));`
- e) `Helper.printStackTrace(new Throwable("E"));`

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

```
public class Fish {  
    public static void main(String[] args) {  
        int numFish = 4;  
        String fishType = "tuna";  
        String anotherFish = numFish + 1;  
        System.out.println(anotherFish + " " + fishType);  
        System.out.println(numFish + " " + 1);  
    }  
}
```

- a) 51tuna
- b) 5tuna
- c) 5
- d) 41
- e) 5 tuna
- f) 4 1
- g) El código no compila

63. ¿Cuál de las siguientes opciones son correctas? (Elija todas las correctas)

```
public class StringBuilders {  
    1 usage  
    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`
- b) `s3 = null`
- c) `s1 = s1`

- d) s3 = a
- e) El código no compila
- f) s2 = s2b
- g) s1 = a

64. ¿A qué hace referencia el principio de Liskov?

- a) Nos indica que una clase no debe tener solo una funcionalidad sino varias para reducir el uso de objetos.
- 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.
- c) Nos indica que cualquier clase se puede extender para agregar funcionalidad, pero no se puede modificar.
- d) Este principio nos indica que dentro del programa una clase puede ser sustituida por su clase padre sin alterar el comportamiento del programa.

65. ¿Qué es un “code smell”?

- a) Un componente de la biblioteca estándar de Java
- b) Un error en tiempo de compilación que se produce en Java
- c) Un indicador de que puede 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.
- d) Una práctica de programación recomendada en Java.

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

Investigación:

Se realizan con las anotaciones Get para lectura y mostrar datos, Post para crear datos o registros, Put para editar un registro, Patch para hacer una actualización parcial y Delete para eliminar registros.

- a) Code, Register, Update, Debug
- b) Create, Read, Update, Delete
- c) Call, Receive, Use, Debug
- d) Customize, Request, Use, Debug

67. ¿Para qué nos sirve utilizar un profile dentro del archivo pom.xml?

Investigación:

Los perfiles (profiles) en Maven son una forma de personalizar la construcción y el empaquetado de un proyecto en función de diferentes entornos o configuraciones.

Los perfiles en Maven se definen dentro de la etiqueta <profiles> en el archivo pom.xml. Cada perfil tiene un identificador (<id>) y puede contener configuraciones específicas, como propiedades personalizadas, dependencias, plugins, repositorios, etc.

Los perfiles se pueden activar manualmente o mediante la detección de ciertas condiciones, como el sistema operativo, la presencia de una propiedad, un archivo específico, etc.

Las principales razones para usar perfiles en Maven incluyen:

Configuraciones específicas de entorno (desarrollo, pruebas, producción).

Compilación y empaquetado con diferentes conjuntos de dependencias.

Activar plugins específicos para ciertas tareas (como herramientas de cobertura de código o análisis de código estático).

Personalizar la construcción para diferentes sistemas operativos o arquitecturas.

a) Etiqueta por la cual podemos definir la versiones de nuestras dependencias.

b) Es la etiqueta por la cual podemos definir las características que tendrá nuestro proyecto al ser compiladas.

c) Etiqueta por la cual definimos los parámetros de conexión a un repositorio.

d) No existe esta etiqueta en Maven.

68. Dadas las siguientes clases Vehicle y Car

```
package my.vehicles;

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

```
package my.vehicles.cars;  
  
import my.vehicles.*;  
  
public class Car extends Vehicle {  
    public Car() {  
        //línea 7  
    }  
}
```

¿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;
- b) Ninguna de las anteriores.
- c) make = "Honda";
- d) year = 2009;
- e) model = "Pilot";

69. Enumere cuatro interfaces de la API colecciones

Investigación:

List: Es una colección ordenada de elementos que permite duplicados. Las implementaciones comunes son ArrayList y LinkedList.

Map: Es una colección de pares clave-valor, donde las claves son únicas. Las implementaciones comunes son HashMap y TreeMap.

Set: Es una colección que no permite elementos duplicados. Las implementaciones comunes son HashSet y TreeSet.

Queue: Es una colección diseñada para mantener el orden de inserción de los elementos. Las implementaciones comunes son LinkedList y PriorityQueue.

- a) List, Map, Set, Queue.
- b) ArrayList, Map, Set, Queue.
- c) List, HashMap, HashSet, PriorityQueue.
- d) List, Map, HashSet, PriorityQueue.

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

```
public class Test5 {  
  
    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 ");}  
}
```

- a) Head
Object
Tail
Side
- b) No compila
- c) Side
Object
Tail
Side
- d) Head
Head
Tail
Tail
- e) Side
Head
Tail
Side

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);  
    }  
}
```

- a) roar roar!!!
- b) roar!!! Roar
- c) Se lanza una excepción
- d) roar!!! roar!!!
- e) roar roar
- f) El código no compila

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

Investigación:

Una clase concreta en Java es una clase que proporciona implementaciones completas para todos sus métodos declarados. A diferencia de las clases abstractas, las clases concretas no tienen métodos abstractos y pueden ser instanciadas directamente.

Una subclase concreta es una clase que extiende (hereda) de otra clase, ya sea concreta o abstracta.

Si la superclase (clase base) contiene métodos abstractos, la subclase concreta está obligada a proporcionar implementaciones completas para esos métodos abstractos heredados.

De lo contrario, la subclase concreta también se convertiría en una clase abstracta y no podría ser instanciada.

- a) Una subclase concreta no se puede marcar como final.
- b) Una subclase concreta debe implementar todos los métodos definidos en una interfaz heredada.
- c) Una subclase concreta debe implementar todos los métodos abstractos heredados.
- d) Una subclase concreta puede declararse como abstracta.
- e) Los métodos abstractos no pueden ser anulados por una subclase concreta.

73. ¿Cuál es la salida del siguiente código?

```
1  public abstract class Catchable {
2      protected abstract void catchAnObject(Object x);
3
4      public static void main(String [] args) {
5          java.util.Date now = new java.util.Date();
6          Catchable target = new MyStringCatcher();
7          target.catchAnObject(now);
8      }
9  }
10
11 class MyStringCatcher extends Catchable {
12     public void catchAnObject(Object x) {
13         System.out.println("Caught object");
14     }
15
16     public void catchAnObject(String s) {
17         System.out.println("Caught string");
18     }
19 }
```

- a) Error de compilación línea 12
- b) Error compilación línea 16
- c) Caught string
- d) Error compilación línea 2
- e) Caught Object

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

```
1  import java.util.Arrays;
2  import java.util.List;
3
4  public class Example {
5
6      public static void main(String[] args) {
7          List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
8
9          int result = numbers.stream()
10             .filter(n -> n % 2 == 0)
11             .reduce(0, (a, b) -> a + b);
12
13          System.out.println(result);
14
15      }
16  }
17
```

- a) 3
- b) 9
- c) 14
- d) 6

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

- a) #include java.util.*;
- b) #include java.util;
- c) import java.util.*;
- d) import java.util;

76. ¿Qué es la cobertura de código?

Investigación:

La cobertura de código (code coverage) es una métrica que se utiliza en la prueba de software para medir la cantidad de código fuente que se ejecuta durante una suite de pruebas.

La cobertura de código generalmente se expresa como un porcentaje que indica qué proporción del código fuente se ha ejecutado durante las pruebas. Un alto porcentaje de cobertura de código indica que la mayor parte del código fuente se ha probado y ejecutado, lo cual aumenta la confianza en la calidad del software.

- a) La cantidad de veces que se ejecuta una línea de código.
- b) La cantidad de errores detectados por una prueba.
- c) La cantidad de código que se ejecuta durante una prueba.
- d) La cantidad de tiempo que tarda una prueba en ejecutarse.

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

Investigación:

Git recomienda un formato específico para los mensajes de commit, que facilita la lectura y comprensión de los cambios realizados. Este formato se conoce como "Conventional Commits" y consta de las siguientes partes:

Tipo de cambio: Un prefijo que describe la naturaleza del cambio. Algunos ejemplos comunes son: feat (nueva característica), fix (corrección de error), docs (cambios en la documentación), refactor (refactorización de código), test (adición o modificación de pruebas), entre otros.

Descripción breve: Una breve descripción del cambio, generalmente en una sola línea de no más de 72 caracteres.

Cuerpo opcional: Una descripción más detallada del cambio, si es necesario. Debe estar separada de la descripción breve por una línea en blanco.

Notas de pie de página: Información adicional, como la referencia a un problema o una tarea específica.

- a) Descripción breve del cambio y nombre del autor.
- b) Tipo de cambio, descripción breve, cuerpo opcional y notas de pie de página.
- c) Solo se necesita una breve descripción del cambio.
- d) Nombre de la rama, descripción detallada del cambio y fecha.

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

Investigación:

el patrón de diseño Singleton es uno de los patrones más conocidos y utilizados en la programación orientada a objetos. Su principal objetivo es garantizar que una clase tenga una única instancia y proporcionar un punto de acceso global a ella.

Explicación del patrón Singleton:

El patrón Singleton se basa en los siguientes principios:

Clase única: Sólo existe una instancia de la clase Singleton en toda la aplicación.

Constructor privado: El constructor de la clase Singleton es privado para evitar que otras clases puedan crear instancias adicionales.

Método estático para acceder a la instancia: La clase Singleton proporciona un método estático que permite acceder a la única instancia disponible.

Inicialización retrasada (Lazy Initialization): La instancia de la clase Singleton no se crea hasta que se necesite por primera vez, lo que mejora el rendimiento y ahorra recursos.

El patrón Singleton es útil en situaciones donde necesitas un control estricto sobre la creación de instancias de una clase, como en la gestión de recursos compartidos, caché, registros, configuraciones globales, entre otros casos de uso. Sin embargo, también es importante tener en cuenta que el uso excesivo de Singletons puede dificultar la prueba unitaria y la modularidad del código, por lo que se debe considerar cuidadosamente su aplicación.

- 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.
- b) El patrón de diseño Singleton es un patrón que se utiliza para abstraer la capa de infraestructura en una aplicación. Se implementa utilizando excepciones y bloques try-catch.
- c) El patrón de diseño Singleton es un patrón que se utiliza para abstraer la capa de presentación en una aplicación. Se implementa con interfaces y clases concretas.
- d) El patrón de diseño Singleton es un patrón que se utiliza para abstraer la capa de negocios en una aplicación. Se implementa utilizando clases abstractas y métodos estáticos.

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

- a) Son exactamente iguales, no hay diferencia de uso.
- b) PATCH requiere se le envíe la entidad completa mientras que PUT solo los atributos a modificar.
- c) PUT requiere se le envíe la entidad completa mientras que PATCH solo los atributos a modificar.
- d) PATCH es un verbo deprecado sustituido por PUT.

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

Investigación:

Estas cuatro anotaciones se utilizan en el marco de trabajo de Spring para marcar clases como beans y agregarlas al contexto de la aplicación Spring. La diferencia principal entre ellas es semántica, lo que significa que indican el propósito o la intención del bean en la arquitectura de la aplicación.

`@Component`: Es una anotación genérica para marcar una clase como un bean de Spring. Puede ser utilizada para cualquier tipo de clase que deba ser manejada por el contenedor de Spring.

`@Service`: Esta anotación se utiliza para marcar clases que representan la capa de servicios en la arquitectura de la aplicación. Es una anotación más específica y semánticamente indica que la clase es un servicio.

`@Repository`: Se utiliza para marcar clases que interactúan con la capa de persistencia de datos, como las clases que acceden a la base de datos o a otros repositorios de datos. Proporciona funcionalidades adicionales para el manejo de excepciones y transacciones.

`@RestController`: Esta anotación es una combinación de `@Controller` y `@ResponseBody`. Se utiliza para marcar clases como controladores REST, lo que significa que manejan las solicitudes HTTP y devuelven respuestas en formato JSON o XML. Es específica para la construcción de APIs RESTful.

Todas estas anotaciones funcionan de la misma manera en términos de crear beans y agregarlos al contexto de la aplicación Spring. La diferencia radica en la semántica y la intención de uso de cada anotación, lo que ayuda a comprender mejor la estructura y el propósito de las clases en la arquitectura de la aplicación.

Es importante tener en cuenta que `@RestController` también es una anotación de Spring que crea un bean, al igual que las otras tres anotaciones mencionadas.

- a) `@Controller` es una anotación que nos ayuda a construir una API REST mientras que `@Service`, `@Component` y `@Repository` solo marcan las clases que se deben de inicializar.
- b) `@Controller`, `@Component` son anotaciones que crean bean y exponen la serialización de las clases mientras que `@Service` y `@Repository` requieren de una inicialización manual.
- 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.
- d) `@Service` y `@Repository` son anotaciones que crean un bean y exponen la serialización de las clases mientras que `@Controller`. `@Component` requiere de una inicialización manual.

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

Investigación:

Las pruebas unitarias (unit testing) son un tipo de pruebas de software que se enfocan en comprobar el correcto funcionamiento de unidades individuales de código, como métodos o funciones. El objetivo principal de las pruebas unitarias es aislar y probar cada componente de forma independiente, verificando su comportamiento y resultados esperados.

- 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.
- d) Hacer que las pruebas dependan de otras pruebas.

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

Investigación:

Las APIs REST (Representational State Transfer) son un estilo arquitectónico para construir servicios web que permiten la comunicación entre diferentes sistemas a través de Internet utilizando el protocolo HTTP.

Las APIs REST se basan en los conceptos clave de recursos, representaciones, mensajes y transferencias de estado. Estas APIs utilizan los métodos HTTP (GET, POST, PUT, DELETE, etc.) para realizar operaciones sobre los recursos, y los recursos se identifican mediante URLs (Uniform Resource Identifiers).

Las principales ventajas de utilizar APIs REST sobre otros tipos de servicios web son su facilidad de implementación y su amplia compatibilidad con diferentes plataformas y tecnologías, lo que las convierte en una opción popular para construir aplicaciones web y móviles modernas.

- a) Mayor seguridad.
- b) Mayor facilidad de implementación. ← DUDA
- c) Mayor velocidad de transferencia de datos.
- d) Mayor compatibilidad con diferentes plataformas. ← DUDA

83. 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));  
    }  
}
```

- a) No compila
- b) [25, 7, 25, 67]
[67, 7, 25]
[7, 25, 67]
[67, 7, 25]
[7, 25, 67]
- c) [25, 7, 67]
[67, 7, 25]
[7, 25, 67]
[67, 7, 25]
[7, 25, 67]
- d) [67, 7, 25]
[67, 7, 25]
[67, 7, 25]
[67, 7, 25]
[67, 7, 25]
- e) [25, 7, 25, 67]
[7, 25, 67]
[67, 7, 25]
[7, 25, 67]
[67, 7, 25]

84. ¿Cuáles son los 4 pilares de la programación orientada a objetos?

- a) Polimorfismo, Coerción, Herencia y Encapsulamiento.
- b) Encapsulamiento, Coerción, Polimorfismo y Abstracción.
- c) Polimorfismo, Herencia, Encapsulamiento y Sincronía.
- d) Polimorfismo, Abstracción, Herencia y Encapsulamiento.

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

- a) jconsole
- b) javaw
- c) interpw
- d) java -wo

86. ¿Qué es un endpoint en una API REST?

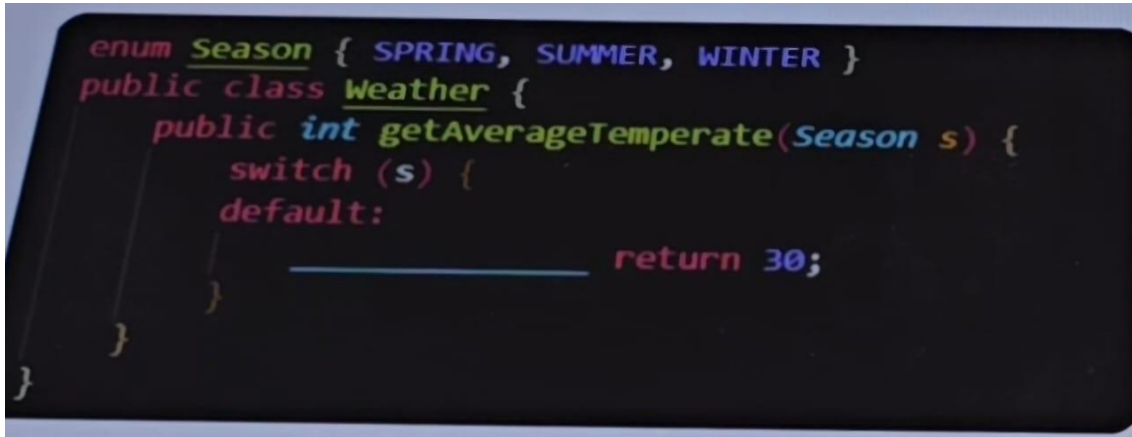
- a) Un endpoint es un objeto que se utiliza para almacenar datos en una API REST.
- b) Un endpoint es un método que se utiliza para procesar datos en una API REST.
- c) Un endpoint es un controlador que se utiliza para administrar una API REST.
- d) Un endpoint es la URL que se utiliza para acceder a una API REST.

87. ¿Cuál es el valor de x e y al final el programa?

```
int x = 0;
do {
    System.out.println(x);
    x++;
} while (x < 10);
int y = 0;
while (y < 10) {
    System.out.println(y);
    y++;
}
```

- a) X=9 y=10
- b) X=10 y=9
- c) X=10 y=10
- d) X=9 y=9

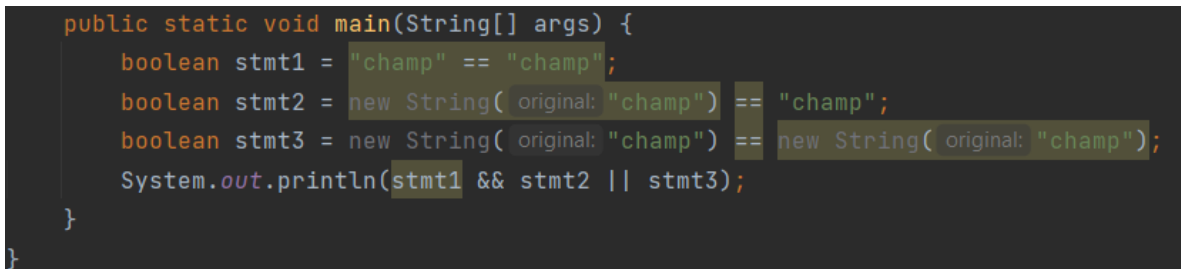
88. ¿Dado el siguiente enum y clase cuál es la opción que puede ir en el espacio en blanco para que el código compile?



```
enum Season { SPRING, SUMMER, WINTER }
public class Weather {
    public int getAverageTemperature(Season s) {
        switch (s) {
            default:
                _____ return 30;
        }
    }
}
```

- a) Ninguno de los anteriores
- b) case SUMMER ->
- c) case Season.Winter:
- d) case FALL:
- e) case Winter, Spring:
- f) case SUMMER | WINTER:

89. ¿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( original: "champ") == "champ";
    boolean stmt3 = new String( original: "champ") == new String( original: "champ");
    System.out.println(stmt1 && stmt2 || stmt3);
}
```

- a) False
- b) no se produce salida
- c) true
- d) error de compilación

90. ¿Cómo se manejan las excepciones en Java?

- a) Las excepciones se manejan con bloques switch case en Java. La excepción `try with Resources` es una forma de lanzar una excepción en un método.
- b) Las excepciones se manejan con bloques while en Java. La excepción `try with Resources` es una forma de manejar excepciones de compilación
- c) las excepciones se manejan con bloques if else en Java. La excepción `try with resource` es una forma de manejar las excepciones en tiempo de ejecución.
- d) Las excepciones 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`
- 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? (Elija 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) Ninguna, el código compila bien
- b) línea 6
- c) línea 9
- d) línea 7
- e) línea 8
- f) línea 5

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

Aquí dejo el código que muestra las líneas que tienen error.

93. ¿Qué es una expresión lambda en Java?

- a) Una instancia de una clase que implementa una interfaz funcional
- b) Una instancia de una clase abstracta que se utiliza para implementar métodos anónimos
- c) Una forma concisa de representar una función anónima que se puede pasar como argumento ← Es correcto
- d) Un método que no tiene cuerpo

94. ¿Qué hace el siguiente código fuente?

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

- a) Muestra los números del 1 al 10
- b) muestra un 10
- c) se queda en un bucle infinito ← Es correcto
- d) muestra los números del 0 al 9

95. ¿Qué son las anotaciones en java?

- a) Una forma de declarar variables en Java.
- b) Una forma de declarar métodos abstractos en Java.
- c) Un mecanismo para etiquetar y procesar código de forma especial.
- d) Comentarios especiales que se utilizan para documentar el código.

96. ¿Qué son las expresiones regulares en Java?

- a) Un mecanismo para validar y manipular fechas y horas.
- b) Un mecanismo para validar y manipula cadenas de caracteres.
- c) Un mecanismo para validar y manipular números enteros.
- d) Un mecanismo para validar y manipular números decimales.

97. ¿Qué es una anotación en Spring?

- a) Una biblioteca de terceros que se utiliza para extender la funcionalidad de Spring
- b) Una clase que se utiliza para definir la estructura de una tabla de base de datos.
- c) Una etiqueta que se utiliza para anotar una clase o un método y proporcionar información adicional al contenedor de Spring.
- d) Un archivo de configuración XML que se utiliza para configurar una aplicación de Spring.

98. ¿Cuál es la salida?

```
import java.util.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 :long = list.stream()
            .filter(a -> a.length() > 10) //línea x
            .count();
        System.out.println(c + " " + list.size());
    }
}
```

- a) Ninguna de las anteriores
- b) 1 3
- c) 1 1
- d) El código no compila en la línea x
- e) 2 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());
    }
}
```

- a) 5
- b) 7
- c) El código compila sin problemas la salida no se puede determinar hasta el tiempo de ejecución.
- d) Ninguna de las anteriores.
- e) El código no compila

100. ¿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?
- a) private
 - b) const
 - c) **final**
 - d) private static
 - e) Ninguna de las anteriores
 - f) Static
101. ¿Qué tipo de excepción se produce cuando se intenta realizar una operación incompatible con el tipo de datos en Java?
- a) ArrayIndexOutOfBoundsException
 - b) ArithmeticException
 - c) IllegalArgumentException
 - d) **ClassCastException**
102. ¿Qué es un starter?
- a) **Es una herramienta que nos facilita la creación y configurar un proyecto cargando las dependencias necesarias para un objetivo.**
 - b) Es el inicializador de un proyecto en Java.
 - c) Componente encargado de realizar las operaciones de balanceador.
103. 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()+" ");
    }
}
```

La respuesta es: **NO COMPILA**

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

docker build

2. ¿Selecione 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)  
            .flatMapToIn(layer)Arrays.string(layer)  
            .flatMapToIn(Arrays::stream)  
            .boxed().map(n -> new int[] (n)).toArray(int[] []::new);
```

```
        System.out.println(Arrays.deepToString(flattened));  
    }  
}
```

[[1], [2], [3], [4], [5], [7], [8]]

[[1, 2]], [[3, 4]], [[5, 6]], [[7, 8]]

[[1, 2], [3,4], [5,6], [7,8]]