1. Son patrones de diseño de microservicios

    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

- Retry: Reintenta una operación que ha fallado, esperando que el problema sea transitorio.
- Circuit Breaker: Previene que una aplicación intente realizar una operación que probablemente fallará, protegiendo así al sistema de sobrecargas.
- Adaptative Lifo: Administra colas de mensajes o tareas de manera adaptativa utilizando una política LIFO (Last In, First Out).
- Bulkhead: Aisla diferentes partes del sistema para prevenir que un fallo en una parte afecte a las demás.

- MQ Strategy: No es un patrón de diseño de microservicios conocido. Aunque MQ (Message Queue) se utiliza en arquitectura de microservicios, no es un patrón de diseño por sí mismo, sino una tecnología o componente que se puede utilizar dentro de un patrón de diseño más amplio como el patrón de mensajería.
- System: No es un patrón de diseño, es un término genérico que puede referirse a cualquier cosa en la arquitectura.
- Process: No es un patrón de diseño, es un concepto que se refiere a una instancia de un programa en ejecución.
- Client: No es un patrón de diseño, es un término que se refiere a una parte del sistema que consume servicios de otro (como un cliente en una arquitectura cliente-servidor).

2. ¿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.
    c) Ambos pueden contener métodos predeterminados.
    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.

a) Verdadero. Desde Java 8, las interfaces pueden contener métodos estáticos. Las clases abstractas también pueden contener métodos estáticos.

b) Verdadero. Las clases abstractas se amplían usando la palabra clave extends, pero las interfaces se implementan usando la palabra clave implements, aunque una interfaz puede extender otra interfaz.

c) Verdadero. Desde Java 8, las interfaces pueden contener métodos predeterminados (default methods). Las clases abstractas también pueden contener métodos con implementación (no abstractos), lo que las hace equivalentes a los métodos predeterminados en interfaces.

d) Falso. Todas las clases en Java heredan de java.lang.Object directa o indirectamente. Las interfaces no heredan de java.lang.Object, aunque los métodos definidos en java.lang.Object están disponibles en las interfaces.

e) Verdadero. Las clases abstractas no pueden ser instanciadas directamente. Las interfaces tampoco pueden ser instanciadas directamente.

f) Verdadero. Las interfaces pueden contener variables que son implícitamente public, static y final. Las clases abstractas pueden contener variables estáticas finales públicas explícitamente.

g) Falso. En las clases abstractas, no todos los métodos tienen que ser abstractos; pueden tener métodos con implementación. En las interfaces, desde Java 8, pueden tener métodos predeterminados y estáticos con implementación, además de los métodos abstractos tradicionales.

3. ¿Cuál no es un objetivo de Maven?

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

- Clean: Elimina los archivos generados previamente.
- Package: Empaqueta el proyecto en un formato específico, como JAR o WAR.
- Install: Instala el paquete en el repositorio local para su uso como dependencia en otros proyectos.

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

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

El comando git clone se utiliza para obtener una copia de un repositorio Git existente en un servidor. Este comando clona el repositorio remoto en tu máquina local, permitiéndote trabajar con el código y contribuir al proyecto si tienes los permisos adecuados.

5. ¿Qué es un repositorio remoto en Git?

    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.

Un repositorio remoto en Git es una versión del proyecto que se encuentra alojada en un servidor Git, como GitHub, GitLab o Bitbucket. Este servidor actúa como un repositorio centralizado al que múltiples desarrolladores pueden acceder y colaborar en el mismo proyecto. Los repositorios remotos permiten compartir cambios entre diferentes colaboradores y mantener una versión centralizada del código fuente.

6. 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) {
        // linea 9
    }
}
```

¿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) Helper.<Throwable>printException(new Exception("C"));
d) Helper.<NullPointerException>printException(new NullPointerException ("D"));
e) Helper. printException(new Throwable("E"));

```
import java.io.FileNotFoundException;

public class Helper {
    public static <U extends Exception> void printException(U u) {
        System.out.println(u.getMessage());
    }

    public static void main(String[] args) {
        Helper.printException(new Exception("B"));
        Helper.printException(new FileNotFoundException("A"));
//Sin import no compila
        Helper.<Throwable>printException(new Exception("C"));
//Bound mismatch: The generic method printException(U) of type pregunta 61 is not
//applicable for the arguments (Exception). The inferred type Throwable is not a valid
//substitute for the bounded parameter <U extends Exception>
        Helper.<NullPointerException>printException(new NullPointerException("D"));
        Helper.printException(new Throwable("E"));
//The method printExcepton in the type Helper is not applicable for the arguments (Throwable)
    }
}
```

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

```java
public class pregunta62 {
    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

```java
String anotherFish = numFish + 1; //Type mismatch: cannot convert from int to String
```

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

```java
public class pregunta63 {
    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

9. ¿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.

Este principio establece que los objetos de una clase hija deben poder ser sustituidos por objetos de su clase padre sin interrumpir el comportamiento del programa. Es decir, los objetos de una subclase deben ser utilizables donde se esperan objetos de la clase base sin necesidad de conocer la diferencia. Esto implica que las subclases deben extender y especializar, no restringir o modificar, el comportamiento de sus clases base.

10. ¿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 dificil 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.

Un "code smell" es una señal o indicador en el código fuente que sugiere la presencia de un problema más profundo. No necesariamente indica un error en sí mismo, pero podría ser una señal de que el código podría estar mejorado o refactorizado para ser más claro, mantenible y eficiente. Identificar y abordar los "code smells" es una parte importante del proceso de desarrollo de software para mantener la calidad del código y evitar problemas en el futuro.

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

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

- Create: Crear nuevos registros o recursos.
- Read: Leer o recuperar registros existentes.
- Update: Actualizar registros existentes.
- Delete: Eliminar registros existentes.

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

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.

Un profile en un archivo pom.xml de Maven nos permite definir conjuntos de configuraciones y tareas que se activan o desactivan en función de ciertas condiciones. Esto puede ser útil para definir diferentes configuraciones de compilación, pruebas o despliegue según el entorno (por ejemplo, desarrollo, pruebas, producción) o según otros factores específicos del proyecto. Los perfiles pueden incluir especificaciones como dependencias adicionales, configuraciones de plugins, variables de entorno, entre otros.

13. 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() {
        // linea 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”;

```
package my.vehicles.cars;

import my.vehicles.*;

public class Car extends Vehicle {
    public Car() {
        mileage = 15285; //The field Vehicle.mileage is not visible
        make = "Honda";
        year = 2009;  //The field Vehicle.year is not visible
        model = "Pilot";
    }
}
```

14. Enumere cuatro interfaces de la API colecciones

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

- List: Representa una colección ordenada y puede contener elementos duplicados.
- Map: Representa una asociación entre claves y valores únicos.
- Set: Representa una colección que no permite elementos duplicados.
- Queue: Representa una colección ordenada de elementos que sigue el principio FIFO (First In, First Out).

- ArrayList, HashMap, HashSet y PriorityQueue no son interfaces, sino implementaciones de las interfaces List, Map, Set y Queue, respectivamente.

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

```
public class pregunta70 {
    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

16. ¿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

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

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.

Una subclase concreta es una clase que proporciona implementaciones concretas para todos los métodos abstractos heredados de sus clases ascendentes o interfaces implementadas. Por lo tanto, debe implementar todos los métodos abstractos heredados para ser considerada como una clase concreta válida.

18. ¿Cuál es la salida del siguiente código?

```
public abstract class pregunta73 {
    protected abstract void catchAnObject(Object x);

    public static void main(String[] args) {
        java.util.Date now = new java.util.Date();
        pregunta73 target = new MyStringCatcher();
        target.catchAnObject(now);
    }
}

class MyStringCatcher extends pregunta73 {
    public void catchAnObject(Object x) {
        System.out.println("Caught object");
    }

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

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

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

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

public class pregunta74 {
    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);
    }
}
```

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

20. ¿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;

La palabra clave import en Java se utiliza para incluir clases de otros paquetes en tu código. Al usar import java.util.*;, estás importando todas las clases del paquete java.util, lo que te permite usarlas en tu código sin tener que hacer referencia completa a cada una de ellas.

21. ¿Qué es la cobertura de código?

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.

La cobertura de código es una medida que indica cuánto del código fuente de un programa ha sido ejecutado durante la ejecución de un conjunto de pruebas. Mide la cantidad de líneas de código, declaraciones, ramas condicionales, métodos o clases que se han ejecutado durante la ejecución de pruebas automatizadas. Es una métrica importante en la evaluación de la efectividad de las pruebas y en la identificación de áreas de código que no están siendo probadas adecuadamente.

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

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.

El formato convencional para un mensaje de commit en Git sigue la estructura de tipo de cambio, descripción breve, cuerpo opcional y notas de pie de página. Este formato facilita la comprensión rápida del propósito y los detalles del commit, lo que ayuda a mantener un historial de cambios claro y organizado.

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

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.

El patrón Singleton se utiliza para asegurar que una clase tenga solo una instancia y proporciona un punto de acceso global a esa instancia. En Java, se implementa haciendo que el constructor de la clase sea privado para evitar que se creen instancias directamente y proporcionando un método estático para obtener la única instancia de la clase. Una variable estática dentro de la clase mantiene la única instancia y la devuelve cuando se solicita mediante el método estático. Esto garantiza que solo exista una instancia de la clase Singleton en todo el sistema.

24. 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.

En REST, el verbo PUT se utiliza para actualizar un recurso completo, lo que significa que se espera que se proporcione toda la entidad con los cambios. Por otro lado, el verbo PATCH se utiliza para realizar modificaciones parciales en un recurso, lo que significa que solo se deben proporcionar los atributos que se desean modificar, en lugar de la entidad completa.

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

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.

Las cuatro anotaciones: @RestController, @Component, @Service y @Repository son anotaciones de Spring que tienen el propósito de crear beans y agregarlos al contexto de Spring. La diferencia principal entre ellas es semántica y convencional, es decir, cómo se utilizan y cuál es su propósito principal en el código. Sin embargo, todas tienen un efecto similar en términos de crear beans y gestionar su ciclo de vida en la aplicación Spring.

26. ¿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.
d) Hacer que las pruebas dependan de otras pruebas.

Es una buena práctica asegurarse de que las pruebas unitarias sean claras y concisas para que sean fáciles de entender y mantener. Las pruebas unitarias deben ser específicas, centradas en una sola funcionalidad o comportamiento, y deben proporcionar una validación clara de la funcionalidad que están probando. Esto facilita la identificación y corrección de errores, así como la comprensión de la funcionalidad que están probando. Las pruebas unitarias también deben ser lo más independientes posible, sin depender de otros factores externos, para que puedan ejecutarse de manera confiable en cualquier entorno.

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

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

Las APIs REST son más simples de implementar en comparación con otros tipos de servicios web como los servicios web SOAP. Utilizan protocolos web estándar como HTTP y pueden ser accedidas mediante URLs, lo que simplifica la implementación y el consumo de los servicios. Esto facilita el desarrollo de APIs REST y reduce el tiempo necesario para implementar y mantener dichas APIs.

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

```java
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.TreeSet;
import java.util.concurrent.ConcurrentSkipListSet;

public class pregunta83 {
    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]

29. ¿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.

- Abstracción: Capacidad de representar conceptos del mundo real en un modelo simplificado en el código. Se logra a través de clases y objetos. Las clases definen las propiedades y comportamientos comunes de un conjunto de objetos, mientras que los objetos son instancias específicas de esas clases. La abstracción nos permite centrarnos en los detalles relevantes para el problema que estamos resolviendo, ignorando los detalles innecesarios.
- Encapsulamiento: Ocultación de los detalles internos de un objeto y la exposición de una interfaz clara y consistente para interactuar con él. Esto se logra definiendo propiedades privadas en una clase y proporcionando métodos públicos para acceder y modificar esas propiedades. El encapsulamiento ayuda a proteger la integridad de los datos y facilita la gestión y el mantenimiento del código al limitar el acceso directo a las propiedades de un objeto.
- Herencia: Mecanismo que permite que una clase herede propiedades y comportamientos de otra clase. Esto promueve la reutilización del código al permitir que las clases compartan funcionalidades comunes. La subclase puede agregar nuevos atributos o métodos, o sobrescribir los existentes para adaptarse a sus necesidades específicas. La herencia también facilita la organización jerárquica de las clases, lo que mejora la estructura y comprensión del código.
- Polimorfismo: Capacidad de un objeto de tomar múltiples formas o comportarse de diferentes maneras en función del contexto en el que se utiliza. Esto se logra a través de la herencia y el uso de métodos polimórficos. Un método polimórfico es aquel que puede comportarse de manera diferente en función del tipo de objeto al que se llama. Esto mejora la flexibilidad del código y permite que las clases interactúen de manera más genérica, lo que facilita la escritura de código más mantenible y escalable.

30. ¿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

La utilidad javaw es similar a java, pero no abre una ventana de consola para la entrada/salida estándar. Se utiliza típicamente para aplicaciones GUI (Interfaz Gráfica de Usuario) en las que no se necesita una interfaz de línea de comandos.

31. ¿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.

En una API REST, un endpoint es una dirección URL que representa un recurso específico en el servidor. Los endpoints permiten a los clientes interactuar con el servidor mediante el envío de solicitudes HTTP (GET, POST, PUT, DELETE, etc.) a estas URLs para realizar diversas operaciones sobre los recursos representados.

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

```java
public class pregunta87 {
    public static void main(String[] args) {
        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

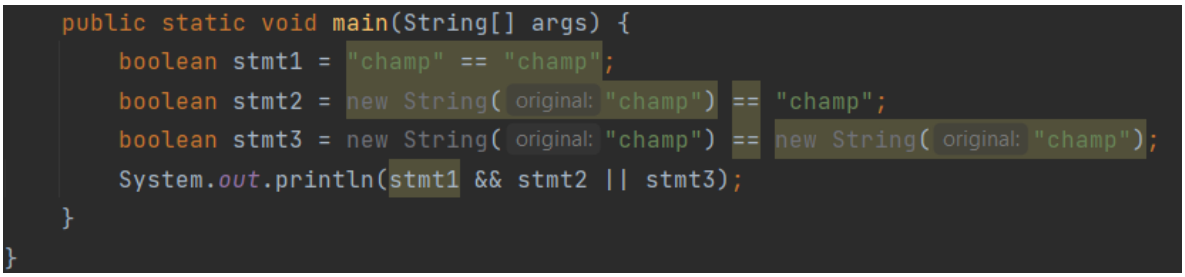
33. ¿Dado el siguiente enum y clase cuál es la opción que puede ir en el espacio en blanco para que el código compile?

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

public class pregunta88 {
    public int getAvergeTemperature(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:

34. ¿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

35. ¿Cómo se manejan las excepciones en Java?

a) Las excepciones el manejan con bloques switch case en Java. La excepción trae with Resources es una forma de lanzar una excepción en un método.
b) Las excepciones se manejan con bloques while en Java. La excepción trae with Resources es una forma de manejar excepciones de compilación
c) las excepciones se manejan con bloques if else en Java. La excepción trae 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 trae with Resources es una forma de cerrar automáticamente los recursos abiertos en un bloque try.

En Java, las excepciones se manejan mediante el uso de bloques try, catch y finally. El bloque try contiene el código que podría lanzar una excepción, el bloque catch contiene el código para manejar esa excepción y el bloque finally contiene el código que se ejecutará independientemente de si se lanzó o no una excepción (generalmente utilizado para liberar recursos). Además, el manejo de recursos con try-with-resources es una forma de asegurar que los recursos (como archivos o conexiones de base de datos) se cierren automáticamente después de que se haya terminado de usar, sin importar si se lanzó una excepción o no. Esto se hace declarando los recursos en el paréntesis del bloque try.

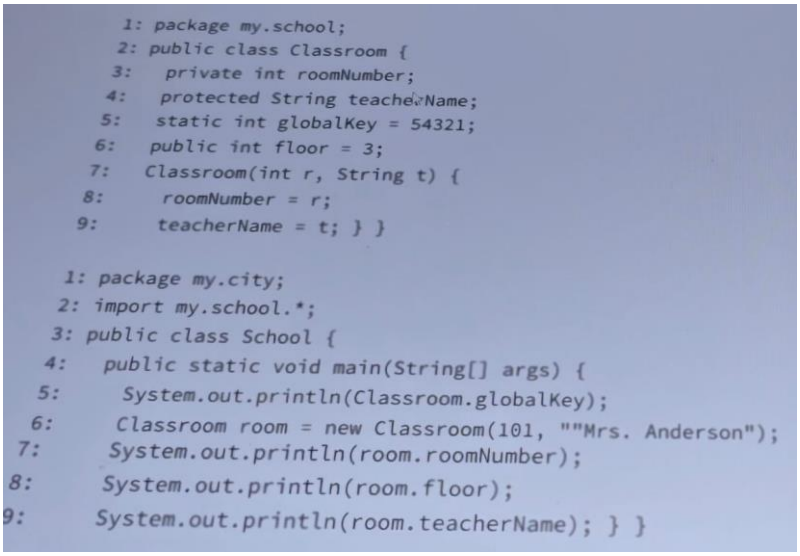
36. ¿Qué clase del paquete java.io permite leer y escribir archivos en ubicaciones específicas dentro de un archivo?

a) File
b) filename filter
c) file descriptor
d) RandomAccessFile

La clase RandomAccessFile del paquete java.io permite leer y escribir datos en ubicaciones específicas dentro de un archivo. Esta clase proporciona métodos para moverse a cualquier posición dentro del archivo

y realizar operaciones de lectura y escritura desde esa posición, lo que la hace ideal para tareas que requieren acceso aleatorio a los datos en un archivo.

37. 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

```java
package my.city;
import my.school.*;
public class School {
    public static void main(String[] args) {
        System.out.println(Classroom.globalKey);
        Classroom room = new Classroom( r: 101, t: "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.

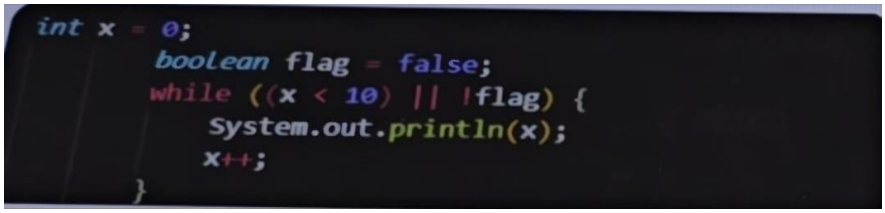
38. ¿Qué es una expresión lambda en Java?

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
d) Un método que no tiene cuerpo

En Java, una expresión lambda es una forma concisa de escribir implementaciones de interfaces funcionales (interfaces con un solo método abstracto). Las expresiones lambda permiten tratar la funcionalidad como una

declaración de método, pero sin necesidad de nombrar un método explícitamente. Se utilizan principalmente para pasar bloques de código como argumentos a métodos que requieren interfaces funcionales, como los métodos de las colecciones en la API de Streams de Java.

39. ¿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

40. ¿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.

Las anotaciones en Java son una forma de agregar metadatos al código. Pueden aplicarse a clases, métodos, campos, parámetros, etc., y son utilizadas por el compilador y las herramientas de tiempo de ejecución para procesar el código de manera especial. Las anotaciones pueden influir en el comportamiento del programa en tiempo de ejecución mediante el uso de reflexión y también pueden utilizarse para generar código, configurar dependencias, validar datos, entre otras cosas.

41. ¿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.

Las expresiones regulares en Java son una poderosa herramienta para buscar, validar y manipular patrones en cadenas de caracteres. Utilizan una sintaxis especial para definir estos patrones y pueden ser utilizadas para tareas como la validación de formatos de entrada (por ejemplo, direcciones de correo electrónico, números de teléfono), la búsqueda y sustitución de texto, y el análisis de cadenas. Java proporciona soporte para expresiones regulares a través de la clase Pattern y la clase Matcher en el paquete java.util.regex.

42. ¿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.

En Spring, las anotaciones son utilizadas para marcar clases, métodos, campos, y parámetros para proporcionar metadatos adicionales que el contenedor de Spring puede usar para configurar y gestionar los componentes de la aplicación. Estas anotaciones facilitan la configuración basada en anotaciones y reducen la necesidad de archivos de configuración XML. Algunas anotaciones comunes en Spring incluyen @Component, @Service, @Repository, @Controller, @Autowired, @Configuration, entre otras.

43. ¿Cuál es la salida?

```java
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

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

```java
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

45. ¿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

No es posible declarar un método default en una interfaz como final. Los métodos default en interfaces están diseñados para ser sobrescritos por las clases que los implementan si es necesario. Por lo tanto, ninguna de las opciones proporcionadas puede evitar que un método default sea sobrescrito.

46. ¿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) IllegalArgumntException
d) ClassCastException

En Java, una ArithmeticException se produce cuando se intenta realizar una operación aritmética que es incompatible con el tipo de datos, como por ejemplo, dividir un número por cero o realizar una operación con tipos de datos incompatibles, como la división de un entero por una cadena.

47. ¿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.

En el contexto de Spring Boot, un "starter" es un conjunto de dependencias preconfiguradas que simplifican la creación de aplicaciones Spring Boot. Los starters incluyen todas las dependencias necesarias para un objetivo específico, como el desarrollo web, el acceso a bases de datos, la seguridad, etc. Al incluir un starter en un proyecto de Spring Boot, se carga automáticamente todas las dependencias necesarias y se configuran de manera predeterminada, lo que facilita el inicio rápido del desarrollo de aplicaciones.

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

```java
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 [OPTIONS] PATH | URL | -

Donde PATH es la ruta al contexto del build, que normalmente contiene el Dockerfile y cualquier otro archivo necesario para la construcción de la imagen. Las opciones pueden incluir etiquetas (-t) para asignar un nombre y una etiqueta a la imagen construida, entre otras opciones para personalizar el proceso de construcción.

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


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

a) [[1], [2], [3], [4], [5], [7], [8]]
b) [[1, 2]], [[3, 4]], [[5, 6]], [[7, 8]]
c) [[1, 2], [3,4], [5,6], [7,8]]