

Exámenes Completos

1. ¿Cuál es el comando utilizado para deshacer el último commit en git?

- a) git reset
- b) git revert
- c) git amend
- d) git checkout

git revert : Esta orden deshace un commit anterior creando un nuevo commit que revierte los cambios introducidos por el commit anterior.

git reset : Esta orden se utiliza para deshacer cambios en el repositorio. Puede ser usada para eliminar cambios del área de preparación (git reset --soft), eliminar cambios del área de preparación y del directorio de trabajo (git reset --mixed), o eliminar cambios completamente (git reset --hard).

git amend :Esta orden se utiliza para realizar cambios en el commit más reciente.

git checkout: Esta orden se utiliza para cambiar entre ramas o para deshacer cambios en el directorio de trabajo. Puede ser utilizada para crear una nueva rama, cambiar a una rama existente, o para descartar cambios en archivos específicos en el directorio de trabajo.

2. ¿Cuál es la diferencia entre una clase abstracta y una interfaz en java 8?

- a) Una clase abstracta puede contener implementaciones de métodos, mientras que una interfaz no puede.
- b) Una clase abstracta puede contener variables de instancia, mientras que una interfaz no puede.
- c) Una interfaz puede contener implementaciones de métodos, mientras que una clase abstracta no puede.
- d) Una interfaz solo puede heredar de una clase, mientras que una clase abstracta puede heredar... (no se ve)
- e) Una interfaz puede ser implementada por múltiples clases mientras que una clase abstracta solo puede ser subclassificada.

Diferencias entre una Clase Abstracta y una Interfaz

Implementación de métodos:

Clase abstracta: Puede contener métodos abstractos (sin implementación) y métodos concretos (con implementación). Las clases que heredan de una clase abstracta deben implementar todos los métodos abstractos o ser ellas mismas clases abstractas.

Interface: Solo puede contener métodos abstractos y no puede tener ningún método con implementación. Las clases que implementan una interfaz deben proporcionar implementaciones para todos los métodos definidos en la interfaz.

Herencia:

Clase abstracta: Las clases abstractas pueden extender otras clases (abstractas o concretas) y pueden implementar interfaces.

Interface: Las interfaces solo pueden extender otras interfaces, pero no pueden extender clases. Una clase puede implementar múltiples interfaces.

Constructores:

Clase abstracta: Puede tener un constructor, que puede ser llamado desde las clases hijas mediante la palabra clave super.

Interface: No puede tener un constructor. Las interfaces no pueden ser instanciadas, por lo que no tienen constructores.

Variables de instancia:

Clase abstracta: Puede tener variables de instancia (campos), métodos estáticos y métodos de instancia.

Interface: No puede tener variables de instancia, métodos estáticos o métodos de instancia. Solo puede contener constantes (variables públicas estáticas y finales) y métodos abstractos.

Uso y flexibilidad:

Clase abstracta: Es más útil cuando se desea proporcionar una implementación base común para varias clases relacionadas. Las clases abstractas pueden contener estado y comportamiento común que las clases hijas pueden heredar y modificar según sea necesario.

Interface: Es más adecuada para definir contratos de comportamiento. Las interfaces son útiles cuando se quiere garantizar que las clases que las implementan tengan un conjunto específico de métodos, sin preocuparse por la implementación real.

3. De los siguientes ¿qué tipos de declaraciones se deben usar para contar la cantidad de monedas de 5 centavos en una matriz de cadenas de varias monedas? (Elija todas las correctas)

- a) Conditional (Duda Preguntar)
- b) Assertion
- c) Assignment
- d) Iteration

Conditional :

Assinment :

Iteration:

Assertion :

4. ¿Qué es un archivo JAR en java?

- a) Un archivo que contiene un archivo de configuración Maven
- b) Un archivo que contiene un archivo de configuración Git.
- c) Un archivo que contiene una clase Java compilada
- d) Un archivo que contiene un archivo de configuración de Spring

Archivo Jar:

Un archivo JAR (Java ARchive) es un archivo comprimido que se utiliza para distribuir y empaquetar uno o más archivos Java, como clases, recursos y metadatos relacionados. Estos archivos se utilizan principalmente para distribuir bibliotecas, aplicaciones o componentes escritos en Java.

Un archivo JAR puede contener:

Clases Java: Los archivos .class que contienen bytecode Java, que son ejecutados por la máquina virtual de Java (JVM).

Recursos: Archivos como imágenes, archivos de configuración, archivos de propiedades, etc., que son utilizados por la aplicación Java.

Metadatos: Archivos MANIFEST.MF que contienen información sobre el contenido del archivo JAR, como la versión de la aplicación, el autor, las dependencias, etc.

Además, los archivos JAR pueden tener una estructura interna que incluye directorios para organizar los archivos dentro del archivo JAR.

5. ¿Qué es la sobrecarga de métodos en Java?

- a) Cuando un método tiene múltiples definiciones con el mismo nombre y tipo de parámetros.
- b) Cuando un método tiene múltiples definiciones con diferentes nombres y cantidades de parámetros.
- c) Cuando un método tiene múltiples definiciones con diferentes tipos de cantidades de parámetros.
- d) Cuando un método tiene múltiples definiciones con diferentes nombres y tipos de parámetros.

Sobrecarga de Métodos en Java: La sobrecarga de métodos en Java es un concepto que permite definir múltiples métodos en una misma clase con el mismo nombre, pero con diferentes listas de parámetros. Esto significa que puedes tener varios métodos con el mismo nombre en una clase, siempre y cuando la lista de tipos o cantidad de parámetros sea diferente entre ellos.

La sobrecarga de métodos es una forma de polimorfismo en Java, donde se puede llamar al mismo método utilizando diferentes argumentos. Cuando se invoca un método sobrecargado, el compilador de Java determina cuál de los métodos sobrecargados debe ser llamado basándose en la lista de argumentos proporcionados.

6. ¿Cuál es la diferencia entre un ArrayList y un LinkedList en Java?

- a) ArrayList es más rápido que un LinkedList para agregar y eliminar elementos.
- b) ArrayList es más eficiente en el uso de memoria que LinkedList.
- c) LinkedList es una clase abstracta mientras que ArrayList es una clase concreta.
- d) **LinkedList es más rápido que ArrayList para acceder a elementos aleatorios.**

La diferencia principal entre un ArrayList y un LinkedList en Java radica en la forma en que se almacenan y acceden a los elementos dentro de la estructura de datos.

ArrayList:

Un ArrayList en Java implementa la interfaz List y utiliza un arreglo dinámico para almacenar los elementos. Esto significa que los elementos se almacenan en un arreglo de tamaño variable que se redimensiona automáticamente según sea necesario.

La ventaja principal de un ArrayList es que proporciona un acceso rápido a los elementos mediante índices, lo que significa que puedes acceder directamente a cualquier elemento de la lista utilizando su posición en el arreglo subyacente.

Sin embargo, la inserción y eliminación de elementos en un ArrayList pueden ser costosas si se realizan en posiciones intermedias de la lista, ya que puede requerir el desplazamiento de otros elementos en el arreglo.

LinkedList:

Un LinkedList en Java implementa la interfaz List pero utiliza una estructura de lista doblemente enlazada para almacenar los elementos. Cada elemento en la lista contiene una referencia al elemento anterior y al siguiente en la secuencia.

La principal ventaja de un LinkedList es que permite una inserción y eliminación eficientes de elementos en cualquier parte de la lista. Esto se debe a que solo se necesitan ajustes en los enlaces de los nodos, sin necesidad de desplazar grandes cantidades de datos como en el caso de un ArrayList.

Sin embargo, el acceso a elementos en un LinkedList puede ser más lento que en un ArrayList, especialmente cuando se accede a elementos de forma secuencial, ya que puede requerir recorrer la lista desde el principio hasta la posición deseada.

En resumen, un ArrayList es más adecuado cuando se requiere un acceso rápido a los elementos mediante índices y cuando la inserción y eliminación se realizan principalmente al final de la lista. Por otro lado, un LinkedList es más adecuado cuando se requieren inserciones y eliminaciones

frecuentes en posiciones intermedias de la lista y cuando el acceso a los elementos no necesita ser tan rápido. La elección entre ambos depende de las necesidades específicas de la aplicación y del tipo de operaciones que se realizarán con la lista.

7. ¿Cuándo se debe usar un bloque finally en una declaración try regular (no una prueba con recursos)?

- a) Cuando no hay bloques catch en una declaración try.
- b) Nunca.
- c) Cuando hay dos o más bloques catch en una sentencia try.
- d) Cuando hay exactamente un bloque catch en una sentencia try.
- e) Cuando el código del programa no termina por sí solo.

El bloque finally en una declaración try regular se debe usar cuando es necesario ejecutar un código determinado sin importar si se produce o no una excepción en el bloque try. El bloque finally es opcional y proporciona una forma de garantizar que se ejecuten ciertas operaciones de limpieza, cierre de recursos, o cualquier otra tarea que deba realizarse independientemente del resultado de la operación en el bloque try.

Cuando hay un bloque finally pero no hay bloques catch, significa que estás utilizando un try sin capturar explícitamente ninguna excepción. En este caso, el bloque finally se ejecutará después de que el bloque try termine de ejecutarse, ya sea con éxito o generando una excepción.

8. ¿Cuál es el propósito principal de los test unitarios?

- a) Comprobar la eficiencia del hardware.
- b) Medir la velocidad de la aplicación.
- c) Ahorrar tiempo en el desarrollo.
- d) Asegurar la calidad del software.

El propósito principal de los test unitarios es verificar de manera aislada el correcto funcionamiento de las unidades individuales de código, típicamente funciones o métodos. Los test unitarios son una práctica fundamental en el desarrollo de software, ya que ayudan a asegurar que cada componente pequeño de una aplicación funciona según lo esperado.

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

```
public class Test3 {  
    public static void main(String[] args) {  
        String cad1 = "hola";  
        String cad2 = new String(original: "hola");  
        String cad3 = "hola";  
        if (cad1 cad2)
```

```

System.out.println("ca1 es igual a cad2");
else System.out.println("cad1 diferente a cad2");
if (cad1 cad3)
System.out.println("cad1 es igual a cad3");
else
System.out.println("cad1 diferente a cad3");
}
}

```

```

public class Test3 {
    public static void main(String[] args) {
        String cad1 = "hola";
        String cad2 = new String( original: "hola");
        String cad3 = "hola";

        if (cad1 == cad2)
            System.out.println("ca1 es igual a cad2");
        else System.out.println("cad1 diferente a cad2");

        if (cad1 == cad3)
            System.out.println("cad1 es igual a cad3");
        else
            System.out.println("cad1 diferente a cad3");
    }
}

```

a) cad1 diferente a cad2

cad1 es igual a cad3

b) ca1 es igual a cad2

ca1 es igual a cad3

c) No compila

d) cad1 diferente a cad2

cad1 diferente a cad3

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

```
1: class Mammal (  
2: public Mammal(int age) (  
3: System.out.print("Mammal");  
4: }  
5:}  
6: public class Platypus extends Mammal (  
7: public Platypus() {  
8: System.out.print("Platypus");  
9:}  
10: public static void main(String[] args) {  
11: new Mammal (5);  
12:}  
13:}
```

```
1: class Mammal {  
2:     public Mammal(int age) {  
3:         System.out.print("Mammal");  
4:     }  
5: }  
6: public class Platypus extends Mammal {  
7:     public Platypus() {  
8:         System.out.print("Platypus");  
9:     }  
10:     public static void main(String[] args) {  
11:         new Mammal(5);  
12:     }  
13: }
```

- a) Mammal.
- b) MammalPlatypus.
- c) El código no se compila en la línea 11.
- d) El código no compila en la línea 8

11. ¿Cómo se manejan las excepciones en java?

- a) Con la instrucción try-catch.
- b) Con la instrucción if-else.
- c) Con la instrucción for.
- d) Con la instrucción while.

El bloque try-catch en Java se utiliza para manejar excepciones que pueden ocurrir durante la ejecución de un programa.

12. ¿La anotación @Ignore es usada para omitir un test por lo que no se ejecuta?

- a) Verdadero
- b) Falso

En Java, la anotación @Ignore se utiliza principalmente en el contexto de pruebas unitarias con bibliotecas como JUnit o TestNG. Cuando se usa con JUnit, por ejemplo, @Ignore se coloca encima de un método de prueba para indicar a JUnit que no debe ejecutar ese método durante la ejecución de la suite de pruebas.

13. ¿Cuál es el resultado de compilar y ejecutar el siguiente código?

```
public class Tester {  
    static {  
        int x = 3;  
    }  
    static int x;  
    public static void main(String[] args) {  
        x--; // line 7  
        System.out.println(x);  
    }  
}
```



```

public class Tester {
    static {
        int x = 3;
    }
    2 usages
    static int x;
    public static void main(String[] args) {
        x--; // line 7
        System.out.println(x);
    }
}

```

- a) Error de compilación en la línea 7, x no se inicializa.
- b) -1**
- c) -2
- d) 0

Al compilar y ejecutar este código Java, la salida será 3.

Aquí está una explicación detallada de por qué:

La clase Tester tiene un campo estático x inicializado con el valor 3 dentro de un bloque de inicialización estática.

También hay otra declaración estática static int x;. Sin embargo, esta declaración no inicializa la variable x con un valor. En su lugar, se refiere a la misma variable x inicializada anteriormente en el bloque estático.

En el método main, la línea x--; decrementa el valor de x en 1. Como x inicialmente tiene el valor 3, después de esta operación, x tendrá el valor 2.

Finalmente, la línea System.out.println(x); imprime el valor actual de x, que es 2.

Sin embargo, en la imagen proporcionada, no se muestra la línea x--;. Por lo tanto, el valor de x permanece como 3, y la salida al ejecutar este código será simplemente 3.

14. ¿Qué es un operador de short circuit?

- a) Sirve para realizar más eficientes las operaciones condicionales evitando ejecutar operaciones si estas ya no son necesarias.**
- b) Operador que nos sirve para crear una nueva clase anónima.

- c) Sirve para lanzar una excepción personalizada.
- d) Es un patrón de arquitectura de microservicios que nos permite evitar el consumo de servicios que están en mantenimiento .

Un operador de short-circuit (circuito corto) en programación se refiere a ciertos operadores lógicos que detienen la evaluación de una expresión tan pronto como el resultado está decidido.

15. ¿Qué es el patrón de diseño DAO y cómo se implementa en Java?

- a) El patrón de diseño DAO es un patrón que se utiliza para abstraer la capa de acceso a datos en una aplicación. Se puede implementar en Java utilizando interfaces y clases concretas.
- b) El patrón de diseño DAO es un patrón que se utiliza para abstraer la capa de negocios de una aplicación. Se puede implementar en Java utilizando clases abstractas y métodos estáticos.
- c) El patrón de diseño DAO es un patrón que se utiliza para abstraer la capa de presentación en una aplicación. Se puede implementar en Java utilizando interfaces y clases concretas.
- d) El patrón de diseño DAO es un patrón que se utiliza para abstraer la capa de infraestructura en una aplicación. Se puede implementar en Java utilizando excepciones y bloques try-catch.

El patrón de diseño DAO (Data Access Object) es un patrón estructural que proporciona una abstracción para las operaciones de acceso a la base de datos. Su principal objetivo es separar la lógica de acceso a datos de la lógica de negocio, lo que facilita el mantenimiento, la escalabilidad y la reutilización del código. Este patrón permite cambiar la fuente de datos (por ejemplo, de una base de datos SQL a una base de datos NoSQL) sin modificar la lógica de negocio.

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

- a) Un endpoint es la URL que se utiliza para acceder a 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 un objeto que se utiliza para almacenar datos en una API REST.

Un endpoint en una API REST es una URL (Uniform Resource Locator) específica que permite a los clientes interactuar con un recurso particular de la API. Los endpoints son las rutas a través de las cuales se exponen los servicios y funcionalidades de la API, y cada endpoint corresponde a una dirección donde se puede realizar una operación particular utilizando métodos HTTP.

17. ¿Qué hace el siguiente programa?

```

public class Palabra {
    public static void main(String[] args) {
        String sPalabra = "palabra";
        int inc = 0;
        int des = sPalabra.length() - 1;
        boolean bError = false;
        while ((inc < des) && (!bError)){
            if (sPalabra.charAt(inc) == sPalabra.charAt(des)){
                inc++;
                des--;
            } else {
                bError = true;
            }
        }
    }
}

```

- a) El programa no compila.
- b) Cuenta las letras que hay en una palabra.
- c) Verifica si una palabra es un palíndromo.
- d) No se ve

El propósito de este código es verificar si la cadena sPalabra es un palíndromo. Un palíndromo es una palabra que se lee igual de adelante hacia atrás y de atrás hacia adelante.

18. ¿Cuál de las siguientes opciones son verdaderas? (elija todas las correctas)

- a) Java es un lenguaje orientado a objetos.
- b) El código Java compilado en Windows puede ejecutarse en Linux.
- c) Java permite la sobrecarga de operadores
- d) Java es un lenguaje de programación funcional. ← No sabemos
- e) Java es un lenguaje procedimental. ← No sabemos
- f) Java tiene punteros a ubicaciones específicas en la memoria.

a) Java es un lenguaje orientado a objetos.

Verdadero. Java es conocido por ser un lenguaje de programación orientado a objetos (OOP), donde el enfoque principal es la creación y manipulación de objetos. Java utiliza conceptos como clases, objetos, herencia, polimorfismo, encapsulación y abstracción.

b) El código Java compilado en Windows puede ejecutarse en Linux.

Verdadero. Una de las características clave de Java es su portabilidad. El código Java se compila en bytecode, que es independiente de la plataforma. Este bytecode puede ejecutarse en cualquier sistema operativo que tenga una Java Virtual Machine (JVM) compatible, como Windows, Linux, macOS, etc.

c) Java permite la sobrecarga de operadores.

Falso. A diferencia de algunos otros lenguajes como C++, Java no permite la sobrecarga de operadores. Los operadores en Java tienen un comportamiento predefinido que no se puede cambiar.

d) Java es un lenguaje de programación funcional.

Falso. Aunque Java ha incorporado algunas características funcionales a partir de Java 8, como las expresiones lambda y la API de Streams, no es considerado un lenguaje de programación funcional puro. Java sigue siendo principalmente un lenguaje orientado a objetos.

e) Java es un lenguaje procedimental.

Falso. Java no es un lenguaje procedimental puro. Aunque se pueden escribir programas de estilo procedimental en Java, el enfoque principal es orientado a objetos. Los lenguajes procedimentales se centran en funciones o procedimientos y no en objetos.

f) Java tiene punteros a ubicaciones específicas en la memoria.

Falso. Java no utiliza punteros como C o C++. En su lugar, Java utiliza referencias para acceder a objetos. Esto es una medida de seguridad para evitar errores comunes y vulnerabilidades relacionadas con el manejo directo de la memoria.

19. ¿Qué es Maven y para qué se utiliza en el desarrollo de aplicaciones?

- a) Maven es un lenguaje de programación. Se utiliza en el desarrollo de aplicaciones Java para escribir... (no se ve)
- b) Maven es un servidor de base de datos. Se utiliza en el desarrollo de aplicaciones java para alojar... (no se ve)
- c) Maven es un sistema de control de versiones. Se utiliza en el desarrollo de aplicaciones java para (no se ve)
- d) Maven es una herramienta de gestión de dependencias. Se utiliza en el desarrollo de aplicaciones... (no se ve) en el proyecto.

Maven es una herramienta de gestión y comprensión de proyectos en Java que se utiliza principalmente para la gestión de dependencias, la construcción y la documentación de proyectos. Es un proyecto de la Apache Software Foundation y se basa en el concepto de un archivo de configuración central, llamado pom.xml (Project Object Model), que define las dependencias, las configuraciones de construcción y otra información sobre el proyecto.

20. ¿Cuál de lo siguiente es cierto? (elija todas las correctas)

- a) javac compila un archivo .java en un archivo .bytecode. ← Si no tuviera el punto sería correcta

- b) Java toma el nombre del archivo .bytecode como parámetro.
- c) **javac compila un archivo .java en un archivo .class**
- d) **Java toma el nombre de la clase como parámetro.**
- e) Java toma el nombre del archivo .class como parámetro.
- f) javac compila un archivo .class como archivo java.

21. ¿Qué es Git y cuáles son algunos de sus comandos básicos?

- a) Git es un lenguaje de programación. Algunos comandos básicos de Git incluyen "print" e "if-else".
- b) Git es una herramienta para el análisis de código. Algunos comandos básicos de Git incluyen "analyze"... (no se ve).
- c) **Git es un sistema de control de versiones. Algunos comandos básicos de Git incluyen "commit" y "push".**
- d) Git es una herramienta para realizar pruebas de software. Algunos comandos básicos de Git incluyen... no se ve, pero no es correcta.

Git básicamente es un sistema de control de versiones y dos de sus comandos más usados son "commit" y "push".

22. Dados los siguientes segmentos de código, ¿Qué respuesta no es una implementación de java válida?

- a) **int variableA = 10;**
float variableB = 10.5f;
int variableC = variableA + variableB;
- b) byte variableA = 10;
double variableB = 10.5f;
double variableC = variableA + variableB;
- c) byte variableA = 10;
float variableB = 10.5f;
float variableC = variableA + variableB;

Problema: En esta opción, variableA es un int y variableB es un float. Cuando sumas un int y un float, el resultado es un float. Intentar asignar un float a un int sin conversión explícita no es válido en Java.

23. ¿Qué escenario es el mejor uso de una excepción?

- a) La computadora se incendió.
- b) No sabe cómo codificar un método.

c) No se encuentra un elemento al buscar en una lista.

d) Se pasa un parámetro inesperado a un método.

e) Quiere recorrer una lista.

Las excepciones en Java son un mecanismo que permite manejar situaciones anómalas o errores que pueden ocurrir durante la ejecución de un programa. El propósito principal de las excepciones es proporcionar una forma estructurada y coherente de manejar los errores, separando la lógica de manejo de errores del código normal del programa. Esto hace que el código sea más legible, mantenible y robusto. Basándonos en este concepto la respuesta podría ser "C" o "D".

24. ¿Qué es un bean en Spring?

a) Un archivo de configuración XML que se utiliza para definir la estructura de una tabla de base de datos.

b) Una instancia de una clase que se administra por el contenedor de Spring.

c) Una herramienta de inyección de dependencias que se utiliza para inyectar dependencias en una clase.

d) Una clase que se utiliza para configurar la conexión a una base de datos.

En el contexto de Spring Framework, un bean es un objeto que se instancia, se monta y se gestiona por el contenedor de Spring. En otras palabras, es un componente fundamental que Spring maneja y controla dentro de su entorno de ejecución.

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

```
public class Test1 extends Concreate{
1 usage
Test1(){
System.out.println(" ");
}
public static void main(String[] args) {
// TODO Auto-generated method stub
new Test1();
}
}
1 usage 1 inheritor
class Concreate extends Send{
1 usage
Concreate(){
System.out.println("c");
}
private Concreate (String s) {
}
}
1 usage 2 inheritors
```

```

Abstract class Send{
2 usages
Send() {
System.out.println("s");
a
}
}
}

```

```

public class Test1 extends Concrete{
1 usage
Test1(){
System.out.println("t ");
}

public static void main(String[] args) {
// TODO: Auto-generated method stub
new Test1();
}
}

1 usage: 1 inheritance
class Concrete extends Send{
1 usage
Concrete(){
System.out.println("c ");
}
private Concrete(String s){
}
}

1 usage: 2 inheritance
abstract class Send{
2 usages
Send(){
System.out.println("s ");
}
}

```

- a) c,s,t
- b) t,s,c
- c) Error en tiempo de ejecución
- d) No compila

NINGUNA ES CORRECTA

26. ¿Cuáles de las siguientes afirmaciones sobre el polimorfismo son verdaderas? (Elija todas las correctas)

- a) Si un método toma una superclase de 3 objetos, cualquiera de esas clases puede pasarse como parámetro del método
- b) Un método que toma un parámetro con tipo java.lang.object tomará cualquier referencia
- c) Una referencia a un objeto se puede convertir a una subclase de objetos en una conversión explícita ← Cast
- d) Todas las excepciones de conversión se pueden detectar en tiempo de compilación
- e) Al definir un método de instancia pública en la súper clase, garantiza que el método específico se llamará al método en la clase principal en tiempo de ejecución

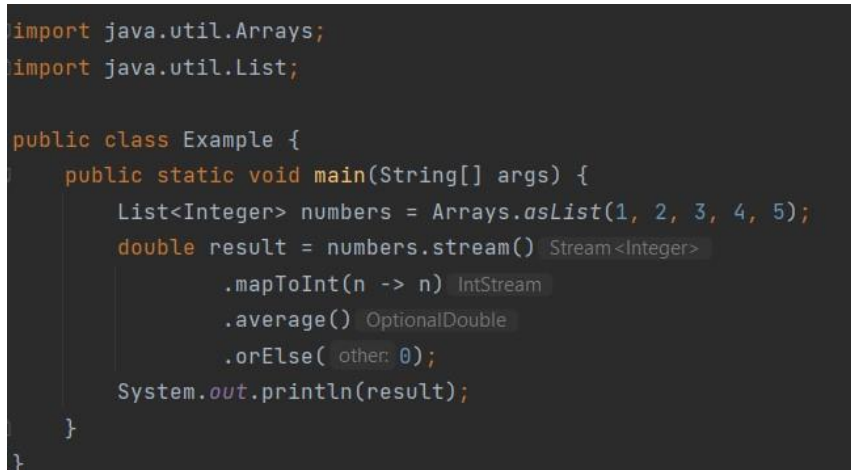
27. ¿Son patrones de diseño de software estructural?

- a) Adapter, Proxy, Prototype y Bridge.
- b) Adapter, Bridge, Proxy y Composite.
- c) Agile, Builder, Singleton y Prototype.
- d) Builder, Singleton y Prototype y Abstract Factory.

Adapter, Bridge, Proxy y Composite son patrones de diseño de software estructural. Estos patrones se centran en cómo componer objetos y clases para formar estructuras más grandes y complejas, facilitando la flexibilidad y reutilización del código.

28. 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);
        double result = numbers.stream()
            .mapToInt(n -> n)
            .average()
            .orElse (0);
        System.out.println(result);
    }
}
```



```
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);
        double result = numbers.stream() Stream<Integer>
            .mapToInt(n -> n) IntStream
            .average() OptionalDouble
            .orElse( other: 0);
        System.out.println(result);
    }
}
```

- a) 3.0
- b) 1
- c) 5
- d) 2.5

29. ¿Qué son las pruebas de integración?

- a) Pruebas que comprueben el rendimiento de la aplicación.
- b) Pruebas que comprueben el funcionamiento de la interfaz de usuario.
- c) Pruebas que comprueban el funcionamiento de varias unidades de código juntas.
- d) Pruebas que comprueban el funcionamiento de una sola unidad de código.

Las pruebas de integración son un tipo de pruebas de software que se centran en verificar la interacción entre diferentes módulos, componentes o servicios de una aplicación para asegurar que funcionan correctamente juntos. En el contexto de desarrollo en Java, estas pruebas son fundamentales para garantizar que los distintos componentes desarrollados individualmente se integren de manera cohesiva.

30. ¿Qué comando se utiliza para enviar los cambios confirmados en un repositorio local al repositorio remoto?

- a) **git push**
- b) it pull
- c) git commit
- d) git add

El comando `git push` se utiliza para cargar los commits locales (cambios confirmados) en un repositorio remoto. Este comando actualiza el repositorio remoto con el contenido de la rama local especificada.

31. Seleccione la respuesta correcta, dado el siguiente bloque de código.

```

class ClassX{
    7 usages
    static int y = 2;
    1 usage
    ClassX(int x){
        this();
        y = y * 2;
    }

    1 usage
    ClassX(){
        y++;
    }
}

1 usage
public class Class2 extends ClassX{
    1 usage
    Class2(){
        super(y);
        y = y + 3;
    }
    public static void main(String[] args) {
        new Class2();
        System.out.println(y);
    }
}

```

- a) Error de compilación
- b) 9
- c) 7
- d) No se ve, pero la correcta es 9

32. ¿Cuál es el comando utilizado para crear una nueva rama en Git?

- a) git branch
- b) git merge
- c) git commit
- d) git push

El comando git branch se utiliza para listar, crear o eliminar ramas en Git. Cuando se utiliza con el nombre de una nueva rama, crea esa nueva rama.

33. ¿Cuál es el resultado de compilar la siguiente clase?

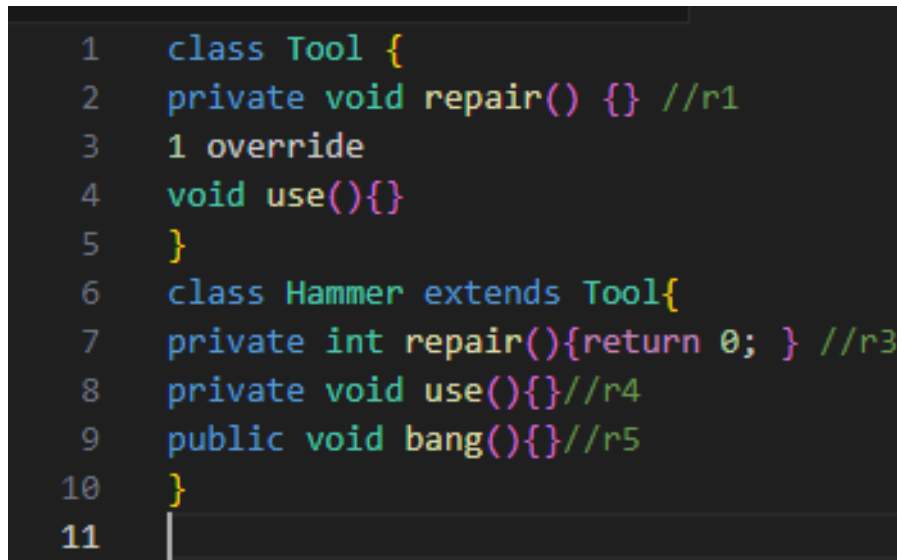
```
public class Book {  
    3 usages  
    private int ISBN;  
    private String title, author;  
    private int pageCount;  
  
    public int hashCode() {  
        return ISBN;  
    }  
    public boolean equals(Object obj){  
        if(!(obj instanceof Book)) {  
            return false;  
        }  
        Book other = (Book) obj;  
    }  
}
```

```
public class Book {  
    3 usages  
    private int ISBN;  
    private String title, author;  
    private int pageCount;  
  
    public int hashCode(){  
        return ISBN;  
    }  
  
    public boolean equals(Object obj){  
        if(!(obj instanceof Book)){  
            return false;  
        }  
        Book other = (Book) obj;  
        return this.ISBN == other.ISBN;  
    }  
}
```

- a) Línea 15 no compila porque other.ISBN es un atributo con modificador de acceso private.
- b) Línea 14 no compila porque no está declarada o manejada ClassCastException
- c) La clase compila satisfactoriamente.

34. ¿Cuál es la primer línea en fallar al compilar?

```
class Tool {  
    private void repair() {} //r1  
    void use(){}  
}  
class Hammer extends Tool{  
    private int repair(){return 0; } //r3  
    private void use(){}//r4  
    public void bang(){}//r5  
}
```



```
1  class Tool {  
2  private void repair() {} //r1  
3  1 override  
4  void use(){}  
5  }  
6  class Hammer extends Tool{  
7  private int repair(){return 0; } //r3  
8  private void use(){}//r4  
9  public void bang(){}//r5  
10 }  
11 |
```

- a) r5
- b) r4
- c) r3
- d) Ninguna de las anteriores.

35. ¿Qué es Git?

- a) Una herramienta de automatización de compilación que se utiliza para compilar y construir un proyecto (no se ve lo demás).
- b) Una herramienta de gestión de software de dependencias que se utiliza para descargar bibliotecas y paquetes en... no se ve
- c) Una herramienta de generación de informes que se utiliza para generar informes sobre el rendimiento... no se ve
- d) Una herramienta de control de versiones que se utiliza para almacenar y administrar el código... no se ve.

Git es un sistema de control de versiones distribuido ampliamente utilizado para el seguimiento de cambios en el código fuente durante el desarrollo de software. Fue creado por Linus Torvalds en 2005 con el propósito de gestionar el desarrollo del kernel de Linux, pero desde entonces se ha convertido en una herramienta esencial para el desarrollo de software en general.

36. ¿Cuál de las siguientes excepciones lanza la JVM? (Elija todas las correctas)

- a) `ArrayIndexOutOfBoundsException`
- b) `NumberFormatException`
- c) `ExceptionInInitializerError`
- d) `Java.io.IOException`
- e) `NullPointerException`

La excepción que lanza directamente la JVM (Java Virtual Machine) es la `ExceptionInInitializerError`. Esta excepción se lanza cuando ocurre un error durante la inicialización estática de una clase o interfaz. Puede ocurrir, por ejemplo, si se produce una excepción no controlada durante la ejecución de un bloque estático de inicialización.

- `ArrayIndexOutOfBoundsException`: Esta excepción se lanza cuando se intenta acceder a un índice fuera del rango válido de un array.
- `NumberFormatException`: Esta excepción se lanza cuando se intenta convertir una cadena en un tipo numérico, pero el formato de la cadena no es válido para ese tipo numérico.
- `Java.io.IOException`: Esta excepción se utiliza para manejar errores relacionados con operaciones de entrada y salida (I/O) en Java, como lectura o escritura de archivos.
- `NullPointerException`: Esta excepción se lanza cuando se intenta acceder a un objeto o método a través de una referencia nula.

37. ¿Cuál es la diferencia entre una clase abstracta y una interfaz en Java 8?

- a) Una interfaz solo puede heredar de una clase, mientras una clase abstracta puede heredar de múltiples interfaces
- b) Una clase abstracta puede contener implementaciones de métodos, mientras que una interfaz no puede.

- c) Una clase abstracta puede contener variables de instancia, mientras que una interfaz no puede.
- d) Una interfaz puede contener implementaciones de métodos, mientras que una clase abstracta no puede.

mientras que las clases abstractas pueden contener tanto métodos abstractos como métodos con implementación, las interfaces en Java 8 solo pueden contener métodos abstractos y métodos con implementación por defecto. Además, las clases pueden extender solo una clase abstracta pero pueden implementar múltiples interfaces. La elección entre una clase abstracta y una interfaz depende de los requisitos de diseño y del comportamiento deseado en la aplicación.

38. ¿Cuál es el comando utilizado para fusionar una rama en Git?

- a) git Branch
- b) git merge
- c) git push
- d) git pull

En Git, el comando utilizado para fusionar una rama en otra rama es git merge. Este comando permite integrar los cambios de una rama (la rama de origen) en la rama actual (la rama de destino).

39. ¿Qué es REST y cuál es su relación con las API web?

- a) REST es un protocolo de comunicación. Su relación con las API web es que las utiliza para definir los endpoints de una API
- b) REST es un lenguaje de programación. Su relación con las API web es que se utiliza para crear aplicaciones web.
- c) REST es un servicio en la nube. Su relación con las API web es que se utiliza para alojar las aplicaciones web.

REST es una arquitectura para aplicaciones web. Su relación con las API web es que se utiliza para definir la estructura y funcionalidades de una API.

En el contexto de las API web, REST se refiere a un conjunto de principios y restricciones que definen cómo deben ser diseñadas y expuestas las interfaces de programación de aplicaciones (API) web. Estas API implementan los principios de REST para permitir la comunicación y la interoperabilidad entre sistemas distribuidos a través del protocolo HTTP.

40. ¿Cuál es el comando utilizado para actualizar la rama local con los cambios de la rama remota en Git?

- a) git checkout
- b) git clone

- c) git push
- d) git pull

El comando utilizado para actualizar la rama local con los cambios de la rama remota en Git es git pull.

git clone: Este comando se utiliza para clonar un repositorio remoto a tu máquina local. Este comando se utiliza principalmente para obtener una copia completa del repositorio remoto por primera vez.

git push: Este comando se utiliza para enviar los cambios de tu rama local al repositorio remoto.

git checkout: Este comando se utiliza para cambiar de una rama a otra, o para restaurar archivos de un commit específico.

41. ¿Qué es un microservicio?

- a) Son componentes que se pueden desplegar de forma independiente, de función múltiple, es decir... no necesariamente están relacionados.
- b) Es un componente que se pueden desplegar de forma independiente y suelen ser de función única, es decir, que están estrechamente relacionados.
- c) Es el conjunto de endpoints contenidos en múltiples desarrollos que se despliegan en conjunto y que están estrechamente relacionados.
- d) Ninguna de las anteriores

Un microservicio es un enfoque arquitectónico para el desarrollo de software donde una aplicación se compone de pequeños servicios independientes, cada uno enfocado en realizar una tarea específica o cumplir una funcionalidad concreta. Cada microservicio es una unidad de desarrollo, despliegue y mantenimiento por sí misma.

42. Dado el siguiente código:

```
public class Main {  
    public static void main(String[] args) {  
        int[] numeros = {1,2,3,4,5};  
        int suma = 0;  
        for (int i = 1; i <= numeros.length; i++){  
            suma += numeros[i];  
        }  
        System.out.println("La suma de los números es: www + suma);  
    }  
}
```

```

public class Main {
    public static void main(String[] args) {
        int[] numeros = {1,2,3,4,5};
        int suma = 0;
        for (int i = 1; i <= numeros.length; i++){
            suma += numeros[i];
        }

        System.out.println("La suma de los números es: " + suma);
    }
}

```

¿Este código compila sin errores?

- a) Si, compila sin errores.
- b) No, hay un error en el ciclo for.
- c) No, hay un error en la inicialización de la variable "suma".
- d) No, hay un error en la declaración del arreglo.

43. ¿Qué método se utiliza para obtener el mensaje de una excepción en Java?

- a) getClass()
- b) printStackTrace()
- c) toString()
- d) getMessage()

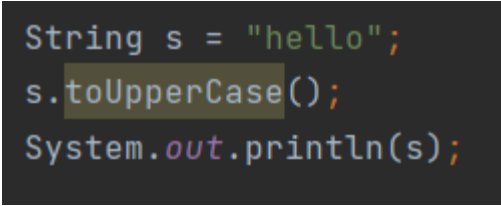
44. ¿Cuál de las siguientes afirmaciones son verdaderas? (Elije todas las correctas)

- a) Puede declarar son lo excepciones no comprobadas (unchecked).
- b) Las excepciones en tiempo de ejecución son lo mismo que las excepciones no comprobadas.
- c) Las excepciones en tiempo de ejecución son lo mismo que las excepciones comprobadas.
- d) Solo puede declarar excepciones comprobadas (checked)
- e) Solo puede manejar subclases de Exception.

En esta te pide que selecciones varias, pero no se puede, la e) también sería correcta.

45. ¿Cuál es el resultado de ejecutar el siguiente código?

```
String s = "hello";  
s.toUpperCase();  
System.out.println(s);
```



```
String s = "hello";  
s.toUpperCase();  
System.out.println(s);
```

- a) NullPointerException
- b) hello
- c) HELLO
- d) Hello

La razón es que en Java, los strings son inmutables, lo que significa que no se pueden modificar después de ser creados. Cuando llamas al método `toUpperCase()` en la variable `s`, este método devuelve un nuevo string que representa `s` en mayúsculas, pero no modifica el valor original de `s`. Por lo tanto, cuando imprimes `s` después de llamar a `toUpperCase()`, sigue manteniendo el mismo valor original en minúsculas, que es "hello".

46. ¿Qué son las pruebas de integración?

- a) Pruebas que comprueben el funcionamiento de la interfaz de usuario.
- b) Pruebas que comprueban el funcionamiento de varias unidades juntas.
- c) Pruebas que comprueben el rendimiento de la aplicación.
- d) Pruebas que comprueben el funcionamiento de una ola unidad de código.

Las pruebas de integración en Java son un tipo de prueba de software que se centran en verificar que los distintos módulos o componentes de una aplicación funcionen correctamente juntos. Estas pruebas se realizan después de las pruebas unitarias y antes de las pruebas de sistema completas. El objetivo es identificar problemas que puedan surgir cuando los componentes individuales se combinan e interactúan entre sí.

47. ¿Cuál es el comando utilizado para crear una nueva rama en Git?

- a) `git commit`
- b) `git branch`
- c) `git merge`
- d) `git push`

El comando utilizado para crear una nueva rama en Git es `git branch`.

- git commit: Se usa para guardar los cambios en el repositorio local.
- git merge: Se usa para fusionar ramas.
- git push: Se usa para enviar los commits locales al repositorio remoto.

48. ¿Cuál es el paquete de importación necesario para usar la clase ArrayList?

- a) `import.java.net.*;`
- b) `import.java.awt.*;`
- c) `import.java.io.*;`
- d) `import.java.util.*;`

La clase ArrayList está ubicada en el paquete `java.util`, por lo que necesitas importar este paquete para utilizar ArrayList en tu código. La opción correcta para importar todo el contenido de `java.util` es `import java.util.*;`.

49. ¿Cuál es el formato de los datos que se envían y reciben en una API REST?

- a) YAML
- b) XML
- c) JSON
- d) Todos los anteriores

En una API REST, el formato de los datos que se envían y reciben suele ser JSON (JavaScript Object Notation), aunque también se puede utilizar XML y otros formatos menos comunes como YAML o CSV. JSON es el formato más popular debido a su simplicidad, legibilidad y compatibilidad con la mayoría de los lenguajes de programación.

50. ¿Cuál es la función del operador de doble dos puntos (::) en Java 8?

- a) El operador doble dos puntos se utiliza para crear una nueva instancia de una clase en Java 8.
- b) EL operador de doble dos puntos no se utiliza en java 8.
- c) El operador de doble dos puntos se utiliza para acceder a métodos estáticos en Java 8.
- d) El operador de doble dos puntos se utiliza para acceder a métodos no estáticos en Java 8.

En Java 8, el operador de doble dos puntos (::) se utiliza para referenciar métodos de forma abreviada, especialmente en contextos donde se espera una interfaz funcional compatible. Esto se conoce como referencias a métodos y es una característica de programación funcional introducida

en Java 8 que permite pasar métodos como argumentos a métodos o construir implementaciones de interfaces funcionales sin necesidad de escribir expresiones lambda.

51. ¿Qué palabra clave se utiliza para definir una excepción personalizada en Java?

- a) try
- b) throw
- c) finally
- d) catch

En Java, throw es una palabra clave que se utiliza para lanzar una excepción explícitamente dentro de un bloque de código. Sirve para indicar que ha ocurrido una condición excepcional que no puede ser manejada dentro del método actual y necesita ser propagada hacia un contexto superior para su manejo.

Cuando utilizas throw, estás creando una instancia de una clase de excepción y la estás "lanzando" (throwing) para notificar al código que está llamando al método actual que ha ocurrido un problema.

52. ¿Qué es un operador de short circuit?

- e) Sirve para realizar más eficientes las operaciones condicionales evitando ejecutar operaciones si estas ya no son necesarias.
- f) Operador que nos sirve para crear una nueva clase anónima.
- g) Sirve para lanzar una excepción personalizada.
- h) Es un patrón de arquitectura de microservicios que nos permite evitar el consumo de servicios que están en mantenimiento.

En Java, un operador de short-circuit es un operador lógico que no evalúa su segundo operando si el primer operando es suficiente para determinar el resultado de la expresión. Esto se utiliza para optimizar el rendimiento y evitar errores innecesarios o costosos al evaluar condiciones.

53. ¿Cuál de los siguientes comandos elimina el directorio target antes de iniciar el proceso de construcción?

- a) mvn site
- b) mvn build
- c) mvn answer
- d) mvn clean

El comando que elimina el directorio target antes de iniciar el proceso de construcción es:

Mvn clean

El comando `mvn clean` ejecuta la fase `clean` del ciclo de vida de Maven, que elimina los archivos y directorios generados previamente durante la compilación y el proceso de construcción, incluido el directorio `target`. Esto garantiza que el entorno esté limpio antes de comenzar el proceso de construcción, lo que puede ser útil para evitar conflictos o problemas causados por archivos obsoletos.

54. ¿Cuál es el comando utilizado para ver el historial de cambios en Git?

- a) `git log`
- b) `git status`
- c) `git commit`
- d) `git diff`

El comando utilizado para ver el historial de cambios en Git es `a) git log`. Este comando muestra una lista de confirmaciones (commits) en orden cronológico inverso, lo que significa que la confirmación más reciente se muestra primero. Cada entrada en el registro de confirmaciones incluye detalles como el autor, la fecha y hora de la confirmación, y un mensaje descriptivo que proporciona información sobre los cambios realizados en esa confirmación.

55. ¿Qué es una expresión lambda en Java 8?

- a) Una expresión lambda es una forma de escribir una clase anónima en Java 8.
- b) Una expresión lambda es una forma de escribir una función anónima en Java 8.
- c) Una expresión lambda es un método que se llama automáticamente cuando se crea un objeto.
- d) Una expresión lambda es un método que se llama de forma explícita desde el código.

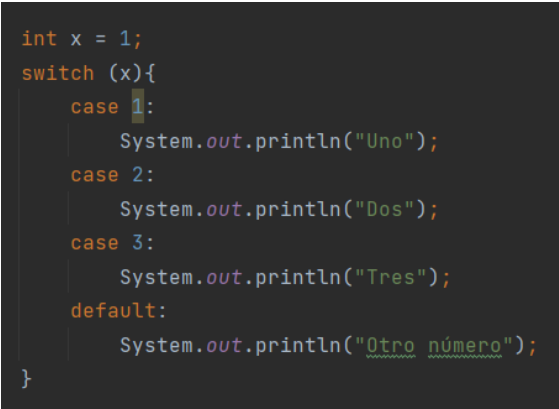
Una expresión lambda en Java 8 es una característica que permite representar de manera concisa y clara una implementación de una interfaz funcional. Proporciona una forma de expresar código de manera más compacta y legible, especialmente cuando se trata de operaciones que implican el uso de interfaces funcionales, como aquellas que requieren la implementación de un único método abstracto.

56. ¿Cuál es la diferencia entre una clase abstracta y una interfaz en Java 8?

- e) Una interfaz solo puede heredar de una clase, mientras una clase abstracta puede heredar de múltiples interfaces
- f) Una clase abstracta puede contener implementaciones de métodos, mientras que una interfaz no puede.
- g) Una clase abstracta puede contener variables de instancia, mientras que una interfaz no puede.
- h) Una interfaz puede contener implementaciones de métodos, mientras que una clase abstracta no puede.

57. ¿Qué muestra el siguiente código fuente por pantalla?

```
int x = 1;
switch (x){
case 1:
System.out.println("Uno");
case 2:
System.out.println("Dos");
case 3:
System.out.println("Tres");
default:
System.out.println("Otro número");
}
```



```
int x = 1;
switch (x){
    case 1:
        System.out.println("Uno");
    case 2:
        System.out.println("Dos");
    case 3:
        System.out.println("Tres");
    default:
        System.out.println("Otro número");
}
```

- a) Dos
- b) Uno Dos Tres Otro número**
- c) Uno
- d) Otro número

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

```
public class Test1 extends Concreate{
1 usage
Test1(){
System.out.println("t");
}
public static void main(String[] args) {
// TODO Auto-generated method stub
```

```
new Test1();
```

```
}
```

```
}
```

```
1 usage 1 inheritor
```

```
class Concreate extends Send{
```

```
1 usage
```

```
Concreate(){
```

```
System.out.println("c");
```

```
}
```

```
private Concreate (String s) {
```

```
ال
```

```
}
```

```
}
```

```
1 usage 2 inheritors
```

```
Jabstract class Send{
```

```
2 usages
```

```
Send(){
```

```
System.out.println("s");
```

```
}
```

```

public class Test1 extends Concreate{
    1 usage
    Test1(){
        System.out.println("t ");
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        new Test1();
    }
}
1 usage 1 inheritor
class Concreate extends Send{
    1 usage
    Concreate(){
        System.out.println("c ");
    }
    private Concreate(String s){

    }
}
1 usage 2 inheritors
abstract class Send{
    2 usages
    Send(){
        System.out.println("s ");
    }
}

```

- a) c,s,t
- b) t,s,c
- c) Error en tiempo de ejecución
- d) No compila

59. ¿Qué es REST y cuál es su relación con las API web?

- d) REST es un protocolo de comunicación. Su relación con las API web es que las utiliza para definir los endpoints de una API
- e) REST es in lenguaje de programación. Su relación con las API web es que se utiliza para crear aplicaciones web.
- f) REST es un servicio en la nube. Su relación con las API web que se utiliza para alojar las aplicaciones web.
- g) REST es una arquitectura para aplicaciones web. Su relación con las API web es que se utiliza para definir la estructura y funcionalidades de una API.

En Java 8, un Stream es una secuencia de elementos que admite operaciones de agregación y secuenciales o paralelas. Proporciona una forma más elegante y funcional de procesar colecciones de datos en Java.

Características Clave de Streams:

Secuenciales o Paralelos: Los streams pueden ser procesados de forma secuencial o paralela, lo que permite aprovechar el paralelismo de forma transparente para mejorar el rendimiento en sistemas multicore.

Operaciones de Agregación: Los streams admiten una variedad de operaciones de agregación, como filtrado, mapeo, ordenamiento, agrupación y reducción, que se pueden encadenar para formar una canalización de procesamiento de datos.

No Almacenan Datos: Los streams no almacenan datos en sí mismos, sino que proporcionan una vista sobre una colección de datos existente o una secuencia de elementos generada dinámicamente.

Perezosos: Las operaciones en un stream son perezosas, lo que significa que no se ejecutan hasta que se accede al resultado final del stream. Esto permite optimizaciones como el cortocircuito (short-circuiting) y la evaluación bajo demanda (lazy evaluation).

60. De los siguientes paquetes, ¿cuáles contienen clases para construir una interfaz gráfica? (Elije todas las que correspondan)

- a) java.net
- b) java.io
- c) javax.swing
- d) java.util
- e) java.awt

61. ¿Cuál de las siguientes líneas deben ir en el espacio en blanco para que el código compile?

```
public class News < _____> { }
```

- a) Solo N
- b) Solo ?
- c) Ninguna de las anteriores
- d) News,y Object
- e) ? y N
- f) N, News y Object

62. ¿Qué es el patrón de diseño DAO y cómo se implementa en Java?

- e) El patrón de diseño DAO es un patrón que se utiliza para abstraer la capa de acceso a datos en una aplicación. Se puede implementar en Java utilizando interfaces y clases concretas.

- f) El patrón de diseño DAO es un patrón que se utiliza para abstraer la capa de negocios de una aplicación. Se puede implementar en Java utilizando clases abstractas y métodos estáticos.
- g) El patrón de diseño DAO es un patrón que se utiliza para abstraer la capa de presentación en una aplicación. Se puede implementar en Java utilizando interfaces y clases concretas.
- h) El patrón de diseño DAO es un patrón que se utiliza para abstraer la capa de infraestructura en una aplicación. Se puede implementar en Java utilizando excepciones y bloques try-catch.

Conceptos Clave del Patrón DAO:

DAO Interface: Define los métodos que realizarán las operaciones CRUD.

DAO Implementation: Implementa la interfaz DAO y contiene la lógica concreta para interactuar con la base de datos.

DTO (Data Transfer Object): También conocido como VO (Value Object), es una clase que representa la estructura de los datos que se manejan.

Data Source: El recurso de la base de datos que se utiliza para realizar las operaciones CRUD.

Conceptos Clave del Patrón DAO:

DAO Interface: Define los métodos que realizarán las operaciones CRUD.

DAO Implementation: Implementa la interfaz DAO y contiene la lógica concreta para interactuar con la base de datos.

DTO (Data Transfer Object): También conocido como VO (Value Object), es una clase que representa la estructura de los datos que se manejan.

Data Source: El recurso de la base de datos que se utiliza para realizar las operaciones CRUD.

63. ¿Cuál de las siguientes líneas deben ir en el espacio en blanco para que el código compile?

```
public class News < _____ > { }
```

- g) Solo N
- h) Solo ?
- i) Ninguna de las anteriores
- j) News,y Object
- k) ? y N
- l) N, News y Object -----**

64. ¿Qué es un stream en Java 8 y para qué se utiliza?

- a) Un objeto que representa una conexión de entrada o salida de datos.
- b) Un objeto que representa una secuencia de elementos y se utiliza para procesar colecciones de forma declarativa.
- c) Un objeto que se utiliza para leer y escribir archivos de texto.
- d) Un objeto que se utiliza para crear y manipular bases de datos.

Los streams se utilizan para realizar operaciones de procesamiento de datos de manera declarativa y funcional en colecciones de elementos. Algunos casos de uso comunes incluyen:

Filtrado de Datos: Seleccionar elementos de una colección que cumplan con ciertos criterios mediante operaciones como `filter()`.

Transformación de Datos: Aplicar una función a cada elemento de una colección para transformarlo en otro valor mediante operaciones como `map()`.

Agregación de Datos: Realizar operaciones de agregación, como sumar, contar o encontrar el máximo/mínimo, en los elementos de una colección mediante operaciones como `sum()`, `count()`, `max()`, `min()`, etc.

Ordenamiento y Agrupación: Ordenar los elementos de una colección o agruparlos en función de ciertos criterios mediante operaciones como `sorted()` y `groupingBy()`.

65. Son patrones de diseño de microservicios

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

Los patrones mencionados son comúnmente utilizados en arquitecturas de microservicios para mejorar la resiliencia y la capacidad de manejo de errores del sistema. Aquí tienes una breve descripción de cada uno:

c) Circuit Breaker (Interruptor de Circuito):

Este patrón se utiliza para detectar y evitar la propagación de fallas en un sistema distribuido. Funciona como un interruptor que se abre cuando un servicio alcanza un umbral de fallo predefinido, evitando que se realicen nuevas llamadas al servicio afectado hasta que se recupere.

d) Retry (Reintentar):

Este patrón se utiliza para reintentar automáticamente una operación que ha fallado temporalmente debido a condiciones transitorias, como fallos de red o sobrecarga de servidores. Puede configurarse para realizar múltiples intentos con un intervalo de tiempo entre ellos.

e) Bulkhead (Cubierta Estanca):

Este patrón se utiliza para evitar que un fallo en un componente del sistema afecte a otros componentes. Divide el sistema en compartimentos (o bulkheads) independientes y limita el impacto de un fallo al aislar los recursos asociados con cada compartimento.

f) Adaptive LIFO (Último en Entrar, Primero en Salir Adaptativo):

Este patrón se utiliza para gestionar la carga en un sistema distribuido de manera adaptativa, dando prioridad a las solicitudes que tienen mayor probabilidad de completarse con éxito. Se adapta dinámicamente a las condiciones de carga y rendimiento del sistema para maximizar la eficiencia y la disponibilidad.

66. ¿Cuándo se debe usar un bloque finally en una declaración try regular (no una prueba con recursos)?

- a) Nunca.
- b) Cuando el código del programa no termina por sí solo.
- c) Cuando hay dos o más bloques catch en una sentencia try.
- d) Cuando hay exactamente un bloque catch en una sentencia try
- e) Cuando no hay bloques catch en una declaración try

El bloque finally en una declaración try regular se utiliza para ejecutar código que debe ejecutarse independientemente de si ocurre una excepción o no, y también incluso si ocurre una excepción y no se captura en ningún bloque catch. Este bloque se ejecutará siempre, garantizando que ciertas operaciones se realicen antes de que finalice la ejecución del programa, independientemente de las circunstancias.

67. De los siguientes ¿qué tipos de declaraciones se deben usar para contar la cantidad de monedas de 5 centavos en una matriz de cadenas de varias monedas?

- a) Assertion
- b) Iteration
- c) Assignment
- d) Conditional

68. ¿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.

69. ¿Cuál no es un objetivo de Maven?

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

Maven es una herramienta de gestión de proyectos y comprensión que se utiliza principalmente en proyectos Java. Fue desarrollado por Apache Software Foundation. El objetivo principal de Maven es simplificar y estandarizar el proceso de construcción y gestión de proyectos, abordando varios aspectos del ciclo de vida del proyecto.

Gestión de Dependencias.

Estandarización del Proceso de Construcción.

Facilidad de Gestión del Proyecto.

Reusabilidad de Código.

Integración con Herramientas de Construcción y Despliegue.

Facilidad de Uso.

70. ¿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 es fundamental para cualquier flujo de trabajo de Git, ya que permite a los desarrolladores obtener rápidamente una copia de trabajo de un proyecto existente. Este comando se utiliza al principio de un proyecto o cuando necesitas colaborar en un repositorio ya existente, proporcionando una manera fácil y eficiente de trabajar con el código fuente y su historial completo.

71. ¿Cuál es el comando utilizado para fusionar una rama en Git?

- a) git push
- b) git branch
- c) git pull
- d) git merge

a) git push: Este comando se utiliza para enviar los cambios locales confirmados a un repositorio remoto. Por ejemplo, después de hacer cambios locales y confirmarlos con git commit, puedes usar git push para subir esos cambios al repositorio remoto.

b) git branch: Este comando se utiliza para listar, crear o eliminar ramas en un repositorio Git. Puedes usar git branch sin argumentos para listar todas las ramas en el repositorio actual, o con argumentos para crear o eliminar ramas.

c) git pull: Este comando se utiliza para recuperar los cambios desde un repositorio remoto y fusionarlos automáticamente con tu repositorio local. Es básicamente equivalente a ejecutar git fetch seguido de git merge. Por ejemplo, puedes usar git pull para actualizar tu repositorio local con los cambios que otros colaboradores han subido al repositorio remoto.

d) git merge: Este comando se utiliza para fusionar una rama específica en otra rama activa. Por ejemplo, si tienes una rama de funcionalidad en la que has estado trabajando y quieres incorporar los cambios en la rama principal, puedes usar git merge para fusionar los cambios de la rama de funcionalidad en la rama principal.

72. ¿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 copia de un repositorio Git que reside en un servidor remoto, como GitHub, GitLab o un servidor Git propio. Este repositorio remoto actúa como un punto central de colaboración para un proyecto de desarrollo de software, permitiendo que múltiples desarrolladores trabajen juntos en el mismo código.

73. ¿Cuál es el comando utilizado para actualizar la rama local con los cambios de la rama remota en Git?

- a) git pull
- b) git push

- c) git clone
- d) git checkout

74. Dada la siguiente clase

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

```
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.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"));

75. ¿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);
    }
}

```

```

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); 11
    }
}

```

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

76. ¿Qué es Git?

- a) Una herramienta de automatización de compilación que se utiliza para compilar un proyecto.
- b) Una herramienta de generación de informes que se utiliza para generar informes sobre el rendimiento de una aplicación.
- c) Una herramienta de gestión de dependencias que se utiliza para descargar bibliotecas y paquetes en un proyecto de Java.
- d) Una herramienta de control de versiones que se utiliza para almacenar y administrar el código fuente de un proyecto.

Git es un sistema de control de versiones distribuido que se utiliza principalmente para el seguimiento de cambios en el código fuente durante el desarrollo de software. Fue creado por

Linus Torvalds en 2005 para el desarrollo del kernel de Linux, pero desde entonces se ha convertido en una herramienta fundamental en el mundo del desarrollo de software.

77. ¿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

Declaración y definición de work:

Se crean dos parámetros StringBuilder llamados a y b.

Dentro del método, a se reasigna a un nuevo StringBuilder con el valor "a".

Se añade "b" al contenido de b.

El método retorna la nueva instancia de a.

Método main:

Se crea s1 como new StringBuilder("s1").

Se crea s2 como new StringBuilder("s2").

Se llama al método work con s1 y s2, asignando el resultado a s3.

Análisis de la ejecución

work(s1, s2):

a se reasigna a new StringBuilder("a"), pero esto no afecta a s1 fuera del método.

b.append("b") modifica s2, agregando "b" al final de su contenido. Ahora s2 es "s2b".

El método retorna el nuevo StringBuilder("a"), asignándolo a s3.

Resultados esperados

s1 permanece igual, es decir, "s1".

s2 se modifica y se convierte en "s2b".

s3 es una nueva instancia de StringBuilder con el valor "a".

c) s1 = s1

d) s3 = a

f) s2 = s2b

78. ¿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.

En Java 8, el operador de doble dos puntos (::) se utiliza en la referencia a métodos, que es una característica introducida en Java 8 como parte de las expresiones lambda y los métodos de referencia. La función principal del operador :: es crear una referencia a un método existente o un constructor para su posterior uso en expresiones lambda o como argumento para métodos funcionales.

Tipos de Referencia de Método:

Referencia a un Método Estático: Clase::metodoEstatico

Se refiere a un método estático en una clase.

Referencia a un Método de Instancia de un Objeto Particular: objeto::metodoDeInstancia

Se refiere a un método de instancia de un objeto particular.

Referencia a un Método de Instancia de un Tipo Arbitrario: Tipo::metodoDeInstancia

Se refiere a un método de instancia de un tipo arbitrario (interfaz funcional) para el que se proporcionan argumentos.

Referencia a un Constructor: Clase::new

Se utiliza para referenciar un constructor de una clase.

79. ¿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.

80. ¿Cuál de las siguientes afirmaciones son verdaderas? (Elije todas las correctas)

- a) Las excepciones de tiempo de ejecución son lo mismo que las excepciones comprobadas.
- b) Las excepciones en tiempo de ejecuciones son lo mismo que las excepciones no comprobadas.
- c) Solo pueden manejar subclases de exception.
- d) Solo puede declarar excepciones comprobadas (checked).
- e) Puede declarar solo excepciones no comprobadas.

81. ¿Cuáles son las salidas del siguiente código? (Elije todas las correctas)

```
public class StrinBuilders {
```

```

    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);
    }
}

```

```

public class StrinBuilders {
    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) El código no compila.

b) s3 = a

c) s1 = a

d) s2 = s2

e) s1 = s1

f) s3 = null

g) s2 = s2b

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

```

public class Test1 extends Concreate{

```

```
Test1(){  
    System.out.println("t");  
}  
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    new Test1();  
}  
}  
class Concreate extends Send{  
    Concreate(){  
        System.out.println("c");  
    }  
    private Concreate (String s) {  
    }  
}  
abstract class Send{  
    Send(){  
        System.out.println("s");  
    }  
}
```

```

public class Test1 extends Concreate{
    1 usage
    Test1(){
        System.out.println("t ");
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        new Test1();
    }
}
1 usage 1 inheritor
class Concreate extends Send{
    1 usage
    Concreate(){
        System.out.println("c ");
    }
    private Concreate(String s){

    }
}
1 usage 2 inheritors
abstract class Send{
    2 usages
    Send(){
        System.out.println("s ");
    }
}

```

- a) Error en tiempo de ejecución.
- b) No compila.
- c) t,s,c
- d) c,s,t

83. ¿Cuál es la función del operador de doble dos puntos (::) en Java 8?

- e) El operador doble dos puntos se utiliza para crear una nueva instancia de una clase en Java 8.
- f) EL operador de doble dos puntos no se utiliza en java 8.
- g) El operador de doble dos puntos se utiliza para acceder a métodos estáticos en Java 8.
- h) El operador de doble dos puntos se utiliza para acceder a métodos no estáticos en Java 8.

En Java 8, el operador de doble dos puntos (::) se utiliza en la referencia a métodos, que es una característica introducida en Java 8 como parte de las expresiones lambda y los métodos de referencia. La función principal del operador :: es crear una referencia a un método existente o un constructor para su posterior uso en expresiones lambda o como argumento para métodos funcionales.

Tipos de Referencia de Método:

Referencia a un Método Estático: Clase::metodoEstatico

Se refiere a un método estático en una clase.

Referencia a un Método de Instancia de un Objeto Particular: objeto::metodoDeInstancia

Se refiere a un método de instancia de un objeto particular.

Referencia a un Método de Instancia de un Tipo Arbitrario: Tipo::metodoDeInstancia

Se refiere a un método de instancia de un tipo arbitrario (interfaz funcional) para el que se proporcionan argumentos.

Referencia a un Constructor: Clase::new

Se utiliza para referenciar un constructor de una clase.

84. ¿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

El acrónimo CRUD en una API REST se refiere a las operaciones básicas que se pueden realizar en cualquier tipo de sistema de gestión de bases de datos o en una interfaz de usuario que interactúa con ellas. CRUD representa las cuatro operaciones fundamentales de manipulación de datos:

Create (Crear): Crear nuevos registros o recursos en la base de datos. En el contexto de una API REST, esto se corresponde con la operación de creación de un nuevo recurso mediante una solicitud POST.

Read (Leer): Leer o recuperar datos existentes de la base de datos. En una API REST, esto se logra mediante una solicitud GET para recuperar uno o varios recursos.

Update (Actualizar): Actualizar registros o recursos existentes en la base de datos con nuevos datos. En una API REST, esto se hace mediante una solicitud PUT o PATCH para modificar un recurso existente.

Delete (Eliminar): Eliminar registros o recursos existentes de la base de datos. En una API REST, esto se logra mediante una solicitud DELETE para eliminar un recurso específico.

85. ¿Qué es un bean en Spring?

- e) Un archivo de configuración XML que se utiliza para definir la estructura de una tabla de base de datos.
- f) Una instancia de una clase que se administra por el contenedor de Spring.

- g) Una herramienta de inyección de dependencias que se utiliza para inyectar dependencias en una clase.
- h) Una clase que se utiliza para configurar la conexión a una base de datos.

Un bean en Spring es un objeto que es gestionado por el contenedor IoC de Spring. La gestión de beans incluye la creación, configuración y el ciclo de vida completo del objeto, facilitando así la inyección de dependencias y promoviendo la modularidad y reutilización de componentes en la aplicación.

86. ¿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.

Utilizar un profile dentro del archivo pom.xml en Maven es útil para configurar y personalizar el proceso de construcción de tu proyecto en función de diferentes entornos, situaciones o condiciones específicas. Los perfiles te permiten definir diferentes configuraciones, como dependencias, plugins, propiedades y otros elementos, que se activan o desactivan según ciertos criterios.

87. ¿Cuál es el comando utilizado para actualizar la rama local con los cambios de la rama remota en Git?

- a) git checkout
- b) git clone
- c) git push
- d) git pull

git checkout: Este comando se utiliza principalmente para cambiar entre ramas en un repositorio Git. También se utiliza para cambiar el estado de los archivos en el directorio de trabajo. Por ejemplo, puedes usar `git checkout <nombre_de_rama>` para cambiar a una rama específica.

git clone: Este comando se utiliza para clonar un repositorio Git existente en un nuevo directorio. Por ejemplo, puedes usar `git clone <URL_del_repositorio>` para crear una copia local de un repositorio remoto.

git push: Este comando se utiliza para enviar los cambios locales confirmados a un repositorio remoto. Por ejemplo, después de hacer cambios locales y confirmarlos con `git commit`, puedes usar `git push` para subir esos cambios al repositorio remoto.

git pull: Este comando se utiliza para recuperar los cambios desde un repositorio remoto y fusionarlos automáticamente con tu repositorio local. Es básicamente equivalente a ejecutar `git fetch` seguido de `git merge`. Por ejemplo, puedes usar `git pull` para actualizar tu repositorio local con los cambios que otros colaboradores han subido al repositorio remoto.

88. 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
    }
}
```

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";

89. 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, Map, Set y Queue son interfaces fundamentales en la colección de clases del marco de trabajo Java Collection Framework. Cada una de estas interfaces tiene sus propias características y usos específicos.

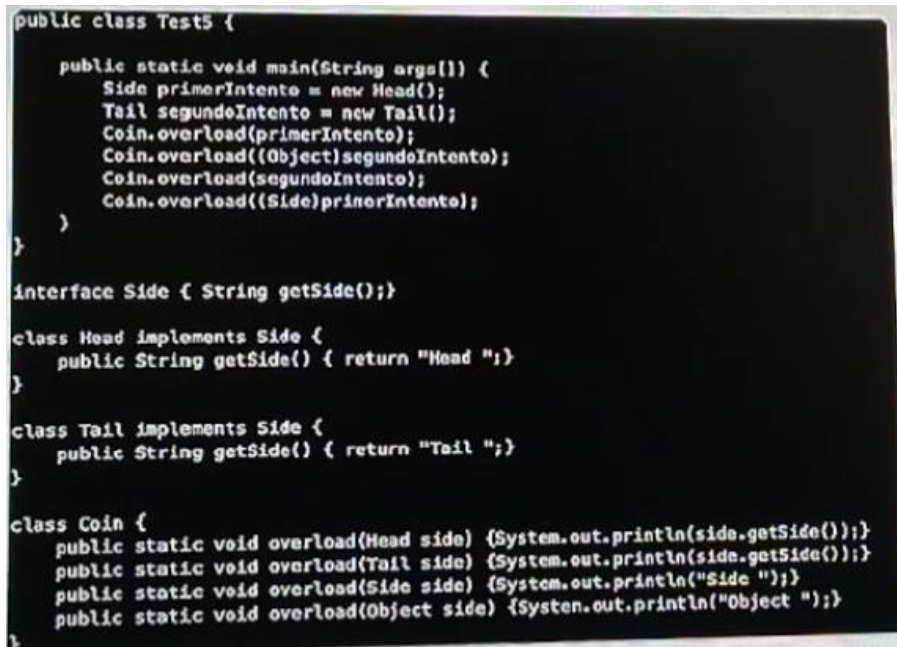
Una List es una colección ordenada que permite elementos duplicados. Cada elemento de la lista tiene un índice asociado, y los elementos pueden ser accesibles por su posición en la lista.

Un Map es una colección de pares clave-valor. Cada clave única está asociada con exactamente un valor. Los mapas no permiten claves duplicadas, pero los valores pueden ser duplicados.

Un Set es una colección que no permite elementos duplicados. No garantiza el orden de los elementos (excepto LinkedHashSet que mantiene el orden de inserción y TreeSet que mantiene el orden natural).

Una Queue es una colección diseñada para mantener elementos antes de su procesamiento. Generalmente, sigue la política FIFO (First-In-First-Out), pero hay implementaciones como PriorityQueue que no sigue FIFO.

90. 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 ");}  
}
```

```
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)prinarIntento);  
        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

91. ¿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);
    } }

```

```

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

92. ¿Cuál de los siguientes es cierto acerca de una subclase completa?

- 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.

93. ¿Cuál es la salida del siguiente código?

```
protected abstract void catchAnObject(Object x);

    public static void main(String[] args) {
        java.util.Date now = 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 abstract class Catchable {
    public void catchAnObject(String s) {
        System.out.println("Caught string");
    }
}
```

```

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

Clase abstracta Catchable:

Contiene un método abstracto catchAnObject que toma un parámetro de tipo Object.

Clase MyStringCatcher:

Extiende Catchable e implementa el método abstracto catchAnObject(Object x) de la clase base.

Tiene un método sobrecargado catchAnObject(String s) que toma un parámetro de tipo String.

Método main:

Crea una instancia de java.util.Date llamada now.

Crea una referencia de tipo Catchable que apunta a una instancia de MyStringCatcher.

Llama al método catchAnObject con el objeto now como argumento.

Ejecución

Cuando se llama a target.catchAnObject(now), la JVM busca el método correspondiente en tiempo de ejecución.

Dado que now es de tipo java.util.Date, que es una subclase de Object pero no una subclase de String, el método llamado será catchAnObject(Object x).

El método catchAnObject(Object x) imprime "Caught object".

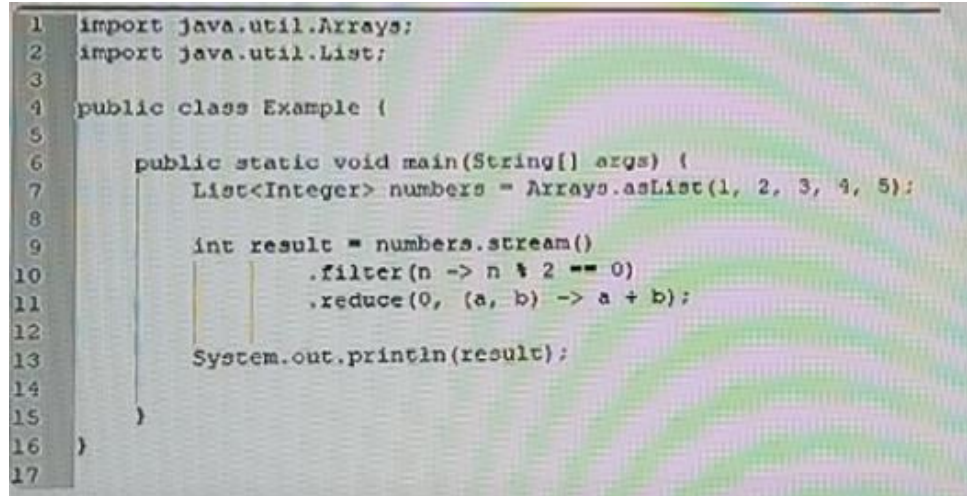
Conclusión

La salida del programa será:

La respuesta correcta es:

e) Caught object

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

A screenshot of a code editor showing Java code. The code is as follows:

```
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
```

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 ->20)

.reduce(0, (a, b) -> a + b);

System.out.println(result);

}

)

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

95. 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;

- 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.

Para incluir clases del paquete java.util en un archivo fuente de Java, debes importar el paquete en tu archivo. Puedes hacerlo utilizando la palabra clave import seguida del nombre completo del paquete. Aquí tienes un ejemplo de cómo hacerlo:

```
import java.util.*;
```

96. ¿Qué es Git?

- e) Una herramienta de automatización de compilación que se utiliza para compilar un proyecto.
- f) Una herramienta de generación de informes que se utiliza para generar informes sobre el rendimiento de una aplicación.
- g) Una herramienta de gestión de dependencias que se utiliza para descargar bibliotecas y paquetes en un proyecto de Java.
- h) Una herramienta de control de versiones que se utiliza para almacenar y administrar el código fuente de un proyecto.

Git es un sistema de control de versiones distribuido ampliamente utilizado para el seguimiento de cambios en el código fuente durante el desarrollo de software. Fue creado por Linus Torvalds en 2005 y se ha convertido en una herramienta fundamental para los desarrolladores en todo el mundo.

97. ¿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.

98. ¿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 de diseño Singleton se utiliza para garantizar que una clase tenga una única instancia y proporcionar un punto de acceso global a esa instancia. Esto es útil cuando se necesita exactamente una instancia de una clase para coordinar acciones en todo el sistema, como un objeto de configuración, un administrador de conexiones de bases de datos o un registro de eventos.

99. ¿Qué es un Microservicio?

- a) Ninguna de las anteriores
- b) Es un componentes que se pueden desplegar de forma independiente en función múltiple, es decir, englobando endpoint que no necesariamente están relacionados.
- c) Es el conjunto de endpoints contenidos en múltiples desarrollos que se despliegan en conjunto y que están estrechamente relacionados.
- d) Es un componentes que se pueden desplegar de forma independiente y que suelen ser de función única, es decir, englobando endpoint y están estrechamente relacionandos.

La clase del paquete java.io que permite leer y escribir archivos en ubicaciones específicas dentro de un archivo se llama RandomAccessFile. Esta clase proporciona métodos para leer y escribir bytes

en cualquier posición dentro de un archivo, lo que le permite acceder a datos de manera no secuencial.

RandomAccessFile es especialmente útil cuando necesitas leer o escribir datos en un archivo de manera aleatoria, es decir, cuando necesitas acceder a partes específicas del archivo sin tener que leer o escribir todo el archivo secuencialmente.

Algunas de las operaciones que puedes realizar con RandomAccessFile incluyen:

Lectura y Escritura en Posiciones Específicas: Puedes leer y escribir bytes en cualquier posición del archivo utilizando los métodos read y write.

Mover el Puntero de Lectura/Escritura: Puedes mover el puntero de lectura o escritura a una posición específica dentro del archivo utilizando el método seek.

Lectura y Escritura de Tipos de Datos Primitivos: Además de leer y escribir bytes individuales, RandomAccessFile proporciona métodos para leer y escribir tipos de datos primitivos como enteros, flotantes, dobles, etc.

100. ¿Cuál no es un objetivo de Maven?

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

101. 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.

Elección entre PUT y PATCH

Usa PUT cuando necesitas actualizar completamente un recurso o garantizar que el recurso tenga un estado específico después de la operación.

Usa PATCH cuando solo necesitas hacer cambios parciales y deseas evitar la sobrecarga de enviar datos que no han cambiado.

102. ¿Qué es una expresión lambda en Java 8?

- a) Una expresión lambda es una forma de escribir una clase anónima en Java 8.
- b) Una expresión lambda es una forma de escribir una función anónima en Java 8.
- c) Una expresión lambda es un método que se llama automáticamente cuando se crea un objeto.
- d) Una expresión lambda es un método que se llama de forma explícita desde el código.

103. ¿Cuál es la diferencia entre una clase abstracta y una interfaz en Java 8?

- a. Una interfaz solo puede heredar de una clase, mientras una clase abstracta puede heredar de múltiples interfaces
- b. Una clase abstracta puede contener implementaciones de métodos, mientras que una interfaz no puede.
- c. Una clase abstracta puede contener variables de instancia, mientras que una interfaz no puede.
- d. Una interfaz puede contener implementaciones de métodos, mientras que una clase abstracta no puede.

104. ¿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 bean y exponen la serialización de las clases mientras que @Controller. @Component requiere de una inicialización manual.

105. ¿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.

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

- a) Mayor seguridad.
- b) Mayor facilidad de implementación.¹
- c) Mayor velocidad de transferencia de datos.
- d) Mayor compatibilidad con diferentes plataformas.

Las APIs REST (Representational State Transfer) son un estilo de arquitectura que ofrece varias ventajas sobre otros tipos de servicios web, como SOAP (Simple Object Access Protocol) y RPC (Remote Procedure Call). A continuación se enumeran algunas de las principales ventajas de usar APIs REST:

Simplicidad y Facilidad de Uso:

Protocolo HTTP: REST utiliza el protocolo HTTP, que es ampliamente conocido y entendido, lo que simplifica el desarrollo y la integración.

Operaciones CRUD: Las operaciones de creación, lectura, actualización y eliminación (CRUD) se corresponden directamente con los métodos HTTP (POST, GET, PUT, DELETE), lo que facilita su comprensión y uso.

Escalabilidad:

Sin Estado (Stateless): Las APIs REST no mantienen estado en el servidor entre las llamadas. Cada solicitud contiene toda la información necesaria, lo que permite una mejor escalabilidad y manejo de la carga.

Cacheabilidad: Las respuestas pueden ser cacheadas, lo que reduce la carga en el servidor y mejora el rendimiento.

Flexibilidad y Extensibilidad:

Formato de Datos: REST no está limitado a un formato de datos específico. Aunque JSON es el más común, también se pueden usar XML, YAML, o cualquier otro formato.

Evolución: Es fácil extender y versionar las APIs REST, permitiendo que nuevas funcionalidades sean añadidas sin romper las versiones existentes.

Interoperabilidad:

Ampliamente Soportado: Debido a su uso del protocolo HTTP, las APIs REST son compatibles con una amplia variedad de clientes, incluyendo navegadores web, aplicaciones móviles y otros servicios web.

Lenguaje Agnóstico: Pueden ser consumidas por clientes escritos en cualquier lenguaje de programación que soporte HTTP, lo que las hace muy interoperables.

Mantenimiento y Depuración:

Legibilidad: Las URLs en REST suelen ser más legibles y semánticas, lo que facilita la depuración y el mantenimiento.

Herramientas: Existe una gran cantidad de herramientas y bibliotecas para consumir y probar APIs REST, lo que facilita el desarrollo y la depuración.

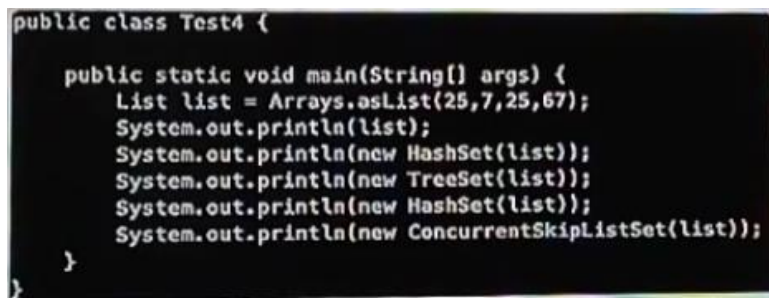
Desempeño:

Menor Sobrecarga: Las APIs REST tienden a ser menos verbosas que las basadas en XML (como SOAP), lo que puede reducir el ancho de banda necesario y mejorar el desempeño general de la aplicación.

Adopción y Comunidad:

Popularidad: REST es ampliamente adoptado en la industria, lo que significa que hay una gran cantidad de recursos, documentación y comunidades disponibles para soporte.

107. 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));  
    }  
}
```

```
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]

108. ¿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);
    }
}
```

```
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

109. ¿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.

110. ¿Cuál es el comando utilizado para ver el historial de cambios en git?

- a) git diff
- b) git status
- c) git log
- d) git commit

git diff: Muestra las diferencias entre el estado actual de tu área de trabajo y el último commit, o entre diferentes commits.

Git status: Muestra el estado actual del repositorio. Te informa sobre los archivos que han cambiado, los que están en el área de ensayo y los que no están rastreados por Git.

git log: Muestra el historial de commits en el repositorio. Incluye información como el autor, la fecha y el mensaje del commit.

Git commit : Registra los cambios en el repositorio. Los cambios que han sido añadidos al área de ensayo se guardan en el historial del repositorio con un mensaje descriptivo.

111. ¿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

Para ejecutar el intérprete de Java sin abrir la ventana de la consola en un sistema basado en MS Windows, puedes utilizar la utilidad de línea de comandos llamada javaw.

112. ¿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.

Un endpoint en una API REST es una URL específica que representa un recurso particular y a la cual se puede acceder para interactuar con ese recurso mediante diversas operaciones HTTP. Cada endpoint es una dirección única en la red que permite a los clientes realizar operaciones específicas como obtener datos, enviar datos, actualizar datos o eliminar datos relacionados con ese recurso.

113. ¿Cuál de lo siguiente es cierto?

- a) java toma el nombre del archivo .bytecode como parámetro.
- b) javac compila un archivo .java en un archivo .class
- c) Java toma el nombre de la clase como parámetro.
- d) javac compila un archivo .class en un archivo .java
- e) javac compila un archivo .java en un archivo .bytecode
- f) Java toma el nombre del archivo .class como parámetro.

c) Java toma el nombre de la clase como parámetro.

Cuando ejecutas un programa en Java, utilizas el comando java seguido del nombre de la clase que contiene el método main. Esta afirmación es cierta porque el comando java toma el nombre de la clase principal (no el nombre del archivo) como parámetro para ejecutar el programa.

d) javac compila un archivo .class en un archivo .java

Esta afirmación parece contener un error en la redacción. La forma correcta de expresarla sería:

"javac compila un archivo .java en un archivo .class"

Esta afirmación es cierta porque el compilador de Java (javac) toma como entrada un archivo fuente con extensión .java y lo compila en un archivo de bytecode con extensión .class, que puede ser ejecutado por la Java Virtual Machine (JVM).

114. ¿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

115. ¿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:

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

- a) Hacer que las pruebas dependan de otras pruebas
- b) probar solo una pequeña parte de una función
- c) ejecutar pruebas con poca frecuencia
- d) asegurarse de que las pruebas sean claras y concisas

117. ¿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

stmt1 compara dos literales de cadena que son idénticos. En Java, los literales de cadena que son idénticos comparten la misma referencia en memoria. Por lo tanto, stmt1 es true.

stmt2 compara una nueva instancia de la cadena "champ" con el literal de cadena "champ". En este caso, la nueva instancia creada con new String("champ") no es la misma referencia que el literal "champ", aunque el contenido sea el mismo. Por lo tanto, stmt2 es false.

stmt3 compara dos nuevas instancias de la cadena "champ", creadas con new String("champ"). Cada instancia creada con new String tiene una referencia diferente, aunque el contenido sea el mismo. Por lo tanto, stmt3 es false.

Finalmente, la expresión en System.out.println(stmt1 && stmt2 || stmt3) se evalúa así:

stmt1 && stmt2 es true && false, lo que da false.

stmt3 es false.

Entonces, false || false es false.

Por lo tanto, el resultado de ejecutar el código es false.

118. ¿Cómo se manejan las excepciones en Java?

- a) Las excepciones se 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.

119. ¿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 del paquete java.io que permite leer y escribir archivos en ubicaciones específicas dentro de un archivo se llama RandomAccessFile. Esta clase es útil cuando necesitas acceder a datos en un archivo de manera aleatoria, es decir, cuando necesitas leer o escribir en ubicaciones específicas dentro del archivo sin tener que leer o escribir todo el archivo secuencialmente.

RandomAccessFile proporciona métodos para leer y escribir bytes en cualquier posición dentro del archivo, y también permite mover el puntero de lectura/escritura a una ubicación específica.

120. 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

121. ¿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

Una expresión lambda en Java es una característica introducida en Java 8 que permite definir funciones anónimas de forma más concisa y flexible. Estas funciones pueden ser tratadas como objetos y pasadas como argumentos a métodos, lo que facilita la escritura de código más modular y expresivo.

Aquí hay algunas características clave de las expresiones lambda en Java:

1. Sintaxis Concisa:

La sintaxis de una expresión lambda es más compacta que la de una clase anónima, lo que reduce el código boilerplate.

2. Tipado Inferido:

En muchos casos, el compilador puede inferir el tipo de los parámetros de la expresión lambda, lo que elimina la necesidad de especificar tipos explícitamente.

3. Interfaz Funcional:

Las expresiones lambda se utilizan principalmente en el contexto de las interfaces funcionales, que son interfaces que tienen un único método abstracto (a menudo denominado "método funcional").

La expresión lambda se puede asignar a una variable de este tipo de interfaz funcional.

122. ¿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
- d) muestra los números del 0 al 9

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

docker build

La función de docker build es construir una imagen Docker a partir de un Dockerfile y un contexto de compilación. Este comando es fundamental en el proceso de crear imágenes personalizadas y automatizar el entorno de ejecución de aplicaciones.

2. ¿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 0 al 9
- b. Muestra un 10
- c. Se queda en un bucle infinito
- d. Muestra los números del 1 al 10

3. ¿Cuál es la diferencia entre una interfaz y una clase abstracta en java?

- a) Una clase abstracta puede contener variables y métodos concretos mientras que una interfaz sólo puede contener métodos abstractos
- b) Una clase abstracta puede ser instanciada mientras que una interfaz no puede ser instanciada
- c) Una interfaz puede ser implementada por múltiples clases
- d) Una interfaz puede contener variables y métodos concretos

En Java, tanto las interfaces como las clases abstractas se utilizan para definir contratos y comportamientos que las clases pueden implementar o extender. Sin embargo, existen diferencias clave entre ellas en cuanto a su propósito, sintaxis y uso. Aquí te presento un desglose detallado de las diferencias:

1. Propósito Principal:

Clase Abstracta:

Se utiliza para compartir una implementación común entre varias clases relacionadas. Puede contener tanto métodos abstractos (sin implementación) como métodos concretos (con implementación).

Permite definir una estructura común y un comportamiento por defecto que puede ser compartido por las subclasses.

Interfaz:

Se utiliza para definir un contrato o conjunto de métodos que una clase debe implementar. No contiene implementación (hasta Java 8) y se centra en definir qué se debe hacer, no cómo hacerlo.

Permite la definición de capacidades que pueden ser implementadas por clases no relacionadas.

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

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

La afirmación "Una subclase concreta debe implementar todos los métodos abstractos heredados" es correcta y se basa en los principios de la programación orientada a objetos, particularmente en el uso de clases abstractas y métodos abstractos.

5. ¿Cuándo se debe usar un bloque finally en una declaración de try regular no una prueba con recursos?

- a) Cuando hay exactamente un bloque catch en una sentencia try
- b) Cuando hay dos o más bloques catch en una sentencia try
- c) Nunca
- d) Cuando no hay bloques catch en una declaración try
- e) cuando el código del programa no termina por sí solo

El bloque finally en una declaración try se utiliza para ejecutar un conjunto de instrucciones que deben ejecutarse siempre, independientemente de si se lanza o no una excepción dentro del bloque try. Esto es útil para realizar tareas de limpieza o liberar recursos que no se manejan automáticamente mediante la declaración de prueba con recursos (try-with-resources), disponible a partir de Java 7.

Cuándo usar un bloque finally:

Liberar Recursos Manualmente:

Cuando estás gestionando recursos que no implementan la interfaz AutoCloseable y, por lo tanto, no se pueden usar con la declaración try-with-resources.

Ejemplo: Cerrar conexiones de bases de datos, sockets, archivos, etc., que se abrieron en el bloque try.

Acciones de Limpieza:

Para realizar tareas de limpieza necesarias que deben ejecutarse pase lo que pase, como borrar archivos temporales, liberar memoria o cerrar hilos.

Restaurar el Estado Original:

Cuando necesitas asegurar que ciertas condiciones se restablezcan independientemente del éxito o fallo de la operación en el bloque try.

Ejemplo: Restablecer configuraciones globales o de entorno.

6. ¿En los verbos REST cuál es la diferencia entre el verbo PATCH y PUT?

PUT requiere se le envíe entidad concreta mientras que PATCH solo los atributos a modificar.

PUT (Actualizar/Reemplazar):

Descripción: El verbo PUT se utiliza para actualizar un recurso existente o crear un nuevo recurso si no existe.

Operación: Reemplaza la representación completa del recurso con los datos proporcionados en la solicitud.

Idempotencia: PUT es idempotente, lo que significa que realizar la misma solicitud varias veces producirá el mismo resultado. Si envías una solicitud PUT varias veces con los mismos datos, el estado del recurso no cambiará después de la primera solicitud.

Uso: Utiliza PUT cuando quieres reemplazar todo el recurso con una nueva representación.

PATCH (Modificación Parcial):

Descripción: El verbo PATCH se utiliza para aplicar actualizaciones parciales a un recurso existente.

Operación: Modifica únicamente los campos especificados en la solicitud, sin afectar otros campos del recurso.

Idempotencia: PATCH no es necesariamente idempotente, aunque puede ser diseñado para serlo. Esto significa que realizar la misma solicitud PATCH varias veces podría producir diferentes resultados si las modificaciones son acumulativas o dependientes del estado actual del recurso.

Uso: Utiliza PATCH cuando quieres modificar parcialmente un recurso, actualizando solo los campos específicos.

7. De los siguientes paquetes ¿Cuáles contienen clases para construir una interfaz

- java.awt
- javax.swing

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

Probar sobre una pequeña parte de una función

Las pruebas unitarias son esenciales para garantizar la calidad del software, detectar errores de manera temprana y facilitar el mantenimiento del código. A continuación, se enumeran algunas buenas prácticas para construir pruebas unitarias efectivas:

1. Escribir Pruebas Independientes:

Cada prueba debe ser autónoma y no depender de otras pruebas. Esto asegura que pueden ejecutarse en cualquier orden sin afectar los resultados.

2. Mantener las Pruebas Simples y Claras:

Las pruebas deben ser fáciles de entender y mantener. Utiliza nombres descriptivos para las pruebas y sigue una estructura clara (Arrange-Act-Assert).

3. Nombrar las Pruebas Adecuadamente:

Usa nombres de pruebas que describan claramente lo que están verificando. Ejemplo: `shouldReturnTrueWhenInputIsValid`.

4. Cubrir Casos Positivos y Negativos:

Es importante probar tanto los casos de éxito (donde el código se comporta como se espera) como los casos de falla (donde el código maneja errores o condiciones excepcionales).

5. Usar Asserts Apropriados:

Verifica los resultados esperados utilizando asserts que comparen los valores esperados con los resultados reales. Ejemplos: `assertEquals`, `assertTrue`, `assertFalse`, `assertThrows`.

6. Aislar Unidades de Código:

Usa técnicas como el mocking para aislar la unidad de código que se está probando de sus dependencias. Herramientas como Mockito en Java, Moq en .NET o unittest.mock en Python son útiles para este propósito.

7. Evitar la Lógica Compleja en las Pruebas:

Las pruebas no deben contener lógica compleja. La complejidad puede introducir errores en las propias pruebas y hacerlas difíciles de entender.

8. Estructura AAA (Arrange-Act-Assert):

Arrange: Configura las condiciones iniciales y los datos necesarios.

Act: Ejecuta la unidad de código que se está probando.

Assert: Verifica que el resultado sea el esperado.

9. Utilizar Fixtures para Configuración Común:

Usa fixtures para configurar datos comunes necesarios para varias pruebas. Esto ayuda a reducir la duplicación de código y facilita el mantenimiento.

10. Automatizar y Ejecutar Frecuentemente:

Integra las pruebas unitarias en el proceso de integración continua (CI) para que se ejecuten automáticamente con cada cambio en el código.

11. Escribir Pruebas Rápidas:

Las pruebas unitarias deben ser rápidas para ejecutar. Las pruebas lentas pueden desalentar a los desarrolladores de ejecutarlas con frecuencia.

12. Evitar Dependencias en el Entorno:

Las pruebas deben ser independientes del entorno en el que se ejecutan. Evita dependencias en bases de datos externas, sistemas de archivos, redes, etc., a menos que estés probando específicamente esas interacciones.

13. Revisar y Refactorizar las Pruebas:

Así como el código de producción, las pruebas unitarias deben ser revisadas y refactorizadas regularmente para mantener su calidad y relevancia.

14. Cobertura de Código:

Utiliza herramientas de cobertura de código para asegurar que un porcentaje significativo del código está cubierto por pruebas. Sin embargo, no te obsesiones solo con los números de cobertura; la calidad de las pruebas es más importante.

15. Documentar Pruebas Complejas:

Si una prueba es compleja, documenta su propósito y lógica para que otros desarrolladores puedan entenderla fácilmente.

9. ¿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));  
}  
}
```

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

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

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

10. Seleccione 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));  
  
    }  
}
```

Output:
[67, 7, 25]

[7, 25, 67]
[67, 7, 25]
[7, 25, 67]

11. Cuáles son las salidas del siguiente código. Elige todas las correctas

```
public class StringBuilders {  
  
    public static StringBuilder work (StringBuilder a,  
    StringBuilder 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);  
    }  
}
```

Output

s3 = null