1. ¿Cuál de los siguientes componentes es parte de una solicitud HTTP?
    a. Headers, scripts, funciones
    b. URL, cookies, archivos adjuntos
    c. URL, headers, cuerpo de la solicitud
    d. Encabezados de la respuesta, cuerpo de la respuesta, estado de la respuesta

2. public class LogicQuestion {
    public static void main(String[] args) {
    int count = 0; //inicia el count
    for (int i = 0; i < 10; i++) { //un bucle que va de 0 hasta 9
    if (i % 2 == 0) { // todos los numeros divisibles en 2 o pares
    count++; // aumenta el count cada vez
    }
    }
    System.out.println("Count: " + count); // Count + 5 ya que cuenta 0 2 4 6 8
    }
    }
    a. Compilation fails
    b. Count 5
    c. Count 4
    d. Count 6

3. Which method is used to sort elements of a List in natural order in Java? (*)
    a. Arrays.sort()
    b. Collections.order()
    c. Collections.sort()
    d. List.sort()

    Arrays.sort() se utiliza para arrays, no para listas.

    Collections.order() es incorrecto (no existe tal método).

    List.sort() es un método válido desde Java 8 y posteriores, pero requiere un Comparator (no usa el orden natural por defecto).

4. ¿Qué significa que un cambio en el software sea retro-compatible?
    a. El cambio corrige errores menores y realiza mejoras de rendimiento
    b. El cambio introduce nuevas funcionalidades que no afectan el comportamiento existente del software
    c. El cambio puede romper la funcionalidad existente, requiriendo ajustes en el código que depende de él
    d. El cambio garantiza que el software será funcional sin necesidad de modificaciones en el código que depende de él

5. class Base { // Se define una base con 2 constructores
```
public Base() { //Constructor vacio
System.out.println("Base constructor");
}
public Base(String message) { // constructor con parametro string
System.out.println("Base constructor with message: " + message);
}
}
class Derived extends Base {
public Derived() {
super("Hello");
System.out.println("Derived constructor");
}
}
public class Test {
public static void main(String[] args) {
Derived derived = new Derived();
}
}
```
    a. Base constructor
       Derived constructor
    b. Derived constructor
       Base constructor with message: Hello
    c. Compile error
    d. Base constructor with message: Hello
       Derived constructor
       Se ejecuta el constructor de Derived, lo que invoca primero el constructor de Base con el argumento "Hello".
       El constructor de Base con el mensaje "Hello" imprime:
       Base constructor with message: Hello

Después, el constructor de Derived continúa y imprime:

Derived constructor

6. ¿Cuál es la principal función de Jfrog Artifactory en un entorno de desarrollo de software?
    a. Proporcionar un entorno de desarrollo integrado (IDE) para aplicaciones Java

       //Este es eclipse
    b. Actuar como un servidor web para alojar sitios HTML y CSS

       //Este en resumen es tomcat
    c. Gestionar y almacenar artefactos de software, como dependencias y bibliotecas, de manera centralizada
    d. Ofrecer un sistema de control de versiones para proyectos de software

       //git / github

7. ¿Cuál de los siguientes métodos de Mockito se utiliza para verificar que un método de un mock ha sido llamado un número específico de veces? (*)

    a. assertCalled(mock, numberOfTimes)
    b. mock.verifyCall(n)
    c. verify() / verify(mock, times(n))
    d. mock.checkInvocations(count)

8. abstract class Animal {
   public abstract void makeSound();
   }
   class Dog extends Animal {
   @Override
   public void makeSound() {
   System.out.println("Bark!");
   }
   }
   public class Test {
   public static void main(String[] args) {
   Animal myDog = new Dog();

```
myDog.makeSound();
}
}
```

a. No se imprime nada
b. No compile
c. Exception al Ejecutarse
d. Bark!

9. ¿Cuál es el propósito principal de la anotación @Test en JUnit?
   a. Ejecutar un método antes de todos los métodos de prueba en una clase
   b. Ejecutar un método después de todos los métodos de prueba en una clase
   c. Ignorar un método de prueba
   d. Marcar un método como un método de prueba

10. ¿Cuál de las siguientes características es fundamental en una base de datos relacional?
    a. Almacenamiento de datos en un sistema de archivos distribuido
    b. Organización de datos en tablas con filas y columnas
    c. Almacenamiento de datos en formato JSON
    d. Utilización de nodos y relaciones para representar datos

11. Which line of code will compile successfully without any additional import statements?

```
public class Program{
public static void main(String[] args) {
// Line A
String str = "Hello World!"; //define e inicia un string esta clase ya esta en java
// Line B
ArrayList<String> list = new ArrayList<>(); // los arraylist se tienen que importar
// Line C
File file = new File("example.txt"); // import java.io.File;
// Line D
URL url = new URL("http://example.com"); // import java.net.URL;
}
}
```

    a. Line B
    b. Line C
    c. Line A
    d. Line D

12. abstract class Shape {
public abstract void draw();
public void printShape() {
System.out.println("This is a shape");
}
}
class Circle extends Shape {
@Override
public void draw() {
System.out.println("Drawing a circle");
}
}
public class Test {
public static void main(String[] args) {
Circle circle = new Circle();
circle.draw();
circle.printShape();
}
}

La clase Shape es una clase abstracta que define un contrato: cualquier clase que la herede debe implementar el método draw().
La clase Circle implementa el método draw() y hereda el método printShape(), que no necesita ser reimplementado.
La clase Test crea un objeto de tipo Circle, llama a los métodos draw() y printShape(), y muestra los mensajes correspondientes en la consola.ea un objeto de tipo `Circle`, llama a los métodos `draw()` y `printShape()`, y muestra los mensajes correspondientes en la consola.

a. No compile
b. This is a shape
   Drawing a circle
c. Drawing a circle
   This is a shape
d. Drawing a circle

13. ¿Cuál es el propósito principal del archivo pom.xml en un proyecto Maven?
a. Contener la documentación del código fuente del proyecto
b. Almacenar configuraciones de la base de datos del proyecto

c. Gestionar la interfaz gráfica de usuario del proyecto
d. Definir las dependencias, plugins y configuraciones del proyecto

Este archivo es fundamental en Maven, ya que especifica las dependencias externas que necesita el proyecto, los plugins que se utilizarán durante el ciclo de vida del proyecto, las configuraciones de compilación, las propiedades del proyecto y otros aspectos como las versiones y la estructura del proyecto.

14. Which section in the pom.xml file specifies the external libraries and dependencies required by the project?
    a. <plugins>
    b. <dependencies>
    c. <buid>
    d. <repositories>

<plugins>:
Esta sección se utiliza para definir los plugins que Maven usará durante el ciclo de vida del proyecto. Los plugins permiten realizar tareas específicas como compilar el código, empaquetar el proyecto, ejecutar pruebas, generar informes, entre otros.
<dependencies>:
En esta sección se definen las dependencias del proyecto. Es decir, las bibliotecas y archivos JAR externos que el proyecto necesita para funcionar correctamente. Cada dependencia se especifica con su grupo, artefacto y versión, y Maven se encarga de descargarlas desde los repositorios.
<build>:
Esta sección configura el proceso de compilación y empaquetado del proyecto. Incluye configuraciones sobre el directorio de salida, el tipo de empaquetado (por ejemplo, JAR, WAR), los plugins de compilación, entre otros. Aquí puedes personalizar cómo Maven construye tu proyecto.
<repositories>:
Aquí se definen los repositorios donde Maven buscará las dependencias que no estén disponibles en el repositorio central de Maven. Pueden ser repositorios internos o públicos adicionales donde se almacenan librerías y artefactos.

15. What will be the output of the following code snippet?

```
public class ScopeTest {
private int value = 10;

public void printValue() {
int value = 20;
System.out.println(this.value);
}
```

```
public static void main(String[] args) {
ScopeTest test = new ScopeTest();
test.printValue();
}
}
```

a. 10
b. Runtime error
c. 20
d. Compilation error

El uso de this.value se refiere a la variable de instancia value de la clase ScopeTest. El prefijo this se utiliza para hacer referencia explícita a la variable de la clase, ya que dentro del método printValue también hay una variable local llamada value. Debido a que this.value hace referencia a la variable de instancia de la clase, se imprimirá el valor de 10 (el valor que se asignó a la variable de instancia, no el valor de la variable local).

16. What is the primary purpose of a Data Transfer Object (DTO) in software design?

a. To encapsulate business logic within the application.
b. To provide a user interface for data input.
c. To transfer data between different layers or tiers of an application
d. To handle database operations directly.

El propósito principal de un Objeto de Transferencia de Datos (DTO) es mover datos entre procesos en el diseño de software. Los DTOs se utilizan comúnmente para transferir datos entre subsistemas o capas de la aplicación.

Beneficios de usar DTOs:
Desacopla la representación de los datos: Los DTOs separan la representación de los datos de la lógica de negocio o el esquema de la base de datos.
Reduce el número de llamadas a métodos: Los DTOs agrupan los datos en una única transferencia, lo que reduce el número de llamadas a métodos.
Facilita la comunicación entre sistemas: Los DTOs se pueden utilizar para comunicar dos sistemas, como una API y un servidor, sin exponer información sensible.

Reduce el tamaño de la carga útil: Los DTOs pueden omitir propiedades para reducir el tamaño de la carga útil.

Evita vulnerabilidades de sobreenvío (over-posting): Los DTOs pueden ayudar a evitar vulnerabilidades de sobreenvío de datos.

Características de los DTOs:

Los DTOs son objetos simples que no tienen ningún comportamiento, excepto para el almacenamiento, la recuperación, la serialización y la deserialización de sus propios datos.

17. Which declaration correctly initializes a boolean variable in Java?

a. boolean f = "true";
b. boolean f = () => f;
c. boolean f = (1 + 0);
d. boolean d = (a < b);
e. boolean b = 0 < 1;
f. boolean a = (10 > 5 && 2 < 3);

a esta definiendo un String, b eso no existe en java, c esta sumando 2 numeros, d esa si supieramos mas de a y b podria ser pero no, e si va a soltar un true, f tambien es valido

18. Which of the following statements about the 'throw' keyword is true?

a. It is used to declare that a method can throw an exception
b. It is used to define the cleanup code that must be executed
c. It is used to manually throw an exception
d. It is used to catch exceptions thrown by other methods

La palabra clave throw en Java se usa para lanzar una excepción explícitamente desde un método o un bloque de código. Normalmente se usa cuando deseas disparar una excepción en ciertas condiciones manualmente, como cuando se valida la entrada o se encuentra una condición de error que necesita ser reportada.

19. Which of the following code snippets will throw a ClassCastException

a. class A {}
class B extends A {}
public class Test {
public static void main(String[] args) {
B obj = new B();
A a = (A) obj;

```
}
}
b. class A {}
class B extends A {}
public class Test {
public static void main(String[] args) {
A obj = new B();
A a = (A) obj;
}
}
c. class A {}
class B extends A {}
public class Test {
public static void main(String[] args) {
A obj = new B();
B b = (B) obj;
}
}
d. class A {}
class B extends A {}
public class Test {
public static void main(String[] args) {
A obj = new A();
B b = (B) obj;
}
}
```

La excepción ClassCastException se lanza cuando intentas hacer un cast (convertir un objeto de un tipo a otro) a una clase que no es compatible con el objeto en cuestión. Fijarse bien en que tipo de objeto se esta creando. Se crea un objeto de tipo A, y luego se intenta hacer un casting a tipo B. Esto no es válido porque obj es una instancia de A, y A no es una subclase de B. El casting entre estos tipos no es posible, por lo tanto, se lanzará una ClassCastException. Este es el fragmento de código que sí lanzará una ClassCastException.

20. ¿Cuál de las siguientes afirmaciones es correcta sobre la aserción assert() en jUnit?
    a. Se utiliza para verificar que una colección contiene un elemento específico
    b. Se utiliza para verificar que dos objetos referencian la misma instancia
    c. Se utiliza para verificar que una condición es verdadera
    d. Se utiliza para verificar que dos valores son iguales

21. ¿Cual es la funcion principal del JDK (Java Development Kit)?
    a. Servir como un servidor web para aplicaciones Java.
    b. Proporcionar un entorno de ejecución para aplicaciones Java.
    c. Ofrecer herramientas necesarias para compilar, depurar y ejecutar aplicaciones Java.
    d. Ninguna opción es correcta.

22. Which of the following statements accurately describe the differences between Comparator and Comparable interfaces in Java?
    a. Comparator allows for multiple ways of comparing objects, while Comparable allows only one way of comparing objects.
    b. Comparable must be implemented by the class whose objects are being compared, whereas Comparator can be implemented by any class
    c. All of the above.
    d. Comparable defines the compareTo method, whereas Comparator defines the compare method.

    1. Comparator permite múltiples formas de comparar objetos, mientras que Comparable solo permite una forma de comparar objetos.
    Verdadero: La interfaz Comparable permite definir un solo orden natural para los objetos de una clase mediante la implementación del método compareTo(). En cambio, la interfaz Comparator permite definir múltiples formas (o estrategias) de comparar objetos implementando el método compare() en diferentes clases comparadoras.
    2. Comparable debe ser implementada por la clase cuyos objetos se van a comparar, mientras que Comparator puede ser implementada por cualquier clase.
    Verdadero: La interfaz Comparable debe ser implementada por la clase cuyos objetos se quieren comparar, ya que define el método compareTo(). Por otro lado, la interfaz Comparator puede ser implementada por cualquier clase externa, lo que significa que puedes definir una lógica de comparación personalizada sin modificar la clase original.
    3. Todos los anteriores.
    Verdadero: Dado que las dos afirmaciones anteriores son correctas, esta opción también lo es.

4. Comparable define el método compareTo(), mientras que Comparator define el método compare().

Verdadero: La interfaz Comparable define el método compareTo(), que se utiliza para comparar el objeto actual con otro objeto del mismo tipo. La interfaz Comparator define el método compare(), que compara dos objetos y devuelve un entero que indica su orden relativo.

Resumen:

Todos los anteriores es la opción correcta porque todas las afirmaciones anteriores sobre las interfaces Comparator y Comparable son precisas.

23. What is the purpose of the "throws" keyword in a method declaration in Java?
    a. To create a new exception instance.
    b. To catch exceptions thrown by other methods.
    c. To throw an exception within the method.
    d. To indicate the exceptions that the method can throw to the caller.

24. ¿Cuales de los siguientes comandos de Git se utilizan para gestionar ramas en un repositorio? (Seleccione todas las que correspondan).

    a. git commit
    b. git checkout, git branch, git merge
    c. git init
    d. git branch, git merge

    En el otro simulador hay una pregunta parecida pero preguntan por comandos que te permiten crear una rama en ese las respuestas serian git checkout -b <nombre de la rama> y git branch <nombre de la rama>

25. ¿Cual de los siguientes patrones de diseño es adecuado para crear una estructura de objetos en forma de arbol para representar jerarquias parte-todo, permitiendo a los clientes tratar objetos individuales y compuestos de manera uniforme?
    a. Patrón Estrategia (Strategy Pattern).
    b. Patrón Adaptador (Adapter Pattern).
    c. Patrón Compuesto (Composite Pattern).
    d. Patrón Fachada (Facade Pattern).

El Patrón Compuesto es adecuado para crear estructuras de objetos en forma de árbol que representan jerarquías parte-todo. Este patrón permite que tanto los objetos individuales (hojas) como los objetos compuestos (nodos) sean tratados de manera uniforme.

Con el Patrón Compuesto, se puede construir una jerarquía donde los objetos simples (como hojas) y los objetos complejos (como nodos que contienen otros objetos) implementan una interfaz común. Esto permite a los clientes trabajar de manera transparente con objetos individuales o compuestos sin tener que preocuparse por las diferencias entre ellos.

Patrón Estrategia (Strategy Pattern): Se utiliza para definir una familia de algoritmos, encapsulándolos y permitiendo que se puedan intercambiar en tiempo de ejecución. No está relacionado con la creación de jerarquías parte-todo.
Patrón Adaptador (Adapter Pattern): Se utiliza para convertir la interfaz de una clase en otra que los clientes esperan, permitiendo que clases incompatibles trabajen juntas. Tampoco está relacionado con jerarquías parte-todo.
Patrón Fachada (Facade Pattern): Proporciona una interfaz simplificada a un conjunto de interfaces en un subsistema, pero no se utiliza para crear jerarquías parte-todo.

26. Which file is used to configure user specific settings in Maven?
    a. user.xml
    b. settings.xml
    c. pom.xml
    d. build.xm

En Maven, el archivo settings.xml se utiliza para configurar ajustes específicos del usuario. Este archivo generalmente se encuentra en el directorio ~/.m2/ (para sistemas basados en UNIX) o en el directorio %USERPROFILE%\.m2\ (para sistemas Windows). Permite configurar preferencias específicas del usuario, como:

Ubicaciones de repositorios (por ejemplo, repositorios privados)

Credenciales de autenticación para repositorios

Configuración de proxy

Ubicación del repositorio local

Espejos (mirrors) para la descarga de artefactos

user.xml: Este archivo no es válido en Maven.

pom.xml: Este archivo se utiliza para configurar ajustes específicos del proyecto, como dependencias, plugins y configuración de construcción. No se utiliza para ajustes específicos del usuario.

build.xml: Este archivo es utilizado por Ant, no por Maven. Ant usa build.xml para la configuración de la construcción, mientras que Maven usa pom.xml.

27. What will be the output of the following code snippets?

```
import java.util.ArrayList;
import java.util.List; // estan importando librerías necesarias
public class GenericTest {
public static <T> void addIfAbsent(List<T> list, T element) { // definiendo un metodo generico, publico y estatico
if (!list.contains(element)) { // si no contiene un elemento
list.add(element); // añade al elemento
}
}
public static void main(String[] args) {
List<String> items = new ArrayList<>(); // esta definiendo una lista
items.add("apple"); //añade apple
items.add("banana"); // añade banana
addIfAbsent(items, "cherry"); añade cherry
addIfAbsent(items, "apple"); // este ya esta entonces no lo añade
System.out.println(items); // imprime
}
}
```

a. [apple, banana, cherry, apple]
b. [banana, cherry]
c. [apple, banana, cherry]
d. [apple, banana]

28. What will be the output of the following code snippet?

```
public class StringConcatenationTest {
public static void main(String[] args) {
String str1 = "Hello"; // define un string “Hello”
String str2 = "World"; // define un string “World”
String str3 = str1 + " " + str2; // concatena 2 strings
String str4 = str1.concat(" ").concat(str2);
String str5 = new StringBuilder().append(str1).append(" ").append(str2).toString();
System.out.println(str1.equals(str2) + " "); // false
System.out.println(str3.equals(str4) + " "); // true
```

System.out.println(str3 == str5 + " "); // false porque las referencias de los objetos son diferentes, aunque el contenido sea el mismo.
System.out.println(str4 == str5); //false porque las referencias de los objetos son diferentes, aunque ambos contienen "Hello World".
}
}

a. false false false false
b. true true false false
c. Compilation fails
d. false true false false

29. Which of the following code snippets will result in a compilation error when implementing the vehicle interface?
interface Vehicle{
void start();
void stop();
}

Aqui el truco esta en los modificadores de acceso, las interfaces sus metodos son public abstract osea que al implementarlos en otras clases tienen que ser public ya que no puedes quitarle visibilidad al metodo

a. public class Car implements Vehicle {public void start(){System.out.println("Car starts");}public void stop(){System.out.println("Car stops");}}
b. public class Bike implements Vehicle {
   public void start() { ... }
   void stop() { ... }
   }
c. public class Truck implements Vehicle {public void start(){System.out.println("Truck starts");}public void stop(){System.out.println("Truck stops");}public void load(){System.out.println("Truck loads");}}
d. public class Scooter implements Vehicle {public void start(){System.out.println("Scooter starts");}public void stop(){System.out.println("Scooter stops");}}

30. What will be the output of the following code snippet?
public class StaticNonStaticBlockTest {
static {
System.out.println("Static block");

```
}
{
System.out.println("Instance block");
}
public StaticNonStaticBlockTest() {
System.out.println("Constructor");
}
public static void staticMethod() {
System.out.println("Static method");
}
public static void main(String[] args) {
StaticNonStaticBlockTest test = new StaticNonStaticBlockTest();
new StaticNonStaticBlockTest();
}
}
```

a. Compilation error
b. Static block
   Instance block
   Constructor
   Instance block
   Constructor
c. Static method
   Static block
   Instance block
   Constructor
d. Static block
   Instance block
   Constructor
   Static method
   Carga de la clase:

   Bloque estático: Se ejecuta una vez al cargar la clase, imprimiendo "Static block".
   Primera instancia: Bloque de instancia: Se ejecuta antes del constructor, imprimiendo "Instance block".
   Constructor: Se ejecuta después del bloque de instancia, imprimiendo "Constructor".

Segunda instancia: Bloque de instancia: Se ejecuta antes del constructor, imprimiendo "Instance block".
Constructor: Se ejecuta después del bloque de instancia, imprimiendo "Constructor".

31. Which of the following is NOT part of the Agile Software development lifecycle?
    a. Planing
    b. Coding
    c. Documenting
    d. Testing

    Planificación: Ágil comienza con la planificación (por ejemplo, planificación de sprints) y continúa a lo largo del proyecto.
    Codificación: La codificación es esencial en Ágil, ya que el enfoque está en desarrollar y entregar software funcional.
    Pruebas: Las pruebas se integran durante todo el proceso de desarrollo para asegurar la calidad y que el producto cumpla con los requisitos.

32. ¿Cuál es la rama principal de Github en la que se integran las nuevas funcionalidades antes de lanzarlas a producción?
    a. develop
    b. analyze

    esto aunque depende de cada proyecto entre esas 2 seria develop, por lo regular son main/master -> develop/developer -> ramas personales u otras

33. Which method override is valid given the following classes?

```
class Parent {
void display() {
System.out.println("Parent");
}
}
class Child extends Parent {
// Override here
}
```

    a. public void display(){
       System.out.println(“Child”);}
    b. static void display(){
       System.out.println(“Child”);}
    c. private void display(){
       System.out.println(“Child”);}
    d. void display(){
       System.out.println(“Child”);}

Para que una clase hija (Child) pueda sobrescribir este método, debe cumplir con algunas reglas de la sobrecarga (override):

El nombre del método debe ser el mismo.
El tipo de retorno debe ser el mismo o un subtipo del tipo de retorno en el método de la clase padre.
El modificador de acceso del método sobrescrito debe ser al menos tan accesible como el del método en la clase padre.
No se puede sobrescribir un método como static si en la clase padre no es static.

34. ¿Cuál de las siguientes afirmaciones describe mejor un Step en el contexto de Spring Batch?
    a. Una interfaz que define los métodos para realizar operaciones CRUD en los datos del trabajo por lotes.
    b. Un componente que se encarga exclusivamente de la validación de datos en un trabajo por lotes.
    c. Una clase que gestiona la configuración de la base de datos utilizada por un trabajo por lotes.
    d. Un objeto de dominio que encapsula una fase independiente y secuencial de un trabajo por lotes.

    "Una interfaz que define los métodos para realizar operaciones CRUD en los datos del trabajo por lotes."

    Incorrecto. Aunque Spring Batch puede involucrar la manipulación de datos, un Step no se refiere específicamente a operaciones CRUD (Crear, Leer, Actualizar, Eliminar). Los Steps se centran en fases del trabajo por lotes, no en la implementación directa de CRUD.
    "Un componente que se encarga exclusivamente de la validación de datos en un trabajo por lotes."

    Incorrecto. Un Step puede implicar validación de datos, pero su responsabilidad no se limita solo a la validación. Los Steps abarcan toda una fase del trabajo por lotes, y dentro de esa fase se pueden incluir validaciones, transformaciones, procesamiento, etc.
    "Una clase que gestiona la configuración de la base de datos utilizada por un trabajo por lotes."

Incorrecto. Esto se refiere más a la configuración de la infraestructura del trabajo por lotes (por ejemplo, configuraciones de la base de datos) y no es el propósito de un Step. Un Step se enfoca en la lógica y el flujo de procesamiento de datos dentro del trabajo.

35. Todo el acceso a los datos debe estar encapsulado en una biblioteca para facilitar la reutilización y control de acceso.

a. Falso
b. Verdadero

36. Which of the following statements is true about arrays in Java?

a. An array is mutable
b. An array is inmutable
c. An array allows multiple dimensions
d. An array has a fixed size

Aqui hay mas de una respuesta correcta aunque si solo nos piden una seria la d, si nos dejan poner mas de una seria d c a

37. Where do you configure the plugins used for various build tasks in Maven's pom.xml?

a. <build>
b. <plugins>
c. <dependencies>
d. <repositories>

En el archivo pom.xml de Maven, los plugins se configuran dentro de la sección <build>. Más específicamente, dentro de la etiqueta <build>, se utiliza la etiqueta <plugins> para definir los plugins que se van a usar para diversas tareas de construcción del proyecto, como compilar el código, ejecutar pruebas, empaquetar el proyecto, etc. tal vez deberia marcar build y plugins

38. En el contexto de bases de datos relacionales, si una transacción cumple con la propiedad de Aislamiento (Isolation) del principio ACID, esto significa que:

a. Las transacciones se ejecutan como si fueran la única operación en el sistema, sin interferencia de otras transacciones concurrentes
b. Una transacción debe mantener la consistencia de la base de datos antes y después de su ejecución
c. Los cambios realizados por una transacción son permanentes y sobreviven a fallos del sistema

d. Las operaciones de una transacción se ejecutan todas o ninguna, no hay estados intermedios

Resumen de las propiedades de ACID:

Propiedad	Descripción
Atomicidad	La transacción es "todo o nada", es decir, se ejecuta completamente o no se ejecuta.
Consistencia	La base de datos pasa de un estado válido a otro estado válido, respetando todas las reglas de integridad.
Aislamiento	Las transacciones concurrentes no deben interferir entre sí, y los efectos de una transacción no deben ser visibles para otras hasta que se complete.
Durabilidad	Los cambios realizados por una transacción confirmada son permanentes, incluso si ocurre un fallo en el sistema.

39. ¿Cuál es el comando en Bash para cambiar el directorio actual a uno especificado?

a. sh
b. cd
c. chdir
d. move

40. En el contexto de Maven, ¿Cuál es la función principal del archivo settings.xml?

a. Especificar las dependencias del proyecto.
b. Generar informes de construcción y documentación.
c. Configurar la información del repositorio local y remoto, así como las credenciales y perfiles de usuario.
d. Definir la estructura de directorios del proyecto.

Especificar las dependencias del proyecto: Esto se hace en el archivo pom.xml, no en el settings.xml.

Generar informes de construcción y documentación: También se maneja dentro de pom.xml a través de plugins específicos, no en settings.xml.

Definir la estructura de directorios del proyecto: Esto también se especifica en pom.xml y no en settings.xml.

41. ¿Cuál es la principal desventaja del antipatrón "contenedor mágico" en el desarrollo de software?

a. Introduce dependencias circulares que son difíciles de resolver.
b. Utiliza un número excesivo de patrones de diseño, complicando la estructura del código.
c. Oculta demasiada lógica de negocio en un contenedor genérico, lo que hace que el código sea difícil de entender y depurar.
d. Depende en gran medida de servicios externos, lo que reduce la portabilidad del software.

El antipatrón "contenedor mágico" se refiere a una situación en la que se crea un contenedor o framework que asume demasiadas responsabilidades, como gestionar dependencias, controlar flujos de trabajo o procesar datos, pero lo hace de una manera tan genérica que oculta la lógica de negocio real que debería ser explícita y clara en el código. Esto puede generar varias complicaciones:

Difícil de entender: El código se vuelve menos transparente porque gran parte de la lógica importante queda oculta en el contenedor, lo que dificulta la comprensión del flujo del programa.

Difícil de depurar: Como el contenedor maneja muchos aspectos del comportamiento, puede ser complicado rastrear y depurar errores, ya que el flujo de ejecución no está claramente definido en el código fuente.

Falta de flexibilidad: Si el contenedor es demasiado genérico, puede ser difícil modificar o extender su comportamiento sin romper la funcionalidad existente.

Las otras opciones:

Introduce dependencias circulares que son difíciles de resolver: Este problema no es una característica común del antipatrón "contenedor mágico", aunque podría ocurrir si el contenedor está mal diseñado. Sin embargo, no es la principal desventaja.

Utiliza un número excesivo de patrones de diseño, complicando la estructura del código: Aunque el uso excesivo de patrones de diseño puede ser una mala práctica, no es lo que caracteriza específicamente al "contenedor mágico". Este antipatrón se centra más en la ocultación de la lógica de negocio.

Depende en gran medida de servicios externos, lo que reduce la portabilidad del software: Esta no es una desventaja directa del antipatrón "contenedor mágico". Los problemas de portabilidad no son necesariamente una característica del uso de un contenedor, aunque puede haber otros antipatrónes que estén relacionados con la dependencia de servicios externos.

42. What will be the result of the following code execution?

```
import java.util.ArrayList;
import java.util.List;

public class ArrayListTest {
public static void main(String[] args) {
List<Integer> list = new ArrayList<>();
list.add(1);
list.add(2);
list.add(3);
list.remove(1); // Esta quitando el elemento del array 1 no el
numero que le pusimos
System.out.println(list);
}
```

}

a. [1, 3]

b. 1 2

43. Which of the following statements accurately describe the relationships that can exist between classes in Java?

a. Both inheritance and composition can be used together to model complex relationships.
b. Inheritance represents an "is-a" relationship where one class derives from another class.
c. Composition represents a "has-a" relationship where one class contains an instance of another class.
d. Composition should be preferred over inheritance to promote code reuse and flexibility.
e. Usage (or association) represents a "uses-a" relationship where one class uses methods or instances of another class.
f. Inheritance should be preferred over composition to promote code reuse and flexibility

a. An array has a fixed size

b. a, b, c ,d, e

c. Documenting

d. All of the above.

44. Which of the following methods can be used to remove all elements from an ArrayList?

a. deleteAll()

b. eraseAll()

c. removeAll()

d. clear()

deleteAll(): Este método no existe en la clase ArrayList en Java.
eraseAll(): Este método tampoco existe en la clase ArrayList.
removeAll(): Aunque removeAll() es un método de la interfaz Collection (que ArrayList implementa), se usa para eliminar todos los elementos que están

contenidos en una colección específica, no todos los elementos de la lista en sí. Requiere un argumento (una colección) para eliminar los elementos que coincidan.