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

**Investigación.**

Al comando git revert se le puede considerar un comando para "deshacer", pero lo cierto es que no lo es en el sentido tradicional. En lugar de eliminar la confirmación del historial del proyecto, resuelve cómo invertir los cambios introducidos por la confirmación y añade una nueva con el contenido inverso resultante. Así, se evita que Git pierda el historial, lo cual resulta importante para la integridad del historial de revisiones y para que la colaboración sea fiable.

Deberías usar la reversión cuando desees aplicar lo contrario a una confirmación del historial del proyecto. Esto puede ser útil, por ejemplo, si estás haciendo el seguimiento de un error y descubres que se introdujo mediante una sola confirmación. En vez de entrar, corregirlo y confirmar una nueva instantánea manualmente, puedes usar git revert para que todo este proceso se lleve a cabo de forma automática.

El comando git revert sirve para deshacer cambios efectuados en el historial de confirmaciones de un repositorio. Otros comandos para "deshacer" como, por ejemplo, git checkout y git reset, mueven los punteros de referencia HEAD y de la rama a una confirmación especificada. El comando git revert también toma una confirmación especificada, pero git revert no mueve los punteros de referencia a esta confirmación.

Respuestas en AZO

1. git reset
2. git revert 🡨 Es correcto
3. git amend
4. git checkout
5. ¿Cuál es la diferencia entre una clase abstracta y una interfaz en java 8?

**INVESTIGACIÓN**

Una clase abstracta puede tener métodos concretos y abstractos, mientras que una interfaz solo puede tener métodos abstractos (a menos que se introduzcan métodos default o static).

Una clase solo puede extender una clase abstracta, pero puede implementar múltiples interfaces.

Las variables en una clase abstracta pueden ser de cualquier tipo (públicas, protegidas, privadas, estáticas, finales, etc.), mientras que en una interfaz, las variables son public, static y final por defecto.

Respuestas en AZO

1. Una clase abstracta puede contener implementaciones de métodos, mientras que una interfaz no puede. 🡨 Esta es la que, aunque es incorrecta sale en varios lugares como correcta.
2. Una clase abstracta puede contener variables de instancia, mientras que una interfaz no puede 🡨Si no sale la de abajo, esta sería las que se acerca más.
3. Una interfaz puede contener implementaciones de métodos, mientras que una clase abstracta no puede.
4. Una interfaz solo puede heredar de una clase, mientras que una clase abstracta puede heredar… (no se ve)
5. Una interfaz puede ser implementada por múltiples clases mientras que una clase abstracta solo puede ser subclasificada. 🡨Si esta opción sale, es correcta
6. 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? (Elije todas las correctas)

**Investigación.**

Las declaraciones que se deben usar para contar la cantidad de monedas de 5 centavos en una matriz de cadenas de varias monedas son la asignación (c) y la iteración (d).

La asignación se utiliza para inicializar una variable que llevará el conteo de las monedas de 5 centavos encontradas, mientras que la iteración se utiliza para recorrer la matriz y contar las monedas de 5 centavos presentes en ella.

a) Conditional (Duda Preguntar)

b) Assertion

c) Assignment Correcta

d) Iteration  Correcta

1. ¿Qué es un archivo JAR en java?

**Investigación.**

Un archivo JAR (Java ARchive) en Java es un archivo que contiene clases compiladas y otros recursos (como imágenes, archivos de configuración, etc.) empaquetados juntos en un solo archivo. Esto facilita la distribución y el despliegue de aplicaciones Java, ya que todas las clases y recursos necesarios están contenidos en un solo archivo.

1. Un archivo que contiene un archivo de configuración Maven
2. Un archivo que contiene un archivo de configuración Git.
3. Un archivo que contiene una clase Java compilada 🡨Correcta
4. Un archivo que contiene un archivo de configuración de Spring
5. ¿Qué es la sobrecarga de métodos en Java?

**Investigación.**

La sobrecarga de métodos en Java es la capacidad de definir múltiples métodos con el mismo nombre en una clase, pero con diferentes listas de parámetros (diferentes tipos de parámetros, diferente número de parámetros o ambos). El compilador de Java distingue estos métodos sobrecargados por sus firmas (nombre del método y lista de parámetros).

1. Cuando un método tiene múltiples definiciones con el mismo nombre y tipo de parámetros.
2. Cuando un método tiene múltiples definiciones con diferentes nombres y cantidades de parámetros.
3. Cuando un método tiene múltiples definiciones con diferentes tipos de cantidades de parámetros. 🡨Correcta
4. Cuando un método tiene múltiples definiciones con diferentes nombres y tipos de parámetros.
5. ¿Cuál es la diferencia entre un ArrayList y un LinkedList en Java?

**Investigación.**

La principal diferencia es la estructura de datos subyacente.

Un ArrayList está respaldado por un array redimensionable, lo que lo hace eficiente para acceder a elementos por índice, pero menos eficiente para insertar o eliminar elementos en posiciones arbitrarias.

Un LinkedList está respaldado por una estructura de datos de lista doblemente enlazada, lo que lo hace más eficiente para insertar o eliminar elementos en posiciones arbitrarias, pero menos eficiente para acceder a elementos por índice.

1. ArrayList es más rápido que un LinkedList para agregar y eliminar elementos.
2. ArrayList es más eficiente en el uso de memoria que LinkedList.
3. LinkedList es una clase abstracta mientras que ArrayList es una clase concreta.
4. LinkedList es más rápido que ArrayList para acceder a elementos aleatorios.
5. ¿Cuándo se debe usar un bloque finally en una declaración try regular (no una prueba con recursos)?

**Investigación.**

Un bloque finally en una declaración try regular (no una prueba con recursos) se debe usar cuando se necesita ejecutar código de limpieza o liberación de recursos, independientemente de si se produce una excepción o no. El bloque finally se ejecutará siempre, ya sea que se haya producido una excepción o no, y se puede utilizar para cerrar archivos, liberar recursos, restaurar el estado del programa, etc.

1. Cuando no hay bloques catch en una declaración try.
2. Nunca.
3. Cuando hay dos o más bloques catch en una sentencia try.
4. Cuando hay exactamente un bloque catch en una sentencia try.
5. Cuando el código del programa no termina por sí solo.
6. ¿Cuál es el propósito principal de los test unitarios?

**Investigación.**

Verificar que las unidades individuales de código (como métodos o clases) funcionen correctamente y produzcan los resultados esperados.

Facilitar el desarrollo y el mantenimiento del código, ya que las pruebas unitarias actúan como documentación ejecutable y permiten detectar errores de forma temprana.

Mejorar la calidad y la confiabilidad del código, al garantizar que cada unidad funcione correctamente antes de integrarla con otras partes del sistema.

Permitir refactorizaciones más seguras, ya que las pruebas unitarias pueden detectar rápidamente si se introduce un error durante el proceso de refactorización.

El propósito principal de los test unitarios es asegurar la calidad del software (d). Estos tests se centran en probar unidades individuales de código, como funciones o métodos, para verificar que funcionan como se espera y para detectar posibles errores o bugs. Si bien pueden contribuir a mejorar la eficiencia del desarrollo al identificar problemas temprano, su objetivo principal es garantizar que el software funcione correctamente y cumpla con los requisitos especificados.

1. Comprobar la eficiencia del hardware.
2. Medir la velocidad de la aplicación.
3. Ahorrar tiempo en el desarrollo.
4. Asegurar la calidad del software.
5. Selecciona la respuesta correcta con respecto al resultado del bloque de código.

Texto

Descripción generada automáticamente

1. cad1 diferente a cad2

cad1 es igual a cad3

1. ca1 es igual a cad2

ca1 es igual a cad3

c) No compila

d) cad1 diferente a cad2

cad1 diferente a cad3

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

cTexto

Descripción generada automáticamente

1. Mammal.
2. MammalPlatypus.
3. El código no se compila en la línea 11.
4. El código no compila en la línea 8
5. ¿Cómo se manejan las excepciones en java?

**Investigación.**

Try-Catch: En un bloque try, se coloca el código que puede lanzar una excepción. Si ocurre una excepción dentro de este bloque, Java busca un bloque catch correspondiente que pueda manejar esa excepción. Si encuentra un bloque catch que coincide con el tipo de excepción, se ejecuta el código dentro de ese bloque catch para manejar la excepción.

Throw: La palabra clave throw se usa para lanzar explícitamente una excepción en un punto específico del código. Esto puede ser útil cuando detectas una condición que justifica una excepción, por ejemplo, validar una entrada de usuario y lanzar una excepción si no cumple con ciertos requisitos.

Finally (Opcional): Puedes usar un bloque finally opcional que se ejecutará siempre, independientemente de si se lanza una excepción o no. Esto se utiliza comúnmente para liberar recursos que estén en uso, como cerrar conexiones de base de datos o archivos.

En resumen, se manejan las excepciones en Java utilizando bloques try-catch para capturar y manejar excepciones específicas y throw para lanzar excepciones cuando sea necesario.

1. Con la instrucción try-catch.
2. Con la instrucción if-else.
3. Con la instrucción for.
4. Con la instrucción while.
5. ¿La anotación @Ignore es usada para omitir un test por lo que no se ejecuta?

**Investigación.**la anotación @Ignore en JUnit (o en otras herramientas de pruebas unitarias en Java) se usa para omitir la ejecución de un test específico. Esto es útil cuando deseas evitar que un test se ejecute temporalmente sin eliminarlo del código, por ejemplo, cuando estás trabajando en una parte del código y temporalmente no deseas que ciertas pruebas se ejecuten. Entonces, al marcar un test con @Ignore, indicas que no deseas que se ejecute hasta que lo actives nuevamente, generalmente quitando la anotación @Ignore.

1. Verdadero
2. Falso

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

Texto

Descripción generada automáticamente

1. Error de compilación en la línea 7, x no se inicializa.
2. -1
3. -2
4. 0
5. ¿Qué es un operador de short circuit?

**Investigación.**

Los operadores de cortocircuito en Java son los operadores lógicos && (AND condicional) y || (OR condicional). Estos operadores permiten que la evaluación de una expresión lógica se detenga tan pronto como se conozca el resultado final, evitando evaluar los operandos restantes.

En el caso del operador &&, si el operando de la izquierda es falso, el resultado completo de la expresión será falso, independientemente del valor del operando de la derecha. Por lo tanto, el operando de la derecha no se evalúa.

En el caso del operador ||, si el operando de la izquierda es verdadero, el resultado completo de la expresión será verdadero, independientemente del valor del operando de la derecha. Por lo tanto, el operando de la derecha no se evalúa.

Esta característica de cortocircuito puede ser útil para evitar la evaluación de expresiones innecesarias y, en algunos casos, prevenir efectos secundarios indeseados.

1. Sirve para realizar más eficientes las operaciones condicionales evitando ejecutar operaciones si estas ya no son necesarias.
2. Operador que nos sirve para crear una nueva clase anónima.
3. Sirve para lanzar una excepción personalizada.
4. Es un patrón de arquitectura de microservicios que nos permite evitar el consumo de servicios que están en mantenimiento.
5. ¿Qué es el patrón de diseño DAO y cómo se implementa en Java?

**Investigación.**

El patrón de diseño DAO (Data Access Object) es un patrón arquitectónico que proporciona una capa de abstracción entre la capa de lógica de negocio y la capa de persistencia de datos (como una base de datos). El objetivo principal del patrón DAO es separar la lógica de acceso a datos de la lógica de negocio, promoviendo un código más modular y mantenible.

En Java, el patrón DAO se implementa creando una interfaz DAO y una o más clases concretas que implementan esta interfaz. La interfaz DAO define los métodos necesarios para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la fuente de datos.

La capa de lógica de negocio interactuaría con la implementación concreta del DAO para realizar operaciones en la fuente de datos, en lugar de interactuar directamente con la fuente de datos.

1. 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.
2. 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.
3. 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.
4. 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.
5. ¿Qué es un endpoint en una API REST?

**Investigación.**

Un endpoint en una API REST (Representational State Transfer) es una URL única que representa un objeto o colección de objetos. Los endpoints definen las operaciones que se pueden realizar en los recursos a través de los métodos HTTP (GET, POST, PUT, DELETE, etc.).

Los endpoints siguen una estructura jerárquica y suelen estar formados por:

La ruta base de la API (por ejemplo, http://ejemplo.com/api/v1)

El nombre del recurso (por ejemplo, /usuarios)

Identificadores de recursos específicos (por ejemplo, /usuarios/1)

Algunos ejemplos de endpoints en una API REST podrían ser:

GET /usuarios (obtener una lista de usuarios)

POST /usuarios (crear un nuevo usuario)

GET /usuarios/1 (obtener detalles del usuario con ID 1)

PUT /usuarios/1 (actualizar el usuario con ID 1)

DELETE /usuarios/1 (eliminar el usuario con ID 1)

Cada endpoint está asociado a una operación específica en un recurso y, en función del método HTTP utilizado, se realizará la acción correspondiente (crear, leer, actualizar o eliminar).

Los endpoints son fundamentales en el diseño de una API REST, ya que definen la interfaz a través de la cual los clientes (aplicaciones, servicios, etc.) interactuarán con los recursos y los datos subyacentes.

Estas explicaciones son relevantes para Java 8, ya que estos conceptos son aplicables independientemente de la versión específica de Java utilizada. Sin embargo, es importante tener en cuenta que Java 8 introdujo algunas características nuevas, como lambdas y streams, que pueden ser útiles para implementar patrones de diseño como DAO y desarrollar APIs REST de manera más concisa y expresiva.

1. Un endpoint es la URL que se utiliza para acceder a una API REST.
2. Un endpoint es un método que se utiliza para procesar datos en una API REST.
3. Un endpoint es un controlador que se utiliza para administrar una API REST.
4. Un endpoint es un objeto que se utiliza para almacenar datos en una API REST.
5. ¿Qué hace el siguiente programa?

Texto

Descripción generada automáticamente

1. El programa no compila.
2. Cuenta las letras que hay, en una palabra.
3. Verifica si una palabra es un palíndromo.
4. ¿Cuál de las siguientes opciones son verdaderas? (elija todas las correctas)
5. Java es un lenguaje orientado a objetos.
6. El código Java compilado en Windows puede ejecutarse en Linux.
7. Java permite la sobrecarga de operadores
8. Java es un lenguaje de programación funcional. 🡨 No sabemos
9. Java es un lenguaje procedimental. 🡨 No sabemos
10. Java tiene punteros a ubicaciones específicas en la memoria.
11. ¿Qué es Maven y para qué se utiliza en el desarrollo de aplicaciones?

**Investigación.**

Maven es una herramienta de gestión y construcción de proyectos de software, ampliamente utilizada en el desarrollo de aplicaciones Java. Proporciona un enfoque estandarizado y estructurado para la construcción, gestión de dependencias, pruebas, empaquetado, distribución y documentación de proyectos.

Las principales características y usos de Maven son:

Gestión de dependencias: Maven simplifica la gestión de dependencias de bibliotecas y frameworks externos que utiliza un proyecto. Estas dependencias se declaran en un archivo central llamado pom.xml (Project Object Model) y Maven se encarga de descargar automáticamente las dependencias necesarias desde repositorios centralizados.

Construcción de proyectos: Maven proporciona un ciclo de vida de construcción bien definido que incluye fases como compilación, pruebas, empaquetado, instalación y despliegue. Esto ayuda a estandarizar y automatizar el proceso de construcción de proyectos.

Reutilización y centralización de información: Maven promueve la reutilización de artefactos y proyectos a través de repositorios centralizados. También centraliza la información de configuración del proyecto en el archivo pom.xml, lo que facilita la gestión y el mantenimiento del proyecto.

Informes y documentación: Maven puede generar informes y documentación a partir del código fuente y las configuraciones del proyecto, incluyendo informes de pruebas, cobertura de código, análisis estático, y documentación de la API.

Plugins: Maven tiene una arquitectura basada en plugins que permiten extender su funcionalidad. Existen numerosos plugins para tareas como compilación, empaquetado, despliegue, análisis de código, generación de informes, entre otros.

Proyectos multi-módulo: Maven facilita la gestión de proyectos complejos que se dividen en varios módulos o subproyectos, lo que permite una mejor organización y reutilización de código.

En el desarrollo de aplicaciones Java, Maven se utiliza ampliamente por varios motivos:

Automatización de tareas: Maven automatiza tareas repetitivas como compilación, empaquetado, pruebas y despliegue, lo que ahorra tiempo y esfuerzo.

Gestión de dependencias: La gestión centralizada de dependencias facilita la incorporación y actualización de bibliotecas externas en el proyecto.

Estandarización de proyectos: Maven impone una estructura de directorios y un ciclo de vida de construcción estandarizados, lo que facilita el trabajo en equipo y la comprensión del proyecto.

Integración con herramientas: Maven se integra fácilmente con entornos de desarrollo (IDEs), servidores de integración continua (CI/CD) y herramientas de calidad de código.

Si bien Maven se originó para proyectos Java, también se ha extendido su uso a otros lenguajes de programación como C#, Ruby, Scala y Python, mediante la creación de plugins específicos.

En resumen, Maven es una herramienta esencial en el desarrollo de aplicaciones Java modernas, ya que simplifica y estandariza muchas tareas relacionadas con la construcción, gestión y distribución de proyectos, lo que permite a los desarrolladores concentrarse más en la lógica de negocio y menos en tareas repetitivas y engorrosas.

1. Maven es un lenguaje de programación. Se utiliza en el desarrollo de aplicaciones Java para escribir código.
2. Maven es un servidor de base de datos. Se utiliza en el desarrollo de aplicaciones java para alojar datos.
3. Maven es un sistema de control de versiones. Se utiliza en el desarrollo de aplicaciones java.
4. Maven es una herramienta de gestión de dependencias. Se utiliza en el desarrollo de aplicaciones en el proyecto.
5. ¿Cuál de lo siguiente es cierto? (elija todas las correctas)
6. javac compila un archivo .java en un archivo .bytecode. 🡨 Si no tuviera el punto sería correcta
7. Java toma el nombre del archivo .bytecode como parámetro.
8. javac compila un archivo .java en un archivo .class
9. Java toma el nombre de la clase como parámetro.
10. Java toma el nombre del archivo .class como parámetro.
11. javac compila un archivo .class como archivo java.
12. ¿Qué es Git y cuáles son algunos de sus comandos básicos?

**Investigación.**

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. Permite a los desarrolladores colaborar en proyectos de forma eficiente, mantener un historial de versiones, realizar seguimiento de cambios, revertir modificaciones, y gestionar ramas para el desarrollo paralelo de funcionalidades.

Comandos básicos de Git:

1. **git init**: Inicializa un nuevo repositorio Git en un directorio.
2. **git clone <URL>**: Clona un repositorio remoto en tu máquina local.
3. **git add <archivo>**: Agrega cambios de un archivo al área de preparación (staging area) para ser incluidos en el próximo commit.
4. **git commit -m "Mensaje del commit"**: Crea un nuevo commit con los cambios en el área de preparación, acompañado de un mensaje descriptivo.
5. **git status**: Muestra el estado actual del repositorio, incluyendo archivos modificados, en el área de preparación y sin seguimiento.
6. **git diff**: Muestra las diferencias entre los cambios en el directorio de trabajo y el área de preparación.
7. **git log**: Muestra el historial de commits del repositorio.
8. **git push**: Sube los commits locales al repositorio remoto.
9. **git pull**: Descarga los cambios del repositorio remoto y los fusiona con el repositorio local.
10. **git branch**: Muestra las ramas existentes en el repositorio.
11. **git checkout <nombre-rama>**: Cambia a una rama específica.
12. **git merge <nombre-rama>**: Fusiona una rama con la rama actual.
13. **git remote -v**: Muestra las URL de los repositorios remotos vinculados.
14. **git fetch**: Descarga los cambios del repositorio remoto sin fusionarlos con el repositorio local.
15. **git reset --hard**: Reinicia el estado del repositorio al último commit, eliminando todos los cambios posteriores.

Estos son solo algunos de los comandos básicos de Git. Hay muchos más comandos y opciones que se pueden utilizar para gestionar de manera efectiva el control de versiones en un proyecto de software.

1. Git es un lenguaje de programación. Algunos comandos básicos de Git incluyen “print” e “if-else”.
2. Git es una herramienta para el análisis de código. Algunos comandos básicos de Git incluyen “analyze”… (no se ve).
3. Git es un sistema de control de versiones. Algunos comandos básicos de Git incluyen “commit” y “push”.
4. Git es una herramienta para realizar pruebas de software. Algunos comandos básicos de Git incluyen… no se ve, pero no es correcta.
5. Dados los siguientes segmentos de código, ¿Qué respuesta no es una implementación de java válida?

**Investigación.**

Esta declaración no es válida porque al intentar sumar un valor entero (variableA) y un valor de punto flotante (variableB), y asignar el resultado a una variable entera (variableC), se produce una pérdida de precisión debido a la conversión de tipos.

En Java, cuando se realiza una operación aritmética entre un valor entero y un valor de punto flotante, el valor entero se convierte automáticamente a punto flotante antes de realizar la operación. Sin embargo, el resultado de esta operación es un valor de punto flotante.

Para asignar el resultado a una variable entera (int), se debe realizar un casting explícito, lo cual truncará la parte decimal del valor de punto flotante. Pero este truncamiento puede llevar a una pérdida de precisión o incluso a un desbordamiento aritmético si el resultado excede el rango de la representación entera.

1. int variableA = 10;  
   float variableB = 10.5f;  
   int variableC = variableA + variableB;
2. byte variableA = 10;  
   double variableB = 10.5f;  
   double variableC = variableA + variableB;
3. byte variableA = 10;  
   float variableB = 10.5f;  
   float variableC = variableA + variableB;
4. ¿Qué escenario es el mejor uso de una excepción?

**Investigación.**

El mejor uso de una excepción es en el escenario (c) cuando no se encuentra un elemento al buscar en una lista. Las excepciones se utilizan para manejar situaciones excepcionales o errores que pueden ocurrir durante la ejecución de un programa y que no se pueden manejar de manera normal. Cuando se busca un elemento en una lista y no se encuentra, esto se considera una situación excepcional que se puede manejar mediante una excepción.

1. La computadora se incendió.
2. No sabe cómo codificar un método.
3. No se encuentra un elemento al buscar en una lista.
4. Se pasa un parámetro inesperado a un método.
5. Quiere recorrer una lista.
6. ¿Qué es un bean en Spring?

**Investigación.**

En Spring, un bean es un objeto que se crea, configura e inicializa mediante el contenedor de Spring. El contenedor de Spring es el núcleo de Spring Framework y es responsable de administrar el ciclo de vida de los beans, así como de inyectar dependencias en ellos.

Los beans en Spring pueden ser clases Java regulares, pero también pueden implementar interfaces específicas de Spring o extender clases base de Spring. Estos beans se definen en el archivo de configuración de Spring (generalmente un archivo XML o mediante anotaciones en el código Java).

Algunas características importantes de los beans en Spring:

1. \*\*Inversión de Control (IoC)\*\*: Spring utiliza el principio de Inversión de Control, donde el contenedor de Spring es responsable de crear y administrar los objetos (beans) en lugar de que las clases los creen directamente.

2. \*\*Inyección de Dependencias (DI)\*\*: Spring puede inyectar automáticamente las dependencias necesarias en los beans. Esto se logra a través de la configuración en el archivo de configuración de Spring o mediante anotaciones en el código.

3. \*\*Ámbito (Scope)\*\*: Spring permite definir el ámbito de los beans, que determina el ciclo de vida y el número de instancias que se crearán. Los ámbitos comunes incluyen singleton (una sola instancia para toda la aplicación), prototype (una nueva instancia cada vez que se solicita el bean), request (una instancia por solicitud HTTP), session (una instancia por sesión HTTP), entre otros.

4. \*\*Ciclo de vida del bean\*\*: Spring proporciona una forma de controlar el ciclo de vida de los beans, permitiendo la ejecución de código personalizado durante la inicialización y la destrucción de los beans.

5. \*\*Configuración flexible\*\*: Los beans en Spring se pueden configurar de varias maneras, ya sea mediante archivos XML, anotaciones en el código Java o incluso mediante programación en Java.

Los beans en Spring son fundamentales para el desarrollo de aplicaciones empresariales, ya que promueven la separación de preocupaciones, la modularidad y la reutilización de código. Además, Spring proporciona una amplia gama de beans preconstruidos para tareas comunes, como acceso a bases de datos, transacciones, procesamiento de mensajes, gestión de seguridad, entre otros.

En resumen, un bean en Spring es un objeto gestionado por el contenedor de Spring, que se crea, configura e inicializa automáticamente. Los beans son la unidad básica de construcción en Spring Framework y proporcionan una forma flexible y modular de desarrollar aplicaciones empresariales.

1. Un archivo de configuración XML que se utiliza para definir la estructura de una tabla de base de datos.
2. Una instancia de una clase que se administra por el contendedor de Spring.
3. Una herramienta de inyección de dependencias que se utiliza para inyectar dependencias en una clase.
4. Una clase que se utiliza para configurar la conexión a una base de datos.
5. Selecciona la respuesta correcta con respecto al resultado del bloque de código

Texto

Descripción generada automáticamente

1. c,s,t
2. t,s,c
3. Error en tiempo de ejecución
4. No compila

**NINGUNA ES CORRECTA**

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

**Investigación.**

1. Si un método toma una superclase de 3 objetos, cualquiera de esas clases puede pasarse como parámetro del método
2. Un método que toma un parámetro con tipo java.lang.object tomará cualquier referencia
3. Una referencia a un objeto se puede convertir a una subclase de objetos en una conversión explícita 🡨 Cast
4. Todas las excepciones de conversión se pueden detectar en tiempo de compilación
5. 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
6. ¿Son patrones de diseño de software estructural?

**Investigación.**

los patrones de diseño estructurales son uno de los tres tipos principales de patrones de diseño de software. Los patrones de diseño estructurales se centran en cómo las clases y objetos se componen para formar estructuras más grandes.

Los patrones de diseño estructurales se dividen en dos categorías principales:

Patrones de clase: Estos patrones se enfocan en la composición de clases o en la herencia.

Algunos ejemplos de patrones de clase son:

Adapter

Bridge

Composite

Decorator

Facade

Flyweight

Proxy

Patrones de objeto: Estos patrones se enfocan en la composición de objetos.

Algunos ejemplos de patrones de objeto son:

Chain of Responsibility

Iterator

Mediator

Observer

Los patrones de diseño estructurales tienen como objetivo solucionar problemas relacionados con la forma en que se organizan las clases y los objetos para formar estructuras más grandes. Estos patrones ayudan a crear código más flexible, reutilizable y mantenible al separar las responsabilidades y promover la composición sobre la herencia.

Además de los patrones de diseño estructurales, existen otros dos tipos principales de patrones de diseño:

Patrones de creación: Se enfocan en la forma de crear objetos de manera flexible y controlada. Ejemplos: Singleton, Factory Method, Abstract Factory, Builder, Prototype.

Patrones de comportamiento: Se enfocan en la comunicación entre objetos y la asignación de responsabilidades entre ellos. Ejemplos: Strategy, Observer, Iterator, Template Method, State, Visitor.

Los patrones de diseño estructurales son una categoría importante de patrones de diseño de software que se centran en la composición de clases y objetos para formar estructuras más grandes y complejas de manera flexible y mantenible.

1. Adapter, Proxy, Prototype y Bridge.
2. Adapter, Bridge, Proxy y Composite.
3. Agile, Builder, Singleton y Prototype.
4. Builder, Singleton y Prototype y Abstract Factory.
5. Seleccione la respuesta que considere correcta dado el siguiente bloque de código.

Texto

Descripción generada automáticamente

1. 3.0
2. 1
3. 5
4. 2.5
5. ¿Qué son las pruebas de integración?

Las pruebas de integración son un tipo de pruebas de software que se enfocan en evaluar la interacción y comunicación entre diferentes componentes, módulos o sistemas de una aplicación. Su objetivo principal es verificar que estos componentes funcionen correctamente cuando se combinan e interactúan entre sí.

Algunas características clave de las pruebas de integración son:

1. **Enfoque en las interfaces**: Las pruebas de integración se centran en las interfaces y los puntos de comunicación entre los componentes, asegurando que los datos se transmitan correctamente y que los componentes interactúen de acuerdo con los requisitos establecidos.
2. **Combinación de componentes**: Se prueban combinaciones de componentes que deben trabajar juntos, como módulos, clases, librerías, bases de datos, servicios externos, etc.
3. **Simulación de entornos**: A menudo se requiere simular entornos cercanos a la producción para realizar pruebas de integración realistas, lo que puede implicar la configuración de infraestructura adicional, como servidores, bases de datos, servicios web, etc.
4. **Detección de defectos de integración**: Las pruebas de integración buscan identificar defectos o problemas que surgen de la interacción entre componentes, como errores de comunicación, incompatibilidades de datos, problemas de rendimiento, entre otros.
5. **Estrategias de integración**: Existen diferentes estrategias para abordar las pruebas de integración, como la integración incremental (bottom-up o top-down), la integración big bang, entre otras.

Las pruebas de integración son cruciales en el desarrollo de software, ya que muchos defectos pueden surgir de la interacción entre componentes, incluso si cada componente individual funciona correctamente. Estas pruebas ayudan a detectar y corregir estos defectos antes de pasar a las pruebas de sistema o de aceptación.

Es importante destacar que las pruebas de integración se diferencian de las pruebas unitarias, las cuales se enfocan en probar componentes o unidades de código individuales de manera aislada. Las pruebas de integración, por otro lado, se centran en la combinación e interacción de estos componentes.

En resumen, las pruebas de integración son un tipo de pruebas de software que evalúan la interacción y comunicación entre diferentes componentes, módulos o sistemas de una aplicación, con el objetivo de identificar y corregir defectos de integración antes de pasar a etapas posteriores de pruebas.

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

**Investigación.**

El comando utilizado para enviar los cambios confirmados (commits) en un repositorio local al repositorio remoto en Git es:

**git push**

Este comando básicamente toma los commits que has hecho en la rama local y los envía al repositorio remoto correspondiente.

1. git push
2. it pull
3. git commit
4. git add
5. Seleccione la respuesta correcta, dado el siguiente bloque de código.

Texto

Descripción generada automáticamente

1. Error de compilación
2. 9
3. 7
4. No se ve, pero la correcta es 9
5. ¿Cuál es el comando utilizado para crear una nueva rama en Git?

**Investigación.**

**git branch nombre\_rama**

Este comando creará una nueva rama a partir de la rama actual en la que te encuentres.

Algunas aclaraciones y ejemplos:

nombre\_rama es el nombre que se desea dar a la nueva rama. Por ejemplo: git branch nueva-funcionalidad.

Para crear una nueva rama y cambiar a ella de inmediato, se utiliza el comando: **git checkout -b nombre\_rama**. Este comando crea la nueva rama y cambia al nuevo entorno de la rama creada.

Para ver las ramas existentes dentro del repositorio local, se utiliza el comando: **git branch**. La rama actual estará marcada con un asterisco \*.

Una vez creada la nueva rama, se tiene que cambiar a ella para poder realizar commits en esa rama utilizando el comando: **git checkout nombre\_rama.**

Es recomendable crear ramas nuevas para desarrollar nuevas funcionalidades, corregir errores o experimentar con cambios, en lugar de trabajar directamente en la rama principal (generalmente main o master). Esto facilita la gestión de los cambios y permite un flujo de trabajo más organizado en proyectos con múltiples colaboradores.

1. git branch
2. git merge
3. git commit
4. git push
5. ¿Cuál es el resultado de compilar la siguiente clase?

Texto

Descripción generada automáticamente

1. Línea 15 no compila porque other.ISBN es un atributo con modificador de acceso private.
2. Línea 14 no compila porque no está declarada o manejada ClassCastException
3. No compila porque no está sobreescribiendo el método equals correctamente.
4. La clase compila satisfactoriamente.
5. ¿Cuál es la primer línea en fallar al compilar?

Texto

Descripción generada automáticamente

1. r5
2. r4
3. r3
4. Ninguna de las anteriores.
5. ¿Qué es Git?

**Investigación.**

Git es un sistema de control de versiones distribuido diseñado para manejar todo tipo de proyectos con velocidad y eficiencia. Su propósito principal es registrar los cambios en los archivos a lo largo del tiempo, permitiendo a los desarrolladores colaborar, gestionar y revertir cambios en el código fuente de manera efectiva.

Las características clave de Git son:

**Distribuido:** Cada desarrollador tiene una copia completa del historial del proyecto, lo que permite trabajar sin necesidad de una conexión constante a un servidor central.

**Ramas y fusiones:** Git facilita la creación de ramas para desarrollar nuevas características o corregir errores de forma aislada, y luego fusionar esos cambios de vuelta a la rama principal.

**Integridad de los datos:** Cada cambio en Git es registrado con un hash SHA-1, asegurando que los cambios no pueden ser modificados sin que Git lo detecte.

**Rendimiento:** Git está optimizado para el rendimiento, permitiendo operaciones rápidas incluso en proyectos grandes.

**Flexibilidad:** Git puede ser utilizado para una amplia gama de flujos de trabajo y estilos de desarrollo, desde pequeños proyectos personales hasta grandes proyectos con miles de colaboradores.

Git se utiliza ampliamente en la industria del software y es fundamental para prácticas de desarrollo modernas como la integración continua (CI) y el despliegue continuo (CD).

1. Una herramienta de automatización de compilación que se utiliza para compilar un proyecto.
2. Una herramienta de generación de informes que se utiliza para generar informes sobre el rendimiento de una aplicación.
3. Una herramienta de gestión de dependencias que se utiliza para descargar bibliotecas y paquetes en un proyecto de Java.
4. Una herramienta de control de versiones que se utiliza para almacenar y administrar el código fuente de un proyecto.
5. ¿Cuál de las siguientes excepciones lanza la JVM? (Elija todas las correctas)

**Investigación.**

**ArrayIndexOutOfBoundsException:** Esta excepción se produce cuando se intenta acceder a un índice de array fuera de los límites del mismo.

**NumberFormatException:** Esta excepción se produce cuando se intenta convertir una cadena de texto a un número, pero la cadena no tiene el formato correcto para el tipo numérico deseado.

**ExceptionInInitializerError:** Esta no es una excepción lanzada por la JVM, sino un error. Es una subclase de LinkageError y se produce cuando una clase inicializadora arroja una excepción.

**Java.io.IOException:** Esta no es una excepción lanzada directamente por la JVM. Es una excepción de la API de Java que se produce cuando ocurre un error de entrada/salida.

**NullPointerException:** Esta excepción se produce cuando se intenta acceder a un miembro (campo o método) de un objeto que tiene un valor nulo.

Las excepciones ArrayIndexOutOfBoundsException, NumberFormatException y NullPointerException son excepciones comunes lanzadas por la JVM cuando se producen errores en tiempo de ejecución relacionados con accesos incorrectos a arreglos, conversiones de cadenas a números y referencias nulas, respectivamente.

1. ArrayIndexOutOfBoundException
2. NumberFormatException <-correcta en investigación
3. ExceptionInInitializerError <-correcta en guía
4. Java.io.IOException
5. NullPointerException
6. ¿Cuál es el comando utilizado para fusionar una rama en Git?

**Investigación.**

**git merge nombre\_rama**

Este comando combina los cambios de la rama especificada (nombre\_rama) con la rama actual en la que se esté posicionado.

Algunas de las principales consideraciones importantes sobre el comando git merge son:

Rama actual: El comando git merge incorporará los cambios de nombre\_rama a la rama en la que se encuentre actualmente. Puedes verificar en qué rama se encuentra con git branch (la rama actual está marcada con un \*).

Conflictos de fusión: Si hay conflictos entre los cambios en las dos ramas (por ejemplo, si se modificaron las mismas líneas de código), Git no podrá fusionar automáticamente. En ese caso, deberás resolver manualmente los conflictos editando los archivos afectados.

Commit de fusión: Después de una fusión exitosa, Git creará un nuevo commit de fusión que combina el historial de ambas ramas.

Actualizar antes de fusionar: Antes de fusionar, es una buena práctica asegurarse de que tu rama actual esté actualizada con los últimos cambios del repositorio remoto ejecutando git pull.

Fast-forward merge: Si la rama que intentas fusionar no tiene commits adicionales desde que se bifurcó de la rama actual, Git puede realizar una fusión rápida (fast-forward) sin crear un commit de fusión.

Ejemplos de uso:

git merge develop: Fusiona la rama develop en la rama actual.

git merge --no-ff nueva-funcionalidad: Fusiona la rama nueva-funcionalidad en la rama actual, creando un commit de fusión incluso si es posible una fusión rápida.

Es importante fusionar ramas con cuidado y resolver cualquier conflicto que pueda surgir para mantener la integridad del historial de confirmaciones del proyecto.

1. git Branch
2. git merge
3. git push
4. git pull
5. ¿Qué es REST y cuál es su relación con las API web?

**Investigación.**

REST (Representational State Transfer) es un estilo arquitectónico para diseñar sistemas de software distribuidos basados en la web, principalmente para servicios web. Tiene una estrecha relación con las API web, ya que proporciona una forma estandarizada y estructurada de construirlas.

REST establece una serie de principios y restricciones para crear APIs web escalables, mantenibles y fáciles de consumir. Algunas de sus características principales son:

Arquitectura cliente-servidor: Separa las responsabilidades del cliente y el servidor, lo que permite una mayor escalabilidad y portabilidad.

Sin estado: Cada solicitud HTTP contiene toda la información necesaria para ser procesada por el servidor, lo que facilita el escalado y la tolerancia a fallos.

Cacheable: Las respuestas indicarán si pueden ser almacenadas en caché o no, lo que mejora el rendimiento y la eficiencia.

Interfaz uniforme: Define una interfaz genérica basada en recursos, donde cada recurso se identifica mediante una URI y se manipula a través de los métodos HTTP estándar (GET, POST, PUT, DELETE, etc.).

Sistema de capas: Permite la existencia de intermediarios (proxies, gateways, etc.) que pueden mejorar la escalabilidad y la seguridad.

Código bajo demanda (opcional): Permite la transferencia de código ejecutable, como applets o scripts, para extender la funcionalidad del cliente.

Las API web que siguen estos principios REST se conocen como APIs RESTful. Estas APIs utilizan URLs para identificar recursos, métodos HTTP para especificar acciones sobre esos recursos, y formatos como JSON o XML para representar los datos transferidos.

La relación entre REST y las APIs web es que REST proporciona una arquitectura y un conjunto de reglas para diseñar APIs web de manera consistente, escalable y fácil de consumir. Las APIs RESTful se han vuelto muy populares debido a su simplicidad, escalabilidad y compatibilidad con diferentes tipos de clientes (navegadores web, aplicaciones móviles, etc.).

1. REST es un protocolo de comunicación. Su relación con las API web es que las utiliza para definir los endpoints de una API
2. REST es in lenguaje de programación. Su relación con las API web es que se utiliza para crear aplicaciones web.
3. REST es un servicio en la nube. Su relación con las API web que se utiliza para alojar las aplicaciones web.
4. 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.
5. ¿Cuál es el comando utilizado para actualizar la rama local con los cambios de la rama remota en Git?

**Investigación.**

**git pull**

Este comando básicamente combina dos operaciones en una:

git fetch: Descarga los objetos (commits, archivos, etc.) desde el repositorio remoto a tu repositorio local.

git merge: Fusiona los commits descargados en la rama local actual.

Sin embargo, es importante tener en cuenta que, por defecto, el comando git pull intentará fusionar automáticamente los cambios remotos en tu rama local. Si hay conflictos entre los cambios locales y remotos, Git no podrá completar la fusión automáticamente y se resuelven manualmente.

Es una buena práctica ejecutar **git pull** antes de comenzar a trabajar en un proyecto para asegurarte de tener los últimos cambios del repositorio remoto en la rama local.

1. git checkout
2. git clone
3. git push
4. git pull
5. ¿Qué es un microservicio?

**Investigación.**

Un microservicio es un enfoque arquitectónico para el desarrollo de aplicaciones, donde una aplicación grande y compleja se divide en pequeños servicios independientes que se ejecutan en su propio proceso y se comunican entre sí a través de interfaces bien definidas, generalmente utilizando protocolos ligeros como HTTP/REST o mensajería. Cada microservicio es responsable de una funcionalidad específica y puede ser desarrollado, desplegado y escalado de manera independiente.

1. Es un componente que se pueden desplegar de forma independiente en función múltiple, es decir, englobando endpoint que no necesariamente están relacionados.
2. 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. 🡨DUDA
3. Es el conjunto de endpoints contenidos en múltiples desarrollos que se despliegan en conjunto y que están estrechamente relacionados.
4. Ninguna de las anteriores
5. Dado el siguiente código:

Texto

Descripción generada automáticamente

¿Este código compila sin errores?

1. Si, compila sin errores.
2. No, hay un error en el ciclo for.
3. No, hay un error en la inicialización de la variable “suma”.
4. No, hay un error en la declaración del arreglo.
5. ¿Qué método se utiliza para obtener el mensaje de una excepción en Java?

**Investigación.**

En Java, para obtener el mensaje de una excepción, se utiliza el método getMessage(). Este método forma parte de la clase Throwable, que es la superclase de todas las excepciones y errores en Java.

Además de getMessage(), la clase Throwable también proporciona otros métodos útiles, como printStackTrace(), que imprime la traza de la pila de llamadas, y toString(), que devuelve una representación en forma de cadena de la excepción y su mensaje.

a) getClass()

b) printStackTrace()

c) toString()

d) getMessage()

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

**Investigación.**

Las excepciones en tiempo de ejecución (RuntimeException y sus subclases) son consideradas excepciones no comprobadas (unchecked exceptions) en Java.

El compilador de Java no obliga al programador a manejar o declarar estas excepciones de manera explícita.

Ejemplos de excepciones en tiempo de ejecución son: NullPointerException, ArrayIndexOutOfBoundsException, IllegalArgumentException.

1. Puede declarar solo excepciones no comprobadas (unchecked).
2. Las excepciones en tiempo de ejecución son lo mismo que las excepciones no comprobadas.
3. Las excepciones en tiempo de ejecución son lo mismo que las excepciones comprobadas.
4. Solo puede declarar excepciones comprobadas (checked)
5. Solo puede manejar subclases de Exception.

**En esta te pide que selecciones varias, pero no se puede**

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

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

1. NullPointerException
2. hello
3. HELLO
4. Hello
5. ¿Cuál es el paquete de importación necesario para usar la clase ArrayList?

**Investigación.**

La clase ArrayList se encuentra en el paquete java.util de la biblioteca estándar de Java.

Para poder utilizar la clase ArrayList en tu código Java, debes importar el paquete java.util

1. import.java.net.\*;
2. import.java.awt.\*;
3. import.java.io. \*;
4. import.java.util.\*;

**No debería llevar punto entre el import y java, lo que haría que ninguna es correcta**

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

**Investigación.**

JSON es un formato ligero de intercambio de datos, fácil de leer y escribir para humanos, y fácil de analizar y generar para máquinas. Se basa en un subconjunto del lenguaje de programación JavaScript, pero es independiente del lenguaje y se utiliza ampliamente en diversas tecnologías y entornos.

Aunque XML también se utiliza en algunas APIs RESTful, JSON se ha convertido en el formato predominante debido a su simplicidad y eficiencia. JSON es más compacto y ligero que XML, lo que lo hace más adecuado para transferencias de datos en aplicaciones web y móviles.

YAML (YAML Ain't Markup Language) es otro formato de serialización de datos similar a JSON, pero no es tan ampliamente utilizado en APIs RESTful como JSON.

Aún que XML y YAML son formatos válidos y pueden usarse en APIs RESTful, JSON es el formato de datos más común y ampliamente adoptado para el intercambio de datos en este tipo de APIs.

1. YAML
2. XML
3. JSON <-Correcta en investigación.
4. Todos los anteriores
5. ¿Cuál es la función del operador de doble dos puntos (::) en Java 8?

**Investigación.**

El operador de doble dos puntos (::) es una característica introducida en Java 8 como parte de las expresiones lambda y hace referencia a métodos. Se llama operador de referencia de método (method reference operator).

**NombreClase::nombreMetodoEstático**

Esto permite hacer referencia directa a un método estático de una clase, en lugar de tener que definir una expresión lambda completa. Es una forma más concisa y legible de acceder a métodos estáticos cuando se trabaja con lambdas y funciones de alto orden.

El operador de doble dos puntos (::) en Java 8 se utiliza para hacer referencia a métodos estáticos de una manera más concisa y legible cuando se trabaja con expresiones lambda y funciones de alto orden.

1. El operador doble dos puntos se utiliza para crear una nueva instancia de una clase en Java 8. correcta 3
2. EL operador de doble dos puntos no se utiliza en java 8.
3. El operador de doble dos puntos se utiliza para acceder a métodos estáticos en Java 8. 🡨 Esta es correcta 1
4. El operador de doble dos puntos se utiliza para acceder a métodos no estáticos en Java 8. 🡨 Esta también es correcta en ciertos escenarios 2
5. ¿Qué palabra clave se utiliza para definir una excepción personalizada en Java?

**Investigación.**

Para definir una excepción personalizada en JAVA, el proceso implica extender la clase Exception o alguna de sus subclases, lo cual se hace utilizando la palabra clave class.

try - Se usa para comenzar un bloque de código que se va a monitorear para excepciones.

throw - Se usa para lanzar una excepción.

finally - Se usa para definir un bloque de código que se ejecutará siempre, después de que se ejecuten los bloques try y catch.

catch - Se usa para capturar una excepción lanzada en un bloque try.

Ninguna de las palabras clave se utiliza para definir una excepción personalizada.

1. try
2. throw
3. finally
4. catch
5. ¿Cuál de los siguientes comandos elimina el directorio target antes de iniciar el proceso de construcción?

**Investigación.**

El comando mvn clean se utiliza para limpiar el directorio target, donde Maven coloca los archivos generados durante la construcción del proyecto. Esto incluye eliminar los archivos compilados y cualquier otro artefacto generado. Después de ejecutar mvn clean, puedes ejecutar otros comandos de construcción como mvn install o mvn package para construir tu proyecto desde un estado limpio.

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

**Investigación.**

El comando git log se utiliza en Git para mostrar el historial de commits (confirmaciones) realizados en un repositorio. Este comando proporciona una gran cantidad de información útil sobre los cambios realizados en el código fuente y quién los realizó.

1. git log
2. git status
3. git commit
4. git diff
5. ¿Qué es una expresión lambda en Java 8?

**Investigación.**

Una expresión lambda, introducida en Java 8, es una forma concisa de representar una interfaz funcional anónima (una interfaz con un solo método abstracto) utilizando una notación más compacta y legible.

Las expresiones lambda permiten tratar el código como datos, lo que facilita el uso de programación funcional en Java. Esencialmente, una expresión lambda es una función anónima que se puede pasar como argumento a un método, almacenar en una variable o devolver desde otro método.

Una expresión lambda es una forma de escribir una función anónima en Java 8.

Las expresiones lambda, introducidas en Java 8, permiten representar interfaces funcionales anónimas (interfaces con un solo método abstracto) de forma más concisa y legible, utilizando una notación similar a las funciones anónimas de otros lenguajes de programación funcional.

1. Una expresión lambda es una forma de escribir una clase anónima en Java 8.
2. Una expresión lambda es una forma de escribir una función anónima en Java 8. <-Correcta en investigación
3. Una expresión lambda es un método que se llama automáticamente cuando se crea un objeto.
4. Una expresión lambda es un método que se llama de forma explícita desde el código.
5. ¿Qué muestra el siguiente código fuente por pantalla?

Texto

Descripción generada automáticamente

1. Dos
2. Uno Dos Tres Otro número
3. Uno
4. Otro número
5. De los siguientes paquetes, ¿cuáles contienen clases para construir una interfaz gráfica? (Elije todas las que correspondan)
6. ¿Cuál de las siguientes líneas deben ir en el espacio en blanco para que el código compile?

public class News < \_\_\_\_\_> { }

**Investigación.**

a) "Solo N" es incorrecto, ya que se requieren dos tipos de parámetros.

b) "Solo ?" es incorrecto, ya que el signo de interrogación no es un nombre de tipo válido en Java.

c) "Ninguna de las anteriores" es incorrecto, porque se requieren tipos de parámetros para que la clase genérica compile.

e) "? y N" es incorrecto, porque el signo de interrogación no es un nombre de tipo válido, y además se requiere el tipo Object como límite superior.

f) "N, News y Object" es incorrecto, porque el tipo de parámetro News solo debe aparecer una vez

En Java, al definir una clase genérica, se debe especificar el nombre de uno o más tipos de parámetros entre los símbolos < >. Estos tipos de parámetros son conocidos como parámetros de tipo, y se pueden utilizar dentro de la definición de la clase para definir campos, métodos y demás.

En el caso de la clase News, para que compile correctamente, se deben proporcionar dos tipos de parámetros: el primero es el nombre de la propia clase (News), y el segundo es el tipo de objeto que se utilizará como límite superior (Object).

1. Solo N
2. Solo ?
3. Ninguna de las anteriores
4. News,y Object <-Correcta en investigación.
5. ? y N
6. N, News y Object
7. ¿Qué es un stream en Java 8 y para qué se utiliza?

**Investigación.**

Un stream en Java 8 es una secuencia de elementos que soporta diferentes tipos de operaciones para procesar datos de manera declarativa. Las principales características clave de los streams en Java 8 son:

Permiten procesar colecciones de datos de manera eficiente utilizando operaciones como filter, map, reduce, etc.

Soportan procesamiento en paralelo para aprovechar múltiples núcleos de CPU

Utilizan lazy evaluation, lo que significa que las operaciones no se ejecutan hasta que se invoca una operación terminal

Pueden manejar diferentes tipos de datos como objetos, primitivos int/long/double, etc.

Los usos comunes de los streams en Java 8 incluyen:

Filtrar y transformar datos de colecciones de manera concisa y legible

Realizar agregaciones y reducciones sobre datos de manera paralela

Generar y procesar secuencias infinitas de datos

Integrar con APIs como I/O, bases de datos, etc. para procesar datos de manera declarativa

Los streams en Java 8 permiten procesar datos de manera eficiente, concisa y paralela, facilitando la programación funcional en Java. Permiten separar la lógica de procesamiento de datos de la implementación de bajo nivel, resultando en código más legible y mantenible.

Un stream en Java 8 es un objeto que representa una secuencia de elementos y se utiliza para procesar colecciones de forma declarativa.

1. Un objeto que representa una conexión de entrada o salida de datos.
2. Un objeto que representa una secuencia de elementos y se utiliza para procesar colecciones de forma declarativa.
3. Un objeto que se utiliza para leer y escribir archivos de texto.
4. Un objeto que se utiliza para crear y manipular bases de datos.