## **CONOCIMIENTOS GENERALES DE PROGRAMACIÓN**

## **1.**¿Qué es una clase? ¿Qué es un objeto? ¿Cual es la diferencia entre ellos?

*Clase* : Una clase es una abstracción de objetos reales. Una clase es un arquetipo, un modelo, un plano que sirve como base para construir diferentes objetos y contiene la definición de varios atributos, métodos o acciones que son comunes para cada objeto.

*Objeto* : Esta es una instancia específica de una clase. Tiene valores específicos para cada atributo.

## 2.     ¿Qué es la encapsulación?

La encapsulación significa que todos los valores de los atributos de un objeto específico están envueltos dentro del objeto y, por lo tanto, no son accesibles para objetos extraños. El objetivo es asegurarse de que los datos "sensibles" estén ocultos para los usuarios. Esto se logra declarando los atributos como privados. Para permitir el acceso, se hace a través de métodos get/set.

Razones:

* Mejor control de los atributos y métodos de clase.
* Las variables de clase pueden ser de solo lectura (si omite el método set) o de solo escritura (si omite el método get)
* Flexible: el programador puede cambiar una parte del código sin afectar otras partes
* Mayor seguridad de los datos

Fuente :  [W3schools](https://www.w3schools.com/java/java_encapsulation.asp)

## **PREGUNTAS JAVA**

## 3.     ¿Qué es una clase abstracta?

Una clase abstracta es una clase que no puede ser instanciada. Por lo general, sirve como supermodelo para otras clases que heredan sus atributos y métodos. Proporciona una interfaz común que permite intercambiar subclases con todas las demás subclases.

## **4.**¿Qué es una clase inmutable?

Una clase inmutable es una clase cuyas instancias no se pueden modificar. Se inicializa en el momento de la construcción y no se puede modificar posteriormente en la ejecución del programa.

Son eficientes y seguros, también seguros para subprocesos. Son especialmente útiles en aplicaciones concurrentes.

## **5.**¿Qué es una interfaz?

Una interfaz es un tipo de referencia en Java. Es una colección de métodos abstractos. Contiene solo la declaración de los métodos, pero su cuerpo se implementa en otro lugar. Otra clase implementa la interfaz, por lo que hereda los métodos abstractos. La clase implementador contiene la definición de los métodos.

## **6.**¿Puedes crear el cuerpo de un método en una interfaz?

Normalmente No. El cuerpo del método debe crearse en la clase del implementador. Sin embargo, esta funcionalidad ha sido añadida en Java 8 a través de los llamados Métodos Predeterminados.

## **7.**¿Cuáles son los métodos predeterminados?

Los métodos predeterminados son métodos cuya implementación está presente en las propias interfaces. Solo están disponibles desde Java 8 en adelante. Por lo tanto, un implementador no necesita anular los métodos de una interfaz (pero puede hacerlo).

## **8.**¿Qué es la anulación de métodos?

La anulación de métodos es posible gracias a la herencia. Cuando tenemos una superclase con el método A, cada subclase que hereda de ella puede tener diferentes implementaciones del método, según las necesidades de la clase. En palabras simples: la anulación de métodos está presente cuando una subclase proporciona la implementación específica del método que ha sido declarado por su clase principal. La anulación de métodos debe cumplir tres criterios:

* El método debe tener el mismo nombre en las super y subclases.
* El método debe tener los mismos parámetros que en la clase principal.
* Debe ser una relación IS-A (herencia).

La anulación es posible a través de la anotación: @Override  
Fuente:  [Oracle](https://docs.oracle.com/javase/tutorial/java/IandI/override.html)

## **9.**¿Puede anular la construcción de una clase?

No, los constructores no se pueden anular.

## **10.**¿Puede anular los métodos estáticos?

No. Cuando creamos un método estático en una subclase con el mismo nombre y parámetros que en la clase superior, el método estático en la superclase se oculta.   
**Fuente:**[**Oráculo**](https://docs.oracle.com/javase/tutorial/java/IandI/override.html)

## **11.**¿Qué es una "construcción predeterminada" de una clase?

Un constructor predeterminado es un constructor sin parámetros formales y sin cláusula throws, generado automáticamente por el compilador si no se ha especificado ningún otro constructor.   
Fuente: [Oráculo](https://docs.oracle.com/javase/specs/jls/se7/html/jls-8.html" \l "jls-8.8.9)

## **12.**¿Qué significa sobrecarga de métodos?

La sobrecarga de métodos ocurre cuando tenemos múltiples métodos con el mismo nombre, pero diferentes parámetros. La sobrecarga puede ocurrir cambiando el número de argumentos o el tipo de datos.  
Fuente:  [Javatpoint](https://www.javatpoint.com/method-overloading-in-java)

## **13.**¿En qué consiste el patrón de diseño, Singleton?

Singleton es un patrón de diseño que asegura que solo existe una única instancia de un objeto. Lo logra al declarar un objeto estático de sí mismo dentro de la clase y hacer que el constructor sea privado. Por lo tanto, la única forma de acceder a él es a través del objeto estático. El patrón singleton también pertenece a los patrones creacionales y se basa en la afirmación de que la clase contiene una instancia estática de sí misma. Se puede acceder a esta instancia a través de un método estático llamado getInstance(). El constructor de esta clase debe ser privado. Por lo tanto, esta clase no se puede instanciar, lo que garantiza que solo existe un único objeto de esta clase en todo momento. **Fuente:**[**Tutorialspoint**](https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm)

## **14.**¿Qué es el patrón de fábrica?

El patrón de fábrica es un patrón de diseño en el que, en lugar de usar el operador habitual new para crear objetos, creamos una clase especializada que maneja la creación de múltiples objetos. Para crear un nuevo objeto, hacemos uso de una clase Factory que se encarga de establecer los atributos y valores apropiados.  
**Fuente:**[**Tutorialspoint**](https://www.tutorialspoint.com/design_pattern/factory_pattern.htm)

## **15.**¿Es posible la herencia múltiple en Java?

Java no permite extenderse desde más de una clase para evitar problemas de *múltiples herencias de estado.*   
Sin embargo, Java permite *la herencia múltiple de tipos* , que es la capacidad de una clase para implementar más de una interfaz.  
Fuente:  [Oráculo](https://docs.oracle.com/javase/tutorial/java/IandI/multipleinheritance.html)

## **16.**¿Qué es la composición en Java?

La composición define una relación de agregación en el sentido que tiene Clase “A”-una clase “B”. Un ejemplo puede ser que la clase COCHE tenga una clase Rueda. En este sentido, uno de los atributos de una clase es otra clase.

## **17.**¿Qué son los modificadores de acceso en Java?

Los modificadores de acceso son un conjunto de palabras clave que permiten modificar los niveles de acceso de clases, variables, métodos y constructores. Hay 4 modificadores de acceso en Java:

* *Predeterminado* : No se define ningún modificador de acceso. Las variables y los métodos se vuelven visibles para todo el paquete.
* *Privado:* los miembros solo son visibles dentro de la misma clase.
* *Protegido:* las variables y los métodos son visibles dentro de la clase y cualquier otra subclase que herede de ella.
* *Público:* los miembros públicos son accesibles en todas partes.

Fuente:  [Tutorialspoint](https://www.tutorialspoint.com/java/java_access_modifiers.htm)

## **18.**¿Qué es una clase interna?

Una clase interna, también llamada *clase anidada* , es una clase en la que la declaración forma parte de la declaración de otra clase. En términos generales, hay dos tipos de clases anidadas: estáticas y no estáticas.  
Los beneficios de las clases anidadas incluyen: Permite el uso de modificadores de acceso dentro de la clase, lo que hace posible la creación de clases privadas. Además, las clases internas tienen acceso a las variables privadas de la clase externa.  
Fuente:  [Tutorialspoint](https://www.tutorialspoint.com/java/java_innerclasses.htm)

## **19.**¿Qué es una clase local?

Una clase local es una clase definida en un bloque. Por ejemplo, puede definir una clase local en el cuerpo de un método, un bucle for o una cláusula if.  
Fuente:  [Oráculo](https://docs.oracle.com/javase/tutorial/java/javaOO/localclasses.html)

## **20.**¿Qué es una clase anónima? ¿Cuándo podemos usarlo?

Una clase anónima es una clase local sin nombre. Es decir: una clase que está definida en un bloque (un for, un method, un if) y no tiene nombre. Los usamos solo si necesitamos usar una clase local solo una vez.  
Se recomienda utilizar clases anónimas cuando sea necesario declarar campos o métodos adicionales.  
Fuente:  [Oracle-1](https://docs.oracle.com/javase/tutorial/java/javaOO/anonymousclasses.html)  
[Oracle-2](https://docs.oracle.com/javase/tutorial/java/javaOO/whentouse.html)

## **21.**¿Qué es una expresión, una declaración y un bloque?

Una *expresión* es una construcción formada por variables, operadores e invocaciones de métodos, que se construyen de acuerdo con la sintaxis del lenguaje, que evalúa un solo valor. Ejemplo: *cadencia = 0, 1\*2\*3, (x+y)/100.*  
*Las declaraciones* son aproximadamente equivalentes a oraciones en lenguajes naturales. Una declaración forma una unidad completa de ejecución. Los siguientes tipos de expresiones se pueden convertir en una declaración terminando la expresión con un punto y coma (;). Ejemplo:

// declaración de asignación  
aValue = 8933.234;  
// declaración de incremento

unValor++;  
// instrucción de invocación del método  
System.out.println("¡Hola mundo!");  
// sentencia de creación de objeto  
Bicycle myBike = new Bicycle();

Un *bloque* es un grupo de cero o más declaraciones entre llaves equilibradas y se puede usar en cualquier lugar donde se permita una sola declaración. El siguiente ejemplo, BlockDemo, ilustra el uso de bloques:

clase BlockDemo {

     public static void main(String[] args) {  
          boolean condition = true;

          if (condición) { **// comienza el bloque 1**

               System.out.println("La condición es verdadera");

          } **// finaliza el bloque uno**

          else { **// comienza el bloque 2**

               System.out.println("La condición es falsa.");

          } **// bloque final 2**

     }

}

Fuente:  [Oráculo](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/expressions.html)

## **22.**¿Qué es la inyección de dependencia?

La inyección de dependencia básicamente proporciona los objetos que un objeto necesita (sus dependencias) en lugar de que los construya él mismo. Es una técnica muy útil para realizar pruebas, ya que permite simular o eliminar dependencias.  
Las dependencias se pueden inyectar en objetos por cualquier medio (como inyección de constructor o inyección de setter). Incluso se pueden usar marcos de inyección de dependencia especializados (por ejemplo, Spring) para hacer eso, pero ciertamente no son necesarios.  
  
Ejemplo:

public SomeClass() {

myObject = Factory.getObject();  
}

Aplicando la inyección de dependencia al código anterior, tendrás:

public SomeClass (MyClass myObject) {

    this.miObjeto = miObjeto;

}

Ahora, myObject se recibe como argumento, y SomeClass no tiene que ocuparse de su creación/inicialización. Por lo tanto, la dependencia myObject se está inyectando en SomeClass.

## **23.**¿Qué es el marco de primavera?

Spring Framework es un marco de aplicación para Java. Cualquier aplicación Java puede utilizar las funciones principales de Spring, pero las extensiones permiten la creación de aplicaciones web sobre Java EE. Es una fuente abierta.  
Fuente: [Wikipedia](https://en.wikipedia.org/wiki/Spring_Framework)

## **24.**¿Para qué se usa la palabra clave transitoria?

Antes de entender la palabra clave transitoria, uno tiene que entender el concepto de serialización.  
*La serialización* es el proceso de hacer que el estado del objeto sea persistente. Eso significa que el estado del objeto se convierte en un flujo de bytes y se almacena en un archivo. De la misma manera, podemos usar la deserialización para recuperar el estado del objeto de los bytes.  
La palabra clave *transitoria* indica que una variable no forma parte del estado persistente de un objeto. Esto significa que el contenido de la variable no debe persistir en un archivo y debe derivarse mediante programación de otros campos. Considere el siguiente ejemplo:

clase GalleryImage implementa Serializable

{

    Imagen de imagen privada;

    imagen transitoria privada thumbnailImage;

    privado void generarThumbnail()  
    {  
        // Generar miniatura.  
    }

    privado vacío readObject (ObjectInputStream inputStream)

            lanza IOException, ClassNotFoundException  
    {  
        inputStream.defaultReadObject();  
        generar miniatura();  
    }      
}

En el ejemplo, la imagen de atributo es serializable. Es decir, forma parte del estado persistente del objeto. Por otro lado, thumbnailImage es *transitorio* , por lo tanto, no persistirá. Este ejemplo ahorraría capacidad de almacenamiento.

## 25 **.**¿Cómo podemos comparar dos cadenas en Java?

La comparación de cadenas en Java es un tema especial. En lugar de usar el operador habitual ==, deberíamos usar el método *equals()* para verificar si el valor de dos cadenas es el mismo.  
  
== prueba la igualdad de referencia (si son el mismo objeto).  
  
.equals() prueba la igualdad de valores (si son lógicamente "iguales").

// Estos dos tienen el mismo valor

nuevo String("prueba").equals("prueba") // --> verdadero

// ... pero no son el mismo objeto

new String("prueba") == "prueba" // --> falso

// ... estos tampoco

nueva cadena("prueba") == nueva cadena("prueba") // --> falso

// ... pero estos son porque los literales están internados por

// el compilador y así referirse al mismo objeto  
"test" == "test" // --> true

// ... los literales de cadena son concatenados por el compilador

// y se internan los resultados.  
"prueba" == "te" + "st" // --> verdadero

## **26.**¿Qué es el método hashCode()? ¿Para qué se usa esto?

El método hashCode() ( *en una cadena* ) es un cálculo complejo del tipo:  
  
s[0]\*31^(n - 1) + s[1]\*31^(n - 2) + ... + s[n - 1]  
  
Usando aritmética int, donde s[i] es el i-ésimo carácter de la cadena, n es la longitud de la cadena y ^ indica exponenciación. (El valor hash de la cadena vacía es cero).  
El método hashCode() *en un objeto* devuelve la dirección de memoria del objeto en hexadecimal. Por definición, si dos objetos son iguales, su código hash también debe ser igual.  
**Fuente:**[**Oracle**](https://docs.oracle.com/javase/tutorial/java/IandI/objectclass.html)  
  
Hashcode se utiliza para *el hashing de cubos* , es decir, para almacenar objetos en un Hashmap. Los mapas hash generalmente se dividen en cubos y cada cubo puede contener varios pares clave/valor. El hashCode() del objeto determina en qué depósito entra, a través de esta expresión: object.hashCode() % n, donde n = el número total de depósitos y % es el operador de módulo.  
  
La mayoría de las veces, los objetos estarán bien distribuidos en los cubos, pero no tiene garantía de adónde van. Esto depende de los datos y de la función hashCode.  
**Fuente:**[**Stackoverflow**](https://stackoverflow.com/questions/18636576/what-is-meant-by-number-of-buckets-in-the-hashmap)  
  
Si realiza una solicitud de contiene (), el mapa hash tomará el código hash del elemento, luego buscará el depósito al que apunta el código hash. Si hay más de un elemento en el cubo, el hashmap usará el método equals() para evaluar si los objetos son iguales.  
Fuente :  [Stackoverflow](https://stackoverflow.com/questions/3563847/what-is-the-use-of-hashcode-in-java)

## **27.**¿Cuál es el uso del método equals()?

El método equals nos permite evaluar si dos objetos son iguales. El método tiene cuatro características: reflexivo, simétrico, transitivo, consistente. El método equals para la clase Object implementa la relación de equivalencia más exigente posible en los objetos; es decir, para cualquier valor de referencia no nulo x e y, este método devuelve verdadero si y solo si x e y se refieren al mismo objeto (x == y tiene el valor verdadero).  
Fuente :  [oráculo](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html" \l "equals(java.lang.Object))

## **28.**¿Cuál es la relación entre equals() y hashCode()?

Cada vez que se invalida el método equals, también se debe invalidar el método hashcode(), a fin de mantener el contrato general para el método hashCode(), que establece que los objetos iguales deben tener códigos hash iguales.  
**Fuente:  [Oráculo](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html" \l "equals(java.lang.Object))**

## **29.**¿Cuáles son los principales elementos que contiene la Biblioteca de Colecciones?

Mapas, Conjuntos y Listas. Los tres son interfaces que se pueden implementar de manera diferente a través de varios implementadores también contenidos en la biblioteca.

## **30.**¿Cuál es la diferencia entre una Lista, un Conjunto y un Mapa?

**Lista:** Representa una secuencia ordenada de objetos. Cada elemento tiene un índice y se puede acceder a ellos, iterarlos y eliminarlos según el orden en que aparecen. Las listas proporcionan una colección ordenada e indexada que  **puede contener duplicados** .  
  
**Conjunto:**es una  colección desordenada de objetos únicos. **Los conjuntos no permiten duplicados.**Ciertas implementaciones mantienen el orden.  
  
**Mapa:** Un mapa es una secuencia ordenada de objetos que consta de una  **clave, pares de valores** . Puede contener valores duplicados.

## **31.**¿Cuál es la diferencia entre ArrayList y LinkedList?

Ambas son implementaciones de la interfaz List. La diferencia es el elemento que utilizan para almacenar los elementos. ArrayList usa, como se esperaba, una matriz dinámica para almacenar los elementos. LinkedLists usa una lista doblemente enlazada. Los trajes de ArrayList son mejores para almacenar y acceder a datos, mientras que LL es mejor para manipular datos (son más rápidos cuando se eliminan porque no se requiere cambio de bits).  
  
**Fuente:**[**Javatpoint**](https://www.javatpoint.com/difference-between-arraylist-and-linkedlist)

## **32.**¿Qué son los mapas?

Maps es una interfaz disponible en la biblioteca de colecciones de Java, que contiene una colección de pares clave-valor.

## **33.**¿Es un mapa una clase o una interfaz?

Java Map es una interfaz que representa un mapeo entre una clave y un valor. Las implementaciones del Mapa incluyen:

* java.util.HashMap
* java.util.Hashtable
* ja va.util.EnumMap
* java.util.IdentityHashMap
* java.util.LinkedHashMap
* java.util.Propiedades
* java.util.TreeMap
* java.util.WeakHashMap

## **34.**¿Puedes mencionar algunos usos típicos de listas, conjuntos y mapas?

El uso de cada colección depende de lo que estamos tratando de lograr. Por ejemplo, si queremos identificar elementos basados ​​en un índice, debemos usar una lista. Si tenemos que asegurarnos de que no haya duplicados, deberíamos usar un conjunto. Por último, los mapas se utilizan para claves, pares de valores, como países y monedas, etiquetas y definiciones de idiomas, etc.

## **35.**¿Para qué se utiliza la palabra clave*final*?

La palabra clave *final* se utiliza para indicar que elementos específicos no sufrirán más cambios. Se puede aplicar a una variable, un método o una clase. En cada caso, indica algo un poco diferente, como sigue:

* *Variable final* : el valor de la variable no cambiará. En otras palabras, es una constante.
* *Método final* : el método no se puede anular.
* *Clase final* : la clase no puede convertirse en una superclase. Detiene la herencia.

Fuente :  [Javatpoint](https://www.javatpoint.com/final-keyword)

## **36.**¿Qué es una excepción?

Una excepción (o evento excepcional) es un problema que surge durante la ejecución de un programa. Cuando ocurre una Excepción, el flujo normal del programa se interrumpe y el programa/Aplicación finaliza de manera anormal, lo que no se recomienda, por lo tanto, estas excepciones deben manejarse.  
  
Hay 3 clases de excepciones:

* *Excepciones comprobadas:*  el compilador comprueba esto en el momento de la compilación.
* *Excepciones no comprobadas:* Excepción que se produce en el momento de la ejecución. También se denominan: Excepciones de tiempo de ejecución. Incluyen bugs, errores de lógica, uso indebido de API, entre otros.
* *Errores.*Estas no son excepciones en absoluto, sino problemas que surgen fuera del control del usuario o del programador. Los errores generalmente se ignoran en su código porque rara vez puede hacer algo al respecto. Por ejemplo, si se produce un desbordamiento de pila, surgirá un error. También se ignoran en el momento de la compilación.

Fuente :  [Tutorialspoint](https://www.tutorialspoint.com/java/java_exceptions.htm)

## **37.**¿Cuál es la diferencia entre una excepción y una excepción de tiempo de ejecución?

Esto está relacionado con el tipo de excepciones: las excepciones se verifican mientras que las excepciones de tiempo de ejecución no se verifican. Las excepciones marcadas requieren que maneje la excepción en una captura o declare el método como arrojando la excepción.  
*RuntimeExceptions* son una subclase de Exception. En general, las RuntimeExceptions son excepciones que se pueden evitar mediante programación. Por ejemplo: NullPointerException. No es necesario capturar estas excepciones.  
  
Fuente :  [Stackoverflow](https://stackoverflow.com/questions/2190161/difference-between-java-lang-runtimeexception-and-java-lang-exception)

## **38.**¿Necesita capturar una RuntimeException?

No. No es necesario detectar una excepción de tiempo de ejecución, pero como buena práctica general, se recomienda hacer algo al respecto. La mejor práctica sería identificar el origen de la excepción de tiempo de ejecución y solucionar el problema.

## **39.**¿Necesita agregar un lanzamiento en el método al crear una excepción de tiempo de ejecución?

No. RunTimeExceptions pertenece a la familia de excepciones no comprobadas, lo que significa que no es necesario gestionarlas. Por lo tanto, no hay necesidad de un intento/captura, ni de una declaración de lanzamientos.

## **40.**¿Qué es un error en Java?

Un Error es un tipo específico de Throwable, al igual que una Excepción. La clase Error se usa para representar problemas que están fuera del control de la aplicación habitual: errores de JVM, falta de memoria, problemas al verificar el código de bytes: estas son cosas que no debe manejar porque si ocurren, las cosas ya están tan mal que su código es poco probable que sea capaz de manejarlo con cordura.

## **41.**¿Qué son los genéricos en Java?

Generics en Java nos permite crear métodos y clases de tipos no específicos. Por lo tanto, el mismo método se puede utilizar para varios tipos de datos. De la misma manera, la misma clase puede contener diferentes tipos de datos.  
  
*Los métodos genéricos* son métodos que se pueden llamar con argumentos de diferentes tipos. Según los tipos de argumentos pasados ​​al método genérico, el compilador maneja cada llamada de método de manera adecuada. Ejemplo:  clase pública GenericMethodTest {  
  
  
   // método genérico printArray  
   public static < E > void printArray ( E [] inputArray ) {  
      // Mostrar elementos de la matriz  
      para ( elemento E : inputArray ) {  
         sistema \_ fuera \_ printf ( "%s" , elemento );  
      }  
      sistema \_ fuera \_ imprimirln ();  
   }

   public static void principal ( String args []) {  
      // Crear arreglos de Integer, Double y Character  
      Entero [] intArray = { 1 , 2 , 3 , 4 , 5 };  
      Double [] doubleArray = { 1.1 , 2.2 , 3.3 , 4.4 };  
      Carácter [] charArray = { 'H' , 'E' , 'L' , 'L' , 'O' };  
      sistema \_ fuera \_ println ( "Array integerArray contiene:" );  
      imprimirArray ( intArray ); // pasar una matriz de enteros    
      sistema \_ fuera \_ println ( "\nArray doubleArray contiene:" );  
      imprimirArray ( dobleArray ); // pasar una matriz doble    
      sistema \_ fuera \_ println ( "\nArray characterArray contiene:" );  
      imprimirArray ( charArray ); // pasar una matriz de caracteres

   }

*Las clases genéricas* parecen clases no genéricas, pero el nombre de la clase va seguido de una sección de parámetros de tipo. Ejemplo:

Caja de clase pública < T > {

   privado T t ;

   añadir vacío público ( T t ) {

      esto \_ t = t ;  
   }

   público T obtener () {

      devolver t ;  
   }

   public static void main ( String [] args ) {

      Box < Entero > integerBox = new Box < Entero >();  
      Caja < Cadena > cadena Caja = nueva Caja < Cadena >();

      IntegerBox . agregar ( nuevo Entero ( 10 ));

      caja de cuerdas . agregar ( nueva cadena ( "Hola mundo" ));

      sistema \_ fuera \_ printf ( "Valor entero:%d\n\n" , integerBox . get ());

      sistema \_ fuera \_ printf ( "Valor de cadena:%s\n" , stringBox . get ());  
   }  
}  
Fuente :  [Tutorialspoint](https://www.tutorialspoint.com/java/java_generics.htm)

## **42.**¿Qué es StackOverflow? ¿Cuándo se produce?

Stackoverflow es un error que se produce cuando la pila del programa se ha quedado sin memoria. La pila del programa almacena la dirección lógica de cada función llamada. Por lo tanto, si una función llama a una gran cantidad de funciones (como un bucle infinito o una función recursiva), la pila se quedará sin recursos y generará StackOverflowError.  
  
La causa común de un desbordamiento de pila es una mala llamada recursiva. Por lo general, esto se produce cuando las funciones recursivas no tienen la condición de finalización correcta, por lo que termina llamándose a sí mismo para siempre.  
Fuente :  [Stackoverflow](https://stackoverflow.com/questions/214741/what-is-a-stackoverflowerror)

## **43.**¿Qué es una función recursiva?

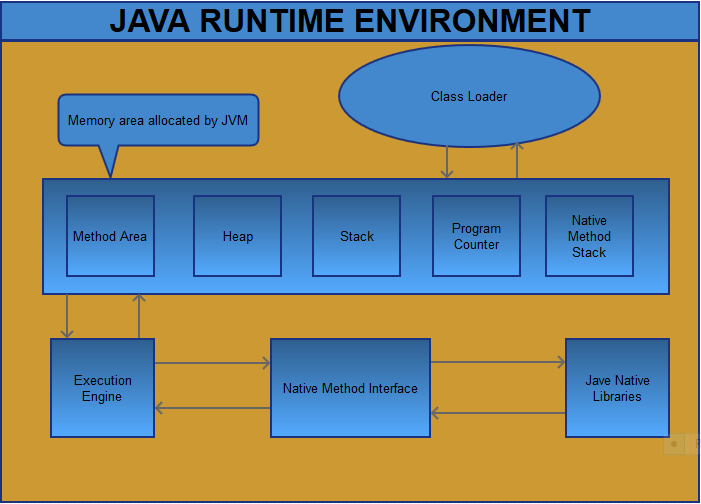
Una función recursiva es una función que se llama a sí misma. Un uso típico de las funciones recursivas es calcular el factorial de un número, las torres de Hanoi, el máximo común divisor, búsqueda binaria, entre otros.

## **44.**Además de la pila, ¿cuáles son los otros segmentos de la memoria en JVM?

Cuando una JVM ejecuta un programa compilado en Java, durante el tiempo de ejecución tiene 5 segmentos de memoria:  
  
*Registro de contador de programa (PC).*Este segmento almacena la dirección de memoria de las instrucciones de Java Virtual Machine (JVM) que se están ejecutando actualmente. Se crea un contador de programa cada vez que se crea un hilo nuevo. El contador del programa mantiene un puntero a las declaraciones actuales que se ejecutan en el subproceso actual.

*Área de método* : se utiliza para almacenar información diversa sobre el programa que se está ejecutando. La información incluye: información de tipo (ya sea una clase o una interfaz, modificadores de tipo, superclase, nombre totalmente calificado), un grupo de constantes (para almacenar literales), información de campo (nombre, tipo, modificadores) e información de método (nombre, valor de retorno). tipo, #, tipo y orden de parámetros, modificadores).

*Java Stack:* almacena datos específicos de subprocesos. Variables locales, referencias de objetos a un montón, etc. Cada vez que un subproceso ingresa a un nuevo método, se creará un nuevo bloque llamado marco de pila dentro de la memoria de pila para almacenar los detalles específicos del método. Cuando el subproceso complete la ejecución de ese método, el marco de pila correspondiente se eliminará de la memoria de pila.  
  
*Java Heap:* es el almacenamiento principal dentro de JVM y guarda todos los objetos nuevos creados durante el tiempo de ejecución. Heap mem es común para todos los subprocesos y tiene dos partes lógicas: Young y Old Generation. Young Generation almacena los objetos recién creados. Después de un tiempo, estos objetos pasan a la parte de generación anterior. Si la memoria total asignada no es suficiente para almacenar los nuevos objetos, JVM arrojará un java.lang.OutOfMemoryError.

[](https://javabeat.net/wp-content/uploads/2016/02/jvm-memory-allocation.png)

[**Fuente:**](https://www.blogger.com/blogger.g?blogID=8783164366670118553)*Pila de métodos nativos.*Similar a Java Stack, almacena los datos del subproceso para las llamadas a métodos nativos.  
  
Fuente:  [Javabeat](https://javabeat.net/jvm-memory/)

## **45.**¿Qué hace el Recolector de Basura?

El recolector de basura es un programa que se ejecuta en la máquina virtual Java y se deshace de los objetos que ya no están siendo utilizados por una aplicación Java. Es una forma de gestión automática de la memoria.  
  
Cuando se ejecuta una aplicación típica de Java, se están creando nuevos objetos, pero después de cierto tiempo, esos objetos ya no se usan. El recolector de basura buscará esos objetos y se deshará de ellos, liberando la memoria para que otros objetos nuevos puedan usar esa parte de la memoria.  
  
**Fuente:** [Stackoverflow](https://stackoverflow.com/questions/3798424/what-is-the-garbage-collector-in-java)

## **46.**​​¿Qué son los *hilos*? ¿Para qué se usan?

Un hilo es un hilo de ejecución en un programa. La máquina virtual de Java permite que una aplicación tenga varios subprocesos de ejecución que se ejecutan simultáneamente.  
  
Los subprocesos se utilizan en la programación concurrente para tener múltiples líneas de ejecución y múltiples partes del código ejecutándose simultáneamente, según las capacidades de la máquina.  
  
Fuente:  [Oráculo](https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html)

## **47.**¿Cuál es el uso de la palabra clave*sincronizado* en Java?

La palabra clave sincronizada de Java es una herramienta esencial en la programación concurrente en Java. Su propósito general es permitir solo un subproceso a la vez en una sección particular del código, lo que nos permite proteger, por ejemplo, las variables o los datos para que no se corrompan por modificaciones simultáneas de diferentes subprocesos.  
  
En su nivel más simple, un bloque de código que está marcado como sincronizado en Java le dice a la JVM: "solo permita un hilo aquí a la vez".  
  
Fuente:  [Javamex](https://www.javamex.com/tutorials/synchronization_concurrency_synchronized1.shtml)

## **48.**¿Cuáles son los dos tipos de sincronización compatibles con Java?

El lenguaje de programación Java proporciona dos modismos básicos de sincronización: *métodos sincronizados* y *sentencias sincronizadas* .  
  
*Los métodos sincronizados* se declaran agregando la palabra clave *sincronizada* en la declaración del método. Hacer que un método esté sincronizado tiene dos efectos:

* Primero, no es posible que se intercalen dos invocaciones de métodos sincronizados en el mismo objeto. Cuando un subproceso ejecuta un método sincronizado para un objeto, todos los demás subprocesos que invocan métodos sincronizados para el mismo objeto se bloquean (suspenden la ejecución) hasta que el primer subproceso termina con el objeto.
* En segundo lugar, cuando sale un método sincronizado, establece automáticamente una relación de suceso anterior con cualquier invocación posterior de un método sincronizado para el mismo objeto. Esto garantiza que los cambios en el estado del objeto sean visibles para todos los subprocesos.

*Las declaraciones sincronizadas* son otra forma de introducir la programación concurrente en Java. A diferencia de los métodos sincronizados, las declaraciones sincronizadas deben especificar el objeto que proporciona el bloqueo intrínseco.  
  
Fuente:  [Oráculo](https://docs.oracle.com/javase/tutorial/essential/concurrency/locksync.html)

## PREGUNTAS RELACIONADAS CON LA BASE DE DATOS

## 49. ¿Qué es una transacción en una base de datos?

Las transacciones son un grupo de una o más operaciones de base de datos que forman una sola unidad lógica (tienen un propósito específico).  
  
Encontraremos el caso en que no se puede lograr una operación lógica con una sola declaración de la base de datos (por varias razones, a saber: se deben actualizar varias tablas, se deben realizar cálculos, etc.). Pero, ¿qué sucede si una de las operaciones falla? Si persistimos con los cambios, eso nos llevará a un estado inconsistente en la Base de Datos.  
  
Las transacciones nos permiten agrupar todas estas operaciones en una sola unidad lógica que realiza todas las operaciones y confirma el resultado solo después de que se haya ejecutado la última operación, manteniendo así la coherencia en la base de datos.  
  
Si ocurre un error, la base de datos permite realizar un *rollback* , que devuelve la base de datos al estado inicial, antes de que la transacción comenzara su ejecución.  
  
Por ejemplo, supongamos que tenemos un supermercado. Tenemos el producto entidad que es un catálogo de los productos disponibles. Esta entidad contiene el atributo in\_stock que es el número de artículos que tenemos en stock.  
  
Cada vez que vendemos un ejemplar de este producto, almacenamos la cantidad que vendemos en otra entidad (llamémosla VENTA\_DETALLE) y tenemos que restar la cantidad vendida del atributo en\_stock. Por tanto, nuestra transacción consta de dos operaciones:

1. Almacenar la cantidad vendida en la entidad VENTA\_DETALLE.
2. Restando la cantidad vendida del atributo in\_stock.

Fuente:  [Oráculo](https://docs.oracle.com/database/121/CNCPT/transact.htm" \l "CNCPT016)

## **50.**¿Qué son los índices en una base de datos? ¿Para qué se usan? ¿Cuándo deben crearse?

Los índices en una base de datos se crean para mejorar la eficiencia de las búsquedas. Permiten una recuperación más rápida de registros en una base de datos.  
  
Los índices deben crearse siempre que sepamos que las búsquedas se realizarán en un campo específico. Si sabemos que el usuario estará realizando búsquedas en base a un campo específico, debemos crear un índice sobre ese campo para agilizar la recuperación de la información.  
  
Fuente:  [Techonthenet](https://www.techonthenet.com/oracle/indexes.php)  
  
[PREGUNTA ADICIONAL]

## **51.**¿Qué tipo de índices existen en Oracle?

Los índices se pueden clasificar de la siguiente manera:  
  
*Índices de árbol B:* estos índices son el tipo de índice estándar. Son excelentes para la clave principal y los índices altamente selectivos. Usados ​​como índices concatenados, los índices de árbol B pueden recuperar datos ordenados por las columnas indexadas. Los índices de árbol B tienen los siguientes subtipos:  
  
*Tablas organizadas por índice:*una tabla organizada por índice difiere de una organizada por montón porque los datos son en sí mismos el índice.  
  
*Índices de clave inversa:* en este tipo de índice, los bytes de la clave de índice se invierten, por ejemplo, 103 se almacena como 301. La inversión de bytes distribuye las inserciones en el índice en muchos bloques.  
  
*Índices descendentes:* este tipo de índice almacena datos en una columna o columnas en particular en orden descendente.  
  
*Índices de clúster de árbol B:*este tipo de índice se utiliza para indexar una clave de clúster de tabla. En lugar de apuntar a una fila, la clave apunta al bloque que contiene filas relacionadas con la clave de clúster.  
  
*Mapa de bits e índices de unión de mapa de bits:* en un índice de mapa de bits, una entrada de índice utiliza un mapa de bits para apuntar a varias filas. Por el contrario, una entrada de índice de árbol B apunta a una sola fila. Un índice de unión de mapa de bits es un índice de mapa de bits para la unión de dos o más tablas.  
  
*Índices basados ​​en funciones:*este tipo de índice incluye columnas que se transforman mediante una función, como la función SUPERIOR, o se incluyen en una expresión. Los índices de árbol B o mapa de bits pueden estar basados ​​en funciones.  
  
*Índices de dominio de aplicación:* este tipo de índice lo crea un usuario para los datos en un dominio específico de la aplicación. El índice físico no necesita utilizar una estructura de índice tradicional y puede almacenarse en la base de datos de Oracle como tablas o externamente como un archivo.  
  
  
Fuente *:*[oráculo](https://docs.oracle.com/cd/E11882_01/server.112/e40540/indexiot.htm" \l "CNCPT1170)