CURSO JAVA

El nombre del archivo debe de ser igual a la clase.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Atajos:

Sout: Genera método de impresión.

Psvm: Genera funcion main

Comentarios: Se realizan como en c, con //

VARIABLES

¿Qué es una Variable?

Una variable es un contenedor que almacena datos que pueden cambiar durante la ejecución de un programa. Cada variable tiene un nombre, un tipo de datos, y un valor.

Tipos de Variables en Java

1. Variables Primitivas:
   1. Enteros: byte, short, int, long
   2. Punto Flotante: float, double
   3. Carácter: char
   4. Booleano: boolean
2. Variables de Referencia: Almacenan referencias a objetos, cualquier clase o tipo definido por el usuario
   1. Cadenas: String. Ej “Karla”
   2. Arreglos: int[]
   3. Objetos de clase: Cualquier istancia de una clase definida por el usuario.

Declaración y Asignación de Variables

Sintaxis de Declaración: Para declarar una variable en Java, necesitas especificar el tipo de datos seguido del nombre de la variable, siendo la siguiente sintaxis: tipo nombreVariable; el valor se puede asignar al momento de la declaración o posteriormente.

Para imprimir las variables podremos usar el println y posteriormente el nombre de la variable, tiene cierta similitud con C.

FUNCIÓN DE LA RAM EN JAVA

Cuando ejecutas un programa Java, el manejo de variables y la gestión de memoria se realizan principalmente en la RAM. Aquí está una explicación detallada de cómo funciona este proceso:

1. Carga del Programa en Memoria: Cuando ejecutas un programa Java:

• Compilación: El código fuente Java (.java) se compila en bytecode (.class) por el compilador javac.

• Ejecución: La JVM (Java Virtual Machine) carga el bytecode en la memoria RAM para su ejecución.

2. Áreas de Memoria en la JVM dentro de la memoria RAM

La JVM gestiona la memoria en diferentes áreas de la RAM, cada una con su propósito

específico:

1. Stack:

-Es el área de memoria donde se almacenan las variables locales, así como los valores de variables primitivas definidas dentro de un método.

-El nombre de la variable almacena la dirección de memoria.

-Cada vez que se llama a un método, se crea un nuevo marco (frame) en el stack que contiene las variables locales y los datos necesarios para la ejecución del método.

-Cada vez que sobreescribimos los valores se modifican los valores en la misma dirección de memoria

2. Heap:

-Es el área donde se almacenan los objetos y sus datos asociados, los strings, listas y objetos de clase.

-La memoria en el heap es gestionada automáticamente por el recolector de basura (Garbage Collector).

-El valor que se almacena la variable no es el valor en si mismo sino la referencia al objeto, aquí de forma similar a C al modificar los valores se modifica una dirección de memoria nueva. Pues el nombre almacena la dirección y en la memoria heap se crea dicho objeto string posteriormente si modificamos se crea uno nuevo en la memoria heap y ahora en la memoria stack se modifica la memoria que se estaba almacenando.

Para modificar cadenas se puede hacer como en Python solo reasignando el valor de la cadena, aquí no soltara error.

Nota: Para imprimir con una cadena adicional existen distintos métodos, uno es el concatenación que se realiza mediante el operador +. Su sintaxis seria System.out.println("El tipo de byte es" + variable); aclaración si la variable es de un tipo diferente de string se convertirá en string para su impresión.

REGLAS Y BUENAS PRÁCTICAS EN NOMBRES DE VARIABLES EN JAVA

Los nombres de variables en Java son esenciales para la legibilidad y mantenibilidad del código. Seguir buenas prácticas y convenciones de nomenclatura ayuda a que el código sea más comprensible y profesional. A continuación, se detallan las reglas y buenas prácticas para nombrar variables en Java, con ejemplos.

Reglas Básicas para Nombres de Variables

1. Debe comenzar con una letra, un símbolo de dólar ($) o un guion bajo (\_):
   1. Correcto: nombre, \_nombre, $nombre
   2. Incorrecto: 1nombre, -nombre
2. No puede contener espacios ni caracteres especiales:
   1. Correcto: nombreCliente
   2. Incorrecto: nombre cliente, nombre-cliente
3. No puede ser una palabra reservada de Java:
   1. Correcto: int numero
   2. Incorrecto: int int
4. Distingue entre mayúsculas y minúsculas:
   1. nombre, Nombre, NOMBRE son variables diferentes

Buenas Prácticas

1. Usar Camel Case para Nombres de Variables:
   1. Utilizar camel case para nombres de variables, comenzando con minúscula y cada nueva palabra con mayúscula.
   2. Ejemplo: nombreCompleto, numeroDeTelefono
2. Ser Descriptivo y Claro:
   1. Los nombres de las variables deben describir claramente su propósito.
   2. Ejemplo: edadPersona, precioProducto
3. Prefijos y Sufijos Claros (si es necesario):
   1. Usar prefijos como is o has para variables booleanas.
   2. Ejemplo: isActivo, hasSaldo
4. Evitar Nombres de Variables de una Sola Letra:
   1. A menos que se usen en bucles pequeños o contextos muy específicos.
   2. Ejemplo: int i en un bucle for, pero preferir indice fuera de ese contexto.
5. No Abusar de Abreviaturas:
   1. Usar nombres completos siempre que sea posible para mayor claridad.
   2. Ejemplo: totalPiezas en lugar de totPzs
6. Usar Nombres Significativos Incluso en Variables Temporales:
   1. Asegurarse de que los nombres de las variables temporales también sean descriptivos.
   2. Ejemplo: sumaParcial en lugar de te

TIPO VAR EN JAVA

El tipo var fue introducido en Java 10 como una forma de inferencia de tipos en variables locales. Permite a los desarrolladores declarar variables sin especificar explícitamente su tipo, haciendo que el código sea más conciso y legible.

Sintaxis Básica de var

La sintaxis para usar var es muy sencilla. Simplemente reemplaza el tipo de la variable con var y deja que el compilador infiera el tipo basándose en el valor inicial asignado. La sintaxis es var variableNombre= valor;

Limitaciones y Reglas de Uso de var

1. Solo para Variables Locales:
   1. var solo puede ser usado en variables locales dentro de métodos, inicializadores de bucles, y bloques de inicialización.
   2. No puede ser usado para variables de clase, variables de instancia, o parámetros de métodos.
2. Debe ser Inicializada al Declararse:
   1. La variable debe ser inicializada en la misma línea donde se declara.
3. El Tipo Debe ser Inferrible:
   1. El compilador debe ser capaz de inferir el tipo de la variable a partir del valor asignado. No se le puede asignar null.
4. El tipo de dato nunca debe de cambiar.

Buenas Prácticas al Usar var

-Claridad del Código: Usa var solo cuando la inferencia del tipo sea obvia y clara para otros desarrolladores que lean el código. Ejemplo: var nombre = "Juan"; // Claro: el tipo es obviamente String

¿QUÉ ES LA CONCATENACIÓN DE CADENAS?

La concatenación de cadenas es el proceso de unir dos o más cadenas para formar una nueva cadena. En Java, existen varios métodos para concatenar cadenas, cada uno con sus propias características y usos.

Métodos de Concatenación en Java

1. Operador +

2. Método concat()

3. Clase StringBuilder

4. Clase StringBuffer

5. Método String.join()

1. Operador +

El operador + es el método más sencillo y común para concatenar cadenas en Java. Es fácil de usar y leer. Sintaxis: String resultado = cadena1 + cadena2; Ejemplo

String resultado= “Hola”+ “ ”+ “Mundo”

Cuando se concatena una cadena con un valor numérico, el valor numérico se convierte en una cadena y luego se une. Cuando una cadena es el primer operando, el operador + se comporta como un operador de concatenación. Si un número es el primer operando y se concatenan otros números sin paréntesis, se realizará la suma en lugar de la concatenación.

¿QUÉ ES UNA CONSTANTE?

Una constante en Java es una variable cuyo valor no puede ser cambiado una vez que ha sido asignado. Las constantes se utilizan para valores que se sabe que no cambiarán durante la ejecución del programa, proporcionando claridad y evitando errores. Sintaxis para Declarar Constantes: Para declarar una constante en Java, se utiliza la palabra clave final. Una vez asignado un valor a una variable final, no se puede cambiar. La sintaxis es la siguiente final tipo nombreConstante = valor;

El nombre de las constantes debe de estar en mayúsculas y generalmente si tiene varias palabras se separa por “\_”. Ademas de que siempre deben de inicializarse los valores.

Ejemplos de Declaración de Constantes

1. Constante de Tipo Entero: final int DIAS\_EN\_SEMANA = 7;

2. Constante de Tipo Doble: final double PI = 3.14159;

Para usar las constantes se usa como variables, solo se coloca el nombre y listo.

Beneficios de Usar Constantes

1. Claridad del Código:

o Las constantes hacen que el código sea más fácil de leer y entender, ya que se puede ver claramente cuáles valores no deben cambiar.

2. Mantenibilidad:

o Facilitan la actualización del código, ya que los valores que pueden cambiar se definen en un solo lugar.

3. Prevención de Errores:

o Reducen la posibilidad de errores accidentales al evitar cambios inadvertidos en valores críticos.

Mejores Prácticas al Usar Constantes

1. Nombres de Constantes en Mayúsculas:
   1. Es una convención común en Java escribir los nombres de las constantes en mayúsculas, usando guiones bajos para separar palabras.
   2. Ejemplo: MAX\_VALUE, URL\_BASE, TAMANIO\_BUFFER.
2. Uso de final:
   1. Asegúrate de que todas las constantes estén marcadas con la palabra clave final.
3. Declaración en una Clase Separada:
   1. Si tienes muchas constantes, considera declararlas en una clase separada para mantener el código organizado.

NOTA: Al imprimirse un string sin inicializar el programa mandara error.

¿QUÉ ES UNA CADENA EN JAVA?

Una cadena en Java es una secuencia de caracteres. Las cadenas se utilizan para almacenar y manipular texto. En Java, las cadenas son objetos de la clase String del paquete java.lang

Sintaxis para Crear Cadenas

Existen varias formas de crear cadenas en Java:

1. Literal de Cadena: String saludo = "Hola, Mundo!"; es la forma primitiva de crearla

2. Usando el Constructor de String: String saludo = new String("Hola, Mundo!"); la palabra new reserva espacio de memoria en heat y strings indica el tipo de objeto a crear. Es la sintaxis formal.

3. Cadenas de múltiples líneas: Se colocan tripes comillas para indicar el numero de líneas a generar, siendo que podremos ir colocando enter para cada línea y escribiendo en ellas, siendo que la impresión se realizara así.

MANEJOS DE INDICES DE CADENAS

Los índices de una cadena estan indexados de manera secuencial comenzando desde 0 hasta la longitud de la cadena menos 1, es decir el primer carácter está en el índice 0 de la cadena y el ultimo carácter en el índice n-1 donde n es el largo de la cadena, es como si fuera un arreglo.

Se puede acceder a dichos elementos del string a apartir de sus índices, para ello se utiliza el metodo charAT que nos permite obtener cada posición del índice. Para utilizarla se usa la siguiente sintaxis: cadena.charAT(índice\_a\_obtener). La obtención la podemos asignar a una variable o incluso solamente podemos colocarla en un println

INMUTABILIDAD DE CADENAS

Una vez que se crea una cadena, los caracteres dentro de ella no pueden ser modificados. Si se desea modificar una cadena se tiene que crear un nuevo objeto de tipo string y asignarlo a nuestra variable.

COMPARACION DE CADENAS

Para comparar dos cadenas obviamente se requiere de dos cadenas, se puede utilizar distintos formas.

Operador ==: Compara la dirección de memoria de las dos cadenas, si es el mismo da true en caso contrario da false, si las cadenas son iguales pero su dirección de memoria es diferente dará false.

Si se usa “new String” se reserva un nuevo espacio de memoria por lo que si generamos cadenas con distintos métodos y también con new string pero iguales en contenido, la igualdad del operador == nos dará false.

Método equal: Compara el contenido de las cadenas, siendo que su sintaxis es cadena1.equal(cadena2)

METODOS DE CADENAS

Método lenght: Se utiliza para obtener el largo de una cadena, se utiliza con la siguiente sintaxis “cadena.length()”

Método replace: Se utiliza para reemplazar una subcadena o caracteres, su sintaxis es “cadena.replace(“subcadena a remplazar”, “subcadena nueva”)”. Si reemplazas un carácter dentro de una cadena con "", en la mayoría de los lenguajes, la cadena se acorta y los caracteres siguientes se desplazan automáticamente a la izquierda.

Método toUpperCase(): Se utiliza para convertir toda la cadena en mayúsculas. Su sintaxis es “cadena.toUpperCase()”

Método toLowerCase(): Se utiliza para generar una cadena a partir de una cadena en minúsculas, su sintaxis es “cadena.toLowerCase”

Método trim: Se utiliza para eliminar espacios al inicio y final de la cadena, su sintaxis es “cadena.trim()” o también puede ser el método strip con misma sintaxis.

SUBCADENAS

Una subcadena es obtener una parte de una cadena original.

Existen varios métodos para trabajar con subcadenas. Métodos Principales para el Manejo de Subcadenas

1. substring(int beginIndex): Devuelve una nueva cadena que es una subcadena de esta cadena. La subcadena comienza en el índice especificado y se extiende hasta el final de la cadena.

2. substring(int beginIndex, int endIndex): Devuelve una nueva cadena que es una subcadena de esta cadena. La subcadena comienza en el índice especificado y se extiende hasta el carácter en el índice endIndex - 1.

3. indexOf(String str): Devuelve el índice de la primera aparición de la subcadena especificada en esta cadena. Si la subcadena no se encuentra, devuelve -1. Busca la primera aparición.

4. lastIndexOf(String str): Devuelve el índice de la última aparición de la subcadena especificada en esta cadena. Si la subcadena no se encuentra, devuelve -1. Busca la ultima aparición, si solo hay una devuelve esa.

REEMPLAZAR SUBCADENAS EN JAVA

Para realizarlo se debe usar en conjunto el método replace usando su configuración de buscar subcadenas es decir porq su sintaxis es cadena.replace(“cadena\_original”,”cadena nueva”)

INSERTAR SUBCADENA:

insert(int offset, String str): Inserta el string en la posición especificada, siendo que su sintaxis es la mencionada.

ELIMINAR SUBCADENA

delete(int start, int end): Elimina una subcadena de caracteres desde la posición start hasta la posición end (exclusiva)

MAS DE CONCATENACION

Como habíamos dicho hay varios métodos de realizar la concatenación de cadenas ademas del operador +.

El primero es el método concat(): Al igual que + concatena varias cadenas, el método concat() de la clase String concatena la cadena especificada al final de la cadena actual. Sintaxis: String resultado = cadena1.concat(cadena2); se puede usar concat con una cadena desde cero usando “”.concat(cadenas…), se pueden pasar mas de dos cadenas a dicho método separando por “,” las cadenas.

El segundo es la Clase StringBuilder: StringBuilder es más eficiente para concatenaciones repetitivas y en bucles, ya que es mutable y no crea nuevas instancias de cadena. Genera una sola cadena y no varias cadenas. StringBuilder en Java es una clase que se utiliza para almacenar secuencias de caracteres que se pueden modificar. Es una alternativa a la clase String, que es inmutable (es decir, no se puede cambiar una vez creada). StringBuilder permite hacer modificaciones directamente en la secuencia de caracteres sin crear un nuevo objeto string cada vez, lo cual mejora el rendimiento, especialmente cuando se realizan muchas modificaciones a una cadena. Su sintaxis es

var sb = new StringBuilder();

sb.append(cadena1);

sb.append(cadena2);

var resultado = sb.toString();

El método to.String es para obtener el resultado final de stringbuilder, generándose así la cadena final.

Clase StringBuffer: StringBuffer es similar a StringBuilder, pero es seguro para hilos (thread-safe) para varios procesos a la vez, lo que lo hace adecuado para entornos multihilo es decir que se ejcutan varios procesos se estan ejecutando al mismo tiempo, su sintaxis es:

var sb = new StringBuffer();

sb.append(cadena1);

sb.append(cadena2);

var resultado = sb.toString();

Método String.join(): String.join() es útil cuando se necesita unir una colección de cadenas con un delimitador, es decir que se va añadiendo algo entre ellas. Su sintaxis es var resultado = String.join(delimitador, cadena1, cadena2); El delimitador puede ser “” así siendo que se verán solo las cadenas unidas directamente.

Texto

El contenido generado por IA puede ser incorrecto.

CARACTERES ESPECIALES

Actúan de manera distinta cuando las mandamos a imprimir. Pracitcamente son los caracteres como \n. Algunos son los siguientes

\n salto de línea

\t tabulación

\’ comilla simple

\” comilla doble

Paquete: Archivo que almacena varias clases.

LEER DATOS

La clase Scanner es parte del paquete java.util y se utiliza para leer la entrada del usuario desde varias fuentes, como el teclado, archivos, cadenas y flujos de entrada.

Para leer datos ingresados por el usuario en Java, puedes utilizar la clase Scanner. Esta clase facilita la captura de datos, ya sean numéricos o de texto.

A continuación, te mostraré cómo hacerlo línea por línea y detallaré los métodos de la clase Scanner.

Paso 1: Importar la clase Scanner

Para utilizar la clase Scanner, primero debes importarla desde la librería java.util. Esto se hace al principio de tu archivo Java:

import java.util.Scanner;

Esta línea permite que tu programa utilice la clase Scanner.

Paso 2: Crear un objeto Scanner

Luego, debes crear un objeto de la clase Scanner. Este objeto se utiliza para leer la entrada del usuario desde el teclado (System.in):

Scanner scanner = new Scanner(System.in);

Aquí, scanner es el nombre del objeto que hemos creado. Puedes elegir cualquier nombre válido para tu objeto.

Paso 3: Leer datos del usuario

Ahora, puedes usar los métodos de la clase Scanner para leer diferentes tipos de datos ingresados por el usuario. Aquí tienes algunos de los métodos más comunes:

nextLine(): Lee una línea completa de texto.

next(): Lee la siguiente palabra (hasta que encuentra un espacio).

nextBoolean() Lee un valor booleano introducido por el usuario.

nextByte() Lee un valor byte introducido por el usuario.

nextDouble() Lee un valor double introducido por el usuario.

nextFloat() Lee un valor float introducido por el usuario.

nextInt() Lee un valor int introducido por el usuario.

nextLine() Lee una línea completa como un String.

nextLong() Lee un valor long introducido por el usuario.

nextShort() Lee un valor short introducido por el usuario.

Aquí tienes un ejemplo de cómo leer un nombre y una edad del usuario:

System.out.println("Ingrese su nombre: ");

String nombre = scanner.nextLine();

System.out.println("Ingrese su edad: ");

int edad = scanner.nextInt();

Buenas Prácticas

Limpiar el buffer: Después de leer un número con nextInt(), nextDouble(), etc., es recomendable limpiar el buffer antes de leer una línea siguiente con nextLine(), pues dichos next no consumen el salto de línea pero el nextline si. Cuando se llama a nextInt, nextDouble, etc., se lee el valor numérico, pero no se

consume el carácter de nueva línea (Enter) que el usuario presiona después de ingresar el valor. Esto deja el carácter de nueva línea en el buffer, y cuando se llama a nextLine, éste lee el carácter de nueva línea restante en lugar de la siguiente línea de entrada. Esto se puede arreglarse agregando una llamada a nextLine() después de leer el número o algún dato.

Otra forma es mediante la conversión de datos utilizando las clases wrapper (clases envolventes) que proporcionan una forma de utilizar tipos de datos primitivos como objetos. Estas clases se encuentran en el paquete java.lang y cada tipo primitivo tiene una clase wrapper correspondiente. A continuación, se muestra cómo utilizar estas clases wrapper para convertir datos ingresados desde la consola.

Clases Wrapper Disponibles en Java

1. Integer: Para convertir cadenas en enteros.

2. Float: Para convertir cadenas en números de punto flotante de precisión simple.

3. Double: Para convertir cadenas en números de punto flotante de doble precisión.

4. Boolean: Para convertir cadenas en valores booleanos.

5. Long: Para convertir cadenas en números largos.

6. Short: Para convertir cadenas en números cortos.

7. Byte: Para convertir cadenas en bytes.

8. Character: No tiene un método parse, pero puede ser utilizado para convertir un único carácter

Para usar estas clases se hace lo siguiente.

Se declara una variable string que se inicializara como la variable escaner con el método nextline. Posteriormente se declara una variable ya sea con tipo var o la clase que nosotros queramos y se inicializa usando la siguiente sintaxis “clase.parseTipoDatoPrimitivo(variableQueAlmacenaNextLine())”. Para verlo en ejemplo se puede ver así una conversión de tipo int, ejemplo:

String lectura=scanner.nextline()

var variable\_dato=Integer.Parseint(lectura)

Así se puede escribir, pero otra forma mejor elaborada y con menos líneas de código o mejor comprimido es de la siguiente forma

var variable\_dato=Integer.ParseInt(scanner.nextline())

Siendo asi mas detallado y con menor uso de variables.

Cerrar el Scanner: Siempre cierra el Scanner cuando ya no lo necesites para liberar los recursos, usando el método close. Para ello se debe de usar la siguiente sintaxis variable\_con\_objeto\_scaneer.close().