NOTAS JAVA:

Angular, Estudiar Frontend, Springboot, y estudiar JAVA para la empresa.

Cuantos videos voy a ver en una semana?:

5h 30min \* días(9 dias(3 semanas en la empresa)) = 49h 30 min

1 video = 15min, yo tardo = 40 min

15 min de video = 3 videos de 5 min(promedio de videos).

1 video de 5 min = yo tardo = 15min

En 1h = 4 videos.

En un dia: 5h(20 videos) 30min(2 videos).

5h 30min = 22 videos

49h 30min = x?

X= 198 videos

Que Es un programa?

* Un programa informatico es un conjunto de instrucciones que una vez ejecutadas realizaran una o varias tareas en una computadora

En programación dividos tareas complejas en tareas mas siples que nos ayuden a ejecutarlas

SENTENCIAS: Es la forma que tenemos de comunicarnos con la maquina. ORDENES.

Existen 10 lenguajes activo actualmente

Cuando programamos estamos introduciendo información o instrucciones que entienda. El computador solo entiende código-maquina, solo lo entiende la maquina, para el ser humano es muy difícil. Por lo cual los lenguajes nos ayudan a suplir esta función de tener que entender el código-maquina.

JAVA: es un lenguaje interpretado.

A la hora de programar, obviamente es texto plano.

IDE: Es el acrónimo de INTEGRATED DEVELOPMENT ENVIRONMENT: como el COV, es para aplicaciones de MAC.

Eclipe y Net pings son para JAVA, el ECLIPSE es el IDe mas extendido para JAVA.

Las ventajas de los IDE’s son muchas, cuando lo usemos vamos a encontrarnos que las líneas de código enumeradas, ventanas, nos mostrara algún error si lo hay.

HISTORIA JAVA:

Año 1991:

Bill Gats es el rey de los sistemas operativos con el Microsoft M.S. Dos 5.0

Kernel Linux

Video 5. Curso: INTRODUCCION JAVA:

JDK(JAVA DEVELOPED KIT)de java: INSTALAR JAVA:

1. PONER en el buscador JDF downloads. Clickeamos en Java SE downloads.
2. Diferencia entre JRE y JdK: JRE: Es el kit de desarrollo mientras que JRE: es un entorno de ejecución. En nuestro caso necesitamos.
3. Aceptamos la licencia.
4. Y escogemos el sistema operativo nuestro.
5. Nos pondrá Windows x86 o x64. El x64 es 64 bits y el x86 es 32 bits.

DESCARGAR DOWNLOADS INTELLIJ:

1. Poner en Google dwoloand intellij
2. Entrar en JETBRAINS
3. Descargar la versión community
4. Instalamos el programa: en la opción de CREATE DESKTOP SHORTCUT, ponemos la opción que tenga nuestro PC.
5. Damos en next todo hasta que cominece a instalar
6. Abrimos el INTELLIJ
7. Damos en la opción DO NOY IMPORT SETTING(no importar configuracion)
8. Damos ok
9. Si no nos abre el INTELLIJ, lo abrimos, aceptamos el DATA SHARING, la licencia,
10. Nos preguntara ya en el programa el tema, y damos todo en NEXT de nuevo, hasya dar a STAR USING INTELLIJ IDEA

Componentes básicos JAVA:

Creamos una clase en Intellij IDEA dando click derecho a src, para después poner java: crear clase.

Despues le ponemos el nombre a la clase y le damos aceptar.

Ejercicio Clase Java:

public class Main {  
  
// "comentarios para una linea". necesitamos agregar un metodo para poder ejecutar la aplicacion. Este metodo se conoce como el metodo MOINE  
 // ponemos psvm y le damos a tabulacion para poder que nos muestre la estructura del metodo.  
  
// Por el momento los conceptos basicos los veremos en otro proyecto u en otra clase, esto es para verificar si nustro primer codigo se ejecuta correctamente

public static void main(String[] args: Nos permite ejecutar el codigo  
  
  
 public static void main(String[] args) {  
  
 // SOUT: nos permite crear automaticamente una linea de mensaje  
  
 // IMPORTANTE PARA PODER EJECUTAR LA LINEA DE CODIGO, TENEMOS QUE DARLE CLICK AL ICONO VERDE COMO TRIANGULO VERDE AL REVES, LE DAMOS.  
 // A CLICK IZQUIERO Y LE DAMOS A "RUN NOMBREDELACLASE".  
 // SI NO NOS MUESTRA ESTE ICONO PODEMOS IR A LA CARPETA SRC, DAR CLICK DERECHO EN LA CLASE Y PONER "RUN"  
 // OTRA OPCION ES DARLE CLICK DERECHO AL CIDO MISMO Y PONER TAMBIEN "RUN NOMBREDELACLASE"  
   
 // System.out.println(); : Nos permite imprimir un string.  
  
 System.*out*.println("Hola mundo");  
  
 // TIP: OTRA FORMA ED CREAR ESTE SOUT Y OTROS CODIGOS MAS RAPIDO EN LA SINTAXIS, SIMPLEMENTE EN ESTE CASO PONEMOs  
 // "holamundo".sout y le damos a tabulacion para que nos cree el codigo. Asi:  
  
 System.*out*.println("HOla mundo de nuevo");  
  
 // METODO ELENE:  
  
 /\*  
 pARA PONER COMENTARIOS EN VARIAS LINEAS  
 \*/  
  
 }  
  
}

VARIABLES EN JAVA: Las variables en java son como en los demás lenguajes nos permiten almacenar daros en nuestro programa. El objetivo de declarar una variable es reservar espacio de memoria dependiendo del tipo que vayamos a utilizar. Las variables nos permitirán hacer programas dinámicos. Los valores cambiaran durante la interacción con el usuario y el programa, es decir los datos entrar en una variable y esta puede ser sujeto a cambio a lo largo del proyecto.

En java existen distintos tipos de datos, además se maneja una claficiacion amplia, pero en resumen, tendríamos el siguiente esquema:

Tipos de datos en java:

Tipo Ovie o también Tipos Refenciados: Podemos clasificarlos entre:

Clases, Interfaces y Arreglos

Tipo Primitivos: Podemos clasificarlos entre:

Tipos Enteros: Entre los cuales tenemos:

Tipo BYTE: ocupa 8 bits

Tipo SHORT: ocupa 16 bits

Tipo CHAR: ocupa 16 bits, pero maneja el código Unicode para almacenar valores de tipo CHAR. Una constante de tipo carácter se define como un carácter encerrado entre comillas simples. El valor de una constante de tipo carácter es el valor numérico de ese carácter en la tabla o código de caracteres. Es decir, un código o cierta cantidad de números representan un carácter.

Tipos Flotantes:

Tipo INT: ocupa 32 bits

Tipo Long: ocupa 64 bits

Tipo Flotantes: Entre los cuales tenemos:

Tipo Float: ocupa 32 bits

Tipo double: ocupa 64 bits

Tipo Boolean: Tambien es un tipo primitivo y puede ealmacenar solo el valor de True or False. Su valor por default es FALSE.

Los tipos primitivos son eficientes porque nos ayudan a ahorrar en recursos con el objetivo de que nuestros algoritmos sean los mas eficientes.

Cada uno de los tipos de los Primitivos tienen su propio valor default en JAVA, los cuales son:

Byte: 0

Short: 0

Char: \u0000

Int: 0

Long: 0

Float: 0.0

Double: 0.0

Los primitivos los usaremos mas que todo como atributos en las clases en JAVA.

MANEJO BASICO DE CADENAS STRINGS.

En java el manejo de cadenas es un tipo OBVIE o Referencia, con estas podemos crear objetos a base de clases, necesitaremos usar el operador NEW para crear un nuevo objeto, pero

AL usar el String en una variable nos permitirá delcarar que esa variable es un string, no confundir con los primitivos, pero no, el String no es un primitivo. Esto es debido a que el String se usa mucho y por lo tanto se decidio que se pueda poner al lado de una variable, ya que comúnmente al lado de las variables se coloca que tipo de dato primitivo que queremos que sea la variable: string saludo = “Hola Mundo”.

A diferencia de otros lenguajes no se permite la sobrecarga de operadores lo que significa que no podemos alterar la función de los operadores para hacer lo que nosotros deseemos. Sin embargo, existe una excepción a esta regla cuando manejamos tipos cadena , cuando usamos el operador “+” y se detecta un tipo string en la operación se dice que tenemos el “contexto string”, y por lo tanto en lugar de sumar valores lo que hace es concatenar(UNIR) los valores/Strings que se encuentran dentro de la operación.

IMPORTANTE: AL momento de usar cadenas en JAVA es importante tener ne cuenta mas adelante es la comparación de objetos en JAVA. Es esa sección solo aclararemos que al utilizar objetos no se utiliza el símbolo igual con lo mismo que nos pasaba con los primitvos, sino que samos el método “equals”. Recordemos lo usamos cuando vamos a comparar objetos que tienen cadenas y su contenido.

Ejm Comparacion: saludo.equals(“Hola Mundo”);

SE VERA MAS ADELANTE.

CARACTERES DE ESPACE AL UTILIZAR CADENAS: Son utilizados al momento de desplegar mensajes en la salida estándar.

\t: Inserta un tabulador

\b: Inserta un retroceso(backspace)

\n: Inserta una nueva linea

\r: Inserta un retorno de carro

\f: Se mueve a la siguiente pagina (Form feed). Se utiliza para impresoras, no en consola

\’: inserta una commila simple.

\”: Inserta una comilla doble.

\\: Inserta una barra invertida.

En JAVA, el nombre de las variables comúnmente empiezan por minúscula, sin embargo, si la variables tiene mas palabras dentro del nombre de la variable, entonces estas empiezan por Mayuscula, esto se le conoce como notación de camello o altas y bajas.

public class Main {  
  
 public static void main(String[] args) {  
  
// Si queremos repetir una cadena de string, para usarlo en diferentes partes del sistema.  
  
 /\*  
 Nos viene el inconveniete de que tendriamos que repetir varias veces la misma linea de codigo. Por lo cual,  
 usamos las variables, ejm:  
 \*/  
  
 String holaMundo = "Hola Mundo";  
 System.*out*.println(holaMundo);  
 System.*out*.println(holaMundo);  
 System.*out*.println(holaMundo);  
  
 // Para crear una cadena string en una variable ponemos: "Cadena de String".var . Despues nos dira que le indiquemos el nombre de la variable  
 // y le damos tabulacion de nuevo.  
  
 // String Cadena = "Cadena de String";  
  
 /\*  
 En las ultimas versiones de IAVA en lugar de definir una variable utlizando el tipo fijo, en este caso el string.  
 usamos "var" y esto en automatico va a crear una variable del tipo string. AL ponerle "var" al lado del nombre de la variable lo que conseguimos es  
 que el "var" consiga ver que tipo de dato es el valor de la variable.  
   
 Podemos agregar una linea de codigo que solo sea dar intro y que la siguiente linea de codigo se vaya abajo.  
 como un salto de pagina con simplemente dejar vacio donde va el string la variable. Ejm:  
   
 System.out.println();  
 \*/  
  
  
 var Cadena = "Cadena de String";  
 System.*out*.println(Cadena);  
   
 /\*  
 Si queremos que cuando cambiemos el nombre de la variable, se cambien el nombre de la variable donde teniamos colocada  
 el nombre de la variable en disntitas pares del codigo, ponemos en el nombre de la variable donde la declaramos, le damos click derecho  
 le damos a "REfactor" y despues le damos a "Rename".  
 \*/  
  
 var nombre = "Juan";  
 System.*out*.println(nombre);  
 nombre = "Carlos";  
 System.*out*.println(nombre);  
   
   
   
 /\* como vemos se puso nombre = "Carlos", si hacemos esto, estamos indicando que la variable "nombre", se ha cambiado  
 por Carlos, esto sin tener que modificar el tipo de dato. Simplemente estamos colocando un nuevo valor a la variable.  
 \*/  
  
 }  
  
}

CONCATENAR VARIABLES DE CADENAS EN JAVA:

Para concatenar en JAVA utlizamos el símbolo “+”. Dentro de SOUT. Ejm:

var usuario = "Juan";  
var saludar = "Hola";  
  
System.*out*.println(saludar + usuario);

// Saludos, mi nombre es Juan  
  
System.*out*.println("Saludos, mi nombre es " + usuario);

// TIP: Si queremos que cuando hagamos el SOAT ya nos salga con "concatenacion". Escribimos soutv. Lo que hara es tomar  
 // el nombre de la ultima variable que hemos declarado y la pondra en concatenacion con un strings que en el contenido  
 // tendra tambien el nombre de la variable.  
  
 System.*out*.println("saludar = " + saludar);

CARACTERES ESPECIALES EN JAVA:

Ejm:

\n: Salto de línea:

System.*out*.println("Nueva Linea: \n" + nombre2);

Nueva Linea:

Carla

\t: tabulación: un carácter de espacio amplio, que hace lo mismo que la tecla tabulación del teclado.

System.*out*.println("Nueva Linea: \t" + nombre2);

Tabulacion: Carla

\b: Retroceso

System.*out*.println("Nueva Linea: \b" + nombre2);

Retroceso:Carla

\r: Retorno de carro. Coloca el curso al lado izquierdo y después ejecuta lo que viene después.

Como vemos en el ejemplo no escribió “Retorno de carro:”, si no que directamente nos escribió la variable nombre2. Esto es porque puso el curso a la izquiera y ejecuto lo que venia después que era la variable.

System.*out*.println("Retorno de carro: \r" + nombre2);

Carla

\’: comillas simples

System.*out*.println("Comilla Simple: \'" + nombre2 + "\'");

\”: comillas dobles

System.*out*.println("Comilla Simple: \”" + nombre2 + "\”");

\’: comillas simples

System.*out*.println("Comilla Simple: \'" + nombre2 + "\'");

CLASE SCANNER EN JAVA:

// Con esta esamos importando a libreria donde esta el SCANNER para poder usarlo  
import java.util.Scanner;

Nos permite definir una variable para poder leer los valores que proporcione el usuario. Por ahora, Scanner la usamos en este ejercicio para leer información de la consola, y especificamos donde se encuentra este componente dentro de las Librearias de IAVA con system.in. pero mas adelante.

Scanner sc = new Scanner (System.*in*);

Scanner(Llamando a la clase de la libreria) sc(nombredelaVariableScanner) = new Scanner[otra vez llamando a la clase Scanner](System.in[para ingresar en la consola y poder escribir]);

IMPORTANTE: la clase en JAVA debe llamarse igual al nombre de la carpeta que lo junta. Ya que no si no es asi, las funciones y las llamadas de una librería nos dará error por este motivo.

// Al usar scanner.nextLine() estamos solicitando a la variable para que el valor de la variable se envie a Scanner,  
/\* y cuando ejecutremos el codigo, nos pedire en la consola que pongamos el valor de la variable. Como vemos es mediante el  
 Scanner podemos solicitarle a una variable que envie su valor mendiante scanner.nextline, y este valor se lo introducimos nosotros  
 en la consola \*/  
  
 var usuario = sc.nextLine();  
 String saludar = "Saludos";  
  
 System.*out*.println(saludar + " " + usuario);

// Un consejo es hacer un SOUT antes del nextLine en la variable que queremos solicitar que nos de información, en donde en este SOUT le indiquemos al usuario que introduzca su usuario para que el nextLine funcione.

TIPOS PRIMITIVOS: TIPOS INTEGER:

Dentro de ellos comenzamos por los tipo HINY:

TIPS:

To open any class in the editor quickly, press Ctrl+N (Navigate | Class) and start typing the name of the class. Choose the class from a drop-down list that appears.

You can open any file in your project in a similar way by using Ctrl+Mayús+N (Navigate | File).

The Code Completion feature lets you quickly complete different kinds of statements in the code. For example, start typing a class name and press Ctrl+Espacio to complete it. When multiple choices are available, they are shown in the lookup list.

You can quickly find all places where a particular class, method or variable is used in the whole project by positioning the caret at the symbol's name or at its usage in code and pressing Alt+F7 (Edit | Find | Find Usages in the popup menu).

To quickly see the documentation for a class or method at caret, press Ctrl+Q (View | Quick Documentation).

import jdk.swing.interop.SwingInterOpUtils;  
  
public class Integers {  
  
 public static void main(String[] args)  
 {  
 // byte, short, int, long: pueden recibir tanto valores positivos como negativos y veremos sus diferencias entre ellos.  
  
 // EXisten clases que nos permite observar la cantidad de bits que ocupa cada uno de estos tipos.  
  
 // Para el tipo BYTE tenemos lo siguiente:  
  
  
 // Tenemos la constante SIZE la cual nos permitira saber el tamañaño de este tipo de bit  
  
 // En este caso nos indicara el numero de bits que ocupa este string  
 // ESTAS SINTEXIS NOS PERMITITRAN VER LA CANTIDAD DE BYTES U OTRA CANTIDAD DE LOS TIPOS INTEGER.  
 // BYTE.SIZa: Nos permite ver la cantidad de bits que tiene un BYTE.  
 // BYTE.BYTES: Nos permite ver el tamaño de bytes que tiene un BYTE. En este caso obviamente  
 // solo nos va amostrar 1, porque 1 byte = 1 byte = 8 bits.  
  
 // BYTE  
  
 System.*out*.println("bits tipo BYTE: " + Byte.*SIZE*);  
 System.*out*.println("bytes tipo BYTE: " + Byte.*BYTES*);  
  
 // Byte.MIN\_VALUE y Byte.MAX\_VALUE: Nos permite ver el maximo valor positivo y minimo valor negativo que soporta este tipo BYTE.  
 // Como vemos nos dice que el MVP es 127 y MinimoVN es -128  
 System.*out*.println("valor maximo tipo byte: " + Byte.*MAX\_VALUE*);  
  
 System.*out*.println("valor minimo tipo byte: " + Byte.*MIN\_VALUE*);  
  
 System.*out*.println();  
  
 // SHORT  
  
 System.*out*.println("bits tipo short: " + Short.*SIZE*);  
 System.*out*.println("bytes tipo short: " + Short.*BYTES*);  
 System.*out*.println("valor maximo tipo short: " + Short.*MAX\_VALUE*);  
  
 System.*out*.println("valor minimo tipo short: " + Short.*MIN\_VALUE*);  
  
 System.*out*.println();  
  
 // INTEGER  
  
 System.*out*.println("bits tipo integer: " + Integer.*SIZE*);  
 System.*out*.println("bytes tipo integer: " + Integer.*BYTES*);  
 System.*out*.println("valor maximo tipo integer: " + Integer.*MAX\_VALUE*);  
  
 System.*out*.println("valor minimo tipo integer: " + Integer.*MIN\_VALUE*);  
  
 System.*out*.println();  
  
 // LONG  
  
 System.*out*.println("bits tipo long: " + Long.*SIZE*);  
 System.*out*.println("bytes tipo long: " + Long.*BYTES*);  
 System.*out*.println("valor maximo tipo long: " + Long.*MAX\_VALUE*);  
  
 System.*out*.println("valor minimo tipo long: " + Long.*MIN\_VALUE*);  
  
  
 }  
}

En la consola nos muestra:

bits tipo BYTE: 8

bytes tipo BYTE: 1

valor maximo tipo byte: 127

valor minimo tipo byte: -128

bits tipo short: 16

bytes tipo short: 2

valor maximo tipo short: 32767

valor minimo tipo short: -32768

bits tipo integer: 32

bytes tipo integer: 4

valor maximo tipo integer: 2147483647

valor minimo tipo integer: -2147483648

bits tipo long: 64

bytes tipo long: 8

valor maximo tipo long: 9223372036854775807

valor minimo tipo long: -9223372036854775808

// DECLAREMOS VARIABLE SCON ESTOS TIPOS INTEGERS:  
  
// EJm:  
  
byte byteVar = 15;  
System.*out*.println(byteVar);  
short shortVar = 100;  
System.*out*.println(shortVar);  
int intVar = 500;  
System.*out*.println(intVar);  
long longVar = 1000;  
System.*out*.println(longVar);  
  
// EN JAVA, EL DAFAULT PARA VAR SI EL VALOR DE LA VARIABLE ES TIPO INTEGER, ES INT  
  
var numero = 10;  
  
System.*out*.println(numero);  
  
// Si queremos convertir este default en algun otro tipo de integer, ponemos al lado derecha del valor de la variable  
// una letra mayuscula, la cual va a hacer la primera letra de cada tipo de INTEGER. ejm:  
  
var numero2 = 10L;  
  
System.*out*.println(numero2);  
  
// Aqui le estamos indicando que ya no sea INT sino que al ponerle la "L" de LONG, se convierte esta variable en tipo LONG.  
  
  
// CON LOS TIPO INTEGER podemos usar otros sistemas de decimales, como lo son decimales, hexadecimal y octal:  
  
byte decimal = 10;  
System.*out*.println("decimal = " + decimal);  
  
byte hexadecimal = 0xa;  
System.*out*.println("hexadecimal = " + hexadecimal);  
  
byte octal = 012;  
System.*out*.println("octal = " + octal);

public class Flotantes {  
  
 public static void main(String[] args) {  
  
 // Tipos flotantes  
  
 // ESTAS SINTEXIS NOS PERMITITRAN VER LA CANTIDAD DE BYTES U OTRA CANTIDAD DE LOS TIPOS INTEGER.  
  
 // Tipo FLOAT, DOUBLE  
  
 // A difernecia de los tipos INTEGERS, los tipos FLOTANTES incluyen una fraccion en su valor, es decir "decimales".  
  
// FLOAT  
  
 System.*out*.println("bits tipo float: " + Float.*SIZE*);  
 System.*out*.println("bytes tipo float: " + Float.*BYTES*);  
  
 // Byte.MIN\_VALUE y Byte.MAX\_VALUE: Nos permite ver el maximo valor positivo y minimo valor negativo que soporta este tipo BYTE.  
 // Como vemos nos dice que el MVP es 127 y MinimoVN es -128  
 System.*out*.println("valor maximo tipo float: " + Float.*MAX\_VALUE*);  
  
 System.*out*.println("valor minimo tipo float: " + Float.*MIN\_VALUE*);  
  
 System.*out*.println();  
  
 // DOUBLE  
  
 System.*out*.println("bits tipo double: " + Double.*SIZE*);  
 System.*out*.println("bytes tipo double: " + Double.*BYTES*);  
  
 // Byte.MIN\_VALUE y Byte.MAX\_VALUE: Nos permite ver el maximo valor positivo y minimo valor negativo que soporta este tipo BYTE.  
 // Como vemos nos dice que el MVP es 127 y MinimoVN es -128  
 System.*out*.println("valor maximo tipo double: " + Double.*MAX\_VALUE*);  
  
 System.*out*.println("valor minimo tipo double: " + Double.*MIN\_VALUE*);  
  
 System.*out*.println();  
  
 // IMPORTANTE Los DEFAULT de los tipos FLOTANTES en JAVA son DOUBLE, por lo cual auqnue al declarar una variable  
 // le pongamos FLOAT en la variable nos dara error, porque JAVA entiende el valor que le demos oomo DOUBLE. ejm:  
  
 // float floatVar = 0.5; Como vemos nos da error, por lo cual para solucionar esto es simplemente poner como hicimos  
 // con los INTEGERS, es poner en el valor de la variable la primera letra del tipo de FLOTANTES que queremos, en este caso  
 // seria "F" de FLOAT.  
  
 float floatVar2 = 0.5F;  
 double doubleVar = 0.5D;  
  
 // Si ponemos var en las variables que queremos que sean TIPOS FLOTANTES, tenemos que tener en cuenta que DEFAULT con var  
 // es double, por lo cual si vamos a usar var, es mejor indicar que el valor es float con "F".  
  
 var floarVar3 = 0.05F;  
  
  
  
  
  
  
  
 }  
}

public class Char {  
  
 public static void main(String[] args) {  
 // CHAR  
  
 // RECORDEMOS: Los valores del char se basan en la tabla UNICODE para repsentar cada uno de sus valores  
  
 // Usamos character para indicar el CHAR  
  
 System.*out*.println("bits tipo char: " + Character.*SIZE*);  
 System.*out*.println("bytes tipo char: " + Character.*BYTES*);  
 System.*out*.println("valor maximo tipo char: " + Character.*MAX\_VALUE*);  
  
 System.*out*.println("valor minimo tipo char: " + Character.*MIN\_VALUE*);  
  
 System.*out*.println();  
  
 // Para CHAR usamos comillas simples '' al momento de asignarle su valor, hay dos formas de poner el valor de un char  
 // con codigo y con numeros normales. COn codigo para que funciones ponemos \oodigoCHAR.  
  
 char c = '\u0021'; // este codigo representa "!"  
  
 System.*out*.println("c = " + c);  
  
 // si lo hicieramos con numeros, pero sin comillas simples n habria ningun problema:  
  
 char c2 = 33;  
  
 System.*out*.println("c2 = " + c2);  
  
 // Aunque hay una tercera opcion y es simplemente colocando el caracter que queremos usar, poniendolo en las  
 // comillas simples, ademas podemos poner var pero solamente en este opcion y en la opcion del codigo, no en la de  
 // los numeros porque el DEFAULT de en ese caso seria INT no char obviamente.  
  
 char c3 = '!';  
  
 System.*out*.println("c3 = " + c3);  
  
 }  
  
}

TIP: para poder ejecutar una línea de código en JAVA, dar click izquierdo en el numero de línea que quieras, se pondrá un circulo rojo sobre ella. Poner RUN y te mostrara un panel en donde te mostrara por cada vez que ejecutes una nueva línea, te mostrara el resultado de esa línea de código.

CONVERSION DE DATOS TIPO PRIMITIVOS:   
import java.util.Scanner;  
  
public class ConversionPrimitivos {  
  
 public static void main(String[] args) {  
  
  
 // convertir tipo STRING a tipo primitivo  
 // PARSE: Permite convertir cadenas de string en algun tipo de primitivo.  
 // Su estructura es nombreTipoprimitivo.parseNomenclaturaTipoPrimitivo(s: "Cadena de string")  
 // podemos hacer el parse con los tipos primitvos y con los flotantes tambien: algunos son:  
 // parseFloat, parseLong, parseBoolean, etc. La excepcion seria el CHAR  
 // EJM con tipo INT:  
  
 int edad = Integer.*parseInt*("20");  
 System.*out*.println("edad = " + edad);  
   
 // EJM con tipo DOUBLE  
   
 double valorPi = Double.*parseDouble*("3.14");  
 System.*out*.println("valorPi = " + valorPi);  
  
 // Si queremos que la cadena de texto se convierta en char, tenemos que tener en cuenta que el CHAR es para caracteres  
 // no para cadenas, pero si queremos sacar un caracter de una cadena, entonces usamos el metodo chartAT  
  
 //charAT: NOs permite extraer de una cadena, un caracter. Para usarlo, hay que tener en cuenta que cada caracter tien su  
 // propia posicion, es decir empieza desde cero en la primera letra, y va aumentando sucesivamete.  
  
 // EJM: si queremos extraer de una cadena que sea "Hola" el caracter "o", entonces hacemos:  
  
 char c = "Hola".charAt(1);  
 System.*out*.println("c = " + c); // c = o;  
  
 // SCANNER con TIPOS PRIMITVOS:  
  
 // por ahora sabemos que el scanner lo usamos on strings, pero podemos hacerlo tambien con primitivos  
 // Lo cual haremos:  
  
// declaramos la variable SCANNER  
 Scanner sc = new Scanner(System.*in*);  
  
 // declaramos la variable primitiva, en este caso INT, y usamos parse para que lo que introduzcamos dentro del parse  
 // se convierta de STRING a INT.  
 // La diferencia ahora es que dentro del PARSE vamos a colocar el metodo NEXTLINE, con el que solictiabamos  
 // a la variable que nos introdujera el valor para que el SCANNER lo colocara.  
 // Como vemos lo que hacemos es que mediante el SCANNEr ponemos un string,y este string se convierte en INT mediante el PARSE  
 // si escribieramos en el SCANNER letras, nos daria error porque ya con PARSE nos lo convirtio en INT  
  
 int edad2 = Integer.*parseInt*(sc.nextLine());  
 System.*out*.println("edad2 = " + edad2);  
  
 // Ahora con CHAr que recordemos era con charAT. Donde vemos que simplemente el nextLint() sumple la posicion del string,  
 // obviamente ya que lo pondremos en consola, y con el .charAT() despues de este, le estamos indicando al string que extraiga  
 // un caracter de la cadena de string.  
  
 char caracter = sc.nextLine().charAt(0); // en este caso el primer caracter de la cadena de string  
 System.*out*.println("caracter = " + caracter);  
  
 // CONVERSION TIPO PRIMITIVO A STRING  
 // VALUE OF: Nos permite convertir un tipo primitivo en String. Es un metodo de STRING  
 // EJM:  
  
 String valorPienTexto = String.*valueOf*(3.14); // vamos a convertit 3.14(que es DOUBLE). en string.  
 System.*out*.println("valorPienTexto = " + valorPienTexto);  
  
  
 // Otra forma de hacerlo mas rapido, es simplemente con concatenacion, es decir, cuando le ponemos a la variable un valor  
 // en forma de cadena, por ejemplo:  
  
 String valorPienTexto2 = "" + valorPienTexto; // al hacer esto, como vemos colocamos al inicio del valor de la variable un string vacio  
 // y lo concatenamos con +. Lo que hara esto es que lo que le siga se convertira automaticamente en STRING, ya que la consola  
 // lo leera asi.  
 System.*out*.println("valorPienTexto2 = " + valorPienTexto2);  
  
 }

public class Boolean {  
  
 public static void main(String[] args) {  
  
 // BOOLEAN  
  
 // RECORDEMOS: nos permite poner TRUE or FALSE como sus constantes y valores.  
  
  
  
 // System.out.println("valor TRUE Boolean: " + Boolean.TRUE);  
  
 // System.out.println("valor FALSE Boolean: " + Boolean.FALSE);  
  
 System.*out*.println();  
  
 boolean booleanVar = false;  
  
 // Aunque no hemos visto los condicionales en JAVA podemos hacer uno para comprobar la variable de bollean

// como vemos en un condicional si en el if(nombrevariable), solo tiene el nombre de la variable, al ser boolean si dentro del condicional no le indicamos nada, siempre será true, es decir que si.

El condicional dira (si solo esta el nombre de la variable, entonces es true).  
  
 if(booleanVar)  
 System.*out*.println("la variable tiene un valor TRUE");  
  
 else  
  
 System.*out*.println("la variable tiene un valor FALSE");  
  
 // EJErcicio con BOOLEAN para verificar si esa persona es adulta o no.  
  
 // hacemos una variable que contenga la edad de la persona  
  
 int edad = 23;  
  
 // como vemos dentro una variable tambien podemos poner signos de operacion para crear operaciones o indicaciones en este  
 // caso. IMPORTANTE: Como vemos var adulto es un boolean. PORQUE?. POrque al darle una indicacion o mini condicion dentro de la variable  
 // lo que genera es boolean de que esa indicaicon es TRUE, en este caso al usar los signos >=, le estamos indicando  
 // es mayor o igual y esto se vuelve TRUE. Digamos que fuera !=(no es igual), tambien seria TRUE. Da igual que indicacion le pongamos, será true a menos que le indiquemos lo constrario en un condicional.  
 // estamos asegurando siempre que es TRUE, a menos de que le digamos lo contrario. En este caso con el condicional "if". hacemos  
 // esta tarea.  
 boolean adulto = edad >= 18;  
  
 if (adulto)  
 System.*out*.println("Eres mayor de edad, puede pasar");  
 else  
 System.*out*.println("Eres menor de edad, no puedes pasar");  
  
 }  
}

Julio 9/2019

public class Operadores {  
  
 /\*  
 Operadores aritmeticos: +, -, \*, /, %(operacion modulo).  
Operadores de relaacion: <, >, >=, >=, !=, ==, = (como en PHP).

Operadores logicos: && o &, || o |, !(negacion), ^(exord a nivel de bits)  
 Operadores Unitarios: ~,  
 Operadores a nivel de bits:  
 Operadores de asignacion:  
 Operador COndicional:  
 Prioridad y orden de evaluacion: Son aquellos simbolos que se utitlizan para mirar la "jerarquia de un codgio y otro"  
  
 PALABRAS RESERVADAS: Ten en cuenta que si vamos a darle nombre a una variable, ten cuidado con ponerle alguna lpalabra que esta  
 predefinida en JAVA  
 \*/  
  
 public static void main(String[] args) {  
 // Operadores Artimeticos  
  
 // SUMA: "+"  
  
 // IMPORTANTE: EL VAR, no lo podemos usar cuando usamos una COMPOSICION, es decir,  
 // una linea de codigo con mas de una variable. Porque?. Porque puede que otra variable sea de otro tipo.  
 // JAVA nos dice que tenemos que especificar que tipo son esas variables.  
  
 int a=3, b=2; // Como vemos con ',' podemos separar las variables en una misma linea de codigo  
  
 // Ahora efectuamos la operacion pero es importante tener en cuenta que lo tenemos que poner en una variable  
 // para poder imprimirlo despues, tambien lo podemos poner en un metodo como en PHP. pero mas adelante.  
 // ademas de que el tipo debe ser como sus anteriores variables de la operacion obviamente.  
  
 int suma = a + b;  
  
 System.*out*.println("Resultado = de a + b = " + suma);  
  
 /\*  
 IMPORTANTE: Se pued eponer una operacion dentro de un SOUT?. Si, pero es importante que tengas  
 en cuenta que si no especificamos cual es la operacion, nos lo va a concatenar todos, porque como sabemos  
 usamos para concatenar en JAVA el "+", ademas de que si tenemos un string al lado, JAVA lo entendera como  
 una cadena de STRING, por lo cual para especifcar la operacion usamos "()".  
 \*/  
  
 System.*out*.println("Resultado 2 = " + (a + b));  
  
  
 /\*  
 Aunque es IMPORTANTE tambien que tengamos en cuenta que JAVa lee el codigo en CASCADA, por lo cual si ponemos  
 en el SOUT, el string a lo ultimo, no tendriamos que especificar cual es la operacion con "()".  
 \*/  
  
 System.*out*.println(a + b + " Resultado");  
  
 // RESTA  
  
 int resta = a - b;  
 System.*out*.println("resta Resultado = " + resta);  
  
 // o tambien:  
  
 System.*out*.println("resta = " + (a - b));  
  
 // MULTIPLICACION  
  
 int multiplicacion = a \* b;  
 System.*out*.println("resultado multiplicacion = " + multiplicacion);  
  
 // DIVISION  
  
 int division = a / b;  
 System.*out*.println("division = " + division);  
  
 // CUESTION: Si usamos otro tipo(por ahora son operaicone snumericas), como por ejemplo un DOUBLE  
 // entonces dentro de la variable que contendrá la operación o en unas de las variable que hace parte de la operación  
 // especificamos que tipo es, si es flotantes o primitivos, en este caso queremos que sea double.  
 // Por lo cual podemos especificar que a = 3D, o en sumar = 3D + b;  
  
 Double numeroDouble = 3.5;  
  
 // Como vemos nos dará error si en la variable de la operación le damos INT, ya que tenemos que poner DOUBLE  
  
 Double sumaConDouble = numeroDouble + b;  
  
 System.*out*.println("sumaConDouble = " + sumaConDouble );  
  
 // o tambien:  
  
 Double sumaConDouble2 = 3.5D + b;  
  
 System.*out*.println("sumaConDouble2 = " + sumaConDouble2 );  
  
 // MODULO: Nos regresa el residuo de una division  
  
 int modulo = a % b;  
 System.*out*.println("Modulo = " + modulo );  
  
 // Para saber si un numero es IMPAR o PAR, usamos "%" y lo divimos entre 2. En donde si nos regresa un 1= es impar, si es un 0=PAR  
  
 int parImpar = 5 % 2;  
  
 System.*out*.println('5 es : ' + parImpar);  
  
  
 // Podemos aplicar un condicional simple, indicando si es impar o par.  
  
 if(parImpar == 0)  
 System.*out*.println("El numero es par");  
  
 else  
 System.*out*.println("El numero es impar");  
 }  
  
}

EJERCICIO:

1. import java.util.Scanner;
3. public class Main {
4. public static void main(String[] args) {
5. Scanner scanner = new Scanner(System.in);
6. // Escribe tu solucion aqui
7. System.out.println("Proporciona el alto:");
8. int alto = Integer.parseInt(scanner.nextLine());
9. System.out.println("Proporciona el ancho:");
10. int ancho = Integer.parseInt(scanner.nextLine());
11. int area = alto \* ancho;
12. int perimetro = (alto + ancho) \* 2;
13. System.out.println("Area: " + area);
14. System.out.println("Perimetro: " + perimetro);
15. }
16. }

OPERADORES DE ASIGANCION:

public class OperadoresAsignacion {  
  
 public static void main(String[] args) {  
 // OPERADORES DE ASIGNACION  
  
 // ASIGNACION: igualamos/asignamos  
  
 int a = 3, b = 5;  
  
 // como vemos estamos asigando una variable igual a otra  
  
 int c = a;  
  
 // COMPOSICION: NOS PERMITE AGREGAR ALGO A LA VARIABLE. +=, -=, \*=, /=  
 // EJM:  
 a += 1; // Aqui le decimos que: como sabemos a = 3, y al ponerle +=. Le decimos que le agregue dependiendo que signo de operacion u otro tenga  
 // en este caso que le sume 1. Con AGREGAR me refiero a efectuar o implementar algo sobre una variable, pueden ser operadores aritmeticos,  
 // logicos, etc.  
  
 System.*out*.println("a + 1" = a); // Como vemos es como si le dijeramos: a = a + 1.  
  
 // Nos permite ahorrar codigo, para que sea mas optimo.  
  
  
  
  
  
 }  
  
}

Operador UNARIO:

public class OperadoresUnarios {  
  
 public static void main(String[] args) {  
  
 // OPERADORES UNARIOS:  
  
 // "~": Permite cambiar de signo a un variable INT. Invierte de positivo a negativo en este caso  
  
 int a = 3;  
 int b = ~a;  
  
 System.*out*.println("a negativo = " + b);  
  
 // OPERADOR TIPO BOOLEAN: En el caso del BOOLEAN, si ponemos "!", seria lo mismo que con "~". Invierte el valor  
 // EN este caso de TRUE a FALSE  
  
 boolean c = true;  
 boolean d = !c;  
  
 System.*out*.println("c Ahora en false = " + d);  
  
 // PRE-Incremento y PRe-Decremento: Va a incrementar o decrementar en 1  
  
 // PRE-Incremento  
  
 int preIncremento = 3;  
 int preIncremento2 = ++preIncremento;  
  
 // PRE-Decremento  
  
 int preDecremento = 3;  
 int preDEcremento2 = --preDecremento;  
  
  
  
 // POST-Incremento y POST-Decremento: Va a incrementar o decrementar en 1  
  
 // POST-Incremento  
  
 int postIncremento = 3;  
 int postIncremento2 = postIncremento++;  
  
 // POST-Decremento  
  
 int postDecremento = 3;  
 int postDecremento2 = postDecremento--;  
  
   
  
 }  
}

OPERADORES DE IGUALDAD Y RELACIONAL:

public class OpIgualdadRelacional {  
 public static void main(String[] args) {  
  
  
 // OPERADORES DE IGUALDAD  
  
 // EN este caso nos dara en SOUt false, porque a diferencia de los ejercicios anteriores explicados en la clase "boolean".  
 // Aqui ya estamos poniendo de iuna variable boolean, otras variables dentro de el. Por lo cual cambia el tema de que siempre es TRUE.  
 // Lo que sucede en el siguiente ejemplo, es que al imprimir boolean c. Este se pregunta si la operacion de igual es cierta o no.  
 // ya que no lo toma como simepre TRUe a menos de que se lo señalemos en un condicional,  
 // En este caso se pregunta si las 2 variables son iguales, si lo son pone TRUe y si no pone FALSE, que es loque queremos que indique  
 // TAMBIEN AGREGAR QUE AL USAR BOOLEAN DE ESTA FORMA: ES COMO SI HICIERAMOS UN MINI CONDICIONAL, PAR QUE NOS SALGA TRUE OR FALSE  
  
 int a = 3, b = 2;  
 //Consejo: ponerlo entre parentesis para no tener problemas con la lectura del SOUT, en algun momento.  
 boolean c = (a == b);  
  
 System.*out*.println("c = " + c);  
  
 // "==" Nos permite decir que un valor es igual totalmente a otro valor, a direfencia del "="  
 // que es para asignar un valor. El "==" lo usamos solo para tipos primitivos  
  
 // El equivalente de "==" para usarlo en TIpo Strings es el metodo "equals()"  
 // EQUALS(): Nos permite comparar si los Strings son iguales o no. Si son iguales es TRUE y no es, entonces es FALSE.  
  
 String d = "Hola", e = "HOla";  
  
 boolean f = d.equals(e);  
 System.*out*.println("f = " + f);  
  
  
 // OPERADORES RELACIONALES: Operadores de relaacion: <, >, >=, >=  
  
 // ejemplo con boolean pero ya no con koperadores de igualdad sino de relacionales  
  
 boolean g = (a > b);  
 System.*out*.println("g = " + g);  
  
 boolean h = (a > b);  
 System.*out*.println("h = " + h);  
  
 boolean i = (a <= b);  
 System.*out*.println("i = " + i);  
  
 boolean j = (a >= b);  
 System.*out*.println("j = " + j);  
  
  
 }  
}

}

OPERADORES CONDICIONALES:

public class OperadoresCondicionales {  
  
 public static void main(String[] args) {  
  
 // OPERADORES CONDICIONALES: Son aquellos signos que usamos en un condicional. (OPERADORES DE CORTO CIRCUITO)  
  
 // && = "y". ambos son verdades  
  
 // EJECIRCIO Hacer que un numero este dentro de un rango esperado, por ejemplo dl rango de 0 a 10  
  
 int a = 1;  
 int valorMinimo = 0, valorMaximo = 10;  
 boolean resultado = a >= valorMinimo && a <= valorMaximo;  
 System.*out*.println("a = " + resultado);  
  
 // AUNQUE tambien se puede hacer en un condicional  
  
 if (a >= valorMinimo && a <= valorMaximo)  
 System.*out*.println("Dentro de rango");  
 else  
 System.*out*.println("Fuera de rango");  
  
 // || o | = "o". si uno o otro es verdadero  
  
 boolean vacaiones = true;  
 boolean diaDescanso = false;  
  
 if (vacaiones || diaDescanso)  
 System.*out*.println("Tengo dia libre");  
  
 else  
 System.*out*.println("No tengo dia libre");  
 }  
  
}

OPERADOR TERNARIO:

public class OperadorTernario {  
 public static void main(String[] args) {  
  
 // OPERADOR TERNARIO: Nos sirve paa realizar operaciones tipo IF ELSE sin necesidad de usar estos dos, es decir,  
 // cumple la misma funcion que IF ELSE pero sin usar estos dos terminos  
  
 // ejercicio de par e impar:  
  
 int numero = 10;  
  
 // ?= Abre la condicion, cumple la fuincion de IF  
 // := Cumple la funcion de ELSE  
  
 // Como vemos ? seria If, : seria ELSE y los string "". Serian lo que hemos hecho hasta ahora que es colocar sout's, auque podemos poner lo que queremos  
 // como un true or false por ejemplo, y mas cosas que veremos mas adelante  
 // ESTRUCTURA: variable = (OPERACION o condicion que queremos) ? "" : "";. Los () son opcionales  
  
 var resultado = (numero % 2 == 0) ? "El numero es par" : "El numero es impar";  
  
 System.*out*.println("resultado = " + resultado);  
 }  
}

EJERCICIO TERNARIO:

1. import java.util.Scanner;
3. public class Main {
5. public static void main(String[] args) {
6. Scanner scanner = new Scanner(System.in);
7. System.out.println("Proporciona el numero1:");
8. int numero1 = Integer.parseInt(scanner.nextLine());
9. System.out.println("Proporciona el numero2:");
10. int numero2 = Integer.parseInt(scanner.nextLine());
11. System.out.println("El numero mayor es:");
12. System.out.println(numero1 > numero2 ? numero1 : numero2);
13. }
14. }

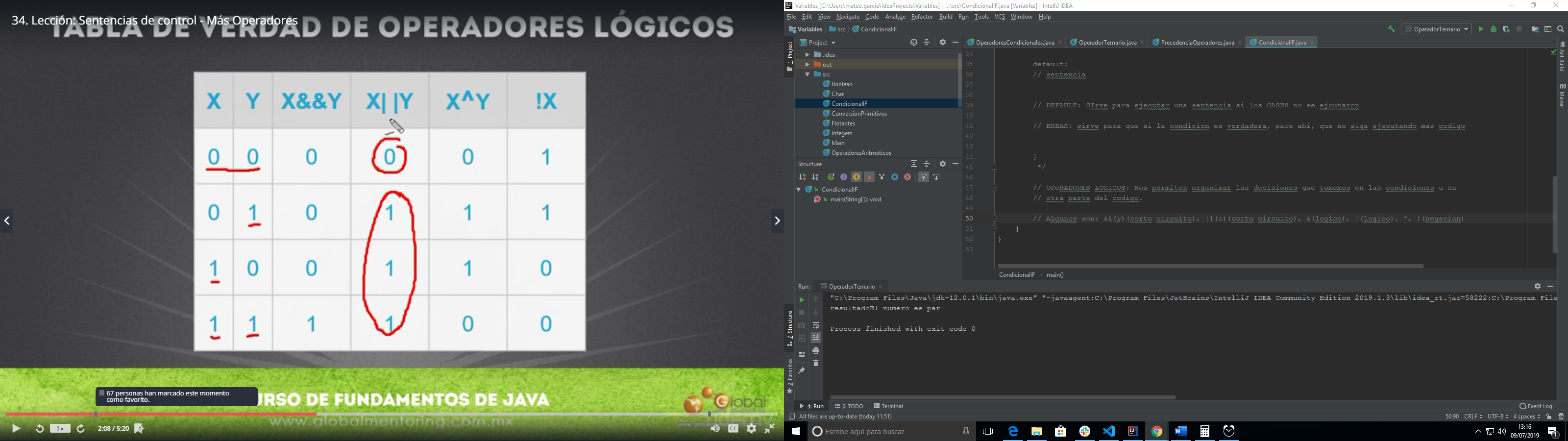
PRECEDENCIAS OPERADORES:

public class PrecedenciaOperadores {  
  
 public static void main(String[] args) {  
  
 // PRECEDENCIA: Como un conjunto de operaciones se va a diferenciar, separar de unos de otros, y que no creee conflictos entre ellos.  
 // al momento de dar el resultado final de ese conjunto de operaciones.  
  
 var x = 5;  
 int y = 10;  
  
 // como vemos estamos haciendo un pre-incremento y un post-incremento con las variables anteriores y las estamos sumando  
 var z = ++x + y--; // Analicemos primero: x = 6(antes de utilizarse en z), y = 9(despues de utlizarse en z  
 // por lo cual, lo ejecutara despues de hacer la suma, ya a diferencia de x(que lo hizo antes de la suma), este lo hara  
 // despues de la suma). Nos daria 16, ya que x = 6 + y = 9 = 15. PEro nos dara 16: porque realizo la suma antes de ejecutar  
 // el post-decremento de y.  
  
 System.*out*.println(x);  
 System.*out*.println(y);  
 System.*out*.println(z);  
  
 // NOTE: Ten en cuenta que la \* y la /. tienen Prioridad sobre la suma y la resta por lo cual se ejecutara primero,  
 // en el caso de que no usemos () para separar que operaciones son unas de otras.  
  
 var resultado2 = 4 + 5 \* 6 / 3;  
 System.*out*.println(resultado2);  
  
 // Lo ideal seria hacerlo asi  
  
 var resultado3 = 4 + (5 \* 6) / 3;  
 System.*out*.println(resultado3);  
  
 // TABLA DE PRECEDENCIA DE SIMBOLOS:  
 // (): Tiene la mayor prioridad  
 // \*, /: van despues  
 // +, -: van despues  
 // EN RESUMEN: obiamente hay mas cosas entre estas tambien, pero es un pequeño omini resumen para que te hagas una idea.  
  
 }  
  
  
  
  
 }

CONDICIONAL IF:

public class CondicionalIF {  
  
 public static void main(String[] args) {  
  
 // Condicional IF: Si es verdadero, hazme esta sentencia sino lo es hazme otra.  
  
 // if(condicion)  
 // sentencia si es verdadera la condicion  
 // else ()es opcional  
 // sentencia si la condicion no es verdades  
 // No lleva ";" al final del if, mientras que las sentencias dentro de los if si usan ; al final  
  
 // Podemos usar {} en las sentencias, para delimitar la cantidad de codigo en las sentnecias que vayamos a usar  
  
 // SENTENCIA SWITCH: ES otra forma de usar los condicionales, la utlizaremos por si tenemos muchos if's.  
 // soporta tipos como byte, short, int, char o String, entre otros.  
  
 /\*  
 Estructura:  
  
 switch(condicion a medias(porque, ya en los CASES o en los siguientes if's finalizaremos la condicion dependiendo de que queramos)/  
 expresion(tambien podemos simplemnte poner una variable y en los CASES condicionarla, es decir al final la idea es que aqui solo se tenga las variables  
 o lo que vayamos a usar, y ya en los CASES lo condicionares) {  
  
 case valor1:  
 // sentencia  
 break;  
 case valor 2:  
 // sentencia  
 break;  
 case valor 3:  
 // sentencia  
 break;  
  
 default:  
 // sentencia  
  
  
 // DEFAULT: SIrve para ejecutar una sentencia si los CASES no se ejcutaron  
  
 // BREAK: sirve para que si la condicion es verdadera, pare ahi, que no siga ejecutando mas codigo  
  
  
 }  
 \*/  
  
 // OPeRADORES LOGICOS: Nos permiten organizar las decisiones que tomemos en las condiciones u en  
 // otra parte del codigo.  
  
 // CORTO CIRCUITO: Significa alguno o abmos de los valores son falsos dependiendo de que operador logico usemos, ya no tiene sentido evaluar, es decir, por ejmplo  
 // si un valor es false y otro es true. Lo que pasara es que no vera necesario revisar/evaluar el otro valor que es TRUe  
 // porque se intuye que es TRUE. ESto si comparamos o usamos dos valores con operadores logicos corto circuito. ESto nos ahorra tiempo.  
  
  
 // ALgunos son: &&(y)(corto circuito), ||(o)(corto circuito), &(logico), |(logico), ^, !(negacion)  
  
 // ELSE IF: PARA agregar otro condicional si el anterior no se cumple.  
  
 var numero = 1;  
  
 if (numero == 1) {  
  
 System.*out*.println("numero uno");  
 }  
 else if (numero == 2) {  
  
 System.*out*.println("numero dos");  
 }  
 else if (numero == 3) {  
  
 System.*out*.println("numero tres");  
 }  
 else{  
 System.*out*.println("numero desconocido");  
 }  
  
 // Tambien podemos colocar ibviamente variables dentro en donde esta variable vaya cambiando,  
 // su valor dependiendo del condicional que le pongamos, y ya con un sout llamar a esa variable  
  
 }  
}

// Operadores Logicos Tabla



Donde 0: false

1: true

En ^: Exhor: Funcionara si ambos valores son iguales respecto a si son true or false, es decir.

Si x, y son false o true ambos, entonces Exhor será falso, mientras que si son diferentes entonces es true.

En ¡: Diferente de: Entonces si alguno, o ambos son false, ¡ es verdadero y asi sucesivamente, es al revez. Bueno este ya se sabe.

1. import java.util.Scanner;
3. public class Main {
4. public static void main(String[] args) {
5. Scanner scanner = new Scanner(System.in);
6. System.out.println("Proporciona un valor entre 0 y 10:");
7. int calificacion = Integer.parseInt(scanner.nextLine());
8. //Si esta entre 9 y 10 imprimir: A
9. if (calificacion >= 9 && calificacion <= 10)
10. System.out.println("A");
11. //Si esta entre 8 y menor a 9 imprimir: B
12. else if(calificacion >= 8 && calificacion < 9)
13. System.out.println("B");
14. //Si esta entre 7 y menor a 8 imprimir: C
15. else if(calificacion >= 7 && calificacion < 8)
16. System.out.println("C");
17. //Si esta entre 6 y menor a 7 imprimir: D
18. else if(calificacion >= 6 && calificacion < 7)
19. System.out.println("D");
20. //Si esta entre 0 y menor a 6 imprimir: F
21. else if(calificacion >= 0 && calificacion < 6)
22. System.out.println("F");
23. //Si no esta en el rago, imprimir: Valor desconocido
24. else
25. System.out.println("Valor desconocido");
26. }
27. }

import java.util.Scanner;  
public class CondicionalIF {  
  
 public static void main(String[] args) {  
  
 // Condicional IF: Si es verdadero, hazme esta sentencia sino lo es hazme otra.  
  
 // if(condicion)  
 // sentencia si es verdadera la condicion  
 // else ()es opcional  
 // sentencia si la condicion no es verdades  
 // No lleva ";" al final del if, mientras que las sentencias dentro de los if si usan ; al final  
  
 // Podemos usar {} en las sentencias, para delimitar la cantidad de codigo en las sentnecias que vayamos a usar  
  
 // SENTENCIA SWITCH: ES otra forma de usar los condicionales, la utlizaremos por si tenemos muchos if's.  
 // soporta tipos como byte, short, int, char o String, entre otros.  
  
 /\*  
 Estructura:  
  
 switch(condicion a medias(porque, ya en los CASES o en los siguientes if's finalizaremos la condicion dependiendo de que queramos)/  
 expresion(tambien podemos simplemnte poner una variable y en los CASES condicionarla, es decir al final la idea es que aqui solo se tenga las variables  
 o lo que vayamos a usar, y ya en los CASES lo condicionares) {  
  
 case valor1:  
 // sentencia  
 break;  
 case valor 2:  
 // sentencia  
 break;  
 case valor 3:  
 // sentencia  
 break;  
  
 default:  
 // sentencia  
  
  
 // DEFAULT: SIrve para ejecutar una sentencia si los CASES no se ejcutaron  
  
 // BREAK: sirve para que si la condicion es verdadera, pare ahi, que no siga ejecutando mas código.  
  
  
 }  
 \*/  
  
 // OPeRADORES LOGICOS: Nos permiten organizar las decisiones que tomemos en las condiciones u en  
 // otra parte del codigo.  
  
 // CORTO CIRCUITO: Significa alguno o abmos de los valores son falsos dependiendo de que operador logico usemos, ya no tiene sentido evaluar, es decir, por ejmplo  
 // si un valor es false y otro es true. Lo que pasara es que no vera necesario revisar/evaluar el otro valor que es TRUe  
 // porque se intuye que es TRUE. ESto si comparamos o usamos dos valores con operadores logicos corto circuito. ESto nos ahorra tiempo.  
  
  
 // ALgunos son: &&(y)(corto circuito), ||(o)(corto circuito), &(logico), |(logico), ^, !(negacion)  
  
 // ELSE IF: PARA agregar otro condicional si el anterior no se cumple.  
  
 var numero = 1;  
  
 if (numero == 1) {  
  
 System.*out*.println("numero uno");  
 }  
 else if (numero == 2) {  
  
 System.*out*.println("numero dos");  
 }  
 else if (numero == 3) {  
  
 System.*out*.println("numero tres");  
 }  
 else{  
 System.*out*.println("numero desconocido");  
 }  
  
 // Tambien podemos colocar ibviamente variables dentro en donde esta variable vaya cambiando,  
 // su valor dependiendo del condicional que le pongamos, y ya con un sout llamar a esa variable  
  
 // EJEMPLO SWITCH:  
  
 var texto = "numero desconocido";  
  
  
  
 switch(numero){  
  
  
 case 1:  
 texto = "numero 1";  
  
 break;  
  
  
 case 2:  
 texto = "numero 2";  
 break;  
  
  
 case 3:  
 texto = "numero 3";  
 break;  
  
  
 case 4:  
 texto = "numero 4";  
 break;  
  
  
 case 5:  
 texto = "numero 5";  
 break;  
  
  
 default:  
 texto = "Valor desconocido";  
  
 }  
  
 // IMPORTANTE RECORDAR: PARA QUE FUNCIONE UNA VARIABLE TENEMOS QUE IMPRMIRLA con SOUT  
  
 System.*out*.println(texto);  
  
 // Ejercicio IF-ELSE: Calculo Estacion del Año  
  
 // dEfinimos una variable que contendra un numero que representa el mes, un mes cualquiera pondremos  
 Scanner sc = new Scanner (System.*in*);  
 int mes = Integer.*parseInt*(sc.nextLine());  
 String estacion = "MEs no encontrado";  
  
 if (mes == 1 || mes == 2 || mes == 12)  
 estacion = "invierno";  
  
 else if (mes == 3 || mes == 4 || mes == 5)  
 estacion = "Primavera";  
  
 else if (mes == 6 || mes == 7 || mes == 8)  
 estacion = "Verano";  
  
 else if (mes == 9 || mes == 10 || mes == 11)  
 estacion = "Otoño";  
  
 else  
 System.*out*.println("Mes no encontrado");  
  
 System.*out*.println("estacion = " + estacion);  
  
  
  
 // MISMO EJERCICIO PERO CON SWITCH  
  
 Scanner sc2 = new Scanner (System.*in*);  
 int mes2 = Integer.*parseInt*(sc2.nextLine());  
 String estacionSwitch = "MEs no encontrado";  
  
 switch (mes2) {  
  
 case 1: case 2: case 12:  
 estacionSwitch = "invierno";  
 break;  
  
 case 3: case 4: case 5:  
 estacionSwitch = "Primavera";  
 break;  
  
 case 6: case 7: case 8:  
 estacionSwitch = "Verano";  
 break;  
  
 case 9: case 10: case 11:  
 estacionSwitch = "Otoño";  
 break;  
  
 default:  
  
 System.*out*.println("Mes no encontrado");  
  
 }  
  
 System.*out*.println("estacion = " + estacionSwitch);  
  
 }  
}

Julio 11/2019

public class BucleWhile {  
  
 public static void main(String[] args) {  
  
  
 // BUCLEs: nos permiten repetir una cierta cantidad de veces un cnojunto de instrucciones. Esto va a depender de una  
 // condicion, la cual debemos constrolar para saber cuantas veces se ejecutara esa cantidad de instrucciones y no que sea infinito.  
  
 // Condicion explicita: Es cuando no requiere la participacion del usuario, y el mismo programa se encarga de ejecutarlo por si mismo.  
 // al contrario de las Condiciones implicitas, en donde el usuario si tiene participacion en la ejecucion del codigo  
  
 // WHILE: Es un bucle, que permite ejecutar una cantidad de instrucciones si la condicion es verdadera, mientras que si la condicion  
 // no es verdadera no hara el buble. AQui necesitamos tener el control de la cantidad de veces que se ejecutara el bucle  
 // ya que sino, hara bucles se ejecutaran bucles infinitos  
  
 // FOR: Es un buble. EStructura: REvias la condicion, si es cierta ejecuta el codigo a repetir, incrementa o decrementa y se revisa la condicion  
 // de nuevo. En el bucle for a diferencia del WHILE, tenemos secciones ya definidas para controlar el bucle hasta el punto en donde  
 // el bucle debe parar, que seria cuando la condicion ya es falsa.  
  
 /\*  
 ESTRUCTURA:  
 for(inicializacion(variables de control o contadores, se utliza para iniciar el contador); condicion; iteracion(se incrementa o decrementa)) {  
  
 // codigo que queremos ejecutar  
  
  
 }  
  
 TIP: si queremos que nos salga el for con la estrucutra ya hecha ponemos:  
  
 fori. (A esto se el denomina SHORTCUT).  
  
 Simplemente seria cambiar el nombre de la variable y ponerle condicional.  
 \*/  
  
 /\*  
  
 WHILE DO: Lo mismo que while, pero al tener do, el codigo se ejecutara al menos una vez, es decir, si la condicion es falsa  
 de todas formas te ejecutara el codigo una vez.  
  
 ESTRUCTURA:  
  
 do{  
 System.out.println("i = " + i);  
 i++;  
 } while(i <= 3);  
 \*/  
 System.*out*.println();  
 var texto = "impresion";  
  
 // nos lo imprira de manera infinita, ya que la condicion siempre es verdadera  
  
 // while(true) {  
 // System.out.println("texto = " + texto);  
   
 // }  
  
 var i = 0;  
   
 // Como vemos, si la condicion la condicionamos hasta cierto punto, y le ponemos un incremento/decremento no habra bucle infinito.  
   
 while(i <= 3) {  
  
 System.*out*.println("i = " + i);  
  
 i++;  
 }  
  
 for (i = 0; i < 3; i++) {  
  
 System.*out*.println("i = " + i);  
 }  
  
 /\*  
 BREAK: REcordemos que el BREAK BREAK: sirve para que si la condicion es verdadera, pare ahi, que no siga ejecutando mas código.  
 Pero hay una cuestion, el BREAK al estar dentro de un bucle, si este esta en una condicion es verdadera, el BREAK lo que hace  
 es parar ahi y romper el bucle, es decir, ya no nos ejecutara el bucle si la condicion es verdadera. ESto sirve mucho pero a la vez,  
 puede darnos problemas dependiendo de lo que queremos.  
  
 CONTINUE: CONTINUE sirve para indicarle a la condicion y al contenido de que si  
 si la condicion es verdadera, esntonces no ejecutes el codigo que tiene dentro, mientras que si la condicion es falsa  
 ejecute el codigo. ES como decirle, si la condicion es verdadera pasa de el, si es falsa ejecutalo. todos esto mientras el cONTINUE  
 este antes del codigo a ejecutar si sin corchetes "{}".  
 ADemas el CONTINUE, no rompe el bucle como el BREAK, sino que se sigue ejecutando la itineraicon del codigo, es decir. Otros condcionales o lo que  
 tengamos dentro del bucle  
 \*/  
  
 // ejm: En este caso el rbeak al ver que la condicion es true si la variable "numero" es par, entonces rompe el bucle.  
  
  
  
 for (var numero = 1; numero <= 4; numero++) {  
  
 if (numero % 2 == 0) {  
  
 // como vemos imprime el 2 solamente y no el 4, porque al ver que la condicion es verdadera, rompe el bucle impidiendo  
 // que el 4 que es par, tambien se imprima  
  
 System.*out*.println("numero = " + numero);  
 break;  
  
 }  
  
 }  
  
 for (var numero2 = 1; numero2 < 5; numero2++) {  
  
 if (numero2 % 2 == 0)  
  
 // Como vemos nos imprime el 1 y el 3 porque el continue al ver que la condicion es verdadera, osea los pares, nos los  
 // coloca, mientras que los impares que son falsos dentro de la condicion si los pones.  
 // como vemos tambien no ponemos corchetes y debe ir antes del codigo a ejecutar para que funcione de esta forma.  
 continue;  
  
 System.*out*.println("numero2 = " + numero2);  
  
  
  
  
  
 }  
  
 }  
  
 }

CREACION CLASE:

// CREACION DE CLASES EN JAVA: Recordemos las clses son modelos apra la creacion de objetos.  
// como una plantilla  
// TPIS DE DATOS:  
/\*  
  
 TIPOS REFERENCIADOS(TIPO OBJECT):  
 CLASES, INTERFACES Y ARREGLOS  
  
 . CLase: debe llamarse igual que el archivo  
 \*/  
class CreacionClases {  
  
 // VAriables de instancia de nuestra clase: SOn aquellos atributos de la clase  
  
 // ejm: tipoDato variableAtributo;  
  
 // METODOS: Son los que relamenten contiene le codigo de la clase, son los comportamientos del objeto  
  
 // Estrcutra: public/private tipoDato nombreDelMetodo (argumentos) {

// codigo del metodo  
 // }  
  
 /\*  
 Ejemplo clase:  
  
 class PErsona {  
  
 // ATRIBUTOS  
  
  
 String nombre;  
 int numero;  
  
 // METODOS:  
  
  
 }  
 \*/  
  
  
}

public class Persona {  
  
  
// IMPORTANTE: no poner el METODO MAIN(que es para ejecuta codigo) cuando creamos una clase.  
  
 // CONTEXTO DINAMICO: Hay casos en los que no necesariamente trabajaremos con los objetos sino directamente con las clases  
 // esto se le llama CONTEXTO ESTATICO. Pero el CONTEXTO dinamico es lo que estamos ahciendo ahora, trabajar con obejtos mendiante una clase.  
  
 // ATributos  
  
 String nombre;  
 String apellidos;  
 int telefono;  
  
 // CONSTRUCTORES: Es un metodo que Nos permiten asignar valores a los atributos directamente al momento de crear el objeto  
 // cuando ponemos NEw nombreClase (Aqui ponemos los valores que le asignamos a los atributos mediante el constructor);  
 // Esto nos beneficie ya que on tendremos que hacer un metodo solo para que nos devuelvan los valores que le asignemos,  
 // el constructor no necesita RETURn como si lo hicieramos con metodos, ya que los valores los ponemos directamente en la creacion  
 // de los objetos cmo ya se meciono.  
 // EL nombre del constructor debe ser identico al nombre de la CLase, respectando las minusculas y las mayusculas.  
 // No es obligatorio usar el constructor para poner el valor de todos los atributos, podemos usarlo para aquellos  
 // atributos que son mas importantes.  
 // Si no hay necesidad de usar constructor, no se pone.  
  
 // SOBRECARGA DE METODOS: Si dos omas metodos/constructores tiene el mismo nombre, puede que sea porque habran varias formas para la creacion de los objetos, es decir, al momento de crear dos objetos opr ejemplo, un objeto puede usar uno de los constructores, mientras que con el otro objeto podemos usar el otro constructor  
 // Si hay por ejemplo 2 constructores y en sus parametros uno esta vacio y otro si tiene parametros entonces tenemos dos formas de creacion de  
 //objetos.  
  
   
  
  
 // Metodos  
  
 // public, todos acceder, private: es privado  
  
 // VOID: Le dice al metodo que no va a regresar ningun tipo de informacion, es decir, no regresa ninngun tipo de informacion o valor sobre el metodo  
 // // solamente el valor que tenga la variable. ademas que el tipo de dato no influye en el contenido del metodo.  
 // Si en el caso contrario queremos que nos devuelva informacion, pues hay especificar el tipo de dato que queremos y ponerle RETURN  
 // para que nos retorne el valor:  
 // ejm: int, double, string, etc.  
  
  
 public void deplegarNombre() {  
  
 System.*out*.println("nombre = " + nombre);  
 }  
  
 public void deplegarApellidos() {  
  
 System.*out*.println("apellidos = " + apellidos);  
 }  
  
 public void deplegarTelefono() {  
  
 System.*out*.println("Telefono = " + telefono);  
 }  
  
  
 // Por ahora no podemos ejecutarlo, ya que no tenemos el metodo MAIn, que nos permite ejecutar el codigo, ademas de que  
 // necesitamos crear los objetos  
  
 // CREACION DE UN OBJETO: REcordemos que con objetos reune todos lo de una clase y cada objeto tendra sus propios valores  
  
 // ESTRUCTURA: NOmbredelaClase NombredelObjeto/Variable = new NombredelaClaseDeNuevo(constructor(se vera mas adelante));  
  
 /\*  
 PARA ACCEDER A LOS ATRIBUTOS/METODOS DE LA CLASE PARA USARLOS EN LOS OBJETOS usamos ".".  
 ESTRUCTURA:  
 nombredelObjeto.Atributo = ;  
 nombredelObjeto.Metodo() = ;  
 EJM:  
  
 Persona p1 = new Persona();  
  
 pl.nombre = "Pedro";  
 \*/  
  
 /\*  
 CLASE STRING: EL string de por si en JAVA, es una clase pero no necesita del NEW, no es un tipo primitvo,  
 pero se pone en el mismo sitio que los tipos primitivos  
 \*/  
  
 /\*  
 CREACION DE METODOS EN JAVA:  
  
 ARGUMENTOS = PARAMETROS: Los parametros se usan para poner en ellos ya sean variables, valores para que el  
 contenido del metodo se ejecute. no son obligatorios  
  
 tipo nombreMetodo(lista de argumentos(osea variables)) {  
 // contenido del metodo  
 }  
  
 EL nombre del metodo empieza en minuscula y las demas palabras que lo incluya las podemos colocar si queremos  
 en mayusculas como en las variables  
  
  
 \*/  
  
 // EJM:  
  
 int sumar(int a, int b) {  
  
 // Podemos hacerlo de esta forma: return a + b;  
  
 // Si lo queremos mas comodamente, lo podemos meter en una variable  
  
 int resultado = a + b;  
 return resultado;  
 }  
  
  
}

public class PErsonaObjeto extends Persona{  
  
 // aqui si ponemos el metodo MAIN, porque obviamente queremos que se ejecute el codigo dentro en el este caso  
 // en la creaicon de objetos  
  
 public static void main(String[] args) {  
  
 // Creacion del objeto  
  
 Persona pl = new Persona();  
  
 // Modificar valores del atributo del objeto creado  
  
 pl.nombre = "Pedro";  
  
 // Llamar al metodo  
 pl.deplegarNombre();  
  
  
  
 // Creacion del segundo objeto  
  
 Persona pl2 = new Persona();  
  
 // Modificar valores del atributo del objeto creado  
  
 pl2.nombre = "Alberto";  
  
 // Llamar al metodo  
 pl2.deplegarNombre();  
  
 // podemos poner un metodo dentro de una variable para usarla mas comodamente en otro sitio o por si la necesitamos en algun momento  
  
 int resultado = pl2.sumar(2, 2);  
  
 System.*out*.println("resultado = " + resultado);  
 }  
}

PONER CLASES ARITMETICA Y ARITMETICAOBJETO

package ClaseAritmetica;  
  
public class Aritmetica {  
  
  
  
 // Atributos  
  
 int a;  
 int b;  
  
  
  
 // Constructor con VACIO  
  
 Aritmetica(){}  
  
  
 // COsntructor CON PARAMETROS  
  
 Aritmetica(int arg1, int arg2) {  
  
// Igualamoslos parametros con el nombre de los atributos  
  
 // No usamos int en este caso en los atributos porque estos los pusimos al mometnos de definirlos  
  
 a = arg1;  
 b = arg2;  
 }  
  
 // Metodos  
  
 // NOTA: COMO VEMOS ESTE METODO YA TIENE PARAMETROS PORQUE EL CONSTRUCTOR SUPLE SU FUNCION, SIMPLEMETEN ESTE METODO HACE LA FUNCION DE SUMAR  
  
 // IMPORTANTE EN UN METODO: Un metodo en JAVA solo puede devolver un valor no varios si usamos RETURN  
  
 int Sumar() {  
  
 int resultado = a + b;  
 return resultado;  
 }  
  
 // THIS: En ocasiones un metodo necesit ahacer refefncia al objeto con el que estamos trabajando  
 // el THIS es un operador el cual nos permite acceder al objeto actual y la clase con la que estamos trabajando,  
 // en donde nos servira para ACCEDER a los atributos o metodos de una clase.  
 // En el caso del constructor nos permitira acceder con THIs a los atributos de la clase  
 // Con esto poderemos diferencias que atributos son de la clase y cuales no  
}

package ClaseAritmetica;  
  
public class AritmeticaObjeto{  
  
 public static void main(String[] args) {  
  
 // CReamos el objeto con constructor que tiene parametros  
  
 Aritmetica suma = new Aritmetica(5,5);  
 int resultado = suma.Sumar();  
  
 System.*out*.println("resultado = " + resultado);  
  
  
 // CReamos el objeto con constructor que no tiene parametros  
  
 Aritmetica suma2 = new Aritmetica();  
 // Como vemos ponemos el valor a los atributos como lo vimos en un principio usando el ".".  
  
 suma2.a = 5;  
 suma2.b = 5;  
 int resultado2 = suma2.Sumar();  
  
 System.*out*.println("resultado2 = " + resultado2);  
 }  
}

Julio 12/2019

THIS: // THIS: En ocasiones un metodo necesit ahacer refefncia al objeto con el que estamos trabajando  
// el THIS es un operador el cual nos permite acceder al objeto actual y la clase con la que estamos trabajando,  
// en donde nos servira para ACCEDER a los atributos o metodos de una clase.  
// En el caso del constructor nos permitira acceder con THIs a los atributos de la clase, es ceir a los atributos y métodos de un objeto, que al final tienen las cosas de las clases.  
// Con esto poderemos diferencias que atributos son de la clase y cuales no

Consejo: Es mejor tener claro dos momentos: la creaciond enuestras clases/plantilla y la creación de objetos(en donde se ejcutara el codigo). Aunque en los ejericicos anteriores de la teoría de clases, ene el constructor y en los métodos no hemos usado el THIS, lo común es que siempre se pongo para que quede claro que lo que usamos es los atributos de clase, esto obviamente para diferenciar de los atributos(variables de la clase) y las variables que no son de la clase.

IMPORTANTE: En el constructor los parámetros se deben llamar igual que los atributos de la clase. En donde el parámetro es que hace de intermediaro para unirse/igualarse con el atributo de la clase, a esto se le denomina “ocultamiento del atributo de una clase.” Ya que el parámetro es que hace de intermediario, y para diferenciar obviamente el parámetro con el atributo porque se llaman igual usamos el THIS.

ESTRUCTURA THIS:

This.nombreAtributo

ALCANCE DE UNA VARIABLE: EN java HAY DIFENTENTES TIPOS DE VARIABLES,VARIABLES DE CLASE Y LOCALES, dependiendo donde se declare será la duración de la misma, ha esto se le conoce como alcance de una variable.

VARIABLES DE CLASE: Son los atributos, Pueden usarse en cualquier método de la clase, se inicializan con valores por default, es decir, DEFAULT: que se inicia la variable sola, se podrá usar, además tendrán usar valor por DEFAULT: ya sea cualquier primitvo tendrá un valor por default, mientras que si no son DEFAULT se necesita de un SOUT para poder usarla y asignarle un valor.

VARIABLES LOCALES: Seran los parámetros o alguna variables declarada dentro un método, por lo cual solo se usara en ese método y por eso es local para ese método, se deben inicializar, ya que estas no se van a inicializar por DEFAULT.

THIS para diferenciar variables locales y de clase: ya se mencionano antes, pero si usamos una variable local para introducirle en ella variables de clase, tendríamos que usar THIS para acceder a esas variables de clase.

Tambien podemos usar THIS para métodos, si queremos poner métodos en otros métodos. NO peodemos usar THIS para el contexto Estatico(Lo veremos mas adelante).

AHORA VEREMOS EL THIS Y VARIABLES LOCALES CON LA CLASE ARITMETICA:

package ClaseAritmetica;  
  
public class Aritmetica {  
  
  
  
 // Atributos  
  
 int a;  
 int b;  
  
  
  
 // Constructor con VACIO  
  
 Aritmetica(){}  
  
  
 // COsntructor CON PARAMETROS  
  
 Aritmetica(int arg1, int arg2) {  
  
// Igualamoslos parametros con el nombre de los atributos  
  
 // No usamos int en este caso en los atributos porque estos los pusimos al mometnos de definirlos  
  
 this.a = arg1;  
 this.b = arg2;  
 }  
  
 // Metodos  
  
 // NOTA: COMO VEMOS ESTE METODO YA TIENE PARAMETROS PORQUE EL CONSTRUCTOR SUPLE SU FUNCION, SIMPLEMETEN ESTE METODO HACE LA FUNCION DE SUMAR  
  
 // IMPORTANTE EN UN METODO: Un metodo en JAVA solo puede devolver un valor no varios si usamos RETURN  
  
 int Sumar() {  
  
 int resultado = a + b;  
 return resultado;  
 }  
  
 // THIS: En ocasiones un metodo necesit ahacer refefncia al objeto con el que estamos trabajando  
 // el THIS es un operador el cual nos permite acceder al objeto actual y la clase con la que estamos trabajando,  
 // en donde nos servira para ACCEDER a los atributos o metodos de una clase.  
 // En el caso del constructor nos permitira acceder con THIs a los atributos de la clase  
 // Con esto poderemos diferencias que atributos son de la clase y cuales no  
  
 // MEtodo restar  
  
 int restar() {  
  
 int resultadoResta = a - b;  
  
 return resultadoResta;  
 }  
  
 //Metodo multiplicar  
  
 int multiplicar() {  
  
 int resultadoMultiplicar = a \* b;  
  
 return resultadoMultiplicar;  
 }  
  
 // Metodo dividir  
  
 int dividir() {  
  
 int resultadoDividir = a / b;  
  
 return resultadoDividir;  
 }  
}

package ClaseAritmetica;  
  
public class AritmeticaObjeto{  
  
 public static void main(String[] args) {  
  
 // MAIN al final es un metodo, por lo cual al declarar variables aqui, seran locales.  
 // IMPORTANTE A TENER EN CUENTA: AL usar variables locales, estas se borraran de la memoria  
 /\* Unicamente cuando el codigo termine de ejecutarse, MIENTRAS QUE LAS VARIABLES DE CLASE solo se  
 borraran de memoria cuando se elimine el objeto  
 \*/  
 // Declaramos variables locales pars usarlas dentro del constructor  
  
 int operadorA = 2;  
 int operadorB = 6;  
  
 // CReamos el objeto con constructor que tiene parametros  
  
 Aritmetica suma = new Aritmetica(operadorA,operadorB);  
  
 // REsultado SUma  
 int resultadoSuma = suma.Sumar();  
  
 System.*out*.println("resultado = " + resultadoSuma);  
  
 // REsultado Resta  
 int resultadoResta = suma.restar();  
  
 System.*out*.println("resultado = " + resultadoResta);  
  
 // REsultado Multiplicacion  
 int resultadoMultiplicacion = suma.multiplicar();  
  
 System.*out*.println("resultado = " + resultadoMultiplicacion);  
  
 // REsultado Division  
 int resultadoDivision = suma.dividir();  
  
 System.*out*.println("resultado = " + resultadoDivision);  
  
  
 // CReamos el objeto con constructor que no tiene parametros  
  
 Aritmetica suma2 = new Aritmetica();  
 // Como vemos ponemos el valor a los atributos como lo vimos en un principio usando el ".".  
  
 suma2.a = 5;  
 suma2.b = 5;  
 int resultado2 = suma2.Sumar();  
  
 System.*out*.println("resultado2 = " + resultado2);  
 }  
}

Julio 16/2019

Video 56-76

MEMORIA STACK Y HEAP EN JAVA:

JAVA tiene dos sistemas para almacenar internamente en memoria los datos de los valores que recopilamos

MEMORIA STACK: Se utiliza para almacenar las variables locales y también las variables de función en JAVA, tienen un periodo de vida corto porque almacenar valores únicamente cuando se ejecutan y cuando termina.

MEMORIA HEAP: Se utiliza para almacenar los objetos JAVA incluyendo sus atributos, los bojetos almacenados aqui normalmente tienen un periodo de debida mas prolongado debido a que a menos que la eliminemos no se borrara de la memoria heap, además no guardan el objeto en si sino la referencia(dirección de memoria donde esta almacanado el objeto), y la variable que nombramos para crear el objeto con NEW y el nombredelaClase es la que se encarga de referenciar al objeto, si esta variable se borra, la referencia se borra y por lo tanto, se quita de memoria. Ejm:

Persona p1 = new Persona(); // Aquí p1 es la variable que permite hacer la refencia y que el objeto que creamos con esta variable se meta dentro de la memoria HEAP, a su vez si hacemos esto:

Persona p1 = p2; // Estariamos diciendo que ya hay dos variables que almacenar la referencia del objeto creado.

Las variables y objetos en memoria se ponen en código hexagonal.

PASO Por VALOR: Si queremos usar la variable en otro método que no sea MAIN ya sea ara modificarla, cambiarla, borrarla, etc. Ya sea en métodos, lo que pasara es que en los parámetros pondremos el valor de esa variable para realizar lo que queremos, la cuestión IMPORTANTe aquí es que el valor del atributo o variable original no se modifica, es decir, lo que hicimos en los métodos con referenciar el valor del atributo, lo que paso es que se hizo una “COPIA” del valor de ese atributo sin afectar al valor del atributo original, simplemente en otros métodos que no son MAIN referenciamos la variable que tiene ese valor para usarlo en ese nuevo método, para después meter este método dentro del MAIN para ejecuarlo y ver los cambios o modificaciones que hicimos a esa variable.

EJM:

package PasoporValor;  
  
  
  
public class PasoPorValor {  
  
 // METODO que queremos ejecutar junto con los metodos que tienen el PASO POR VALOR  
  
 public static void main(String[] args) {  
 int x = 10;  
  
 // metodo IMPRIMIR que tiene PASO POR REFERENCIA  
 // COmo vemos introducimos dentro del parametro del metodo la variable que tiene el valor  
 // COmo vemos introducimos dentro del parametro del metodo la variable que tiene la referencia del objeto  
 // en este caso "x"  
 // IMPORTANTE: Como vemos al momento de ejecutar el codigo no nos hace el cambio, esto es debido a que  
 // el PASO POR VALOR; no te hace una referencia en si del valor, sino una copia del original, por lo cual  
 // si queremos que haya cambio, unicamente podemos modificar/cambiar/usar el valor de esa variable dentro del mismo  
 // metodo, en este caso MAIN.  
  
 *imprimir*(x);  
 *cambiarValor*(x);  
 }  
  
 public static void cambiarValor(int variable) {  
  
  
 variable = 200;  
 System.*out*.println("X modificada = " + variable);  
  
 }  
 public static void imprimir(int variable) {  
 System.*out*.println("variable = " + variable);  
  
 }  
}

PASO POR REFERENCIA: Es básicamente poner objetos como parámetros para usarlos en métodos, en lugar de utilizar tipos primitvos. Lo que hacemos al hacer esto es usar dentro de los parámetros la variable que tiene la refencia del objeto(la variable con que creamos el objeto con NEW), porque queremos hacer esto?. Para poder modificar el objeto directamente en un método, esto sin necesidad de crear una copia del objeto.

IMPORTANTE: En el archivo de crear el objeto en donde usamos MAIN, que al final es un método para ejecutar código. Para hacer el PASO POR REFRENCIA, tenemos que tener en cuenta que los métodos donde queremos referenciar los objetos mediante su variable, estos métodos esten fuera del MAIN, para poder referenciarlos, no tendría sentido usarlos dentro del MAIN porque ya no habría necesidad de hacer PASO POR rEFRENCIA porque el objeto lo tenemos en el mismo método que queremos usar, en estos casos es cuando queremos refernciar el obejto en otro METODO. IMPORTANTE también que el método use la misma estructura que el MAIN, porque esa estructura de método nos permite ejecutar código, ejm:

Public static void modificarNombre(){}. Como vemos este otro método que queremos crear, solo cmabiamos el nombre MAIN pero sigue teniendo la misma estructura.

ADEMAS IMPORTANTE TAMBIEN que estos métodos que usamos para hacer PASO por referencia los introduzcamos dentro del METODO MAIN(básicamente usar unos métodos dentro un metodo), para que los usemos obviamente dentro del MAIN y asi ver las modificaciones que hemos hecho, además de que OBVIAMENTE tenemos que poner en los paramwtros de estos métodos la variable que tiene la referencia del objeto.

EJM:

Clase para crear Objeto:

package PasoporReferencia;  
  
public class clasePersona {  
  
 String nombre;  
  
 // REcordemos que si ponemos void le estamos indicand que no nos devuelva ningun valor o informacion,  
 // simplemente que ejecute el codigo que tiene adentro  
  
 public void cambiarNombre(String nuevoNombre) {  
 this.nombre = nuevoNombre;  
 }  
  
 public String obtenerNombre() {  
 return this.nombre;  
 }  
}

Creacion del objeto y PASO POR REFERENCIA:

package PasoporReferencia;  
  
public class PasoPorReferencia {  
  
 // METODO que queremos ejecutar junto con los metodos que tienen el PASO POR VALOR  
  
 public static void main(String[] args) {  
  
 // Objeto  
 clasePersona p = new clasePersona();  
  
 p.cambiarNombre("Juan");  
  
 // metodo IMPRIMIR que tiene PASO POR REFERENCIA  
 // COmo vemos introducimos dentro del parametro del metodo la variable que tiene la referencia del objeto  
 // en este caso ""  
  
 *imprimirNombre*(p); // imprime Juan  
 *modificarPersona*(p); // SE modifica a Carlos  
 *imprimirNombre*(p); // imprime Carlos  
  
 }  
  
 // el parametro objeto como vemos lo usamos en un metodo de la clase "clasePersona", no pasaria nad aporque  
 // al poner en el parametro la variable que tiene referencia del objeto, este objeto automaticamente se vuelve igual  
 // a esa variable(como toda funcion con parametros), es decir, objeto = p. porque en MAIN colocamos modificarPersona(p);  
  
 public static void modificarPersona(clasePersona objeto) {  
  
 objeto.cambiarNombre("Carlos");  
  
 }  
  
 public static void imprimirNombre(clasePersona objeto) {  
  
 System.*out*.println("Nuevo valor del atributo nombre: " + objeto.obtenerNombre());  
  
 }  
  
  
}

REGRESO DE UN METODO: hay 3 maneras de terminar de ejecutar el método que tengamos.

1. Un método regresa

RETURN EN JAVA: Nos sirve para regresarnos un valor, si lo usamos en un método podemos usarlo para ver el resultado de parámetros u otras cosas que queramos, pero en si no es obligatorio poner return, ya que de por si el método lo hace.

Lo que si es importante es que si lo vamos a usar en JAVA y en un método tengamos en cuenta que el valor que va retornar el RETURN debe ser el mismo tipo de dato que el que asigamos al momento de crear la estructura del método, es decir, si en la estructura al momento de crear el método pusimos un INt por ejemplo, el return según lo que quieres debe sacarte un valor con tipo de dato INT. Ademas si metemos un método dentro de una variable esta variable también debe ser del mismo tipo de dato que lo mencionado antes.

RETURN puede ser retornar un valor tipo primitivo y también tipo OBJECT(array, lista, etc.) pero tambien a objetos, simplemente poner return nombrevariableDelObjeto;

Pero si hacemos esto en un método este return debe sacar un objeto con el mismo tipo de clase, ademas no es muy necesario porque dentro del MAIN ya directamente podemos hacerlo sin return.

USO RETURN CONDICIONADO: PODEMOS USAR RETURN también de valores en condicionales

CLASE DENTRO DE OTRA: En JAVA podemos crear varias clases dentro de un mismo archivo, pero lo IMPORTANTe aquí es que 1 clase de ellas sea publica y es la que tiene que tener el mismo nombre que el archivo. Mientras que las demás no deberían tener public

IMPORTANTE: VAR no se puede usar en un METODO NI EN LOS PARAMETROS, además si usamos VOID en un método y ponemos RETURN OBVIAMENTE no nos va devolver nada, porque VOID solo ejecuta no muestra.

PODEMOS USAR LA PALABRA THIS: Para permitir a un objeto enviarse el mismo como parámetro

LA PALABRA THIS NO SE PUEDE USAR EN UN CONTEXTO ESTATICO PORQUE: Aun no se han creado Objetos, por lo cual al momentod e crear el obejto podemos usar la palabra THIS

Tipo OBJECT: para usar objectos en los parametro ejm: public void imprimir(Object 1Objeto){}. Y únicamente guardan la referencia

USO DE LA PALABRA THIS COMO REFERENCIA: Podemos usar THIS como lo hicimos en PASO POR REFERENCIA, que era con los objetos, pero hay unos cambios:

1. Si creamos varias clases en un mismo archivo y queremos referenciar el THIS unas con otras, es importante saber que el THIS que pongamos en esa clase, siempre va a pertenecer a ESA CLASE, entonces como lo referenciamos?. Simple, si creamos un objeto dentro otra clase1 que no sea la clase principal que tenga el MAIN, y el constructor u otro método de ese objeto de la clase2 le ponemos en el parametro la palabra THIS que pertenece a la clase1, porque el objeto lo creamos en la clase1, que crees que pasa?. Si tenemos en cuenta que cada clase tiene su proipo THIS, al momento de poner el THIs dentro del parametro del constructor o un método, lo que hacemos es que ese método o constructor tenga ya en su parametro el THIS de esa clase1, por lo cual al ejecutarse el código que tenga ese método u constructor de la clase 2, ya tendríamos referenciado el THIs de la clase1, si tenemos por ejmplo un SOUT, para imprmir el parametro, nos mostrara el THIS refenciado de la clase1, pero si en el mismo código ponemos un SOUt pero no del parametro sino del THIS de la clase2, no nos mostrara obviamente el THIs de la clase1 porque ahora estamos en la clase2.

public class PalabraThis {  
 public static void main(String[] args) {  
 Persona persona = new Persona("Juan");  
 }  
}  
  
class Persona{  
 String nombre;  
  
 Persona(String nombre){  
 this.nombre = nombre;  
  
 //imprimir el objeto persona  
 Imprimir imprimir = new Imprimir();  
 imprimir.imprimir(this);//this en el parametro de la clase Imprimir, lo cual estariamos referenciando el THIS de la clase Persona  
 }  
}  
  
class Imprimir{  
 public void imprimir(Persona p){  
 System.out.println("impresion argumento: " + p);//valor del THIS referenciado de la clase Persona  
 System.out.println("impresion objeto actual (this)" + this);//this de la clase Imprimir  
 }  
}

Si en vez de borrar el objeto si ya no lo necesitamos, podemos asignarle a la variable que contiene el objeto un NULL, para que se borre pero que en el código permanezca por si algo.

System.go(): Recolector de basura. Solo planifica la llama, pero no significa que va a borrar, simplemente lo esta analizando para después ver sis si lo elimina

ENCAPSULAMIENTO EN JAVA: Nos permite aislar clases de otras clases, y asi no permitir que tengan acceso a los atributos o a los métodos que no deseemos eprmititr,.

ESTADO de un clase: En que forma están protegidos o que permisos hay para acceder. Esta generalmente oculto, y par asaber en que estado se encuentra la clase, tenemos los modificadores de acceso: PUBLIC, PRIVATE, DEFAULT y PROTECTED.

PUBLIC: Permite el acceso desde cualquier clase a cualquier método o atributo, y cualquier lugar de l carpeta en donde se almacene el archivo con la clase.

PRIVATE: Permite acceder solo desde la misma clase asignándole a los métodos o atributos.

PROTECTED: Permite acceder únicamente desde la clase-padre y la clase-hijo.

En atributos: Se ponen antes del tipo de dato y el nombre de la variable.

En métodos: Se ponen antes de otros modificadores(static, tipo de dato, void, etc.) y del nombredelMetodo(parametros){}.

Por lo cual si necesitamos de un atributo o método pero están en PRIVATE, necesitaremos de un método que sea PUBLIC, ya que al introducir el atributo o metodo en un método con PUBLIC, lo que pasara es que automáticamente los PRIVATe se volverán PUBLIC y se podrán usar, es importante tener en cuenta que mediante un método o atributos a un constructor podemos modificar los atributos o métodos con ese tipo de acceso PRIVATE…

En java y en tros lenguajes, una forma de organizar los emtodos que nos permitirán modificar el estado de los atributos y métodos que esten en PRIVATE es GETTER y SETTER, en si es simplemente NOMENCLATURA no es parte del código ni ndad, va en el nombre, es para indicar mejor para que sirve ese método de una forma optima, comúnmente el get y el set empiezan con minúsculas:

GET: Nos permite visualizar el valor del atributo o método, ya sea con un RETURN o lo que pongas, pero simplemente para visualizar, no tiene parámetros, NO olvidar poner THIS para saber que son los atributos o métodos de la clase.

SET: Nos permite modificar el valor del atributo o metodo, tiene parámetros y tiene mas código que permite modificarlo, no e un simple RETURN para visualizar sino que puede tener mas cosas.

IS: Lo mismo que GET pero para BOOLEAN

CONTEXTO ESTATICO: Se trata del momento cuando Las clases se guardan en memoria, es decir, todo el proceso de ejecución en la aplicación en donde se cargan las clases en memoria, el contexto estatico se podrá usar hasta que se eliminen las clases de la memoria. EL contexto estatico se carga en memoria antes del contexto dinamico en donde las clases que vayamos a utilizar en primera instancia. Ademas el contexto estatico tiene una mayor duración hasta que se eliminen las clases, también el contexto estatico incluye al contexto dinamico ya que mediante esta el contexto estatico, el contexto dinamico puede crear y cargar los objetos, esto solo periódicamente y no viceversa.

STATIC: Nos permititra usar el contexto estatico. Si definimos un método o atributo como estatico, lo que estamos indicando es que estos pertenecen a la clase y no al objeto, además este método o atributo solo se crea una vez, por lo cual sin importar cuantos objetos se creen solo habrá ese atributp/variable o método, y si un objeto accede a una variable estatica. Estavera el mismo que valor que si lo ve otro objeto también, por lo cual si un objeto modifica el valor de una variable estatica, esta se modificara también para el resto de los objetos. En cambio si creamos un atributo sin usar la palabra static, cuando creemos el objeto esta varibale estará asociada al objeto y no a la clase. Utilizando la palabra STATIC podemos interactuar con las variables de la clase sin necesidad de crear un objeto, COMO?:

Asi: nombreClase.atributo/metodoEstatico;

Si queremos usar las varibales estaticas en otras clases, estos deben tener obviamente el estado PUBLIc para poder acceder a ellas.

El operador THIs no es posible usarse en el Contexto estatico porque el THIs comienza a funcionar cuando creamos algun objeto, es decir, el THIS comienza a funcionar en el contexto dinamico, por lo cual el tHIS no funciona en un contexto estatico ya que se carga primero el contexto estatico y despues el dinamico. Por lo cual si usamos el THIs en un método ESTATICO por ejemplo sin haber creado un objeto previamente, este dará error. En cambio, si queremos meter un método dinamico a un método estatico si se puede mientras ya se ahya creado los objeto. Por lo cual podemos acceder desde un método dinamico a un método estatico simere y cuando ya se halla creado el objeto en el método estatico.

Podemos acceder desde un método estatico a un método dinamico siempre y cuando al estar en el método dinamico escribamos: nombreClase.atributo/metodoEstatico; para saber que es estatico.

EJM:

Al declarar una variable/atributo en una clase como estatica, lo que bucamos cada vez creemos un objeto, es que se incrementara el valor de esa variable, podemos usar el constructor para que nos facilite las cosas(no ponemos THIS porque recordemos que solo el THIS funciona al momento de crear un objeto). Lo que queremos entonces es que mediante este atributo estatico que estará en el constructor con un “++” vaya sumando su propio valor cada vez que se cree un objeto, como sabemos que va a hacer eso?. Recuerda que el estatico es de las clases, por lo cual esta variable/método siempre estará ahí al momento de crear un objeto, sino fuera estatica, esta variable se reiniciaría cada vez que creamos un objeto, porque ya no pertenece a la clase sino al objeto, es decir, esa variable no estatica es única para ese objeto “islas diferentes”, entonces al ser estatico, es de la clase, por lo cual mediante esta variable sabremos cuantos objetos fueron creados a partir de la clase. Ademas si igualamos esta variable con otro atributo que no es estatico como por ejemplo “id” mediante el contador “++”. Este atributo no estatico podemos usarlos en los objetos y asi saber que numero dentro de la cantidad de objetos que fueron creados es ese objeto como si fuera un “id unico”. Podemos crear un método estatico para esa variable estatico y que con RETURN nos devuelva el valor(que en este caso seria el total del contador).

IMPORTANTE: Si modificamos el atributo/método estatico de la clase, todos los objetos que tengamos verán este cambio

Ya en el MAIN(que es estatico), cuando creemos el objeto, y usemos el método estatico, tenemos que poner nombreClase.metodoEstatico(); para acceder a este método estatico ya que no es del objeto sino de la clase.

public class EjemploEstatico {  
 public static void main(String[] args) {  
 Persona persona1 = new Persona("Juan")**;** System.*out*.println(persona1)**;** Persona persona2 = new Persona("Karla")**;** System.*out*.println(persona2)**;** System.*out*.println("ContadorPersonas = " + Persona.*getContadorPersonas*())**;** }  
}  
  
class Persona{  
 private int idPersona**;** private String nombre**;** private static int *contadorPersonas***;** public Persona(String nombre){  
 this.idPersona = *contadorPersonas*++**;** this.nombre = nombre**;** }  
  
 public int getIdPersona(){  
 return this.idPersona**;** }  
  
 public void setIdPersona(int idPersona){  
 this.idPersona = idPersona**;** }  
  
 public String getNombre(){  
 return this.nombre**;** }  
  
 public void setNombre(String nombre){  
 this.nombre = nombre**;** }  
  
 public static int getContadorPersonas(){  
 return *contadorPersonas***;** }  
  
 public String toString(){  
 return "idPersona: " + idPersona + " nombre: " + nombre**;** }  
  
}

CONTEXTO DINAMICO: Una vez que la clase ya esta cargada en memoria, es posible empezar a crear objetos de dicha clase a esto se le conoce como contexto dinamico, en donde se cargan los objetos.

TIP: METODO TOSTRING(): Nos permite concatenar los atributos no estáticos, lo podemos usar si queremos imprimir todos los atributos.

Como lo llamamos, no se llama se hace automáticamente mediante el println del SOUT

HERENCIA EN JAVA: Se trata de la herencia de codigo de dos o mas clases entre ellas principalmente varias clases heredan de una clase-padre, las clases-hijos heredan todas los atributos, métodos, etc. Esto nos viene bien cuando queremos evitar duplicación de código, para diseñar clasa mas parecidas al mundo real. En donde los atributos y métodos o constructor que tenga la clase-padre serán el código general que heredaran las clases-hijos, y las clases-hijos además de heredar, tendrán sus propios métodos y atributos específicos de su clase, es decir, podemos tomar la clase padre como el conjunto de atirbutos y métodos que las clases hijos comparten entre si reunidas en un sola clase que a su vez las clases-hijos heredan.

Es importante saber que si usamos en la clase-padre con PRIVATE en algunos atributos o métodos, no se podrán heredar a las clases-hijos, únicamente con PUBLIC, pero al ser public todos lo verán si fuera en la vida real, por lo cual usamos PROTECTED para heredar clase-padre a clase-hijos sin tener que usar PUBLIC y asi saber que son seguras.

Para poner en la clase hijo la clase padre usamos EXTENDS:

Class claseHijo extends ClasePadre

Para acceder a los atributos o métodos heredados de la clase-padre. Usamos SUPER, podemos usar SUPER para llamar al constructor de la clase-padre, en dode los atributos del constructor-padre, los ponemos en el constructor-hijo con SUPER. IMPORTANTE: El super debe ser la primera línea de código del constructor-hijo, debido a que va aser la llamada al constructor-padre.

RECUERDA: El nombre del constructor será el mismo que el de la clase

Se podría decir que la clase-padre, hereda siempre de una clase llamada “Object” en Java que viene predeterminada en todas las clases, siempre y cuando no usemos herencias, si las usamos pues obviamente esta clase heredara a la clase-padre, no hay necesidad de poner extends aquí, ya que viene predefinida, además podemos usar los “metodos” de esa clase “Object” y una de ellas es “toString”

METODO “toString()”: Lo usaremos para mostrar los valores de los atributosen cierto momento

Para poner el toString, vamos a la opción Code, arriba a la izquierda del programa en la barra de opciones, damos en Generate y damos a toString(), nos mostrara una pantalla en donde nos mostrara los atributos que se ingresaran en toString(), damos OK y listo.

Si vamos a usar un método de la clase-padre dentro de la clase-hijo, es decir, sobreescribir en el código del método ya definido con su propio codigo en la clase-padre, usamos el @override tanto en la clase-padre como en la clase-hijo para indicar que ese método viene de una clase superior al que se esta usando. Es importante saber que para llamar a ese método-padre al método-hijo usamos también SUPER, porque?, porque se sobreescribe mas no se elimina el código orginal de ese método, por lo cual debemos llamar a ese método-padre para poder usarlo también en el método-hijo.

El @override se pondrá solo cuando usemos el toString.

TIP: Podemos hacer que JAVA nos genere métodos GET y SET de cada atributo si los necesitamos para ello damos click derecho en el código, damos click en REFACTOr y damos click despuesen ENCAPSULATE FIELDS, nos abrirá un cuadro nos mostrara las opciones de que atributos queremos ponerle SET y GET, quitamos la opción de “Use accessors even when field is accesible”, damos en Refactor, y ya los tendremos.

Tipo de dato: DATE: Nos permite crear una nueva fecha, pero para que nos funcione tenemos que importar la librería(paquetes) propia de JAVA que es : import java.util.Date; y para poner la fecha actual usamos Date()

EJM:

Clase padre:

package Herencia**;**public class PersonaPadre {  
  
 private String nombre**;** private char genero**;** private int edad**;** private String direccion**;** //Constructor vacio  
 public PersonaPadre(){}  
  
 //Constructor con 1 argumento  
 public PersonaPadre(String nombre){  
 this.nombre = nombre**;** }  
  
 //Constructor completo  
 public PersonaPadre(String nombre**,** char genero**,** int edad**,** String direccion){  
 this.nombre = nombre**;** this.genero = genero**;** this.edad = edad**;** this.direccion = direccion**;** }  
  
  
 public String getNombre() {  
 return nombre**;** }  
  
 public void setNombre(String nombre) {  
 this.nombre = nombre**;** }  
  
 public char getGenero() {  
 return genero**;** }  
  
 public void setGenero(char genero) {  
 this.genero = genero**;** }  
  
 public int getEdad() {  
 return edad**;** }  
  
 public void setEdad(int edad) {  
 this.edad = edad**;** }  
  
 public String getDireccion() {  
 return direccion**;** }  
  
 public void setDireccion(String direccion) {  
 this.direccion = direccion**;** }  
  
 @Override  
 public String toString() {  
 return "Persona{" +  
 "nombre='" + nombre + '\'' +  
 ", genero=" + genero +  
 ", edad=" + edad +  
 ", direccion='" + direccion + '\'' +  
 '}'**;** }  
}

Clases Hijos:

package Herencia**;**public class Empleado extends PersonaPadre{  
  
 private int idEmpleado**;** private double sueldo**;** private static int *contadorEmpleados***;** public Empleado(String nombre**,** double sueldo){

// En Super podemos meter el o los atributos que queramos colocar,

Lo colocamos entre (), recordando también que los atributos que vayamos a poner también los ponemos en los parámetros del constructor  
 super(nombre)**;** this.idEmpleado = ++*contadorEmpleados***;** this.sueldo = sueldo**;** }  
  
 public int getIdEmpleado() {  
 return idEmpleado**;** }  
  
 public void setIdEmpleado(int idEmpleado) {  
 this.idEmpleado = idEmpleado**;** }  
  
 public double getSueldo() {  
 return sueldo**;** }  
  
 public void setSueldo(double sueldo) {  
 this.sueldo = sueldo**;** }  
  
 @Override  
 public String toString() {  
 return super.toString() + "Empleado{" +  
 "idEmpleado=" + idEmpleado +  
 ", sueldo=" + sueldo +  
 '}'**;** }  
}

package Herencia**;**import java.util.Date**;**public class Cliente extends PersonaPadre {  
 private int idCliente **;** private Date fechaRegistro**;** private boolean vip**;** private static int *contadorClientes***;** public Cliente(Date fechaRegistro**,** boolean vip){  
 this.idCliente = ++*contadorClientes***;** this.fechaRegistro = fechaRegistro**;** this.vip = vip**;** }  
  
  
 public int getIdCliente() {  
 return idCliente**;** }  
  
 public void setIdCliente(int idCliente) {  
 this.idCliente = idCliente**;** }  
  
 public Date getFechaRegistro() {  
 return fechaRegistro**;** }  
  
 public void setFechaRegistro(Date fechaRegistro) {  
 this.fechaRegistro = fechaRegistro**;** }  
  
 public boolean isVip() {  
 return vip**;** }  
  
 public void setVip(boolean vip) {  
 this.vip = vip**;** }  
  
 @Override  
 public String toString() {  
 return super.toString() + "Cliente{" +  
 "idCliente=" + idCliente +  
 ", fechaRegistro=" + fechaRegistro +  
 ", vip=" + vip +  
 '}'**;** }  
}

Creacion y ejecución de los objetos:

package Herencia**;**import java.util.Date**;**public class EjemploHerencia {  
 public static void main(String[] args) {  
 Empleado empleado1 = new Empleado("Juan"**, 25000**)**;** empleado1.setEdad(**28**)**;** empleado1.setGenero('M')**;** empleado1.setDireccion("Lomas #13")**;** System.*out*.println("empleado1 = " + empleado1)**;** Cliente cliente1 = new Cliente(new Date()**,** true)**;** cliente1.setNombre("Karla")**;** cliente1.setEdad(**31**)**;** cliente1.setGenero('F')**;** cliente1.setDireccion("Rayuela #10")**;** System.*out*.println("cliente1 = " + cliente1)**;** }  
}

SOBRECARGAS DE CONSTRUCTORES: Se trata del uso de mas de un constructor en una clase, a su vez un constructor cargado es aquel que tiene parámetros. Tambien podemos poner un constructor en PRIVATE y posteriormente usarlo en otro constructor que sea PUBLIC para que este pase de ser PRIVATE a PUBLIC, a su vez para meter ese constructor dentro del otro usamos THIS, ya que al final son parte también de esa clase. Pero la diferencia aquí viene en que no ponemos el nombre del constructor sino simplemente usamos () ASi: this.(); . Ejm:

private Persona() {  
   
 this.idPersona = ++contadorPersonas**;**}  
  
public Persona(String nombre**,** int edad) {  
   
 this.()**;** this.nombre = nombre**;** this.edad = edad**;**}

Julio 23/2019

Sobrecarga de Metodos: Es ofrecer mas de una opción de algunos de los métodos definidos en la clase, para que exista sobrecarga de métodos, deben haber dos o mas métodos con el mismo nombre de método pero con tipo de datos y argumentos deifenrentes. Para que el recopilador sepa que hay una sobrecarga es importante que los métodos con el mismo nombre que el método original deben tener diferentes los argumentos.

Ejm:

Int sumar(int a, int b){

Return a + b;

}

// Metodo con el mismo nombre pero con diferente parámetros, ene ste caso el tipo de dato

double sumar(double a, double b){

Return a + b;

}

El orden en los argumentos en JAVA si importa, ya que en este caso pueden haber otros métodos de sobrecarga, en donde los parámetros(que son igual a argumentos):

1. Double a, int b
2. Int a, double b

Es decir, aunque los dos tengan double e int. Nos la mismo la primera forma que la segunda.

IMPORTANTe: Obviamente el compilador revisa el tipo de dato, no el nombre del parámetro porque no influye en nada, ya que puede ser cualquier nombre. Tampoco importa el tipo de dato en el RETURN que definimos al inicio del método antes del nombre del método. Solamente se revisan si los argumentos son diferentes, es decir, el orden y el tipo de parámetros/argumentos. El método sobrecargado puede estar en la misma clase o en una clase padre, los argumentos pueden ser tipos primitivos o object.

Si no podemos usar tipo Long, entonces es mejor usar Double, en caso de numero ya que tiene mas bites que el tipo int.

Long puede ser double o float,

Int puede ser float o char

Float puede ser double

Int puede puede ser double si llega el caso de que necesitemos decimales.

Char puede ser int, un Short puede ser int

EJM sobrecarga Metodos:

Aquí en este ejemplo usamos los métodos en modo estatico por lo cual podemos usarlos irectamente en el MAIN sin crear los objetos.

public class Operaciones {  
  
 //Metodo sumar  
 public static int sumar(int a**,** int b){  
 System.out.println("metodo sumar(int, int)")**;** return a+b**;** }  
  
 //Sobrecarga de metodo sumar  
 public static double sumar(double a**,** double b){  
 System.out.println("metodo sumar(double, double)")**;** return a+b**;** }  
  
 public static double sumar(int a **,** double b){  
 System.out.println("metodo sumar(int, double)")**;** return a+b**;** }  
  
 public static double sumar(double b**,** int a){  
 System.out.println("metodo sumar(double, int")**;** return a+b**;** }  
}

MAIN:

public class SobrecargaMetodos {  
 public static void main(String[] args) {  
 System.out.println("Resultado 1:" + Operaciones.sumar(**3,4**))**;** System.out.println("Resultado 2: " + Operaciones.sumar(**5,4.1**))**;** System.out.println("Resultado 3:" + Operaciones.sumar(**7.1, 6**))**;** //¿Cual es el metodo que se manda llamar?  
 System.out.println("Resultado 4:" + Operaciones.sumar(**3, 4L**))**;** System.out.println("Resultado 5:" + Operaciones.sumar(**5F,** 'A'))**;** }  
}

PAQUETES EN JAVA:

Es importante definir la organización de las claes, los paquetes/carpetas nos permiten organizar clases, evitan conflictos entre nombres de las clases, limitan el acceso a sus clases y también el JAVa tiene su propia biblioteca de paquetes esxistentes:

* Convencion en nombre de paquetes en Java:

1. Todo el nombre del paquete debe estar en minúsculas
2. Se acostumbra a escribir el nombre del dominio web de manera invertida. Ejm: global.com -> com.global
3. Ejemplo de proyecto, si este tiene dominio: com.global.miproyecto
4. Ejemplo de subpaquetes los cuales serian en este ejemplo contabilidad y administracion: com.global.contabilidad.miproyecto, com.global.adminsitracion.miproyecto

Si queremos importar un paquete ponemos al principio del código:

Import com.dominio.nombreClase;

Y al llama run método en MAIN ponemos nombreClase.nombreMetodo();

Si queremos importar todas las clases de un paquete ponemos en nombreClase un \*, que nos indica que vamos a importar todas las clases. Si la clase tiene métodos o atributos privados pues obviamente no vamos apoder usarlos.

IMPORT ESTATICO: En JAVa es posible importar los métodos estáticos y de esta manera simplificar la sintaxis de los emtodos estáticos. Como importamos un método estatico:

Import static com.dominio.nombreClase.nombreMetodoEstatico;

Ponemos STATIC para asignar que ese métodos que vamos a importar es STATIC, como sedijo anteriormente esto nos benedifica en la sintaxis cuando vamos a escribir en el código MAIN el método estatico, ya que simplemente es colocar el nombre del Metodo Estatico.

Algunos d ellos paquetes mas importantes en JAVA: java.lang(contiene las clases escenciales, se importa implícitamente sin necesidas de una sentencia import, java.util, java.io, java.net, java.applet, java.awt

Otra forma sin el STATIC IMPORT/ IMPORT pero precisamente con mas sintaxis/codigo al momento de poner el método ESTATICO en el MAIn es poner lo del IMPORT en el MAIN, asi:

Com.dominio.nombreClase.nombreMetodo();

EJM:

PAQUETE con clase:

package com.gm**;**public class Utileria {  
 public static void imprimir(String s){  
 System.out.println("s = " + s)**;** }  
}

archivo donde vamos a importar el paquete:

import static com.gm.Utileria.imprimir**;**public class EjemploPaquetes {  
 public static void main(String[] args) {  
 imprimir("Hola")**;** }  
}

USO DE LA PALABRA “FINAL” EN JAVA:

FINAL en JAVA: Es como crear una constante en JAVA

Usos:

1. En variables: Evita cambiar el valor que almacena la variable, es decir, cuando la inicalizamos ya no es posible modificar su valor. Recordemos que en el caso de las variable donde creamos los objetos, están no es que tengan el valor del objeto sino que tiene la referencia dondese ubica el mismo, en donde es posible cambiar el estado(public, private, etc) pero no el valor que en este caso es la referencia. Comunmente en variables que tengan la palabra FINAL tiene también STATIC, ya que es mas cómodo acceder directamente a ella, ya que para que vamos a crear un objeto si esta variable no puede cambiar su valor porque es una constante. Las constantes en JAVA comúnmente se ponen en MAYUSCULAS.
2. FINAL en métodos: Al hacer esto estamos indicando que una subclase no puede sobreescribir(polimorfismo)/modificar en ese emtodo porque ese método al igual que en la variable no puede cambiar
3. En clases: Evita que se cree una subclase.

EJM: Math.PI; // PI ya tiene un valor definido por lo cual es una constante

En clases teneoms: public final class nombreClase{}

En variables con STATIC: public static final int variable = VALOR/REFERENCIA OBJETO;

// Al usar referencia objeto en FINAL no podemos cambiar la referencia o creación del objeto pero si el contenido del objeto, como?, asi:

Al ser STATIC:

nombreClase.nombrevariableObjeto.nombreMetodo().

Como vemos estamos accediendo a los métodos de la variable que esSTATIC y que tambien tiene FINAL para modiciar el contenido de ese objeto. Esta variable que tiene la referencia del objeto la creamos en la clase donde le ponemos FINLA y STATIC, esto para que nos de en MAIN y podamos acceder también a los métodos.

En métodos: public final void/tipoDato metodoNombre() {}

Ejm: public final class ClaseFinal {  
  
 //Variables marcadas como finales  
 public static final int VAR\_PRIMITIVO = **10;** // CReacion de la variable qu contiene la referncia del objeto en la clase para ya en MAIN poder usarla con los objetos, respecto  
 // a cambiar el contenido mas no la referencia del objeto ya que tiene FINAL  
  
 public static final Persona VAR\_PERSONA = new Persona()**;** //metodo final, lo estudiaremos a detalle en el tema de sobreescritura  
 public final void metodoFinal(){  
  
 }  
  
}  
  
// No nos dejara crear una sub clase  
//class Hija extends ClaseFinal{  
  
// El metodo con final no nos dejara modificar el metodo en la subclase  
// public final void metodoFinal(){  
// }  
//}

Archivo con MAIN:

public class EjemploPalabraFinal {  
 public static void main(String[] args) {  
 //Modificar un atributo final, no es posible  
 //ClaseFinal.VAR\_PRIMITIVO = 15;  
  
 //Modificar la referencia de un atributo de tipo Object, no es posible  
 //ClaseFinal.VAR\_PERSONA = new Persona();  
  
 ClaseFinal.VAR\_PERSONA.setNombre("Juan")**;** System.out.println(ClaseFinal.VAR\_PERSONA.getNombre())**;** ClaseFinal.VAR\_PERSONA.setNombre("Carlos")**;** System.out.println(ClaseFinal.VAR\_PERSONA.getNombre())**;** }  
}

ARRAYS en JAVA: Los arrays son grupos de datos los cuales nos permiten tener información, los cuales se dividen por elementos de información.

Es importante saber que los elementos de un array, tienen una llave, es decir llave->valor. Donde la llave a menos que se lo cambiemos por otro tipo de dato, serán números o INT, y el valor será el elemento que nosotros le demos. Estas llaves tienen una secuencia, es decir, cada vez que agreguemos un elemento al array, llave va a ir aumentando. Las llaves empiezan desde 0, y cada vez que vamos introduciendo elementos se aumenta, en cambio si borramos algún elemento disminuye. Ejm pero no es sintaxis de JAVA:

Array = [

0 => “HOla”

1=> “Como estas”

2=> 55

3=> 4

4=> “Hola Mundo”

]

Solamente las llaves con INT cambiaran a menos que se lo cambiemos.

ARRAYS sintaxis:

Declaracion de un ARRAy en JAVA:

Tipo [] nombreArray

O también:

Tipo nombreArray []

EJM con tipo primitivo:

Int [] enteros, bollean [] banderas;

EJM con tipo Object: En donde almacenen referencias de objetos

Persona[] personaArray;, // Como vemos usamos el nombre de la clase para ser un tipo Object

Con STRINGS:

String[] nombres;

INSTANCIAR ARRAYS:

Ahora que ya esta declarada, debemos introducir la cantidad de elementos que tendrá ese array. Tendremos que crear un array como si fuera un objeto

Sintaxis: nombreArray = new tipo[CantidadDeElementos];

EJM con primitivos: enteros = new int[10];

EJM con Objetcs: personaArray = new Persona[13];

EJM con Strings: nombres = new String[8];

INICIALIZAR LOS ELEMENTOS DE UN ARRAY: Ahora que ya tenemos la cantidad, vamos a poner valores a los elementos del array de la siguiente manera:

nombreArray[posicionNumero/índice] = valor;

donde: Indice: es la posición en la que queremos colocar el valor dentro del array.

EJM primitivos:

Enteros[0] = 15;

Enteros[1] = 55;

EJM tipo OBJECT:

personaArray[0] = new Persona();

EJM String:

Nombres[0] = “Juan”;

Si ponemos mas elementos de los que inicialmente le asignamos, nos dará errro obviamente, además si no rellenamos toda la cantidad del array antes iniciamos, están se rellenaran solas con el DAFAULT del tipo de dato que estemos usando.

EXTRAER ELEMENTOS DE UN ARRAY: Si queremos usar un elemento de un array,

Sintaxis: tipo de dato variableReceptora = nombreArray[indice];

Int variable = enteros[0];

Persona variable = personaArray[0]

String variable = nombres[1]

Julio 25/2019

Sintaxis Simplificada de ARRAYS: Si queremos menos cidog y mas simplificado la delcaracion, instanciar e inicializar los elementos de un ARRAY. Podemos hacer de la siguiente manera:

tipoDato [] nombreArray = {lista con los valores/elementos separados por “,”}.

En donde: [] son para indicar que vamos a crear un array.

Ya de por si al colocar los valores dentor de los {} ya no necesitamos indicar la cantidad de elementos del array como tampoco ir en cad aposicion/index del array colocando los valores, esto es mas cómodo respecto a ala sintaxis. Ademas que simplimeente al colocar los valores dentro de los {} ya de por si estamos indicando la cantidad de valores/elementos.

EJM:

Int [] edades = {10, 23, 41, 68, 7}

Persona[] personaArray = {new Persona(), new Persona(“Juan”)}

String nombre = {“Karla”, “Arturo”, “Antonio”, “Roberto”}

Pero la cuesitoj aquí es que solo podemos usar esta sintaxis si solo sabemos la cantidad de Vlores que tendrá ese array, si queremos una cantidad pero no rellenarla por completa usamos la forma como se enseño antes.

EJM de BUCLE FOR para recorrer/visualizar el array, en donde cada vuelta del bucle va revisando/recorriendo cada posición:

LENGHT en un ARRAY: Nos permite contar la cantidad de elementos que tiene un array, esto nos viene bien en un bucle ya que si “i” < nombreArray.length, “i” será menor a la cantidad de ese array. Por lo cual cuando i haga las vueltas necesarias parara cuando ve que esta en la ultima vuelta, ya que le indicamos que tiene que ser menos a la cantifaf total de elementos de array. EJM:

For (int i = 0; i < nombre.length; i++){

Nombre[i]

}

Asi que por cada vueta que de imprimirá en este caso el elemento del array en el que este

public class EjemploArreglos {  
 public static void main(String[] args) {  
 //1. Declaremos un arreglo de enteros  
 int edades[]**;** //2. Instanciar el arreglo de enteros  
 edades = new int[**3**]**;** //3. Inicializamos los valores del arreglo de enteros  
 edades[**0**] = **30;** edades[**1**] = **15;** edades[**2**] = **20;** //Imprimimos los valores del arreglo a la salida estandar  
 //4. leemos cada valor del arreglo  
 System.out.println("Arreglo enteros indice 0: " + edades[**0**])**;** System.out.println("Arreglo enteros indice 1: " + edades[**1**])**;** System.out.println("Arreglo enteros indice 2: " + edades[**2**])**;** //1.Declarar un arreglo de tipo object  
 Persona personas[]**;** //2. Instanciar el arreglo de tipo object  
 personas = new Persona[**4**]**;** //3. Inicializar los valores del arreglo  
 personas[**0**] = new Persona("Juan")**;** personas[**1**] = new Persona("Karla")**;** //4. imprimir los valores del arreglo  
 System.out.println("Arreglo personas indice 0: " + personas[**0**])**;** System.out.println("Arreglo personas indice 1: " + personas[**1**])**;** System.out.println("Arreglo personas indice 2: " + personas[**2**])**;** System.out.println("Arreglo personas indice 3: " + personas[**3**])**;** //Arreglo utilizando notacion simplificada  
 String nombres[] = {"Sara"**,** "Laura"**,** "Carlos"**,** "Carmen"}**;** //imprimir los elementos de un arreglo  
 for(int i=**0;** i < nombres.length **;** i++){  
 System.out.println("Arreglo String indice " + i + ": " + nombres[i])**;** }  
  
 }  
}

MATRICES EN JAVA: Las matrices en JAVA son básicamente dos arrays los cuales albergan en ellos una “tabla” por asi decirlo, en donde en cada cuadro de la tabla alberga un valor y este valor tendrá dos posiciones: posicoin de la columna y la posición de la fila

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |
| 0 | 78 | 66 | 78 | 87 |
| 1 | 434 | 566 | 66 | 989 |
| 2 | 98 | 989 | 98 | 99 |

EJM: el 566 esta en las posiciones: columna: 1, fila: 1.

Por lo cual un array se encargara de las posiciones de las columnas, mientras que el segundo array se encargara de las posiciones de las filas, dando lugar a que al momento de querer extraer un valor de una matriz, tengamos dos []. Donde un [fila] mientras que el otro será [columna]. Por lo cual si queremos extraer un valor tendremos que poner en que posiciones se encuetra ese valor y definirla dentro de los [].

DECLARACION DE UNA MATRIZ:

tipoDato [] [] nombreMatriz;

o también:

tioDato nombreMatriz [] [];

Ejm primitivo:

Int [] [] enteros;

Boolean banderas[] []

Ejm tipo Object:

Persona [] [] personaMatriz;

String nombre [] [];

INSERCION DE UNA MATRIZ: Donde especificaremos la cantidad de elementos que tendrán las filas y las columnas de la matriz

nombreMatriz = new tipoDato[filas] [columnas]

Ejm:

Enteros = new int [2] [2]

PersonaMatriz = new Persona[4] [2]

Nombres = new String[5] [5]

INSERTAR VALORES EN LAS MATRICES:

nombreMatriz[posición\_fila][posición\_columna] = valor;

Ejm:

Enteros[0][0] = 15;

personaMatriz[1][1] = new Persona()

nombres[4][3] = new String(“String”)

EXTRAER ELEMENTOS DE UNA MATRIZ:

VariableReceptora = nombreMatriz[posición\_fila][posición\_columna]

AL igual que en el array, la miatriz también tiene una forma mas rápida de hacer todas estas cosas con menos código:

tipoDato [] [] nombreMatriz = {{lista valores(posicionFila 0)}}, {lista de valores(posicionFila 1)}},

La cuestión importante aquí es que los valores que pongamos dentro de los segundos {}. Seran los elementos que iran en una fila, al igual que en array este conjunto de elementos que serán una fila, tendrán su posición en el {} principal. Donde la primera fila de valores ser ael 0, y asi va aumentando si creas mas filas de valores. Y donde quedan las columnas?, muy sencillo las columnas serán la posición de cada valor dentro del subcorchetes{}

. Donde el primero será 0 y asi sucesivamente según cuantos elementos tenga esa fila. EJM:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Int [] [] edades = | {fila0{10(columna0),23(columna1),41(columna2)}, | {fila1{23(columna0),22(columna1),21(columna2)}, | {fila2{‘1(columna0),10(columna1),10(columna2)}, | } |

Como imprimir cuantas filas y columnas tiene una matriz?:

Con filas: como te imaginaras las filas son las posiciones del {} principal por lo cual no tenemos que poner [] para especificar.

Con columnas: En cambio, las columnas que son las posiciones de los valores dentro del subconrchete{}, si tenemos que usar [] para indicar cual fila queremos ver y cuantas columnas de esa fila tiene. Lo que siempre nos dará el mismo numero porque todas las filas tendrás esa misma cantidad de columnas, es decir, no puede haber una fila con un elemento sin columna.

Y obviamente usando el .length para que nos cuente la cantidad:

Filas:

Edades.lenght;

Columnas:

edades[0].length.

FOR con matriz para imprimir los valores: Haremos dos bucles de FOr, porque el primero será para las filas, mientras que el segundo será para las columnas, asi:

public class EjemploMatrices {  
 public static void main(String[] args) {  
 //1. Declarar una matriz de int  
 int edades[][]**;** //2. Instanciar la matriz de int  
 edades = new int[**3**][**2**]**;** //3. instanciar los valores de la matriz  
 edades[**0**][**0**] = **30;** edades[**0**][**1**] = **15;** edades[**1**][**0**] = **20;** edades[**1**][**1**] = **45;** edades[**2**][**0**] = **5;** edades[**2**][**1**] = **38;** //4.imprimir  
 System.out.println("Matriz enteros indice 0 0: " + edades[**0**][**0**])**;** System.out.println("Matriz enteros indice 2 0: " + edades[**2**][**0**])**;** //1. Declarar e instanciar matriz tipo object  
 Persona personas[][] = new Persona[**1**][**2**]**;** //2. inicializamos valores  
 personas[**0**][**0**] = new Persona("Juan")**;** personas[**0**][**1**] = new Persona("Karla")**;** //3. imprimimos valores  
 System.out.println("Matriz persona indice 0 0: " + personas[**0**][**0**])**;** System.out.println("Matriz persona indice 0 1: " + personas[**0**][**1**])**;** //System.out.println("Matriz persona indice 0 0: " + personas[0][2]);  
  
 System.out.println()**;** for (int i = **0;** i < personas.length**;** i++){  
 for(int j=**0;** j < personas[i].length**;** j++){  
 System.out.println("Matriz personas indice: " + i + " - " + j + ": " + personas[i][j])**;** }  
 }  
 System.out.println()**;** String nombres [][] = {{"Teresa"**,** "Cesar"**,**"William"}**,** {"Yesenia"**,** "Esteban"**,** "Maria"}}**;** for (int i = **0;** i < nombres.length**;** i++){

La “i” la ponemos en nombre[i].length porque “i” como vemos es la fila actual de esa vuelta.  
 for(int j=**0;** j < nombres[i].length**;** j++){  
 System.out.println("Matriz nombres indice: " + i + " - " + j + ": " + nombres[i][j])**;** }  
 }  
  
  
 }  
}