**Sintaxis Java**

////////////////////////////////////////

Sebastian Bedoya 311 786 3286

Juan Durango 301 392 5911

Jhon Sanchez iroh314@gmail.com

///////////////////////////////////////////////////

Its used **to develop** mobile apps, web apps, desktop apps, games and much more

public class MyClass {

public static void **main**(String[] args) {

System.out.println("Hello World");

}

}

**MyClass** myObj = new **MyClass**();

Math.**max**(x,y)

**Características:**

Trabaja en diferentes plataformas

Open-source

Free

Gran comunidad

Es OO

java **-version**

Bajar java:

<https://www.oracle.com/java/technologies/javase-jdk15-downloads.html>

En las variables del sistema en windows > Variables de entorno > Variables del sistema > path > Editar > Nuevo > C:\Program Files\Java\jdk-15\bin

Every application begins with a class name, and that class **must match** the filename

Compilar:

**javac** MyClass.java

Ejecutar:

**java** MyClass

Instalar el tool para Eclipse:

<https://spring.io/tools>

/////////////////////////////////////////

Every line of code that runs in Java must be **inside a class**

A class should always start with an **uppercase first letter**

Is **case-sensitive**

**Every program** must contain the main() method

String double quotes

int

float

**char** - single characters surrounded by single quotes

boolean -true or false

**String** name = "John";

**int** myNum = 15;

variables **final** no pueden ser modificadas

**float** myFloatNum = 5.99f;

**double** myNum = 19.99d;

**char** myLetter = 'D';

**Válidos**:

nombre1

flauta\_2

$mouse

\_hoja

**Inválidos:**

EmpiezoConMayuscula

tieneEspacioAlINicio

int

///////////////////////////////////////

Two groups:

**Primitive data types:**

byte, short, int, long, float, double, boolean and char

Has always a value

Cannot call methods to perform certain operations

Starts with a lowercase letter

The size of depends on the data type

**Non-primitive data types:**

String, Arrays and Classes

Called reference types because they refer to objects

Are created by the programmer

Can be used to call methods to perform certain operations

Can be null

Starts with an uppercase letter

They all have the same size

byte 1 byte -128 to 127

short 2 bytes -32,768 to 32,767

int 4 bytes -2,147,483,648 to 2,147,483,647

long 8 bytes -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

float 4 bytes 6 to 7 decimal digits

double 8 bytes 15 decimal digits

boolean 1 bit

char 2 bytes a single character/letter or ASCII value

With long you should **end the value with** an "L"

Float can store fractional numbers from 3.4**e**−038 to 3.4e+038

The **precision** of float is only six or seven decimal digits, while double variables have a precision of about 15 digits

Scientific number:

float f1 = 35**e**3f;

double d1 = 12**E**4d;

**ASCII** values:

char a = 65

//////////////////////////////////

**Widening Casting** (automatically) byte -> short -> char -> int -> long -> float -> double

**Narrowing Casting** (manually) double -> float -> long -> int -> char -> short -> byte

Automatic casting: int to double:

double myDouble **=** myInt;

Manual casting: double to int:

int myInt = (**int**) myDouble;

+=

!=

**&&**

**||**

!() not

////////////////////////////////

txt.**toUpperCase**();

txt.toLowerCase();

Indice de la primer ocurrencia:

txt.indexOf("locate")

**0 is t**he first position in a string

firstName.**concat**(lastName)

**\n** New Line

\r Carriage Return

\t Tab

\b Backspace

\f Form Feed

////////////////////////////////////////

String x = "10";

int y = 20;

String z = x + y; // z will be **1020** (a String)

Math.**max**(5, 10);

Math.**sqrt**(64);

Math.**abs**(-4.7);

Entre 0.0 y sin llegar a 1.0

Math.**random**();

int randomNum = (int)(Math.**random() \* 101**); // 0 to 100

////////////////////////////////////////////////////////////////////////

String result = (time < 18) **?** "Good day." : "Good evening.";

int day = 4;

**switch** (day) {

case 1:

System.out.println("Monday");

break;

case 7:

System.out.println("Sunday");

break;

}

The **default** keyword specifies some code to run if there is no case match

//////////////////////////////////////////////////////////////////////////////

int i = 0;

**while** (i < 5) {

System.out.println(i);

i++;

}

int i = 0;

do {

System.out.println(i);

i++;

}

**while** (i < 5);

**String[]** cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (**String i : cars**) {

Los imprime todos menos el 4:

for (int i = 0; i < 10; i++) {

if (i == 4) {

**continue**;

}

System.out.println(i);

}

}

////////////////////////////////////////

String[] cars;

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

cars.length;

for (String i : cars) {

int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

charAt() Returns the character at the specified index (position) char

codePointAt() Returns the Unicode of the character at the specified index int

codePointBefore() Returns the Unicode of the character before the specified index int

codePointCount() Returns the Unicode in the specified text range of this String int

compareTo() Compares two strings lexicographically int

compareToIgnoreCase() Compares two strings lexicographically, ignoring case differences int

concat() Appends a string to the end of another string String

contains() Checks whether a string contains a sequence of characters boolean

contentEquals() Checks whether a string contains the exact same sequence of characters of the specified CharSequence or StringBuffer boolean

copyValueOf() Returns a String that represents the characters of the character array String

endsWith() Checks whether a string ends with the specified character(s) boolean

equals() Compares two strings. Returns true if the strings are equal, and false if not boolean

equalsIgnoreCase() Compares two strings, ignoring case considerations boolean

format() Returns a formatted string using the specified locale, format string, and arguments String

getBytes() Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array byte[]

getChars() Copies characters from a string to an array of chars void

/////////////////////////////////////////////////////////////////////////////////////////////

hashCode() Returns the hash code of a string int

indexOf() Returns the position of the first found occurrence of specified characters in a string int

intern() Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index String

isEmpty() Checks whether a string is empty or not boolean

lastIndexOf() Returns the position of the last found occurrence of specified characters in a string int

length() Returns the length of a specified string int

matches() Searches a string for a match against a regular expression, and returns the matches boolean

offsetByCodePoints() Returns the index within this String that is offset from the given index by codePointOffset code points int

regionMatches() Tests if two string regions are equal boolean

replace() Searches a string for a specified value, and returns a new string where the specified values are replaced String

replaceFirst() Replaces the first occurrence of a substring that matches the given regular expression with the given replacement String

replaceAll() Replaces each substring of this string that matches the given regular expression with the given replacement String

//////////////////////////////////////////////////////////////////////////////////////////////

split() Splits a string into an array of substrings String[]

startsWith() Checks whether a string starts with specified characters boolean

subSequence() Returns a new character sequence that is a subsequence of this sequence CharSequence

substring() Extracts the characters from a string, beginning at a specified start position, and through the specified number of character String

toCharArray() Converts this string to a new character array char[]

toLowerCase() Converts a string to lower case letters String

toString() Returns the value of a String object String

toUpperCase() Converts a string to upper case letters String

trim() Removes whitespace from both ends of a string String

valueOf() Returns the primitive value of a String object String

//////////////////////////////////////////////////////////////////////////////////

JRE Java Runtime Environment (antes lo llamaban JVM)

El programa es compilado y luego interpretado por la JVM Java Virtual Machine

Bajar JRE >> instalarlo >> agregar en las variables del sistema en path

static means that the method belongs to the MyClass class and not an object of the MyClass class

static void myMethod(String fname, int age) {

///////////////////////////////////////////////////////////////////////////////

Paquetes -> java, javax

Java -> java.awt, java.util, java.io

Javax -> javax.activity, javax.annotation

Java.util -> java.util.regex

**Si** una clase tiene un **método abstracto**, todas las clases que hereden, deben sobrecargar ese método

Empleado **extends** Persona. Luego hay que llamar al constructor de la clase padre: **super**()

Un **socket necesita** la dir del sv y el puerto de recepción

En el cliente -> **OutputStream**

En el sv -> **InputStream**

Evento en un botón:

private class EnviaTexto implements ActionListener {

@Override

public void actionPerformed(ActionEvent e) {

try{

//esta es la ip del equipo q sale en ipconfig

Socket misocket = new Socket(“192.168.56.1”, 9999);

} catch (UnknownHostExcepton e1) {

E1.printStackTrace();

} catch (IOException e1) {

System.out.println(e1.getMessage());

}

}

}

/////////////////////////////////////////////////////////////

Procedural programming is about writing procedures or methods that perform **operations on the data**

**Over procedural programming, OOP:**

Is faster

Easier to execute

Provides a clear structure for the programs

Helps to keep the Java code DRY"Don't Repeat Yourself"

Makes the code easier to maintain, modify and debug

Makes it possible to create full reusable applications with less code and shorter development time

public **class** MyClass {

A class should always start with an **uppercase** first letter

MyClass **myObj** = new MyClass();

**Static**. It can be accessed without creating an object:

static void myMethod() {

myStaticMethod(); // Call the static method

///////////////////////////////////////////////////

**El constructor**

Tiene el mismo nombre que la clase

It cannot have a return type (like void)

If you do not create a class constructor yourself, Java creates one for you

**Default**. The class is only accessible by classes in the same package. This is used when you don't specify a modifier

**Final**. The class cannot be inherited by other classes

**Abstract**. Can only be used in an abstract class, and can only be used on methods. The method does not have a body

**Transient** Attributes and methods are skipped when serializing the object containing them

**Synchronized** Methods can only be accessed by one thread at a time

**Volatile** The value of an attribute is not cached thread-locally, and is always read from the "main memory"

////////////////////////////////////////////////////////////

**Encapsulation**

Is to make sure that "sensitive" data is hidden

Las variables son privadas y los getter y setter son los que accesan y actualizan

// **Getter**

public String getName() {

return name;

}

// **Setter**

public void setName(String newName) {

this.name = newName;

}

**Advantages of encapsulation**:

Flexible: the programmer can change one part of the code without affecting other parts

Increased security of data

Better control of class attributes and methods

////////////////////////////////////////////////////////

**Package**

Is used to group related classes

Think of it as a folder in a file directory. We use packages to avoid name conflicts, and to write a better maintainable code.

Two categories:

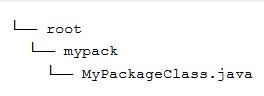
Built-in Packages (packages from the Java API)

User-defined Packages (create your own packages)

import package.name.Class; // Import a single class

import package.name.\*; // Import the whole package

To create your own package, you need to understand that Java uses a file system directory to store them. Just like folders on your computer:



package mypack;

class MyPackageClass {

public static void main(String[] args) {

System.out.println("This is my package!");

}

}

Save the file as MyPackageClass.java, and compile it:

javac MyPackageClass.java

-d destination:

javac -d . MyPackageClass.java

This forces the compiler to create the "mypack" package.

////////////////////////////////////////////////////////////////

Subclass -> child

Superclass -> parent

Una clase con final no puede ser heredada

**Interfaz**

Conjunto de directrices que debe cumplir la clase

Solo tiene métodos abstractos y constantes

Class Jefatura extends Empleado implements Interfaz1, Interfaz2

Public int compareTo(Object miObjeto) {

Empleado **otroEmpleado** = (Empleado) miObjeto;

If(this.sueldo < **otroEmpleado**.sueldo( {

///////////////////////////////////////////////////////////

**JDBC** Java Data Base Connectivity

Jdbc:**mysql**://localhost:9999/gestionPedidos

Driver:protocoloDriver

**Sql**:

Jdbc:odbc:DSN\_gestionPedidos

**Oracle**:

Jdbd:oracle:juand@servidor:9999:gestionPedidos

//////////////////////////////////////////////////////////

Convertir Eclipse SE en EE:

Eclipse > Help > Install new software > Work with > All available sites > Checar la opcion Web, XML…

Tomcat:

Localhost:8080

Para que Eclipse trabaje con Tomcat:

Eclipse > Window > Show View > Other > Server > Servers

Pestaña Servers > enlace No servers… > Select the server type > Apache > Tomcat (la versión que tenga instalada) > Next > Browse

En la pestaña servers debería aparecer Tomcat

/////////////////////////////////////////////////////////////////////////////

nextBoolean()

nextByte()

nextDouble()

nextFloat()

nextInt()

nextLine()

nextLong()

nextShort()

Java does not have a built-in Date class, but we can import the **java.time** package

LocalDate **(yyyy-MM-dd))**

LocalTime **(HH-mm-ss-ns))**

LocalDateTime  **(yyyy-MM-dd-HH-mm-ss-ns)**

**DateTimeFormatter**  Formatter for displaying and parsing date-time objects

LocalDate myObj = **LocalDate**.now();

LocalTime myObj = **LocalTime**.now();

LocalDateTime myObj = **LocalDateTime**.now();

LocalDateTime myDateObj = LocalDateTime.now();

System.out.println("Before formatting: " + myDateObj);

DateTimeFormatter myFormatObj = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");

String formattedDate = myDateObj.format(myFormatObj);

System.out.println("After formatting: " + formattedDate);

ofPattern()

yyyy-MM-dd "1988-09-29"

dd/MM/yyyy "29/09/1988"

dd-MMM-yyyy "29-Sep-1988"

E, MMM dd yyyy "Thu, Sep 29 1988"

//////////////////////////////////////////////////////////

**Bases de datos**

Principio de sustitución:

Clase Empleado -> Clase Jefatura

Empleado director\_comercial = new Jefatura(“Natalia”, 85000, 2011, 05, 06);

If (director\_comercial instanceof Jefatura)

En google:

Driver jdbc mysql

Click der sobre el proyecto > Properties > Libraries > Add external jar > ir donde está el archivo

try {

Connection miConexion = DriverManager.getConnection(“jdbc:mysql://localhost:3306/pruebas”, “root”, “”);

Statement miStatement = miConexion.createStatement();

ResultSet miResultset = miStatement.executeQuery(“Select \* from productos”);

while(miResultset.next()) {

System.out.println(miResultset.getString(“NombreArticulo”) + “ “ + miResultset.getString(“CodigoArticulo”));

}

} catch(Exception e) {

System.out.println(“No conecta”);

}

/////////////////////////////////////////////

**ArrayList**

**Resizable** array, which can be found in the **java.util** package

The size of an array cannot be modified

Elements **are** **objects**

ArrayList<String> cars = new ArrayList<String>();

cars.add("Volvo");

cars.get(0);

cars.remove(0);

cars.clear();

cars.size();

for (int i = 0; i < cars.size(); i++) {

for (String i : cars) {

Collections.sort(cars);

when:

You want to access random items frequently

You only need to add or remove elements at the end of the list

////////////////////////////////////////////

**LinkedList**

Is a collection which can contain many objects of the **same type**

Tiene los mismos métodos que ArrayList

Has a regular array inside it

Has a link to the first container and each container has a link to the next

**LinkedList<String>** cars = new LinkedList<String>();

when:

You only use the list by looping through it instead of accessing random items

You frequently need to add and remove items from the beginning or middle of the

list

addFirst()

addLast()

removeFirst()

removeLast()

getFirst()

getLast()

///////////////////////////////////////

**HashMap**

Store items in "key/value" pairs

It can store **different types**

**HashMap<String, String>** capitalCities = new HashMap<String, String>();

capitalCities.put("England", "London");

Print keys:

for (String i : capitalCities.keySet()) {

Print values:

for (String i : capitalCities.values()) {

//////////////////////////////////////////////

**HashSet**

Every item is **unique**

**HashSet<String>** cars = new HashSet<String>();

cars.contains("Mazda");

for (String i : cars) {

//////////////////////////////////////////

**Iterator**

Object that can be used to loop through collections

Get the iterator:

Iterator<String> it = cars.iterator();

while(it.hasNext()) {

System.out.println(it.next());

Trying to remove items using a for loop or a for-each loop would not work correctly because the collection is changing size at the same time that the code is trying to loop

//////////////////////////////////////////////////

**Wrapper Classes**

provide a way to use primitive data types as objects

Primitive Data Type Wrapper Class

byte Byte

short Short

int Integer

long Long

float Float

double Double

boolean Boolean

char Character

**Invalid**:

ArrayList<int> myNumbers = new ArrayList<int>();

Valid:

ArrayList<Integer> myNumbers = new ArrayList<Integer>();

To create a wrapper object, use the wrapper class instead of the primitive type. To get the value, you can just print the object:

**Character** myChar = 'A';

System.out.println(myChar);

get the value:

System.out.println(myChar.**charValue**());

String myString = myInt.toString();

System.out.println(myString.**length**());

///////////////////////////////////////////////////

**Exceptions**

try {

} catch (Exception e) {

} finally {

}

Create a custom error:

throw new ArithmeticException("Access denied - You must be at least 18 years old.");

////////////////////////////////////////////////////////////////////////////////////////////////

**Regular Expression**

Sequence of characters that forms a search pattern

Java does not have a built-in Regular Expression class, but we can import the java.util.regex package

Pattern Class - Defines a pattern (to be used in a search)

Matcher Class - Used to search for the pattern

PatternSyntaxException Class - Indicates syntax error in a regular expression pattern

Pattern pattern = Pattern.compile("w3schools", Pattern.CASE\_INSENSITIVE);

Matcher **matcher** = pattern.matcher("Visit W3Schools!");

boolean matchFound = **matcher**.find();

if(matchFound) {

Flags in the compile() method change how the search is performed. Here are a few of them:

Pattern.CASE\_INSENSITIVE: The case of letters will be ignored when performing a search.

Pattern.LITERAL: Special characters in the pattern will not have any special meaning and will be treated as ordinary characters

Pattern.UNICODE\_CASE: Use it together with the CASE\_INSENSITIVE flag to also ignore the case of letters outside of the English alphabet

[abc] from the options

[^abc] NOT between the brackets

[0-9] range 0 to 9

| any one of the patterns separated by | as in: cat|dog|fish

. just one instance of any character

^ at the beginning of a string as in: ^Hello

$ at the end of the string as in: World$

\d digit

\s whitespace character

\b beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b

\uxxxx Unicode character specified by the hexadecimal number xxxx

n+ at least one n

n\* zero or more occurrences of n

n? zero or one occurrences of n

n{x} a sequence of X n's

n{x,y} a sequence of X to Y n's

n{x,} a sequence of at least X n's

If your expression needs to search for one of the special characters you can use a backslash ( \ ) to escape them. In Java, backslashes in strings need to be escaped themselves, so two backslashes are needed to escape special characters. For example, to search for one or more question marks you can use the following expression: "\\?"

//////////////////////////////////////////////////////////////////////////////////////////

**Threads**

Do multiple things at the same time

Can be used to perform complicated tasks in the background

public class MyClass extends Thread {

public void run() {

public class MyClass implements Runnable {

public void run() {

public class MyClass extends Thread {

public static void main(String[] args) {

MyClass thread = new MyClass();

thread.start();

System.out.println("This code is outside of the thread");

}

public void run() {

System.out.println("This code is running in a thread");

When a class extends the Thread class, you cannot extend any other class, but by implementing the Runnable interface, it is possible to extend from another class as well, like: class MyClass extends OtherClass implements Runnable.

isAlive() method of the thread to check whether the thread has finished running before using any attributes that the thread can change.

while(thread.isAlive()) {

////////////////////////////////////////

**Lambda Expressions**

Short block of code which takes in parameters and returns a value

**To use more than one parameter**, wrap them in parentheses:

They have to immediately return a value, and they **cannot contain variables**, assignments or statements such as if or for. In order to do more complex operations, a code block can be used with curly braces. If the lambda expression needs to return a value, then the code block should have a return statement

**ArrayList**<Integer> numbers = new ArrayList<Integer>();

numbers.add(5);

numbers.add(9);

numbers.add(8);

numbers.add(1);

numbers.**forEach**( (n) -> { System.out.println(n); } );

Lambda expressions can be **stored in variables** if the variable's type is an interface which has only one method. The lambda expression should have the same number of parameters and the same return type as that method. Java has many of these kinds of interfaces built in, such as the Consumer interface (found in the java.util package)

import java.util.ArrayList;

import java.util.function.Consumer;

public class MyClass {

public static void main(String[] args) {

ArrayList<Integer> numbers = new ArrayList<Integer>();

numbers.add(5);

numbers.add(9);

numbers.add(8);

numbers.add(1);

Consumer<Integer> method = (n) -> { System.out.println(n); };

numbers.forEach( method );

To use a lambda expression in a method, the method should have a parameter with a single-method interface as its type. Calling the interface's method will run the lambda expression:

Create a method which takes a lambda expression as a parameter:

interface StringFunction {

String run(String str);

}

public class MyClass {

public static void main(String[] args) {

StringFunction exclaim = (s) -> s + "!";

StringFunction ask = (s) -> s + "?";

**printFormatted**("Hello", exclaim);

**printFormatted**("Hello", ask);

}

public static void **printFormatted**(String str, StringFunction format) {

String result = format.run(s

/////////////////////////////////////////////////////////////////////////////

**Bases de datos**

Insert

try {

Connection miConexion = DriverManager.getConnection(“jdbc:mysql://localhost:3306/pruebas”, “root”, “”);

Statement miStatement = miConexion.createStatement();

String **instruccionSql** = “Insert into Productos (CodigoArticulo, NombreArticulo) values (‘AR77’, “Pantalón”)“;

miStatement.executeUpdate(**instruccionSql**);

System.out.println(“Datos insertados”);

} catch(Exception e) {

System.out.println(“No conecta”);

}

Update

try {

Connection miConexion = DriverManager.getConnection(“jdbc:mysql://localhost:3306/pruebas”, “root”, “”);

Statement miStatement = miConexion.createStatement();

String **instruccionSql** = “Update productos set precio = precio\*2 where CodigoArticulo =’AR77’“;

miStatement.executeUpdate(**instruccionSql**);

System.out.println(“Datos modificados”);

} catch(Exception e) {

System.out.println(“No conecta”);

}

Delete

try {

Connection miConexion = DriverManager.getConnection(“jdbc:mysql://localhost:3306/pruebas”, “root”, “”);

Statement miStatement = miConexion.createStatement();

String instruccionSql = “Delete from Productos where CodigoArticulo = ‘AR77’“;

miStatement.executeUpdate(instruccionSql);

System.out.println(“Datos eliminados”);

} catch(Exception e) {

System.out.println(“No conecta”);

}

////////////////////////////////////////////////////

**Files**

From the java.io package

**File** myObj = new **File**("filename.txt");

canRead() Boolean the file is readable or not

canWrite() Boolean the file is writable or not

createNewFile() Boolean Creates an empty file

delete() Boolean Deletes a file

exists() Boolean the file exists

getName() String Returns the name of the file

getAbsolutePath() String Returns the absolute pathname of the file

length() Long Returns the size of the file in bytes

list() String[] Returns an array of the files in the directory

mkdir() Boolean Creates a directory

////////////////////////////////////////

**Create**

import java.io.File; // Import the File class

import java.io.IOException; // Import the IOException class to handle errors

public class CreateFile {

public static void main(String[] args) {

try {

File myObj = new File("filename.txt");

if (myObj.createNewFile()) {

System.out.println("File created: " + myObj.getName());

} else {

System.out.println("File already exists.");

}

} catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

}

File myObj = new File("C:\\Users\\MyName\\filename.txt");

**Write**

import java.io.FileWriter;

import java.io.IOException; // Import the IOException class to handle errors

public class WriteToFile {

public static void main(String[] args) {

try {

FileWriter myWriter = new FileWriter("filename.txt");

myWriter.write("Files in Java might be tricky, but it is fun enough!");

myWriter.close();

System.out.println("Successfully wrote to the file.");

} catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

}

//////////////////////////

Read

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner; // Import the Scanner class to read text files

public class ReadFile {

public static void main(String[] args) {

try {

File myObj = new File("filename.txt");

Scanner **myReader** = new Scanner(myObj);

while (**myReader**.hasNextLine()) {

String data = **myReader**.nextLine();

System.out.println(data);

}

**myReader**.close();

} catch (FileNotFoundException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

}

**File Information**

import java.io.File;

public class GetFileInfo {

public static void main(String[] args) {

File myObj = new File("filename.txt");

if (myObj.exists()) {

System.out.println("File name: " + myObj.getName());

System.out.println("Absolute path: " + myObj.getAbsolutePath());

System.out.println("Writeable: " + myObj.canWrite());

System.out.println("Readable " + myObj.canRead());

System.out.println("File size in bytes " + myObj.length());

} else {

System.out.println("The file does not exist.");

}

}

}

//////////////////////

**Delete a file**

import java.io.File; // Import the File class

public class DeleteFile {

public static void main(String[] args) {

File myObj = new File("filename.txt");

if (myObj.delete()) {

System.out.println("Deleted the file: " + myObj.getName());

} else {

System.out.println("Failed to delete the file.");

}

}

}

**Delete a Folder**

import java.io.File;

public class DeleteFolder {

public static void main(String[] args) {

File myObj = new File("C:\\Users\\MyName\\Test");

if (myObj.delete()) {

System.out.println("Deleted the folder: " + myObj.getName());

} else {

System.out.println("Failed to delete the folder.");

}

}

}

///////////////////////////

Declarando un array:

int intArray[];

Asignando memoria al array:

intArray = new int[20];

Combinando ambas declaraciones en una:

int[] intArray = new int[20];

int[] intArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 };

for (int i = 0; i < arr.length; i++)

Student es una clase definida por el usuario:

Student[] arr = new Student[7];

El array Student contiene siete espacios de memoria, cada uno del tamaño de la clase Student

arr[0] = new Student(1,"uno");

System.out.println("Elemento en " + i + " : " +

arr[i].roll\_no +" "+ arr[i].name);

2 dimensiones:

int[][] intArray = new int[10][20];

3 dimensiones:

int[][][] intArray = new int[10][20][10];

Parámetro:

public static void sum(int[] arr)

return new int[]{1,2,3};

El nuevo array que contiene copias de los elementos del array original en lugar de referencias:

int intArray[] = {1,2,3};

int cloneArray[] = intArray.clone();

//////////////////////////////////

String str = new String("Hola");

"Javadesdecero.es".length();

"Javadesdecero.es".charAt(3);

El último no está incluido:

.substring(4,9);

s1.concat(s2);

s.indexOf("Cero");

Devuelve el índice dentro de la cadena de la primera aparición de la cadena especificada, comenzando en el índice especificado:

s.indexOf('a',3);

s.lastIndexOf('a');

Compara este String con el objeto especificado:

"Java".equals("Java");

"Java".equalsIgnoreCase("Java");

int salida = s1.compareTo(s2); // donde s1 y s2 son

// strings que se comparan

Esto devuelve la diferencia s1-s2. Si :

salida < 0 // s1 es menor que s2

salida = 0 // s1 y s2 son iguales

salida > 0 // s1 es mayor que s2

s1.compareToIgnoreCase(s2);

.toLowerCase();

Eliminando espacios en blanco en ambos extremos:

.trim();

Devuelve una nueva cadena al reemplazar todas las ocurrencias:

palabra1.replace('y' ,'j');

String str[]={"Java", "desde","Cero"};

str[1]="Curso";

for (String s : str)

El contenido de un objeto String es inmutable

Cuando necesite una cadena que sea una variación de una que ya existe, simplemente cree una nueva cadena que contenga los cambios deseados

Las variables de referencia de cadena pueden, por supuesto, cambiar el objeto al que hacen referencia

Un argumento de línea de comandos es la información que sigue directamente el nombre del programa en la línea de comando cuando se ejecuta

////////////////////////////

Modificador/Acceso Clase Paquete Subclase Todos

public Sí Sí Sí Sí

protected Sí Sí Sí No

default Sí Sí No No

private Sí No No No

Private

Cualquier otra clase del mismo paquete no podrá acceder a estos miembros.

Las clases e interfaces no se pueden declarar como privadas (private)

Protected

son accesibles dentro del mismo paquete o sub-clases en paquetes diferentes.

7 modificadores que no son de acceso o, a veces, también llamados especificadores, proporcionan información sobre su comportamiento a la JVM:

static

final

abstract

synchronized

transient

volatile

native

/////////////////////////

boolean equalTo(Demo obj) {

Caja(Caja ob)

{

ancho = ob.ancho;

alto = ob.alto;

largo = ob.largo;

}

Caja miCaja = new Caja(10, 20, 15);

Caja miclon = new Caja(miCaja);

///////////////////////////

////////////////////////////

////////////////////

///////////////////

///////////////////