D. Defina un ejemplo de código (5 líneas mínimo,15 líneas máximo) en cada lenguaje seleccionado donde se visualicen los distintos tipos de variables que soporta el lenguaje en cuanto al tiempo de vida, y realice una tabla donde se sitúen todos los identificadores junto con los valores de sus atributos tal como se vió en la práctica correspondiente.

Java

1 public class Variables {

2 private static int varClase; // Variable de clase

3 private int varInstancia ; // Variable de instancia

4 public void unMetodo(int varParametro) {

5 Object varLocal; // Variable local

6 varClase = 1;

7 varInstancia = 2;

8 varLocal = new Object;

9 System.out.println("Valor de la variable de clase: " + varClase);

10 System.out.println("Valor de la variable de instancia: " + varInstancia);

11 System.out.println("Valor de la variable local: " + varLocal );

12 }

13 }

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Tipo | r-valor | Alcance | Tiempo de vida |
| varClase | Automática | 0 - 1 | 3 - 13 | <1-13> |
| varInstancia | Automática | 0 - 2 | 4 - 13 | 1-13 |
| varLocal | Automática | Null – (dir. De memoria) | 9-12 | 4-12 |
| varParametro | Automática | ¿? | 5-12 | 4-12 |

Hay 4 tipos de variables en cuanto al tiempo de vida:

* variables de instancia
* variables de clase
* variables locales
* variables de parámetro

Variable de instancia: Es una variable que pertenece a un objeto. Se ha declara dentro de una clase, pero fuera de cualquier método o bloque. Su alcance es en toda la clase excepto en los métodos estáticos. Su tiempo de vida es mientras que el objeto de la clase se encuentre en memoria. La variable “varInstancia” es de este tipo, se declara en la línea 3. Al momento de instanciarse un objeto de tipo “Variables”, se reservará un espacio de memoria para esta variable. Este espacio estará reservado hasta que el “garbage collector” elimine de memoria al objeto que referencia a la variable.

Variable de clase: Es una variable declarada como estática dentro de una clase y fuera de cualquier bloque o método. Su alcance es en toda la clase. Su tiempo de vida es desde el comienzo hasta la finalización del programa o hasta que la clase no se encuentre más en memoria. (es cargada en memoria por el classLoader. La variable “varClase” es de este tipo, se declara en la línea 2. Durante el runtime, el classLoader carga la clase en memoria junto con sus variables estáticas. Estas estarán en memoria hasta la finalización del programa.

Variable local: Es una variable declarada dentro de un método. Su alcance es desde el punto de la declaración hasta el fin del cuerpo del método. Su tiempo de vida es desde el comienzo hasta la finalización de la ejecución del método.

Variable de parámetro: es una variable que se ha pasado a un método cuándo este es llamado. Su alcance es el cuerpo del método. Su tiempo de vida es desde el comienzo hasta la finalización de la ejecución del método.

**Javascript**

1 var variableGlobal; // Variables global (toda la aplicación).

2 console.log("variableGlobal linea2: "+ variableGlobal);

3 variableGlobal = "global";

4 function unaFuncion() {

5 console.log("variableLocal linea5: "+variableLocal);

6 if (true) {

7 let variableBloque = "bloque"; // Variable de bloque

8 variableGlobal2 = "global2" //variable global declarado dentro de una funcion

9 }

10 var variableLocal = "local"; // Variable local (dentro de la función)

11 console.log("variableLocal linea11: "+variableLocal);

12 }

13 unaFuncion();

14 console.log("variableGlobal1 linea 14: "+variableGlobal);

15 console.log("variableGlobal2 linea 15: "+variableGlobal2);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Tipo | r-valor | Alcance | Tiempo de vida |
| varGlobal | Automática | Undefined- “global” | 1-15 | 1-15 |
| varGlobal2 | Automática | Undefined-“global2” | 1-15 | 1-15 |
| varLocal | Automática | “local” | 5-12 | 5-12 |
| varBloque | Automática | “bloque” | 8-9 | 7-9 |

Tiempo de vida: El tiempo de vida de una variable comienza cuando es declarada. Si es una variable local, su tiempo de vida finaliza cuando la función finaliza. En caso de ser una variable global, su tiempo de vida comienza al ejecutarse el programa y finaliza cuando la ventana o tab del navegador es cerrada.

Javascript tiene 3 tipos de variables en cuanto a su alcance:

Variable de bloque: Una variable de bloque solo será visible dentro del bloque donde se definió. Un bloque está rodeado de llaves “{}”. Para que sea de bloque debe declararse con let o const dentro de un blque.

Local variable: A local variable will be visible only within a function where it is defined. Function parameters are always local to that function. The completion of a function deletes the local variable.

Global variable: A global variable has a global scope which means it can be defined anywhere in your JavaScript code. Global variables delete when the web browser is closed. However if a new page is loaded in the same browser window, then it remains.

JAVASCRIPT ENGLISH

The lifetime of a variable starts when it is declared. If the variable is in local scope of a function, its lifetime ends when the function ends. If the variable is in global scope, its lifetime ends when the browser tab or window is closed.

Local variable: A local variable will be visible only within a function where it is defined. Function parameters are always local to that function. The completion of a function deletes the local variable.

Global variable: A global variable has a global scope which means it can be defined anywhere in your JavaScript code. Global variables delete when the web browser is closed. However if a new page is loaded in the same browser window, then it remains.

Java

public class testMain {

static double varGlobal1; //se inicializa automáticamente en 0.0

static StringBuilder varGlobal2; //se inicializa automáticamente en null

static int variable = 1; //se inicializa automáticamente en 0

public static void main(String[] args) {

varGlobal2 = new StringBuilder("asd"); //la variable ahora contiene la direccion de memoria de la heap

double varLocal1; //no se inicializa con ningún valor

int j = 3;

double variable = 4.0;

varLocal1 = 2;

varGlobal1 = 1.0;

System.out.println("variables: "+ varGlobal1 +" "+ varGlobal2 +" "+variable +" "+j);

}

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | L-valor | r-valor | Alcance | Tiempo de vida |
| varGlobal1 | Automática | 0.0 | 3 – 14 | 1-14 |
| varGlobal2 | Automática | null | 4-14 | 1-14 |
| varGlobal2^ | Dinámica | basura | 4-14 | 6-14 |
| variable (línea 4) | Automática | 1 | 5-8 | 1-14 |
| VarLocal1 | Automática | indef | 10-13 | 5-13 |
| Main() |  |  | 6-13 | 1-14 |
| variable (línea 9) | Automatica | 4.0 | 10-13 | 5-13 |

Explicar un poco tiempo de vida:

Javascript

1 var variableGlobal; // Variables global (toda la aplicación).

2 console.log("variableGlobal linea2: "+ variableGlobal);

3 variableGlobal = "global";

4 function unaFuncion() {

5 console.log("variableLocal linea5: "+variableLocal);

6 if (true) {

7 let variableBloque = "bloque"; // Variable de bloque

8 variableGlobal2 = "global2" //variable global declarado dentro de una funcion

9 }

10 var variableLocal = "local"; // Variable local (dentro de la función)

11 console.log("variableLocal linea11: "+variableLocal);

12 }

13 unaFuncion();

14 console.log("variableGlobal1 linea 14: "+variableGlobal);

15 console.log("variableGlobal2 linea 15: "+variableGlobal2);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Tipo | r-valor | Alcance | Tiempo de vida |
| varGlobal | Automática | Undefined- “global” | 1-15 | 1-15 |
| varGlobal2 | Automática | Undefined-“global2” | 1-15 | 1-15 |
| varLocal | Automática | “local” | 5-12 | 5-12 |
| varBloque | Automática | “bloque” | 8-9 | 7-9 |

|  |  |
| --- | --- |
| Java | Javascript |
| 1 public class testMain {  2 static double varGlobal1;  3 static StringBuilder varGlobal2;  4 static int variable = 1;  5 **public static void main**(String[] args) {  6 varGlobal2 = new StringBuilder("asd");  7 double varLocal1;  8 int j = 3;  9 double variable = 4.0;  10 varLocal1 = 2;  11 varGlobal1 = 1.0;  12 System.out.println("variables: "+ varGlobal1 +" "+ varGlobal2 +" "+variable +" "+j);  13 }  14} | var variableGlobal;  console.log("variableGlobal linea2: "+ variableGlobal);  3 variableGlobal = 1;  4 **function unaFuncion()** {  5 console.log("variableLocal linea5: "+variableLocal);  6 if (true) {  7 let variableBloque = new Date(); // Variable de bloque  8 variableGlobal2 =true; //variable global declarado dentro de una funcion  9 }  10 var variableLocal = new Date(); // Variable local (dentro de la función)  11 console.log("variableLocal linea11: "+variableLocal);  12 }  13 unaFuncion();  14 console.log("variableGlobal1 linea 14: "+variableGlobal);  15 console.log("variableGlobal2 linea 15: "+variableGlobal2); |

Javascript

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Tipo | r-valor | Alcance | Tiempo de vida |
| varGlobal | Automática | Undefined | 1-15 | 1-15 |
| varGlobal2 | Automática | Undefined | 1-15 | 1-15 |
|  |  |  |  |  |
| varLocal^ | Dinámica | [object] | 4-12 | 10-12 |
|  |  |  |  |  |
| varBloque^ | Dinámica | [object] | 8-9 | 6-9 |

Java

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | L-valor | r-valor | Alcance | Tiempo de vida |
| varGlobal1 | Automática | 0.0 | 3 – 14 | 1-14 |
|  |  |  |  |  |
| varGlobal2^ | Dinámica | indefinido | 4-14 | 6-14 |
| variable (línea 4) | Automática | 0.0 | 5-8 | 1-14 |
| VarLocal1 | Automática | indefindo | 10-13 | 5-13 |
| Main() |  |  | 6-13 | 5-13 ¿? |
| variable (línea 9) | Automatica | indefinido | 10-13 | 5-13 |

Javascript

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Identificador | Tipo | r-valor | Alcance | Tiempo de vida |
| varGlobal | Automática | Undefined | 1-15 | 1-15 |
| varGlobal2 | Automática | Undefined | 1-15 | 1-15 |
|  |  |  |  |  |
| varLocal^ | Dinámica | ¿?? [object] | 5-12 | 10-12 |
|  |  |  |  |  |
| varBloque^ | Dinámica | ¿?? [object] | 8-9 | 7-9 |

Ambos lenguajes realizan su liberación de memoria mediante un recolector de basura de forma automática. El recolector de basura es un programa que corre por detrás del programa principal que se encarga de identificar los objetos que no se encuentran referenciados y se eliminan de memoria.

[https://felixgerschau.com/javascript-memory-management/#:~:text=JavaScript%20engines%20have%20two%20places,The%20memory%20heap%20and%20stack](https://felixgerschau.com/javascript-memory-management/" \l ":~:text=JavaScript engines have two places,The memory heap and stack).

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Memory_management>

<https://www.geeksforgeeks.org/java-memory-management/>

<https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html>

<https://developer.mozilla.org/en-US/docs/Glossary/Hoisting>