ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

ECYS | DIDT
DESARROLLO DE TRANSFERENCIA TECNOLÓGICA

Día, Fecha:	Jueves, 03/08/2023
Hora de inicio:	07:10

Introducción a la programación y computación 1 [F]

José Eduardo Morales García

Algoritmo

Un algoritmo es un conjunto finito de instrucciones o pasos bien definidos y no ambiguos que se utilizan para resolver un problema o llevar a cabo una tarea en un número finito de pasos.

Características

- Precisión
- Finitud
- Entrada
- Salida
- Determinismo
- Eficiencia

Ejemplo de algoritmo

```
ALGORITMO SumarDosNumeros
  ENTRADA: num1, num2 (dos números enteros)
  SALIDA: resultado (la suma de los dos números)

  PASO 1: Leer el valor de num1 desde el usuario
  PASO 2: Leer el valor de num2 desde el usuario
  PASO 3: Sumar num1 y num2 y almacenar el resultado en la variable resultad
  PASO 4: Imprimir el valor de resultado como salida
FIN DEL ALGORITMO
```

Casting (casteo)

En el contexto de la programación en Java, "casting" (o "casteo") se refiere a la conversión de un objeto de un tipo a otro. Esto puede ser necesario cuando se quiere tratar un objeto de una clase específica como si fuera de otra clase relacionada o cuando se necesita cambiar el tipo de datos de una variable.

Hay dos tipos principales de cast en Java:

- Cast implícito (upcasting)
- Cast explícito (downcasting)

Cast implícito (upcasting)

- Ocurre cuando un objeto se convierte a un tipo más general o una superclase. En este caso, no es necesario usar una sintaxis especial; la conversión se realiza automáticamente.

```java
public class ExplicitCastingExample {
    public static void main(String[] args) {
        // Casteo explícito de valores primitivos
        double numeroDoble = 3.14159;
        int numeroEntero = (int) numeroDoble; // Casteo explícito de double

        // Imprimir los valores casteados
        System.out.println("Número doble: " + numeroDoble);
        System.out.println("Número entero: " + numeroEntero);
    }
}
```

Cast explícito (downcasting)

- Ocurre cuando un objeto se convierte a un tipo más específico o una subclase. En este caso, se requiere una sintaxis especial para realizar la conversión y además, es necesario tener cuidado, ya que, si el objeto no es compatible con el tipo al que se intenta convertir, se generará una excepción en tiempo de ejecución (ClassCastException).

```java
public class ExplicitCastingExample {
    public static void main(String[] args) {
        // Casteo explícito de valores primitivos
        int numeroEntero = 65;
        char caracter = (char) numeroEntero; // Casteo explícito de int a ch

        // Imprimir los valores casteados
        System.out.println("Número entero: " + numeroEntero);
        System.out.println("Carácter: " + caracter);
    }
}
```

Tipos de Operadores

- Unarios
- Aritméticos
- Relacionales
- Lógicos
- Ternario

Descripción	Operadores
operadores posfijos	op++ op--
operadores unarios	++op --op +op -op ~ !
multiplicación y división	* / %
suma y resta	+ -
desplazamiento	<< >> >>>
operadores relacionales	< > <= =>
equivalencia	== !=
operador AND	&
operador XOR	^
operador OR	|
AND booleano	&&
OR booleano	||
condicional	?:
operadores de asignación	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Palabras reservadas

- Una palabra con un significado y valor especial para el lenguaje de programación en el cual se trabaja.

abstract	continue	finally	int	public	throw
assert	default	float	interface	return	throws
boolean	do	for	long	short	transient
break	double	goto	native	static	true
byte	else	if	new	strictfp	try
case	enum	implements	null	super	void
catch	extends	import	package	switch	volatile
class	false	inner	private	synchronized	
const	final	instanceof	protected	this	while

Input y Output

En Java, el manejo de la entrada (input) y salida (output) se realiza principalmente mediante las clases del paquete java.io o java.util. Esta librería proporciona funcionalidades para leer y escribir datos desde y hacia diferentes fuentes, como el teclado, archivos, la consola, entre otros.

Por ejemplo:

- Scanner
- Println o Print
- FileWriter

Input y Output

```java
import java.util.Scanner;

public class InputExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Ingresa tu nombre: ");
        String nombre = scanner.nextLine();

        System.out.print("Ingresa tu edad: ");
        int edad = scanner.nextInt();

        System.out.println("¡Hola, " + nombre + "! Tienes " + edad + " años.
    }
}
```

Input y Output

```java
public class OutputExample {
    public static void main(String[] args) {
        String mensaje = "¡Hola, mundo!";
        int numero = 42;

        System.out.println(mensaje);
        System.out.println("El número es: " + numero);
    }
}
```

Estructuras de control

```
Proceso Adivina_Numero

    intentos<-10
    num_secreto <- azar(100)+1

    Escribir "Adivine el numero (de 1 a 100):"
    Leer num_ingresado
    Mientras num_secreto<>num_ingresado Y intentos>1 Hacer
        Si num_secreto>num_ingresado Entonces
            Escribir "Muy bajo"
        Sino
            Escribir "Muy alto"
        FinSi
        intentos <- intentos-1
        Escribir "Le quedan ",intentos," intentos:"
        Leer num_ingresado
    FinMientras

    Si num_secreto=num_ingresado Entonces
        Escribir "Exacto! Usted adivino en ",11-intentos," intentos."
    Sino
        Escribir "El numero era: ",num_secreto
    FinSi

FinProceso
```

Estructuras de control

Las **estructuras de control** son aquellas que determinan el comportamiento de un programa.

- **Selectivas,** permiten que la ejecución de un programa o conjunto de instrucciones se ejecuten conforme una condición o criterio.
- **Iterativas,** permiten la ejecución de un bloque de código o conjunto de instrucciones de forma repetitiva.

Estructuras de control

Las **estructuras de control selectivas:**

1. **If, if-else**
2. **switch**

Estructuras de control

Estructura de Selección IF,

Permiten la ejecución de un bloque si cumple con dos únicas alternativas, **verdadero** o **falso.**

Estructuras de control

```java
if(carrera == "Ingenieria en Ciencias y Sistemas"){
  //...
  System.out.println("La mejor carrera del mundo");
}
// -- esta es opcional, se utiliza conforme a lo que requerimos conforme a nuestro flujo
else {
  //...
  System.out.println("La mejor carrera del mundo es Ingenieria en Ciencias y Sistemas");
}
```

Estructuras de control

Estructura de Selección SWITCH,

Esta sentencia se utiliza para elegir una entre múltiples opciones, únicamente evalúa valores puntuales.

Estructuras de control

```
switch(opcion)
  {
      case "1" -> {
          //caso si el valor ingresado es "1"
      }
      case "2" -> {
          //caso si el valor ingresado es "2"
      }
      case "3" -> {
         //caso si el valor ingresado es "3"
      }
      default  -> {
        //caso si el valor ingresado no forma parte en ninguna de las anteriores
      }
  }
```

Estructuras de control

Estructura Iterativa WHILE,

Esta sentencia se utiliza como un ciclo de ejecución, en el cual al cumplirse cierta condición repetirá los bloques hasta que este deje de cumplir con la misma.

Estructuras de control

```
while(continues)
{
  /*
        Esta operara siempre el bloque repetitivamente
        hasta que continues represente un falso logico

  */
  System.out.println("Repetir");
}
```

Estructuras de control

Estructura Iterativa FOR,

Esta sentencia se utiliza como un ciclo de ejecución, en el cual se ejecuta un número determinado de veces dentro de su ciclo de ejecución.

Estructuras de control

```java
for(int a=0; a<15; a++)
{
     /*
        Esta operara siempre el bloque repetitivamente
        hasta que a sea un numero igual a 15

    */
    System.out.println("imprimiendo el valor " + a + " veces");
}
```

Estructuras de control

Estructura Iterativa DO…WHILE,

Esta sentencia se utiliza como un ciclo de ejecución, en el cual se ejecuta para desarrollar un conjunto de instrucciones al menos una vez o varias veces.

Estructuras de control

```
Do
{
  /*
        Esta operara siempre el bloque 1 vez, luego
        lo realizara repetitivamente hasta que continues
        represente un falso logico

  */
  System.out.println("Repetir")
}while(continues)
```

¿Qué son los arreglos en Java?

- Los arreglos en Java son estructuras de datos que permiten almacenar y acceder a varios valores de un mismo tipo de forma ordenada. Los arreglos se pueden crear para almacenar cualquier tipo de dato, incluyendo datos primitivos y no primitivos.

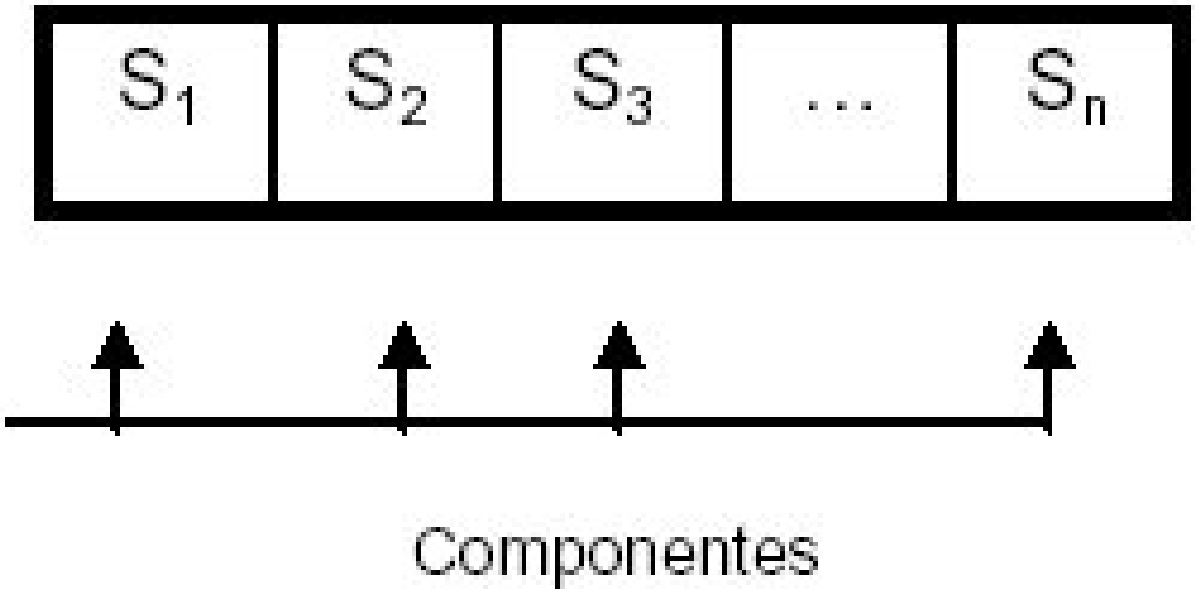
Para crear un arreglo en Java, se utiliza la sintaxis:

```
tipoDeDato[] nombreArreglo = new tipoDeDato[tamaño];
```

Los elementos del arreglo se pueden acceder utilizando el nombre del arreglo seguido de un índice entre corchetes, por ejemplo:

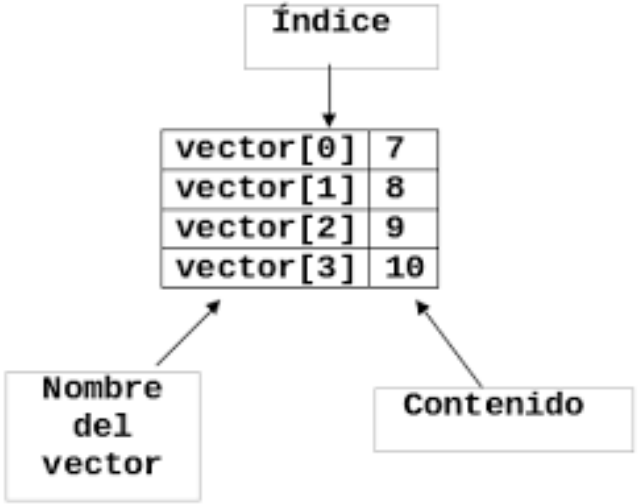
```
nombreArreglo[índice];
```

Vectores



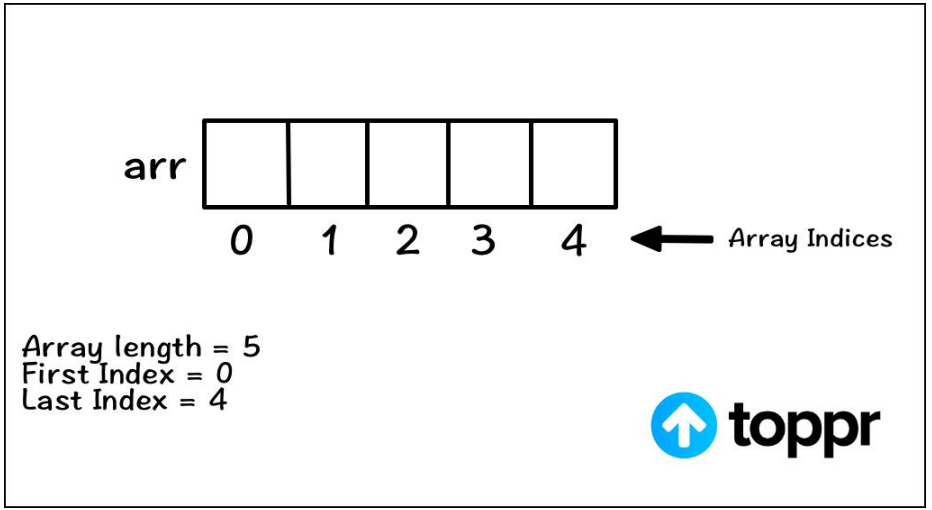
Vectores

Es un tipo de estructura unidimensional d
contiene una serie de datos contiguos d

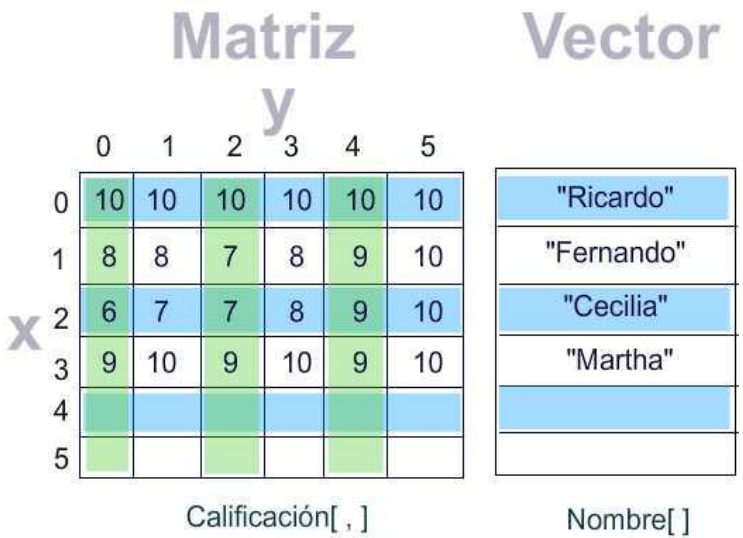


Vectores

Un vector está definido por el tipo de dato que se desea representar, pueden ser definidos por tipos primitivos y no primitivos.

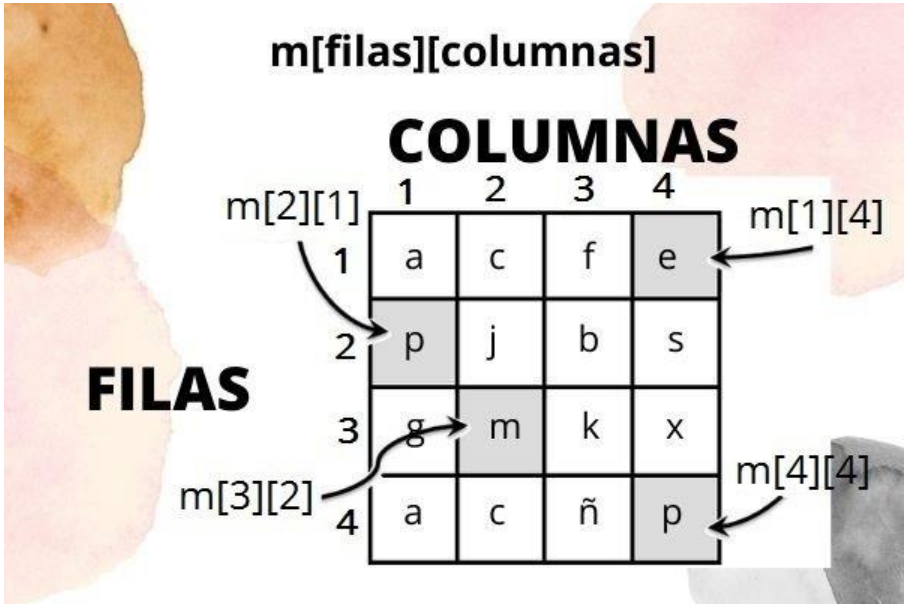


Matrices


Matriz
y
Vector
0 1 2 3 4 5
x
0 10 10 10 10 10 10
1 8 8 7 8 9 10
2 6 7 7 8 9 10
3 9 10 9 10 9 10
4
5
"Ricardo"
"Fernando"
"Cecilia"
"Martha"
Calificación[,]
Nombre[ ]

Matrices

Una matriz por su parte es un conjunto o colección de arreglos, es decir es un vector de vectores.



Matrices

Son regularmente empleados, para almacenar conjuntos de valores, de un mismo tipo, en el que podemos ordenar y agrupar conforme a lo requiramos.

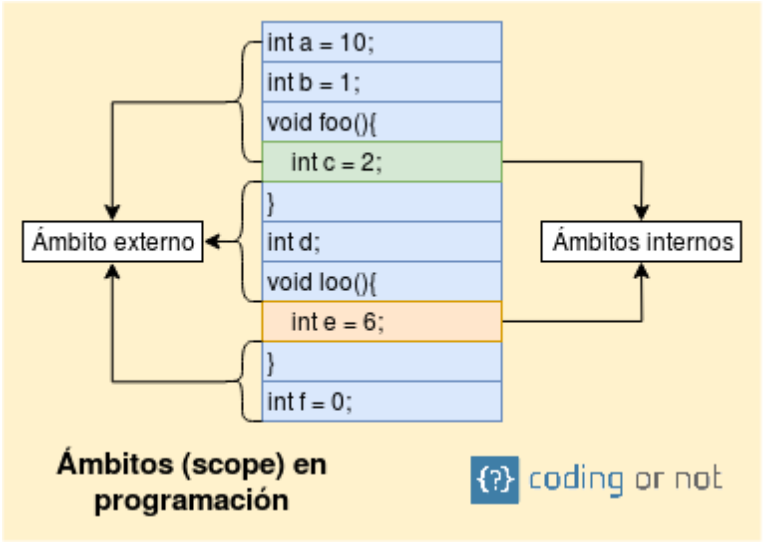
Calendario[semana, nombredia] nxm

Semana \ nombredia	D	L	m	M	J	V	S
1	0	0	0	0	1	2	3
2	4	5	6	7	8	9	10
3	11	12	13	14	15	16	17
4	18	19	20	21	22	23	24
5	25	26	27	28	0	0	0

Ámbito

Ámbito, es el contexto otorgado conforme a los permisos dentro de un programa. (“donde es accesible un valor dentro del programa”)

- **Global**
- **Local**



Fundamentos de programación

Ámbito local, únicamente es accesible dentro del bloque al que pertenece.

Ámbito Global, accesible desde cualquier lugar

```
main() {
    int a = 1;                          B1
    int b = 1;
    {
        int b = 2;                      B2
        {
            int a = 3;                  B3
            cout << a << b;
        }
        {
            int b = 4;                  B4
            cout << a << b;
        }
        cout << a << b;
    }
    cout << a << b;
}
```

Dudas y Preguntas

Parte practica