El futuro digital es de todos

MinTIC

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Misión TIC 2022

SEMANA 1

INICIAMOS 8:05PM

UNIVERSIDAD DE ANTIOQUIA
1803
Facultad de Ingeniería

Luisa Fernanda Restrepo.

Agenda

- Variables
- Input
- String
- Estructuras de selección
- Estructuras de iteración
- Arreglos

Comentarios

- Java permite comentar líneas utilizando doble slash //
- Y permite comentar múltiples líneas mediante /* */

Ejemplo:

```
public class Principal {
    public static void main(String[] args) {
        //comentario de una linea

        /*
         comentario
         de
         muchas
         lineas
         */
    }
}
```

Variables

Luisa Fernanda Restrepo.

¿Qué es una variable?

- Las **variables** son espacios reservados en la **memoria** que, como su nombre indica, pueden cambiar de contenido a lo largo de la ejecución de un programa.
- Las variables guardan **valores** (los datos de la aplicación).
- En java se declaran variables de la siguiente forma:

```
tipoVariable nombreVariable;
```

¿Cómo de declaraba una variable en Python?

¿Qué tipos de variables recuerdas?

Tipos de variables primitivas

Tipo	Tamaño	Rango	Literal (ejemplo)
byte	8 bit	[-128 , 127]	100
short	16 bit	[-32.768 , 32.767]	10000
int	32 bit	[-231 , -231-1]	100000
long	64 bit	[-263 , 263-1]	100000l ó 100000L
float	32 bit		4.25f ó 4.25F
double	64 bit		42.5 ó 42.5d ó 42.5D ó 4.25e1
boolean	1 bit	true / false	false
char	16 bit	['\u0000' , '\uffff']	'a'

Ejemplos de declaración de variables

Ejercicio [tiposVariables]

- Almacene en variables los siguientes datos que aparecen en su documento de identidad en el método “main” de una nueva clase llamada Identidad. Y luego imprima los valores.
  - Nombres
  - Apellidos
  - Número de documento de identidad
  - Edad
  - Género

Asignación de valores

- Una vez se define una variable, esta solo puede contener valores del tipo del que se creo la variable.

Diga cuales de las siguiente asignaciones no son validas:

```
int num = 8;
int texto = 10;
String mensaje = "Hola";
String hola = 5;
int num2 = 3.0;
double y = 4.5;
boolean x = false;
boolean = true;
```

Operadores Numéricos

Nombre	Operador	Ejemplo	Resultado
Suma	+	5 + 2	7
Resta	-	5 - 2	3
Multiplicación	*	5 * 2	10
División	/	5 / 2	2.5
Módulo	%	5 % 2	1

Ejercicio asignación

```java
public class Principal {
    public static void main(String[] args) {
        short a, b;
        a = 100;
        b = 200;
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("a+b = " + (a+b));
    }
}
```

¿Qué imprime?

Opciones

```
a = 100
b = 200
a+b = 300
```

Operadores de asignación aumentada

Nombre	Operador	Ejemplo	Resultado
Asignación de suma	+=	i += 8	i = i + 8
Asignación de resta	-=	i -= 8	i = i - 8
Asignación de multiplicación	*=	i *= 8	i = i * 8
Asignación de división	/=	i /= 8	i = i / 8
Asignación de módulo	%=	i %=8	i = i % 8

Ejercicio [ExpresionAritmética]

- Dada la expresión algebraica, escríbala en lenguaje de programación:

```
a = 5, b = 3;   c = 8,   d = 4, e = 2,
```

$$a - \frac{b}{c} + \frac{b - \frac{c}{d}}{e}$$

Operadores relacionales

radio = 3

Nombre	Operador	Ejemplo	Resultado
Menor que	<	radio < 0	false
Menor que o igual a	<=	radio <= 0	false
Mayor que	>	radio > 0	true
Mayor que o igual a	>=	radio >= 0	true
Igual a	==	radio == 0	false
Diferente a	!=	radio != 0	true

radio = 0 ¿Cuáles serían los resultados?

Operadores lógicos

age = 24; weight = 140

Nombre	Operador	Ejemplo	Resultado
and	&&	(age > 28) && (weight <= 140)	false
or	||	(age > 34) || (weight >= 150)	false
not	!	!(age > 18)	true
Exclusive or	^	(age > 34) ^ (weight >= 140)	true

Ejercicio [operadoresRelaciones]

- Traduzca el código de Python a Java:

```
a = 16;
b = True;
c = 4;
d = 8;
e = 5
print(a > c and e <=d)
print(b or (d - e) > a / c)
print(not b)
print(not b and c < d or a/d <= e)
```

Input

Luisa Fernanda Restrepo.

Input

- Para capturar información por pantalla (del teclado del usuario) utilizamos la clase de Java llamada “Scanner”.

```
import java.util.Scanner;
public class InputPrueba
{
    public static void main(String[] args) {
        Scanner entrada= new Scanner(System.in);

        System.out.println("Ingrese un dato: ");
        String dato = entrada.next();
        System.out.println("dato: ");
    }
}
```

¿Qué pasa si en lugar de ingresar un texto cuando me piden el dato, ingreso un número?

Input - II

- En las líneas anteriores estamos definiendo un objeto llamado “entrada” de tipo Scanner (esto lo veremos en las clases siguientes). Que luego podremos utilizar con alguno de los siguientes métodos que corresponden a los tipos de datos descritos previamente.

Tipo	Método
Entero muy corto	`nextByte`
Entero corto	`nextShort`
Entero normal	`nextInt`
Entero largo	`nextLong`
Real simple	`nextFloat`
Real doble	`nextDouble`
Booleano	`nextBoolean`
Cadena de texto (String)	`next`

Ejercicio [rectángulo]

- Cree una nueva clase llamada Rectángulo con su método main. Pídale al usuario por pantalla la base de un rectángulo y su altura. Y luego muéstrele en pantalla el calculo del área.

```java
import java.util.Scanner;

public class Rectangulo {
    public static void main(String[] args) {
        Scanner entrada= new Scanner(System.in);
        System.out.println("Ingrese la base:");
        double base = entrada.nextDouble();
        System.out.println("Ingrese la altura:");
        double altura = entrada.nextDouble();
        double area = base*altura;
        System.out.println("el area es: " + area);
    }
}
```

Opciones

```
Ingrese la base:
10
Ingrese la altura:
4
el area es: 40.0
```

Errores comunes

- Encuentre los errores en el siguiente código:

```
public class Principal {
    public staic void main(String[] args) {

      double c,r;
      int z;
      Scanner scan = new Scanner(System.in);
      System.oyt.println("Ingrese el radio");
      z = scan.nextDouble();

    }
}
```

staic -> debe ser static

Falta importar la librería Scanner

System.oyt -> debe ser System.out…

No se puede asignar a una variable “int” un valor “double”

String

Luisa Fernanda Restrepo.

Clase String

- En Java existen 2 tipos de datos para manejar textos, el tipo **char**, y el tipo **String**.
- El tipo **char** se utiliza para representar un solo carácter.
- El tipo **String** se utiliza para representar una cadena de caracteres.

Declarando un String

```
public class PrincipalString
{
    public static void main(String args[]) {
        String msj = "Bienvenido al Curso 035";
        System.out.println(msj);
    }
}
```

¿Qué imprime?

Concatenación

- Una de las operaciones mas útiles cuando se trabaja con Strings es la concatenación. Esta operación se realiza mediante el uso del símbolo “+”.
- Dos Strings pueden ser combinados para crear un nuevo String.
- Consideremos el siguiente ejemplo:

```
String firstName, lastName, fullName;
firstName = "Maya";
lastName = "Smith";
fullName = firstName + " " + lastName;
```

Concatenación – II

- En el ejemplo anterior se utilizó el operador “+” para concatenar Strings. Sin embargo, en muchas casos hemos utilizado el operador “+” para sumar números.
- Cuando un operador representa más de una operación, se le llama **operador sobrecargado**.
- El compilador de Java tratará el operador "+" como una suma cuando ambos operandos (izq y der) sean números; de lo contrario, serán tratados como una concatenación de Strings.

¿Qué imprime?

```
num1 + num2 = 23
5 = num1 + num2
num1 + num2 = 5
```

```java
public static void main(String args[]) {
    int num1 = 2;
    int num2 = 3;
    String str1 = new String("num1 + num2 = ");
    String str2 = new String(" = num1 + num2");
    System.out.println(str1 + num1 + num2);
    System.out.println(num1 + num2 + str2);
    System.out.println(str1 + (num1 + num2));
}
```

Métodos

- Hay más de 50 métodos definidos en la clase String que se pueden encontrar en el documento de especificación de la API de Java: https://docs.oracle.com/javase/9/docs/api/java/lang/String.html

- En esta sección discutiremos 6 de esos métodos:
  - Length
  - charAt
  - indexOf
  - substring
  - equals
  - equalsIgnoreCase

Algunos métodos adicionales

TABLE 4.7 Simple Methods for `String` Objects

Method	*Description*
`length()`	Returns the number of characters in this string.
`charAt(index)`	Returns the character at the specified index from this string.
`concat(s1)`	Returns a new string that concatenates this string with string `s1`.
`toUpperCase()`	Returns a new string with all letters in uppercase.
`toLowerCase()`	Returns a new string with all letters in lowercase.
`trim()`	Returns a new string with whitespace characters trimmed on both sides.

Length

- Podemos usar el método *length()* para devolver el número de caracteres en una cadena.

Codifíquelo

¿Qué imprime?

```
public class PrincipalString
{
    public static void main(String args[]) {
        String message = "Welcome to Java";
        System.out.println("The length of " + message + " is "
            + message.length());
    }

}
```

Opciones

```
The length of Welcome to Java is 15
```

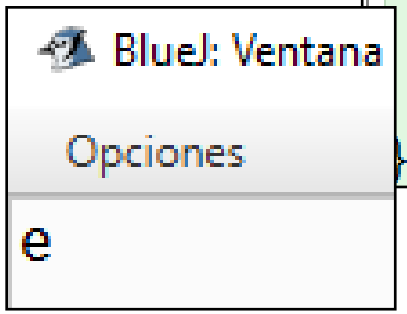
charAt

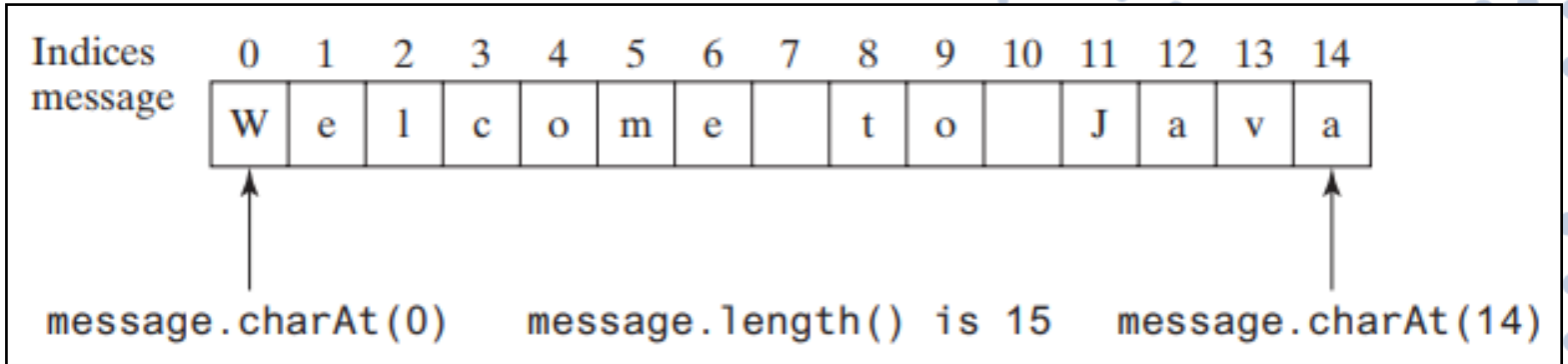
- El método *s.charAt(index)* se utiliza para recuperar un carácter específico en una cadena s. Donde el índice está entre 0 y s.length()-1.

Codifíquelo

```
public class PrincipalString
{
    public static void main(String args[]) {
        String message = "Welcome to Java";
        System.out.println(message.charAt(6));
    }
}
```

¿Qué imprime?





Búsqueda de Strings dentro de Strings

TABLE 4.10 The `String` Class Contains the Methods for Finding Substrings

Method	*Description*
`index Of (ch)`	Returns the index of the first occurrence of `ch` in the string. Returns `-1` if not matched.
`indexOf(ch, fromIndex)`	Returns the index of the first occurrence of `ch` after `fromIndex` in the string. Returns `-1` if not matched.
`indexOf(s)`	Returns the index of the first occurrence of string `s` in this string. Returns `-1` if not matched.
`indexOf(s, fromIndex)`	Returns the index of the first occurrence of string `s` in this string after `fromIndex`. Returns `-1` if not matched.
`lastIndexOf(ch)`	Returns the index of the last occurrence of `ch` in the string. Returns `-1` if not matched.
`lastIndexOf(ch, fromIndex)`	Returns the index of the last occurrence of `ch` before `fromIndex` in this string. Returns `-1` if not matched.
`lastIndexOf(s)`	Returns the index of the last occurrence of string `s`. Returns `-1` if not matched.
`lastIndexOf(s, fromIndex)`	Returns the index of the last occurrence of string `s` before `fromIndex`. Returns `-1` if not matched.

indexOf

- El método *s.indexOf(s2)* retorna el índice de la primera aparición de la cadena *s2* en *s*. Devuelve −1 si no existe *s2* dentro de *s*.

```
"Welcome to Java".indexOf('W') returns 0.
"Welcome to Java".indexOf('o') returns 4.
"Welcome to Java".indexOf('o', 5) returns 9.
"Welcome to Java".indexOf("come") returns 3.
"Welcome to Java".indexOf("Java", 5) returns 11.
"Welcome to Java".indexOf("java", 5) returns -1.

"Welcome to Java".lastIndexOf('W') returns 0.
"Welcome to Java".lastIndexOf('o') returns 9.
"Welcome to Java".lastIndexOf('o', 5) returns 4.
"Welcome to Java".lastIndexOf("come") returns 3.
"Welcome to Java".lastIndexOf("Java", 5) returns -1.
"Welcome to Java".lastIndexOf("Java") returns 11.
```

substring

- Podemos obtener un solo carácter de una cadena utilizando el método *charAt*. Adicionalmente, podemos obtener una subcadena de una cadena utilizando el método de *substring*.

Codifíquelo

```
public class PrincipalString
{
    public static void main(String [] args)  {
        String message = "Welcome to Java";
        String subm = message.substring(0,11);
        String subm2 = message.substring(11);
        System.out.println(subm);
        System.out.println(subm2);
    }
}
```

¿Qué imprime?

Opciones

```
Welcome to
Java
```

Comparación de Strings

TABLE 4.8 Comparison Methods for `String` Objects

Method	*Description*
`equals(s1)`	Returns true if this string is equal to string `s1`.
`equalsIgnoreCase(s1)`	Returns true if this string is equal to string `s1`; it is case insensitive.
`compareTo(s1)`	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than `s1`.
`compareToIgnoreCase(s1)`	Same as `compareTo` except that the comparison is case insensitive.
`startsWith(prefix)`	Returns true if this string starts with the specified prefix.
`endsWith(suffix)`	Returns true if this string ends with the specified suffix.
`contains(s1)`	Returns true if `s1` is a substring in this string.

Equals

- El método *equals* sirve para comparar si el contenido de dos Strings es exactamente el mismo.
- El método *equalsIgnoreCase* sirve para comparar si el contenido de dos Strings es exactamente el mismo (ignorando mayúsculas y minúsculas).

```
public class PrincipalString
{
    public static void main(String args[]) {
        String message = "Welcome to Java";
        String message2 = "welcome to java";
        System.out.println(message.equals(message2));
        System.out.println(message.equalsIgnoreCase(message2));
    }

}
```

¿Qué imprime?

Codifíquelo

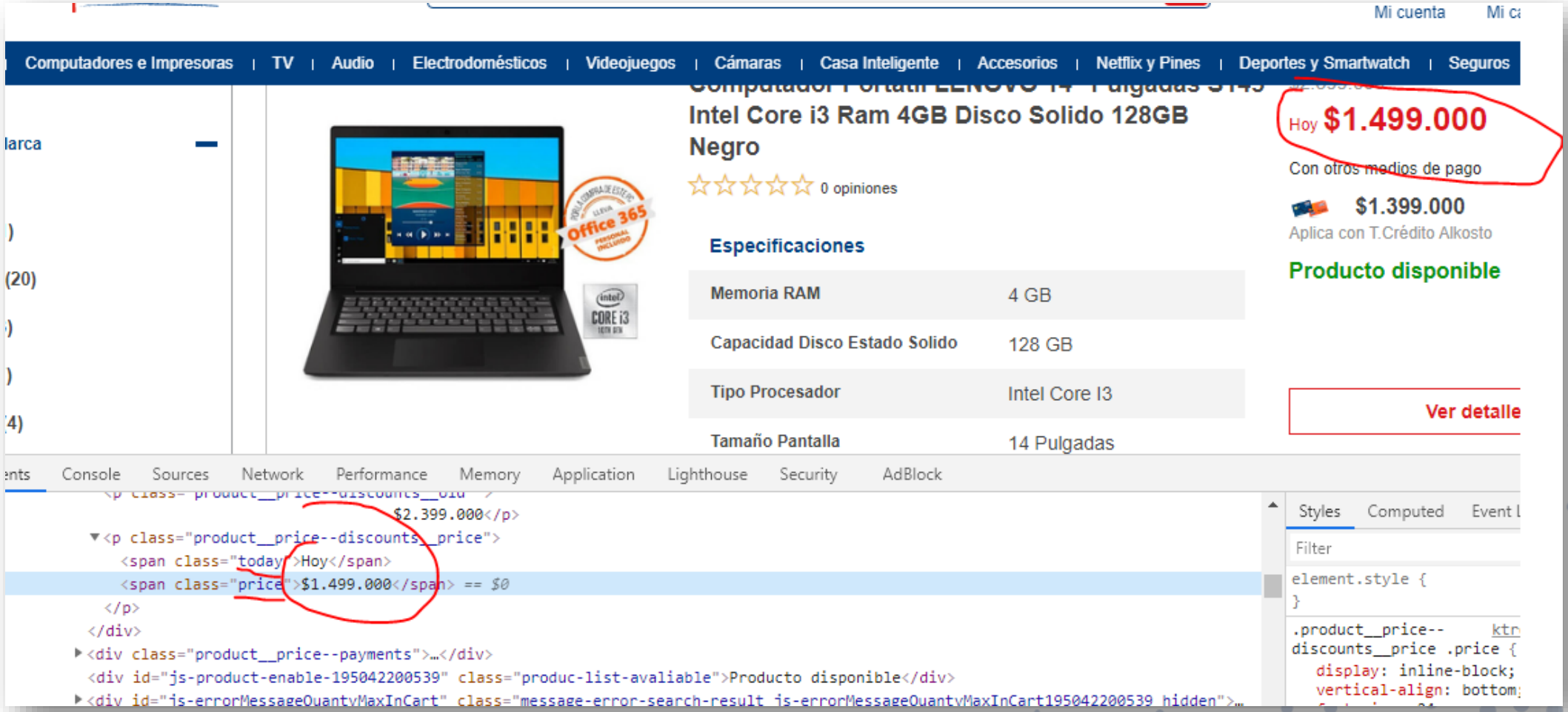
Opciones

```
false
true
```

Nota: para comparar 2 “char”, se debe utilizar “==“, ya que char es un tipo primitivo el cual no tiene disponible el método equals

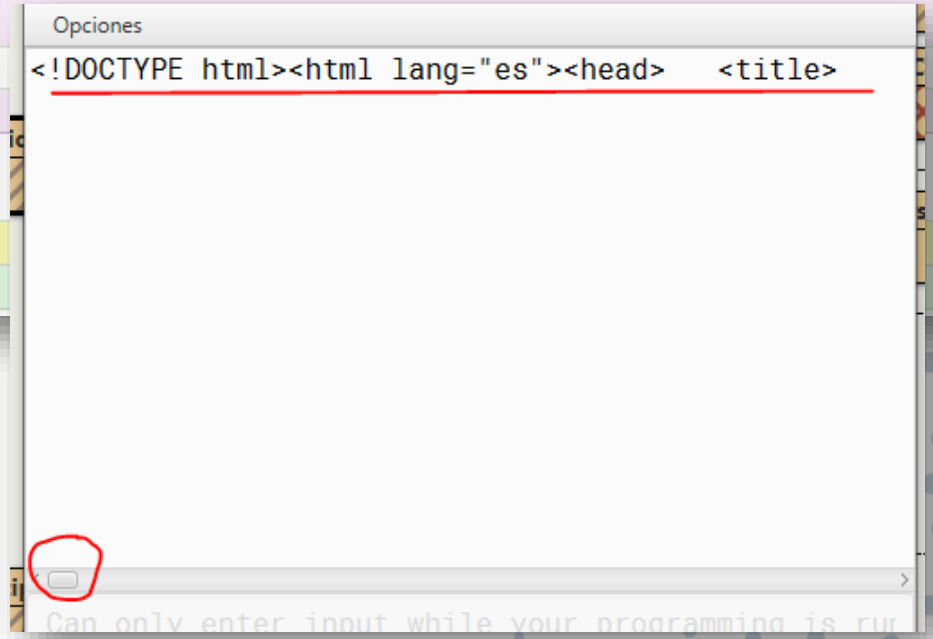
Ejercicio - extraerInfoPW

- A continuación, encontrará un código que le permitirá extraer el código fuente de un sitio web (ese código quedará almacenado en un String).
- El sitio web a analizar es: https://www.ktronix.com/computadores-tablets/computadores-portatiles/c/BI_104_KTRON
- La idea de este ejercicio es desarrollar un programa en Java, que me permita extraer automáticamente la información de los productos que se están vendiendo en la ruta anterior. Extraer el nombre y el precio de cada producto.
- La idea es que usted “navegue” a través de ese String, y extraiga el nombre, marca y precio de los productos que aparecen en esa ruta.
- **Sugerencia:** utilice los elementos vistos en esta clase para solucionar este ejercicio. indexOf, substring, length, etc.



TIP: identifique que expresiones regulares se repiten y dentro de que subcadenas se almacenan los datos que usted desea extraer

```java
import java.net.*;
import java.io.*;
public class EjercicioEstrella
{
    public static void main(String[] args) throws Exception
    {
        String rutaJumbo = "https://www.ktronix.com/computadores-tablets/computadores-portatiles/c/BI_104_KTRON";
        URL url = new URL(rutaJumbo);
        BufferedReader reader = new BufferedReader(new InputStreamReader(url.openStream()));
        String codigoFuente = "";
        String linea;
        while ((linea = reader.readLine()) != null)
        {
            codigoFuente = codigoFuente + linea;
        }
        reader.close();
        System.out.println(codigoFuente);
    }
}
```



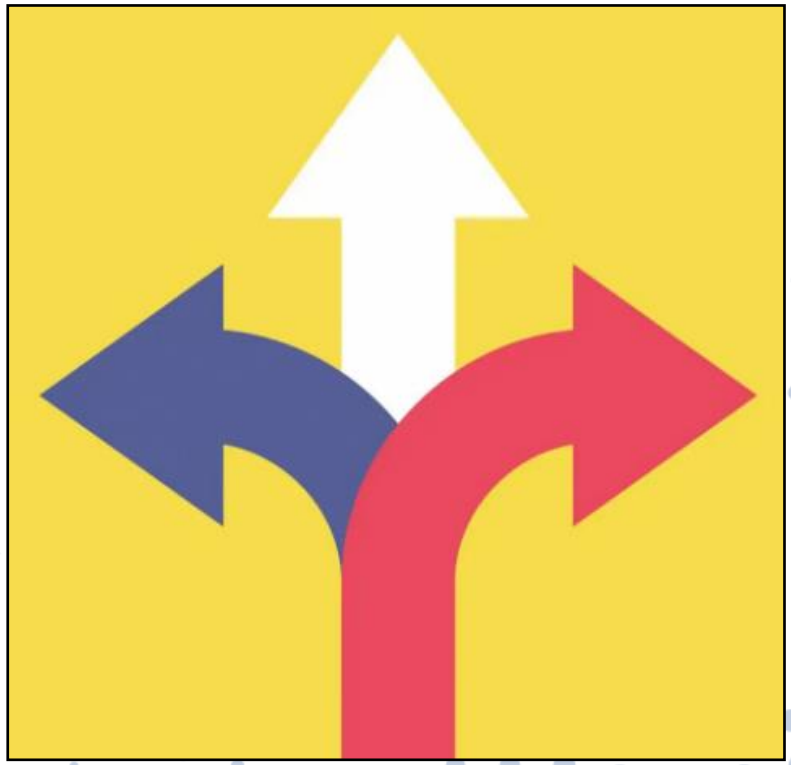
Estructuras de Selección

Luisa Fernanda Restrepo.

Estructuras de selección

¿Para qué sirven?

- Para poder definir un **orden diferente** a nuestra código, dependiendo de las entradas de los usuarios, o de los valores de ciertas variables. Necesitamos utilizar "**estructuras de selección**".



Estructura if-then

Es la más básica de la estructuras de selección.

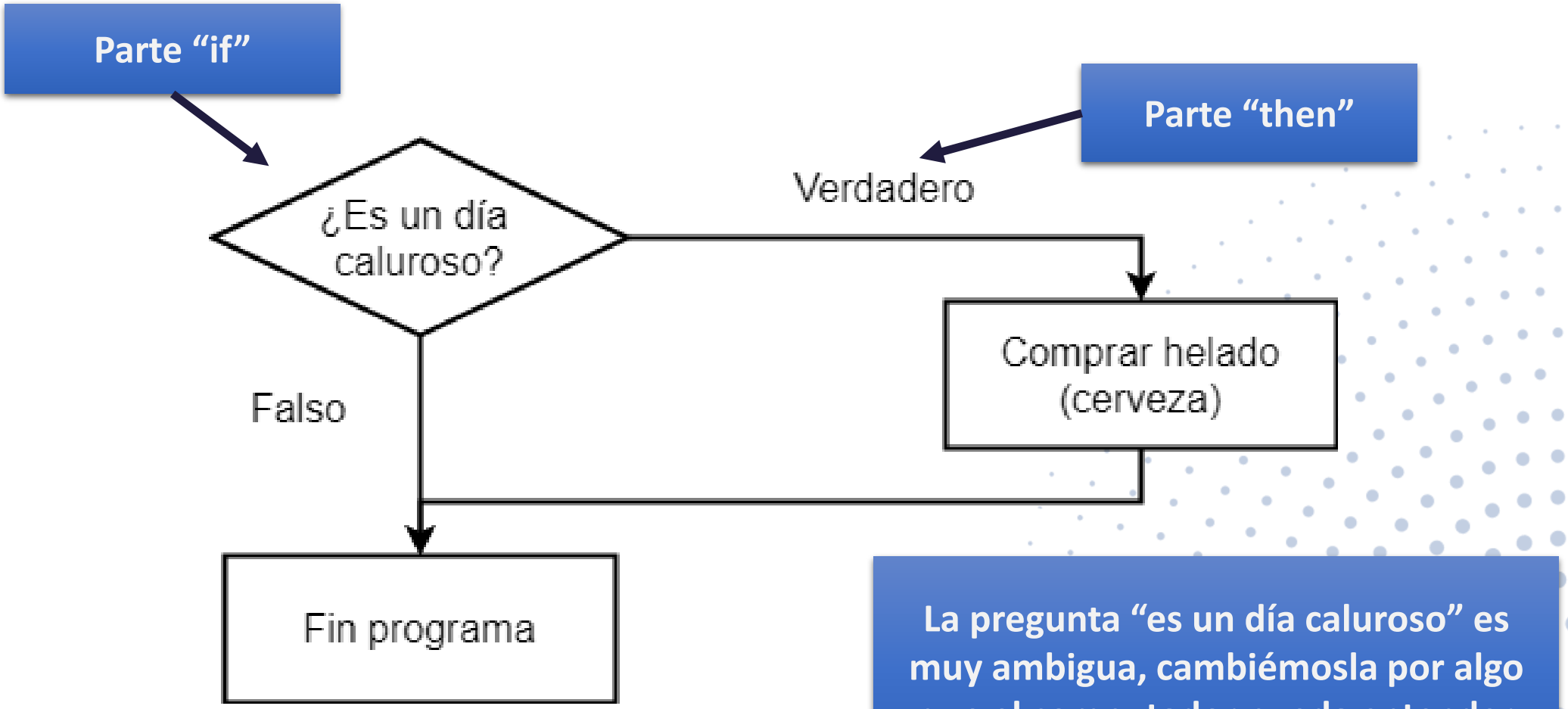
Su funcionamiento es el siguiente:

- Si una condición particular es verdadera, entonces la porción “then” es ejecutada; de lo contrario, la porción “then” no es ejecutada.

Ejemplo:

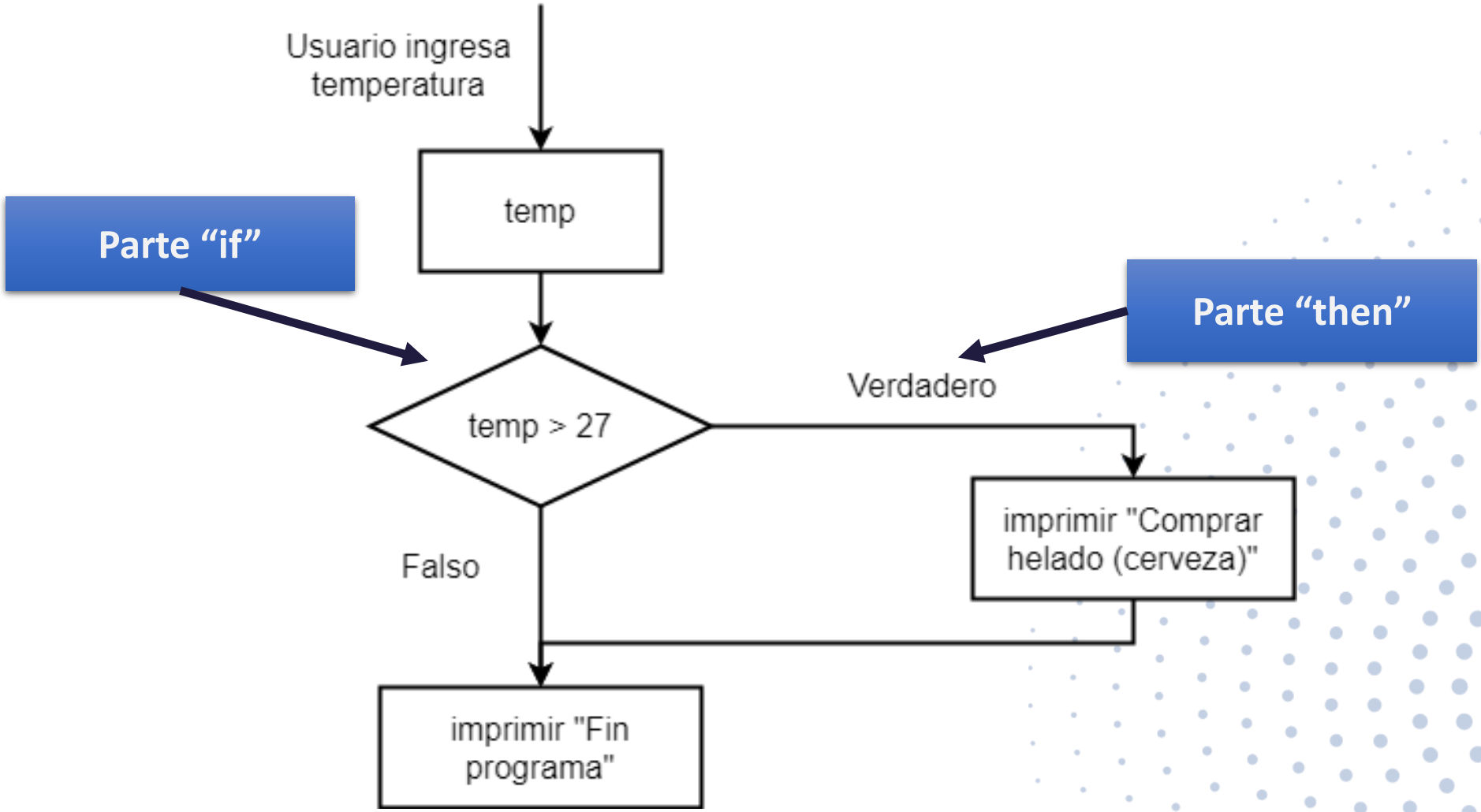
- Similar a lo lenguajes naturales donde se puede decir: ¿Es un día caluroso? Entonces, compro un helado (una cerveza).

Estructura if-then – II



La pregunta "es un día caluroso" es muy ambigua, cambiémosla por algo que el computador pueda entender

Estructura if-then – III



Código en Java

Parte "if"

Codifique el siguiente programa

```
import java.util.Scanner;
public class Principal {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Ingrese la temperatura: ");
        int temp = scan.nextInt();
        if(temp > 27){
            System.out.println("Comprar helado (cerveza)");
        }
        System.out.println("Fin programa");
    }
}
```

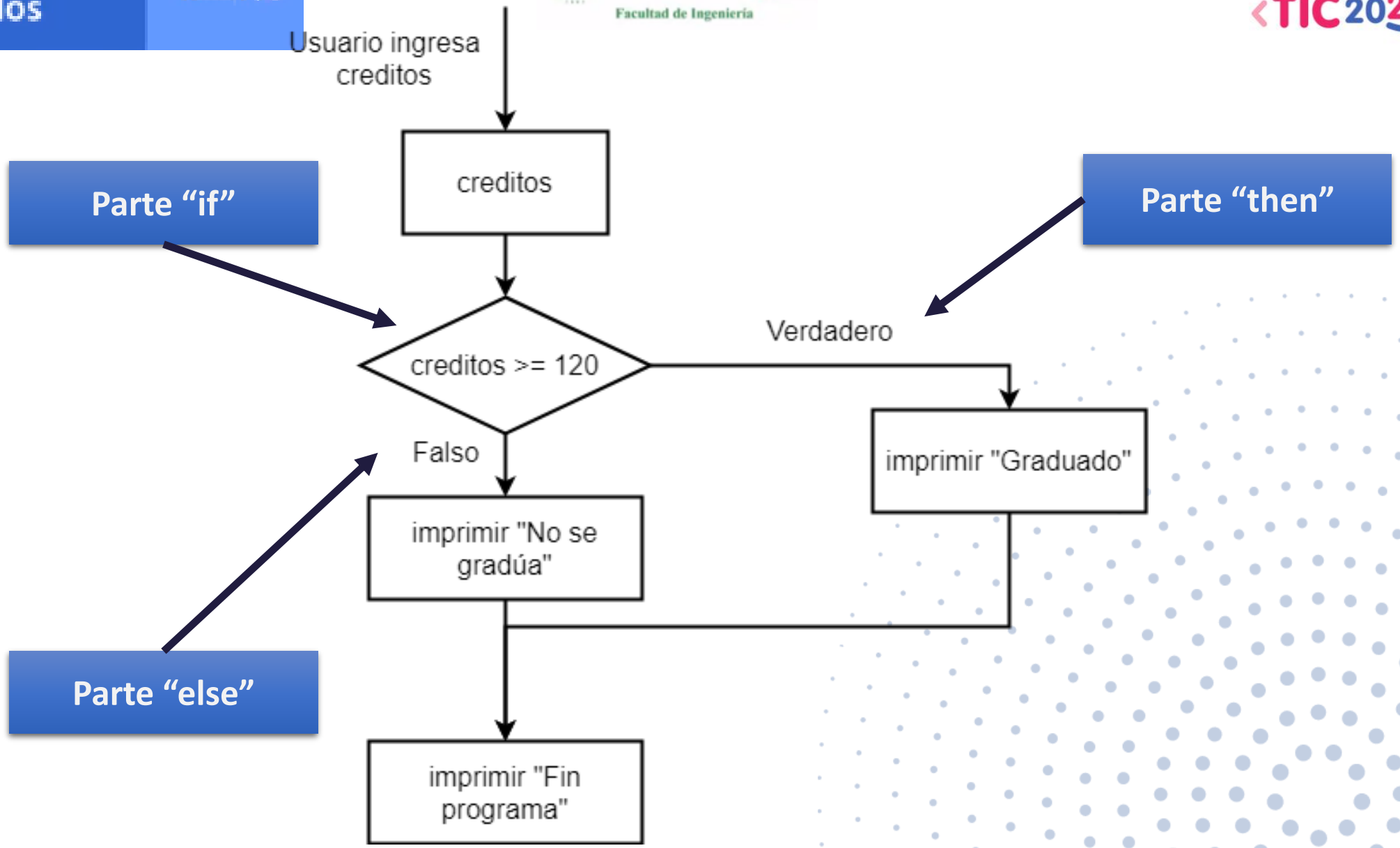
Parte "then" (lo que hay entre las llaves)

Estructura If-then-else

- La estructura If-then es muy necesaria para lograr cambiar el control de flujo de un programa (ofrecer varios caminos).

- Sin embargo, que pasa si se requiere que el programa deba hacer el algo en ambas alternativas: caso verdadero y caso falso.

  - **Ejemplo:** si el total de créditos aprobados es mayor o igual a 120 la persona se gradúa; en caso contrario no se gradúa.


Usuario ingresa creditos
creditos
Parte "if"
Parte "then"
creditos >= 120
Verdadero
Falso
imprimir "Graduado"
imprimir "No se gradúa"
Parte "else"
imprimir "Fin programa"

Código en Java

Parte "if"

Codifique el siguiente programa

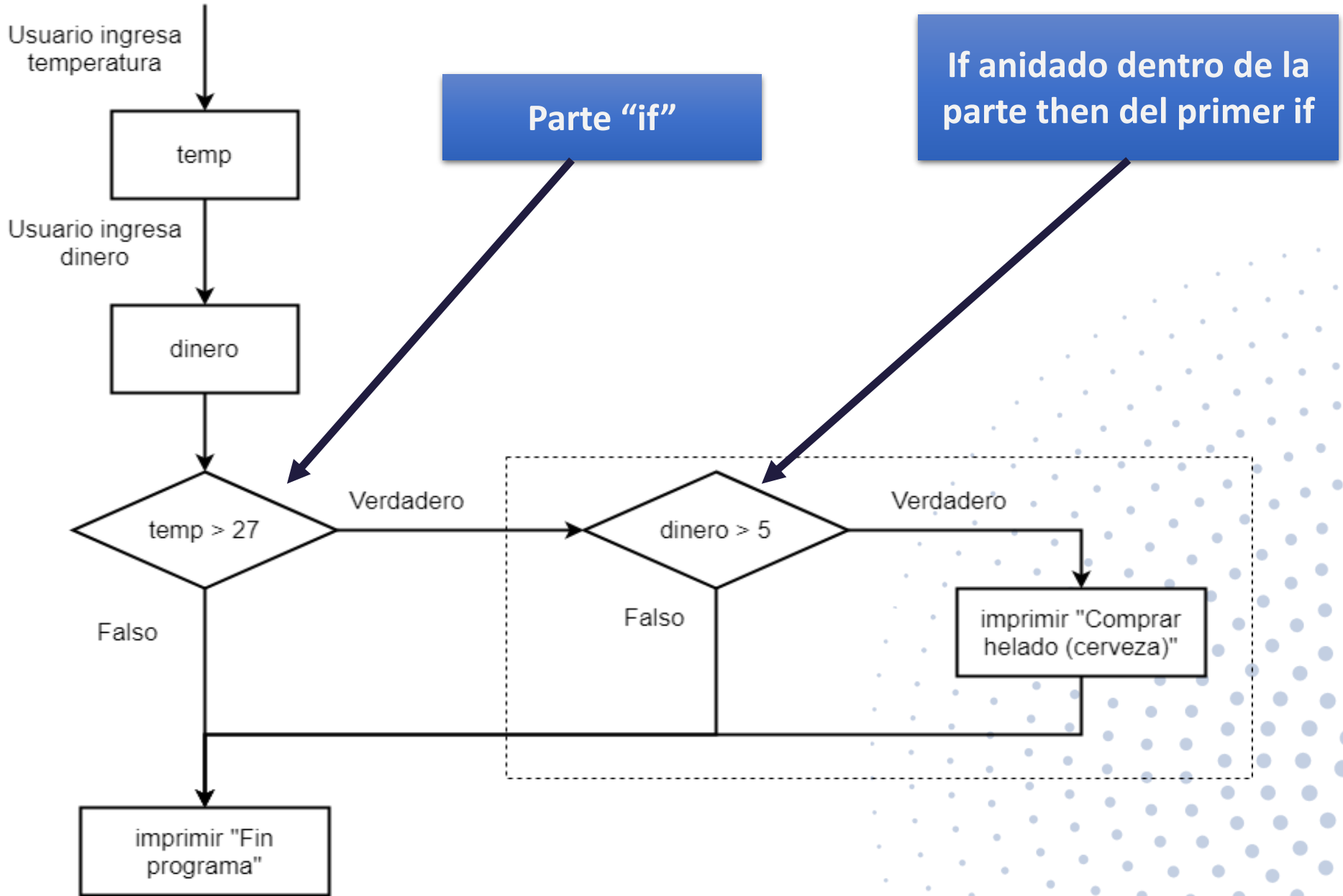
Parte "then"

Parte "else"

```
import java.util.Scanner;
public class Principal {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Ingrese los créditos aprobados: ");
        int creAprobados = scan.nextInt();
        if(creAprobados >= 120){
            System.out.println("Graduado");
        }else{
            System.out.println("No se gradúa");
        }
        System.out.println("Fin programa");
    }
}
```

If anidados

También es posible contener ifs, dentro de otros ifs. Veamos el siguiente ejemplo

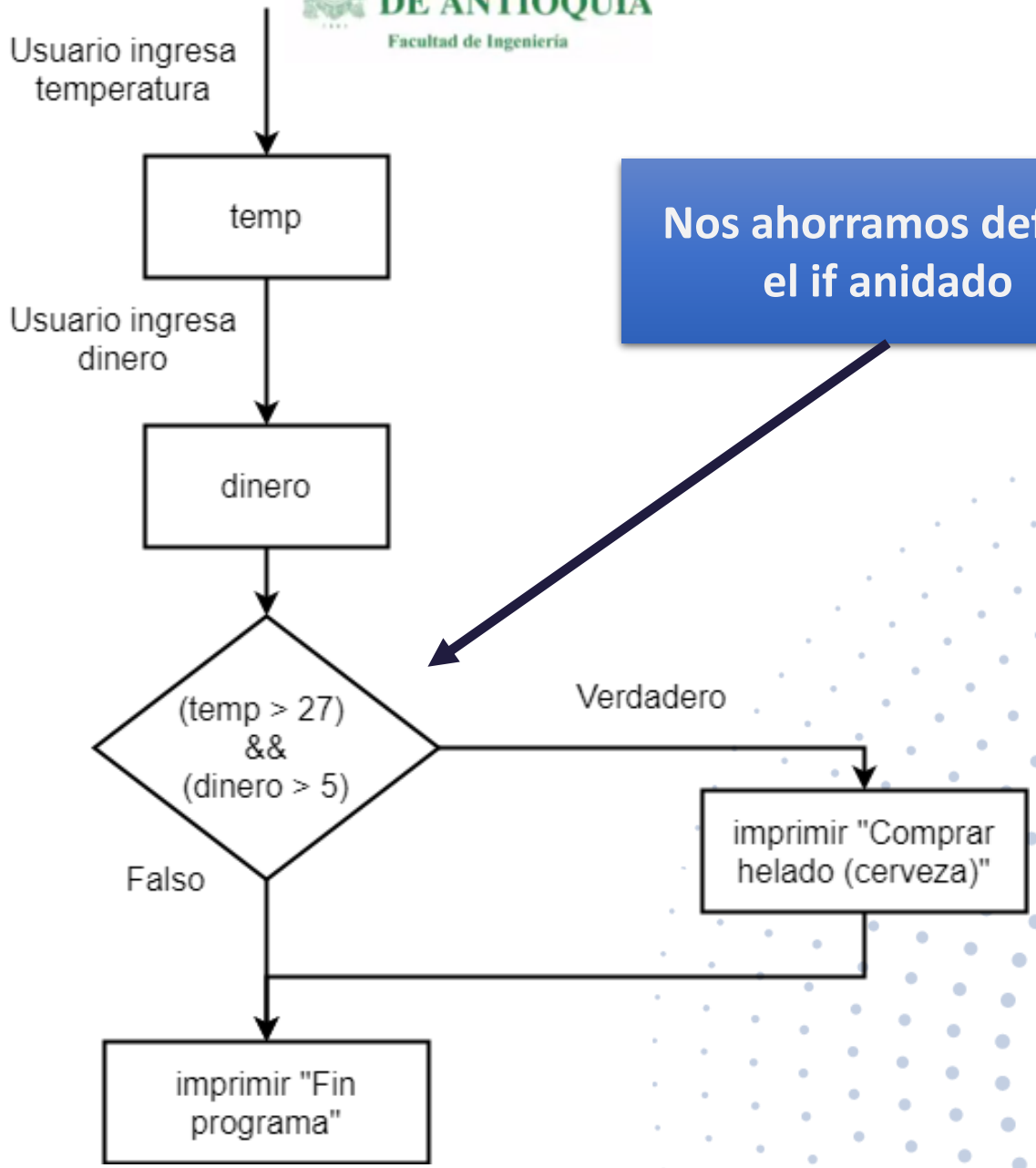

Usuario ingresa temperatura
temp
Usuario ingresa dinero
dinero
temp > 27
Verdadero
Falso
dinero > 5
Verdadero
Falso
imprimir "Comprar helado (cerveza)"
imprimir "Fin programa"
Parte “if”
If anidado dentro de la parte then del primer if

Código en Java

```java
import java.util.Scanner;
public class Principal {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Ingrese la temperatura: ");
        int temp = scan.nextInt();
        System.out.println("Ingrese el dinero: ");
        int dinero = scan.nextInt();
        if(temp > 27){
            if(dinero > 5){
                System.out.println("Comprar helado (cerveza)");
            }
        }
        System.out.println("Fin programa");
    }
}
```

Opciones

```
Ingrese la temperatura:
30
Ingrese el dinero:
6
Comprar helado (cerveza)
Fin programa
```



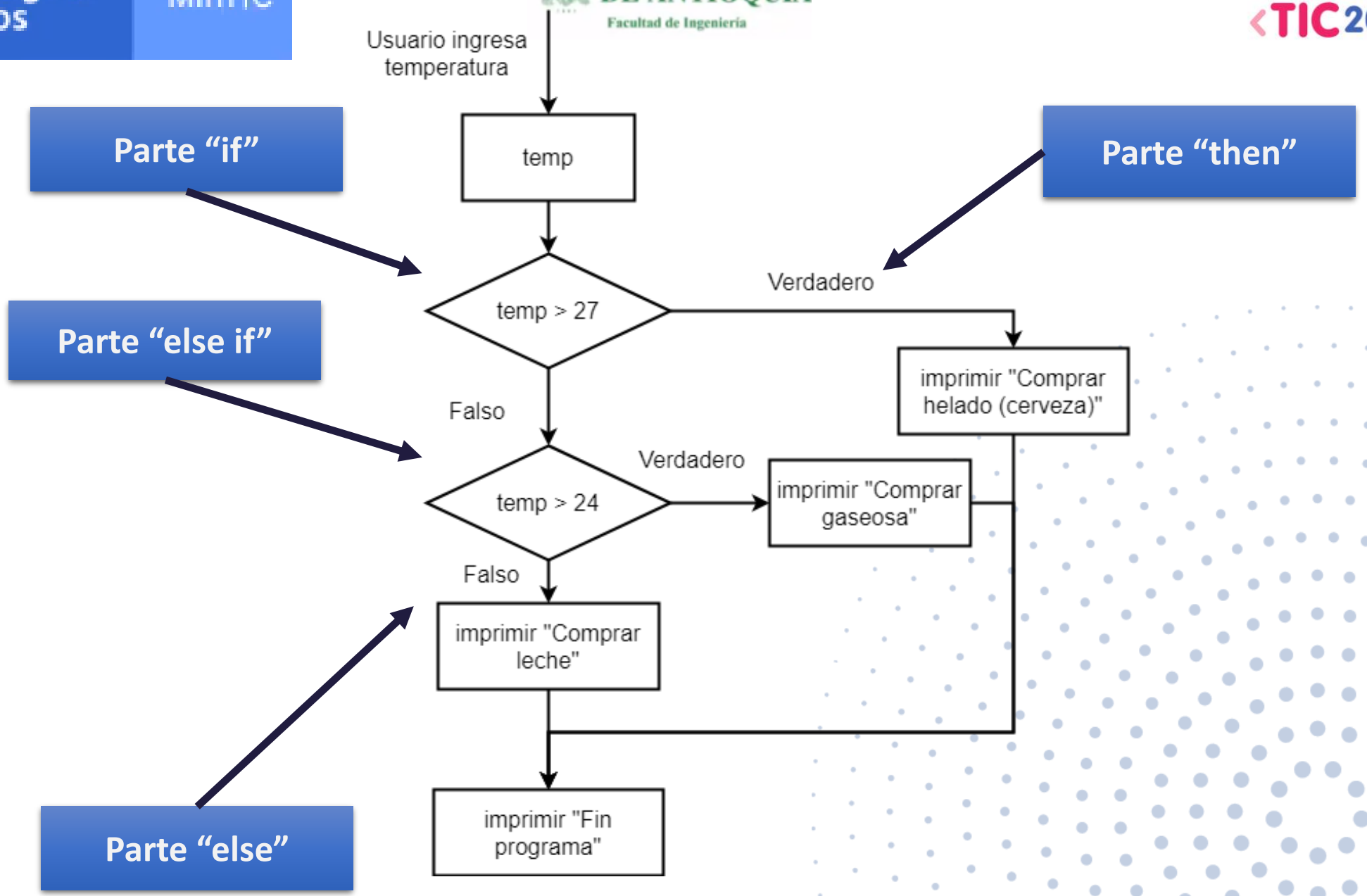
Nos ahorramos definir el if anidado

Código en Java

```java
import java.util.Scanner;
public class Principal {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Ingrese la temperatura: ");
        int temp = scan.nextInt();
        System.out.println("Ingrese el dinero: ");
        int dinero = scan.nextInt();
        if((temp > 27) && (dinero > 5)){
                System.out.println("Comprar helado (cerveza)");
        }
        System.out.println("Fin programa");
    }
}
```

Estructura If-then-else if-…

- La estructura If-then-else if-… nos permite ofrecer múltiples caminos dependiendo de múltiples comprobaciones.
- El **else if** sirve para añadir una nueva condición en caso de que la condición anterior no se cumpla (sea falsa).
- Una instrucción *if* puede incluir cualquier número de bloques *else if*.
- "*else if*" significa (de lo contrario, si …).
- Como máximo, se ejecuta un solo *else if*, y ese *else if* se ejecutará solo si todas las condiciones anteriores se evaluaron como falsas.
  - **Ejemplo:** si la temperatura es mayor a 27 compramos helado (cerveza); **de lo contrario, si** la temperatura es mayor a 24 compramos gaseosa; en caso contrario, compramos leche.


Usuario ingresa temperatura
temp
Parte "if"
Parte "then"
temp > 27
Verdadero
Parte "else if"
imprimir "Comprar helado (cerveza)"
Falso
Verdadero
temp > 24
imprimir "Comprar gaseosa"
Falso
imprimir "Comprar leche"
Parte "else"
imprimir "Fin programa"

Código en Java

```java
import java.util.Scanner;
public class PrincipalCondicionales
{
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Ingrese la temperatura: ");
        int temp = scan.nextInt();
        if(temp > 27){
            System.out.println("Comprar helado (cerveza)");
        }else if(temp > 24){
            System.out.println("Comprar gaseosa");
        }else{
            System.out.println("Comprar leche");
        }
        System.out.println("Fin programa");
    }
}
```

Pero que otra estructura utilizar si tuviéramos muchos else-if

Ejercicio [operadoresRelaciones]

- Traduzca el código de Python a Java. Los datos de la lotería del usuario se debe solicitar por la consola.

```
loteria = 54
loteriaD = loteria//10
loteriaU = loteria%10

usuario = 86
usuarioD = usuario//10
usuarioU = usuario%10

if(loteria == usuario):
        print("Ganó 10000")
elif((loteriaD == usuarioU) and (usuarioD == loteriaU)):
        print("Ganó 3000")
elif((loteriaD == usuarioD or loteriaD == usuarioU) or (loteriaU == usuarioU or loteriaU== usuarioD)):
        print("Ganó 1000")
else:   print("Ninguna coincidencia, lo siento")
```

Switch

- Java provee una estructura llamada “switch”, que facilita la codificación en el caso anterior.

```
import java.util.Scanner;
public class Principal {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Ingrese la nota: ");
        int nota = scan.nextInt();
        switch(nota){
            case 10:
                System.out.println("Teso");
                break;
            case 9:
                System.out.println("Tesito");
                break;
            case 8:
                System.out.println("Vas bien");
                break;
            default:
                System.out.println("Puedes mejorar");
                break;
        }
    }
}
```

Ejercicio [operacionesSwitch]

- Cree dos variables enteras, inicializados con los valores que desee.
- Solicite al usuario un número:
  - Si ingresa 1, el sistema debe imprimir la suma de los dos números.
  - Si ingresa 2, el sistema debe imprimir la resta de los dos números.
  - Si ingresa 3, el sistema debe imprimir la multiplicación de los dos números.
  - Si ingresa 4, el sistema debe imprimir la división de los dos números.
  - Si ingresa otro número, el sistema debe imprimir “Opción desconocida”.

Estructuras de Iteración

Luisa Fernanda Restrepo.

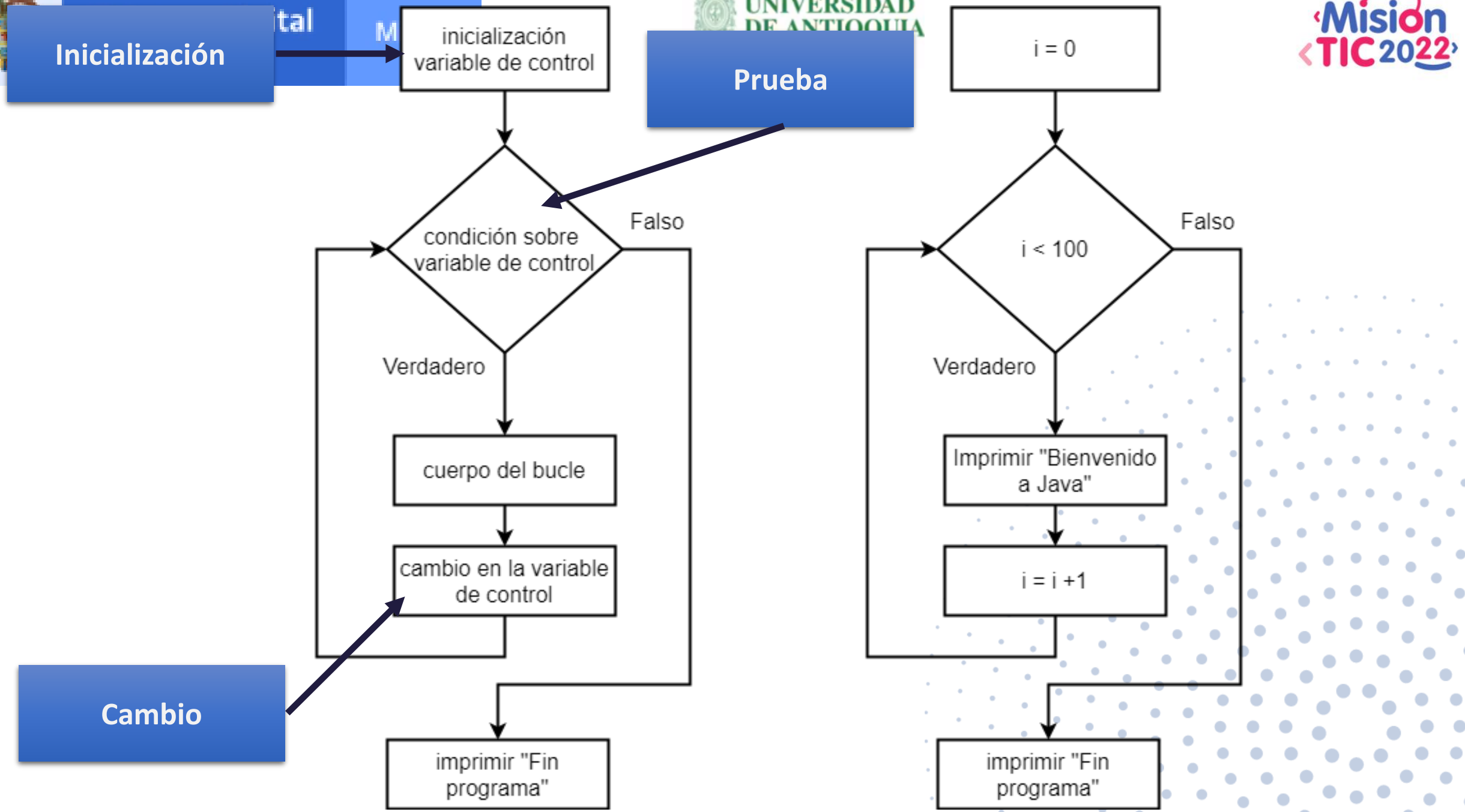
Bucles

- Afortunadamente, Java provee unas estructuras llamadas “bucles” (loop).
- Los **bucles** permiten controlar cuantas veces se realiza una operación, o una secuencia de operaciones.
- Usando un bucle, se le puede decir a la computadora que muestre un texto cien veces sin tener que codificar “la declaración de impresión” cien veces. Veamos un ejemplo:

```
public class Principal {
    public static void main(String[] args) {
        for(int i = 0; i < 100; i++){
            System.out.println("Bienvenido a Java");
        }
    }
}
```

Opciones

```
Bienvenido a Java
Bienvenido a Java
Bienvenido a Java
Bienvenido a Java
Bienvenido a Java
Bienvenido a Java
Bienvenido a Java
Bienvenido a Java
Bienvenido a Java
```


Inicialización
inicialización variable de control
Prueba
condición sobre variable de control
Falso
Verdadero
cuerpo del bucle
cambio en la variable de control
Cambio
imprimir "Fin programa"
i = 0
i < 100
Falso
Verdadero
Imprimir "Bienvenido a Java"
i = i +1
imprimir "Fin programa"

Bucles – II

Java proporciona tres tipos de estructuras de bucles las cuales se detallarán a continuación: “for”, “while”, y “do-while”.

While

- Un bucle **while** ejecuta instrucciones repetidamente mientras la condición es verdadera.
- Generalmente, la sintaxis para el bucle while es la siguiente:

```
inicializacion-variable-control
while (condicion-variable-control)
{
    // cuerpo del while
    cambio-variable-control
}
```

While código Java

Inicialización

Prueba

```
public class Principal {
    public static void main(String[] args) {
        int i = 0;
        while(i<100){
            System.out.println(i);
            i++;
        }
    }
}
```

Cambio

Modifique el programa anterior, imprima los números 5, 10, 15, … hasta el 100

Bucles controlados

- En el programa anterior, conocemos exactamente cuantas veces se ejecutará el bucle while. A esto se le conoce como **bucle controlado por contador**.
- Otra técnica común para controlar un ciclo es definir un valor especial que recogeremos del usuario.
- Este valor de entrada especial, conocido como **valor centinela**, significará el final de la ejecución de un bucle.
- Un bucle que utiliza un valor centinela para controlar su ejecución se denomina **bucle controlado por centinela**.

Bucle controlado por centinela

```java
import java.util.Scanner;
public class Principal {
    public static void main(String[] args) {
        System.out.println("Ingrese un entero no negativo,");
        System.out.println("o un entero negativo para parar: ");
        Scanner scan = new Scanner(System.in);
        int num = scan.nextInt();
        while(num > 0) {
            // cuerpo del bucle
            System.out.println("Ingrese un entero no negativo,");
            System.out.println("o un entero negativo para parar: ");
            num = scan.nextInt();
        }
        System.out.println("Fin programa");
    }
}
```

Ejercicio [whileCentinela]

- Inicialice un contador en 0.
- Desarrolle un ciclo while que le pida al usuario un número.
    1. Si el número es mayor a cero, súmele ese valor al contador, e imprima por pantalla el contador.
    2. Si el número es menor a cero, restele ese valor al contador, e imprima por pantalla el contador.
    3. Si el número es igual a cero, finalice la ejecución del while.

Do while

- Un ciclo **do-while** es lo mismo que un ciclo while, excepto que ejecuta primero el cuerpo del ciclo y luego verifica la condición de continuación del ciclo.
- Generalmente, la sintaxis para el bucle do while es la siguiente:

```
inicializacion-variable-control
do{
    // cuerpo del while
    cambio-variable-control
}
while (condicion-variable-control);
```

Do while – ejemplo

```java
public class PrincipalBucles
{
    public static void main(String[] args) {
        int i = 1;
        while(i<11){
            System.out.println(i);
            i++;
        }
    }
}
```

```java
public class PrincipalBucles
{
    public static void main(String[] args) {
        int i = 1;
        do{
            System.out.println(i);
            i++;
        }while(i<11);
    }
}
```

¿Diferencias entre el while y el do-while?

Do while vs while

- La diferencia entre un bucle while y un bucle do-while es el orden en que se evalúa la condición de continuación del bucle, y el orden en el que se ejecuta el cuerpo del bucle.
- En el caso de un bucle do while, el cuerpo del bucle se ejecuta al menos una vez.
- Siempre se puede escribir un bucle utilizando el bucle while o el bucle do-while.
- A veces, una es una opción más conveniente que la otra.

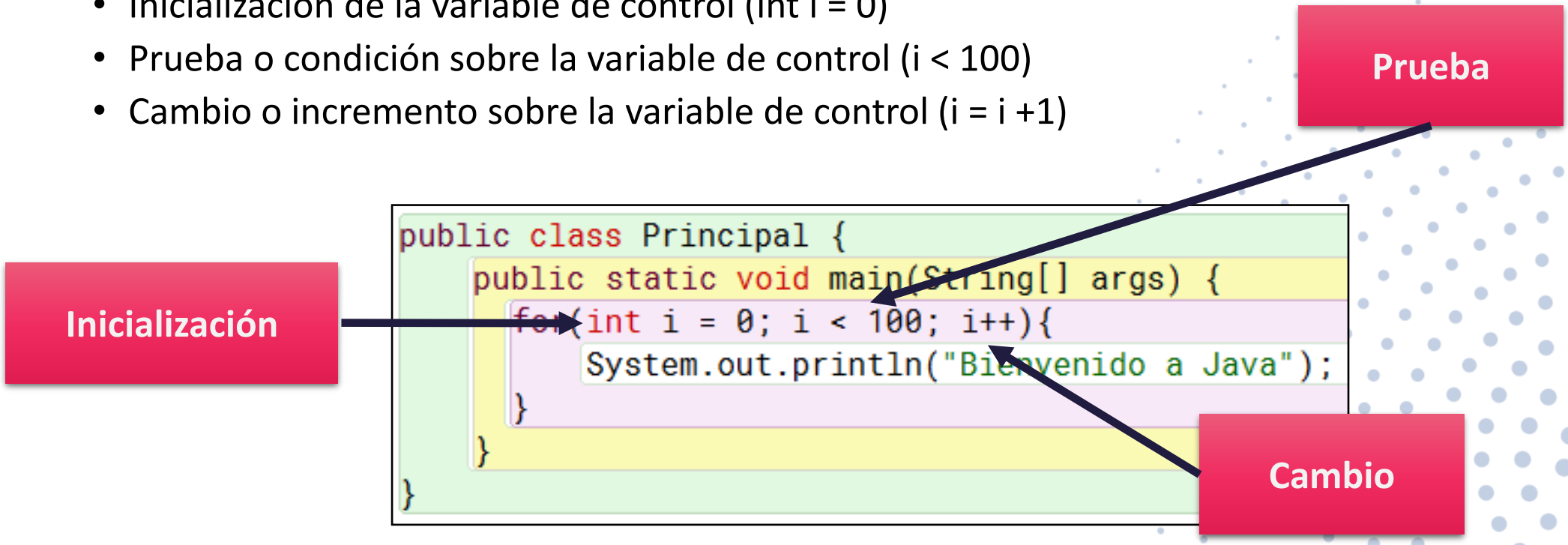
- *Utilice un bucle do-while si tiene declaraciones dentro del bucle que deben ejecutarse al menos una vez.*

Ejercicio [doWhileCentinela]

- Modifique el ejercicio “whileCentinela” para que funcione como un do while.

Bucle for

- La declaración del bucle “**for**” comienza con la palabra clave “for”.
- Luego siguen un par de paréntesis que encierran la estructura de control del bucle “for”. La cual esta definida por 3 partes (separadas entre si por punto y coma “;”):
  - Inicialización de la variable de control (int i = 0)
  - Prueba o condición sobre la variable de control (i < 100)
  - Cambio o incremento sobre la variable de control (i = i +1)



Ejercicio

- Cree un nuevo programa que imprima los números pares del 20 al 2.
- **Nota:** no utilice condicionales.

```
public class Principal {
    public static void main(String[] args) {
        for(int i = 20; i > 0; i = i-2){
            System.out.println(i);
        }
    }
}
```

Do while – ejemplo

```
public class PrincipalBucles
{
    public static void main(String[] args) {
        for(int i = 1; i < 11; i++){
            System.out.println(i);
        }
    }
}
```

¿Diferencias entre el while y el do-while?

```
public class PrincipalBucles
{
    public static void main(String[] args) {
        int i = 1;
        while(i<11){
            System.out.println(i);
            i++;
        }
    }
}
```

```
public class PrincipalBucles
{
    public static void main(String[] args) {
        int i = 1;
        do{
            System.out.println(i);
            i++;
        }while(i<11);
    }
}
```

Arreglos

Luisa Fernanda Restrepo.

Introducción - II

- Suponga que se desea almacenar el valor de 100 números enteros (edades de 100 personas).

- Con lo visto hasta el momento, nos tocaría definir 100 variables tipo *int*, y almacenar en cada una de ellas el valor de cada edad de cada persona.

- Con un arreglo, podemos definir una sola variable, que tenga un tamaño de 100, y almacenar en ella todas las 100 edades.

Tipos de arreglos

- Existen 2 tipos de arreglos (estáticos y dinámicos):

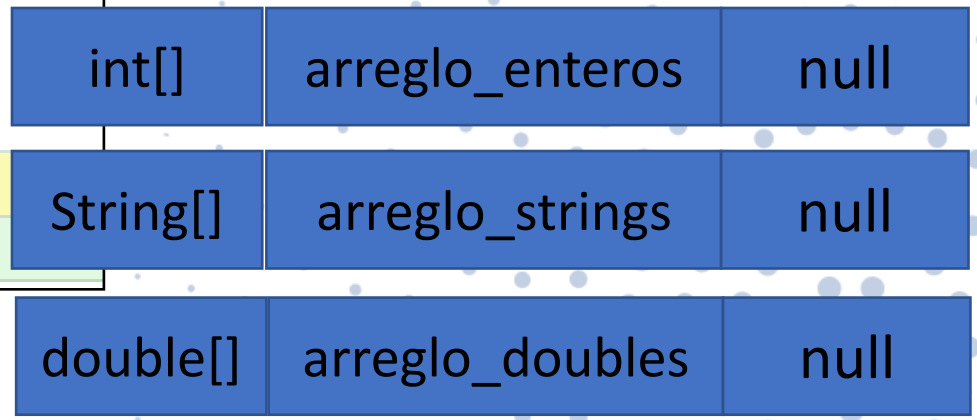
- Un **arreglo estático** tiene un número fijo de elementos (tamaño/longitud) que queda determinado desde la creación de la estructura, y a partir de allí permanece fijo.
  - Ejemplo: *int[] arreglo_estatico_a = new int[5];*

- Un **arreglo dinámico** es similar a uno estático, sin embargo, el tamaño de un arreglo dinámico se puede ampliar y contraer durante la ejecución del programa.
  - Ejemplo: *ArrayList<Integer> arreglo_dinamico_a = new ArrayList<Integer>();*

Declaración

- Para declarar una variable que almacene arreglos, basta con agregar corchetes al tipo del dato.

```
public class PrincipalArreglos
{
    public static void main(String[] args){
        int[] arreglo_enteros;
        String[] arreglo_strings;
        double[] arreglo_doubles;
    }
}
```



Declaración – II

- Para inicializar un arreglo debemos utilizar la palabra reservada ***new*** seguida del **tipo de dato**, y luego utilizar corchetes y definir dentro de los **corchetes** la **cantidad** de elementos que deseamos almacenar.

```
public class PrincipalArreglos
{
    public static void main(String[] args){
        int[] arr_int = new int[3];
    }
}
```

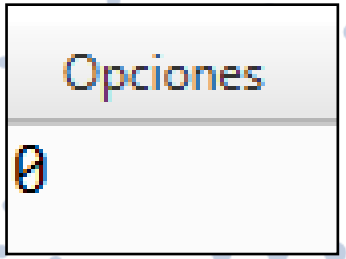
Definición de un arreglo de enteros que podría almacenar hasta 3 enteros

Acceso

- Para **acceder** a un elemento individual de un arreglo, basta con colocar el nombre de la variable, seguido por el índice del elemento al que se quiere acceder.

```
public class PrincipalArreglos
{
    public static void main(String[] args){
        int[] arr_int = new int[3];
        System.out.println(arr_int[2]);
    }
}
```

¿Qué imprime?



¿Qué pasa si accedo a la posición arr_int[3]?

R/= ArrayIndexOutOfBoundsException

Acceso y asignación

- Veamos un ejemplo de acceso y asignación de valores a posiciones de un arreglo.

Codifíquelo

```
public class PrincipalArreglos
{
    public static void main(String[] args){
        int[] arr_int = new int[3];
        arr_int[1]=4;
        System.out.println(arr_int[1]);
        int[] arr_int2 = arr_int;
        arr_int2[1]=8;
        System.out.println(arr_int[1]);
        System.out.println(arr_int2[1]);
    }
}
```

¿Qué imprime?

Opciones

4

8

8

Ejercicio [creacionArreglo]

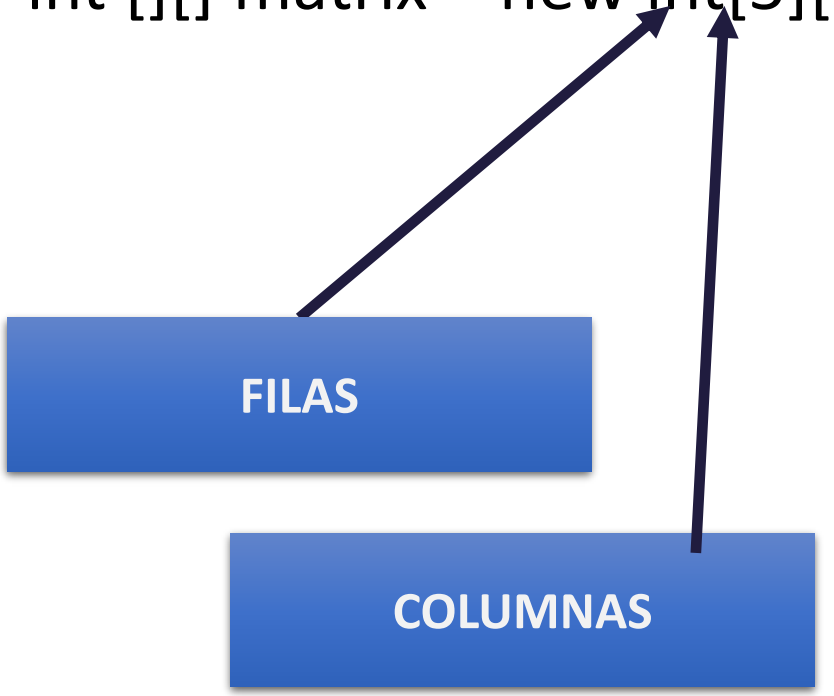
- Pídale al usuario un tamaño de un arreglo por pantalla (int)
- Cree un arreglo del tamaño recibido.
- Modifique todas los valores de todas las posiciones del arreglo creado anteriormente, para que ahora todos contengan el número 5.

Ejemplo:

- Usuario ingresa 4
- Creamos un arreglo de tamaño 4.
- A todas las 4 posiciones del arreglo anterior les establecemos el valor 5.

Arreglos de dos dimensiones

```
int [][] matrix = new int[5][5];
```



	[0]	[1]	[2]	[3]	[4]
[0]	0	0	0	0	0
[1]	0	0	0	0	0
[2]	0	0	0	0	0
[3]	0	0	0	0	0
[4]	0	0	0	0	0

```
matrix = new int[5][5];
```

Arreglos de dos dimensiones

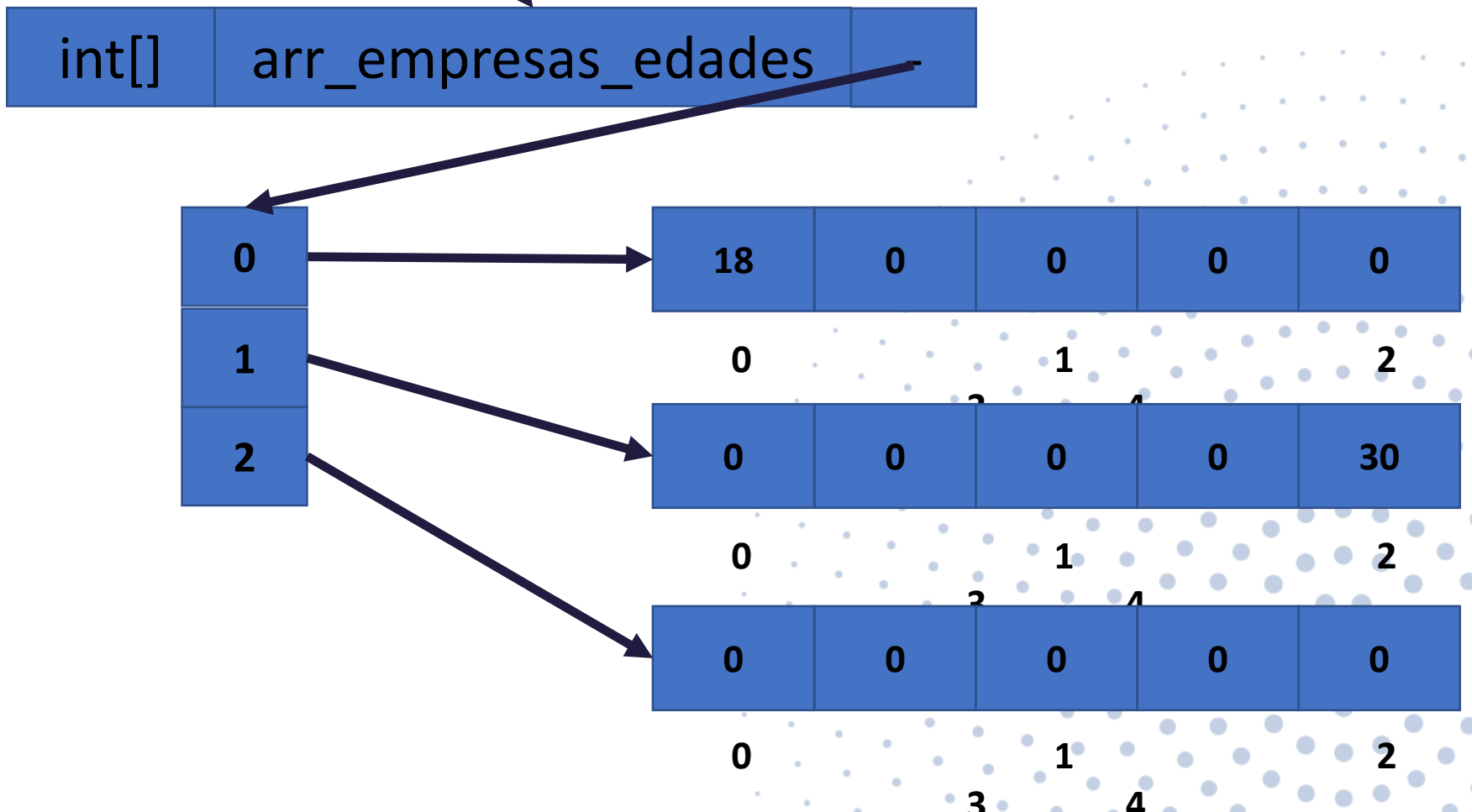
- Java también permite definir arreglos de dos dimensiones (Almacenar una matrix o tabla).
- En este caso basta con agregar otro par de corchetes a la definición del arreglo.

```
public class ArreglosParte2
{
    public static void main(String[] args){
        int[][] arr_empresas_edades = new int[10][5];
        arr_empresas_edades[0][0] = 18;
        arr_empresas_edades[1][4] = 30;
        System.out.println(arr_empresas_edades[1][4]);
        System.out.println(arr_empresas_edades[4][4]);
    }
}
```

Opciones
30
0

Simulamos almacenar las edades correspondientes a 5 empleados de 10 empresas

```
int[][] arr_empresas_edades = new int[3][5];
arr_empresas_edades[0][0] = 18;
arr_empresas_edades[1][4] = 30;
```



Entrada y salida

Codifíquelo

```
public static void main(String[] args){
    Scanner scan = new Scanner(System.in);
    int[][] arr_empresas_edades = new int[2][5];
    for(int i=0; i<arr_empresas_edades.length; i++){
        System.out.println("--Ingresando datos de la empresa "+i+"--");
        for(int j=0; j<arr_empresas_edades[0].length; j++){
            System.out.print("Ingresando edad empleado "+j+" - ");
            System.out.println("empresa "+i+" :");
            arr_empresas_edades[i][j] = scan.nextInt();
        }
    }

    for(int i=0; i<arr_empresas_edades.length; i++){
        System.out.println("--Empresa "+i+"--");
        for(int j=0; j<arr_empresas_edades[0].length; j++){
            System.out.print("Edad empleado "+j+" :");
            System.out.println(arr_empresas_edades[i][j]);
        }
    }
}
```

```
--Empresa 0--
Edad empleado 0 :10
Edad empleado 1 :43
Edad empleado 2 :25
Edad empleado 3 :67
Edad empleado 4 :87
--Empresa 1--
Edad empleado 0 :34
Edad empleado 1 :21
Edad empleado 2 :23
Edad empleado 3 :45
Edad empleado 4 :65
```

Ejercicio [promedioArreglos]

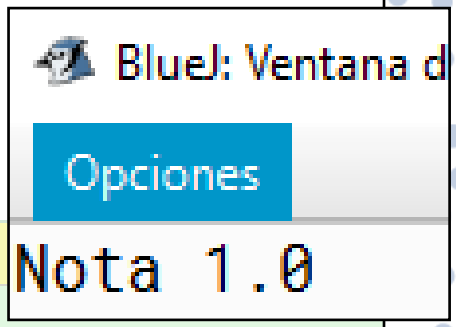
- Modifique el código anterior para que muestre el promedio de edades de los trabajadores de cada empresa.

Arreglos asimétricos

- También es posible definir arreglos de dos dimensiones asimétricos. Veamos el siguiente ejemplo.

```
public class ArreglosParte2
{
    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        int[][] arr_empresas_edades = new int[2][];
        arr_empresas_edades[0] = new int[5]; // empresa con 5 empleados
        arr_empresas_edades[1] = new int[3]; // empresa con 3 empleados

        double[][] arr_estu_notas = {{1.0, 5.0, 4.8},
                                     {3.0, 2.0}};
        System.out.println("Nota "+arr_estu_notas[0][0]);
    }
}
```



Bonus: Declaración – Arreglos dinámicos

```java
import java.util.ArrayList;
public class PrincipalArreglos
{
    public static void main(String[] args) {
        ArrayList<Integer> edades = new ArrayList<Integer>();
        ArrayList<String> nombres = new ArrayList<String>();
    }
}
```

Primitive Data Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

Se deben utilizar envoltorios, no se pueden definir con tipos primitivos.

Bonus: Acceso, agregación, eliminación, y asignación – arreglos dinámicos

```java
import java.util.ArrayList;
public class PrincipalArreglos
{
    public static void main(String[] args) {
        ArrayList<Integer> edades = new ArrayList<Integer>();
        edades.add(4); edades.add(8); edades.add(60);
        System.out.println(edades.get(1));
        edades.set(0,40);
        System.out.println(edades.get(0));
        edades.remove(0);
        System.out.println(edades.get(0));
        System.out.println(edades.size());
    }
}
```

ArrayList es un arreglo dinámico. No hay que definir tamaño, el tamaño interno es dinámico.

add sirve para agregar elementos al arreglo

¿Qué imprime?

remove sirve para eliminar elementos del arreglo (recibe la posición)

set sirve para asignar valores a posiciones especificas

get sirve para acceder a elementos del arreglo

Opciones

8
40
8
2

Referencias

Basado en el material elaborado por: Daniel Correa (docente EAFIT).

Liang, Y. D. (2017). Introduction to Java programming: comprehensive version. Eleventh edition. Pearson Education.

Streib, J. T., & Soma, T. (2014). *Guide to Java*. Springer Verlag.