

Hour of code 2022

Introducción a Java

Alejandra Gavino-Dias





Índice

✿ TIPOS DE DATOS

✿ ESTRUCTURAS DE CONTROL

✿ MÉTODOS:
PROCEDIMIENTOS Y
FUNCIONES

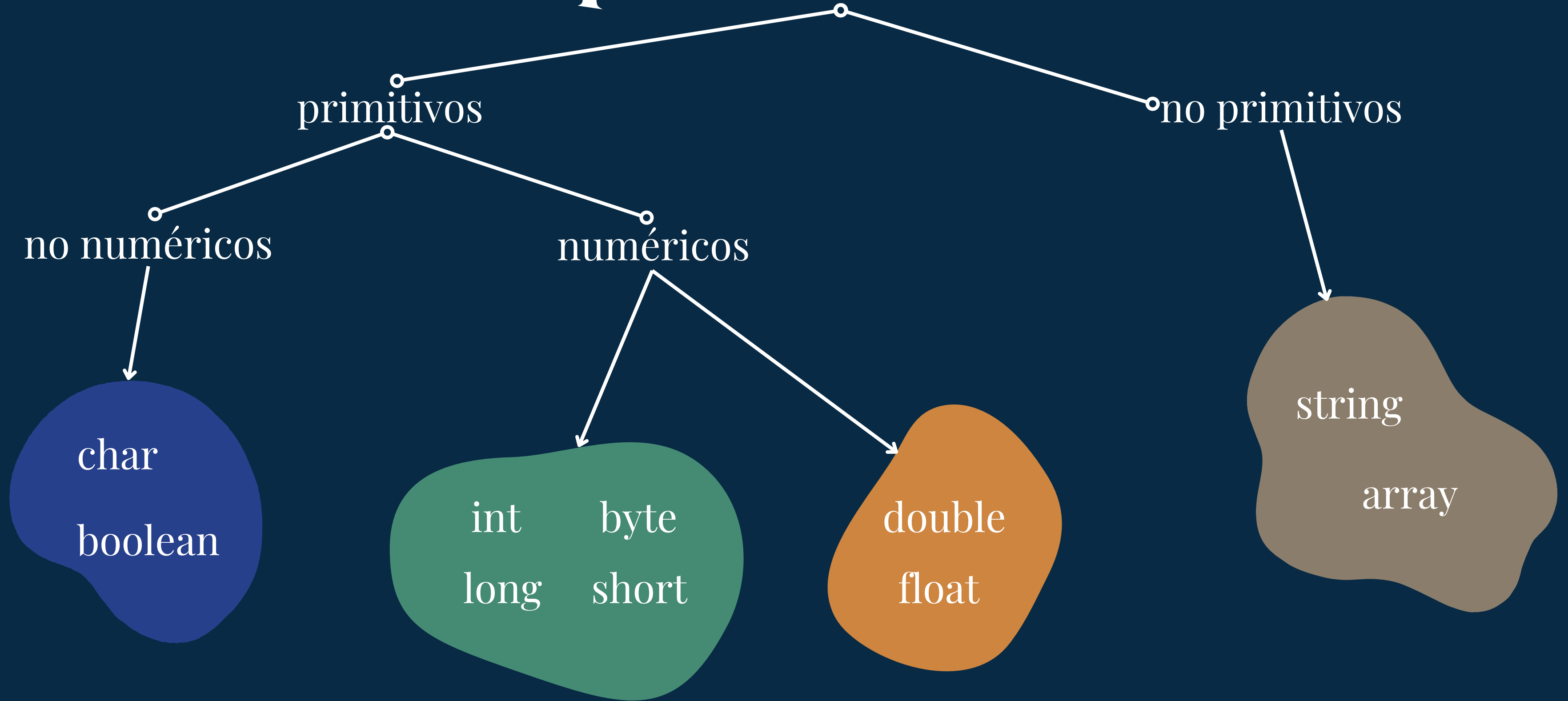
✿ FICHEROS Y
EXCEPCIONES

✿ CLASES

✿ MALAS PRÁCTICAS



Tipos de datos



Operadores

Boolean

| OR

& AND

^ XOR

! NOT

== EQUAL

!= NOT EQUAL

A	B	A B A OR B	A & B A AND B	A^B	!A
TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
TRUE	FALSE	TRUE	FALSE	TRUE	FALSE
FALSE	TRUE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

Operadores

Boolean

operaciones entre expresiones

A = 5
B = 3

$(A+B == 8) \ \&\& \ (A-B == 2)$

TRUE

$A > 3 \ \&\& \ B > 3$

FALSE

$A > 3 \ || \ B > 3$

TRUE

Operadores numéricos

aritméticos

+

-

*

/

%

incremento y
decremento

++

--

+=

-=

+=

relacionales

>

<

>=

<=

+funciones de Math!



muy útil para las estructuras de control!

Operadores numéricos

incremento y
decremento

++	var=5 var++ >> ?
--	var=5 var-- >> ?
+=	var+=3 equivalente a: ?
-=	var-=3 equivalente a: ?
=	var=2 equivalente a: ?

Operadores numéricos

incremento y
decremento

++	var=5	var++	>>	var=6
--	var=5	var--	>>	var=4
+=	var+=3	equivalente a:	var=var+3	>>var=5+3
-=	var-=3	equivalente a:	var=var-3	>>var=5-3
=	var=2	equivalente a:	var=var*2	>>var=5*2

Operadores numéricos

para datos numéricos enteros como byte, int, long o short, podemos usar sin problemas la comparación lógica ==

estas comparaciones pueden dar problemas con tipos de datos como double o float

```
public class Ej {  
    public static void main(String[] args) {  
        double d1 = 0;  
  
        for (int i = 1; i <= 8; i++) {  
            d1 += 0.1;  
            /*esto es lo mismo que d1=d1+0.1*/  
        }  
        double d2 = 0.1 * 8;  
  
        System.out.println(d1);  
        System.out.println(d2);  
    }  
}
```

```
0.7999999999999999  
0.8
```

Operadores numéricos solución al problema

asignamos un valor para epsilon (ϵ)
como error de precisión
-si no hubiera problemas de precisión:

$$d1-d2=0$$

-con errores de precisión:

$$d1-d2=\epsilon$$

```
double epsilon = 0.000001;  
double resta = d1-d2;  
boolean valor;  
valor=resta<epsilon;  
System.out.println(valor);
```

TRUE !!!

Operadores

conversión de tipos

si son tipos de datos compatibles entre sí:

```
int i = 450      >>450
long l = i       >>450
double d = i     >>450.0
```

por ejemplo:
char y boolean no son
compatibles entre sí

~~boolean b = T
char c = b~~

MAL

byte -> short -> char -> int -> long -> float -> double

Operadores

conversión de tipos

conversión explícita

```
double d= 9.64;           >>9.64  
int i = (int) d;          >>9
```

→ casting manual

double -> float -> long -> int -> char -> short -> byte

Operadores

conversión de tipos

conversión explícita

de int a String

```
int n = 10;  
String num = String.valueOf(n);
```

de String a int

```
String num = "10";  
int n = Integer.parseInt(num);
```

lo usaremos solo para validar
resultados!

Operadores

conversión de tipos

conversión explícita

de int a String

```
int n = 10;  
String s = ""+n;  
System.out.println(s);  
  
if(s.getClass()==String.class) {  
    System.out.println("String");  
}  
>>String
```

esto devolverá que
es String!!!

.getClass() solo la usaremos
para validar en este ejemplo,
esta función tiene más fondo

Operadores

String

```
System.out.println()  
.length()  
.charAt()  
.indexOf()
```

```
String cadena = "Hora del codigo"
```

```
System.out.println(cadena.length());  
>>15  
System.out.println(cadena.charAt(2));  
>>r  
System.out.println(cadena.indexOf("o"));  
>>1
```

Aunque haya varias 'o' nos
devuelve el índice de la primera
aparición

Operadores

Arrays

definición array

`nombre = new tipo[tamaño];`

`vector = new int[5];`

`tipo[] nombre = new tipo[tamaño];`

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

`tipo[] nombre = {contenido};`

`int[] vector= {0,1,2,3,4};`

Operadores

Arrays

definición array

nombre = new tipo[tamaño];

```
vector = new int[5];  
vector[0]=0;  
vector[1]=1;  
vector[2]=2;  
vector[3]=3;  
...  
vector = new int[5]{0,1,2,3,4};
```

```
System.out.println(vector.length);  
>>5
```

```
matriz= new int[2][3];
```

```
//numero de filas
```

```
matriz.length
```

```
//numero de columnas
```

```
matriz[0].length
```

```
//acceso
```

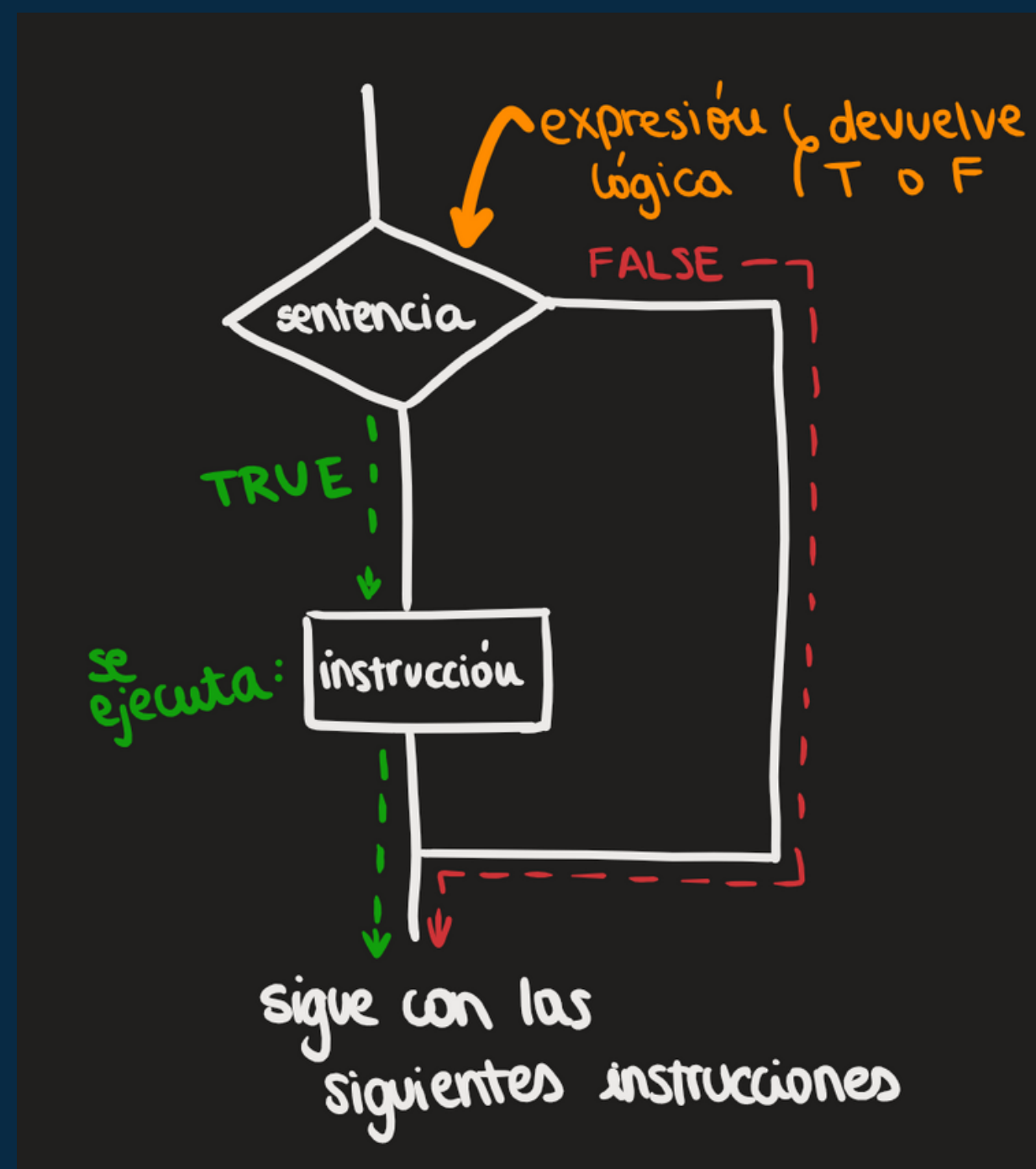
```
int primero = vector[0]
```

recorreremos arrays con bucles
(¡próximamente!)

Estructuras de control

IF

evaluamos una expresión condicional, si esta se cumple (TRUE), entonces ejecutaremos el código correspondiente. Si el resultado de la expresión fuera FALSE, no ejecutamos el código y seguimos con la ejecución de nuestro programa



Estructuras de control

IF

```
int a = 16;  
int b = 2;  
if (b%a == 0){  
    System.out.println("resto 0");  
}
```

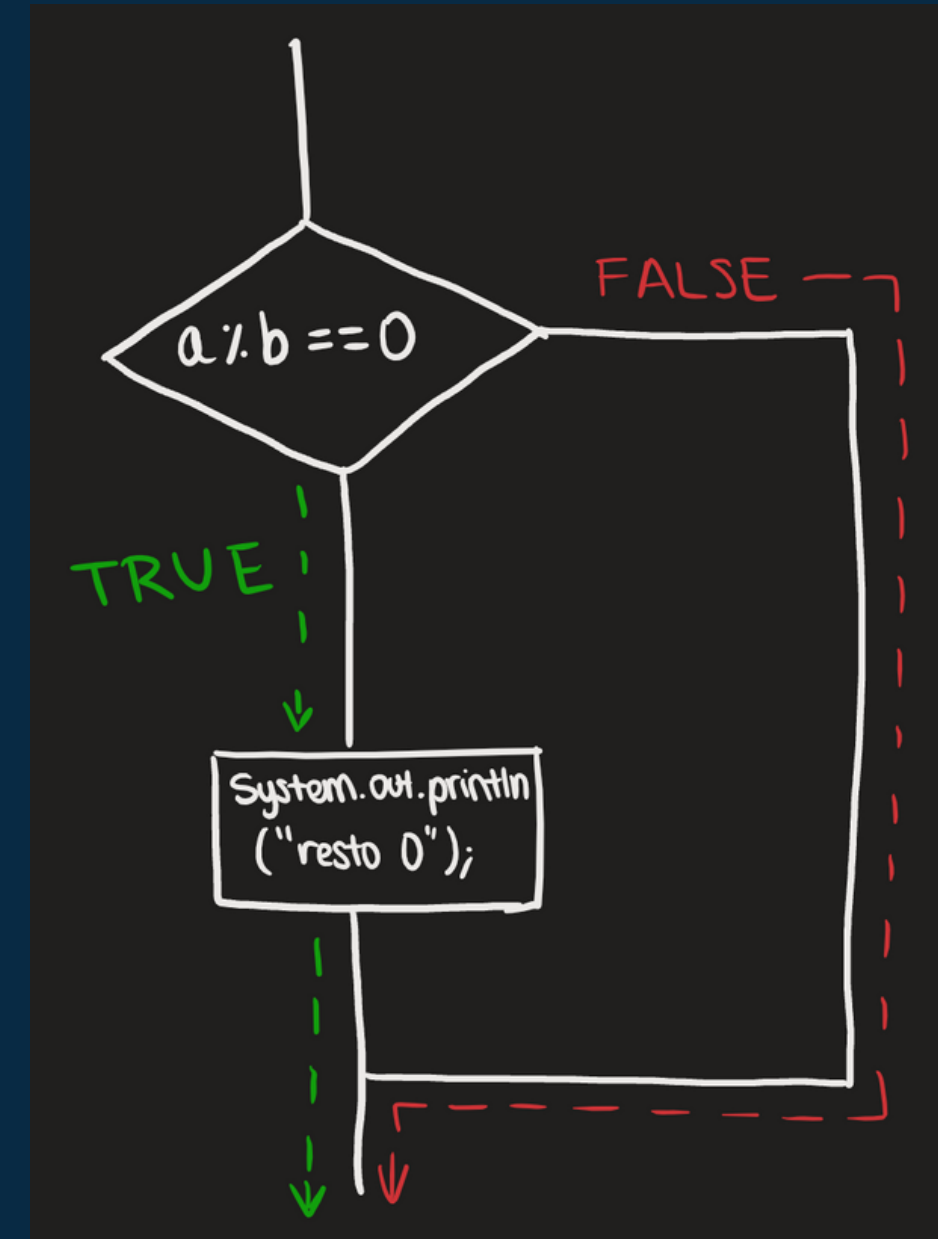
¿Cómo sería el diagrama de flujo?

Estructuras de control

IF

```
int a = 16;  
int b = 2;  
if (a % b == 0){  
    System.out.println("resto 0");  
}
```

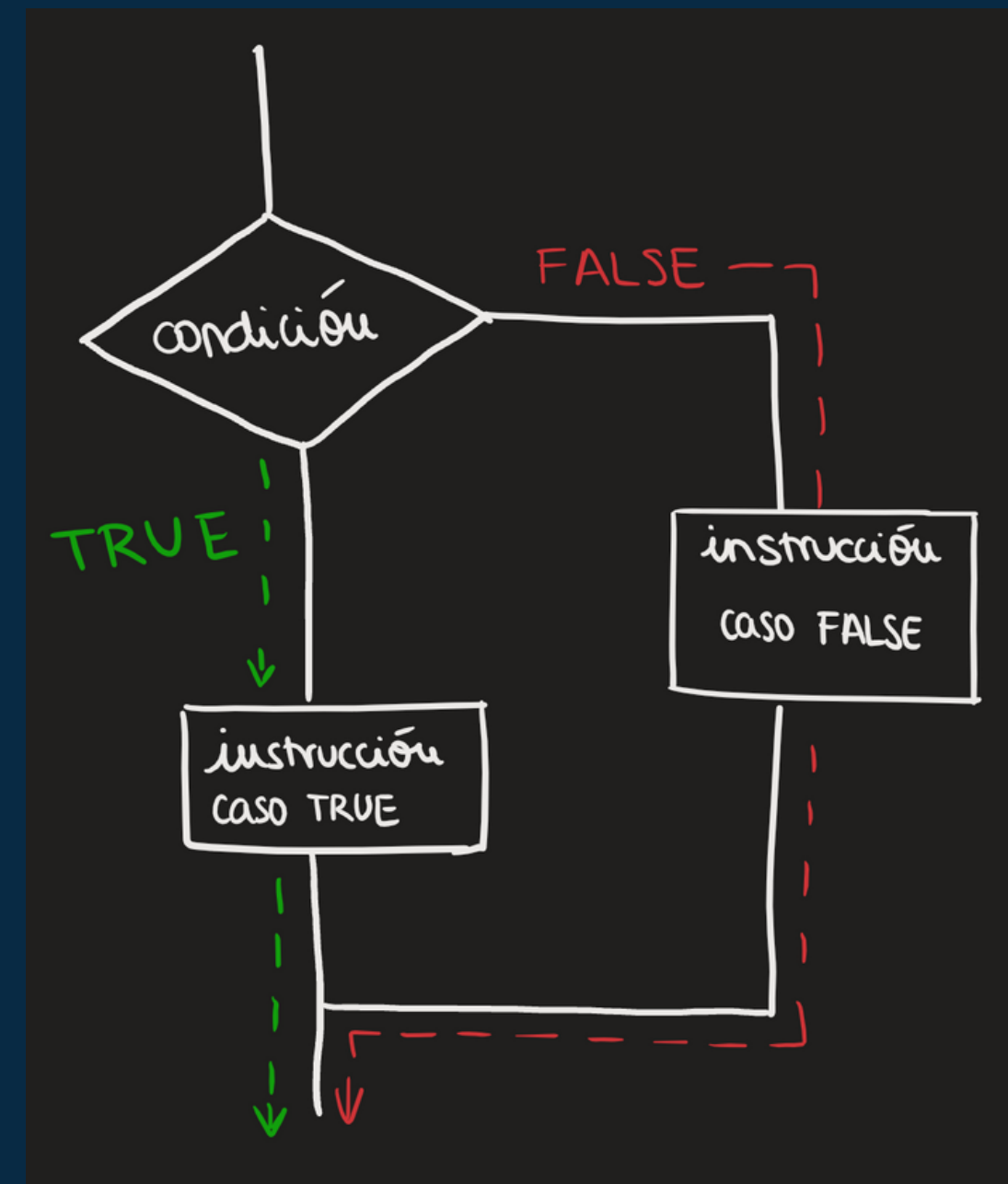
¿y si queremos hacer algo si no se cumple la condición?



Estructuras de control

IF-ELSE

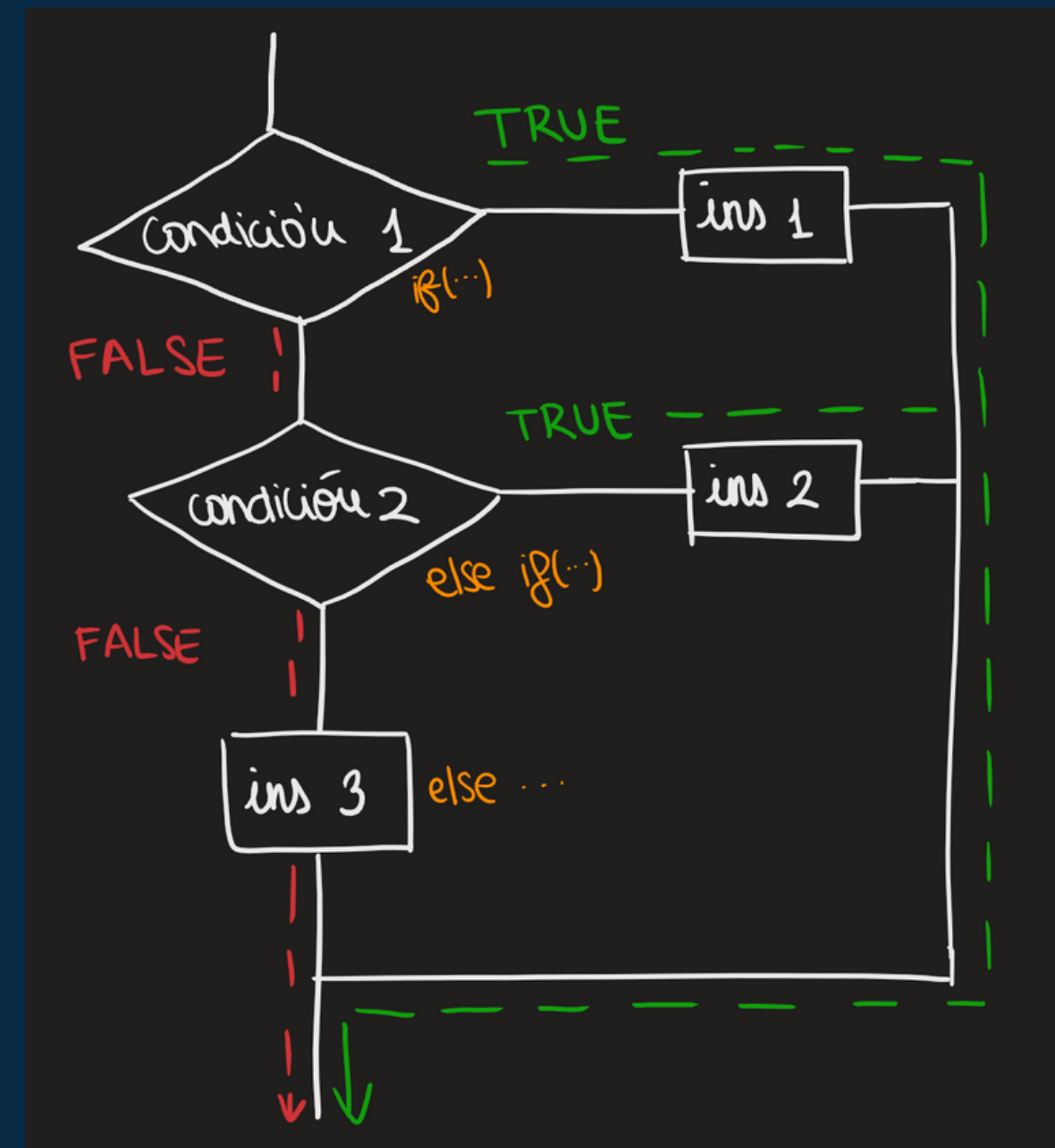
```
int a = 16;  
int b = 2;  
int c = 15;  
if (c%b == 0){  
    System.out.println("es par");  
}else{  
    System.out.println("es impar");  
}
```



Estructuras de control

IF-ELSE

```
int a = ¿0,1,2?  
if (a==0){  
    a++;  
}else if(a==1){  
    a--;  
}else{  
    a=a*a;  
}
```



Bucles

WHILE

evaluamos una expresión, si esta se cumple, ejecutamos el código de dentro del bucle hasta que esta condición se deje de cumplir

```
mientras(condicion==TRUE){  
    ejecuto instrucciones  
}
```

puede que el código del interior del bucle no se ejecute NI UNA SOLA VEZ, si la condición que se evalúa no es cierta.

Bucles

WHILE

```
mientras(condicion==TRUE){  
    ejecuto instrucciones  
}
```

```
int i = 0;  
while (i < 5) {  
    System.out.println(i);  
    i++;  
}
```

si no modifico el valor de i, la condición
SIEMPRE será cierta porque SIEMPRE
será menor que 5
BUCLE INFINITO



Bucles

DO-WHILE

la condición está al final, hacemos el código dentro del bucle hasta que se deje de cumplir la condición

```
haz{  
    ejecuto instrucciones  
}mientras(condicion==TRUE)
```

el código del bucle se va a ejecutar MÍNIMO UNA VEZ, en el caso en el que la condición del final no se cumpla, pero claro, lo de dentro del do está hecho

Bucles

DO-WHILE

```
haz{  
    ejecuto instrucciones  
}mientras(condicion==TRUE)
```

```
int i = 0;  
do{  
    System.out.println(i);  
    i++;  
}while (i < 5);
```

```
i=0  
imprimo >> 0  
i++ -> i=1  
1<5? SÍ
```

```
i=1  
imprimo >> 1  
i++ -> i=2  
2<5? SÍ
```

```
i=2  
imprimo >> 2  
i++ -> i=3  
3<5? SÍ
```

```
i=3  
imprimo >> 3  
i++ -> i=4  
4<5? SÍ
```

```
i=4  
imprimo >> 4  
i++ -> i=5  
5<5? NO
```

SALGO!!!

Bucles

FOR

"sabemos" las veces que queremos que se ejecute este tipo de bucle, se ejecutarán 3 sentencias dentro de este bucle

```
for(sentencia1; sen2; sen3){  
    ejecuto instrucciones  
}
```

sentencia 1: se ejecuta 1 vez, damos valor inicial al iterador
sentencia 2: definimos la condición para la terminación del bucle
sentencia 3: se ejecuta en todas las iteraciones, modificamos iterador



A red signature or scribble in the bottom right corner.

Bucles

FOR

```
for(sentencia1; sen2; sen3){  
    ejecuto instrucciones  
}
```

```
for (int i = 0; i < 5; i++) {  
    System.out.println(i);  
}
```

i=0 imprimo i >> 0 i++ -> i=1 1<5? SÍ	i=3 imprimo i >> 3 i++ -> i=4 4<5? SÍ
-----	-----
i=1 imprimo i >> 1 i++ -> i=2 2<5? SÍ	i=4 imprimo i >> 4 i++ -> i=5 5<5? NO

i=2 imprimo i >> 2 i++ -> i=3 3<5? SÍ	SALGO!!!

Bucles

FOR

recorrido de vectores

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

```
vector[0]=0;
```

```
vector[1]=1;
```

```
vector[2]=2;
```

```
vector[3]=3;
```

```
...
```

```
vector = new int[5]{0,1,2,3,4};
```

```
for (int i = 0; i < vector.length; i++) {  
    System.out.println(i);  
    System.out.println(vector[i]);  
}
```

Métodos

procedimientos y funciones

procedimiento

mandamos a Java que haga algo, imprime esto!

No estamos devolviendo un valor

ejemplo: queremos imprimir las coordenadas de un punto (x,y) en ese formato

```
public static void printCoord(int x, int y){  
    System.out.println("(" + x + ", " + y + ")");  
}
```

función

¿cómo es una función matemática?

$F(x)=3x$ equivalente a $y=3x$

-parámetro x

-operamos con x

-devolvemos el resultado de la función como un nuevo valor y

este valor que se devuelve no se imprime, tenemos que almacenarlo en una variable

```
public static int funcionF(int x){  
    y=3*x;  
    return(y);  
}
```

Excepciones

cuando ocurre un error java se detendrá y mandará un mensaje de error, es decir, una excepción. Estos errores pueden ser controlados con el tratamiento de excepciones y así seguir con la ejecución del código

try: nos permite definir el "bloque de pruebas", será el fragmento de código que va a ser controlado en caso de que ocurra un fallo en él.

catch: definiremos un bloque de código que se llevará a cabo si se ha dado la excepción que capturamos.

```
try{  
    instrucciones que controlamos  
}catch(Exception e){  
    ejecuto en caso de error  
}
```

Excepciones

```
vector = new int[5];  
vector[0]=0;  
vector[1]=1;  
vector[2]=2;  
vector[3]=3;  
...  
vector = new int[5]{0,1,2,3,4};
```

```
System.out.println(vector[10]);
```

ESTO DARÁ ERROR, NUESTRO VECTOR ES DE TAMAÑO 5

>>Exception in thread "main"

java.lang.ArrayIndexOutOfBoundsException: 10

```
try{  
    System.out.println(vector[10]);  
}catch(Exception e){  
    System.out.println("ha ocurrido un error.");  
}
```

>>Ha ocurrido un error.



Excepciones

si añadimos a parte de try catch, la sentencia finally el código del interior de finally se ejecutará al final del try, se haya dado un error o no.

```
try{
    System.out.println(vector[10]);
}catch(Exception e){
    System.out.println("ha ocurrido un error.");
}finally{
    System.out.println(vector[0]);
}
>>
Ha ocurrido un error.
0
```

```
vector = new int[5];
vector[0]=0;
vector[1]=1;
vector[2]=2;
vector[3]=3;
...
vector = new int[5]{0,1,2,3,4};
```

```
e.getMessage();
e.toString();
e.printStackTrace();
```



Ficheros

```
String s = "Primera linea de mi fichero";  
try{  
    PrintWriter fichero = new PrintWriter ("fich.txt");  
    fichero.print(s);  
  
    fichero.close();  
}catch(Exception e){  
    e.printStackTrace();  
}
```

el fichero fich.txt tendrá: Primera linea de mi fichero



vamos comentando
el código...
Ejemplo del tema 3 de
FPROG!

Ficheros

```
Scanner inicial;  
PrintWriter final;
```

```
try{  
    inicial = new Scanner (new File ("fichero_inicial.txt"));  
}catch(Exception e){  
    e.printStackTrace();  
}  
try{  
    final = new PrintWriter ("resultado.txt");  
}catch(Exception ex){  
    ex.printStackTrace();  
    inicial.close();  
}  
int numero;  
while(inicial.hasNextInt()){  
    numero = inicial.nextInt();  
    final.println(numero);  
}  
inicial.close();  
final.close();
```

Clases

EJEMPLO:

Tenemos un vehículo, el cual
tiene ciertas características
que le definen



Clases

EJEMPLO:

Tenemos un vehículo, el cual tiene ciertas características que le definen

el número de ruedas es una característica de los vehículos

el color del vehículo es otra característica

Lo llamaremos: ATRIBUTOS








Clases

EJEMPLO:

```
public class Vehiculo{  
    //atributos o variables de instancia  
    int numeroRuedas;  
    String color;  
}
```



Clases

EJEMPLO:

Si estamos “fabricando” un vehículo tendremos que definir tanto el número de ruedas como el color



color rojo

dos ruedas

Clases

EJEMPLO:

Si estamos "fabricando" un vehículo tendremos que definir tanto el número de ruedas como el color

Pero inicialmente como lo estamos "construyendo" tendrá 0 ruedas y estará sin pintar



cero ruedas

Clases

EJEMPLO:

```
public class Vehiculo{  
    //atributos o variables de instancia  
    int numeroRuedas;  
    String color;  
  
    public Vehiculo(){  
        //constructor  
        this.numeroRuedas=0;  
        this.color="";  
    }  
}
```

Clases

EJEMPLO:

Ya tengo como fabricar un
vehículo para los valores
iniciales de éste.

Ahora quiero fabricar la moto que tenía
en un principio, o el coche, o la bici...

Para cualquier número de ruedas
Para cualquier color



4

gris



2

blanco



2

rojo

Clases

EJEMPLO:

```
public class Vehiculo{  
    //atributos o variables de instancia  
    int numeroRuedas;  
    String color;  
  
    public Vehiculo(){  
        //constructor  
        this.numeroRuedas=0;  
        this.color="";  
    }  
    public Vehiculo(int num, String col){  
        this.numeroRuedas=num;  
        this.color=col;  
    }  
}
```

Clases

EJEMPLO:

```
public class Vehiculo{
    //atributos o variables de instancia
    int numeroRuedas;
    String color;

    public Vehiculo(){
        //constructor
        this.numeroRuedas=0;
        this.color="";
    }
    public Vehiculo(int num, String col){
        //constructor sobrecargado
        this.numeroRuedas=num;
        this.color=col;
    }
}
```

/*en nuestro main queremos crear:
* el coche la moto y la bici*/

```
Vehiculo coche = new Vehiculo();
//en este punto tenemos el vehiculo con 0 ruedas y color ""
coche.numeroRuedas = 4;
coche.color = "gris";
//ahora tenemos el coche que queremos
```

```
Vehiculo moto = new Vehiculo();
moto.numeroRuedas = 2;
moto.color = "rojo";
```

```
Vehiculo bici = new Vehiculo();
bici.numeroRuedas = 2;
bici.color = "blanco";
```


Clases

EJEMPLO:

Ya tengo todos los vehículos
que quería pero y si quiero
ver por ejemplo su velocidad
máxima?

tendré que ponerla como una
característica más de los vehículos

y crear un método para poder ver la
velocidad



Clases

EJEMPLO:

```
public class Vehiculo{
    //atributos o variables de instancia
    int numeroRuedas;
    String color;
    int velMax;

    public Vehiculo(){
        //constructor
        this.numeroRuedas=0;
        this.color="";
        this.velMax = 0;
    }
    public Vehiculo(int num, String col, int vel){
        //constructor sobrecargado
        this.numeroRuedas=num;
        this.color=col;
        this.velMax=vel;
    }
    public static void imprimeVelocidad(Vehiculo v){
        System.out.println(v.velMax);
    }
}
```

```
/*en nuestro main*/
```

```
Vehiculo coche = new Vehiculo();
//en este punto tenemos el vehiculo con 0 ruedas y color ""
coche.numeroRuedas = 4;
coche.color = "gris";
coche.velMax=260;
//ahora tenemos el coche que queremos su velocidad max
```

```
imprimeVelocidad(coche);
>>260
```

Buenas y malas prácticas



LEGIBILIDAD,
ESTRUCTURA Y
SIMPLICIDAD



COMENTARIOS



PRUEBA TU CÓDIGO



NO DUPLIQUES
CÓDIGO



NOMBRA
CORRECTAMENTE
LAS VARIABLES Y
MÉTODOS



ANTES DE PROGRAMAR
HAZ UN ESQUEMA DE
LO QUE QUIERES Y
NECESITAS

¡Gracias por vuestra atención!

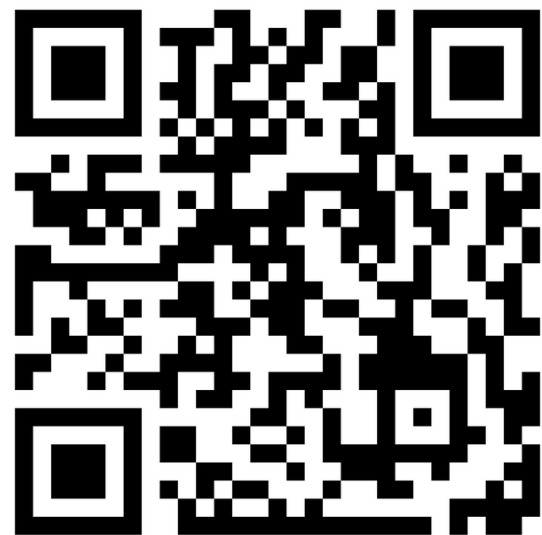
No dudéis en contactar conmigo.

TELEGRAM

aleja_ren

CORREO

alejandra.gavino-dias@estudiantes.uva.es



Github con el código

Para almacenar los NIF de los usuarios de un determinado sistema se utilizan registros de tipo regNIF, que contienen tres campos:

1. de tipo long para almacenar el número del DNI.
2. de tipo int que guarda el número de dígitos significativos de ese DNI .
3. de tipo char que almacena la letra de ese DNI.

Por ejemplo, la cadena “07123456L”, que representa un NIF correcto, se almacenaría en un registro de tipo regNIF guardando:

- 7123456 en el campo de tipo long
- 7 en el campo de tipo int
- ‘L’ en el campo de tipo char.

Se pide:

1)definir regNIF

2)Elaborar un método Java con las siguientes características:

- Parámetros de entrada: cadena de caracteres.
- Valor devuelto: el registro regNIF correspondiente.
- Precondición: la cadena de entrada se corresponde con un NIF correcto
(contiene 9 caracteres, de los cuales los 8 primeros son dígitos y el último una letra mayúscula del alfabeto inglés).

1) definir regNIF

```
public class regNIF{
    long num;
    int dig;
    char letra;

    //constructor
    public regNIF(){
        this.num=0;
        this.dig=0;
        this.letra=' ';
    }

    public regNIF(long n, int d, char l){
        this.num=n;
        this.dig=d;
        this.letra=l;
    }
}
```

1) definir el método

```
public static RegNIF dni (String cadena){
    int dig, indice = 0;
    long num=0, potencia=1;

    while(cadena.CharAt(indice) != 0)
        indice++;
    dig = 8-indice;
    for (int i=7; i>=indice; i--){
        num = num + (cadena.CharAt(i)-'0')*potencia;
        potencia = potencia * 10;
    }
    RegNIF dni = new RegNIF (num, dig, cadena.CharAt(8));
    return dni;
}
```