

# UT4

## Interacción de objetos. Estructura de control iterativa.

Módulo – Programación (1º)

Ciclos – Desarrollo de Aplicaciones Multiplataforma | Desarrollo de Aplicaciones Web

CI María Ana Sanz

---

# Contenidos

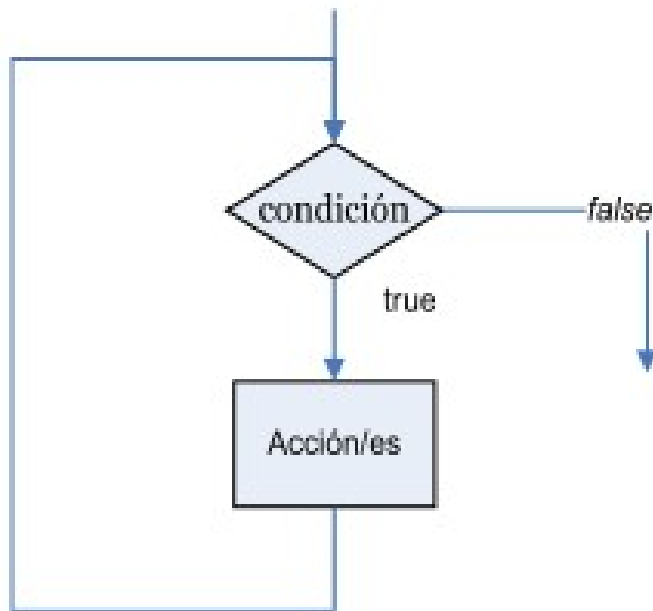
- Abstracción y modularización
- Las clases como tipos
- Diagrama de clases y diagrama de objetos
- Tipos primitivos y tipos referencia
- Creación de nuevos objetos
- Múltiples constructores
- Llamadas a métodos
  - Llamadas internas
  - Llamadas externas
- public / private
- this / null
- Cómo usar Java fuera de BlueJ
  - el método *main()*
- Estructura de control iterativa
  - while
  - for
  - do .. while
- clases Scanner y Random
- Miembros de clase (static)
- Métodos recursivos

# **Estructuras de control iterativas**

# Estructuras de control iterativas

- Sentencias que hemos utilizado hasta ahora
  - escritura - *System.out.println()*
  - asignación - *variable = valor*
  - llamadas a métodos - *minutos.incrementar()*
  - sentencias condicionales - *if / switch*
- Y si queremos repetir un nº determinado o indeterminado de veces una o varias instrucciones?
  - utilizaremos las estructuras de control iterativas para expresar los bucles o iteraciones
  - Java tiene tres estructuras repetitivas
    - **while**
    - **for**
    - **do ... while**

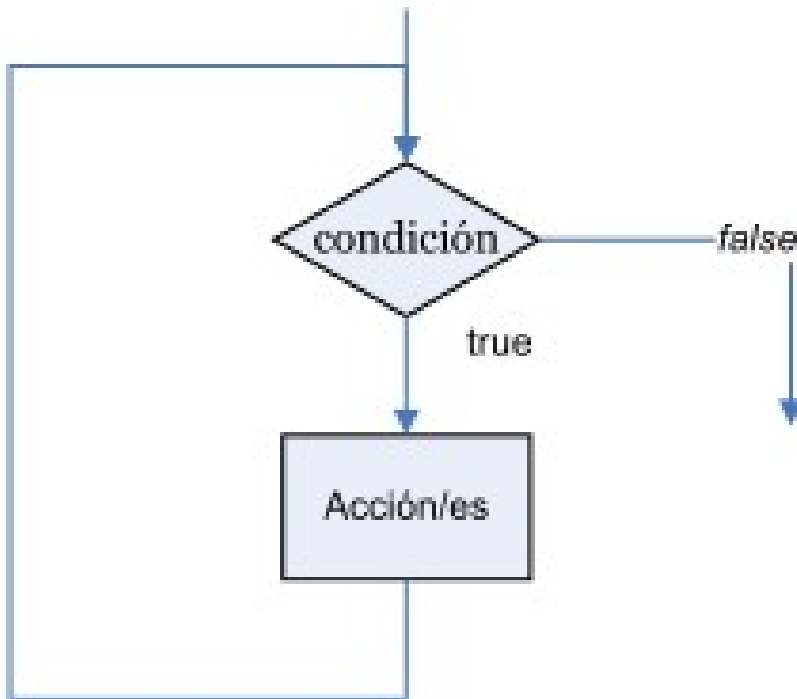
# Sentencia while



```
while (condición)
{
    acción/es a ejecutar
}
```

**Condición** - expresión booleana

# Sentencia while




- la condición se evalúa en primer lugar
- si es cierta se ejecutan las instrucciones dentro del cuerpo del while y se vuelve a evaluar la condición
- si es falsa el bucle termina
- el bucle puede no ejecutarse nunca
- las acciones se repiten un nº determinado o indeterminado de veces
- en el cuerpo del while ha de haber una sentencia que modifique la condición para que el bucle pueda terminar

# Ejemplos while

```
public void ejemplo01()
{
    int veces = 1;
    while (veces <= 10) {
        System.out.println("Saludo: " + veces);
        veces++;
    }
}
```

**variable de control**  
del bucle, actúa  
como contador



```
public void ejemplo02()
{
    int tiradas = 1;
    while (tiradas <= 10) {
        moneda.tirar();
        tiradas++;
    }
}
```

# Ejemplos while

```
public void ejemplo03()
{
    int caras = 0; // es una variable contador
    int cruz = 0; // es una variable contador
    int tiradas = 1; // es una variable contador
    while (tiradas <= 30) {
        moneda.tirar();
        if (moneda.esCara()) {
            caras++;
        }
        else {
            cruz++;
        }
        tiradas++;
    }
    System.out.println("Ha salido cara " + caras + " veces");
    System.out.println("Ha salido cruz " + cruz + " veces");
}
```

contador del nº de  
veces que salió cara  
y cruz

variable de control  
del bucle, actúa  
como contador



# Variables contadores

## ■ Contador

- variable cuyo valor se incrementa en una cantidad fija, positiva o negativa.
  - para contabilizar el número de veces que es necesario realizar una acción (variable de control de un bucle). Ej. `tiradas++`; `contador--`;
  - para contar un suceso particular (asociado o no a un bucle). Ej. `caras++`; `cruz++`;

## ■ Ejer (en papel)

- `reloj.emitirTic()`;
  - emitir 30 ticks de reloj
- `cuenta.ingresar(10)` ; // ingresa 10 euros en una cuenta
  - ingresar 20 veces 10 euros en la cuenta

# Variables contadores

```
int cuantosTics = 1;
while (cuantosTics <= 30) {
    reloj.emitirTic();
    cuantosTics++;
}
```

```
int numeroIngresos = 1;
while (numeroIngresos <= 20) {
    cuenta.ingresar(10);
    numeroIngresos++;
}
```

# Traza de un algoritmo – Ejecución paso a paso

---

- Hacer la traza de un algoritmo es seguir la ejecución de ese algoritmo paso a paso para valores concretos de las variables que intervienen en él.
- Una traza permita comprobar si un algoritmo es incorrecto (no si es correcto, ya que para ello habría que hacer la traza con multitud de valores) .
- Desde un IDE hacer la traza es realizar una ejecución paso a paso estableciendo previamente **puntos de ruptura**

# Traza de un algoritmo – Ejecución paso a paso

- Traza del ejemplo03 (contar cara y cruz)

Traza en papel					
tiradas		cara	cruz		esCara()
1		0	0	1	true
2		1	1		false
3					true
4					
5					
6					
7					
8					
9					
10		2			
11					

# Traza de un algoritmo – Ejecución paso a paso

- Traza en papel del siguiente trozo de código

```
int suma = 0;  
int conta = 1;  
while (conta <= 5) {  
    suma += conta;  
    conta++;  
}  
System.out.println(" " + suma);
```

conta	suma	conta <= 5	Pantalla
	0		
1	0+1	(1<= 5) true	
2	0+1+2	(2<= 5) true	
3	0+1+2+3	(3<= 5) true	
4	0+1+2+3+4	(4<= 5) true	
5	0+1+2+3+4+5	(5<= 5) true	
6		(6<= 5) false	15

**Acumulador**



# Traza de un algoritmo – Ejecución paso a paso

- Traza en papel del siguiente trozo de código

```
public int queHace01(int a, int b)
{
    int p = 1;
    int contador = 1;
    while (contador <= b) {
        p = p * a;
        contador ++;
    }
    return p;
}
```

```
public int queHace02(int a, int b)
{
    int s = 0;
    int contador = 1;
    while (contador <= b) {
        s = s + a;
        contador ++;
    }
    return s;
}
```

# Variables acumuladores

```
public void ejemplo04()
{
    int suma = 0; // variable acumulador
    long producto = 1; // variable acumulador
    int contador = 20; // variable contador de control del bucle
    while (contador >= 1) {
        suma = suma + contador;
        producto = producto * contador;
        contador--;
    }
    System.out.println("La suma vale " + suma);
    System.out.println("El producto vale " + producto);
}
```

**Acumulador** - variable cuyo valor se incrementa sucesivas veces en cantidades variables.

- para obtener un total acumulado de un conjunto de cantidades - inicializar el acumulador a 0. *Ej.  $\text{suma} = \text{suma} + \text{contador}$*
- para obtener un total como producto de distintas cantidades - inicializar a 1  
*Ej.  $\text{producto} = \text{producto} * \text{contador}$ ;*

# Ejemplos para hacer

- **public void ejemploUno(int n)**
  - generar n aleatorios entre 5 y 20 (con Math.random())
  - cuenta los pares
  - sumar los impares
  - multiplicar los que acaban en 2
  - mostrar todos los resultados con println()



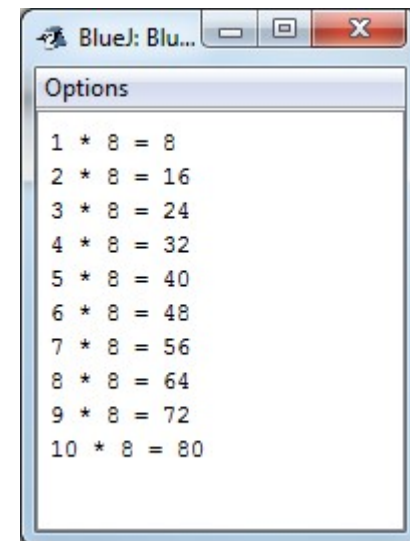
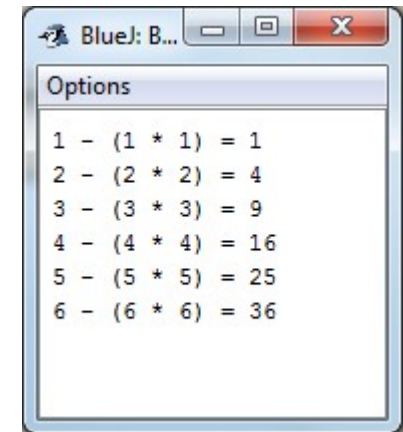
## Ejer. 4.14 (en papel)

---

- a) `public void escribirNumeros(int desde, int hasta)`
- b) `public void mostrarPares()`
- d) `public double sumarSerie(int n)`
- f) `public int sumarDivisores(int numero)`
- g) `public int sumarDigitos(int numero)`


## Ejer. 4.14 (en papel)

- `public void tablaCuadrados(int n)`
  - escribe cada  $n^{\circ}$  y su cuadrado, desde el 1 hasta  $n$
- `public void tablaMultiplicar(int numero)`
- `public int contarDivisores(int numero)`



## Ejer. 4.14 a)

```
public void escribirNumeros(int desde, int hasta)
{
    int columna = 0;
    int conta = desde;
    while (conta <= hasta) {
        System.out.print(conta + "\t");
        columna++;
        if (columna == 5) {
            System.out.println();
            columna = 0;
        }
        conta++;
    }
}
```



**Escribimos así los  
números en filas de 5 en 5**

## Ejer. 4.14 b)

```
public void mostrarPares()
{
    int numero = 2;
    while (numero <= 50) {
        System.out.print(numero + "\t");
        numero += 2;
    }
}
```

## Ejer. 4.14 d)

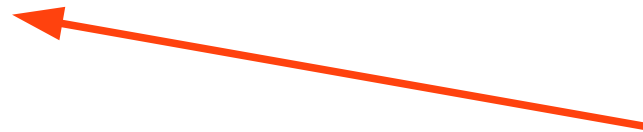
```
public double sumarSerie(int n)
{
    double serie = 0;
    int conta = 1;
    while (conta <= n) {
        serie = serie + (double) 1 / conta;
        conta++;
    }
    return serie;
}
```

## Ejer. 4.14 f)

```
public int sumarDivisores(int numero)
{
    int mitad = numero / 2;
    int sumaDivisores = 1;
    int divisor = 2;
    while (divisor <= mitad) {
        if (numero % divisor == 0) {
            sumaDivisores += divisor;
        }
        divisor++;
    }
    sumaDivisores += numero;
    return sumaDivisores;
}
```

## Ejer. 4.14 g)

```
public int sumarDigitos(int numero)
{
    int cifra;
    int sumaDigitos = 0;
    int auxNumero = numero;
    while (auxNumero != 0) {
        cifra = auxNumero % 10;
        sumaDigitos += cifra;
        auxNumero = auxNumero / 10;
    }
    return Math.abs(sumaDigitos);
}
```



**Por si se introduce un n°  
negativo**

# Ejemplos para hacer en casa

---

- **public void ejemploDos()**
  - generar aleatorios entre 0 y 100
  - parar cuando salga el 0
  - sumar los impares
  - multiplicar los pares
  - contar los múltiplos de 3 y de 6
  - contar los que acaban en 7
  - mostrar todos los resultados con println()



# Ejemplos para hacer en casa

---

- **public void ejemploTres()**
  - asumiendo que `miCirculo.dibujar(x, y)` dibuja un círculo centrado en el punto `x,y`
    - escribir un bucle que dibuje 10 círculos, el primero en el centro (0,0), el segundo en el centro (10, 10), el tercero en el centro (20, 20)
- **public void ejemploCuatro()**
  - generar aleatorios entre 1 y 30 hasta que salga el 5

# Ejemplos para hacer en casa

```
public void ejemploDos()
{
    int termina7 = 0;
    int multiplo3 = 0;
    int multiplo6 = 0;
    int sumaImpares = 0;
    int productoPares = 1;
    int aleatorio = (int) (Math.random() * 101);
    while (aleatorio != 0){
        System.out.print(aleatorio + "\t");
        if (aleatorio % 2 == 0) { // es par
            productoPares *= aleatorio;
            if (aleatorio % 6 == 0) {
                // múltiplo de 6 y de 3
                multiplo6++;
                multiplo3++;
            }
        }
    }
}
```

# Ejemplos para hacer en casa

```
else { // es impar

    sumaImpares += aleatorio;
    if (aleatorio % 3 == 0) { // múltiplo de 3
        multiplo3++;
    }
    if (aleatorio % 10 == 7) { // acaba en 7
        termina7++;
    }
    aleatorio = (int) (Math.random() * 101);
}
System.out.println("\nSuma de impares: " + sumaImpares);
System.out.println("Producto de pares: " + productoPares);
System.out.println("Múltiplos de 3: " + multiplo3);
System.out.println("Múltiplo6: " + multiplo6);
System.out.println("Acaban en 7: " + termina7);
}
```

# Ejemplos para hacer en casa

```
public void ejemploTres()
```

```
{
```

```
    int x = 0;
```

```
    int y = 0;
```

```
    int cuantos = 0;
```

```
    while (cuantos <= 10) {  
        miCirculo.dibujar(x, y);  
        x += 10;  
        y += 10;  
        cuantos++;  
    }
```

```
}
```

```
public void ejemploCuatro()
```

```
{
```

```
    int numero = (int) (Math.random() * 30) + 1;
```

```
    while (numero != 5) {
```

```
        numero = (int) (Math.random() * 30) + 1;
```

```
    }
```

```
}
```

# Variables switch o conmutadores (flags)

---

- Un **conmutador** o **switch (flag)** es una variable que
  - toma dos valores exclusivos ( 0/1, true/false, 1/-1, ...)
- Permite
  - ejecutar alternativamente dos grupos de sentencias dentro de un bucle o
  - recordar la ocurrencia o no de un suceso para salir de un bucle

# Variables switch o conmutadores

```
boolean salioCara = false; // es una variable que actúa como un switch o conmutador
int tiradas = 1; //es una variable contador
while (tiradas <= 30 && ! salioCara) {
    moneda.tirar();
    if (moneda.esCara()) {
        salioCara = true;
    }
    else {
        tiradas++;
    }
}
System.out.println("Salió cara en la tirada " + tiradas);
```

recordar la ocurrencia o no de un  
determinado suceso o para salir de un bucle

# Variables switch o conmutadores

```
/* Suma de pares entre 1 y 100
y producto de impares entre 1 y 100 */

int sumaPar = 0; // es un acumulador
int productoImpar = 1; //es un acumulador
boolean tocaImpar = true; // es un switch
int numero = 1; //es un contador
while (numero <= 100) {
    if (tocaImpar) {
        productoImpar *= numero;
    }
    else {
        sumaPar += numero;
    }
    tocaImpar = ! tocaImpar;
    numero++;
}
```

hacer que se ejecuten alternativamente dos grupos de sentencias dentro de un bucle

# Variables switch o conmutadores

## Traza

sumaPar	productoImpar	totalImpar	numero	numero <= 100
0	1	TRUE	1	1<=100 true
	1*1	FALSE	2	2<=100 true
0+2		TRUE	3	3<=100 true
	1*1*3	FALSE	4	4<=100 true
0+2+4		TRUE	5	5<=100 true
	1*1*5	FALSE	6	6<=100 true
0+2+4+6				
	1*1*5*...*99	FALSE	100	100<=100 true
0+2+4+6+...+100		TRUE	101	101<=100 false



# Ejercicios

---

- c) `public int generarAleatorios()`
- e) `public double sumarSerie(int n)`
- h) `public boolean esPrimo(int numero)`

## Ejer 4.14 c)

```
public int generarAleatorios()
{
    boolean salio99 = false;
    int cuantosDoce = 0;
    int aleatorio = (int) (Math.random() * 100 + 1);
    int i = 1;
    while (i <= 30 && aleatorio != 99) {
        System.out.print(String.format("%4d", aleatorio));
        if (i % 8 == 0) {
            System.out.println();
        }
        if (aleatorio == 12) {
            cuantosDoce++;
        }
        aleatorio = (int) (Math.random() * 100 + 1);
        i = i + 1;
    }
}
```

## Ejer 4.14 e)

```
public double sumarSerieConSigno(int n)
{
    boolean positivo = true;
    double serie = 0;
    int conta = 1;
    while (conta <= n) {
        if (positivo) {
            serie = serie + (double) 1 / conta;
        }
        else {
            serie = serie - (double) 1 / conta;
        }
        positivo = !positivo;
        conta++;
    }
    return serie;
}
```

**flag**



## Ejer 4.14 h)

```
public boolean esPrimoV1(int numero)
{
    return contarDivisores(numero) == 2;
}

private int contarDivisores(int numero)
{
    if (numero == 1) {
        return 1;
    }
    int divisores = 0;
    int divisor = 1;
    while (divisor <= numero) {
        if (numero % divisor == 0) {
            divisores++;
        }
        divisor++;
    }
    return divisores;
}
```

## Ejer 4.14 h)

```
public boolean esPrimoV2(int numero)
{
    return contarDivisores(numero) == 0;
}
```

```
private int contarDivisores(int numero)
{
    if (numero == 1) {
        return 1;
    }
    int divisores = 0;
    int divisor = 2;
    while (divisor <= numero / 2) {
        if (numero % divisor == 0) {
            divisores++;
        }
        divisor++;
    }
    return divisores;
}
```

## Ejer 4.14 h)

```
public boolean esPrimoV3(int numero)
{
    if (numero == 1) {
        return false;
    }
    boolean esPrimo = true;
    int divisor = 2;
    while (divisor <= numero / 2 && esPrimo) {
        if (numero % divisor == 0) {
            esPrimo = false;
        }
        else {
            divisor++;
        }
    }
    return esPrimo;
}
```

```
public boolean esPrimoV4(int numero)
{
    if (numero == 1) {
        return false;
    }
    int divisor = 2;
    while (divisor <= numero / 2) {
        if (numero % divisor == 0) {
            return false;
        }
        divisor++;
    }
    return true;
}
```

# Ejemplos bucles anidados

```
public void ejemploBucleAnidado()
{
    int cuantos = 1;
    while (cuantos <= 20) {
        int aleatorio = (int) (Math.random() * 5) + 5;
        int sumatorio = 0;
        int i = 1;
        while (i <= aleatorio) {
            sumatorio += i;
            i++;
        }
        System.out.println("El sumatorio de " + aleatorio + " es " + sumatorio);
        cuantos++;
    }
}
```

bucle interno

bucle externo

por cada valor de **cuantos** el bucle interno se ejecuta tantas veces como indique **aleatorio**

# Ejemplos bucles anidados - Traza

cuantos	cuantos<=20	aleatorio	sumatorio	i	i<=aleatorio
1	1<=20 true	5	0	1	1<=5 true
			0+1	2	2<=5 true
			0+1+2	3	3<=5 true
			0+1+2+3	4	4<=5 true
			0+1+2+3+4	5	5<=5 true
			0+1+2+3+4+5	6	6<=5 false
2	2<=20 true	4	0	1	1<=4 true
			0+1	2	2<=4 true
			0+1+2	3	3<=4 true
			0+1+2+3	4	4<=4 true
			0+1+2+3+4	4	5<=4 false



## Ejercicios bucles anidados – Ejer 4.15

---

- a) `public void escribirFigura(int n)`
- b) `public void escribirFigura(int n)`
- c) `public void escribirFigura(int n)`
- d) `public void escribirTablasMultiplicar(int numero)`
- e) `public void calcularSumatorios(int cuantos, int limite)`

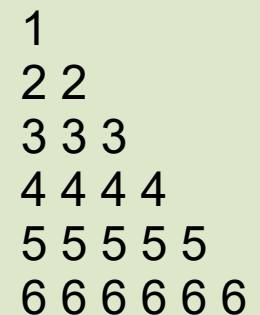
## Ejer 4.15 a)

```
public void escribirFigura(int n)
{
    int fila;
    int columna;
    fila = 1;
    while (fila <= n) {
        columna = 1;
        while (columna <= n) {
            System.out.print(String.format("%2d", fila));
            columna++;
        }
        System.out.println();
        fila++;
    }
}
```

```
1 1 1 1 1 1
2 2 2 2 2 2
3 3 3 3 3 3
4 4 4 4 4 4
5 5 5 5 5 5
6 6 6 6 6 6
```

## Ejer 4.15 b)

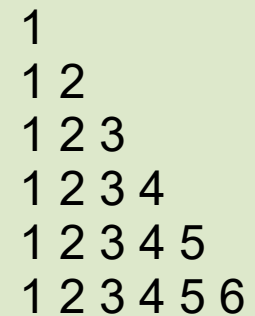
```
public void escribirFigura(int n)
{
    int fila;
    int columna;
    fila = 1;
    while (fila <= n) {
        columna = 1;
        while (columna <= fila) {
            System.out.print(String.format("%2d", fila));
            columna++;
        }
        fila++;
    }
}
```



```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
6 6 6 6 6 6
```

## Ejer 4.15 c)

```
public void escribirFigura(int n)
{
    int fila;
    int columna;
    fila = 1;
    while (fila <= n) {
        columna = 1;
        while (columna <= fila) {
            System.out.print(String.format("%2d", columna));
            columna++;
        }
        System.out.println();
        fila++;
    }
}
```



```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
```

## Ejer 4.15 d)

```
public void escribirTablaMultiplicar(int numero)
{
    int fila;
    int columna;
    int queNumero = 1;
    while (queNumero <= numero) {
        System.out.println("Tabla de multiplicar del " + queNumero);
        System.out.println("*****");
        fila = 1;
        while (fila <= 10) {
            System.out.println(queNumero + " * " +
                               fila + " = " + (queNumero * fila));
            fila++;
        }
        System.out.println("*****");
        queNumero++;
    }
}
```

## Ejer 4.15 e)

```
public void calcularSumatorios(int cuantos, int limite)
{
    int i, j;
    int numero, sumatorio;
    i = 1;
    while (i <= cuantos)
    {
        numero = (int) (Math.random() * limite + 1);
        sumatorio = 0;
        j = 1;
        while (j <= numero)
        {
            sumatorio += j;
            j++;
        }
        System.out.println("El sumatorio de " +
                           numero + " es " + sumatorio);
        i++;
    }
}
```

# Ejercicios adicionales

---

EJAD8 Dado / DemoDado

EJAD09 Interfaz Cuenta Bancaria

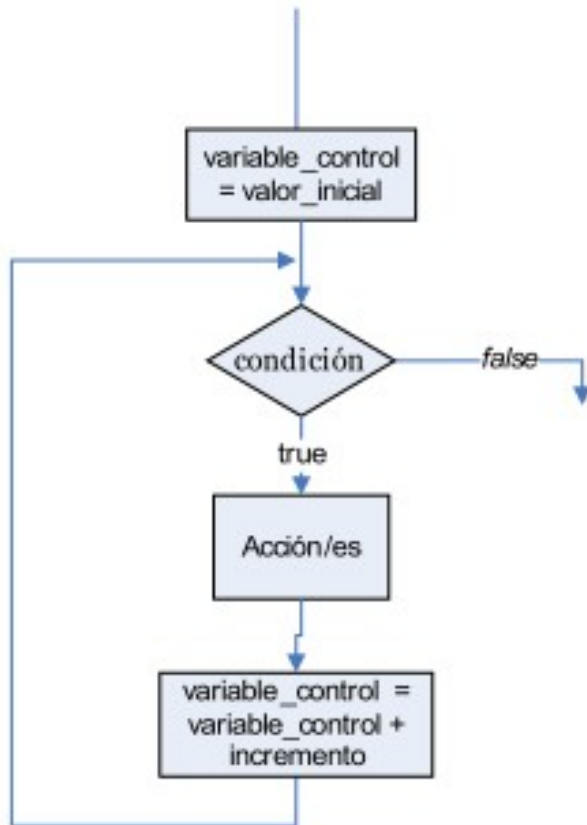
EJAD10 Fracción

EJAD11 Exponencial

EJAD12 Numero

EJAD13 Inversión

# Sentencia for

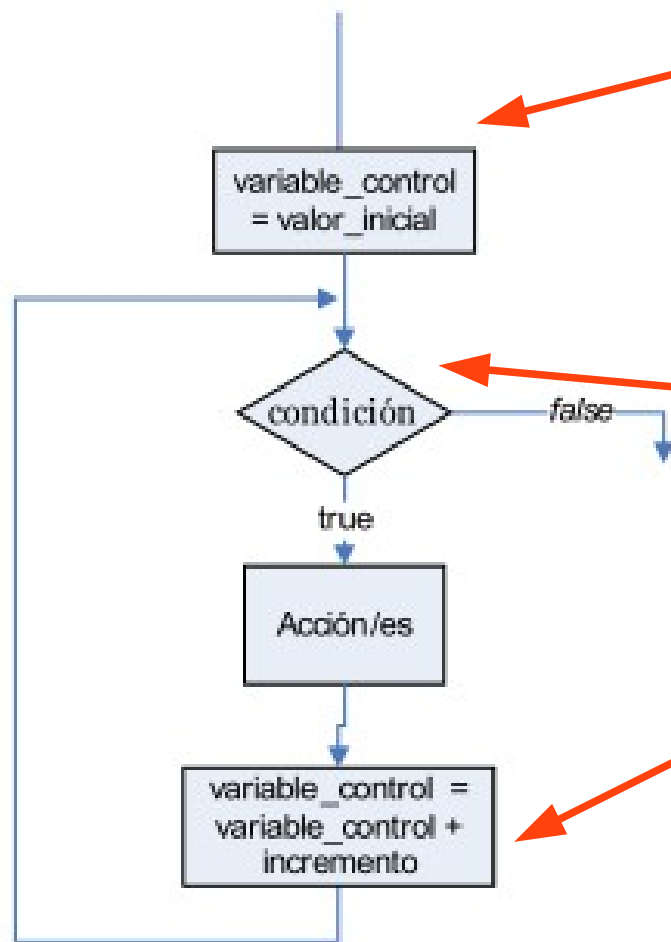


- Permite ejecutar un nº determinado de veces un conjunto de sentencias

```
for (inicialización; condición; incremento)
{
    acción/es a ejecutar
}
```



# Sentencia for



- **inicialización** - asignar un valor inicial a la variable de control del bucle (la que lleva la cuenta del nº de veces que se han de repetir las sentencias)
- **condición** – si se evalúa a `true` se ejecuta el bucle, si es `false` termina
- **incremento** (o decremento) – incrementa (decrementa) la variable de control para continuar la iteración

# Ejemplos for

```
for (int veces = 1; veces <= 10; veces++) {  
    System.out.println("Saludo " + veces);  
}
```

```
for (int tiradas = 1; tiradas <= 30; tiradas++) {  
    moneda.tirar();  
}
```

```
int ncaras = 0;  
int ncruz = 0;  
for (int tiradas = 1; tiradas <= 30; tiradas++) {  
    moneda.tirar();  
    if (moneda.esCara()) {  
        ncaras++;  
    }  
    else {  
        ncruz++;  
    }  
}
```

# Ejercicios for

---

- a) `public void contarParesImpares()`
- b) `public int maximo(int cuantos)`
- d) `public void escribirEstadisticas()`

## Ejer 4.16 a)

```
public void contarParesImpares()
{
    int pares = 0;
    int impares = 0;
    for (int contador = 1; contador <= 20; contador++) {
        int aleatorio = (int) (Math.random() * 50) + 1;
        if (aleatorio % 2 == 0) {
            pares ++;
        }
        else {
            impares ++;
        }
    }

    System.out.println("Pares: " + pares + "\nImpares: " + impares);
}
```

## Ejer 4.16 b)

```
public int maximo(int cuantos)
{
    int maximo = 0;
    for (int contador = 1; contador <= cuantos; contador++) {
        int aleatorio = (int) (Math.random() * 100) + 1;
        if (aleatorio > maximo) {
            maximo = aleatorio;
        }
    }

    return maximo;
}
```

## Ejer 4.16 )

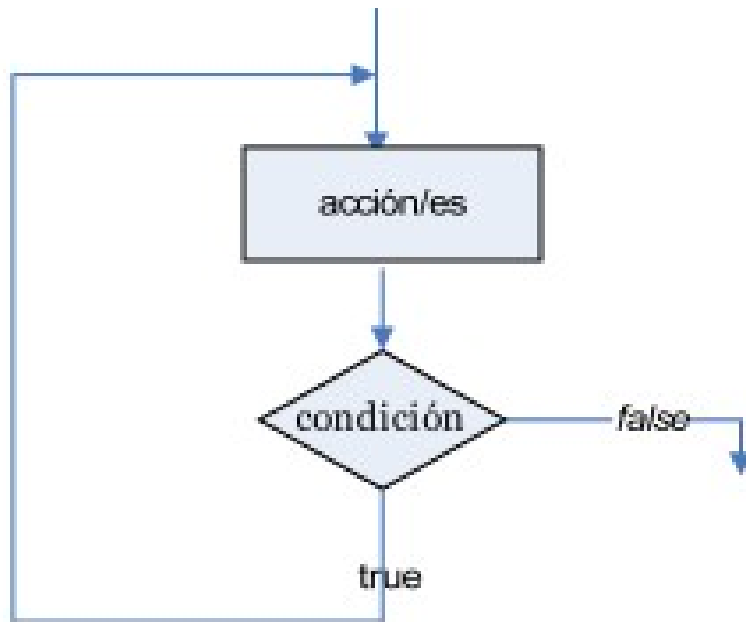
```
public void escribirEstadisticas()
{
    final int TOTAL = 30;
    int suma = 0;
    int maxima = 0;
    int minima = 10;
    int cuantasMaximas = 0;
    int cuantasMinimas = 0;
    for (int contador = 1; contador <= TOTAL; contador++) {
        int nota = (int) (Math.random() * 10) + 1;
        suma += nota;
        if (nota > maxima) {
            maxima = nota;
            cuantasMaximas = 1;
        }
        else if (nota == maxima) {
            cuantasMaximas ++;
        }
    }
}
```

## Ejer 4.16 )

```
        if (nota < minima)    {
            minima = nota;
            cuantasMinimas = 1;
        }
        else if (nota == minima) {
            cuantasMinimas ++;
        }
    }

    System.out.println("Media: " + suma / TOTAL);
    System.out.println("Máxima: " + maxima);
    System.out.println("Total máximas: " + cuantasMaximas);
    System.out.println("Mínima: " + minima);
    System.out.println("Total mínimas: " + cuantasMinimas);
}
```

# Sentencia do ... while



- variación del bucle while
- las sentencias del cuerpo del bucle se ejecutan al menos una vez.

```
do  
{  
    acción/es a ejecutar  
}  
while (condición)
```



# Sentencia do ... while

```
int factorial = 1;
int contador = 1;
do {
    factorial *= contador;
    contador++;
}
while (contador <= n);
return factorial;
```

```
int opcion;
do {
    System.out.println("Teclee opción: ");
    opcion = teclado.nextInt();
}
while (opcion < 1 || opcion > 6);
```

# Ejercicios adicionales

---

EJAD14 Numero (modificar)

EJAD15 Juego Número Secreto

EJAD16 Calculadora Fibonacci

Aula virtual – Tablero ajedrez (AVANZ)

Aula virtual – Círculos concéntricos (AVANZ)

Aula virtual – Figuras geométricas (AVANZ)

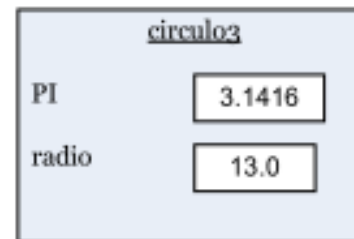
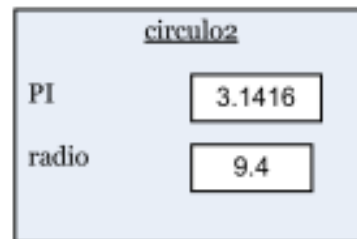
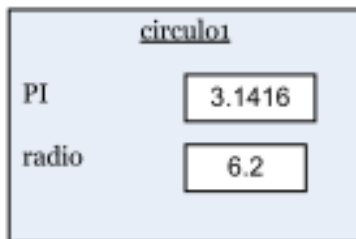
# Miembros de instancias

---

- Hasta ahora hemos visto
  - miembros de instancia, aquellos asociados a los objetos individuales (instancias de las clases)
    - variables de instancia (atributos)
    - constantes
    - métodos
  - cada objeto con sus propios valores y los métodos se invocan sobre los objetos

# Miembros de instancias

```
public class Circulo
{
    private final double PI = 3.1416;
    private double radio;
    .....
    public double getPerimetro() {
        return 2 * PI * radio;
    }
    .....
}
```



# Miembros de clase (miembros static)

---

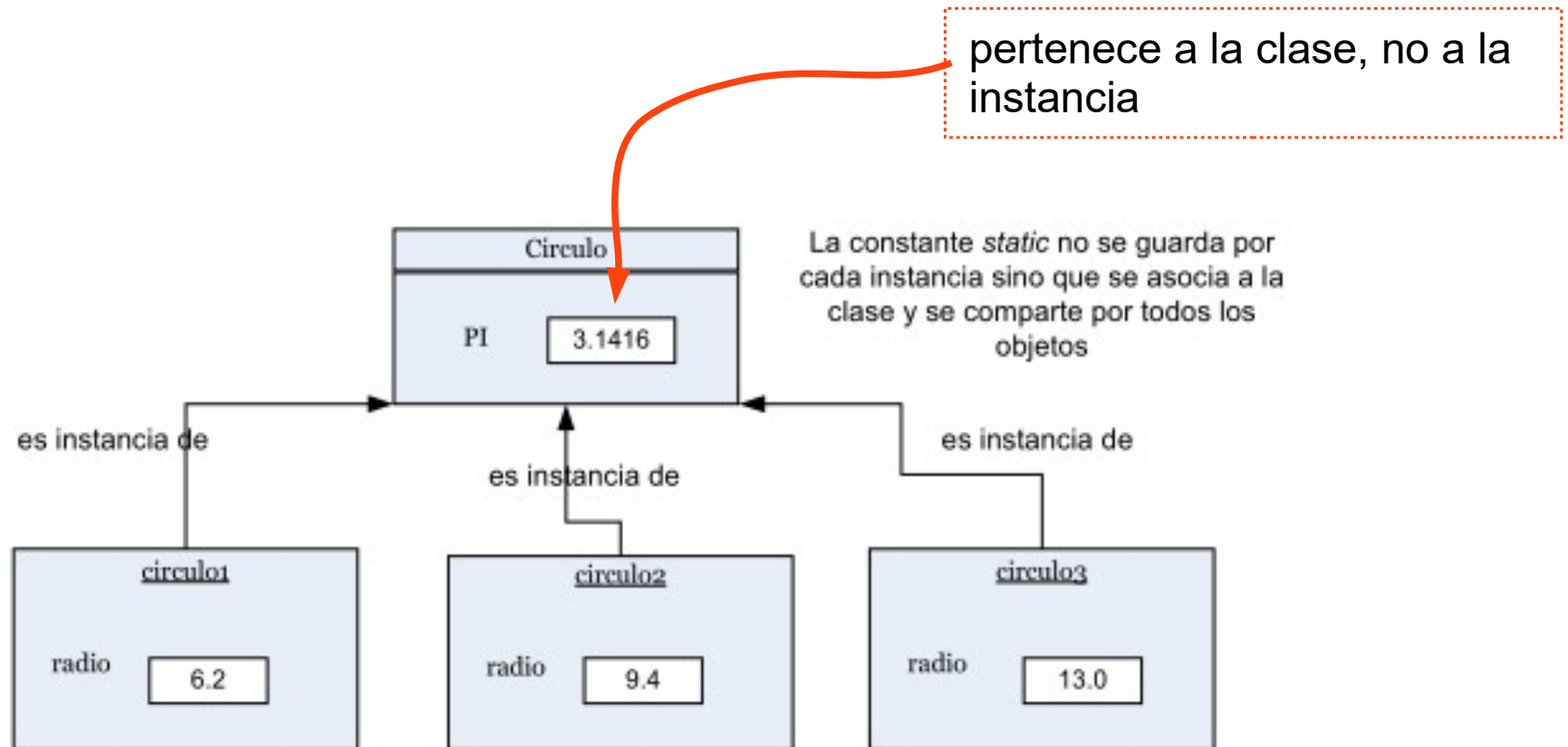
- Los **miembros de clase**
  - se denominan **static**
  - pertenecen a la clase, no a las instancias (los objetos)
  - pueden ser
    - constantes
    - variables
    - métodos

# Constantes de clase (constantes static)

```
private static final double PI = 3.1416; // constante de clase con un valor inicial no  
// modificable
```

```
public class Circulo  
{  
    private static final double PI = 3.1416;  
    private double radio;  
    .....  
  
    public double getPerimetro()  
    {  
        return 2 * PI * radio;  
    }  
    .....  
}
```

# Constantes de clase (constantas static)



# Constantes de clase (constantes static)

- Habitual el uso de constantes static
- Definirlas con visibilidad public para que sean accesibles por todas las clases.
- Un ejemplo de este tipo de constantes lo encontramos en el propio lenguaje Java y la clase Math:

```
public class Math
{
    .....
    public static final double PI = 3.1415192.....;
    public static final double E = 2.718281;
    .....
}
```

**perimetro = 2 \* Math.PI \* radio;**



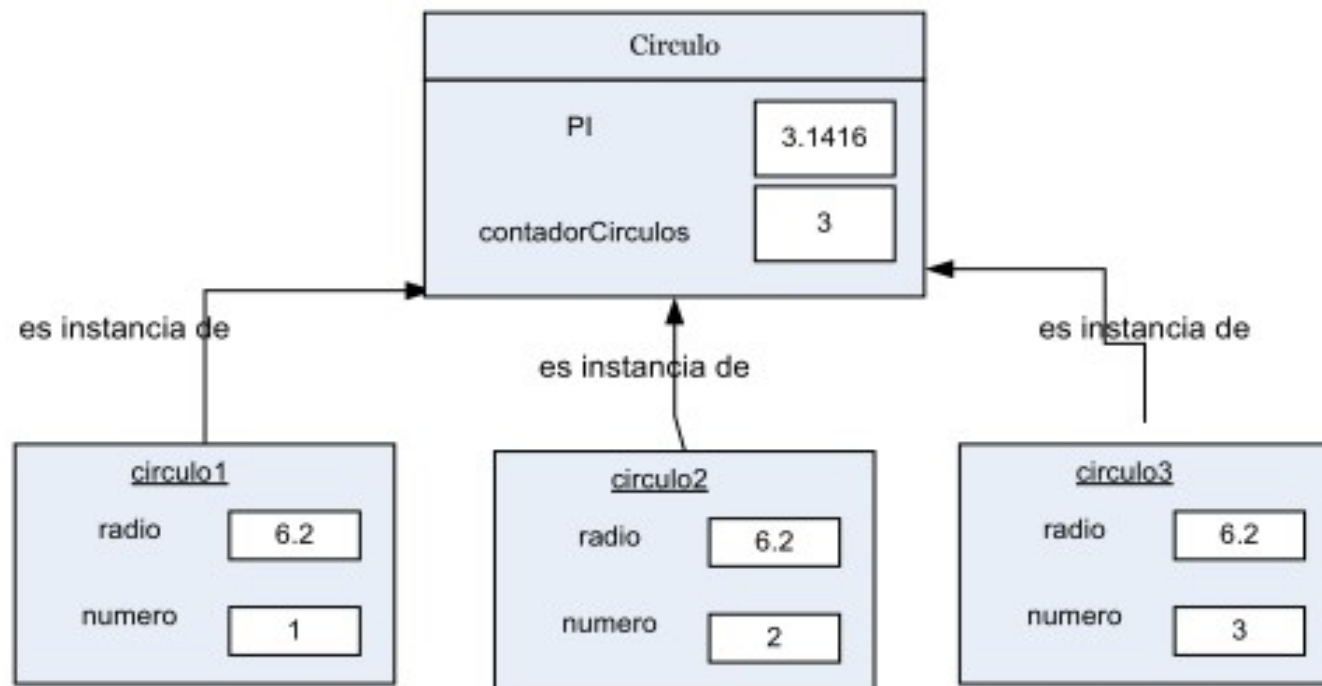
# Variables de clase (variables static)

- Variables que pertenecen a la clase
- **private static int contador;**

```
public class Circulo
{
    private static final double PI = 3.1416;
    private static int contadorCirculos = 0;
    private double radio;
    private int numero;
    .....

    public Circulo()
    {
        contadorCirculos++;
        numero = contadorCirculos;
    }
    .....
}
```

# Variables de clase (variables static)



- Java les asigna un valor por defecto (0, null, 0.0, ...)
- Se definen antes que las variables de instancia y después de las constantes
- Los métodos de instancia pueden acceder a las variables static de la misma forma que lo hacen con las variables de instancia.

# Métodos de clase (métodos static)

- Se asocian a la clase, no a las instancias
  - para ser invocados no se necesita crear ningún objeto (instancia) de la clase.
  - `public static int máximo(int numero1, int numero2)`
  - `public static double factorial(double numero)`
- No pueden acceder a las variables de instancia sólo a las variables de clase.
- Tampoco pueden llamar a los métodos de instancia (al revés sí, los métodos de instancia pueden llamar a los métodos de clase).

# Métodos de clase (métodos static)

## ■ Clase Math

- incorpora varios métodos *static*
- librería de utilidades
  - los métodos devuelven un valor a partir de los parámetros que se les pasan

```
public class Math
{
    .....
    public static double sqrt(double n)
    public static double random()
}
```

```
double resultado = Math.sqrt(89);
```

- Es habitual cuando se define una variable de clase definir también un método de clase para obtener su valor
  - **public static int getContadorCirculos()**

## ■ Desde Java 1.5

- `import static java.lang.Math.abs;`
- `import static java.lang.Math.*;`

```
int nuevoX = abs(destino.getX() - x);
```

# Métodos de clase (métodos static)

```
public class Fraccion
{
    private int numerador;
    private int denominador;
    .....
    private void simplificar()
    {
        int maxComunDivisor = mcd(numerador, denominador);
        numerador = numerador / maxComunDivisor;
        denominador = denominador / maxComunDivisor;
    }

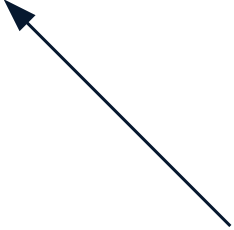
    private static int mcd(int num1, int num2)
    {
    }
}
```

Lo podríamos haber definido como *static*. Su tarea no depende del valor de los atributos sino de los parámetros que recibe.

# Métodos de clase (métodos static)

```
public class Numero
{
    .....
    public boolean hayCifrasRepetidas()
    {
        int aux = this.numero;
        while (aux != 0) {
            int cifra = aux % 10;
            if (estaCifra(aux / 10, cifra)) {
                return true;
            }
            aux = aux / 10;
        }
        return repetidas;
    }
}
```

```
private static boolean estaCifra(int numero, int cifra)
{
}
```

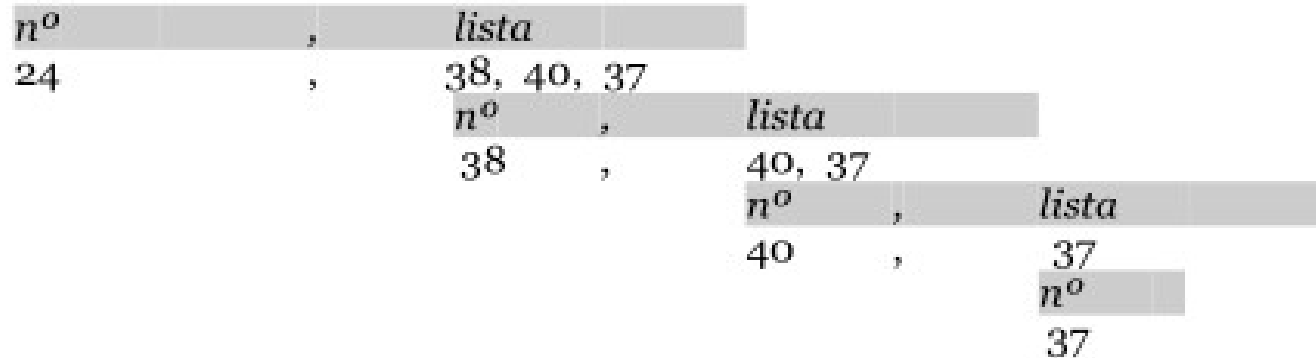


Lo podríamos haber definido como static. Su tarea no depende del valor de los atributos sino de los parámetros que recibe.

# Recursividad

- Qué es la **recursión**
  - técnica utilizada en programación que proporciona una solución elegante y sencilla a cierta clase de problemas
  - alternativa a la iteración
  - menos eficiente (consume recursos, memoria, stack, ...) pero
  - en cierto tipo de problemas las soluciones recursivas son menos complejas
- Una definición es recursiva si
  - el concepto o palabra que se define aparece en la propia definición
    - Por ejemplo, imaginemos la lista de n<sup>º</sup>s : 24, 38, 40, 37  
podríamos definir la lista así,
      - una lista es un n<sup>º</sup> o
      - una lista es un n<sup>º</sup> y una , junto con una lista

# Recursividad



- Detrás de la recursión se esconde el principio del “*divide y vencerás*”
  - el problema inicial se plantea en términos de problemas más pequeños pero de las mismas características, similar al original.
- Un método es recursivo si puede “llamarse” (invocarse) a sí mismo.



# Recursividad - Factorial recursivo

$$5! = 5 * 4 * 3 * 2 *$$

1

$$4! = 4 * 3 * 2 * 1$$

$$3! = 3 * 2 * 1$$

$$2! = 2 * 1$$

$$1! = 1$$

$$0! = 0$$

El factorial de  $5! = 5 * 4! =$

$$5 * 4 * 3! =$$

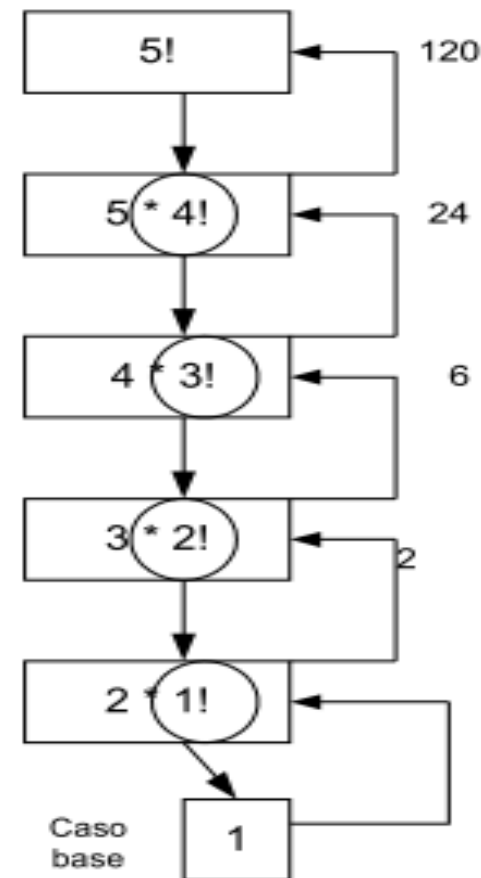
$$5 * 4 * 3 * 2! =$$

$$5 * 4 * 3 * 2 * 1! = \dots$$

- el problema  $n!$  expresado recursivamente queda:
  - 1 si  $n = 1$  o  $n = 0$  es el **caso base** o **caso trivial**, el que se resuelve sin recursión
  - $n * (n - 1)!$  si  $n > 1$  **caso general**

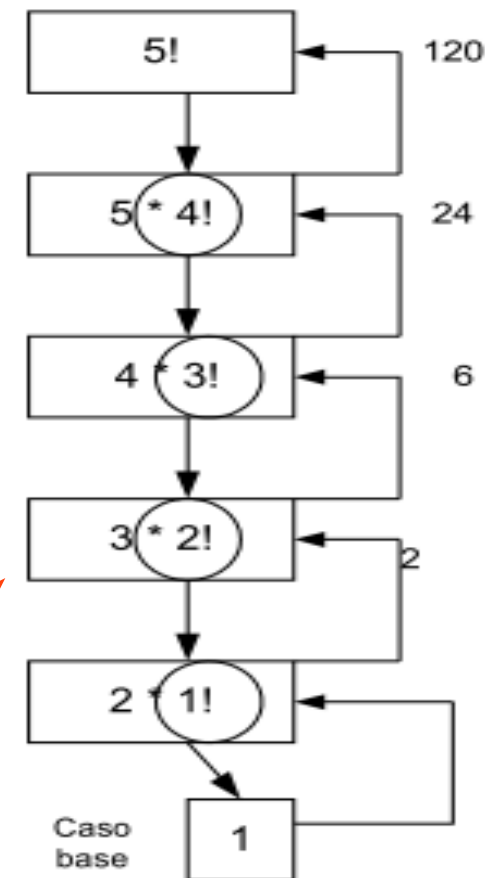
# Recursividad - Factorial recursivo

- En todo problema recursivo hay que encontrar
  - el caso **trivial** – el que se resuelve sin recursión (lo que sería la condición de fin de bucle en la iteración)
  - el caso **general** – incluye la recursión (subproblema idéntico al inicial pero de menor tamaño)



# Recursividad - Factorial recursivo

- En todo problema recursivo hay que encontrar
  - el caso **trivial** – el que se resuelve sin recursión (lo que sería la condición de fin de bucle en la iteración)
  - el caso **general** – incluye la recursión (subproblema idéntico al inicial pero de menor tamaño)



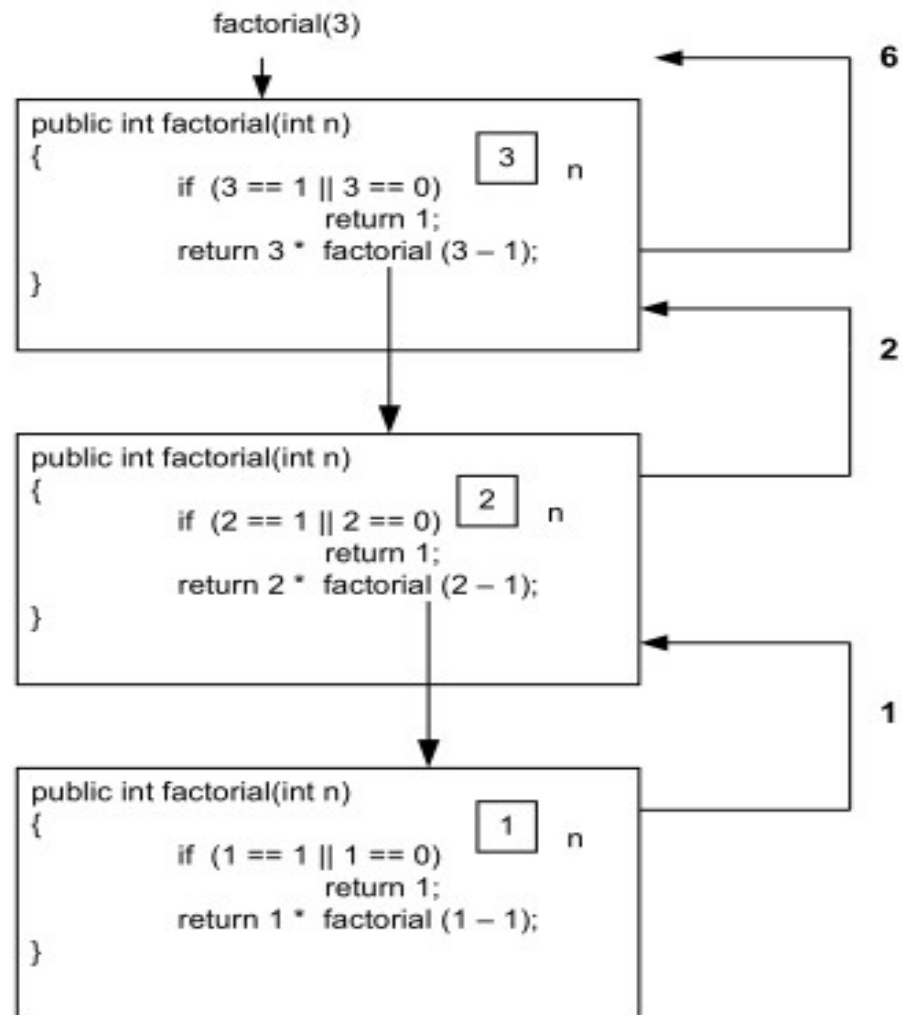
Cada nueva llamada al método supone un nuevo entorno de ejecución para él con nuevos parámetros y variables locales que no son visibles de una llamada a otra.

# Recursividad - Factorial recursivo

```
public int factorial(int n)
{
    if (n == 0 || n == 1) {
        return 1;
    }
    return n * factorial(n - 1);
}
```


```
public int factorial(int n)
{
    int resul;
    if (n == 1 || n == 0) {
        resul = 1;
    }
    else {
        resul = n * factorial(n - 1);
    }
    return resul;
}
```

# Recursividad - Factorial recursivo



# Recursividad - Factorial recursivo

factorial(3)

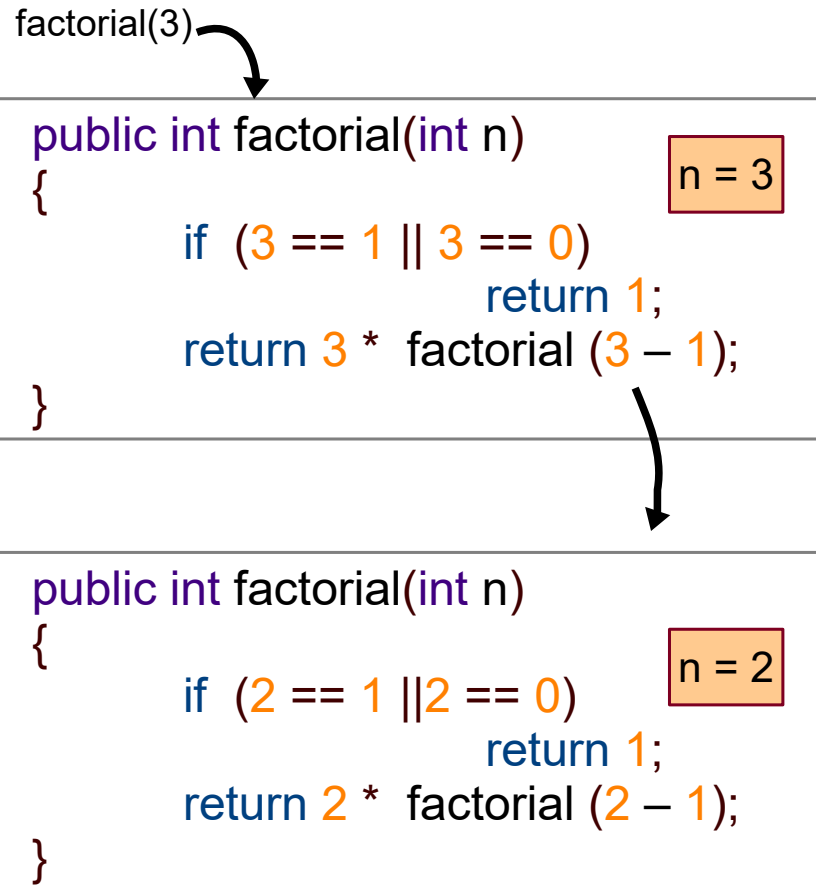


```
public int factorial(int n)
{
    if (3 == 1 || 3 == 0)
        return 1;
    return 3 * factorial (3 - 1);
}
```

n = 3

# Recursividad - Factorial recursivo

factorial(3)



```
public int factorial(int n)
{
    if (3 == 1 || 3 == 0)
        return 1;
    return 3 * factorial (3 - 1);
}
```

n = 3

```
public int factorial(int n)
{
    if (2 == 1 || 2 == 0)
        return 1;
    return 2 * factorial (2 - 1);
}
```

n = 2

# Recursividad - Factorial recursivo

factorial(3)

```
public int factorial(int n)
{
    if (3 == 1 || 3 == 0)
        return 1;
    return 3 * factorial(3 - 1);
}
```

n = 3

```
public int factorial(int n)
{
    if (2 == 1 || 2 == 0)
        return 1;
    return 2 * factorial(2 - 1);
}
```

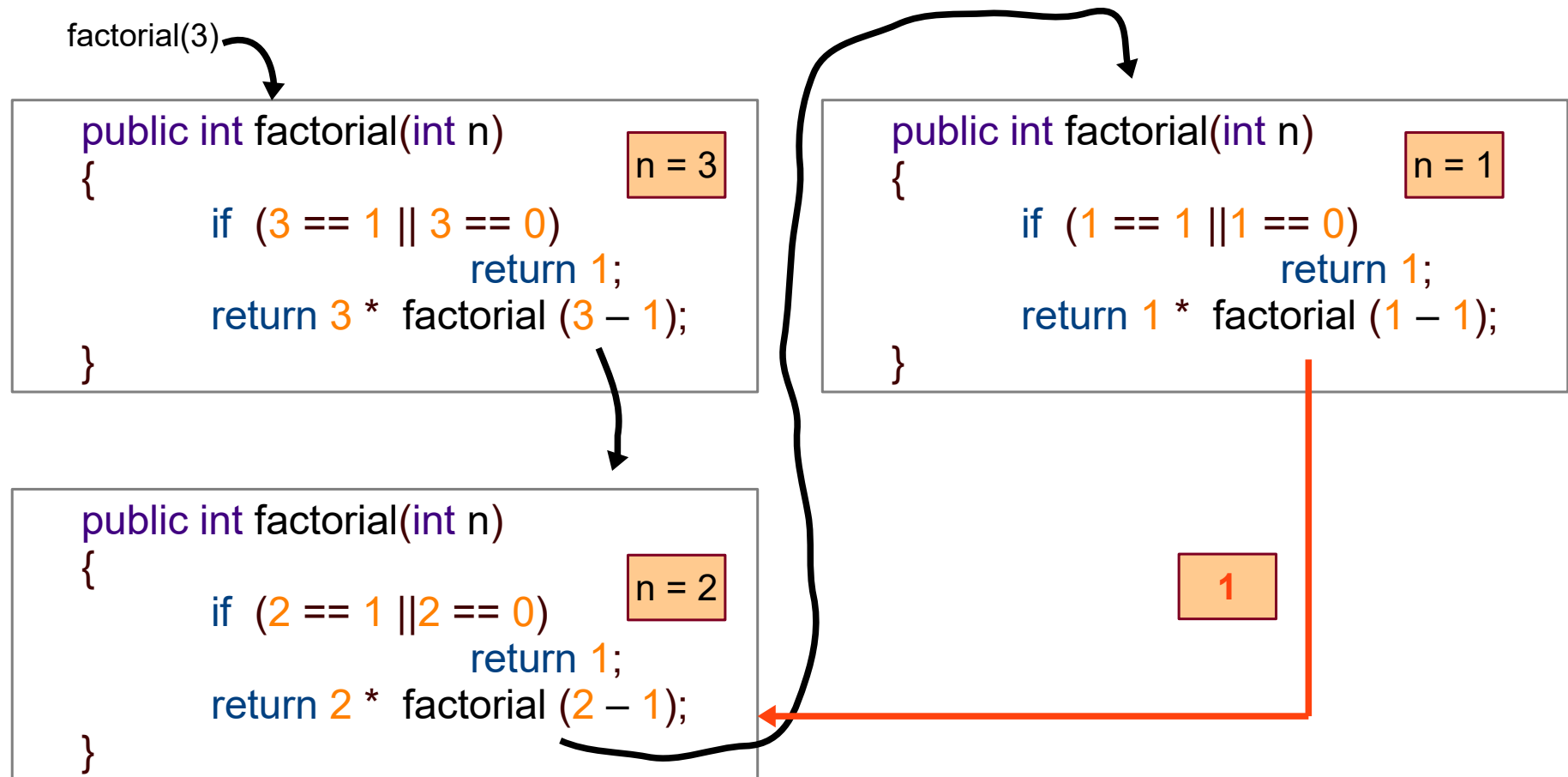
n = 2

```
public int factorial(int n)
{
    if (1 == 1 || 1 == 0)
        return 1;
    return 1 * factorial(1 - 1);
}
```

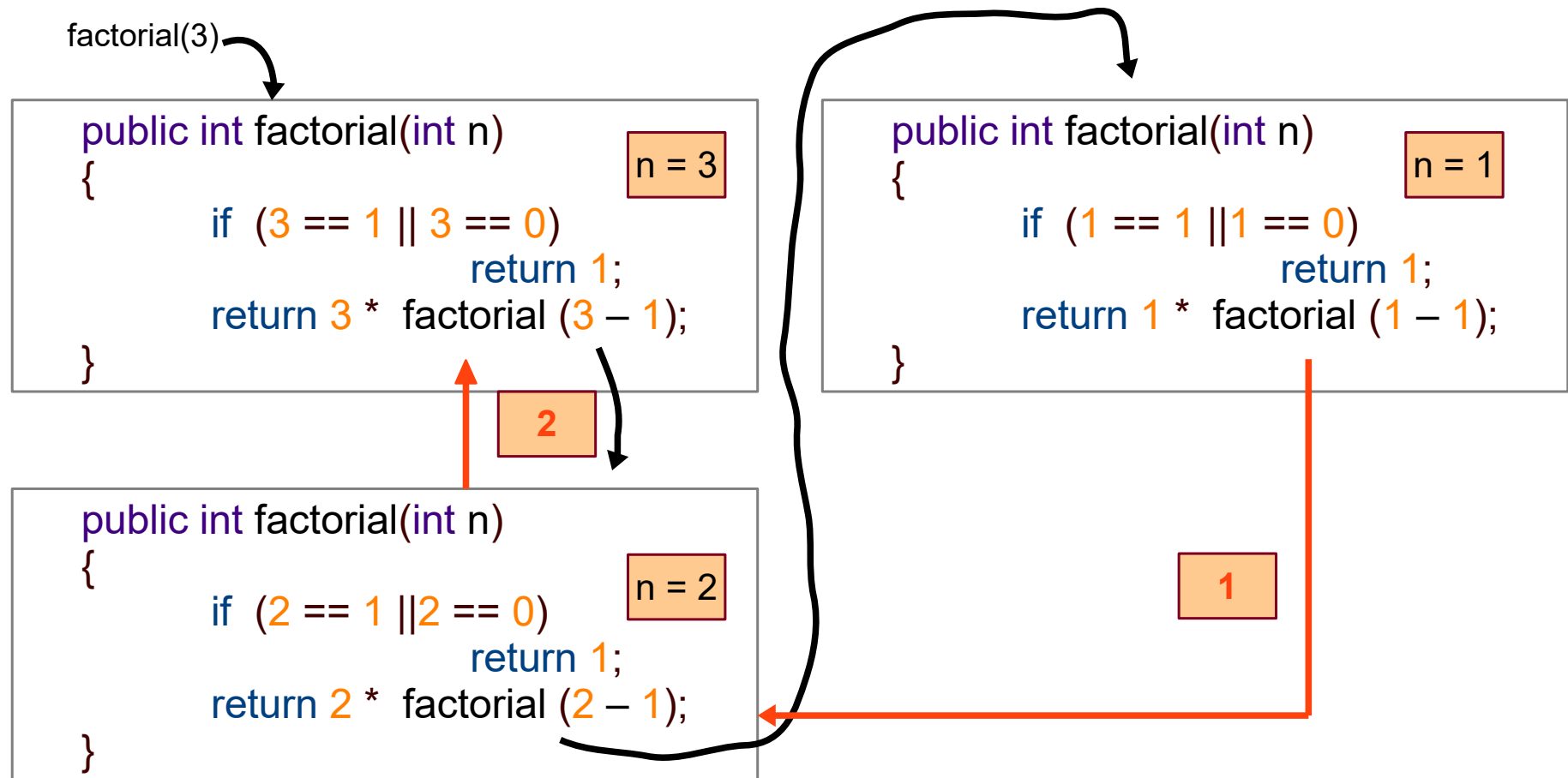
n = 1



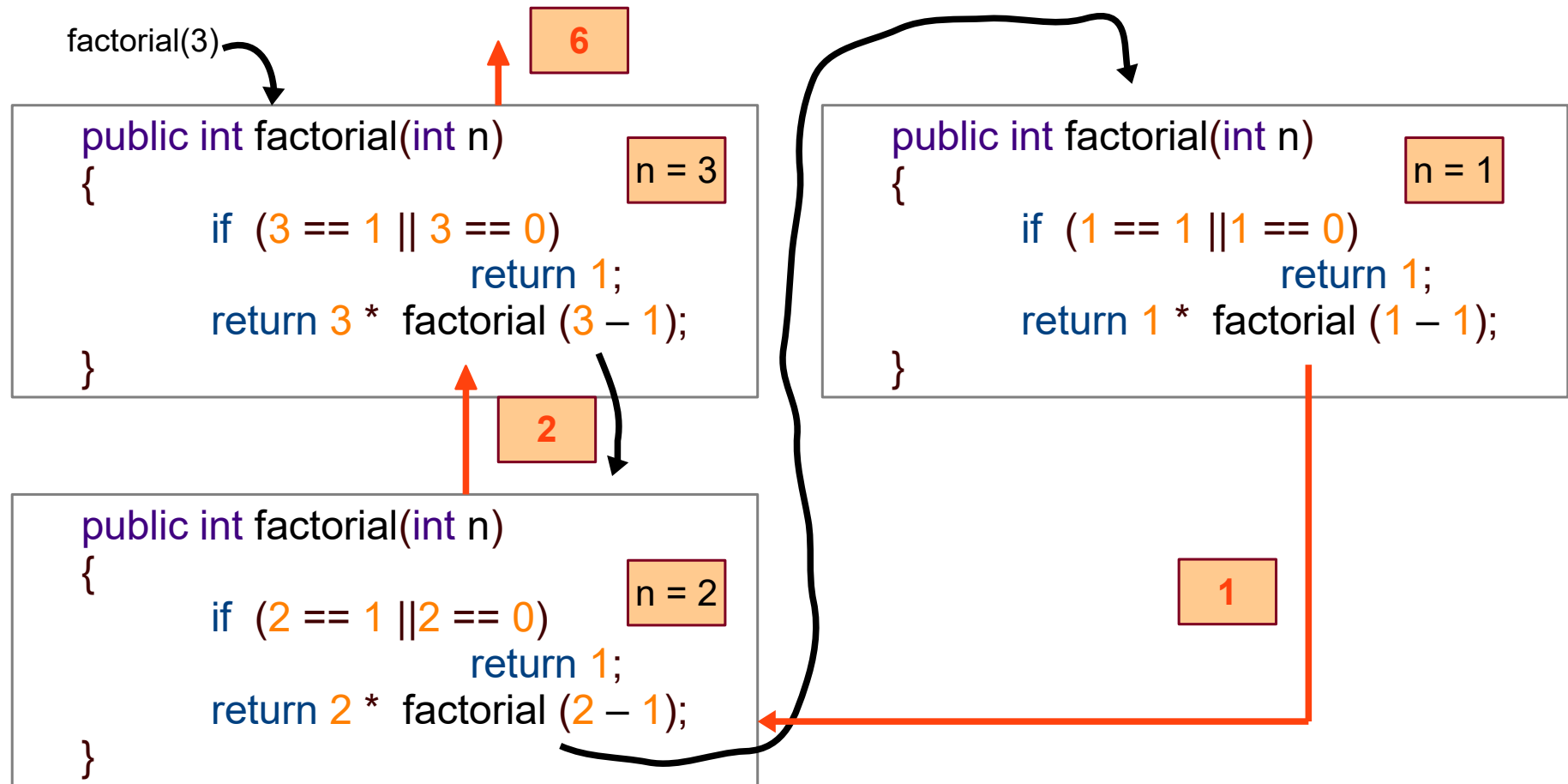
# Recursividad - Factorial recursivo



# Recursividad - Factorial recursivo



# Recursividad - Factorial recursivo



# Otra forma de hacer la traza

- **factorial(3)**
  - return  $3 * \text{factorial}(2)$  //  $3 * 2 = 6$

**factorial(2)**

- $2 * \text{factorial}(1)$  //  $2 * 1 = 2$

**factorial(1)**

- return **1**

# Otro ejemplo recursivo

- Escribir una línea de \*

\*\*\*\*\*

```
public void escribirAsteriscos(int n)
{
    for (int i = 1; i <= n; i++) {
        System.out.print("*");
    }
    System.out.println();
}
```

Versión iterativa

# Otro ejemplo recursivo

- Escribir una línea de \*

```
public void escribirAsteriscos(int n)
{
    if (n == 1)    { // caso base, escribir un *
        System.out.println("*");
    }
    else    { // caso general
        System.out.print("*");
        escribirAsteriscos(n - 1);
    }
}
```

Versión recursiva

# Otro ejemplo recursivo

- Escribir una línea de \*

```
public void escribirAsteriscos(int n)
{
    if (n == 0) { // caso base más simple, con n = 0 y no 1
        System.out.println();
    }
    else { // caso general
        System.out.print("*");
        escribirAsteriscos(n - 1);
    }
}
```

Versión recursiva

# Ejercicio (en papel)

- ¿Qué devolverá el siguiente método recursivo con las llamadas recursivas?
  - `int resul = misterio(0, 3);`
  - `int resul = misterio(10, 7);`
  - `int resul = misterio(5, 5);`
- Dibuja la secuencia de llamadas con los entornos de ejecución

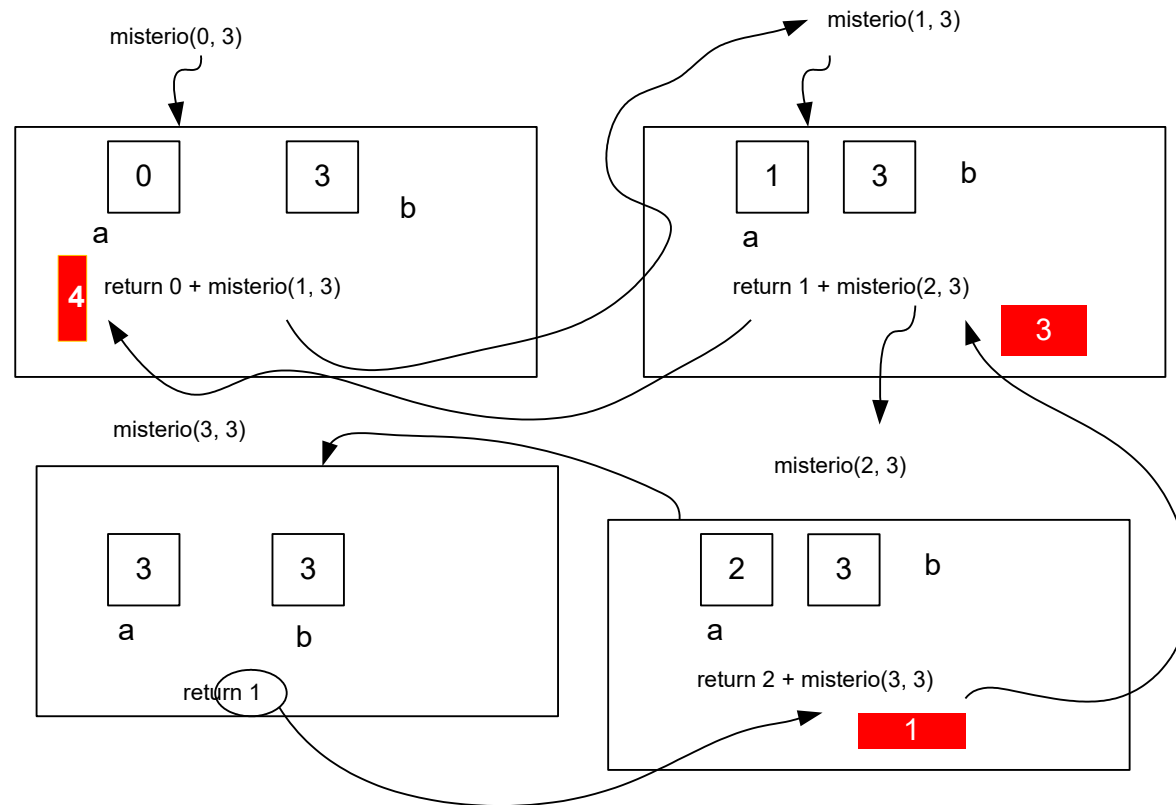
```
public int misterio(int a, int b)
{
    if (a == b) {
        return 1;
    }
    if (a > b) {
        return 0;
    }
    return a + misterio(a + 1, b);
}
```



# Ejercicio (en papel)

- ¿Qué devolverá el siguiente método recursivo con las llamadas recursivas?
  - `int resul = misterio(0, 3);`
    - devuelve **4**
  - `int resul = misterio(10, 7);`
    - devuelve **0**
  - `int resul = misterio(5, 5);`
    - devuelve **1**
- Dibuja la secuencia de llamadas con los entornos de ejecución

# Ejercicio (en papel)



# Ejercicio (en papel)

```
public int misterio(int n)
{
    if (n < 10) {
        return n;
    }
    else {
        int a = n / 10;
        int b = n % 10;
        return misterio(a + b);
    }
}
```

- ¿Resultado de las llamadas recursivas?
  - int resul = misterio(648);
  - int resul = misterio(15);
  - int resul = misterio(1234);

# Ejercicio (en papel)

```
public int misterio(int n)
{
    if (n < 10) {
        return 10 * n + n;
    }
    else {
        int a = misterio(n / 10);
        int b = misterio(n % 10);
        return 100 * a + b;
    }
}
```

- ¿Resultado de las llamadas recursivas?
  - int resul = misterio(348);
  - int resul = misterio(23);

# Ejercicios recursión

---

- Ejer 4.19
- `public int sumarDigitos(int numero)`
- `public int sumar(int a, int b)` – sumar a y b de forma recursiva
- `public int producto(int a, int b)` – multiplicar a y b de forma recursiva
- `public int contarDigitosPares(int numero)`
- `public int aparicionesDe(int numero, int digito)`