Algoritmos y Java

Fernando Bobillo

Resumen de contenidos

- Algoritmos
  - 10 familias de algoritmos importantes
  - Relación con arte y arquitectura
- Conceptos de programación
  - Programación en Java
  - Variables y constantes
  - Tipos de datos simples
  - Composición secuencial, condicional e iterativa
  - Tipos de datos estructurados
  - Modularidad
  - Programación orientada a objetos
  - Programación dirigida a eventos

Algoritmos

Aplicar algoritmos a los datos

Algoritmo

- Un algoritmo es una secuencia
  - ordenada,
  - finita y
  - bien definida

  de pasos para resolver un problema
- Instrucciones a realizar sobre unos datos
  - A partir de datos de entrada, datos de salida
- El nombre procede del matemático persa Al-Juarismi

Secuencia de pasos: receta de cocina

Algunos algoritmos clásicos

- Siglo XVI a. C.: factorizar y raíces cuadradas (babilonios)
- Siglo III a. C.: máximo común divisor (Euclides)
- Siglo II a. C.: números primos (criba de Eratóstones)
- Siglo IX: resolución de ecuaciones (Al-Juarismi)

Algoritmo de Euclides

- Calcula el máximo común divisor (MCD) de 2 enteros
- Algoritmo de Euclides

1. Sea a el número mayor y sea b el número menor
2. Si b es igual a 0, entonces el MCD es a y terminamos
3. Si no, volver al paso 2 con a = b y b = resto de “a entre b”

- Ejemplo: MCD de 12 y 8
  - Paso 1: a = 12, b = 8
  - Paso 2: no se aplica
  - Paso 3: a = 8, b = 4
  - Paso 2: no se aplica
  - Paso 3: a = 4, b = 0
  - Paso 2: el MCD es 4

Algoritmo para calcular la letra del DNI

- Sea n el resto de dividir el número del DNI entre 23
- Devolver la letra que ocupa la posición n+1 en la palabra

  TRWAGMYFPDXBNJZSQVHLCKE
- Ejemplo:

  - n = 99999999 % 23 = 1
  - Letra 2ª de la cadena = R

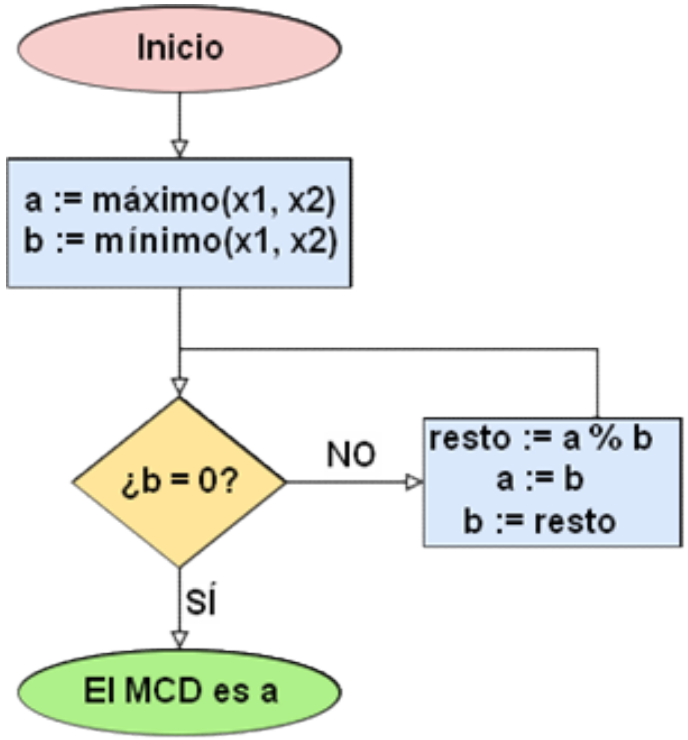
Programación

- Un algoritmo es independiente de su representación
  - Lenguaje natural
  - Diagramas de flujo
  - Pseudocódigo
  - Lenguajes de programación
  - ...
- Programa: codificación concreta de uno o más algoritmos
- Ada Byron: 1ª programadora (1842-43)

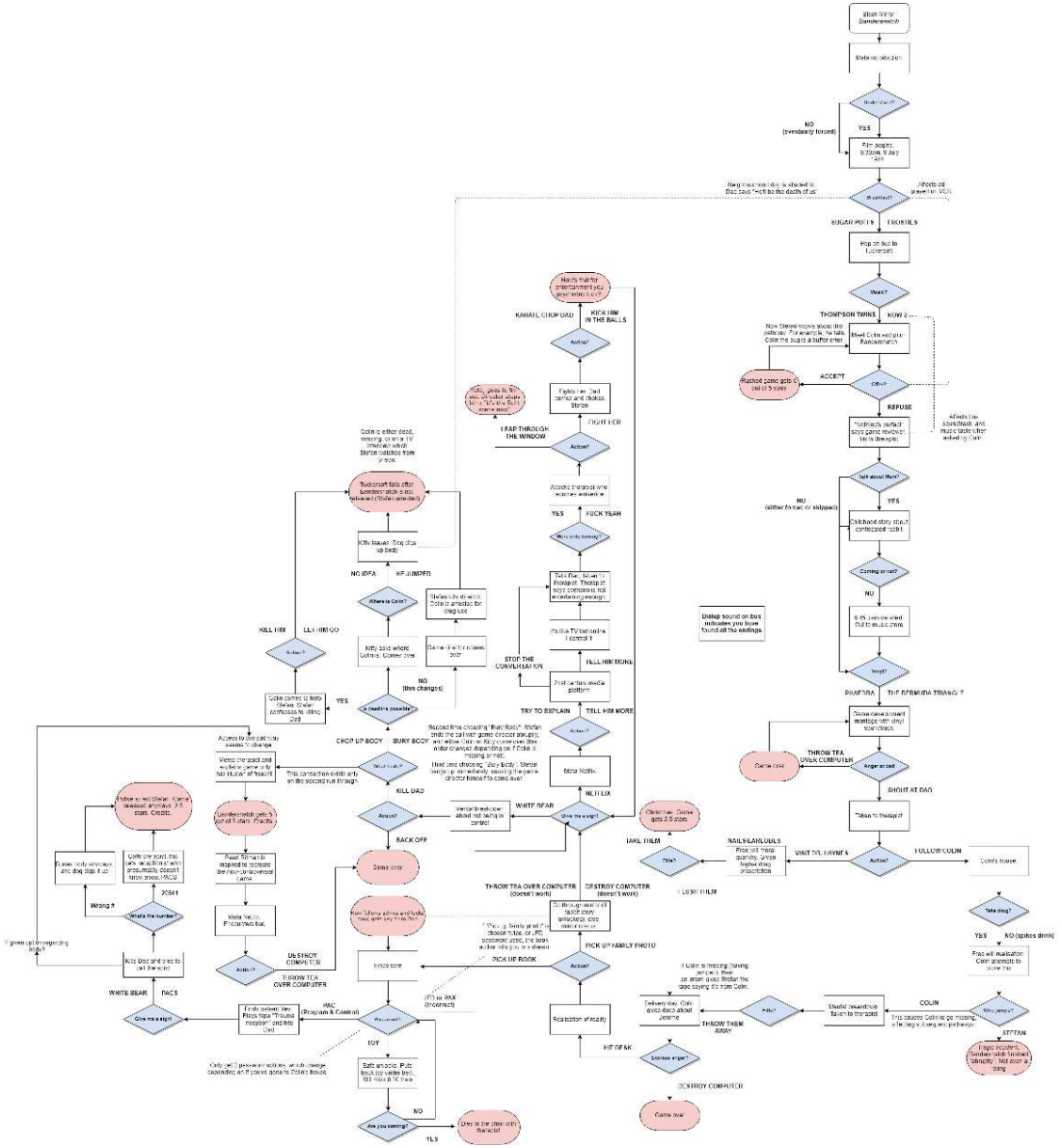
Pseudocódigo y diagrama de flujo

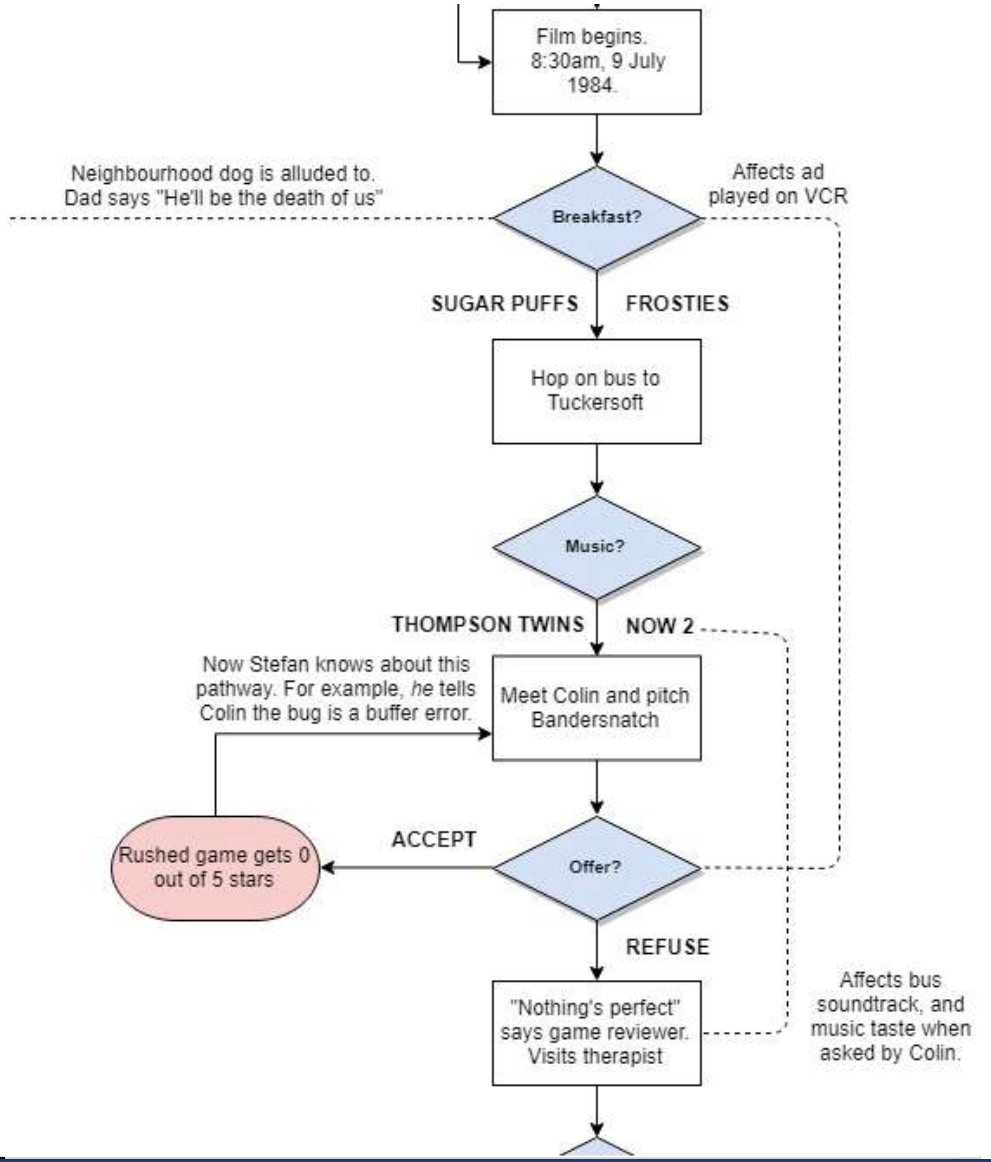
- Pseudocódigo (falso código): sintaxis similar a la de lenguaje de programación pero pensado para la lectura humana
- Diagrama de flujo: representación gráfica

```
{ MCD(x1, x2)
a := máximo(x1, x2)
b := mínimo(x1, x2)
mientras b ≠ 0 repetir
principio
        resto := a % b
        a := b
        b := resto
fin
devolver a
```

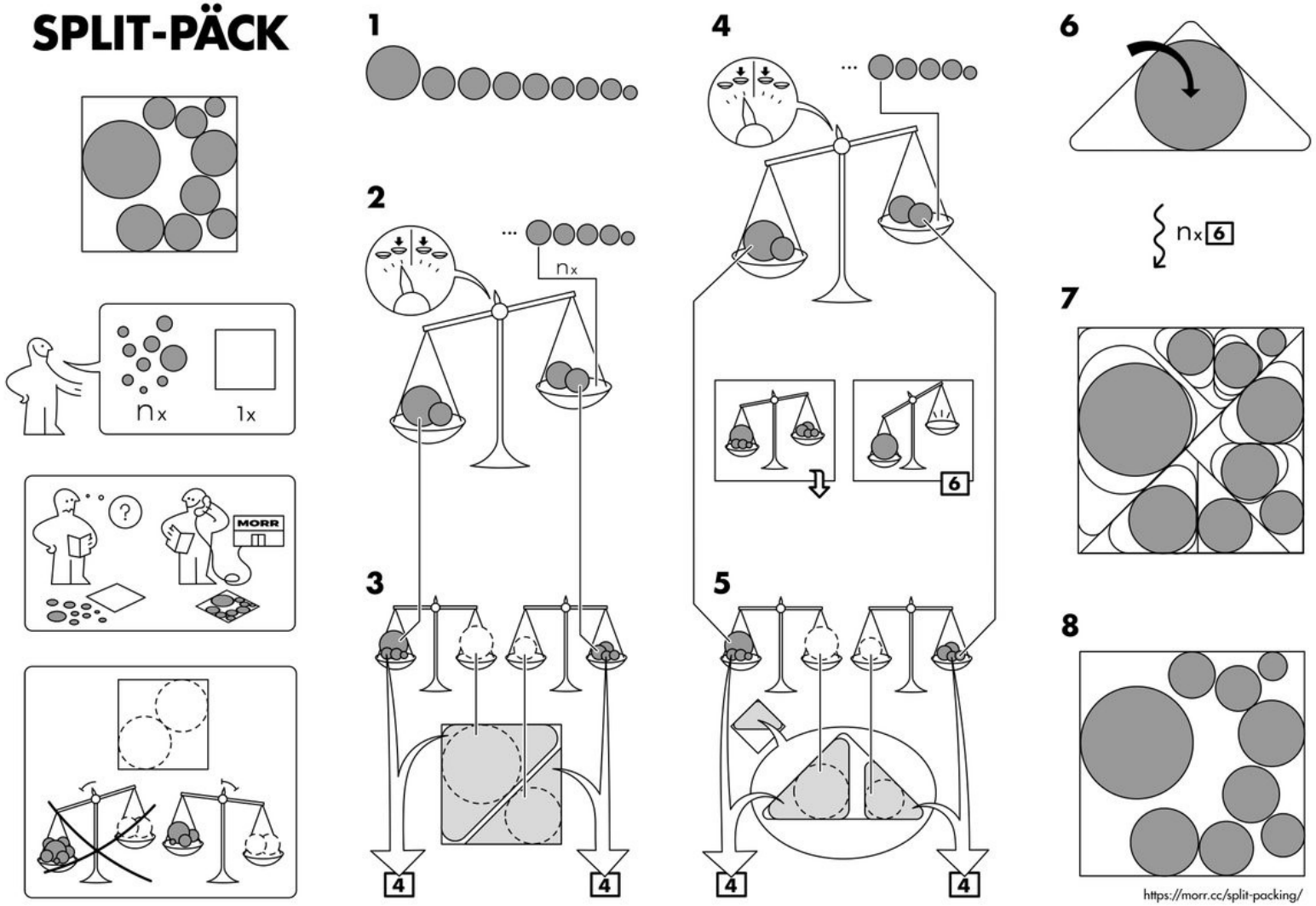


Black Mirror: Bandersnatch





Algoritmo expresado en estilo IKEA

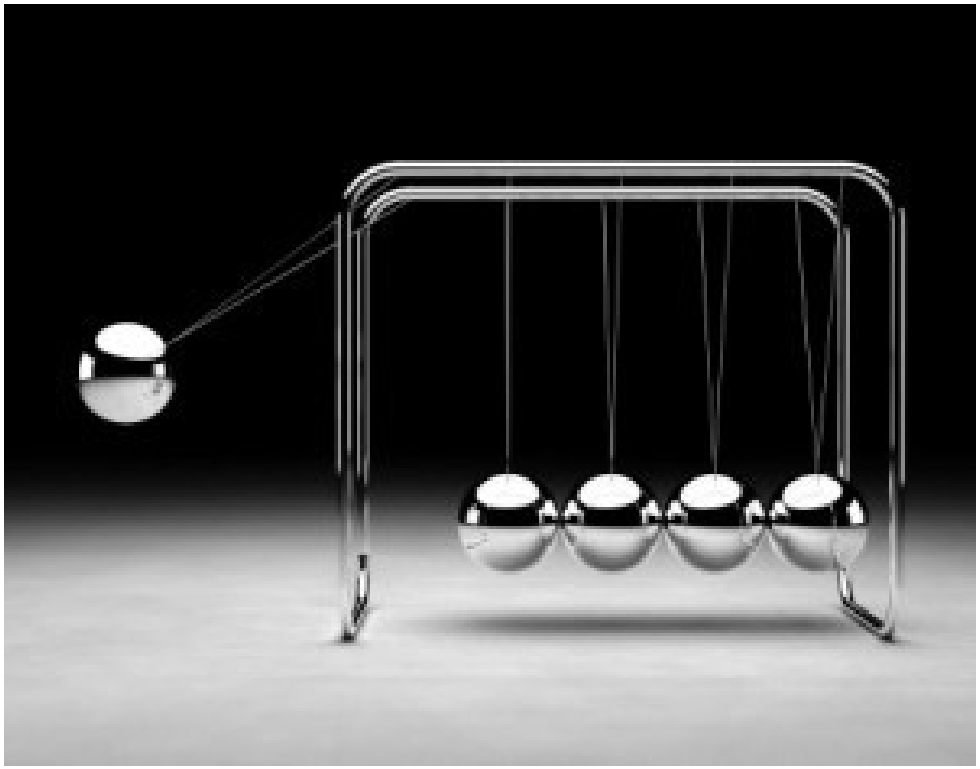


Características de un algoritmo válido

- Carácter finito
- Precisión en la definición de los pasos
- Entrada. Un algoritmo tiene cero o más entradas: cantidades que le son dadas antes de que el algoritmo comience, o dinámicamente mientras el algoritmo se ejecuta
- Salida. Un algoritmo tiene una o más salidas: cantidades que tienen una relación específica con las entradas
- Eficacia. Todas las operaciones deben ser suficientemente básicas como para poder ser hechas en principio de modo exacto y en un tiempo finito por un hombre con lápiz y papel

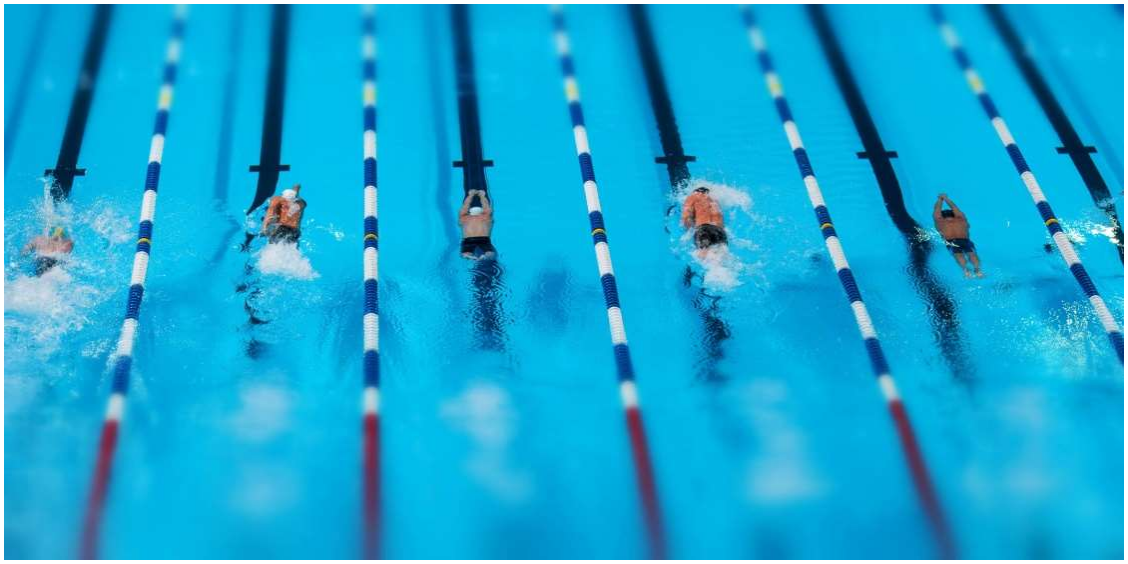
Características de un algoritmo válido

- A partir del carácter finito y de la salida se deduce que ante una misma situación inicial (o valores de entrada) un algoritmo debe proporcionar siempre el mismo resultado (o salida), con excepción de los algoritmos probabilistas



Tipos de algoritmos

- Secuenciales: cuando se realiza una operación cada vez
- Paralelos: realizan varias operaciones de forma simultánea

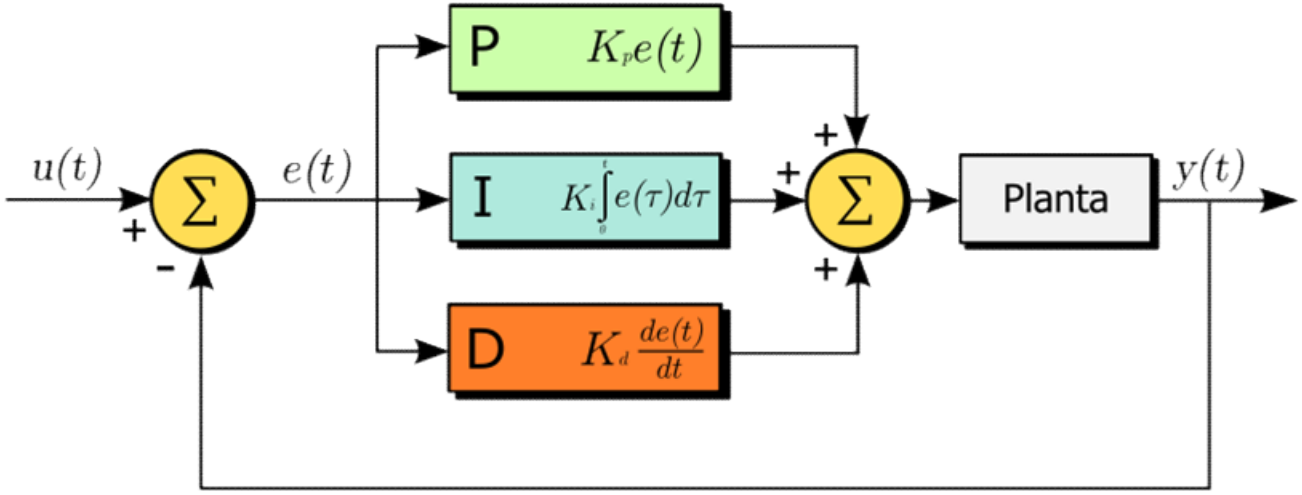


10 familias de algoritmos importantes

1. Control
2. Recuperación de páginas web
3. Ordenación
4. Resumen
5. Criptografía
6. Generación de números aleatorios
7. Reconocimiento de patrones
8. Compresión de datos
9. Caminos en grafos
10. Recomendación

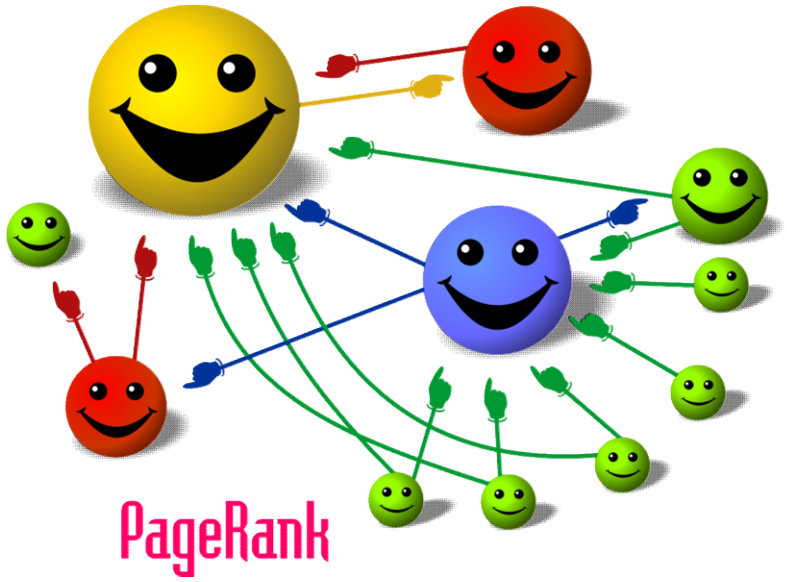
Control

- Controla el funcionamiento de un sistema dinámico de acuerdo a una determinada estrategia
- Ejemplos de aplicaciones
  - Automatización industrial
  - Controladores en electrodomésticos
  - Controladores en vehículos



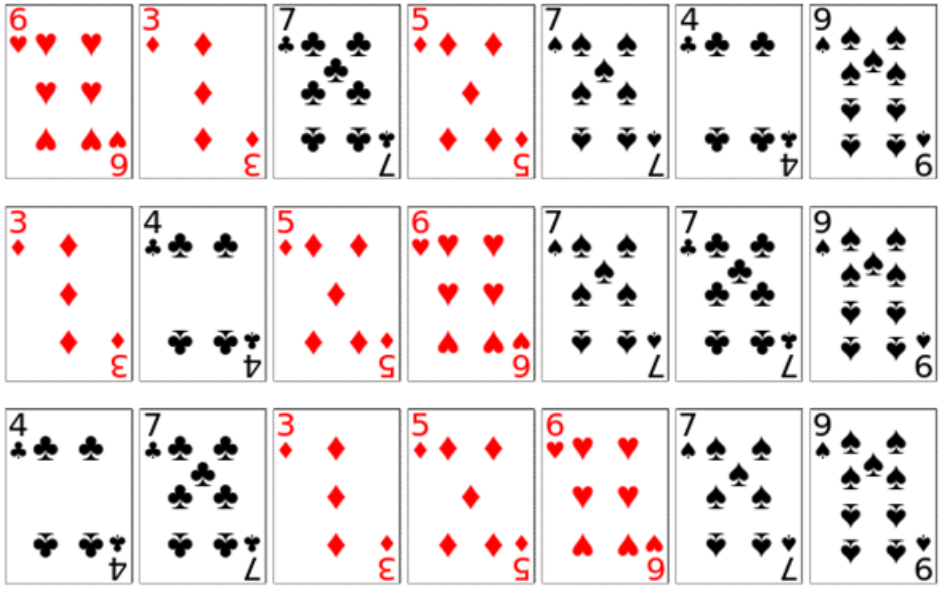
Recuperación de páginas web

- Encontrar páginas web que sean relevantes a la consulta de un usuario y devolverlas ordenadas según su importancia
- El algoritmo de Google (PageRank) tiene en cuenta muchas cosas: nº enlaces a una página, reputación de quien cita…
- Ejemplos de aplicaciones
  - Búsquedas en Google



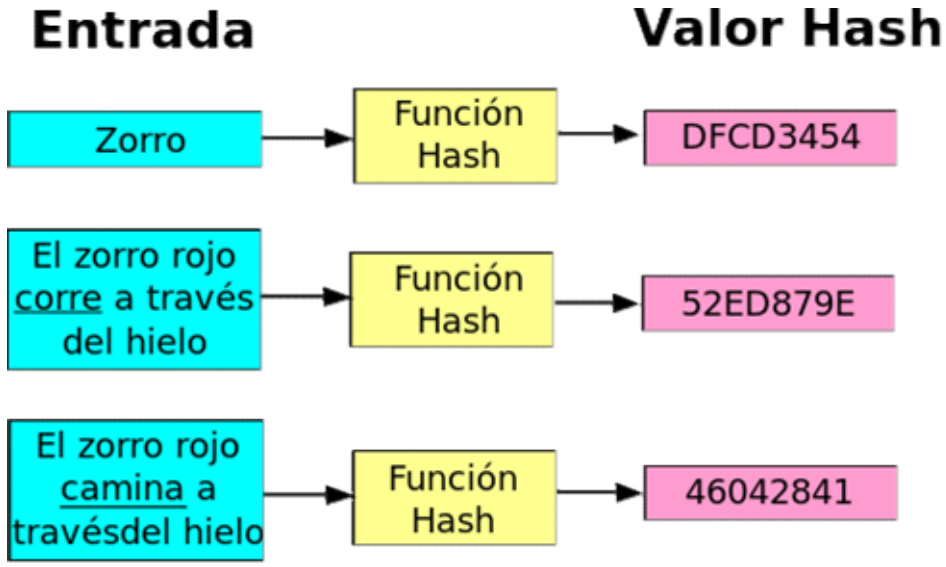
Ordenación

- Dada una lista de objetos, ordenarla según algún criterio
- Ejemplos de aplicaciones
  - Búsquedas más rápidas
  - Frecuencia de aparición de cada objeto
  - Igualdad de dos listas



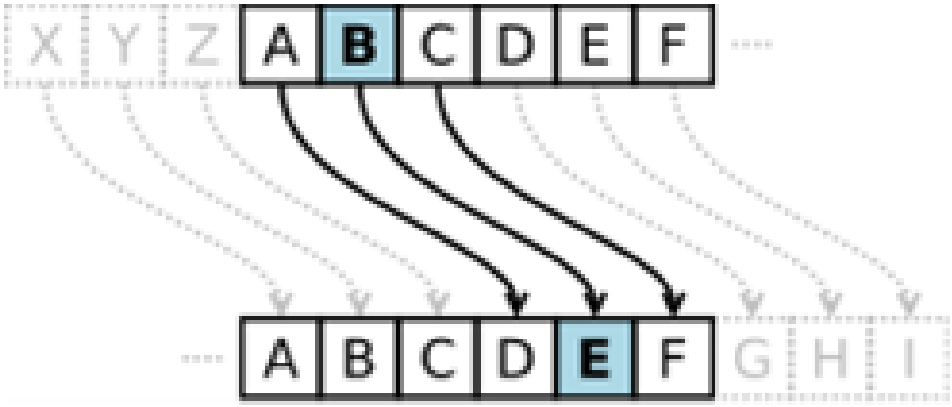
Resumen

- Dada una lista de datos de longitud variable, calcular un resumen (hash) de la lista con un tamaño máximo fijo
- Ejemplos de aplicaciones
  - Verificación de que una descarga de Internet es correcta
  - Búsqueda automática de subtítulos



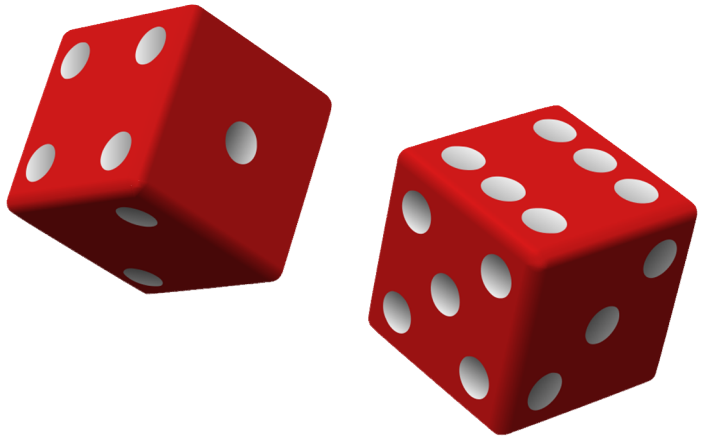
Criptografía

- Codificar un mensaje de manera que solamente el emisor y el receptor puedan decodificarlo y entenderlo
- Ejemplos de aplicaciones
  - Envío de datos sensibles (tarjetas de crédito) por Internet
  - Protección de datos (contraseñas) contra robos
- Ejemplo: cifrado César
  - Cada letra se desplaza *k* posiciones en el alfabeto



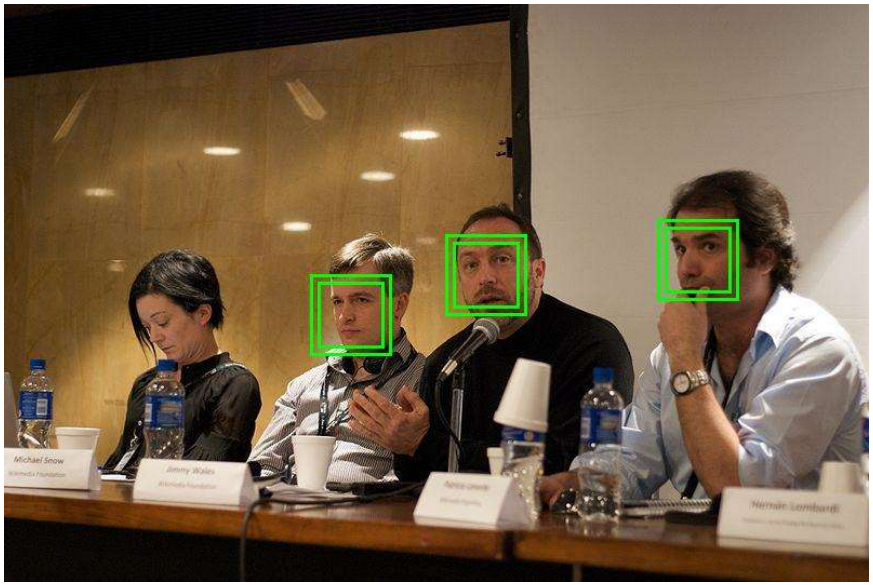
Generación de números aleatorios

- Producir secuencias de números aparentemente aleatorios
- Son pseudoaleatorios (del primer número de la secuencia se pueden obtener los demás) pero superan tests estadísticos
- Uso: algoritmos que no devuelven siempre el mismo resultado
- Ejemplos de aplicaciones
  - Simulación de procesos
  - Videojuegos



Reconocimiento de patrones

- Agrupar datos en dos o más clases conocidas de patrones
- Ejemplos de aplicaciones
  - Reconocimiento de imágenes para escanear textos, reconocer firmas o caras, detectar sospechosos…
  - Reconocimiento de audio para dar órdenes por voz, detectar personas, descubrir qué canción está sonando…



Compresión de datos

- Reducir el tamaño de la representación de una información
  - Puede ser con pérdida o sin pérdida de información
- Ejemplos de aplicaciones
  - Archivos de menor tamaño (sin pérdida) en formato ZIP
  - Imágenes de menor tamaño (con pérdida) en JPG
  - Audios de menor tamaño (con pérdida) en MP3
  - Vídeos de menor tamaño (con pérdida) en MPG

Compresión de datos: RLE

- RLE (*Run Length Encoding*): técnica de compresión de datos basada en la sustitución de los datos repetidos consecutivos
- Una secuencia donde el mismo carácter aparece varios veces se sustituye por un dato especial (marcador) seguido del número de repeticiones y del dato que se repite
  - Solo se sustituye en los casos en que resulte favorable
  - Los marcadores se sustituyen siempre
- Ejemplo: comprimir aaaaaaabbccccddde\fg con el marcador \
  - Resultado: \7abb\4cddde\1\fg
- La sustitución de a y c ahorra espacio, la de d no mejoraría y las demás aumentarían el espacio necesitado

Compresión de datos: códigos de Huffman

- Ejemplo: supongamos un fichero con 100 caracteres
- Código de longitud fija (3 bits/carácter):

a	b	c	d	e	f	Tamaño
000	001	010	011	100	101	300 b

- Códigos de Huffman: usan la frecuencia de aparición de los caracteres para construir una codificación óptima

	a	b	c	d	e	f	Tamaño
Frecuencia	45	13	12	16	9	5	
Código	0	101	100	111	1100	1101	224 b

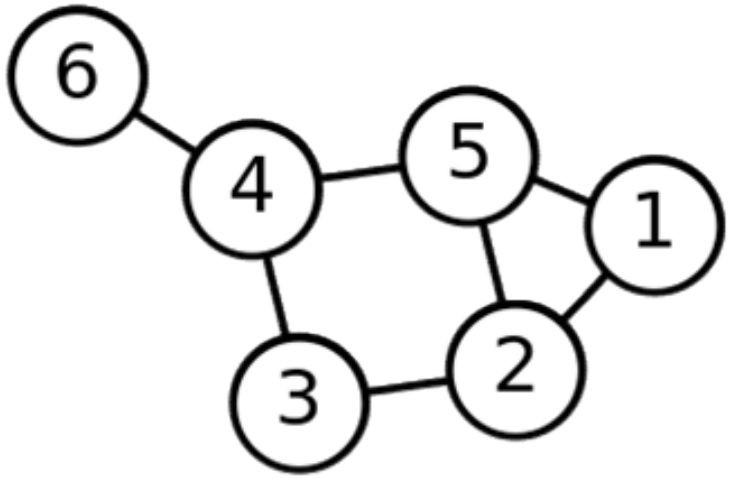
Ejemplo de JPG (original, 1808 KB)

Ejemplo de JPG (75% calidad, 644 KB)

Ejemplo de JPG (10% calidad, 117 KB)

Grafos

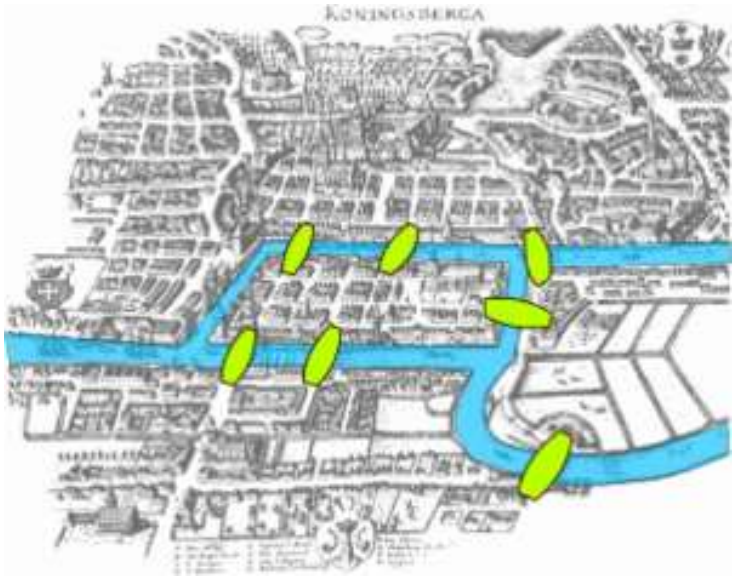
- Grafo: estructura matemática caracterizada por
  - Nodos (vértices)
  - Enlaces (aristas) conectando pares de nodos
- Cada enlace puede tener un sentido y/o una etiqueta
- Aplicaciones: redes de electricidad, síntesis de circuitos, redes de telefonía, redes de Internet, modelos de transporte

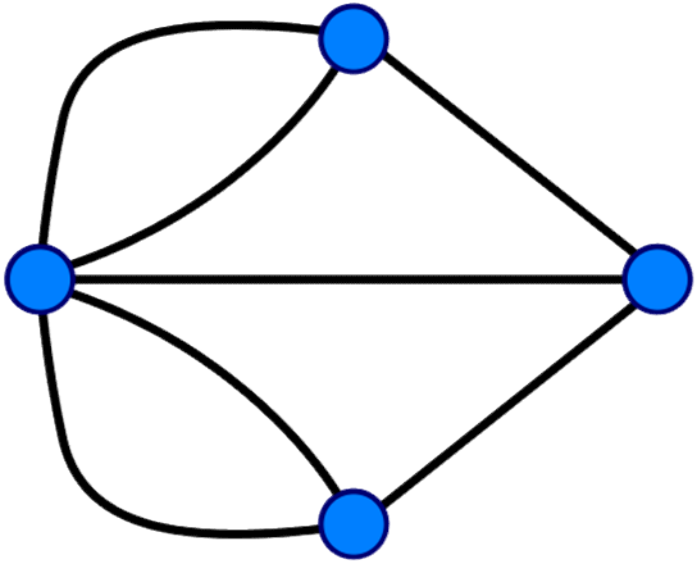


Los puentes de Königsberg

Los puentes de Königsberg

- Königsberg (actual Kaliningrado) tenía 7 puentes
- Problema: encontrar un recorrido para pasar una sola vez por cada puente y regresar al punto inicial

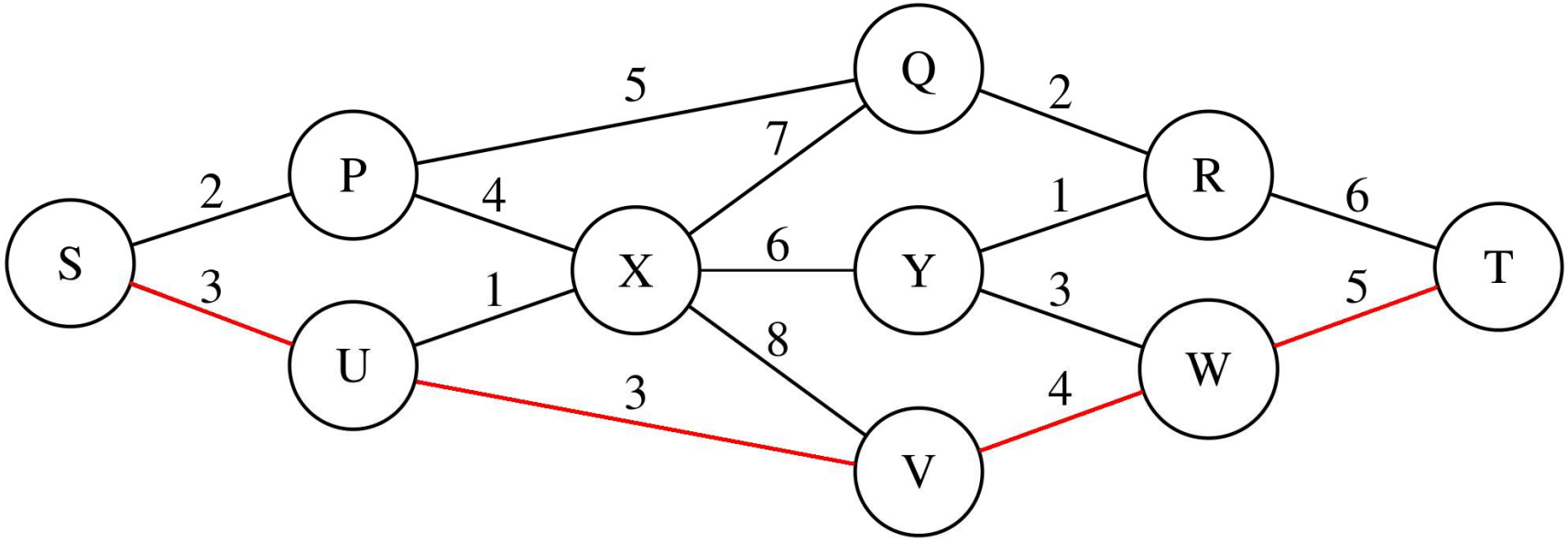




- El matemático Leonhard Euler demostró que no es posible

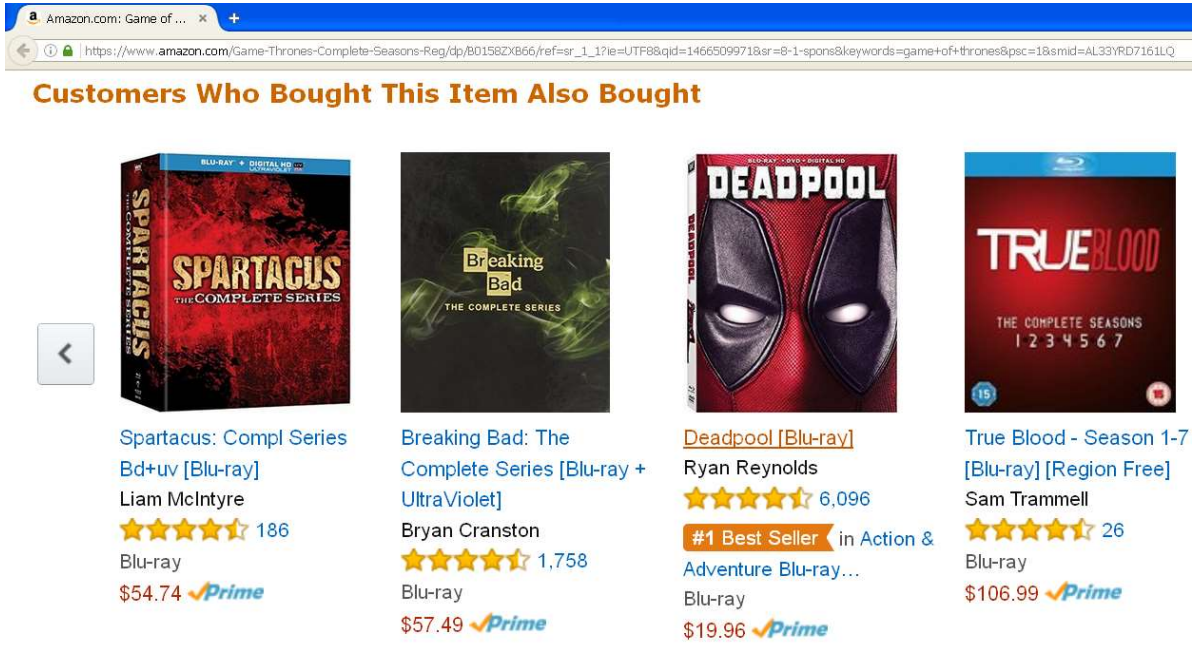
Caminos en grafos

- Dado un grafo, calcular un camino entre dos de sus nodos
- Puede requerirse alguna propiedad: mínima longitud, coste…
- Ejemplos de aplicaciones
  - Ruta más corta entre dos ciudades (Google Maps)
  - Ruta más rápida para enviar un fichero por Internet



Recomendación

- Dar a los usuarios sugerencias personalizadas e inteligentes
- Ejemplos de aplicaciones
  - Productos similares en Amazon
  - Amigos en redes sociales
  - Vídeos, música, noticias en Youtube, Spotify o Heraldo



Collectible Spoons of the 3rd Reich Paperback – April 9, 2009

by James A Yannes (Author)

Be the first to review this item

See all formats and editions

Paperback
$22.64

11 Used from $13.09
22 New from $15.94

Collectible Spoons of the 3rd Reich is a detailed, heavily illustrated reference book containing relevant historical exposition on many of the personal, organizational and commemorative spoons of the 3rd Reich period from 1933 to 1945. These spoons, unlike most other collectibles from this period, were actually owned and used daily by the people and organizations of those times. The book includes many spoon types, for example: Hitler's personal silverware, Red Cross, SS, the U-47 etc. With over 200 photos / graphics and over 19,000 words of text, the book extensively explores the relevant historical highlights which in turn illuminate this unique period in history as reflected by the spoons. These spoons are history that you can hold in your hand and were once in the hands of the German history

See this image

Read more

The Amazon Book Review
Author interviews, book reviews, editors picks, and more. Read it now

Customers Who Viewed This Item Also Viewed

The Encyclopedia of Third Reich Tableware
James A. Yannes
★★★★★ 2
Paperback
$29.82 Prime

Images You Should Not Masturbate To
› Graham Johnson
★★★★☆ 260
Paperback
$9.95 Prime

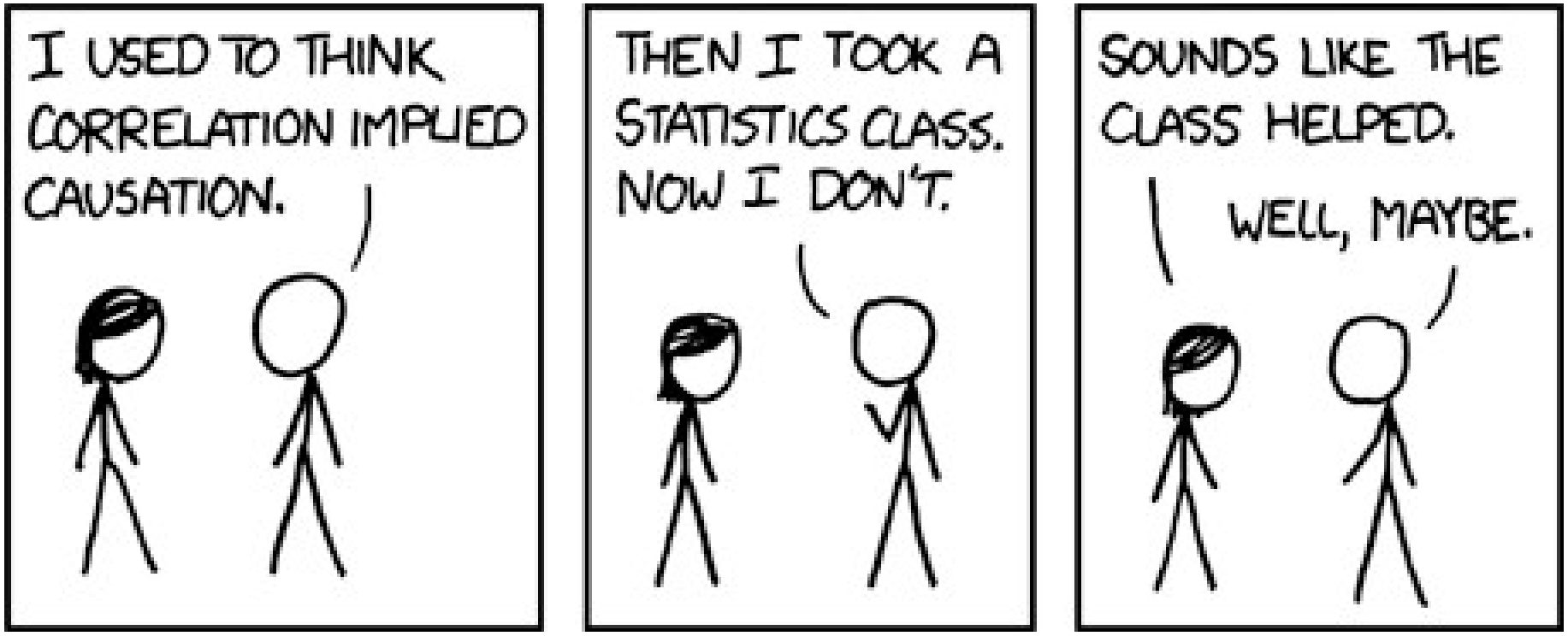
You Could Look It Up: The Reference Shelf From Ancient Babylon to...
Jack Lynch
★★★★★ 9
Hardcover
$20.26 Prime

http://www.amazon.com/Collectible-Spoons-Reich-James-Yannes/dp/1425186955

Descubrimiento de conocimiento

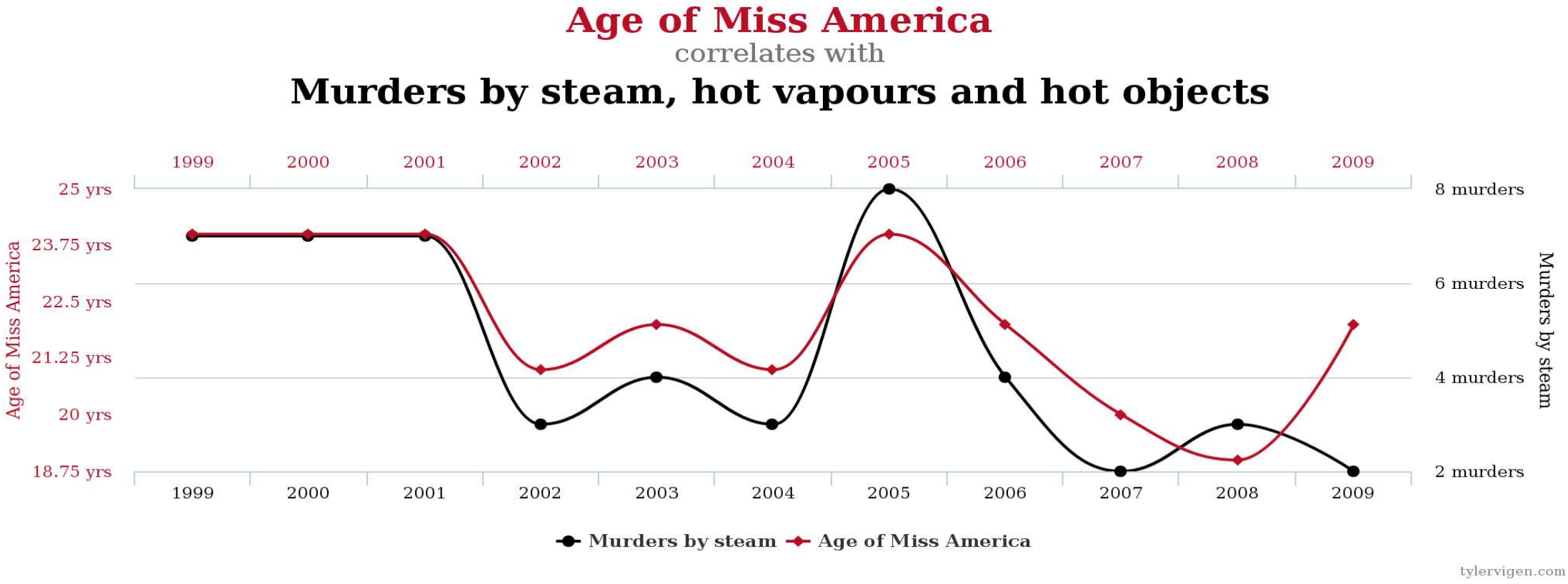
- Minería de datos (*data mining*): proceso que trata de descubrir patrones en grandes volúmenes de conjuntos de datos
  - Nombre confuso: extrae conocimiento, no datos
- El descubrimiento de conocimiento oculto en los datos no es en absoluto un problema trivial
- A veces las correlaciones se deben a una causa común que con frecuencia es desconocida (una variable oculta)
  - Dormir con zapatos + dolor de cabeza: borrachera
  - Estar gordo + usar paraguas: mayor poder adquisitivo
- Importante determinar causa y efecto
  - Cuando los molinos giran con gran velocidad, se observa mucho viento

Descubrimiento de conocimiento



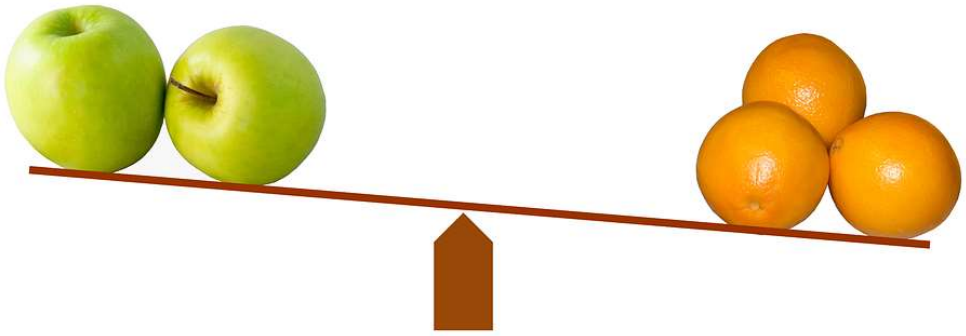
Descubrimiento de conocimiento

- En ocasiones hay correlaciones sin razón aparente
  - Ejemplo: Edad de Miss América y asesinatos por vapor



Comparación de algoritmos

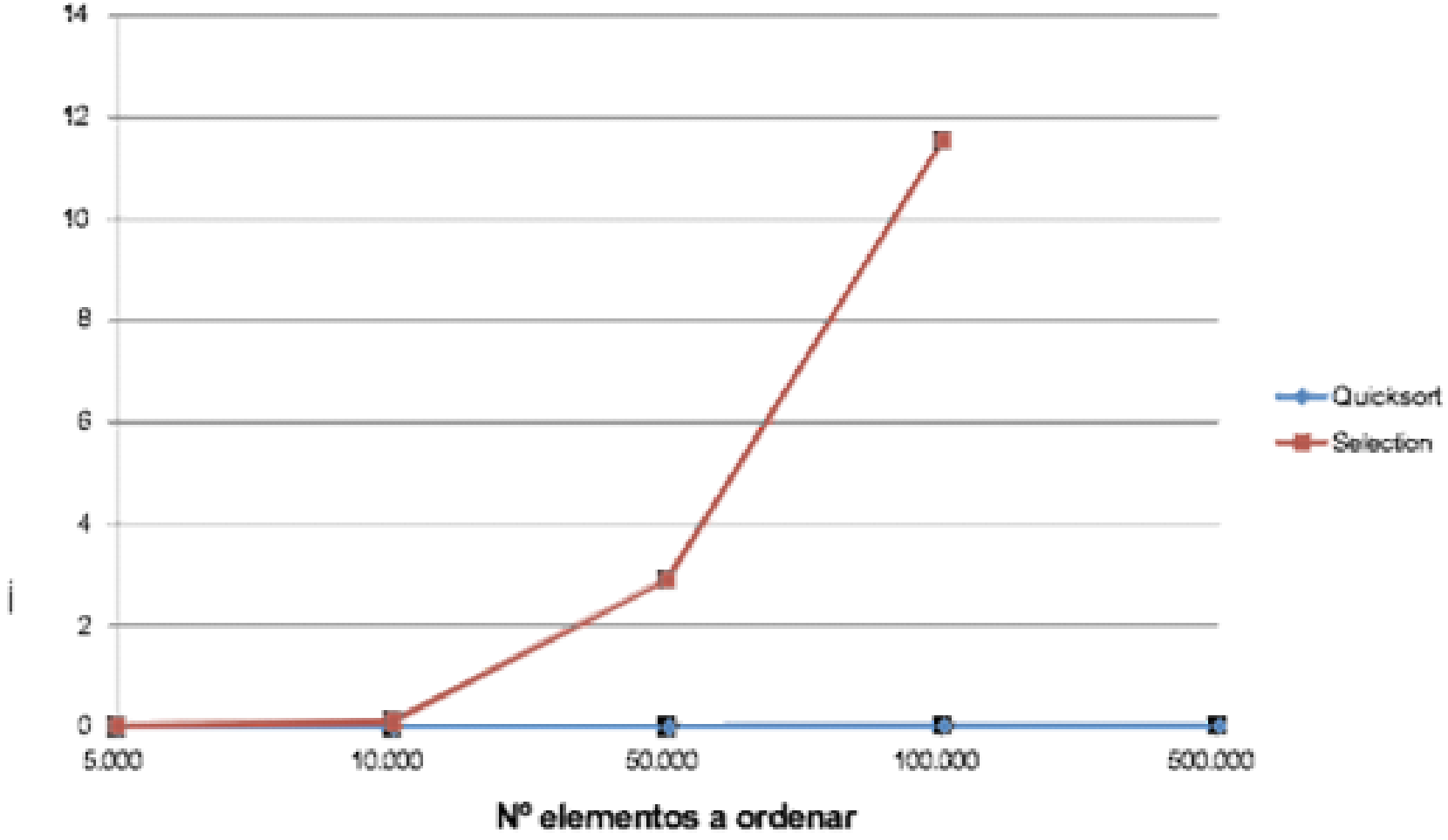
- Solamente comparar algoritmos para el mismo problema
- ¿Qué?
  - Tiempo
  - Memoria
  - Otros (estabilidad)
- ¿Cuándo?
  - Mejor caso
  - Peor caso
  - Caso promedio



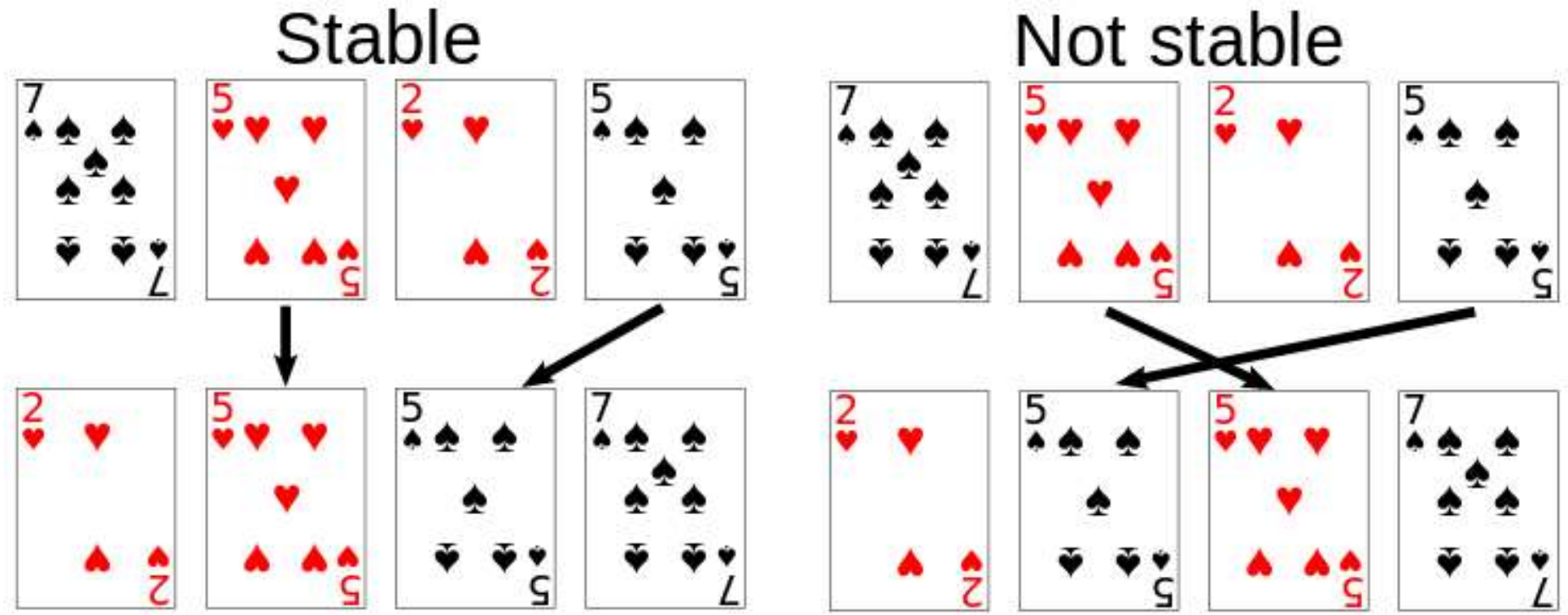
Comparación de algoritmos

Name	Best	Average	Worst	Memory	Stable
Bubble sort	n	n^2	n^2	1	Yes
Selection sort	n^2	n^2	n^2	1	No
Insertion sort	n	n^2	n^2	1	Yes
Merge sort	$n \log n$	$n \log n$	$n \log n$	worst case is n	Yes
In-place merge sort	—	—	$n (\log n)^2$	1	Yes
Quicksort	$n \log n$	$n \log n$	n^2	$\log n$ on average, worst case is n	typical in-place sort is not stable;
Heapsort	$n \log n$	$n \log n$	$n \log n$	1	No

Comparación de algoritmos



Comparación de algoritmos

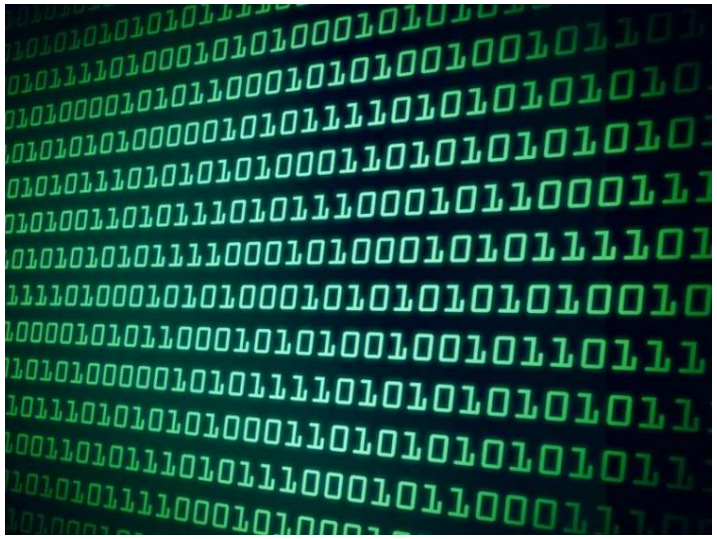


Conceptos de programación

Lenguajes de programación

- Lenguaje máquina
  - El único que entiende el ordenador
  - Basado en ceros y unos
  - Depende del hardware de la máquina
  - Muy difícil programar y depurar
  - En la práctica no se utiliza
  - Ejemplo: almacenar en el registro r1 la suma del registro r2 y 350 en el MIPS R3000 (procesador de la Play Station I)

0010000000100010 0000000101011110



Lenguajes de programación

- Lenguajes de alto nivel
  - Más cercanos al lenguaje natural
  - Independientes de la máquina
  - Requieren ser traducidos a lenguaje máquina
  - Más fácil programar, depurar…
  - Menos eficientes (la traducción no siempre es óptima)
  - Ejemplo de instrucción:

    a = b + 350

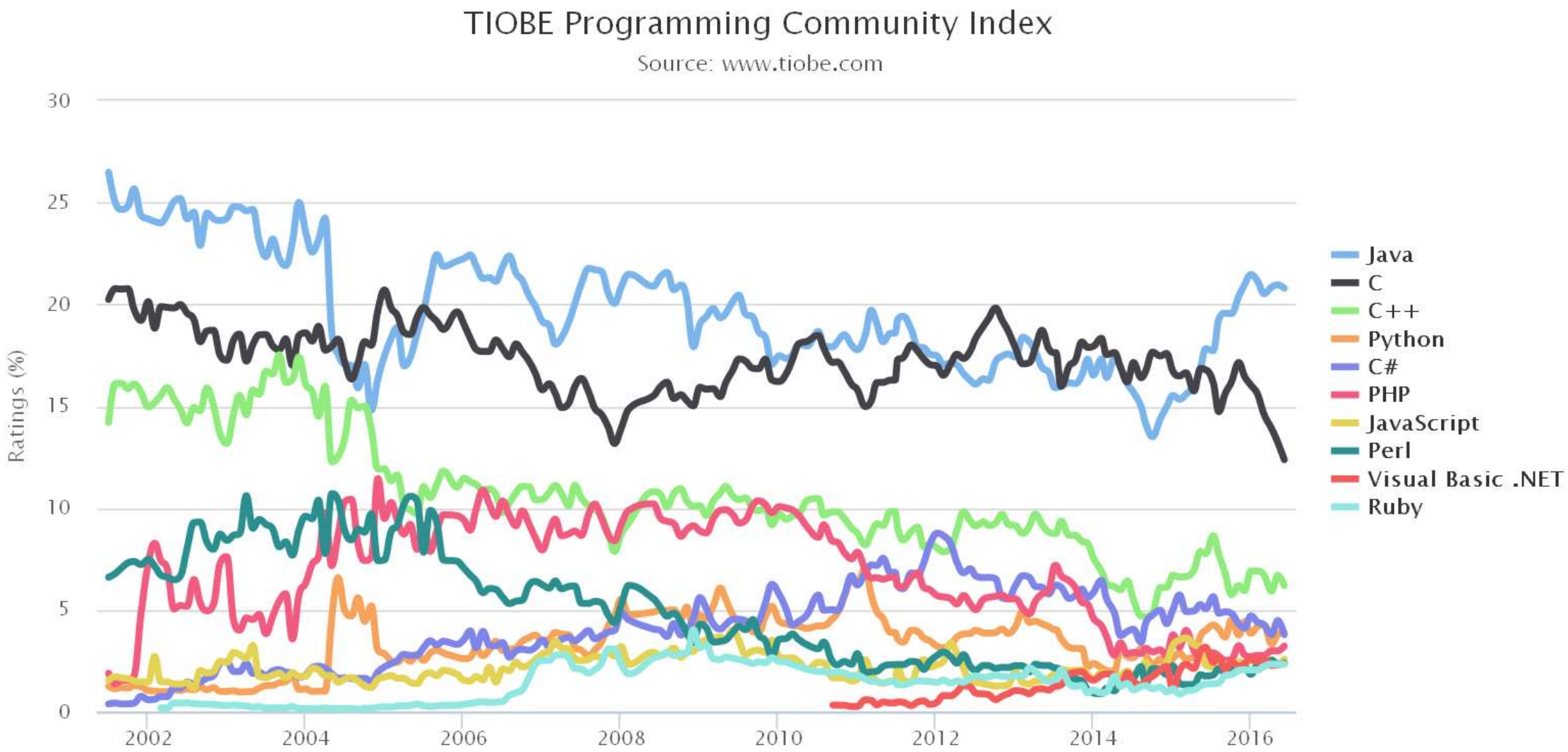
  - Ejemplos: Java, Processing, C, C++, Pascal, Python…

El lenguaje Java

- Probablemente, el lenguaje más popular
  - Datos de IEEE Spectrum, 2015
- Código independiente de SO, hardware...
  - Instalar máquina virtual en ordenador
- Facilita la creación de interfaces gráficas
- Muchas bibliotecas disponibles
  - Más código del que viene "con" Java
- Otros lenguajes se basan en Java
  - Ejemplo: Processing

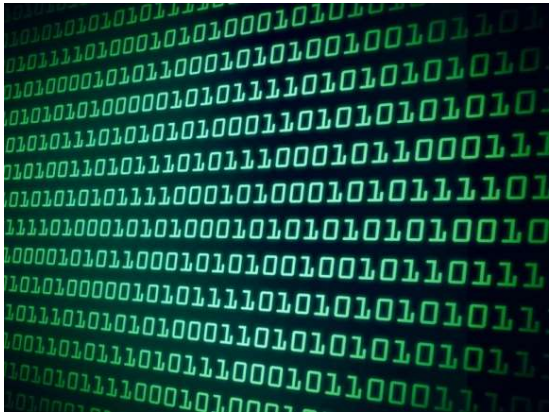
Language Rank	Types
1. Java	
2. C	
3. C++	
4. Python	
5. C#	
6. R	
7. PHP	
8. JavaScript	
9. Ruby	
10. Matlab	

El lenguaje Java



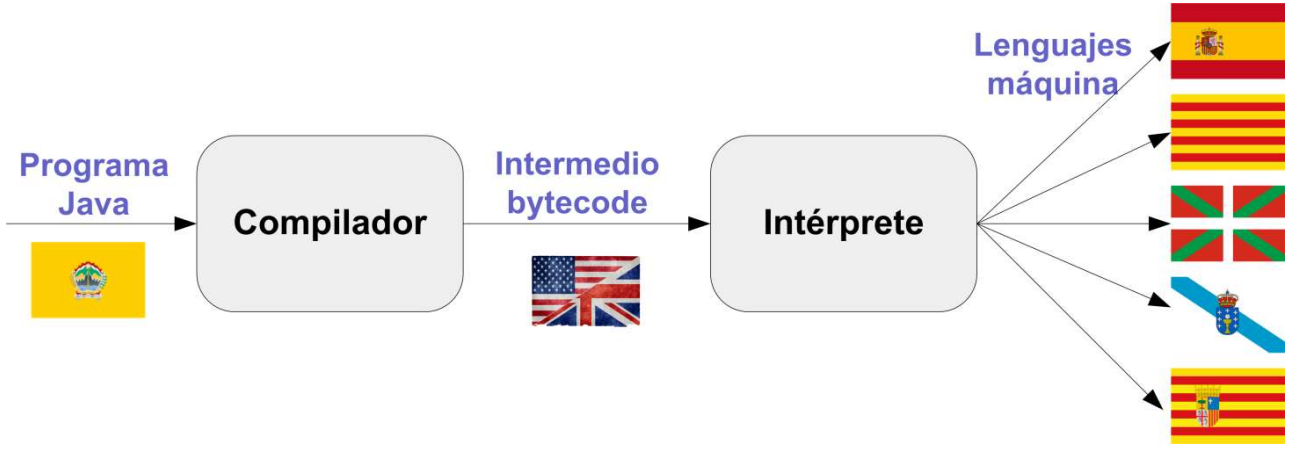
Código fuente y ejecutable

- Código fuente de un programa
  - Fichero de texto con las instrucciones que debe ejecutar un ordenador escritas en un lenguaje de programación,
  - Ha de ser traducido para generar un código ejecutable
- Código o programa ejecutable
  - Fichero binario con las instrucciones que debe ejecutar un ordenador escritas en lenguaje máquina



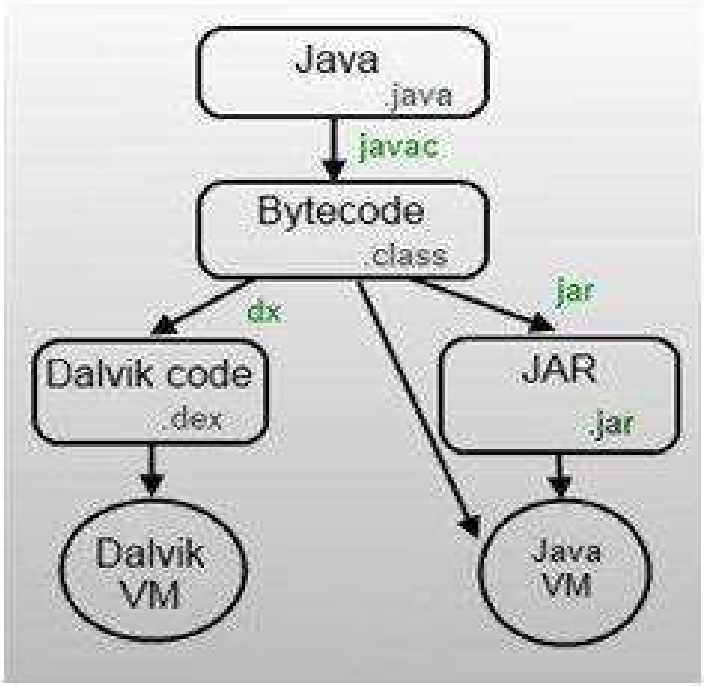
El intérprete de Java

- Compilador: software que traduce de un lenguaje a otro
  - Lo normal es traducir directamente a lenguaje máquina
  - Java: código universal independiente de máquina (*bytecode*)
  - La traducción permite detectar algunos errores
- Intérprete: traduce a lenguaje máquina "en directo", durante la ejecución del programa, teniendo en cuenta hardware, SO…
  - Pueden encontrarse errores en el momento de la ejecución
- La idea es compilar una vez y e interpretar muchas veces



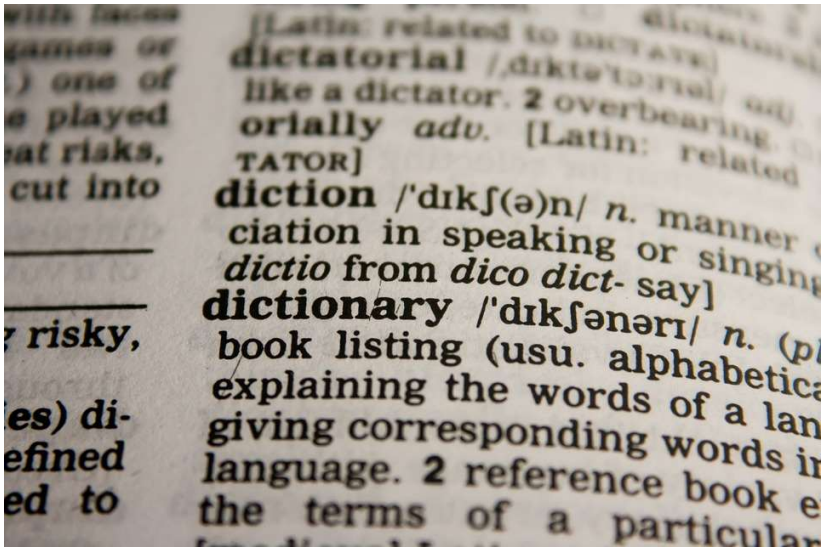
Android y Java

- Lenguaje de programación basado en Java
  - Subconjunto de Java SE
  - Máquina virtual propia: Dalvik
  - Formato dex: ejecutable para Dalvik



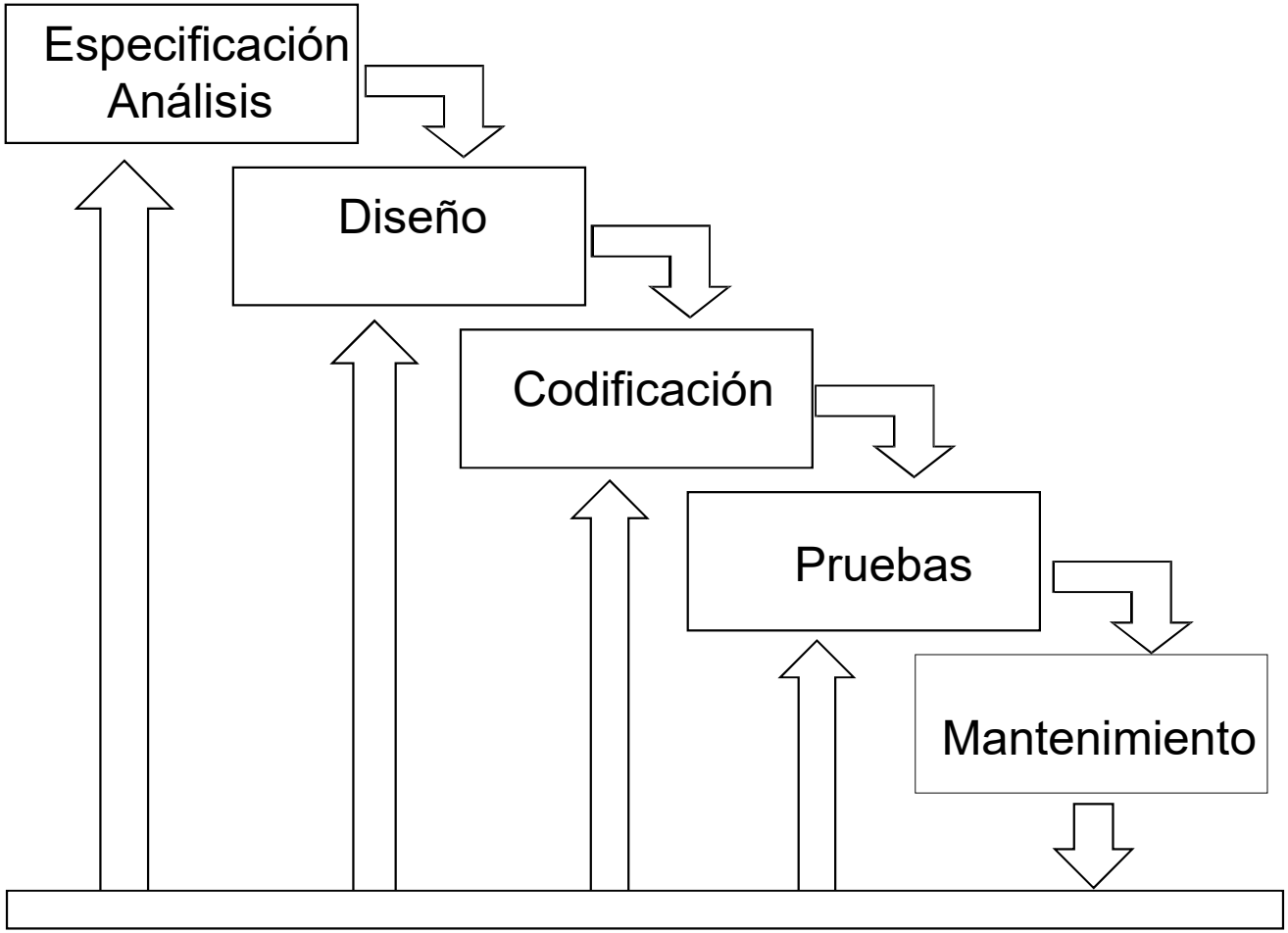
Tipos de errores

- **Niveles** del lenguaje:
  - Léxico: vocabulario, palabras
  - Sintáctico: reglas para formar frases
  - Semántico: significado de las frases
- **Errores** en tiempo de:
  - Compilación
    - Léxicos y sintácticos
    - Detectados por el compilador
  - Ejecución:
    - Semánticos
    - El programa no hace lo que debe
    - No detectados por el compilador



Tipos de errores

Proceso de resolución de problemas



Proceso de resolución de problemas

- Análisis del problema
- Diseño en papel de un algoritmo como solución
- Codificación en un lenguaje de programación (código fuente)
- Pruebas en un amplio conjunto de casos
  - Un programa hace lo que se ordena, no lo que se quiere
  - Un reloj parado da la hora exacta dos veces al día
  - Técnicas de depuración de errores
    - Traza en papel, reproducción de la ejecución paso a paso
    - Con software adecuado
- Mantenimiento (correcciones o actualizaciones)

Pensar antes de codificar

Prima Pensare, Poi Programmare, Perché Programme Poco Pensate Producono Puttanate

Entorno integrado de desarrollo

- Entorno integrado de desarrollo (IDE)
  - Editor de texto
    - Desarrollo de código fuente
  - Compilador
    - Traducir a código máquina/intermedio, crear ejecutables
  - Depurador
    - Detectar y corregir errores, ejecutar paso a paso…
  - Importar y exportar en diferentes formatos
  - …
- En prácticas:

Programación en Java

Agenda

- Un primer programa en Java
- Datos 1: variables y constantes
- Datos 2: tipos de datos primitivos
- Instrucciones 1: entrada y salida
- Instrucciones 2: sentencias
- Instrucciones 3: modularidad
- Datos 3: vectores y matrices
- Introducción a orientación a objetos
- Introducción a orientación a eventos

¡Esto no es curso para aprender a programar en Java!

Un primer programa en Java

- Es un programa muy sencillo que únicamente escribe por pantalla el mensaje "Hola Mundo" (*"Hello world"* en inglés)
  - Disclaimer: la traducción al español no es muy buena
  - Por costumbre, es el primer programa en cada lenguaje
  - Los diferentes programas "Hola Mundo" dan una idea sobre los lenguajes: expresividad, legibilidad, dificultad…

Programa Hola Mundo

```
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println("¡Hola mundo!");
    }
}
```

- Por ahora, pensemos en una clase como una estructura que permite guardar datos e instrucciones (subprogramas)
- Entre todos esos subprogramas, hay un programa principal (`main`) que es el que se ejecutará en primer lugar
- Las instrucciones del `main` determinarán qué hacer después
- La única instrucción del ejemplo permite escribir por pantalla
  - Si el contenido a escribir (indicado entre los paréntesis) está entre comillas, se escribe literalmente

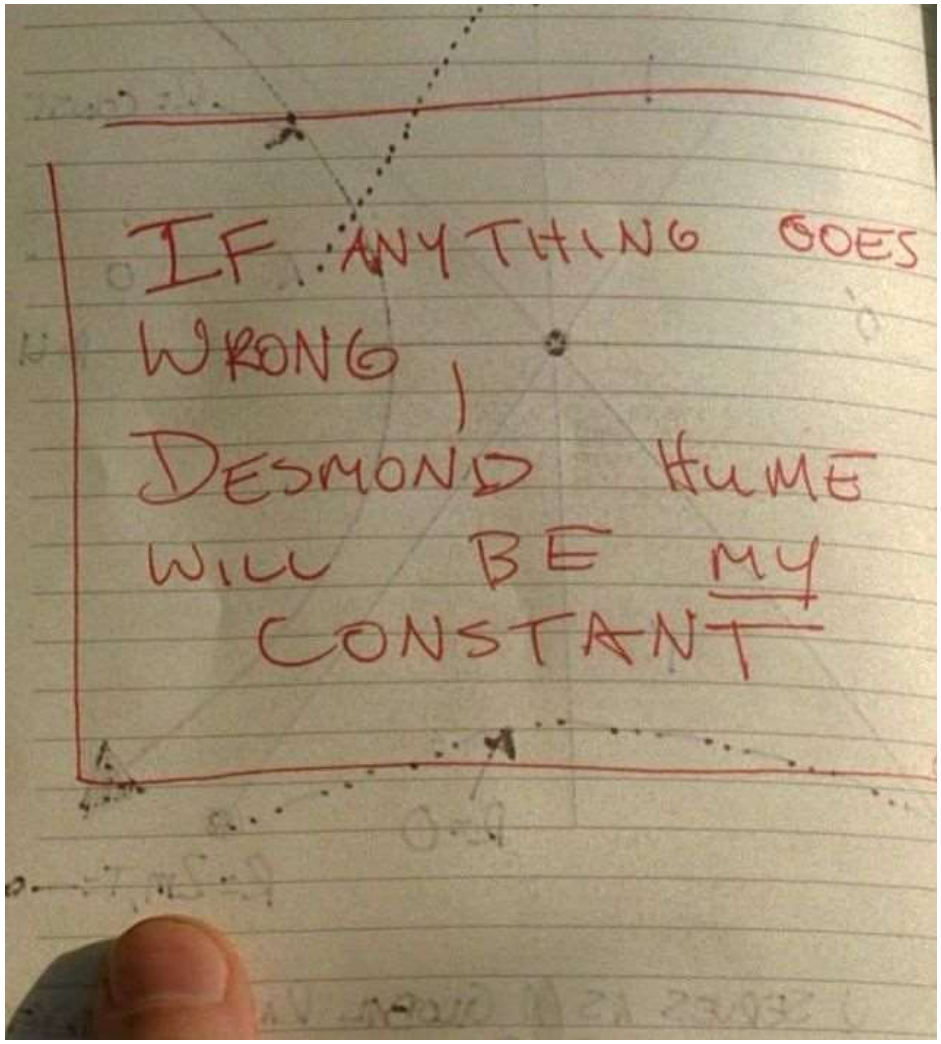
Clases y main

```
public class HolaMundo {
    public static void main(String[] args) {
        ...
    }
}
```

- Por ahora, todos nuestros programas comenzarán así
  - Simplemente completaremos el `main` con instrucciones
  - ! El nombre de la clase puede variar
- La clase irá en un fichero llamado como ella y extensión `.java`
- Veremos en prácticas que al crear un nuevo fichero, aparece automáticamente una plantilla así para rellenarla

Variables y constantes

- Un algoritmo considera
  - Datos
  - Instrucciones
- ¿Cómo podemos representar los datos en un programa?
- Dos mecanismos:
  - Variables, valor alterable
  - Constantes, valor inalterable

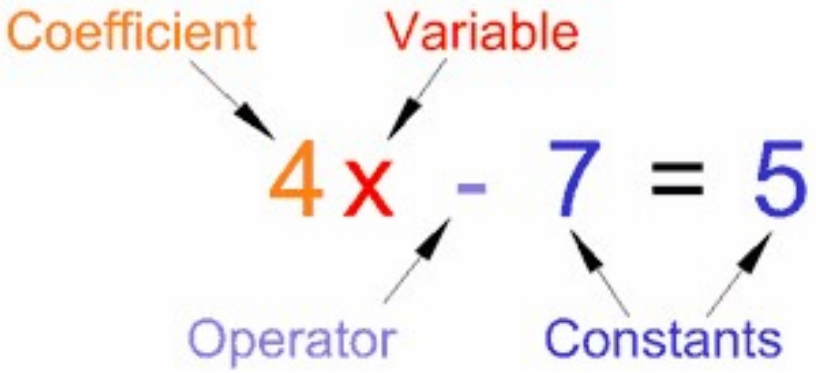


Variable en matemáticas

- Símbolo que representa un elemento no especificado de un conjunto dado
  - Dicho conjunto se denomina dominio de la variable
  - Cada elemento es un valor posible de la variable
- Ejemplo: sea x una variable cuyo universo es el conjunto

  {1, 3, 5, 7, 9, 11, 13}

  x puede reemplazarse por cualquier entero positivo impar ≤ 14

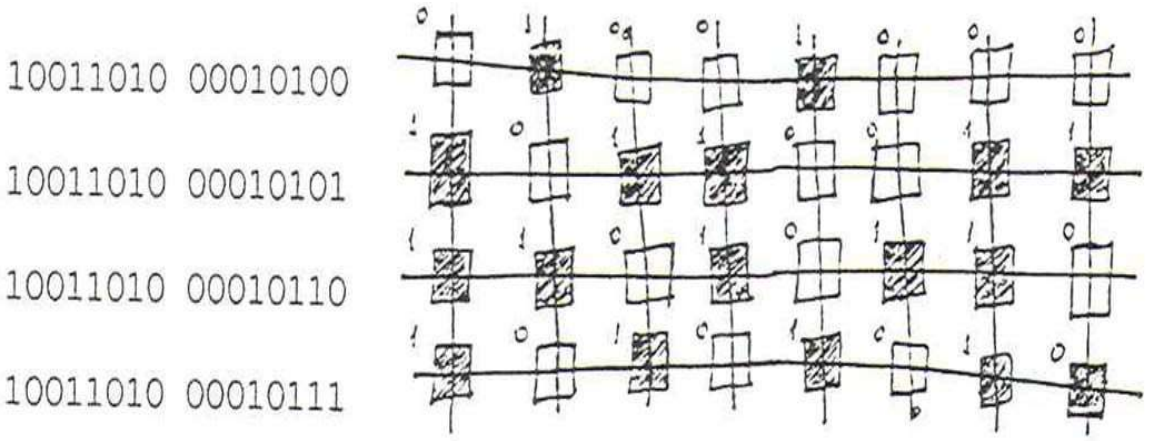


Variables y constantes en informática

- Para acceder un dato hay que conocer
  - Cuántas celdas de memoria lo almacenan
  - Qué celdas lo almacenan (la dirección de memoria)
  - Cómo se representa el dato en las celdas
- Solución: abstracción de una o varias celdas de memoria accediendo al dato representado con un nombre simbólico
- El ordenador se encarga de gestionar:
  - Acceso a memoria con la dirección física de las celdas
  - Representación interna de los valores

Variable en informática

- Concepto físico de variable: corresponde a un área reservada en la memoria principal, de longitud fija o variable



- Concepto lógico de variable: se representa utilizando un identificador (nombre) que hace referencia a un dato en memoria
  - Por ejemplo, “temperatura”

Variable en informática

- Variable: dato de valor alterable durante la ejecución con:
  - Nombre simbólico (identificador): bautiza una zona de memoria y referencia la variable facilitando la legibilidad
  - Tipo de dato, que determina los posibles valores
  - Valor, alterable pero único
- El nombre y el tipo de dato son permanentes, pero el valor puede cambiar durante la ejecución de un programa
  - Eso sí, siempre dentro del dominio de valores permitidos
- Declaración de variables: indicar sus nombre y sus tipos
  - Solo una vez, (típicamente) al principio de los programas
  - Ejemplo: `double temperatura` podrá guardar n^{os} reales

Ejemplo: paso de Celsius a Fahrenheit

```
import java.util.*;

public class CelsiusFahrenheit {

  public static void main(String[] args) {
     double Celsius, Fahrenheit;
     System.out.print("Introducir la cantidad de grados Celsius: ");
     Scanner escaner = new Scanner(System.in);
     Celsius = escaner.nextDouble();
     Fahrenheit = (9.0 / 5.0) * Celsius + 32;
     System.out.println("Cantidad en grados Fahrenheit: " + Fahrenheit );
    }
}
```

Plantilla básica de todo programa Java

Ejemplo: paso de Celsius a Fahrenheit

```
import java.util.*;

public class CelsiusFahrenheit {

  public static void main(String[] args) {
     double Celsius, Fahrenheit;
     System.out.print("Introducir la cantidad de grados Celsius: ");
     Scanner escaner = new Scanner(System.in);
     Celsius = escaner.nextDouble();
     Fahrenheit = (9.0 / 5.0) * Celsius + 32;
     System.out.println("Cantidad en grados Fahrenheit: " + Fahrenheit );
   }
}
```

Declaración de variables de tipo real

Ejemplo: paso de Celsius a Fahrenheit

```
import java.util.*;

public class CelsiusFahrenheit {

  public static void main(String[] args) {
    double Celsius, Fahrenheit;
    System.out.print("Introducir la cantidad de grados Celsius: ");
    Scanner escaner = new Scanner(System.in);
    Celsius = escaner.nextDouble();
    Fahrenheit = (9.0 / 5.0) * Celsius + 32;
    System.out.println("Cantidad en grados Fahrenheit: " + Fahrenheit );
   }
}
```

Escritura por pantalla de un mensaje informativo para el usuario

Ejemplo: paso de Celsius a Fahrenheit

```
import java.util.*;

public class CelsiusFahrenheit {

  public static void main(String[] args) {
    double Celsius, Fahrenheit;
    System.out.print("Introducir la cantidad de grados Celsius: ");
    Scanner escaner = new Scanner(System.in);
    Celsius = escaner.nextDouble();
    Fahrenheit = (9.0 / 5.0) * Celsius + 32;
    System.out.println("Cantidad en grados Fahrenheit: " + Fahrenheit );
   }
}
```

Lectura de datos desde teclado: se lee un nº real y se guarda en `Celsius`

Para poder usar `Scanner`, es preciso importar el lugar donde se define

Ejemplo: paso de Celsius a Fahrenheit

```
import java.util.*;

public class CelsiusFahrenheit {

  public static void main(String[] args) {
    double Celsius, Fahrenheit;
    System.out.print("Introducir la cantidad de grados Celsius: ");
    Scanner escaner = new Scanner(System.in);
    Celsius = escaner.nextDouble();
    Fahrenheit = (9.0 / 5.0) * Celsius + 32;
    System.out.println("Cantidad en grados Fahrenheit: " + Fahrenheit );
   }
}
```

Instrucción que permite calcular el valor de una variable con una fórmula

Ejemplo: paso de Celsius a Fahrenheit

```
import java.util.*;

public class CelsiusFahrenheit {

  public static void main(String[] args) {
    double Celsius, Fahrenheit;
    System.out.print("Introducir la cantidad de grados Celsius: ");
    Scanner escaner = new Scanner(System.in);
    Celsius = escaner.nextDouble();
    Fahrenheit = (9.0 / 5.0) * Celsius + 32;
    System.out.println("Cantidad en grados Fahrenheit: " + Fahrenheit );
  }
}
```

Escritura por pantalla del resultado final

Literales y constantes

- Literal: valor concreto del dominio de un tipo simple o cadena
  - Ejemplos: `9.0`, `5.0`, `32`, `"Cantidad en grados Fahrenheit: "`
- Uso directo de las constantes (se escribe el valor directamente)
  - Los lenguajes tienen reglas para inferir el tipo
    - Si tiene un punto, es un número real
    - Si son dígitos numéricos sin punto, es un entero
    - Si tiene comillas, es una cadena de caracteres
    - Si tiene comillas simples, es un carácter
- Uso simbólico de las constantes
  - Podemos darles un nombre usando la palabra `final`
  - Ejemplo: `final double PI = 3.14159;`

Identificadores

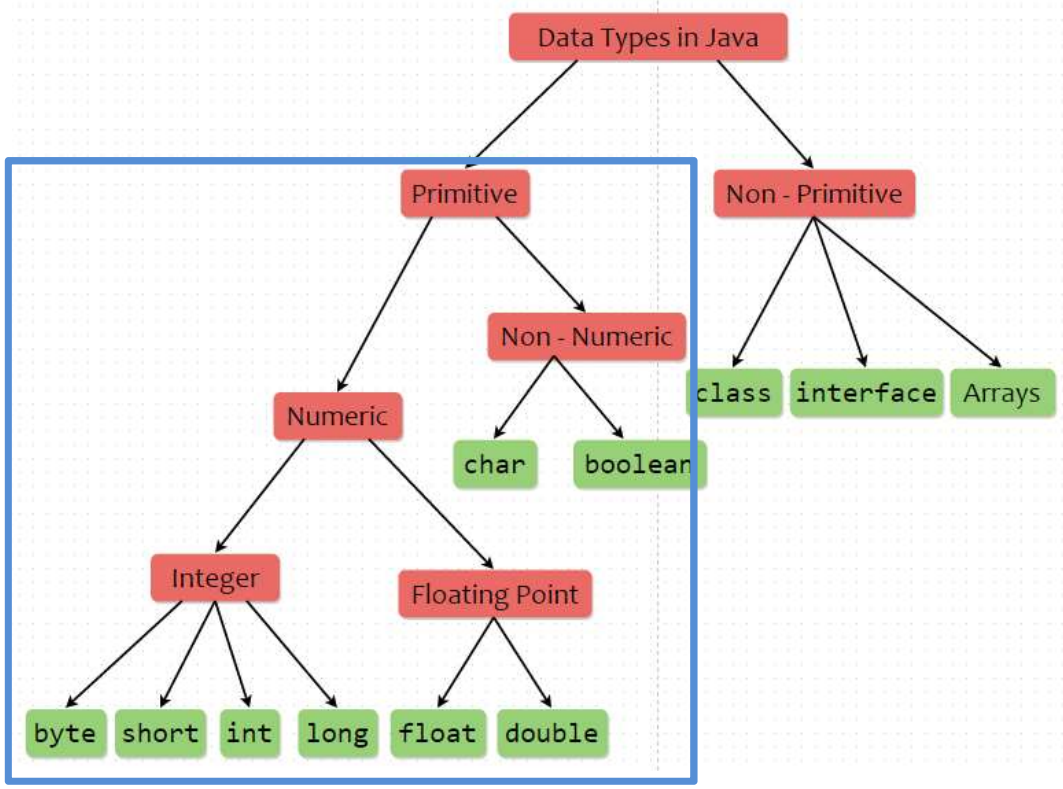
- Al bautizar a un ser humano no valen todos los nombres
- Identificador:
  - Nombre definido por el programador para una variable, constante, clase u otros elementos de un programa
  - Secuencia sin espacios que comienza por una letra y contiene letras, números y subrayado (_)
  - Ejemplos: `temperaturaCelsius`, `temperatura_celsius`...
  - Usar nombres mnemotécnicos relacionados con el objeto
- Palabra reservada:
  - No puede ser utilizada como identificador por tener un significado especial
- Ambos son sensibles a las mayúsculas y minúsculas

Palabras reservadas

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

Tipos de datos primitivos

- Tipos de datos primitivos en Java
  - Enteros
    - `byte`, `short`, `int`, `long`
  - Reales
    - `float`, `double`
  - Caracteres
    - `char`
  - Booleanos
    - `boolean`



Tipo de dato entero

Nombre del tipo	Tamaño en memoria	Rango de valores
`byte`	1 B = 8 bits	[-128, 127] = $[-2^7, 2^7-1]$
`short`	2 B = 16 bits	[-32.768, 32.767] = $[-2^{15}, 2^{15}-1]$
`int`	4 B = 32 bits	[-2.147$_1$483.648, 2.147$_1$483.647] = $[-2^{31}, 2^{31}-1]$
`long`	8 B = 64 bits	[-9$_3$223.372$_2$036.854$_1$775.808, 9$_3$223.372$_2$036.854$_1$775.807] = $[-2^{63}, 2^{63}-1]$

Tipo de dato entero

- Operadores:
  - `+` suma
  - `-` resta (operador binario) / cambio de signo (unario)
  - `*` multiplicación
  - `/` división (el cociente es un entero)
  - `%` resto de la división
  - `++` incremento en una unidad
  - `--` decremento en una unidad

```
x++ : x = x + 1
x-- : x = x - 1
```

- Ejemplos:
  - `int edad = 18;`
  - `int saldo = -1;`

Errores comunes

- Desbordamiento: si sobrepasamos el valor máximo que se puede representar con un tipo entero, se representará un valor incorrecto sin que nos avise de ninguna manera
- División entre cero: produce un error en tiempo de ejecución

Tipo de dato real

Nombre del tipo	Tamaño en memoria	Precisión aproximada	Rango de valores
`float`	4 B = 32 bits	7 dígitos	[-3.4E38, 3.4E38]
`double`	8 B = 64 bits	17 dígitos	[-1.7E308, -4.9E-324] y [9E-324, +1.7E308]

- El nombre `double` se debe a tener doble precisión
- El nombre `float` se debe a que la coma es flotante
  - Ejemplo: 123.45 se representa como $12345 \cdot 10^{-3}$
  - Representación en mantisa y exponente
  - Facilita representar n^{os} grandes y hacer cálculos

Tipo de dato real

- Ejemplos:
  - `double celsius = 21.5;`
  - `double celsius = 2.15E1;`
- Operadores:
  - Los mismos aritméticos de los enteros excepto el resto, pero ahora el resultado es un número real
  - Si intentamos dividir entre 0, obtenemos `NaN`, un valor especial que representa “Not a Number”
  - Si el resultado excede la capacidad de representación, se obtiene `Infinity` o `-Infinity`

Funciones matemáticas

Método	Resultado
`Math.abs(x)`	valor absoluto de x
`Math.sin(a)`	seno del ángulo a en radianes
`Math.cos(a)`	coseno del ángulo a en radianes
`Math.tan(a)`	tangente del ángulo a en radianes
`Math.asin(r)`	ángulo cuyo seno es r
`Math.acos(r)`	ángulo cuyo coseno es r
`Math.atan(r)`	ángulo cuya tangente es r
`Math.atan2(a, b)`	ángulo cuya tangente es a/b
`Math.exp(x)`	e elevado a x
`Math.pow(x, y)`	x elevado a y
`Math.log(x)`	logaritmo natural de x
`Math.sqrt(x)`	raíz cuadrada de x
`Math.ceil(a)`	redondea un número real hacia arriba
`Math.floor(a)`	redondea hacia abajo (parte entera)
`Math.round(x)`	redondea al entero más cercano
`Math.max(a, b)`	máximo de a y b
`Math.min(a, b)`	mínimo de a y b
`Math.E`	Número de Euler
`Math.Pi`	Número π

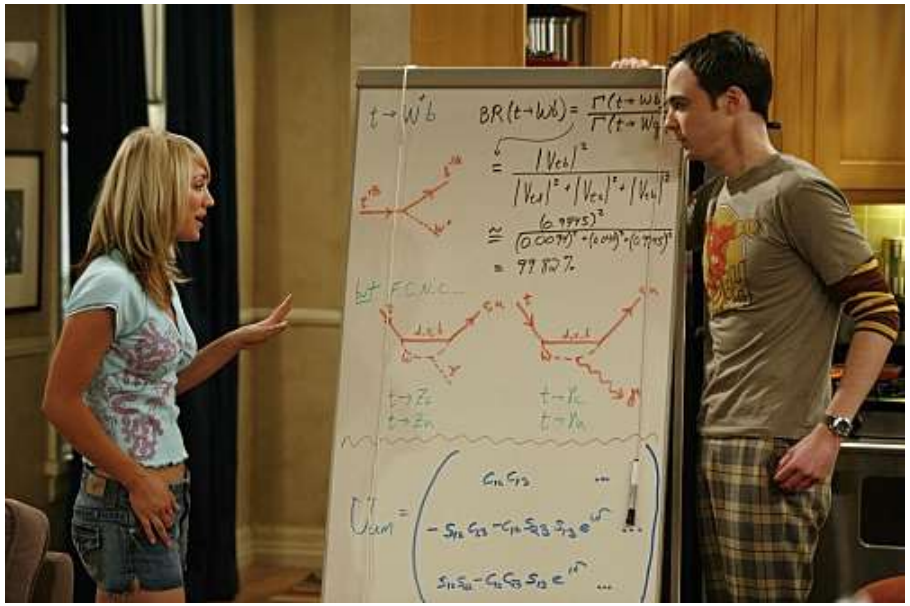
Tipo de dato carácter

Nombre del tipo	Tamaño en memoria	Rango de valores
`char`	2 B	Caracteres del conjunto Unicode

- Ejemplos
  - Letras minúsculas: `"a"`, `"á"`, `"à"`…
  - Letras mayúsculas: `"A"`, `"Á"`, `"À"`…
  - Caracteres numéricos: `"0"`, `"1"`…
  - Caracteres no alfabéticos: `","`, `":"`, `";"`…
  - Salto de línea `"\n"`
  - Tabulador `"\t"`
  - Espacio `" "`

Tipo de dato String

- `String`: tipo de dato que permite representar cadenas de caracteres, es decir, listas de caracteres de tamaño variable
  - Útil para escribir mensajes al usuario por pantalla
- Ejemplo
  - `String nombre = "Fernando"`



Tipo de dato booleano

Nombre del tipo	Tamaño en memoria	Rango de valores
`boolean`	1 B	{`false`, `true`}

Es decir, verdadero o falso

- Operadores
  - `&&` Conjunción (y, *and*)
  - `||` Disyunción (o, *or*)
  - `!` Negación (no, *not*)
  - `^` ~~O exclusivo (*xor*)~~

Cuidado con el lenguaje natural

Tipo de dato booleano

Pregunta	Respuesta
¿Es -1 par Y positivo?	`false`
¿Es -2 par Y positivo?	`false`
¿Es 1 par Y positivo?	`false`
¿Es 2 par Y positivo?	`true`
¿Es -1 par O positivo?	`false`
¿Es -2 par O positivo?	`true`
¿Es 1 par O positivo?	`true`
¿Es 2 par O positivo?	`true`
¿Es 2 NO positivo?	`false`

Tipo de dato booleano

- Tablas de verdad:

a	b	a && b	a || b	!a
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

andres antonietti mayoral ▸ Programadores

Se abre el ascensor y hay un programador dentro, le preguntan:

- ¿Sube o baja?

A lo que el programador responde:

- SI.

Tipo de dato booleano

- Operadores relacionales: comparan dos valores (no booleanos) y devuelven un resultado booleano

`<`	Menor estrictamente que
`<=`	Menor o igual que
`>`	Mayor estrictamente que
`>=`	Mayor o igual que
`==`	Igual que
`!=`	Distinto que

- Ejemplos:
  - `2 + 3 <= 4` devuelve `false`; `2 + 3 != 4` devuelve `true`

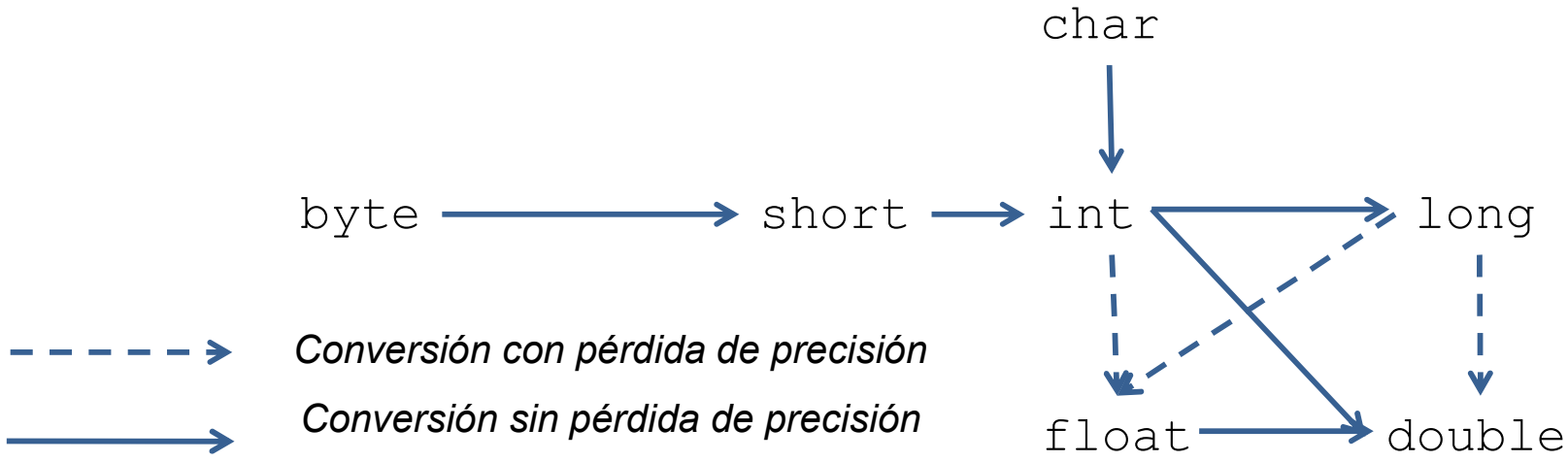
Expresiones

- Expresión informática: cadena de caracteres del lenguaje informático, cuyo alfabeto tiene como símbolos:
  - Números
  - Operadores
  - El alfabeto latino
  - Signos de puntuación
  - ...
- Similares a las matemáticas, solo que estas también pueden incluir alfabeto griego o símbolos propios, e.g., $\int, \sqrt{}$...
- Ejemplo:
  - `Fahrenheit = (9.0 / 5.0) * Celsius + 32`

Compatibilidad de tipos

- *Casting*: convertir un elemento de un tipo a otro
- Los elementos de una expresión deberían tener igual tipo
  - Si hay enteros y reales, Java transforma a reales
  - Si hay varios tipos de enteros/reales, al de más capacidad
  - El programador también puede hacerlo

    Ejemplo: `int resultado = (int) 9.5;`

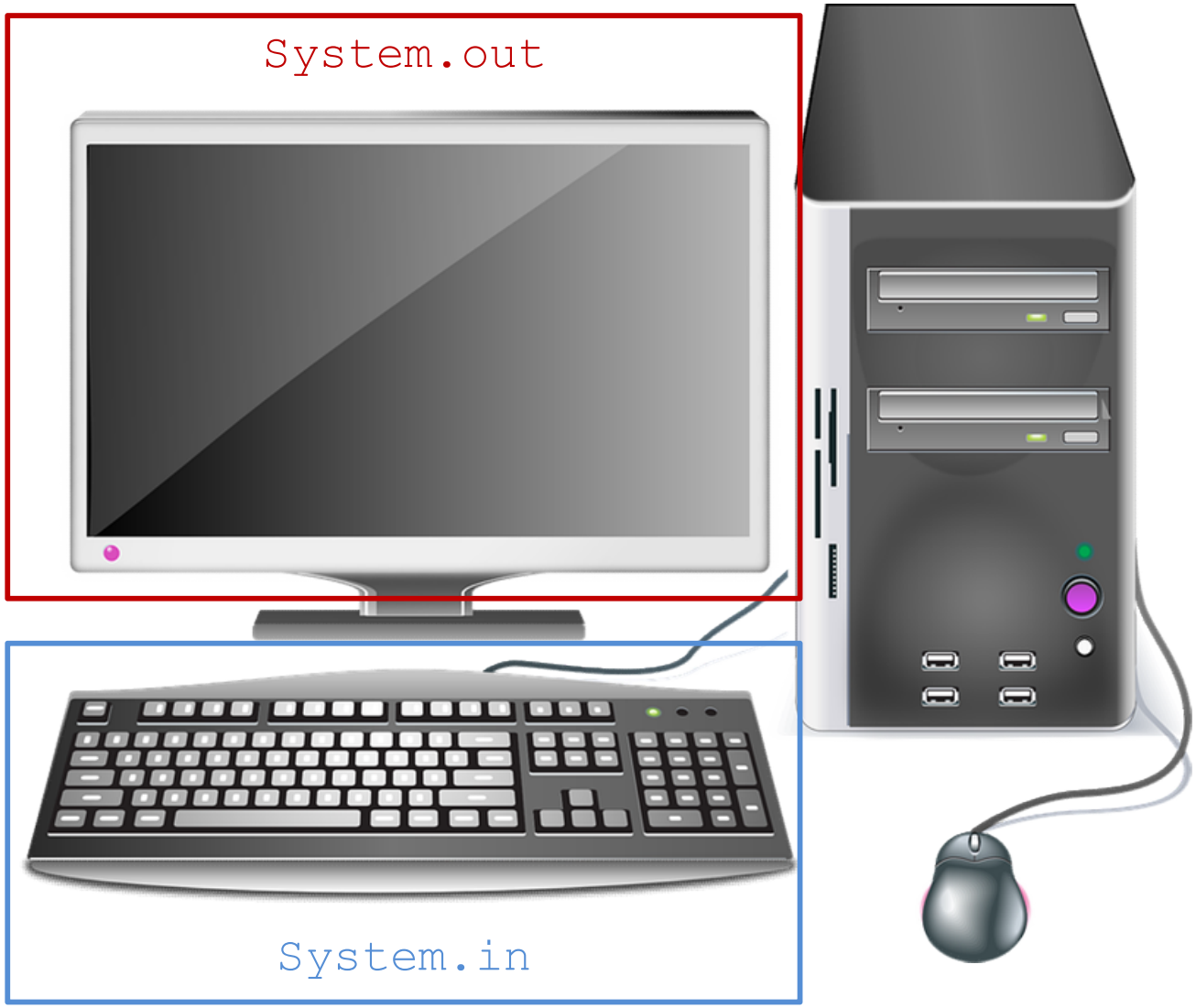


Prioridad de los operadores

- Al igual que en matemáticas, los operadores tienen distinta prioridad (a menor número, mayor prioridad)
  - Para alterar esta prioridad se usan paréntesis
- Niveles de prioridad en Java

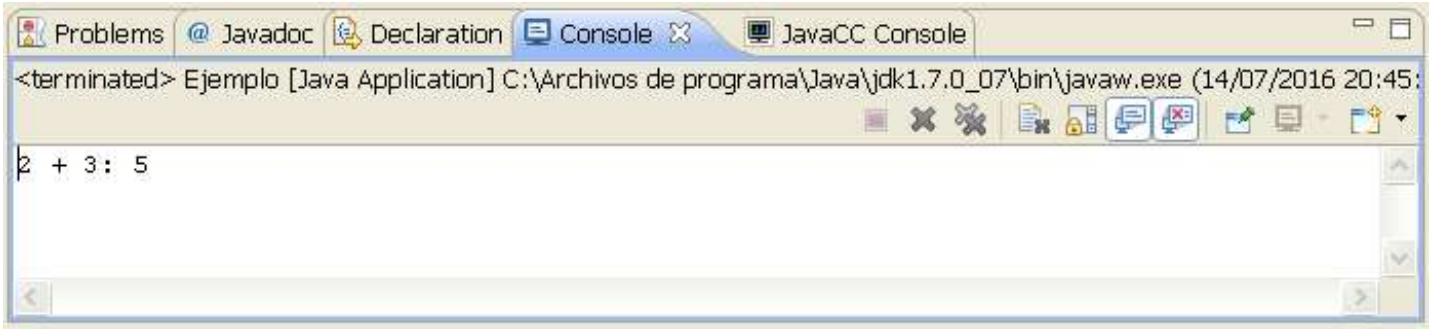
1. `++  --`
2. `!`
3. `*  /  %`
4. `+  -`
5. `<  >  <=  >=`
6. `==  !=`
7. `&&`
8. `||`
9. `=`

Entrada y salida de datos



Salida de datos

- `System.out.println(...)`: escribe algo por pantalla y añade al final un salto de línea (futuras escrituras irán a otra línea)
- `System.out.print(...)`: escribe por pantalla sin salto de línea
- Si en los paréntesis hay varias cosas para escribir, estarán separadas por un operador de concatenación `+`
- Si las cosas a escribir van entre comillas, se consideran cadenas de caracteres y se escriben literalmente
- Si no, se escribe el resultado de evaluar la expresión
- Ejemplo: `System.out.println("2 + 3: " + (2 + 3) );`



Entrada de datos

- Hay que crear un escáner para leer de la entrada estándar

```
Scanner escaner = new Scanner(System.in);
```

- Para ello, hay que importar la biblioteca `java.util`

```
import java.util.*;
```

- Un modo de dar valor a las variables (como la asignación)

```
double Celsius = escaner.nextDouble();
```

  - Ahí se detiene la ejecución del programa hasta que el usuario escriba un valor `double` por teclado y pulse Intro

- Conviene avisar previamente al usuario con un mensaje informativo para que sepa qué tipo de valor debe introducir

```
System.out.print("Introducir la temperatura: ");
```

Entrada de datos

Método	Resultado
`nextInt()`	`int`
`nextFloat()`	`float`
`nextDouble()`	`double`
`nextLine()`	`String`
`nextBoolean()`	`boolean`
`nextByte()`	`byte`
`nextShort()`	`short`
`nextLong()`	`long`

No hay `nextChar`

Comentarios

- Comentarios: fragmentos escritos en lenguaje natural que son ignorados por el ordenador pero que facilitan al humano que lo lea entender lo que quiso hacer el programador
- Ejemplos:
  - `// Comentario hasta el fin de la línea actual`
  - `/* Comentario que comienza aquí,`
    `ocupa varias líneas,`
    `hasta encontrar el final */`

Ejemplo.java

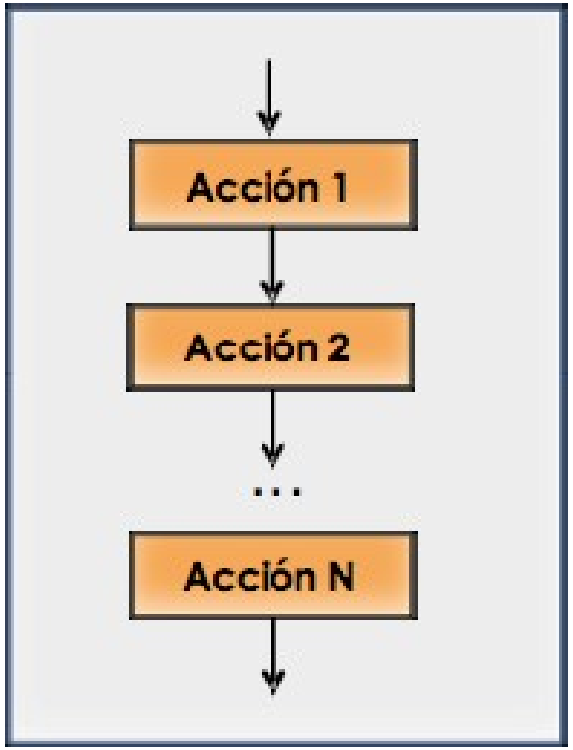
```
1 public class Ejemplo {
2     public static void main(String[] args) {
3         // Comentario hasta el fin de la línea actual
4
5         /* Comentario que comienza aquí,
6            ocupa varias líneas,
7          hasta encontrar el final */
8     }
9 }
```

Instrucciones de un programa

- Para resolver un problema de programación, identificar:
  - Variables necesarias
  - Acciones que componen la secuencia
  - Orden (parcial/total) de ejecución de las acciones
- Teorema de la programación estructurada: "Todo programa puede ser escrito utilizando 3 tipos de estructuras de control"
  - Estructura secuencial
  - Estructura condicional (o de selección)
  - Estructura iterativa (o de repetición

Estructura secuencial

- Las acciones son ejecutadas una después de otra, en el mismo orden en el que han sido escritas



Ejemplo en Java:

```
double Celsius, Fahrenheit;
System.out.print("Introducir la cantidad de Celsius: ");
Scanner escaner = new Scanner(System.in);
Celsius = escaner.nextDouble();
Fahrenheit = (9.0 / 5.0) * Celsius + 32;
System.out.println("Cantidad en Fahrenheit: " + Fahrenheit );
```

El carácter ; indica el fin de una instrucción

Asignación

- Es otra manera de dar valor a una variable
  - `Fahrenheit = (9.0 / 5.0) * Celsius + 32.0`
- El operador en Java es `=`
  - ! No confundirlo con `==`
- Asocia un nuevo valor (el de la expresión a la derecha del operador) a una variable (la de la izquierda)
- En matemáticas, $x = x + 1$ indica que los dos valores a la izquierda y a la derecha del = son iguales
- En informática, `x = x + 1` indica que se debe evaluar `x + 1` y guardar el resultado en `x` (se incrementa su valor en uno)

Ejemplo: intercambiar el valor de 2 variables

```
public class Intercambio {
   public static void main(String[] args) {
        int a = 1;
        int b = a + 1;
        a = 3;
        System.out.println("a " + a);
        System.out.println("b " + b);
        b = a;
        a = b;
        System.out.println("a " + a);
        System.out.println("b " + b);
   }
}
```

```
a = 3
b = 2
```

Al tomar `a` el valor 3, `b` **NO** pasa a valer 3+1=4

Ejemplo: intercambiar el valor de 2 variables

```
public class Intercambio {
   public static void main(String[] args) {
        int a = 1;
        int b = a + 1;
        a = 3;
        System.out.println("a " + a);
        System.out.println("b " + b);
        b = a;
        a = b;
        System.out.println("a " + a);
        System.out.println("b " + b);
   }
}
```

```
a = 3
b = 2
```

```
a = 3
b = 3
```

No funciona: asigna a `b` el valor nuevo de `a` en vez de el anterior

Ejemplo: intercambiar el valor de 2 variables

```
public class Intercambio {
   public static void main(String[] args) {
        int a = 1;
        int b = a + 1;
        a = 3;
        System.out.println("a " + a);
        System.out.println("b " + b);
        int copia = a;
        a = b;
        b = copia;
        System.out.println("a " + a);
        System.out.println("b " + b);
   }
}
```

```
a = 3
b = 2
```

```
a = 2
b = 3
```

Asignaciones especiales

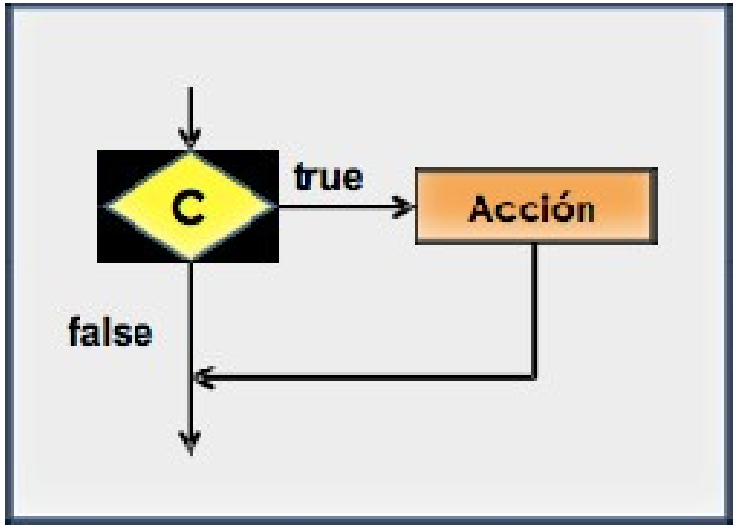
Asignación abreviada	Equivalencia
`x += n`	`x = x + n`
`x -= n`	`x = x - n`
`x *= n`	`x = x * n`
`x /= n`	`x = x / n`
`x %= n`	`X = x % n`
`x = ++y`	`y = y + 1` `x = y`
`x = y++`	`x = y` `y = y + 1`

Estructura condicional

- Frecuentemente se plantean problemas cuya resolución exige la consideración de casos particulares, cada uno de los cuales requiere una estrategia de resolución diferente

Estructura condicional simple

- La acción/es sólo se ejecuta/n si se cumple una condición
- Después, en ambos casos, se continúa el resto del programa



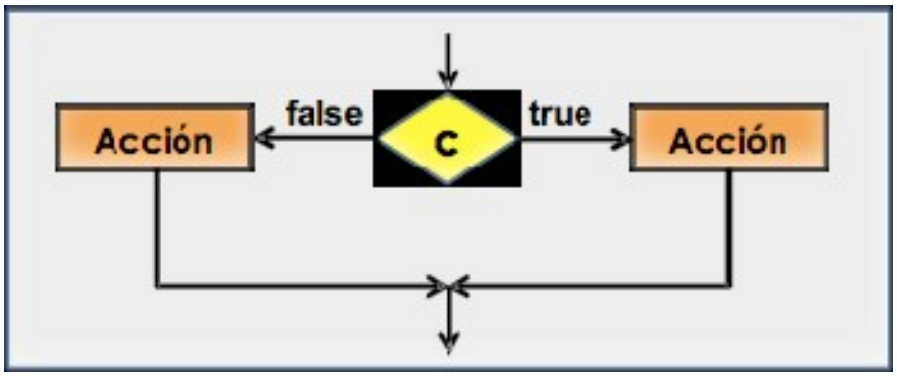
Ejemplos en Java:

```
if (numero >= 0)
    System.out.println("Es entero positivo");

if ((numero >= 0) && (numero < 10)) {
    System.out.println("Es entero positivo");
    System.out.println("Es menor que 10");
}
```

Las llaves { } son obligatorias si hay más de una instrucción

Estructura condicional doble

- Si se cumple una condición se ejecuta una acción, si no se cumple se ejecuta otra, y después continúa la parte común



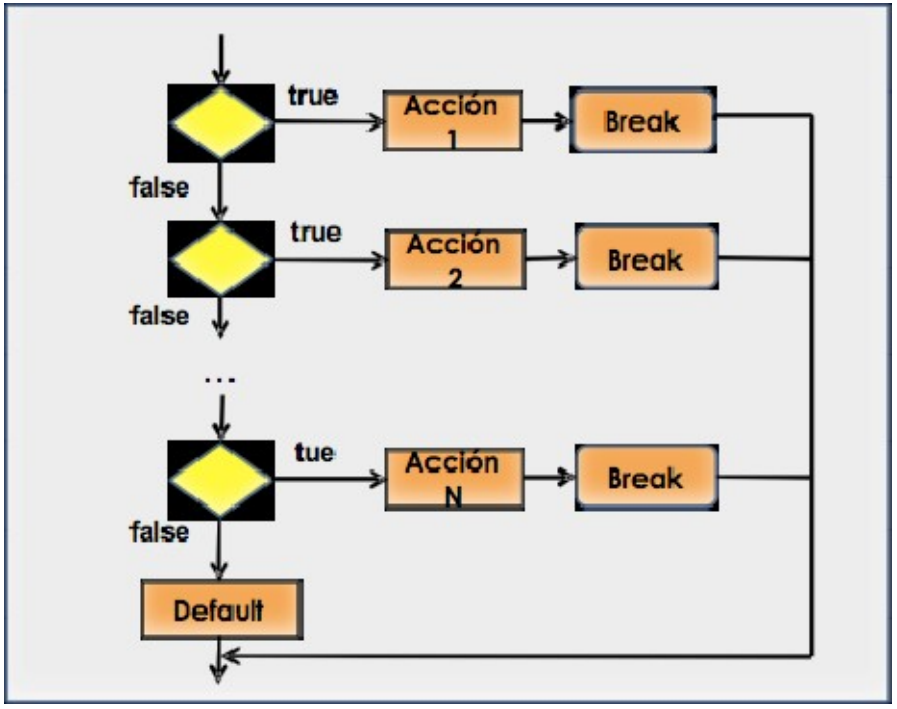
Ejemplos en Java:

```
if (numero >= 0)
   System.out.println("Es entero positivo");
else
   System.out.println("Es entero negativo");
```

{ } si hay más de una instrucción

Estructura condicional múltiple

- Se evalúa una expresión no booleana y se especifica una acción diferente para cada posible valor, e incluso una acción por defecto si el valor es diferente a los especificados

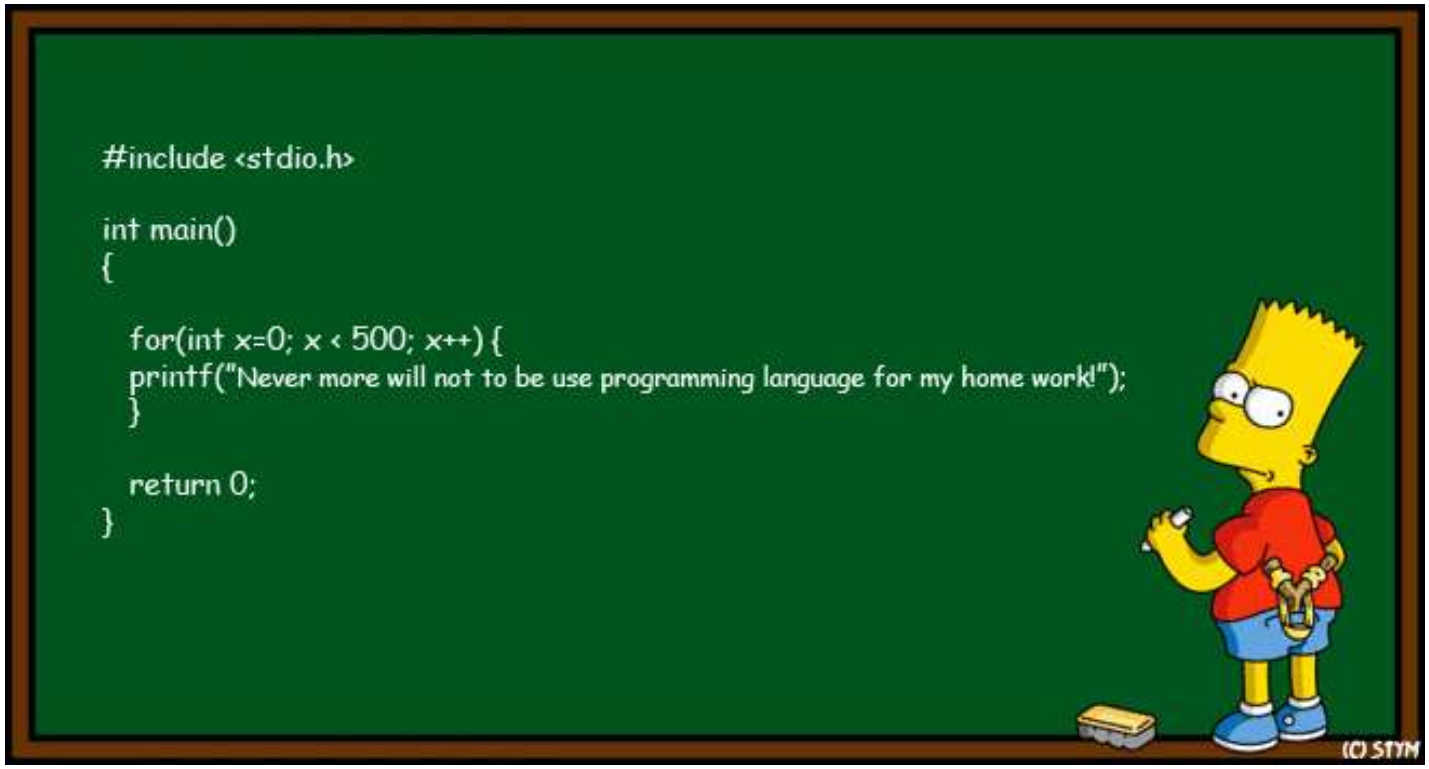


Ejemplos en Java:

```
switch (diaSemana)
{
    case 6:
        System.out.println("Es sábado");
        break;
    case 7:
        System.out.println("Es domingo");
        break;
    default:
      System.out.println("Es laborable");
}
```

Estructura repetitiva

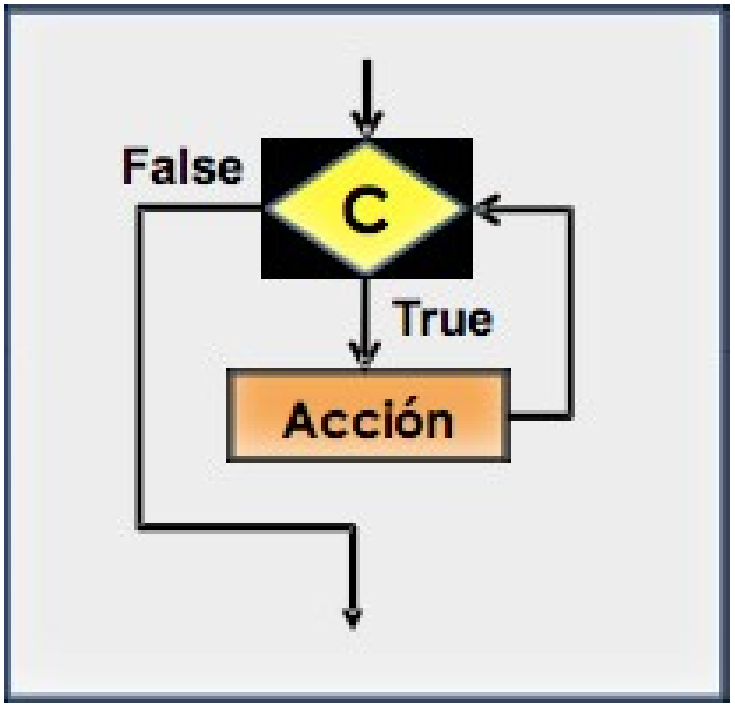
- En algunos problemas, es preciso repetir acciones varias veces, incluso un número desconocido a priori de veces
- Bucle: bloque de sentencias que se ejecutan varias veces, de acuerdo con una condición de parada



~~Java~~
C

Bucle indefinido, control a la entrada

- Se repite un número indefinido de veces (mientras se verifique una condición controlada a la entrada del bucle)

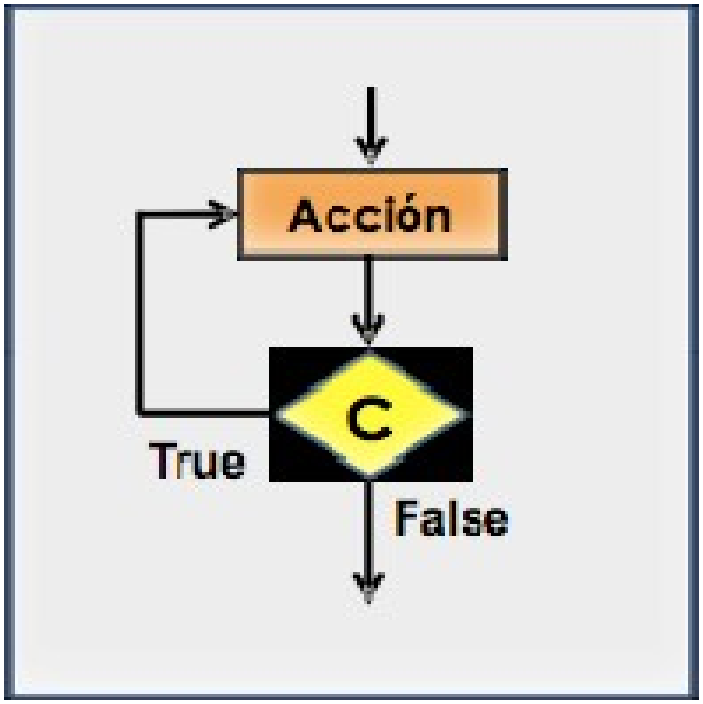


Ejemplo en Java:

```
numero = escaner.nextInt();
while (numero < 0) {
    System.out.println("Leído: " + numero);
    numero = escaner.nextInt();
}
```

Bucle indefinido, control a la salida

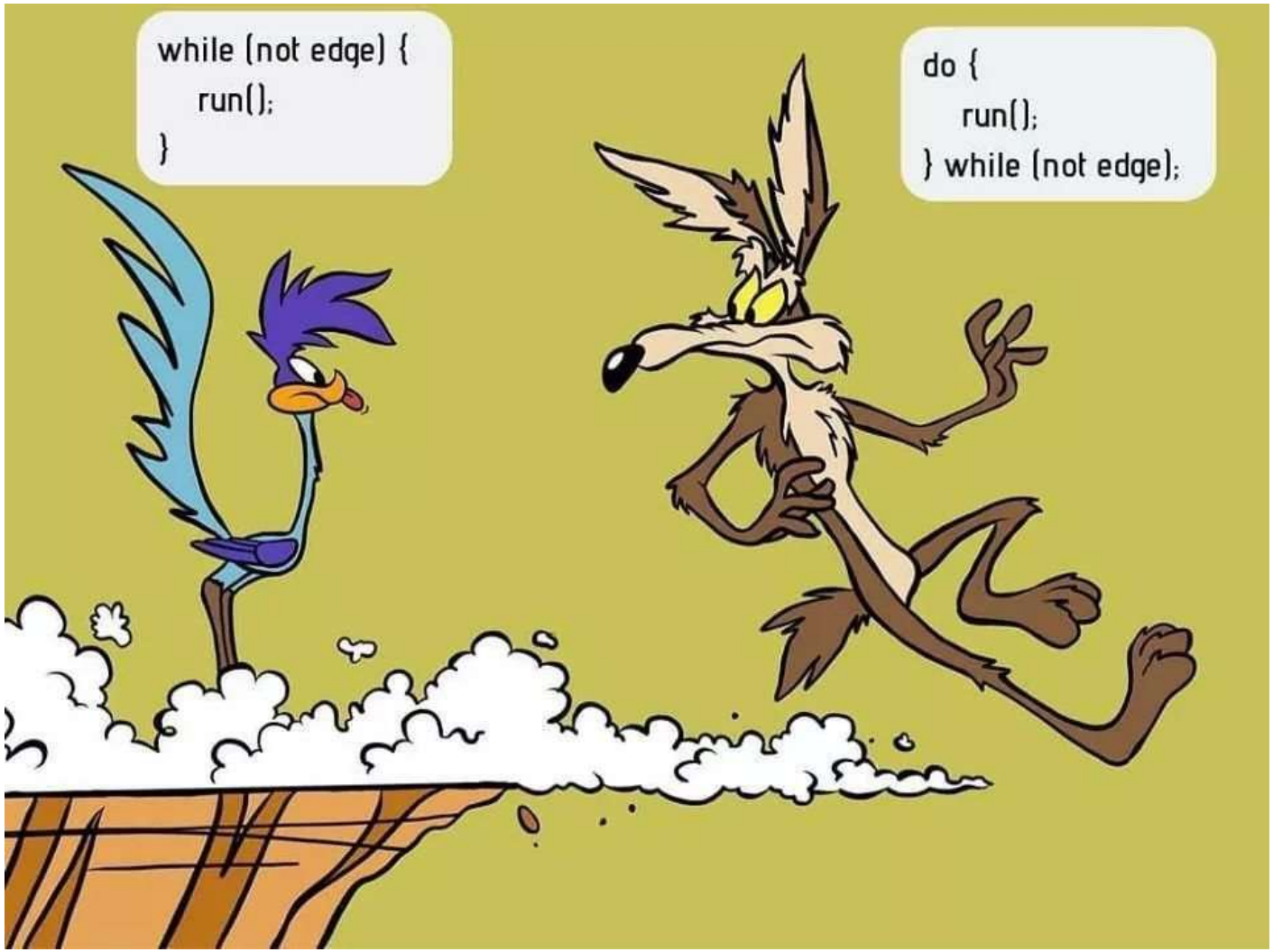
- Se repite un número indefinido de veces (mientras se verifique una condición controlada a la salida del bucle)



Ejemplo en Java:

```
do {
    numero = escaner.nextInt();
    System.out.println("Leído: " + numero);
} while (numero < 0);
```

Bucle indefinido



Bucle definido

- ¿Cómo reescribir un código usando un bucle con control a la salida, de manera que los dos códigos sean equivalentes?

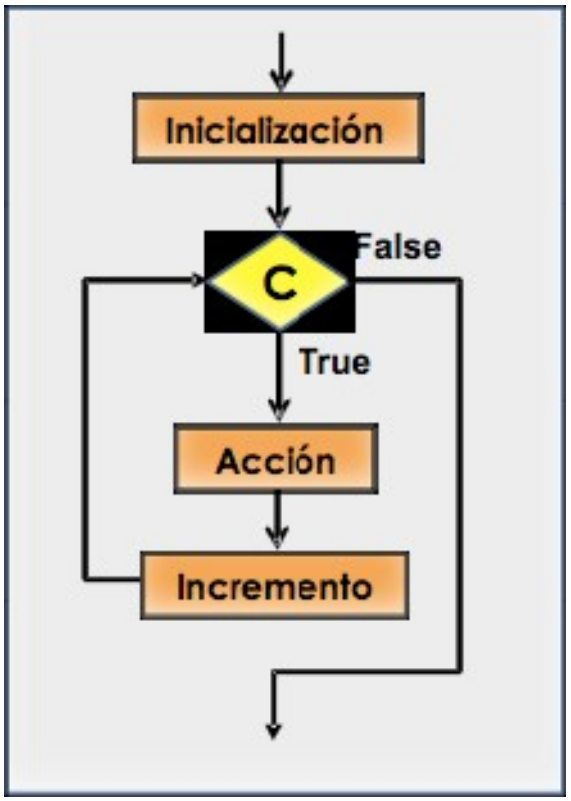
```
do {
   numero = escaner.nextInt();
   System.out.println("Leído: " + numero);
} while (numero < 0);
```

- Solución:

```
numero = escaner.nextInt();
System.out.println("Leído: " + numero);
while (numero < 0) {
   numero = escaner.nextInt();
   System.out.println("Leído: " + numero);
}
```

Bucle definido

- Se repite un número definido de veces: se da a una variable de control un valor inicial, se define cómo cambia sus valores y se define una condición parada en función de su valor

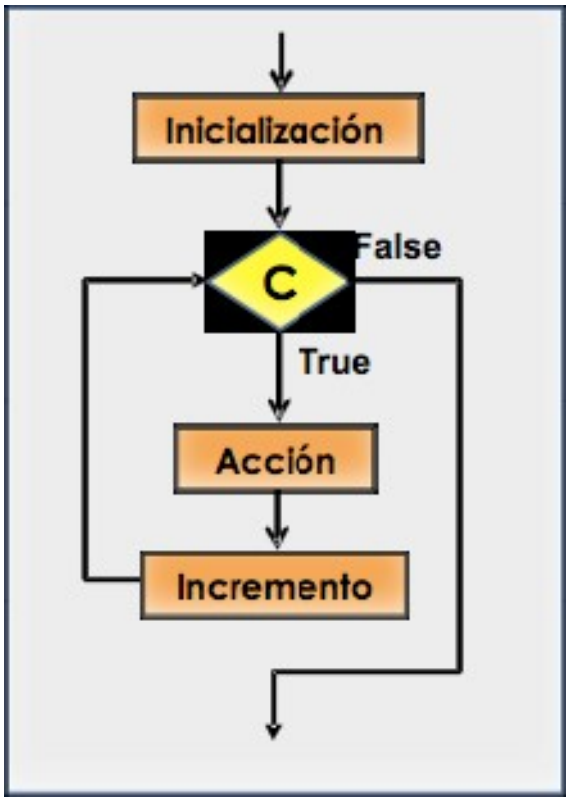


Ejemplo en Java:

```
for (int numero = 1; numero <= 10; numero++)
    System.out.println(numero);
```

Bucle definido

- Se repite un número definido de veces: se da a una variable de control un valor inicial, se define cómo cambia sus valores y se define una condición parada en función de su valor

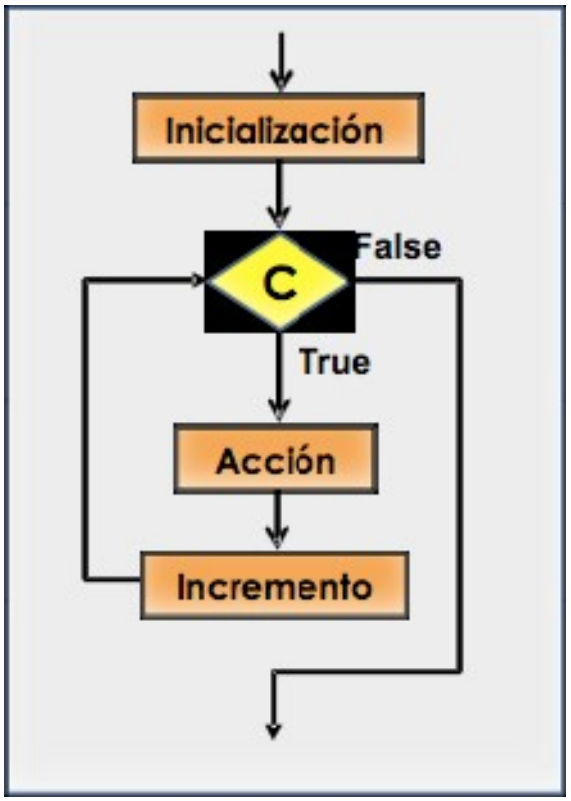


Ejemplo en Java:

```
for (int numero = 1; numero <= 10; numero++)
    System.out.println(numero);
```

Bucle definido

- Se repite un número definido de veces: se da a una variable de control un valor inicial, se define cómo cambia sus valores y se define una condición parada en función de su valor



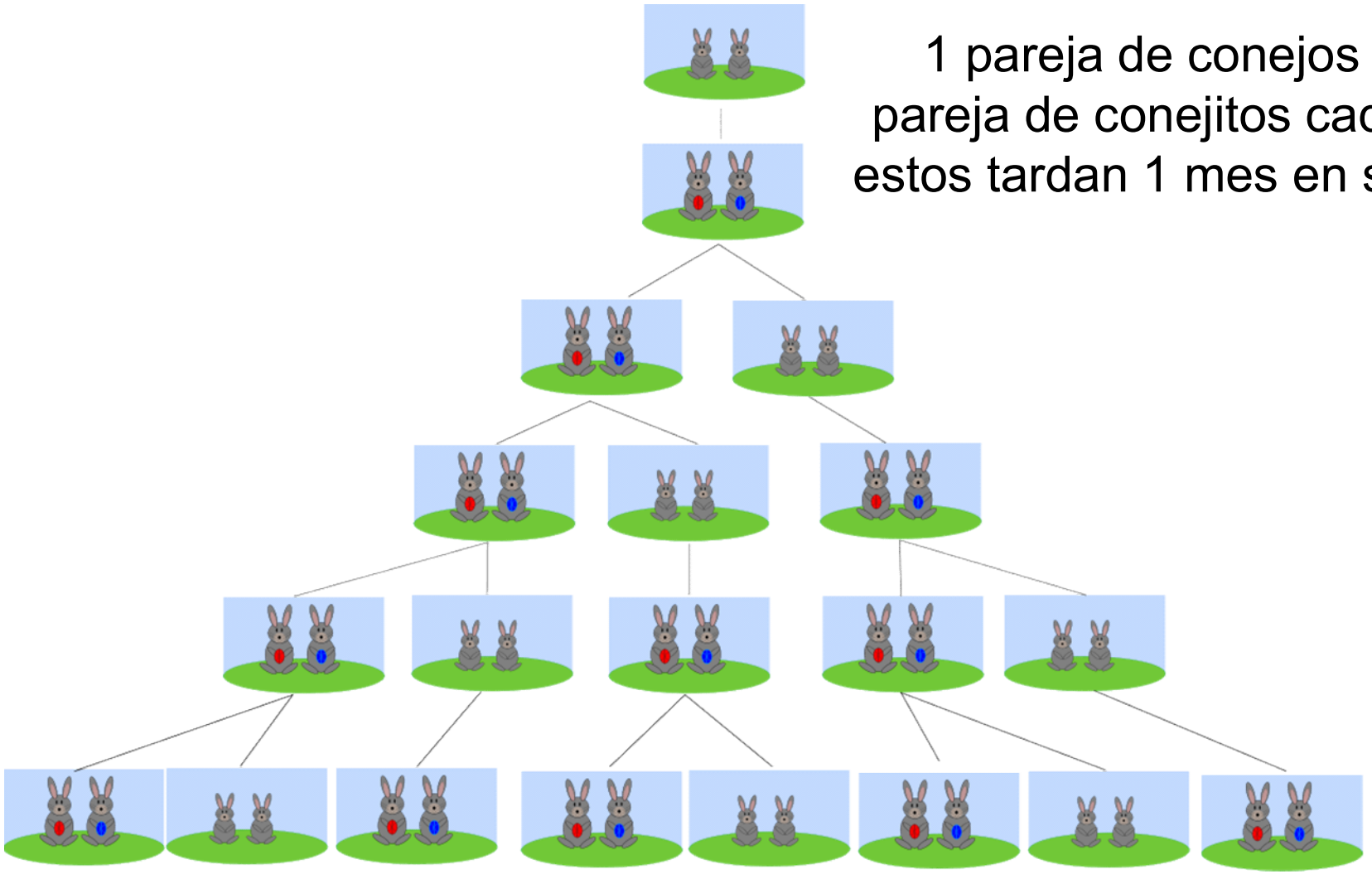
Ejemplo en Java:

```
for (int numero = 1; numero <= 10; numero++)
    System.out.println(numero);
```

Ejemplo: sucesión de Fibonacci

- Sucesión infinita: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 ...
  - Los 2 primeros términos son 0 y 1
  - Los siguientes son la suma de los 2 anteriores
- Autor: Leonardo Pisano “Fibonacci”
- Numerosas aplicaciones: nace como solución al problema del cálculo del número de parejas de conejos

Ejemplo: sucesión de Fibonacci



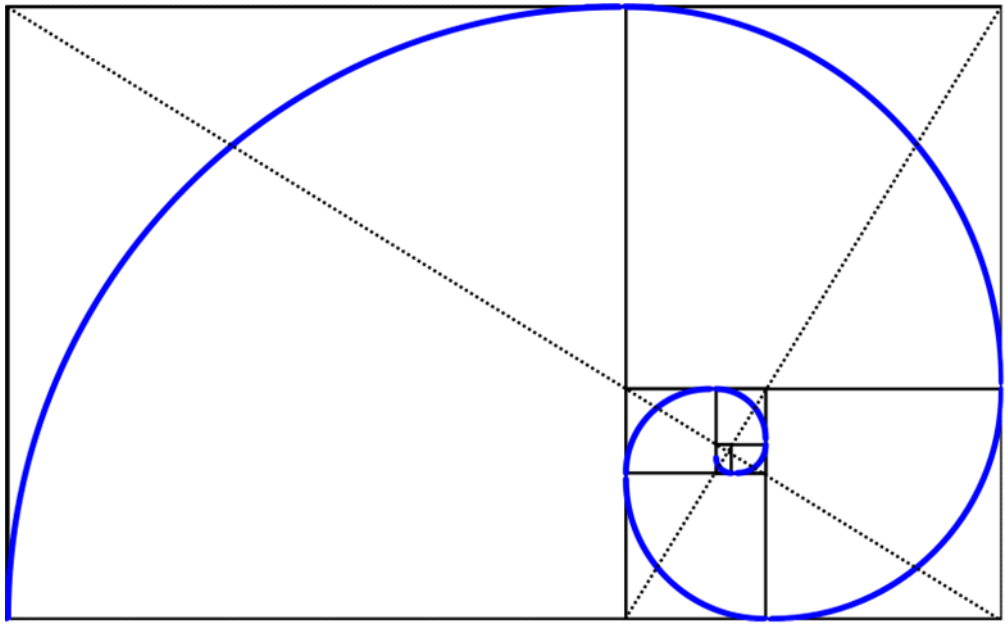
1 pareja de conejos tiene 1 pareja de conejitos cada mes, y estos tardan 1 mes en ser fértiles

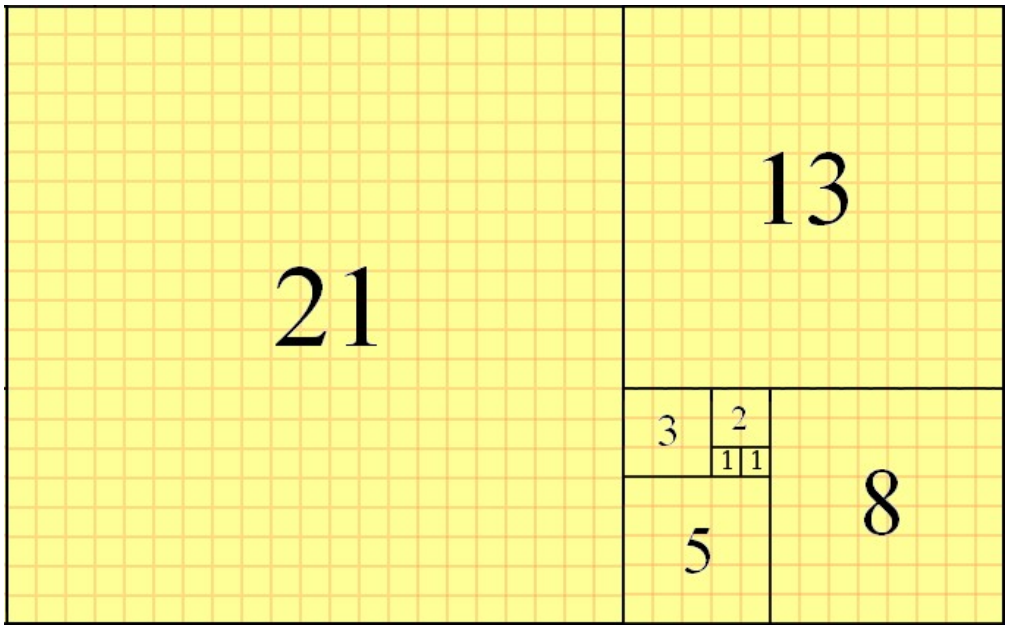
Sucesión de Fibonacci y número áureo

- El cociente entre dos términos consecutivos de la sucesión de Fibonacci F_{n+1} y F_n tiende al número áureo o dorado ϕ

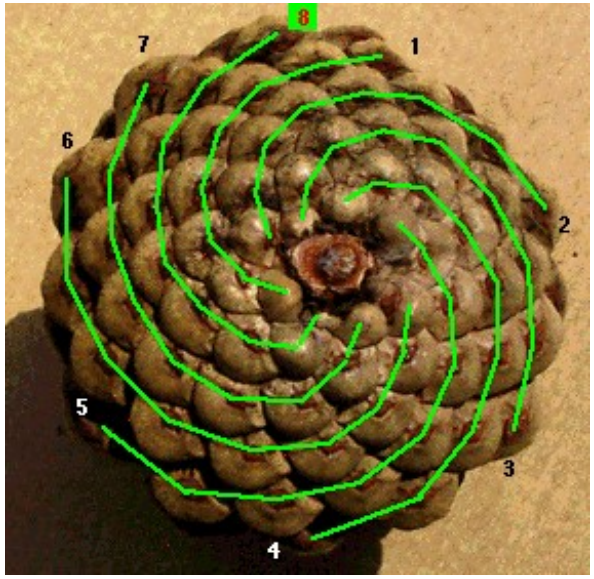
$$\lim_{n\to\infty} \frac{F_{n+1}}{F_n} = \varphi \text{ , donde } \varphi = \frac{1+\sqrt{5}}{2} \approx 1.618034$$

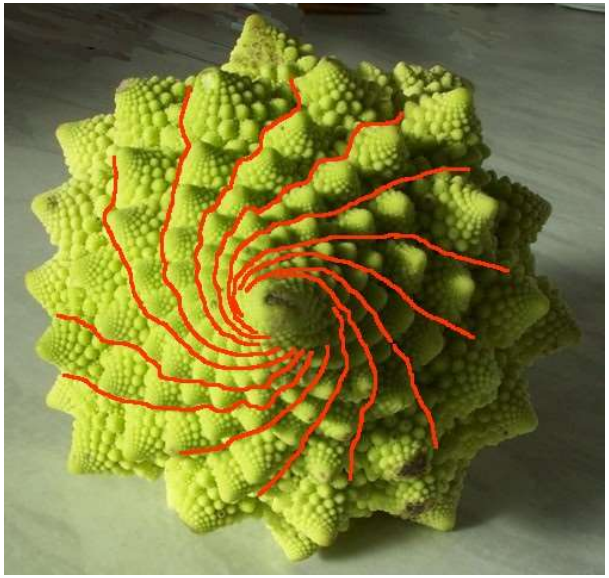
- La espiral áurea tiene como razón de crecimiento ϕ





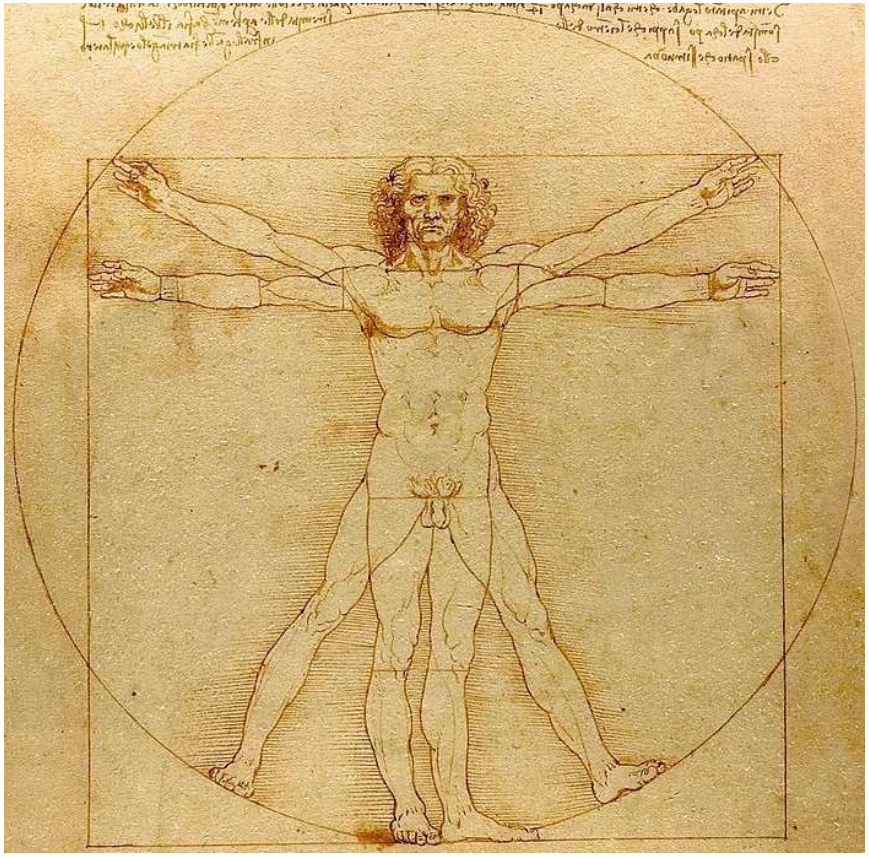
Número áureo en la naturaleza



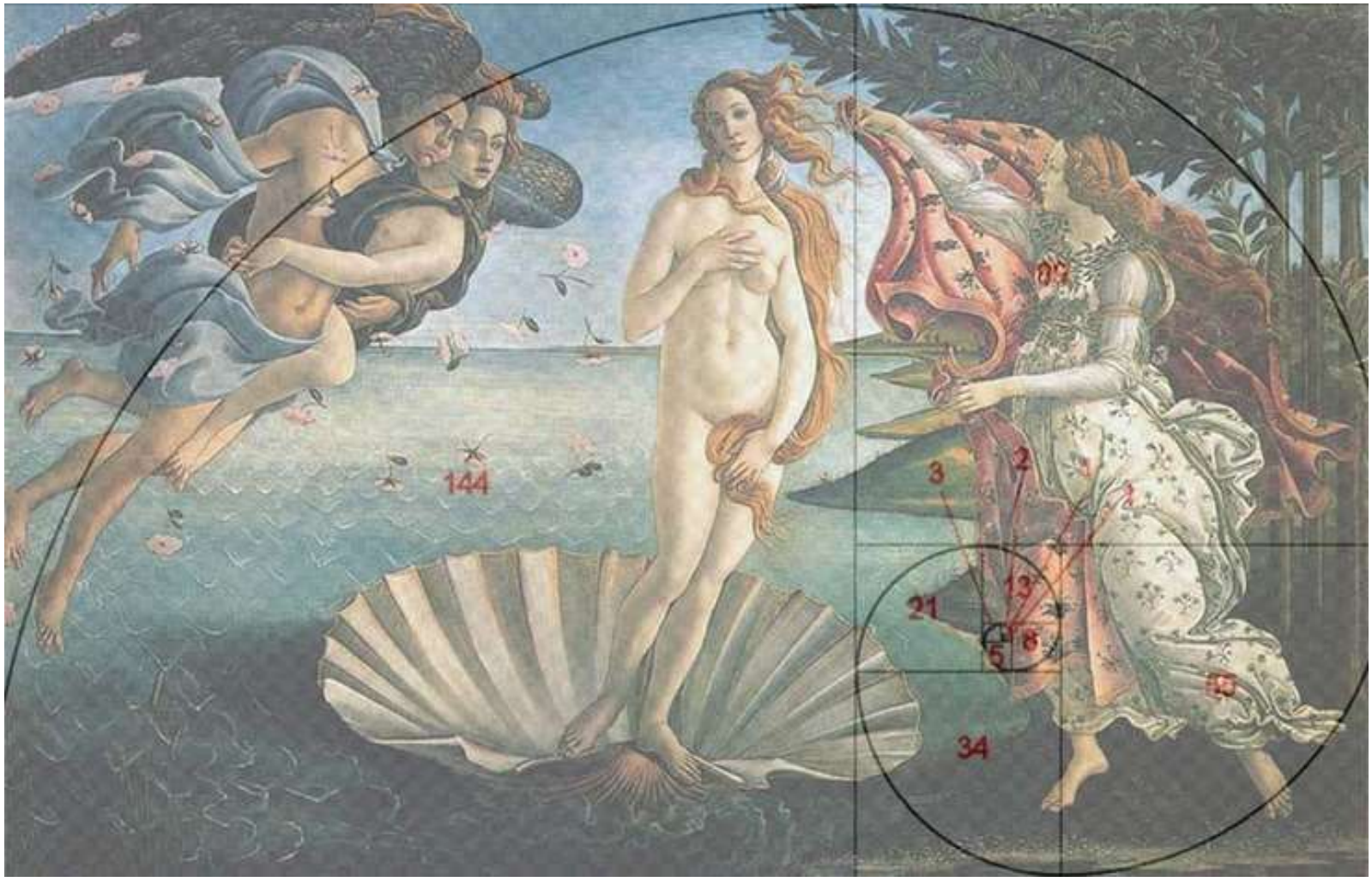


Número áureo en el arte

- Hay muchísimos ejemplos, pero también hay muchos mitos y muchos casos donde aparece por casualidad y no por diseño



Número áureo en el arte



El nacimiento de Venus (Sandro Botticelli)

Número áureo en el arte

La última cena (Salvador Dalí)

Sucesión de Fibonacci con while

```
public class FibonacciWhile {
    public static void main(String[] args) {
        int actual, ant1, ant2;
        ant1 = 1;
        ant2 = 0;
        System.out.println(ant2);
        actual = ant1 + ant2;
        while (actual <= 100) {
            System.out.println(actual);
            actual = ant1 + ant2;
            ant2 = ant1;
            ant1 = actual;
        }
    }
}
```

Términos menores o iguales a 100

Sucesión de Fibonacci con do-while

```
public class FibonacciDoWhile {
   public static void main(String[] args) {
      int actual, ant1, ant2;
      ant1 = 1;
      ant2 = 0;
      System.out.println(ant2);
      actual = ant1 + ant2;
      do {
         System.out.println(actual);
         actual = ant1 + ant2;
         ant2 = ant1;
         ant1 = actual;
      } while (actual <= 100);
   }
}
```

Términos menores o iguales a 100

Sucesión de Fibonacci con for

```
public class FibonacciFor {
    public static void main(String[] args) {
        int actual, ant1, ant2;
        ant1 = 1;
        ant2 = 0;
        System.out.println(ant2);
        System.out.println(ant1);
        for (int i = 3; i <= 10; i++) {
            actual = ant1 + ant2;
            ant2 = ant1;
            ant1 = actual;
            System.out.println(actual);
        }
    }
}
```

10 primeros términos

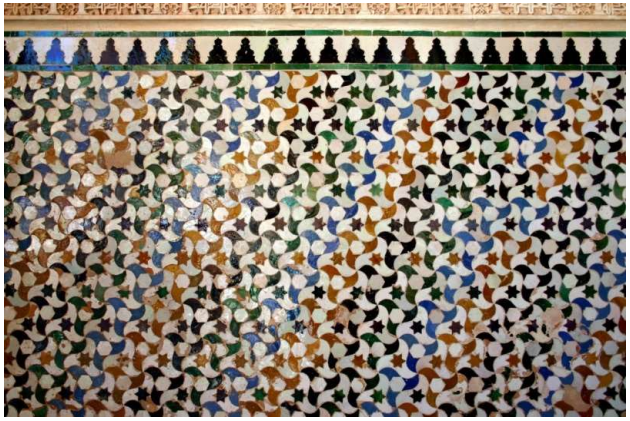
Congruencias

- Calcular el menor natural tal que 98n + 61 es divisible entre 1003

```
public class Congruencias {

   public static void main(String[] args) {
      int candidato;
      candidato = 0;
      while ( (98 * candidato + 61) % 1003) != 0)
         candidato = candidato + 1;
      System.out.println("El número es " + candidato);
   }
}
```

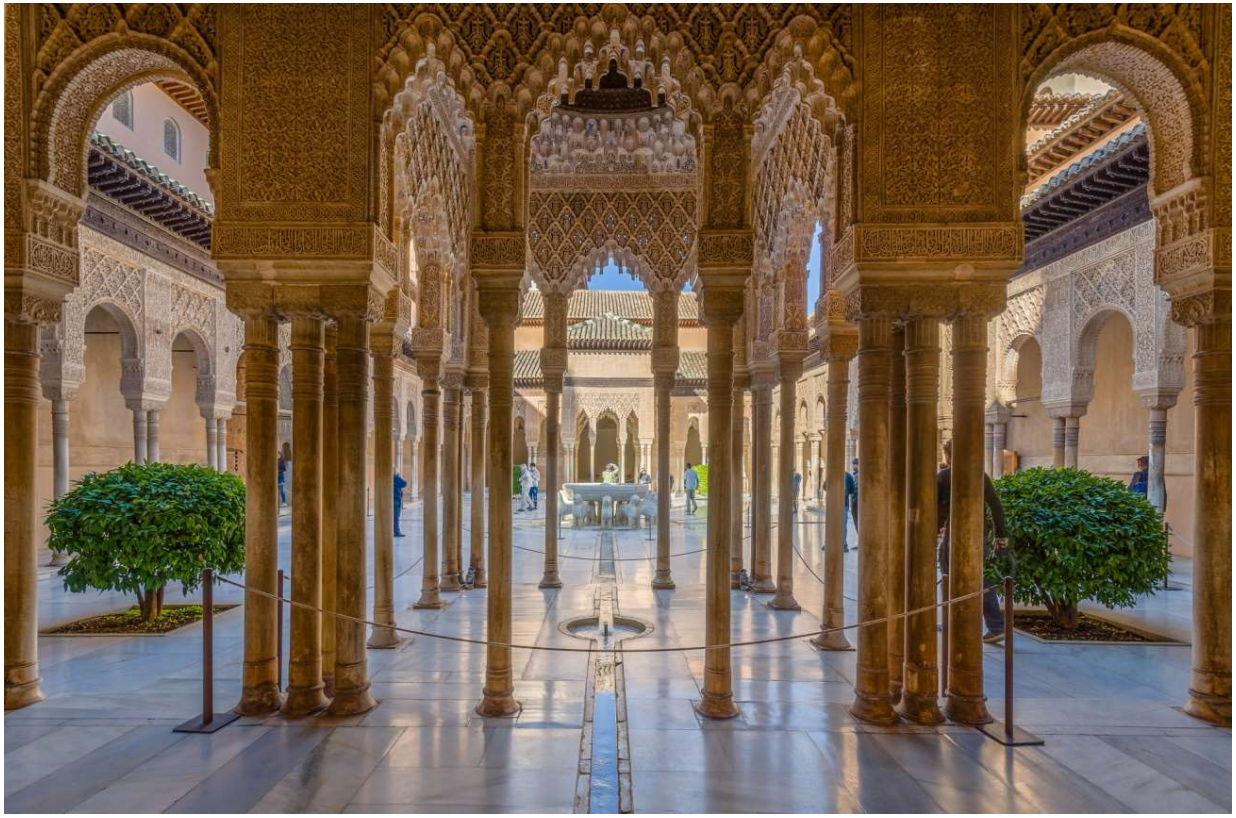
Ejemplo: los mosaicos árabes

- Mosaico (en matemáticas):
  - Composición formada por la repetición de losetas
- Grupos cristalográficos planos:
  - Estructuras básicas para las infinitas decoraciones posibles del plano formado por mosaicos periódicos
- Teorema de Fedorov (1891):
  - 17 grupos cristalográficos planos



Teorema de la Alhambra

- La Alhambra (en Granada) es el único monumento de la antigüedad (anterior al siglo XIX) donde se han hecho dibujos o diseños periódicos con los 17 grupos cristalográficos planos



17 grupos cristalográficos planos

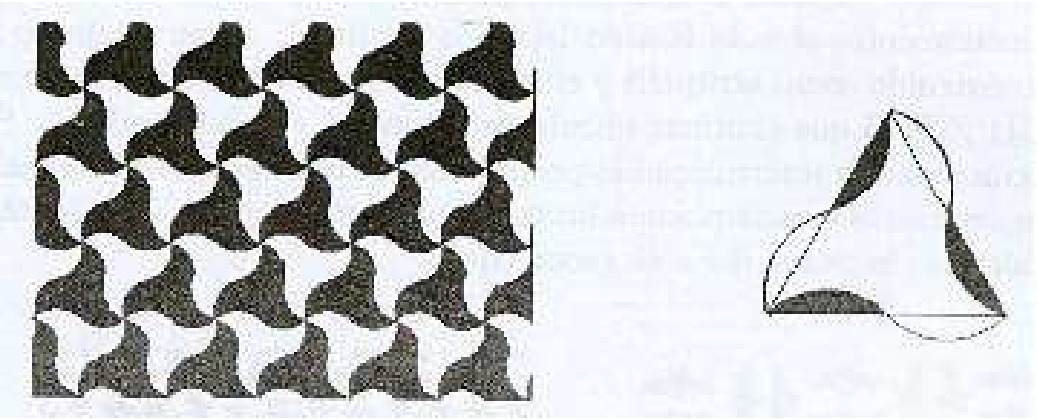
- Formados por combinaciones de 4 transformaciones:
  - Traslación
  - Rotación
  - Reflexión
  - Reflexión con deslizamiento

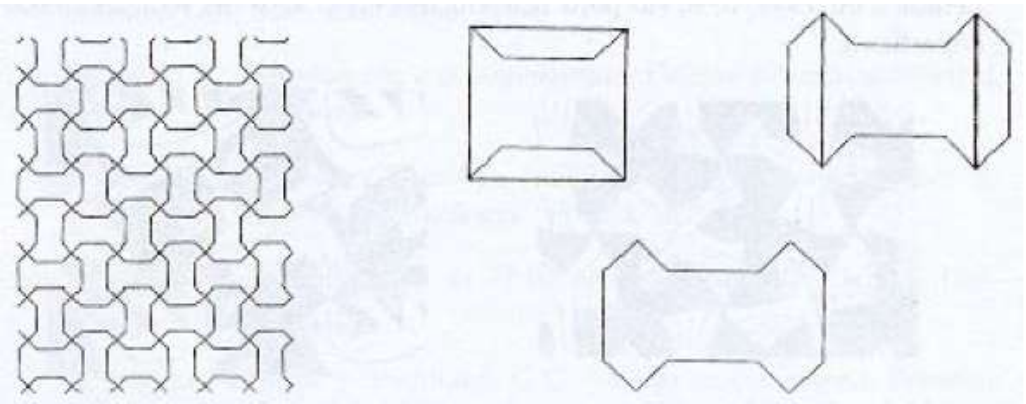
- La notación usual utiliza las letras p, m, g, c para designar el grupo, en función del tipo de isometrías que lo componen
  - *px* indica giros de amplitud 2π/x, donde *x* es un entero
  - *m* (*mirror*) indica simetrías axiales o reflexiones
  - *g* (*glide*) indica que existen simetrías con deslizamiento
  - c indica centrado, que implica un plano de deslizamiento

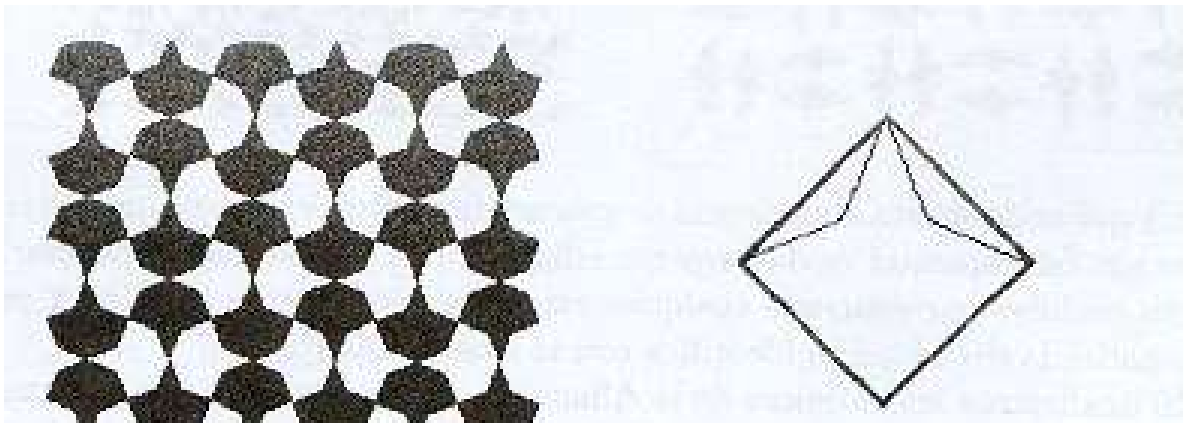
17 grupos cristalográficos planos

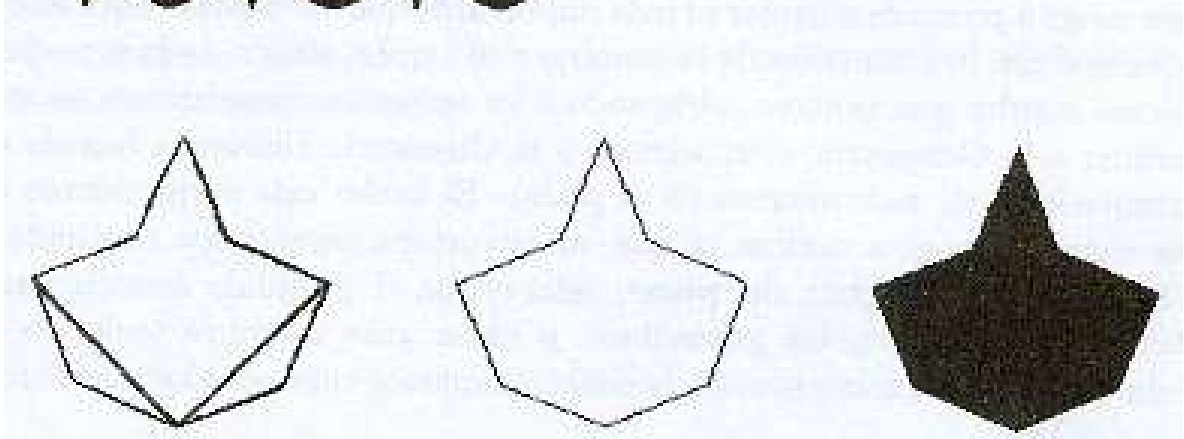
1. p1: 2 traslaciones
2. p3: 2 giros de 120º
3. p31m: 1 simetría axial y 1 giro de 120º
4. p6: 1 simetría central y 1 giro de 120º
5. p6m: 3 simetrías axiales en los lados de un triángulo de ángulos 30-60-90
6. p4: 1 simetría central (o giro de 180º) y 1 giro de 90º
7. p4g: 1 simetría axial y 1 giro de 90º
8. p4m: 3 simetrías axiales en los lados de un triángulo de ángulos 45-45-90
9. cm: 1 simetría axial y 1 simetría con deslizamiento perpendicular
10. cmm: 2 simetrías axiales perpendiculares y 1 simetría central
11. pm: 2 simetrías axiales y 1 traslación
12. pmm: 4 simetrías axiales en los lados de un rectángulo
13. pmg: 1 simetría axial y 2 simetrías centrales
14. p2: giros de 180º (ó 3 simetrías centrales)
15. pg: 2 simetrías con deslizamiento paralelas
16. pgg: 2 simetrías con deslizamiento perpendiculares
17. p3m1: 3 simetrías axiales en lados de triángulo equilátero (ángulos 60-60-60)

Transformaciones de polígonos regulares



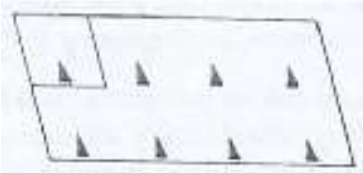




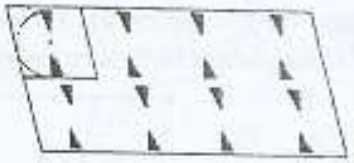


Grupos p

p1

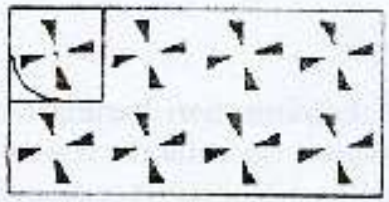


p2

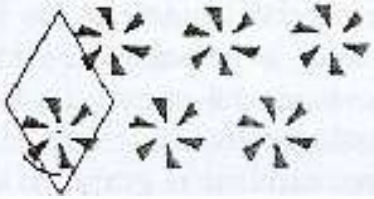


p3

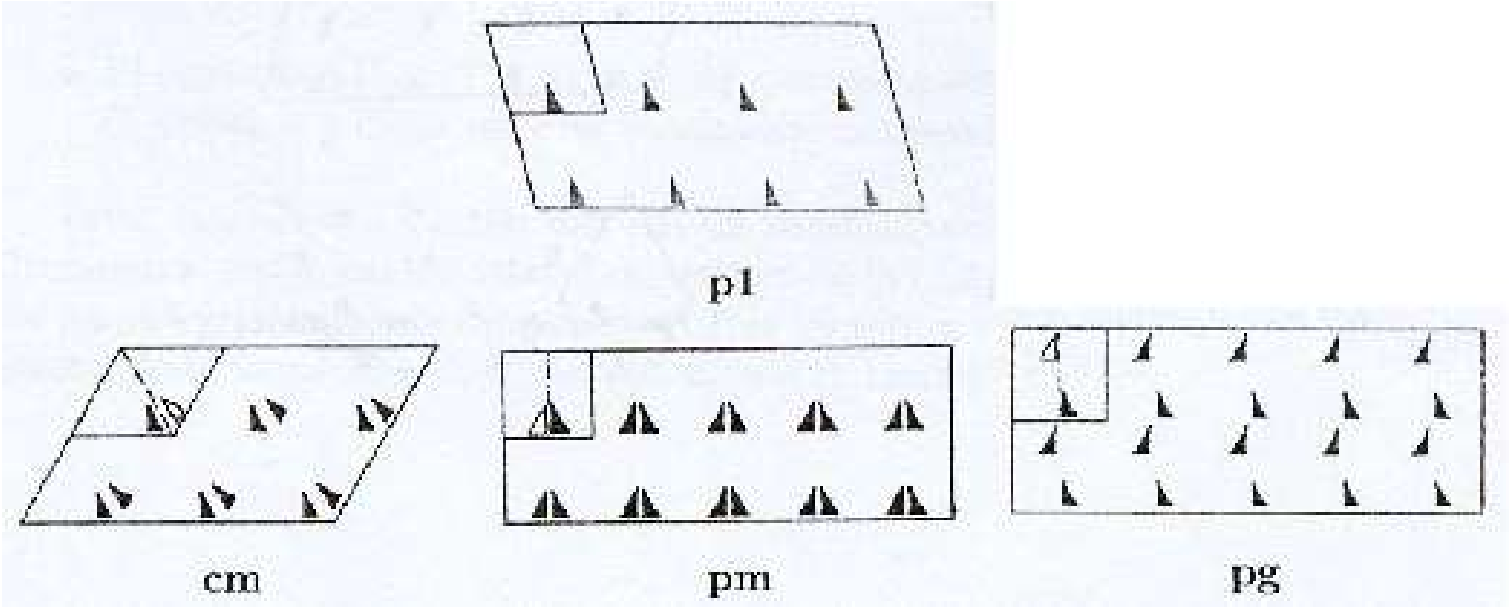
p4



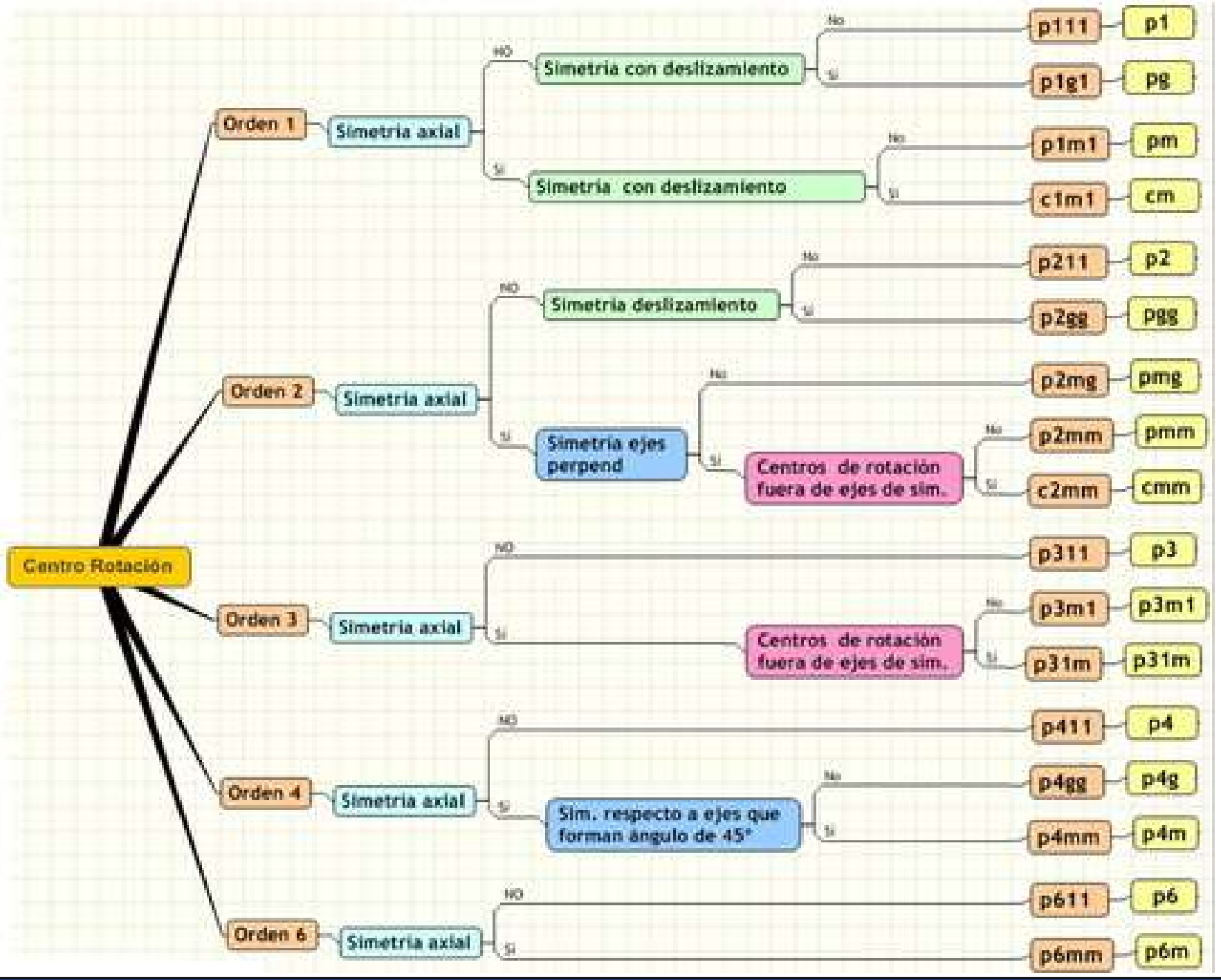
p6



Grupos p1

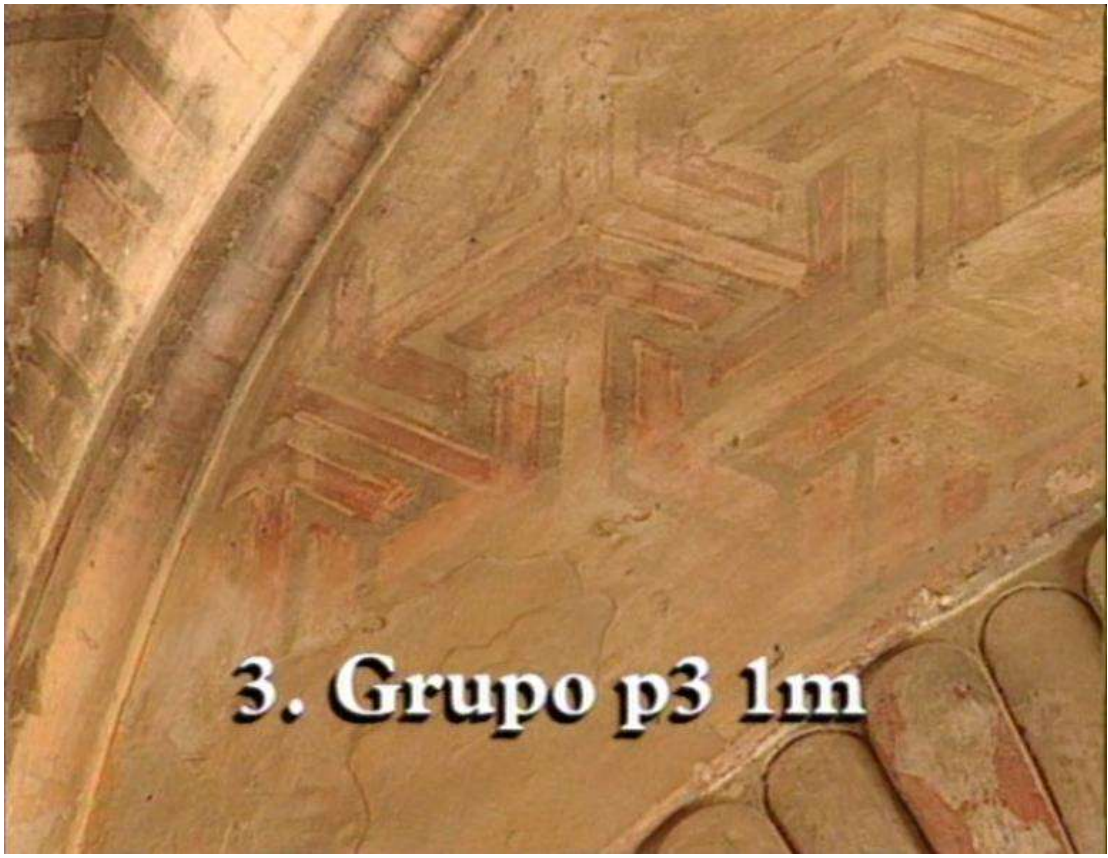


- p1: reflexión NO, simetría con desplazamiento NO
- cm: reflexión SÍ, simetría con desplazamiento SÍ
- pm: reflexión SÍ, simetría con desplazamiento NO
- pg: reflexión NO, simetría con desplazamiento SÍ

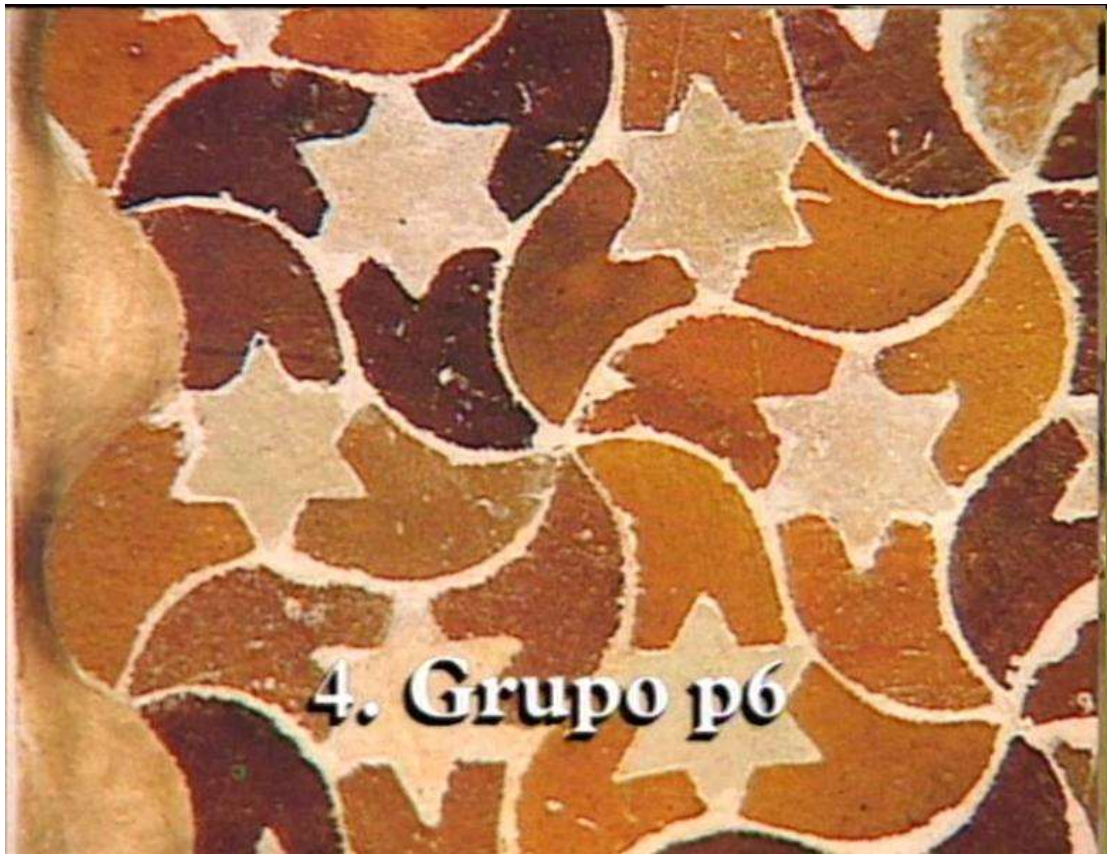

Centro Rotación
Orden 1
Simetria axial
NO
Simetria con deslizamiento
No
p111
p1
Si
p1g1
pg
Si
Simetria con deslizamiento
No
p1m1
pm
Si
c1m1
cm
Orden 2
Simetria axial
NO
Simetria deslizamiento
No
p211
p2
Si
p2gg
pgg
Si
Simetria ejes perpend
No
p2mg
pmg
Si
Centros de rotación fuera de ejes de sim.
No
p2mm
pmm
Si
c2mm
cmm
Orden 3
Simetria axial
NO
p311
p3
Si
Centros de rotación fuera de ejes de sim.
No
p3m1
p3m1
Si
p31m
p31m
Orden 4
Simetria axial
NO
p411
p4
Si
Sim. respecto a ejes que forman ángulo de 45°
No
p4gg
p4g
Si
p4mm
p4m
Orden 6
Simetria axial
NO
p611
p6
Si
p6mm
p6m

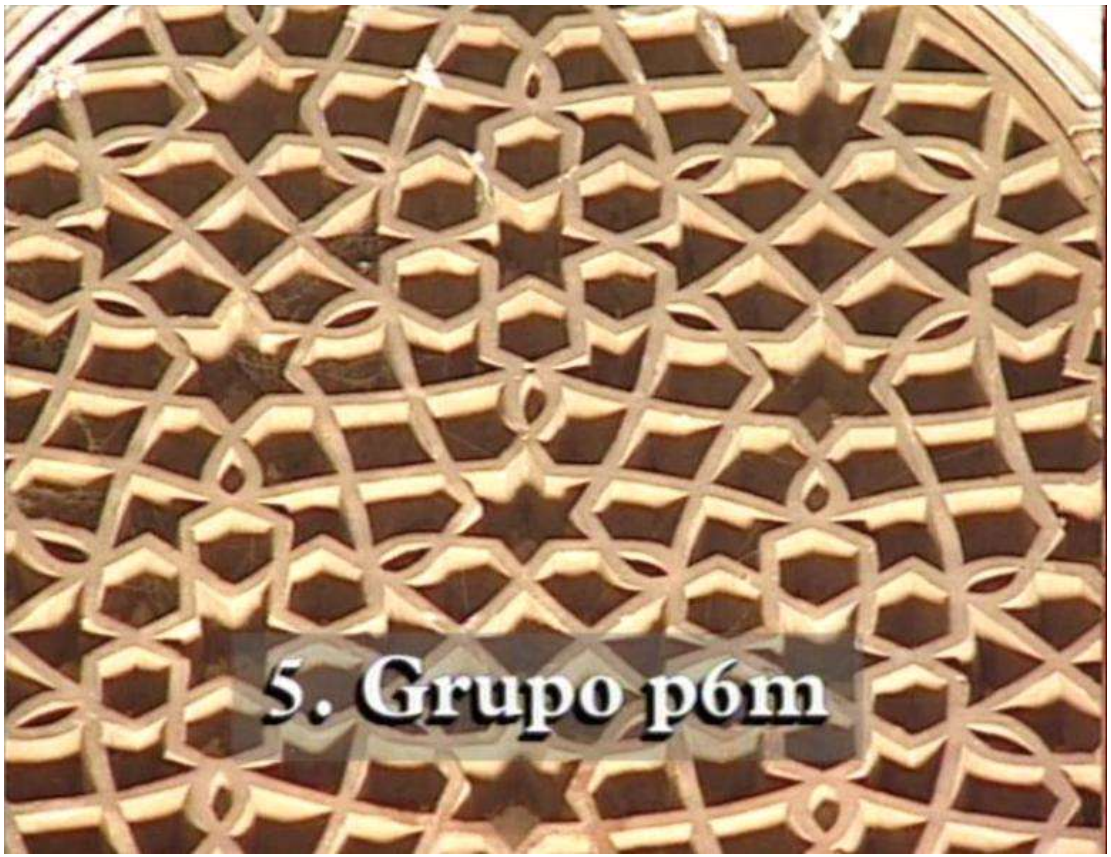
Mosaicos de la Alhambra

Mosaicos de la Alhambra

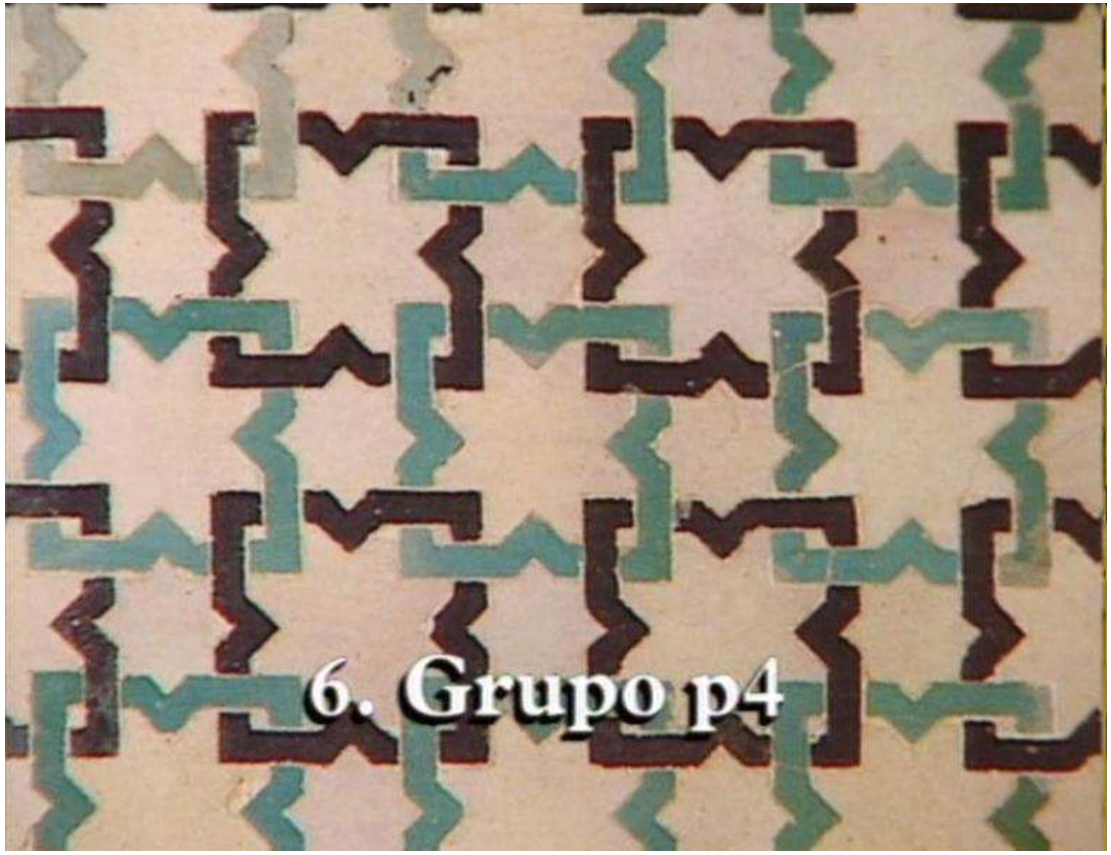


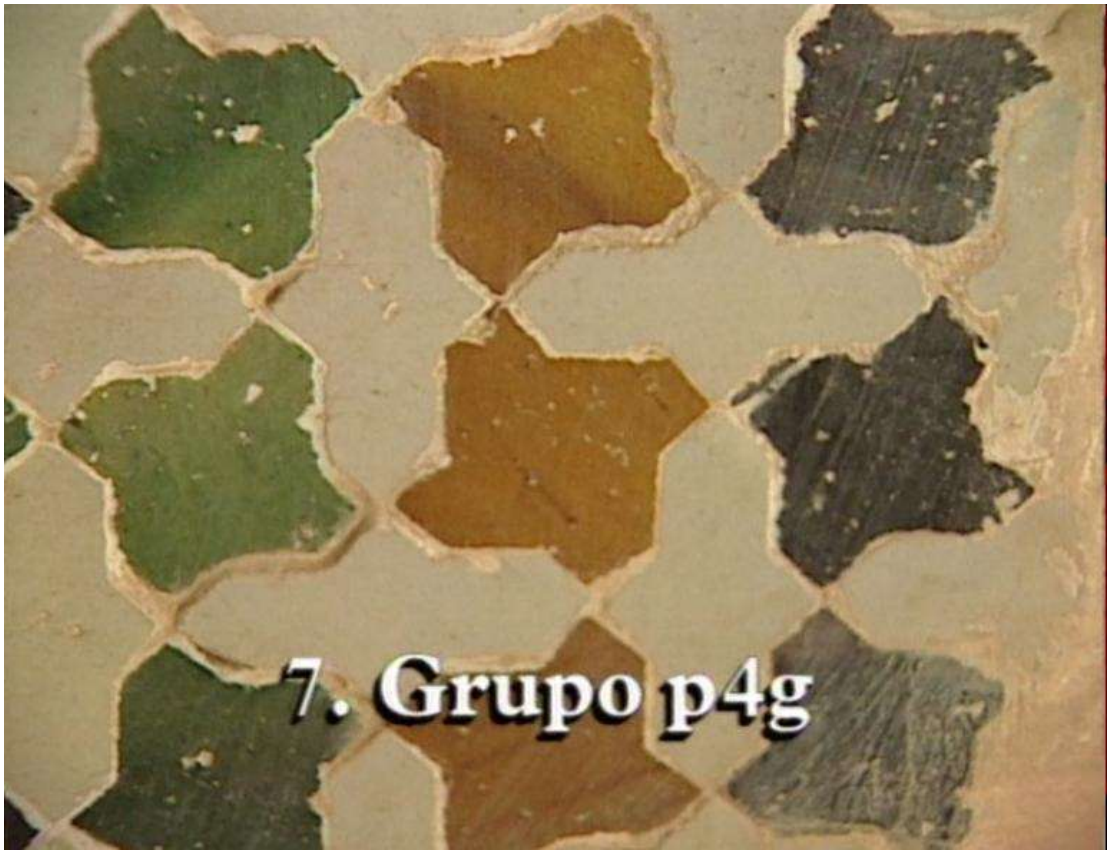
Mosaicos de la Alhambra



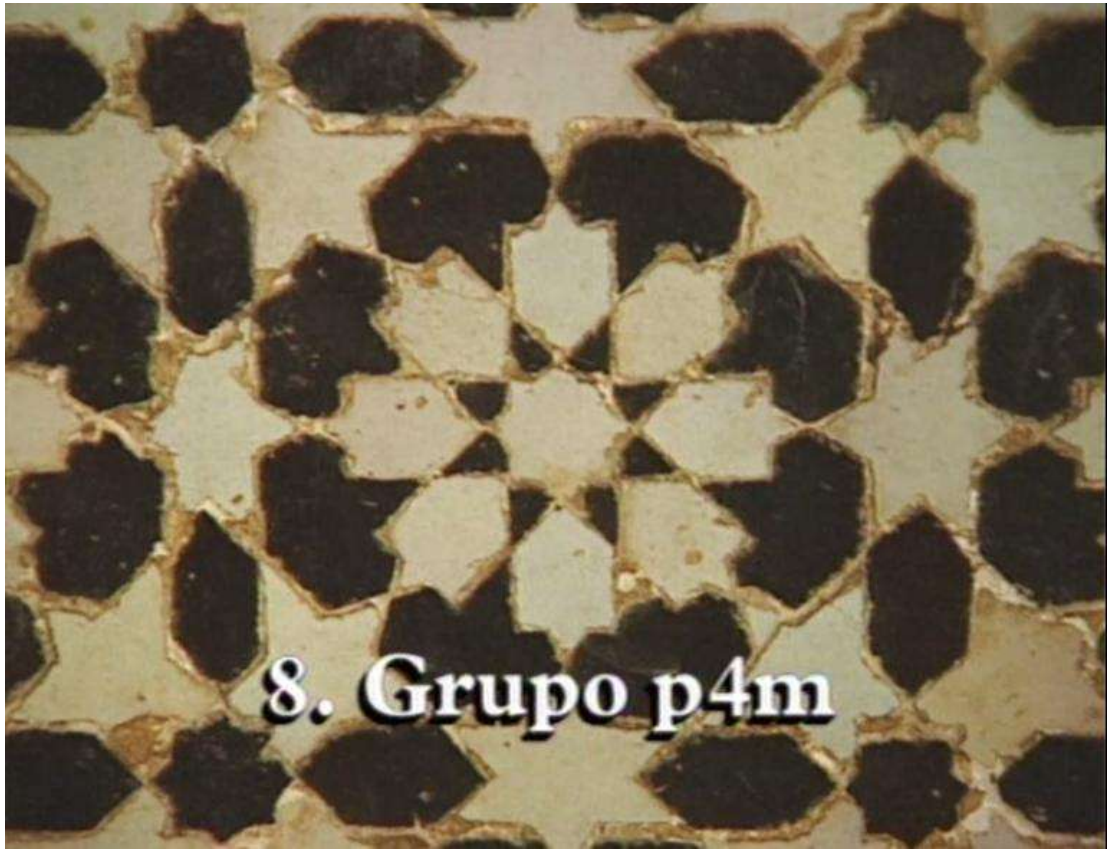


Mosaicos de la Alhambra

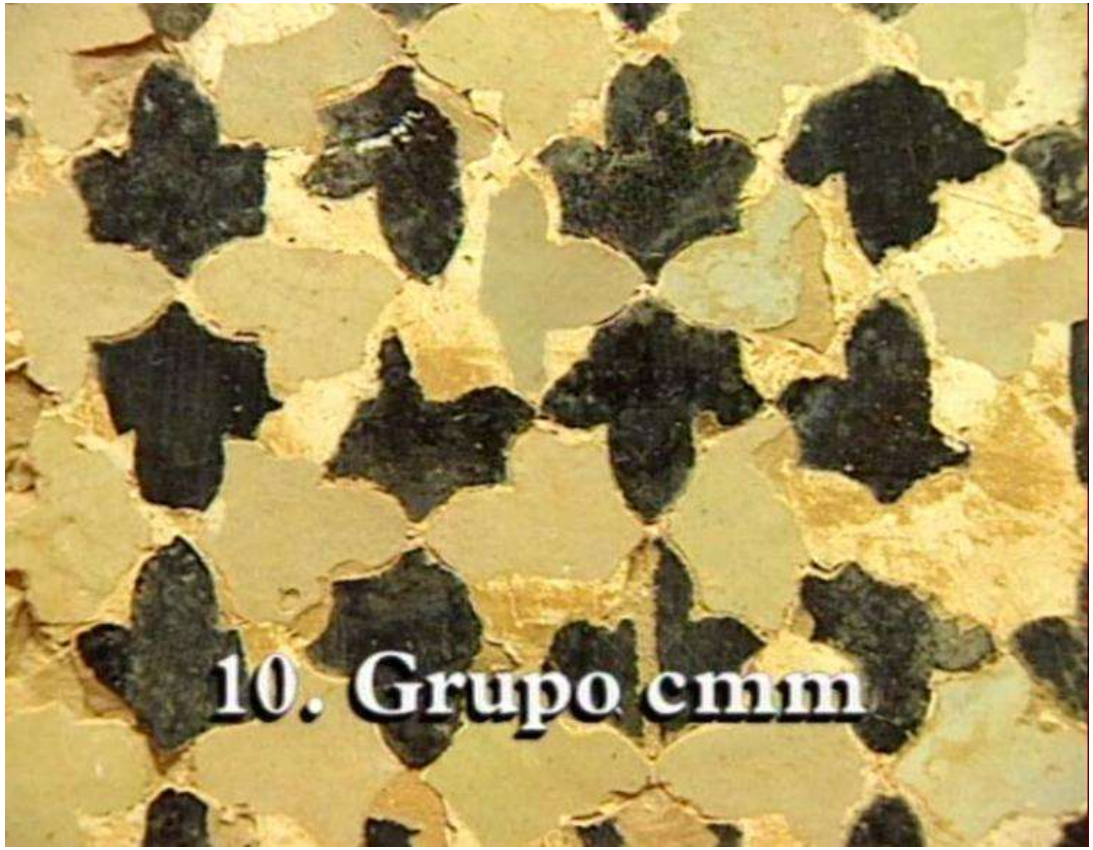




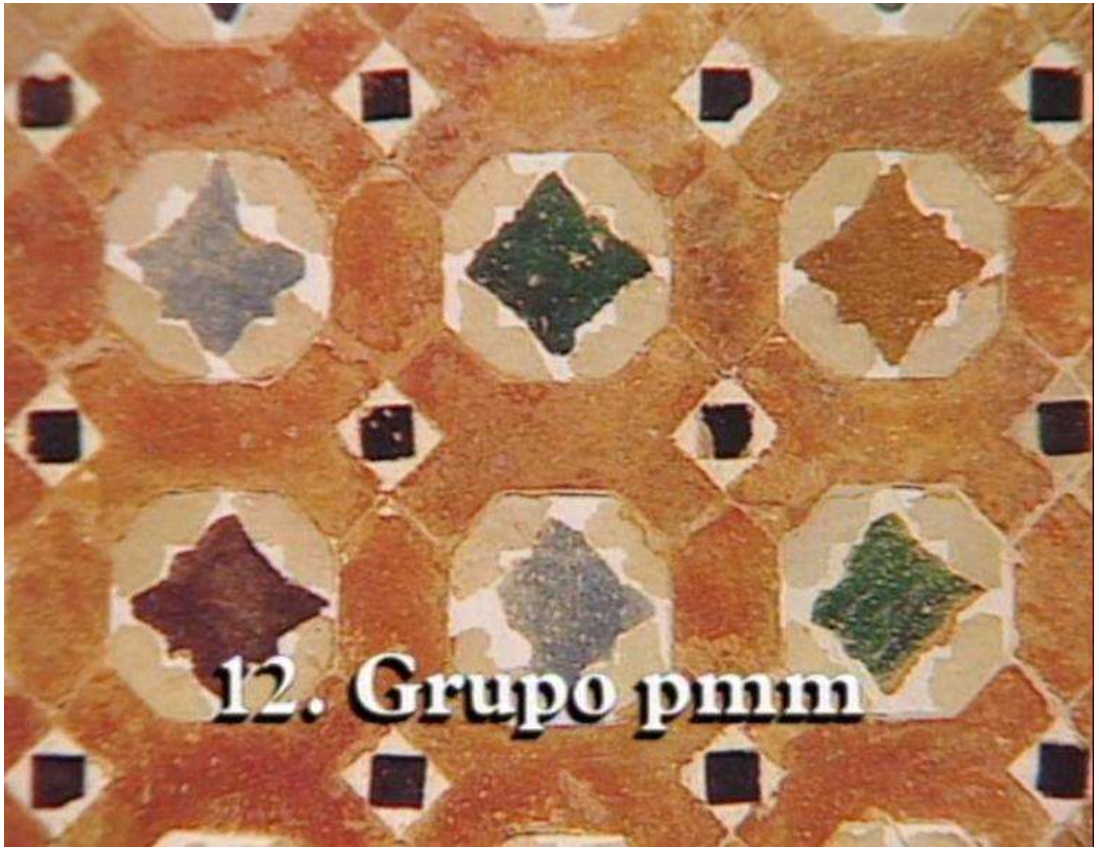
Mosaicos de la Alhambra

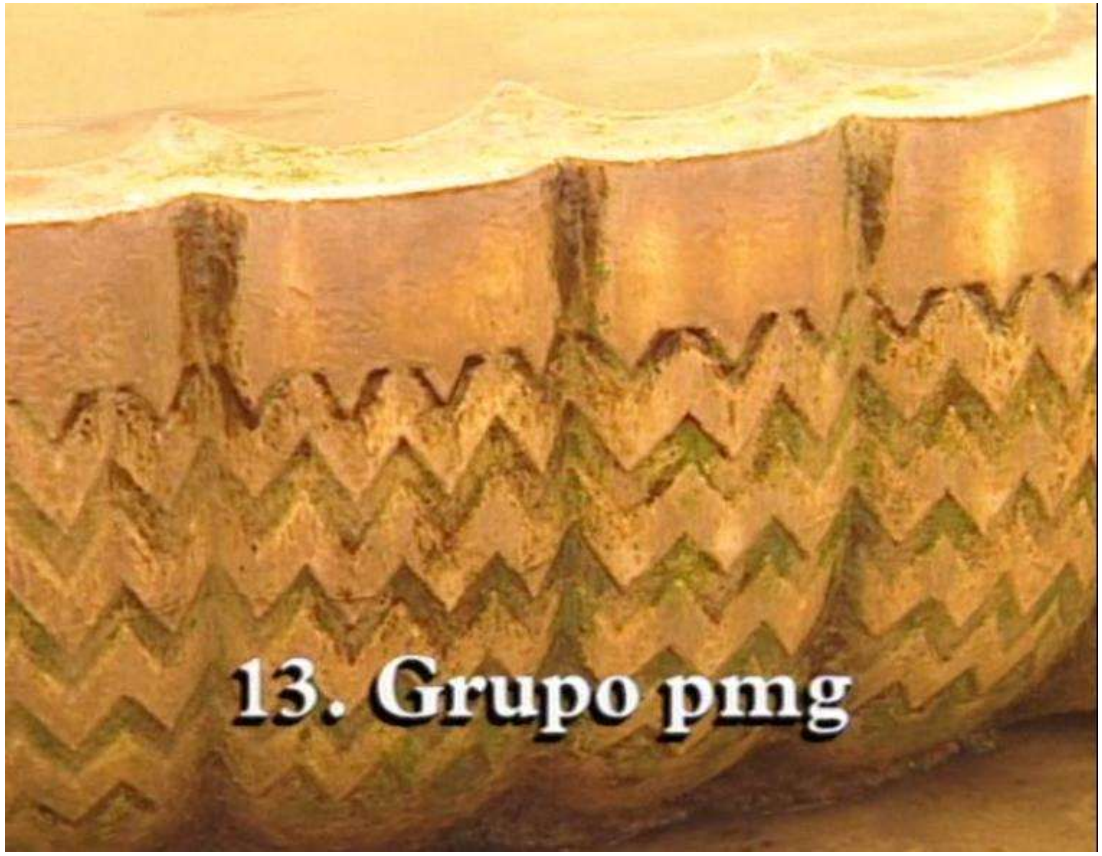


Mosaicos de la Alhambra

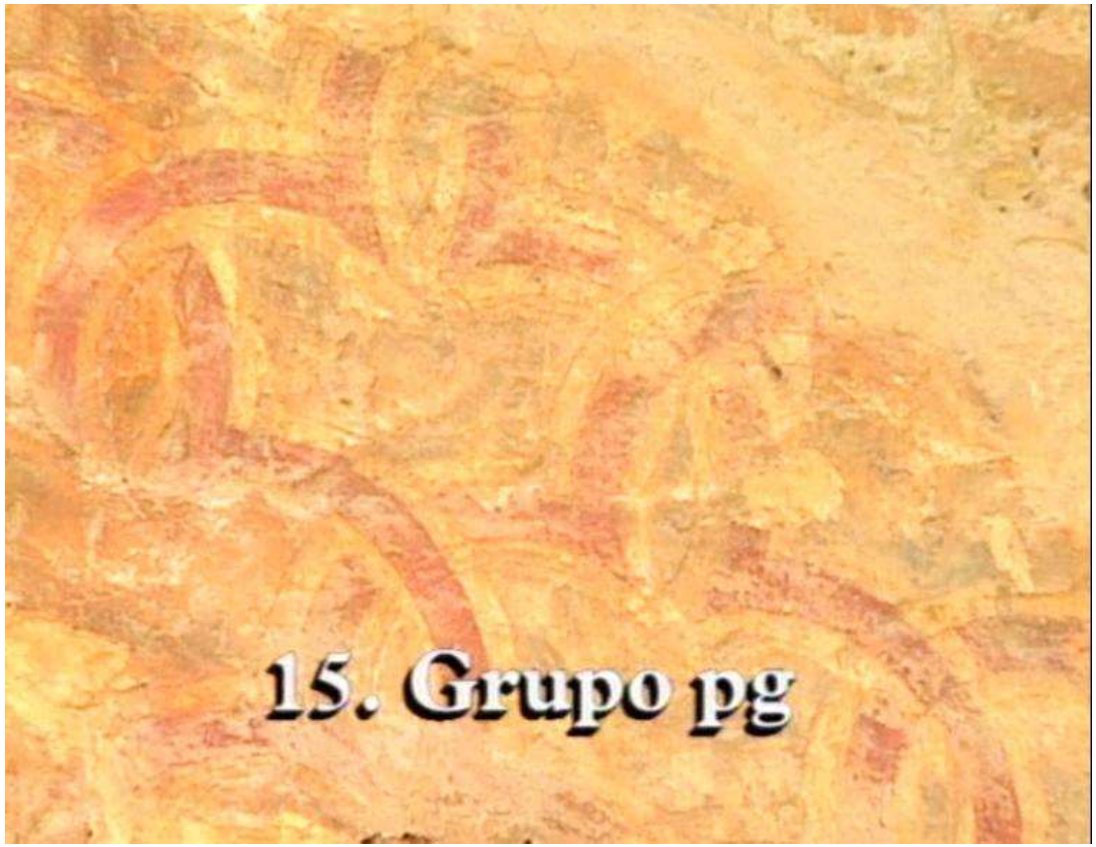


Mosaicos de la Alhambra





Mosaicos de la Alhambra



Mosaicos de la Alhambra

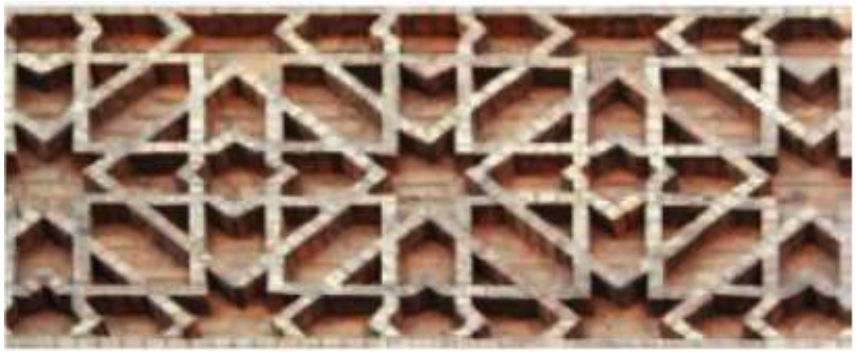
Mosaicos en la Seo de Zaragoza

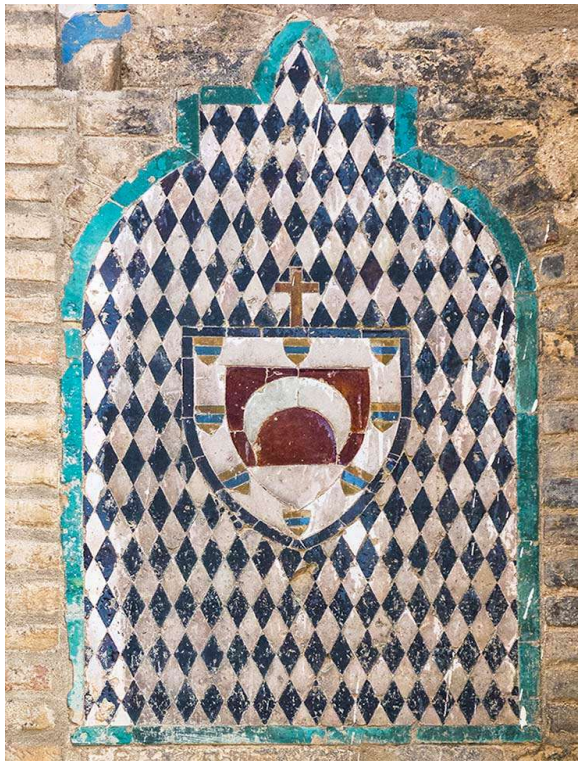
pm

p4m

p3m1

p4





pmm

p6

Vectores y matrices

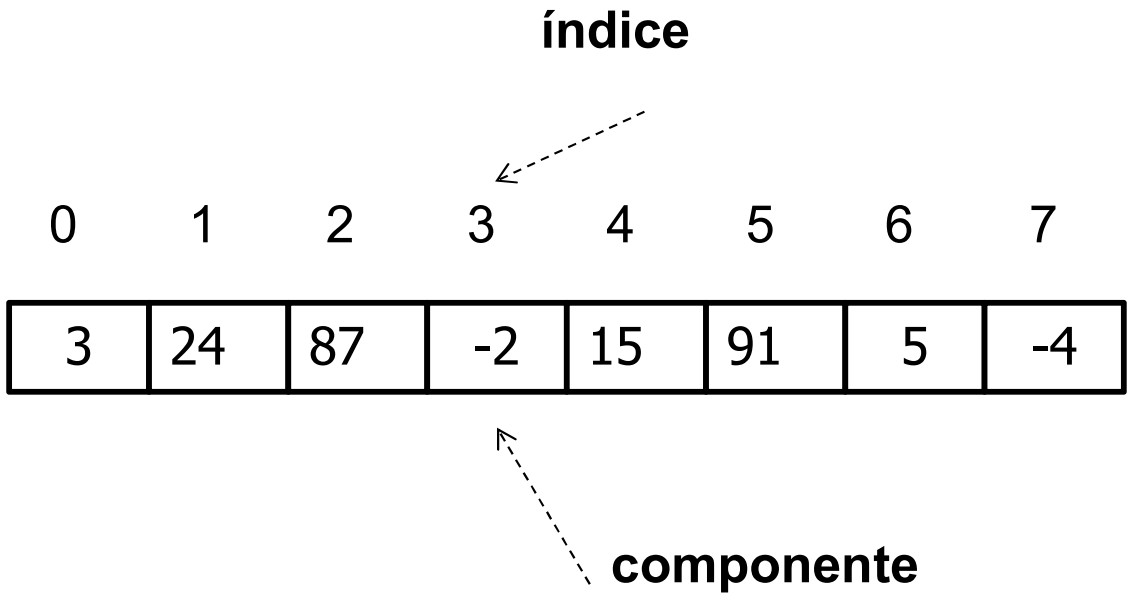
- Problema: dar aprobado general a los alumnos del curso

```
nota1 = 5;
nota2 = 5;
nota3 = 5;
nota4 = 5;
nota5 = 5;
nota6 = 5;
...
```

- ¡n variables y n instrucciones prácticamente iguales!
- Solución: 1 variable compuesta por las notas individuales (posiblemente distitnas) que permita recorrerlas con un bucle

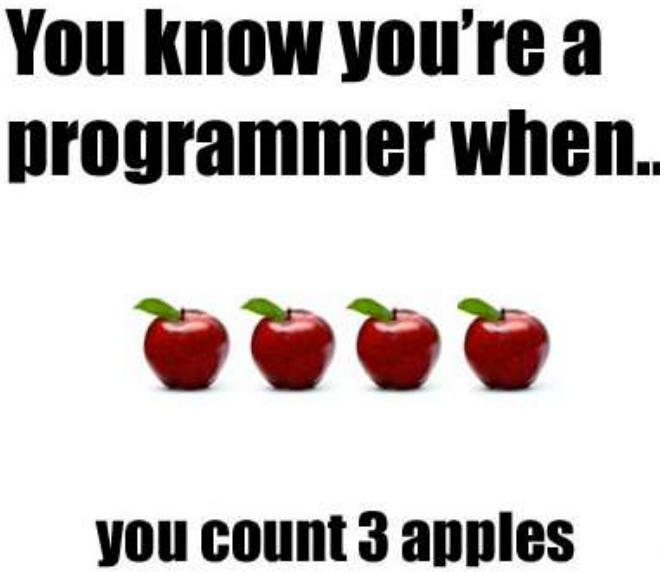
Vectores y matrices

- Un vector (o arreglo, del inglés *array*) es una agrupación de variables del mismo tipo cuyo acceso se realiza por índice
- Pueden tener tantas dimensiones como se desee
- Un vector de 2 o más dimensiones se suele llamar matriz



Vectores y matrices

- Las posiciones empiezan a numerarse en 0 y no 1
  - Un vector de tamaño 3 tiene las posiciones 0, 1 y 2
- Para acceder a una posición concreta se usa el operador `[]`
  - `v[1]`: segunda posición del vector `v`
  - `m[1, 2]`: celda en la segunda fila y primera columna de `m`



Vectores y matrices

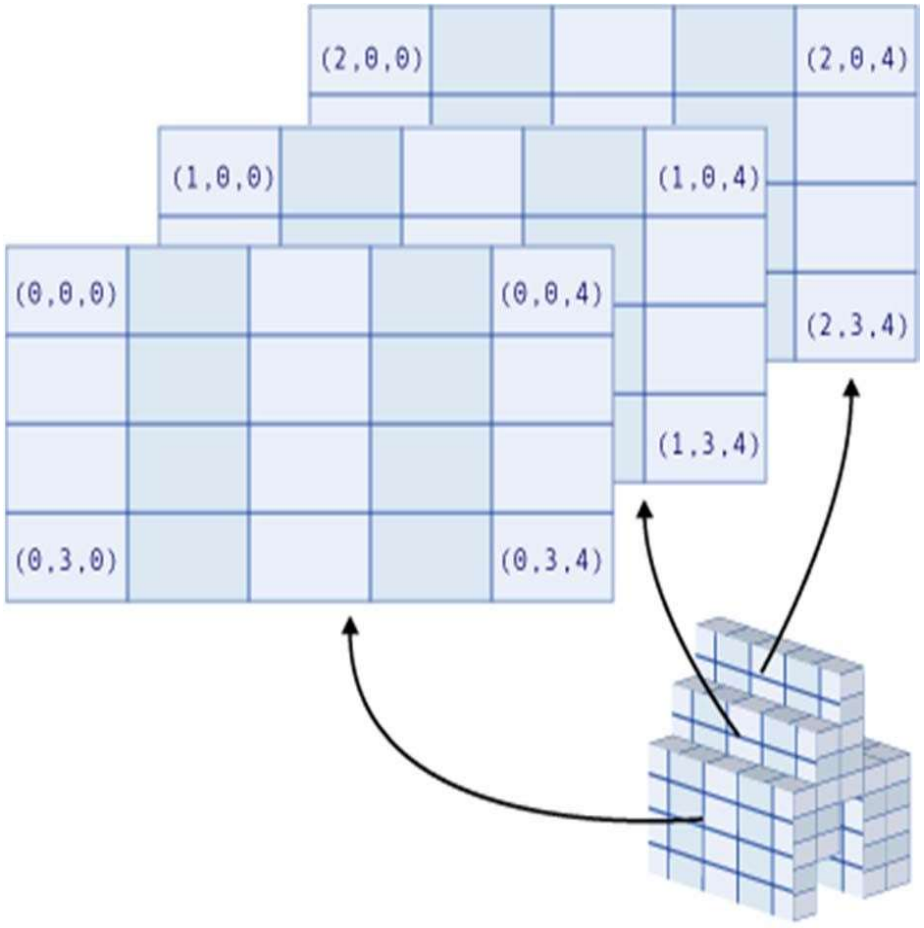
- El tamaño (número de variables) puede variar para cada vector/matriz
- Incluso en cada dimensión puede haber un tamaño diferente
- Como el tamaño puede variar, debe haber alguna manera de conocer el tamaño de cada dimensión: `length`
- Número de filas de una matriz:

```
m.length
```

- Número de columnas de una matriz:

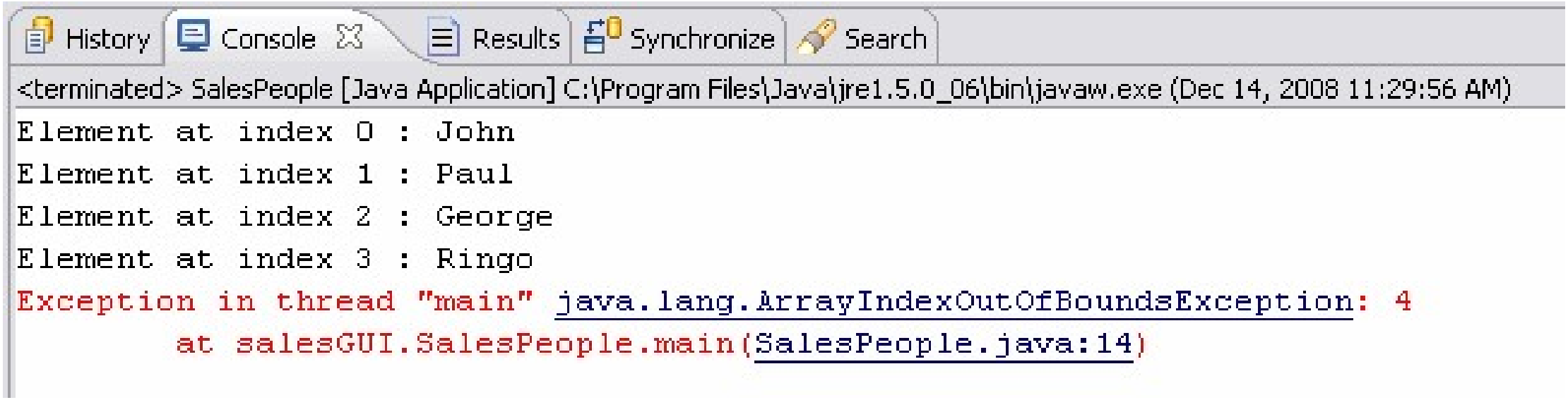
```
m[0].length
```

- Idea similar para otras dimensiones



Vectores y matrices

- Si se intenta acceder a una posición inválida, fuera del intervalo [0, n-1], se crea un error en tiempo de ejecución



Vectores y matrices

- Podemos crearlo en una sola operación:

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

- También podemos crearlo por pasos:
  - Declarar una variable para contenerlo

```
int [][] matriz;
```

  - Indica el tamaño usando new

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

  - Dar valor a cada una de las casillas

```
matriz[0][0] = 1;
matriz[0][1] = 2;
matriz[1][0] = 3;
matriz[1][1] = 4;
```

Ejemplo: cálculo de la media de un vector

```
public class Media {

   public static void main(String[] args) {

      int[] v = {50, 200, 0, 100, 150};
      int suma = 0;
      for (int i = 0; i < v.length; i++)
         suma = suma + v[i];
      double media = ((double) suma) / v.length;
      System.out.println("Media: " + media);

   }

}
```

Creación de un vector de 5 enteros

Ejemplo: cálculo de la media de un vector

```
public class Media {

   public static void main(String[] args) {

      int[] v = {50, 200, 0, 100, 150};
      int suma = 0;
      for (int i = 0; i < v.length; i++)
         suma = suma + v[i];
      double media = ((double) suma) / v.length;
      System.out.println("Media: " + media);

   }

}
```

En cada paso se accede al valor `i`-ésimo del vector y se añade a la suma acumulada

Ejemplo: cálculo de la media de un vector

```
public class Media {

   public static void main(String[] args) {

      int[] v = {50, 200, 0, 100, 150};
      int suma = 0;
      for (int i = 0; i < v.length; i++)
         suma = suma + v[i];
      double media = ((double) suma) / v.length;
      System.out.println("Media: " + media);

   }

}
```

Se repite para toda posición entre 0 y n - 1

Ejemplo: cálculo de la media de un vector

```
public class Media {

   public static void main(String[] args) {

      int[] v = {50, 200, 0, 100, 150};
      int suma = 0;
      for (int i = 0; i < v.length; i++)
         suma = suma + v[i];
      double media = ((double) suma) / v.length;
      System.out.println("Media: " + media);

   }

}
```

Una vez sumados todos los valores del vector, solamente falta dividir entre la longitud n

Ejemplo: cálculo de la media de un vector

```
public class Media {

   public static void main(String[] args) {

      int[] v = {50, 200, 0, 100, 150};
      int suma = 0;
      for (int i = 0; i < v.length; i++)
         suma = suma + v[i];
      double media = ((double) suma) / v.length;
      System.out.println("Media: " + media);

   }

}
```

El método `main` puede recibir un vector de parámetros, aunque nosotros no lo utilizaremos

Vectores y matrices

- Para hacer operaciones con vectores es frecuente utilizar un bucle `for` para hacer operaciones individuales con cada casilla
- ¿Cómo leer un vector de teclado?
  - Bucle `for` para leer cada casilla como ya sabemos
- ¿Cómo escribir un vector por pantalla?
  - Bucle `for` para escribir cada casilla junto a la anterior
- ¿Y si tenemos una matriz de varias dimensiones?
  - Usamos n bucles anidados, uno por dimensión

```
for (int i = 0; i < m.length; i++)
    for (int j = 0; j < m[0].length; j++)
        m[i][j] = escaner.nextInt();
```

Modularidad

- Un problema complejo se descompone en otros más simples
- Un modelo se descompone en módulos que permiten dividir el sistema en otros más tratables

Modularidad

- ¿Qué sucede cuando ejecutamos `sqrt(discriminante)`?
- En una sola línea hemos obtenido la raíz cuadrada, pero para calcularla hacen falta más líneas de código
  - Hemos dividido un problema en varias partes
  - Una de estas partes se puede reutilizar en varios casos
- ¿Cómo calcula Java la raíz cuadrada? ¿método clásico? ¿bisección? ¿Newton-Raphson? ¿serie de Taylor? …
  - Ni sabemos cómo está implementado, ni nos importa
  - Lo importante es que podamos utilizarlo
- El mecanismo básico es la definición de funciones o métodos
  - Los lenguajes permiten definir las nuestras propias para no tener que basarnos únicamente en las predefinidas

Modularidad

- Nº subconjuntos de *p* elementos a partir de un conjunto de *m*

$$C_n^p = \binom{n}{p} = \frac{n!}{p!(n-p)!}$$

- Cálculo del factorial:

```
leer(n, p);
fact := 1;
para (i := 2; i <= n; i = i + 1)
   fact := i * fact;
escribir(fact);
```

Modularidad

- Pseudocódigo para $C_n^p = \binom{n}{p} = \frac{n!}{p!(n-p)!}$

```
leer(n, p);
fact := 1;
para (i := 2; i <= n; i = i + 1)
  fact := i * fact;
factN := fact;
fact := 1;
para (i := 2; i <= p; i = i + 1)
  fact := i * fact;
factM := fact;
fact := 1;
para (i := 2; i <= n - p; i = i + 1)
  fact := i * fact;
factN_M := fact;
comb := factN / (factP * factN_P);
escribir(comb);
```

Modularidad

- Supongamos que existiera una función *factorial(x)*
- Pseudocódigo para $C_n^p = \binom{n}{p} = \frac{n!}{p!(n-p)!}$

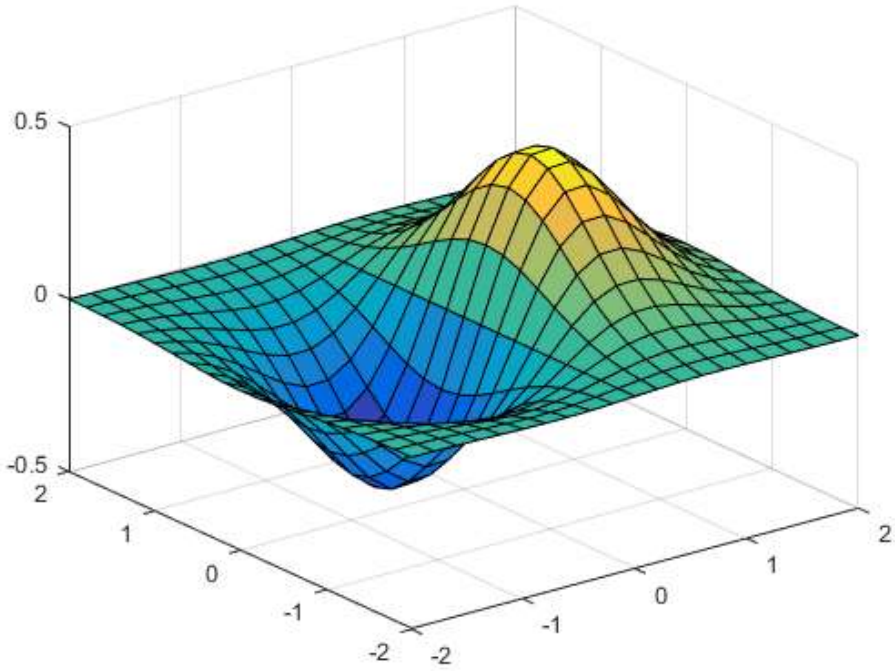
```
leer(n, p);
factN := factorial(n);
factP := factorial(p);
factN_P := factorial(n - p);
comb := factN / (factP * factN_P);
escribir(comb);
```

Ventajas de la modularidad

- Reusabilidad del módulo para diferentes problemas
- Mantenimiento: si fuera necesario modificar el código, solo habría que hacer cambios en un lugar
- Legibilidad del código
- Simplicidad: descomposición de un programa en partes más sencillas de resolver
- Abstracción: podemos usar un módulo como si fuera una instrucción sencilla y sin saber cómo está hecho
- Generalidad: el problema se resuelve para varios casos
- Fiabilidad: las partes se pueden probar por separado

Funciones en matemáticas

- Definición de la función:
  - $f : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$
  - $f(x, y) = x \cdot e^{-(x^2+y^2)}$
- Uso de la función:
  - f (0, 1) = 0
  - f (1, 1) = 1 / e^2

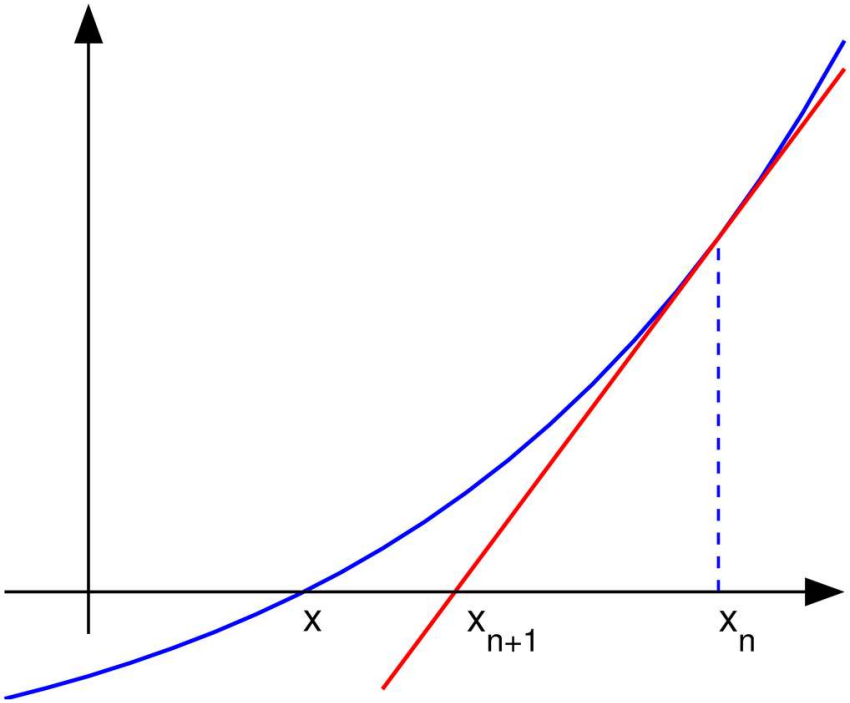


- Parámetros de la función (dominio): 2 reales
- Tipo devuelto por la función (codominio/rango): 1 real

Ejemplo: cálculo de la raíz cuadrada

Método de Newton-Raphson

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x_n^2 - S}{2x_n} = \frac{1}{2}\left(x_n + \frac{S}{x_n}\right)$$




```
public static double sqrt(double s) {
    // Supone que s >= 0
   double EPSILON = 1E-15;
   double t = s;
   while (Math.abs(s - t * t) > EPSILON)
       t = (t + s / t) / 2;
   return t;
}
```

Funciones

- Es diferente la definición de función y la llamada a función
  - Llamada: usa una función en un programa
  - Definición: explica qué hace la función cuando se llame
- Las funciones pueden tener (o no) uno o varios parámetros
  - Como son independientes, son el modo de comunicarse entre sí (si no, ¿cómo sabe de qué valor calcular la raíz?)
  - Al definir la función solamente decimos el nº de parámetros y el tipo de cada uno de ellos (parámetros formales)
  - En el momento de la llamada se dan los parámetros reales
- Una función devuelve un valor al subprograma que la llama
  - Al definir la función solamente decimos el tipo del valor
  - ¿Y si no queremos devolver nada? Se usa el tipo `void`

Flujo de ejecución con módulos

- Al ejecutar una clase Java, se ejecuta el código del `main`
- Si surge una llamada a una función, se detiene la ejecución del subprograma actual y se pasa a ejecutar dicha función
  - Se evalúa el valor de los parámetros reales en la llamada y se asignan a los parámetros formales esperados
  - Se ejecuta el código de la función
  - Si aparece la palabra `return`, se finaliza la ejecución de la función, se devuelve el valor al subprograma que llamó a la función y se continúa la ejecución del subprograma
- Debe haber una coincidencia entre los tipos reales y los esperados (los indicados en la definición de la función)
  - Tipo de los parámetros
  - Tipo del valor devuelto

Ejemplo: cálculo de la raíz cuadrada

```
public class NewtonRaphson {

   public static double sqrt(double s) {
      // Supone que s >= 0
      double EPSILON = 1E-15;
      double t = s;
      while (Math.abs(s - t * t) > EPSILON)
         t = (t + s / t) / 2;
      return t;
   }

   public static void main (String[] args)
   {
      int x = 2;
      double sol = sqrt(2);
      System.out.println("Raiz de " + x + " = " + sol);
   }
}
```

Ejemplo: cálculo de la raíz cuadrada

Código con 2 funciones

```
public class NewtonRaphson {

   public static double sqrt(double s) {
      // Supone que s >= 0
      double EPSILON = 1E-15;
      double t = s;
      while (Math.abs(s - t * t) > EPSILON)
         t = (t + s / t) / 2;
      return t;
   }

   public static void main (String[] args)
   {
      int x = 2;
      double sol = sqrt(2);
      System.out.println("Raiz de " + x + " = " + sol);
   }
}
```

Ejemplo: cálculo de la raíz cuadrada

```
public class NewtonRaphson {

   public static double sqrt(double s) {
      // Supone que s >= 0
      double EPSILON = 1E-15;
      double t = s;
      while (Math.abs(s - t * t) > EPSILON)
         t = (t + s / t) / 2;
      return t;
   }

   public static void main (String[] args)
   {
      int x = 2;
      double sol = sqrt(2);
      System.out.println("Raiz de " + x + " = " + sol);
   }
}
```

Comienza ejecutándose el código del `main`

Ejemplo: cálculo de la raíz cuadrada

```
public class NewtonRaphson {

   public static double sqrt(double s) {
      // Supone que s >= 0
      double EPSILON = 1E-15;
      double t = s;
      while (Math.abs(s - t * t) > EPSILON)
         t = (t + s / t) / 2;
      return t;
   }

   public static void main (String[] args)
   {
      int x = 2;
      double sol = sqrt(2);
      System.out.println("Raiz de " + x + " = " + sol);
   }
}
```

La llamada a `sqrt` modifica el flujo de ejecución y `s` toma el valor 2

Ejemplo: cálculo de la raíz cuadrada

```
public class NewtonRaphson {

   public static double sqrt(double s) {
      // Supone que s >= 0
      double EPSILON = 1E-15;
      double t = s;
      while (Math.abs(s - t * t) > EPSILON)
         t = (t + s / t) / 2;
      return t;
   }

   public static void main (String[] args)
   {
      int x = 2;
      double sol = sqrt(2);
      System.out.println("Raiz de " + x + " = " + sol);
   }
}
```

Se ejecuta el código de la función `sqrt`

Ejemplo: cálculo de la raíz cuadrada

```
public class NewtonRaphson {

   public static double sqrt(double s) {
      // Supone que s >= 0
      double EPSILON = 1E-15;
      double t = s;
      while (Math.abs(s - t * t) > EPSILON)
         t = (t + s / t) / 2;
      return t;
   }

   public static void main (String[] args)
   {
      int x = 2;
      double sol = sqrt(2);
      System.out.println("Raiz de " + x + " = " + sol);
   }
}
```

La función devuelve al `main` el valor de `t` (1.414213562373095)

Ejemplo: cálculo de la raíz cuadrada

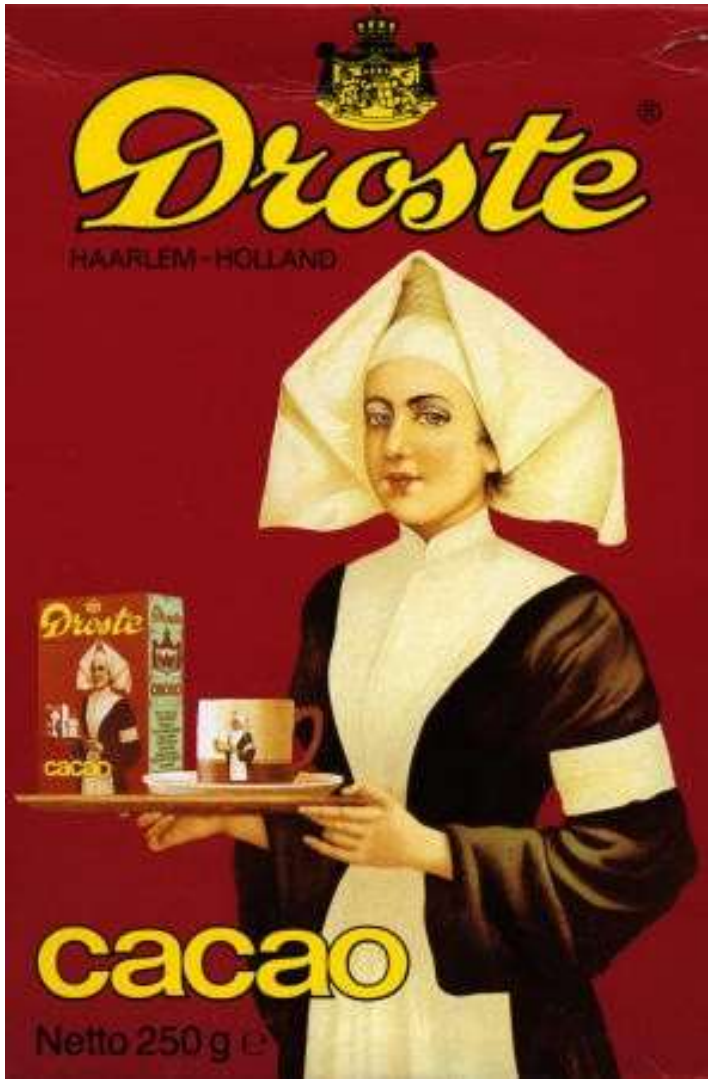
```
public class NewtonRaphson {

   public static double sqrt(double s) {
      // Supone que s >= 0
      double EPSILON = 1E-15;
      double t = s;
      while (Math.abs(s - t * t) > EPSILON)
         t = (t + s / t) / 2;
      return t;
   }

   public static void main (String[] args)
   {
      int x = 2;
      double sol = sqrt(2);
      System.out.println("Raiz de " + x + " = " + sol);
   }
}
```

Continúa la ejecución del código del `main`

Recursividad

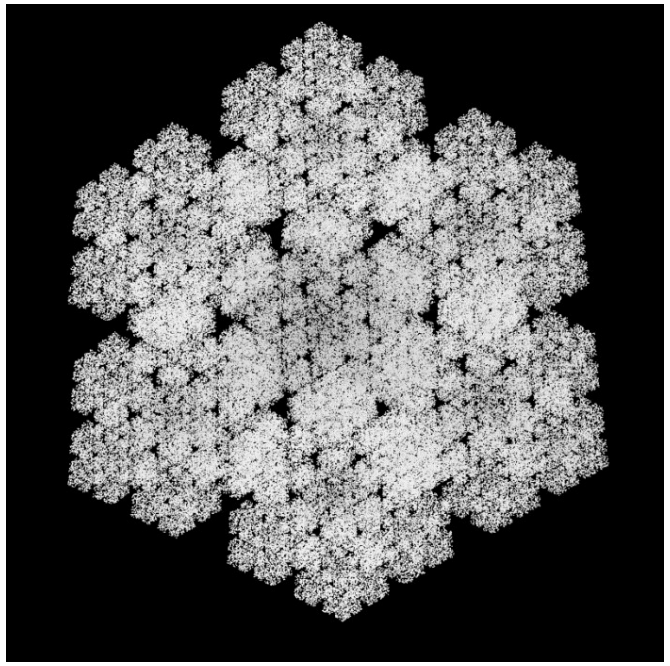
- Una función puede llamarse a sí misma
  - Evitar la recursividad infinita

```
public static int fibonacci (int n)
{
  if (n == 0)
    return 0;
  else
  {
    if (n == 1)
      return 1;
    else
      return fibonacci(n - 1) +
             fibonacci(n - 2);
  }
}
```



Fractales

- Un fractal es una figura
  - Auto-semejante
  - Que contiene copias de sí misma
  - Definida de forma recursiva
- Fractal regular: objeto construido a partir de copias exactas (escaladas) de sí mismo
- Fractal no regular: objeto auto-semejante pero no construido sólo a partir de copias exactas de sí mismo
- Muchos ejemplos en la naturaleza
- Muchas aplicaciones en diseño artificial



Triángulo de Sierpinski

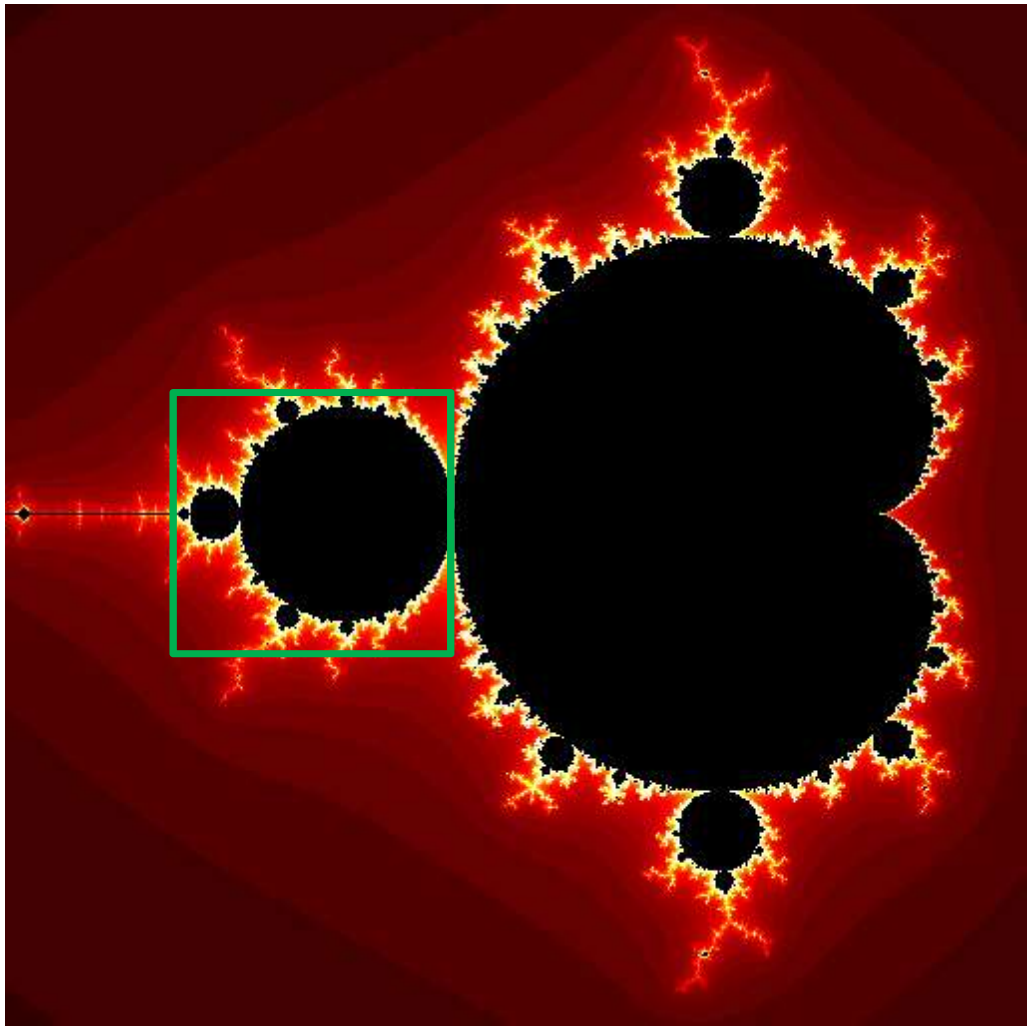
Conjunto de Mandelbrot

- Para todo número complejo c ∈ ℂ, se define la sucesión S

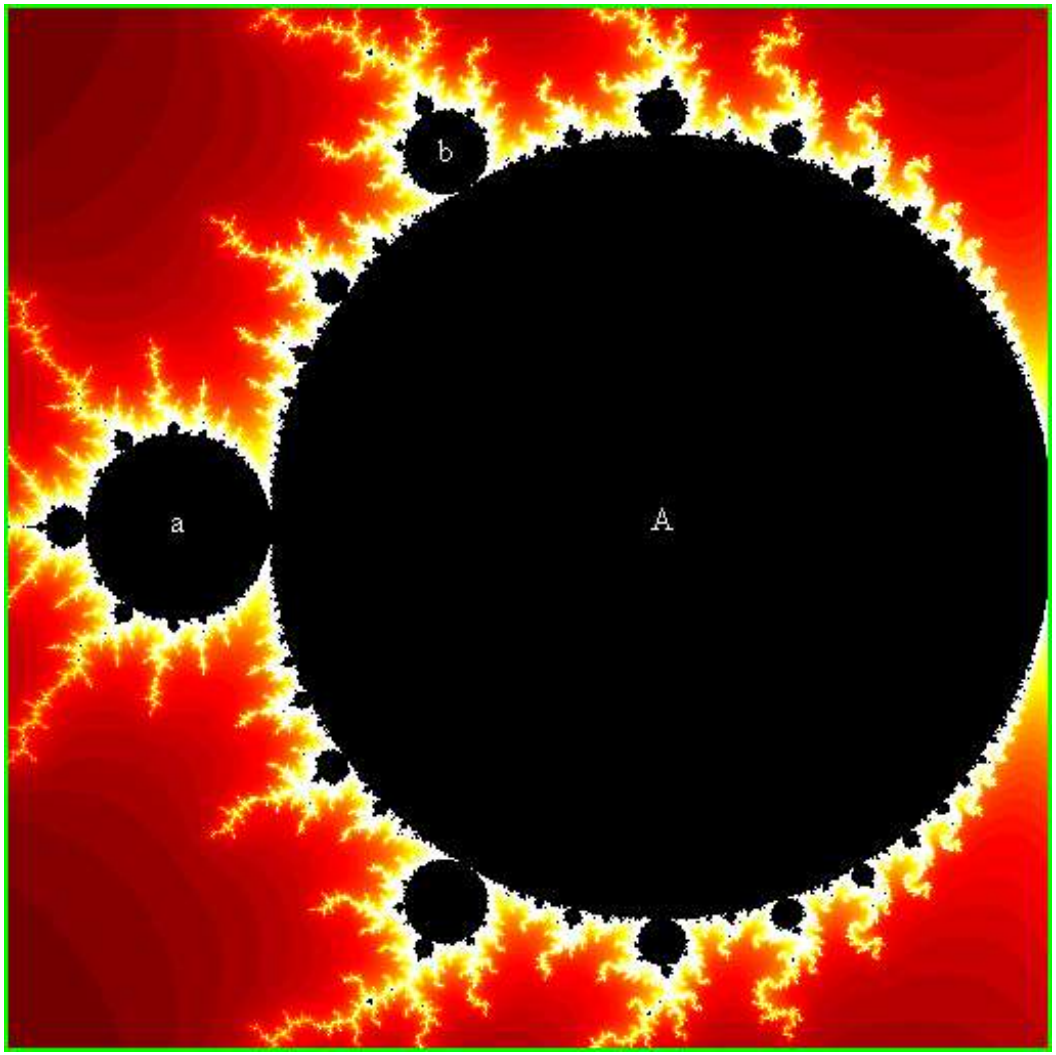
$$\begin{cases} z_0 & = & 0 \\ z_{n+1} & = & z_n^2 + c \end{cases}$$

- Si la sucesión está acotada, se asigna al punto c color negro
- Si diverge, se asigna un color dependiente del número de iteraciones necesarias para detectar la divergencia
- El complejo c = x + y i se representa como el píxel (x, y)
- Ejemplos:
  - Si c = -1, la sucesión 0, -1, 0, -1, 0… converge
  - Si c = 1, la sucesión 0, 1, 2, 5, 26… diverge
- Conjunto de Mandelbrot: c ∈ ℝ tales que S está acotada

Conjunto de Mandelbrot



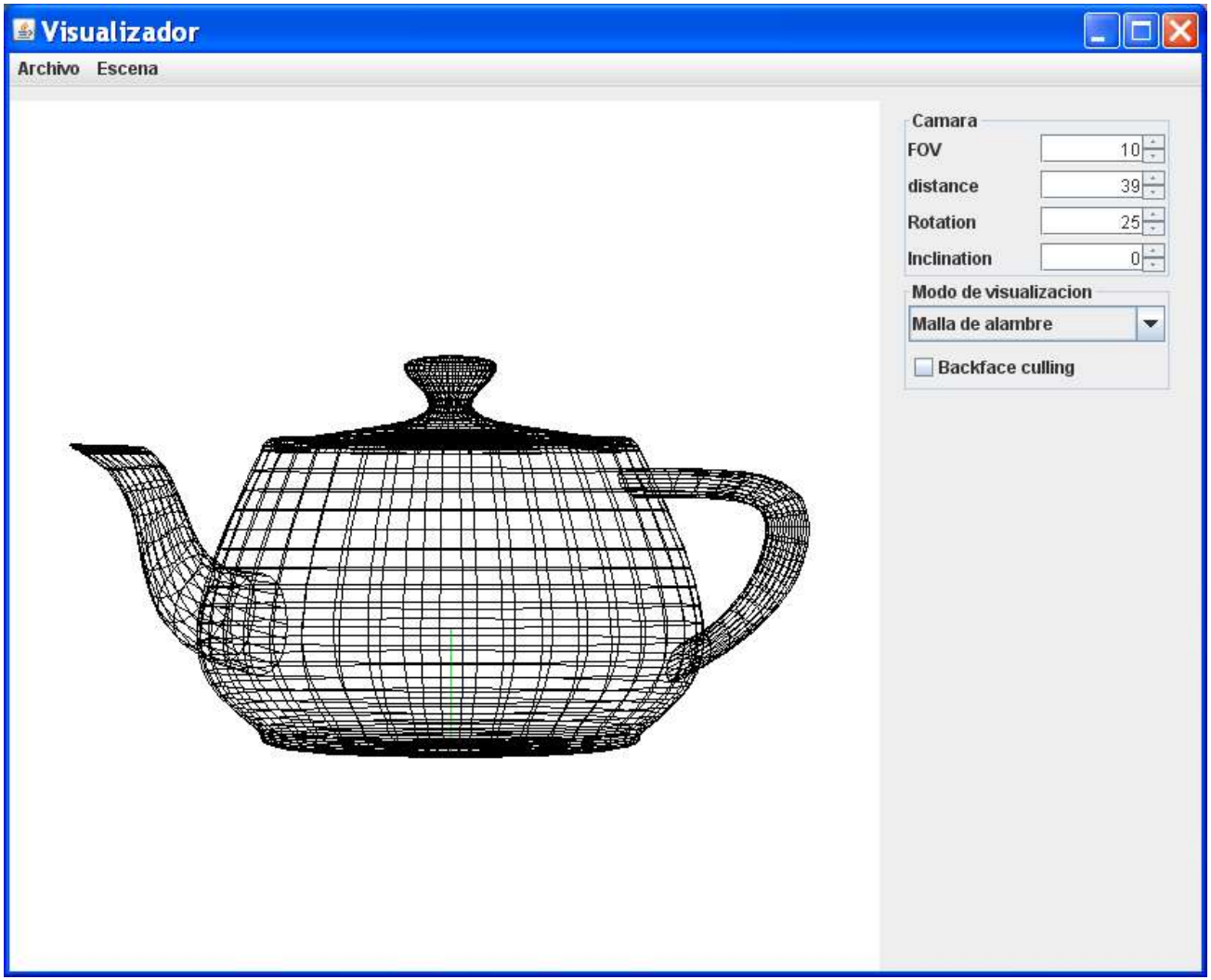
Conjunto de Mandelbrot



Programación orientada a eventos

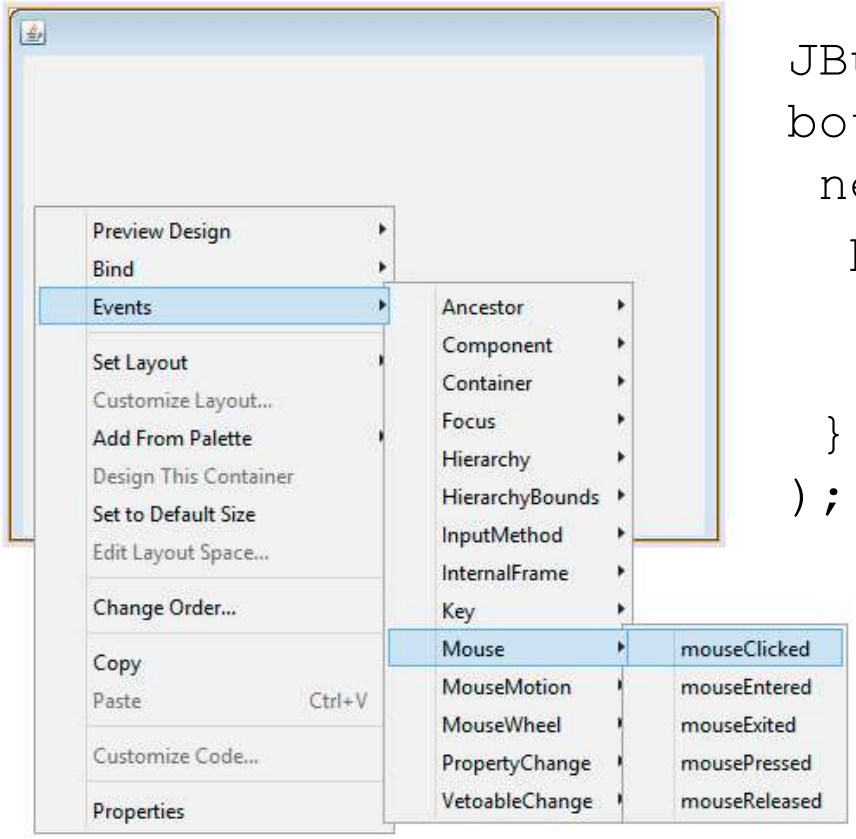
- Java y Processing permiten desarrollar programas siguiendo el paradigma de orientación a eventos
- El flujo de ejecución del programa lo determinan eventos creados por el usuario al interactuar con una interfaz gráfica
  - Ejemplo: interfaces de las aplicaciones de CAD
  - La idea es tener métodos que indiquen qué instrucciones ejecutar cuando se produzca un evento determinado
- Ejemplos de eventos en Java y Processing:
  - `mousePressed`: al pulsar un botón del ratón
  - `keyPressed`: al pulsar cualquier tecla del teclado
- Crear interfaces gráficas en Java: paquetes `awt`, `swing`…

Ejemplo: interfaz gráfica para el visualizador

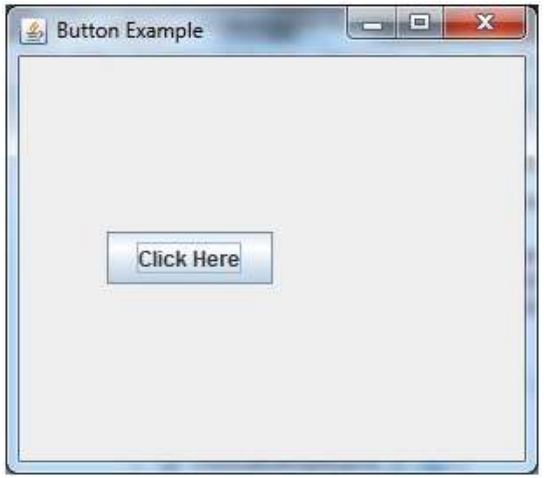


Programación orientada a eventos

- La sintaxis es un poco compleja, pero los entornos de programación ayudan creando un esqueleto que completar




```
JButton boton = new JButton("Click here");
boton.addActionListener(
 new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    System.out.println("Clicked!");
  }
 }
);
```



Programación orientada a objetos

- Java y Processing son lenguajes de programación que siguen el paradigma de orientación a objetos
- En este paradigma, los elementos del problema que se desea resolver se modelan mediante objetos y clases
- Una clase es como un tipo y un objeto como variables del tipo
  - Diferencia: los objetos tienen atributos (datos) y métodos (instrucciones)
- Lo normal es que un programa tenga más de una clase, al contrario de lo que hemos visto hasta ahora
- Al hacer lecturas de teclado, hemos usado la clase Scanner y hemos creado un nuevo objeto de esa clase:

```
Scanner escaner = new Scanner(System.int)
```

Programación orientada a objetos

- Una clase es una colección de objetos similares, un tipo que permite crear variables con características comunes

Programación orientada a objetos

- Atributos y métodos pueden pertenecer a la clase o al objeto
  - Si un atributo es propiedad del objeto, cada objeto tiene una copia diferente, pudiendo tener distinto valor
  - Los métodos de clase permiten ejecutar instrucciones sin haber creado objetos de esa clase
- Ejemplos
  - Un círculo tiene coordenadas (atributo de objeto)
  - Un círculo tiene el valor de π (atributo de clase)
  - Un círculo puede calcular su perímetro (método de objeto)
  - Un círculo puede tener un `main` ejecutable (método de clase)

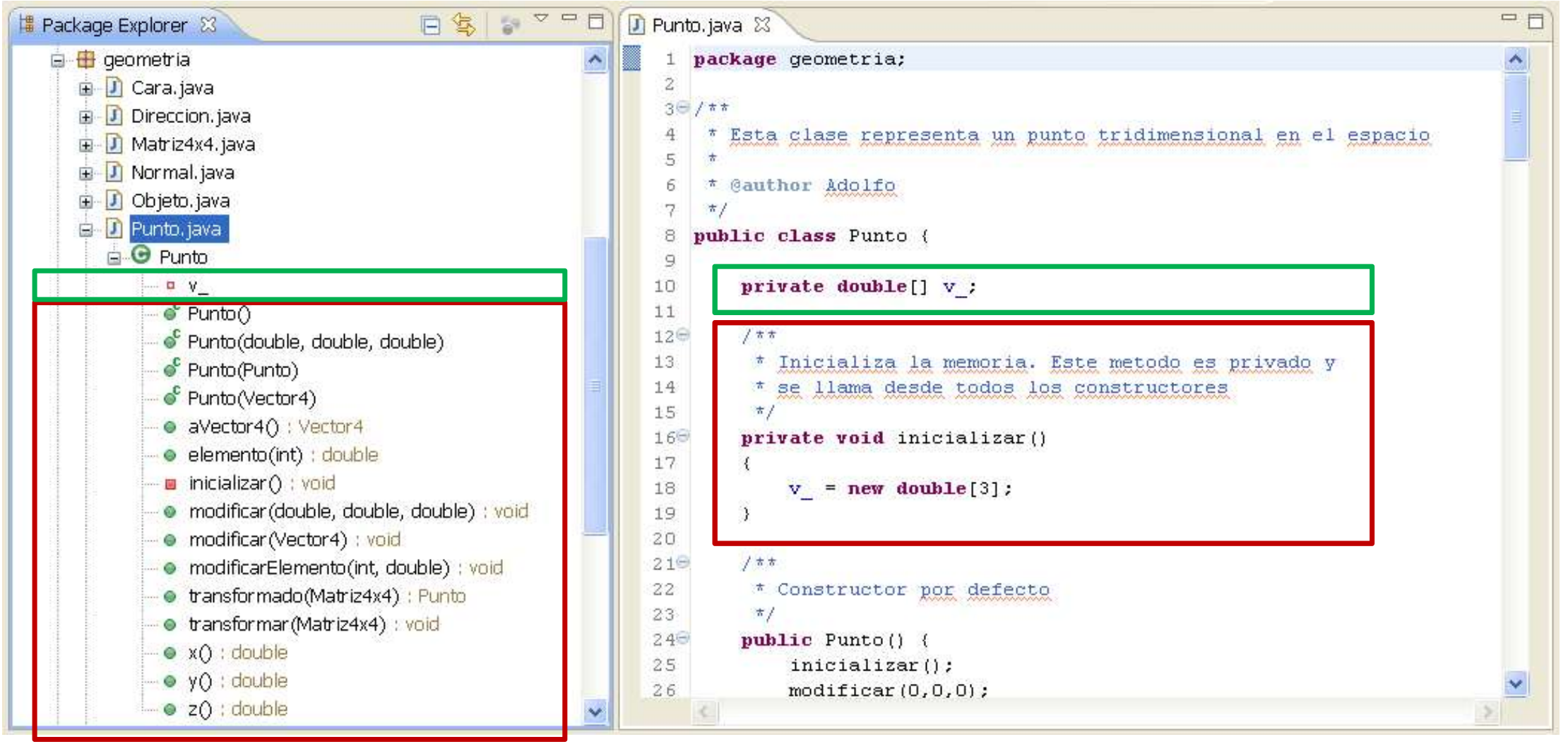
Programación orientada a objetos

- Más ejemplos de métodos y atributos de clase u objeto
  - `nextInt` es un método de objetos de la clase `Scanner`
  - `Math.sqrt` es un método de la clase `Math`
  - `v.length` es un atributo de objetos (de los arrays)
  - `Math.PI` es un atributo de la clase `Math`
- ¿Cómo indicar si pertenece a la clase o al objeto?
  - `static` indica que un método o atributo es de clase
- A veces queremos ocultar información, no permitiendo que se muestren todas las clases por seguridad, comodidad…
  - Las clases, sus métodos y sus atributos pueden ser públicos (`public`), privados (`private`), mixtos (`protected`)…
  - Hasta ahora siempre han sido públicos

Ejemplo: clase Quicksort

```
public class Quicksort {
   public void quicksort(int[] a, int izq, int der) {
      int i = izq;
      int j = der;
      int pivote = a[(izq + der) / 2];
      // ...
   }
   public void array() {
      Random r = new Random();
      Scanner sc = new Scanner(System.in);
      System.out.print("Dimensión de la lista de números: ");
      // ...
   }
   public static void main(String[] args) {
      Quicksort3 q = new Quicksort3();
      q.array();
   }
}
```

Ejemplo: clase Punto



Herencia

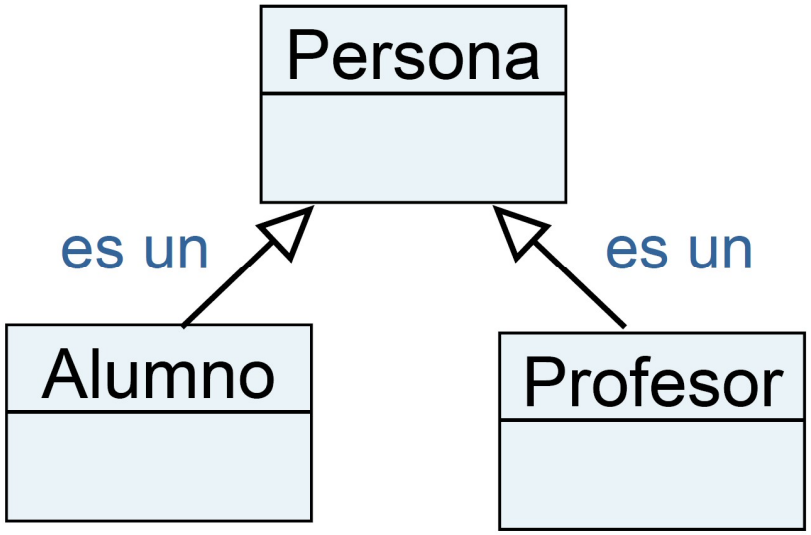
```
public class Alumno
{
        private String nombre;
        private int edad;
        private int nip;
        private Asignatura[] matricula;

        public String getNombre() { ... }

        public int getEdad() { ... }

        public int getNip() { ... }

        public int hallarCursoMasAltoMatriculado() { ... }
}
```

Herencia

```
public class Profesor
{
        private String nombre;
        private int edad;
        private int nip;
        private double sueldoBase;
        private int añosTrabajados;

        public String getNombre() { ... }

        public int getEdad() { ... }

        public int getNip() { ... }

        public int public double hallarSueldo() { ... }
}
```

Herencia

- Las clases `Alumno` y `Profesor` comparten:
  - Algunos atributos: `nombre`, `edad`, `nip`
  - Algunos métodos: `getNombre()`, `getEdad()`, `getNip()`
- ¿Es necesario escribir el mismo código 2 veces?
  - Usando el mecanismo de herencia no es necesario



Herencia

```
public class Persona
{
        private String nombre;
        private int edad;
        private int nip;

        public String getNombre() { ... }

        public int getEdad() { ... }

        public int getNip() { ... }
}
```

Herencia

```
public class Alumno extends Persona
{
        private Asignatura[] matricula;

        public int hallarCursoMasAltoMatriculado() { ... }
}

public class Profesor extends Persona
{
        private double sueldoBase;
        private int añosTrabajados;

        public int public double hallarSueldo() { ... }
}
```

Pros y contras de la POO

- ✓ Abstracción de detalles y encapsulación de información
- ✓ Facilidad de diseño
- ✓ Reusabilidad del código
- ✓ Escalabilidad del código
- ✓ Mantenimiento del código, facilitando los cambios
- ✓ Compresión de datos y sus métodos en clases
- ✓ Fiabilidad, al poder probar partes por separado
- ✗ Mayor dificultad de aprendizaje
- ✗ Menor eficiencia de los programas
- ✗ Mayor dificultad de seguir el flujo de ejecución
- ✗ Sobreestimación del diseño sobre los algoritmos