FUNDAMENTOS DE PROGRAMACIÓN

1 – Introducción a Java

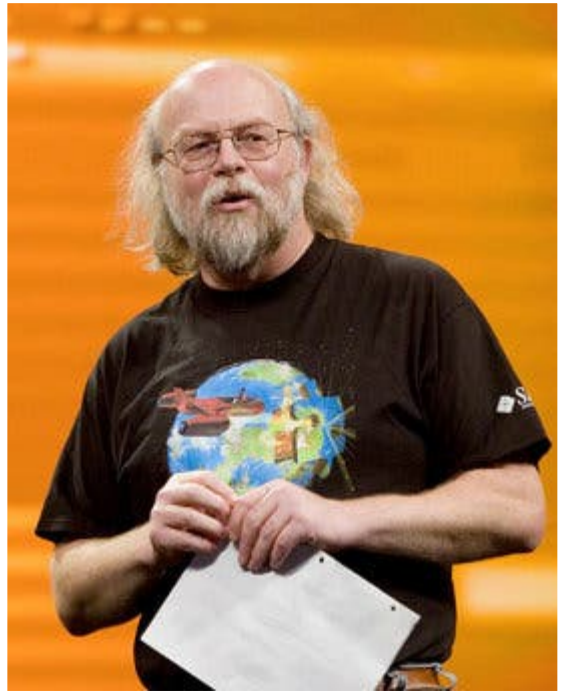
UNIVERSIDAD DE SEVILLA

ÍNDICE

ÍNDICE

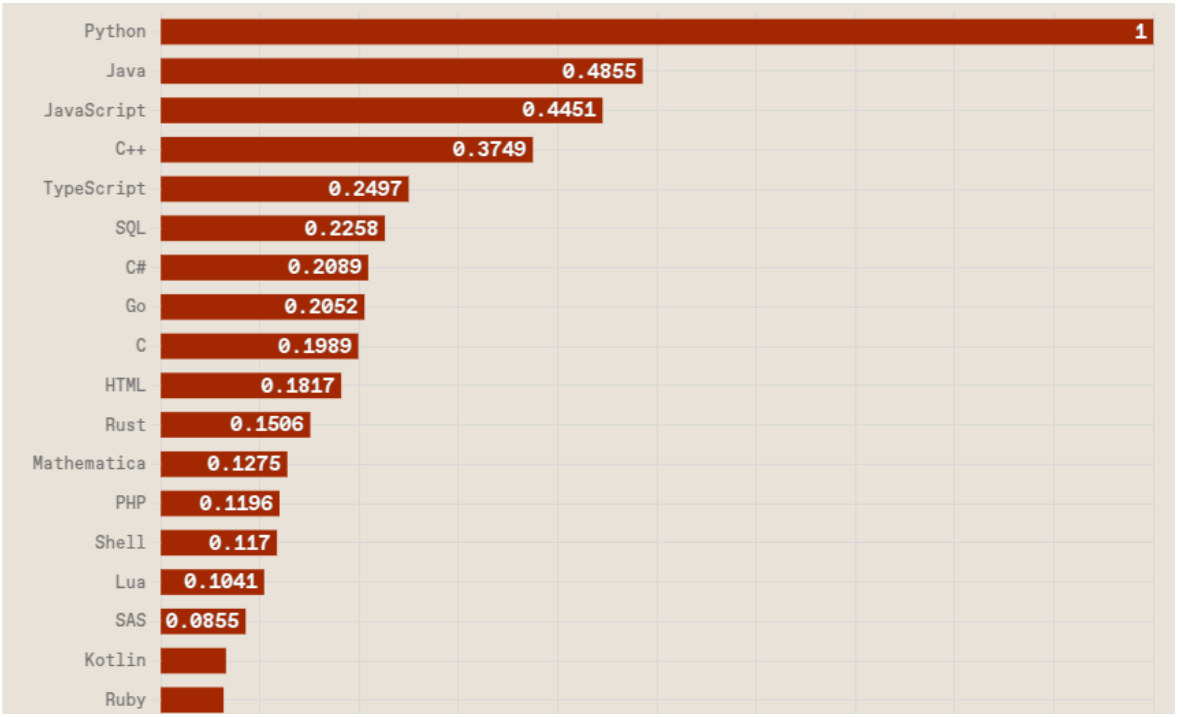
1. Introducción

- Lenguaje de programación para desarrollar aplicaciones.
- Programación orientada a objetos (POO).
- POO surgen en los 80s (Smalltalk) => C++ 90s.
- Desarrollado por James Gosling (Sun Microsystems). JDK (JDK 1.0) en 1991.
- Un lenguaje de programación parecido a C++ en **estructura y sintaxis, orientado a objetos y con una máquina virtual propia**.



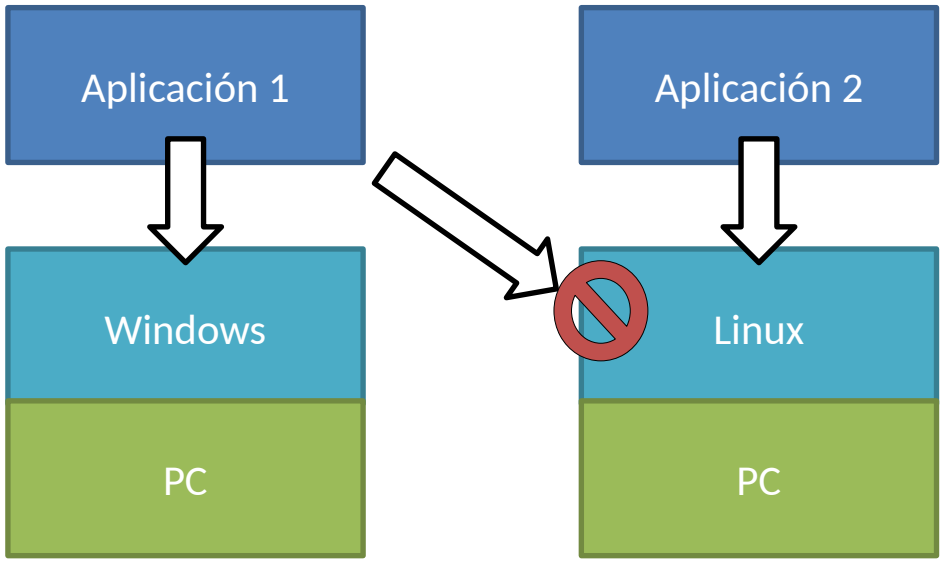
1. Introducción

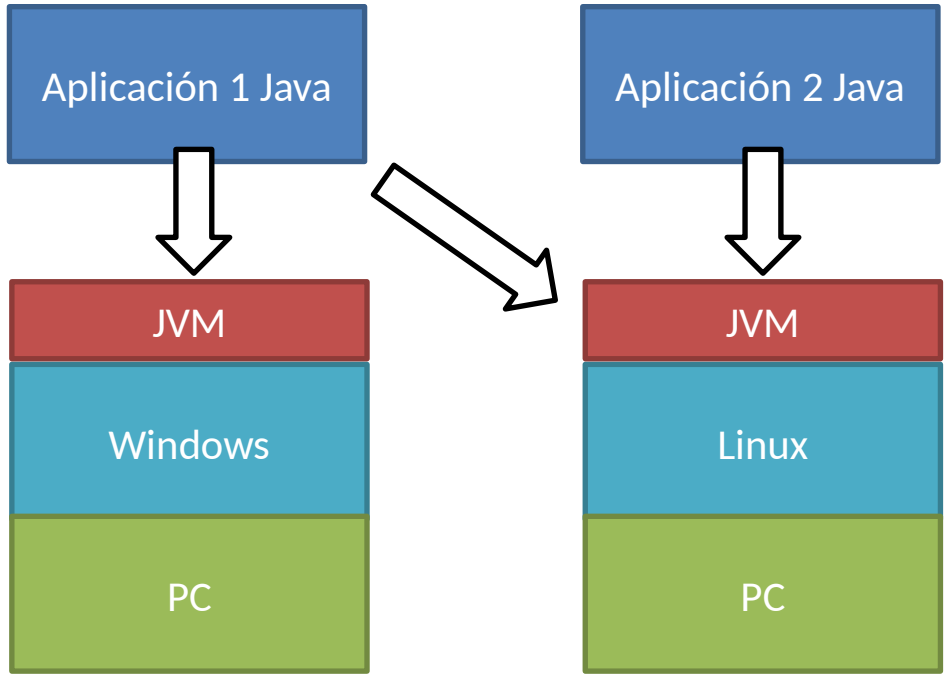
- Top programming languages by IEEE Spectrum 2023



https://spectrum.ieee.org/top-programming-languages-2024

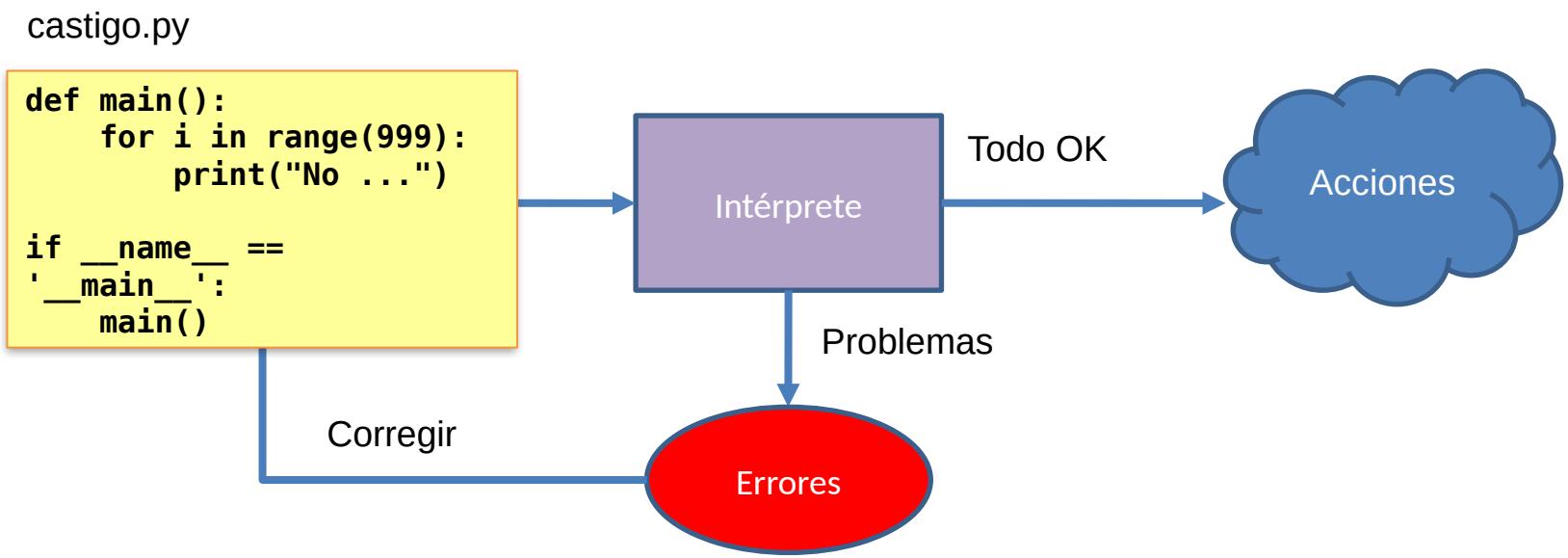
1. Introducción





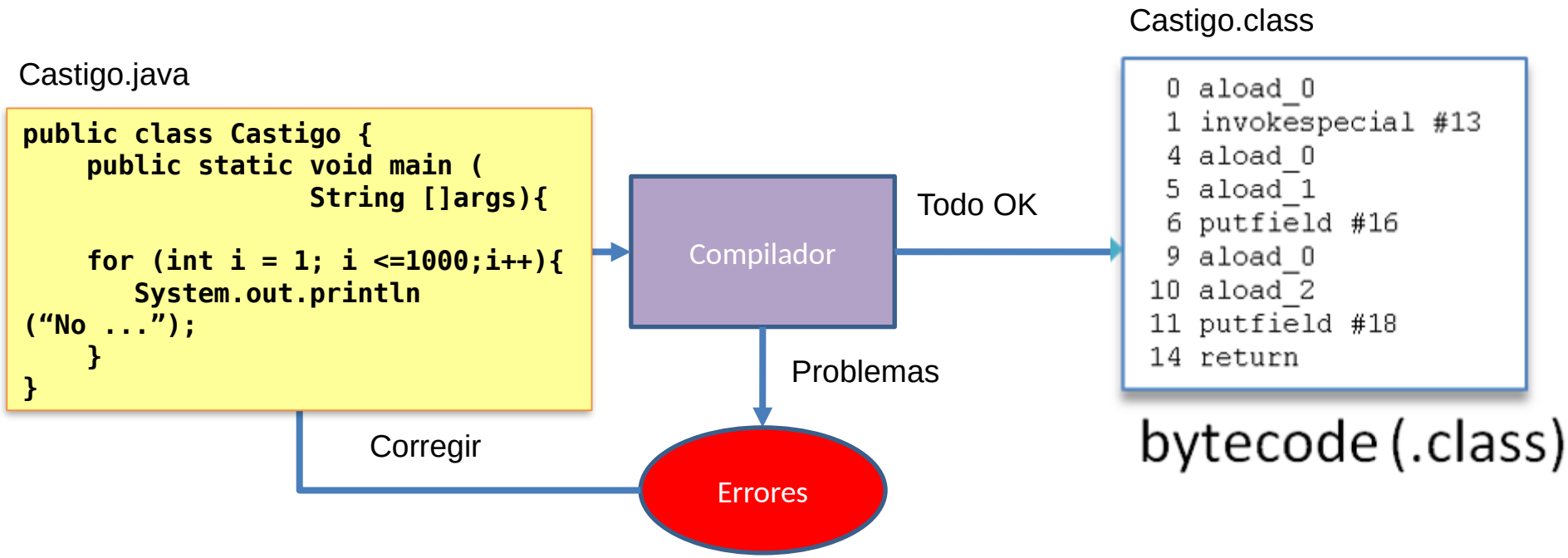
1. Introducción - Intérprete

- Programa informático que **analiza y ejecuta** un programa escrito en un lenguaje de programación



1. Introducción - Compilador

- Programa informático que **traduce** un programa escrito en un lenguaje de programación al lenguaje máquina



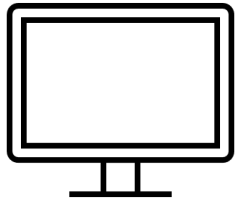
1. Introducción
¿Cómo trabajábamos en Python?

CSV

Carga fichero

test_script.py

Ejecutar funciones



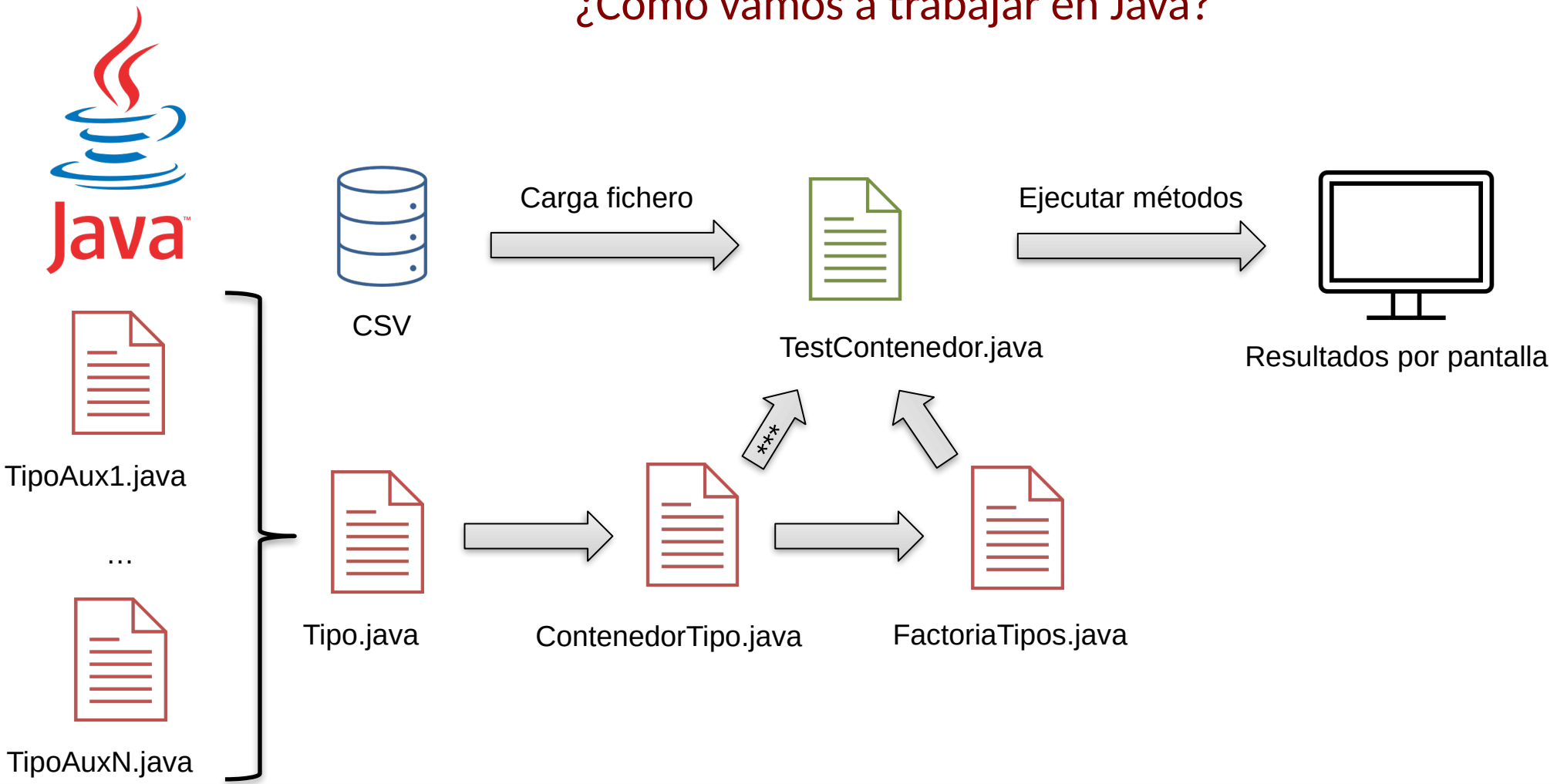
Resultados por pantalla

Importar funciones

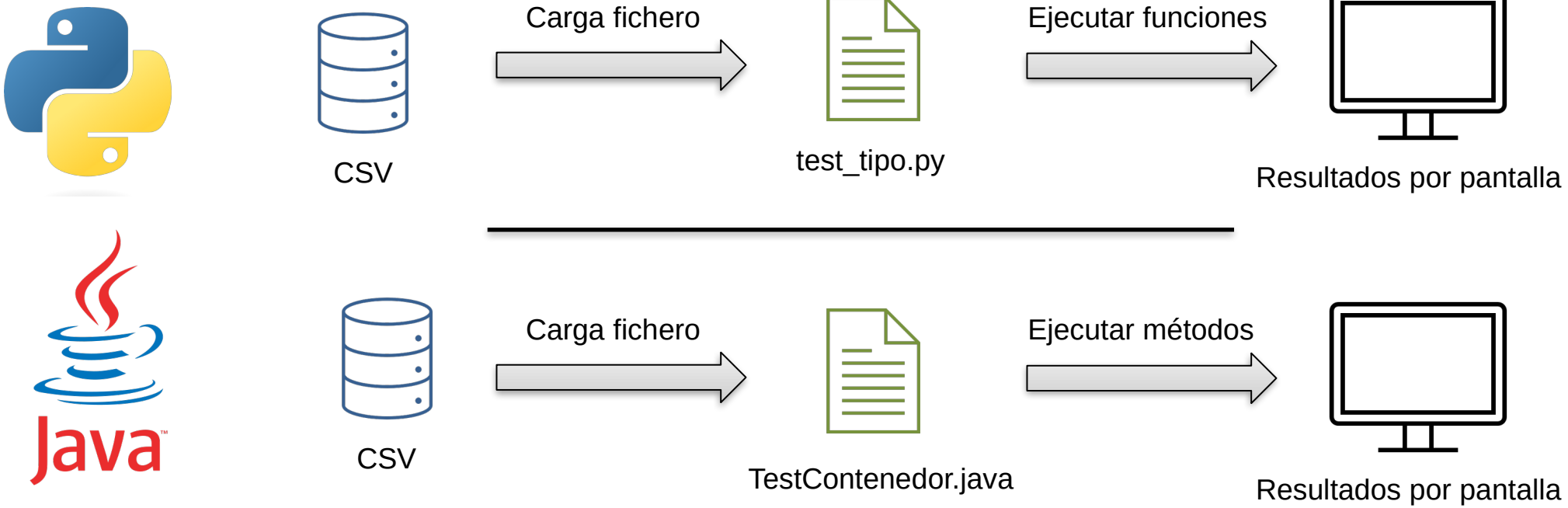
NamedTuple
parser_aux
función_1
función_2
…
función_n
función_aux1
…

script.py

1. Introducción
¿Cómo vamos a trabajar en Java?



1. Introducción
¿Y en qué se parecen?



1. Introducción
¿Y en qué se parecen?

NamedTuple
parser_aux
~~función_1~~
~~función_2~~
…
~~función_n~~
~~función_aux1~~
…

script.py

TipoAux1.java

…

TipoAuxN.java

Tipo.java

1. Introducción
¿Y en qué se parecen?

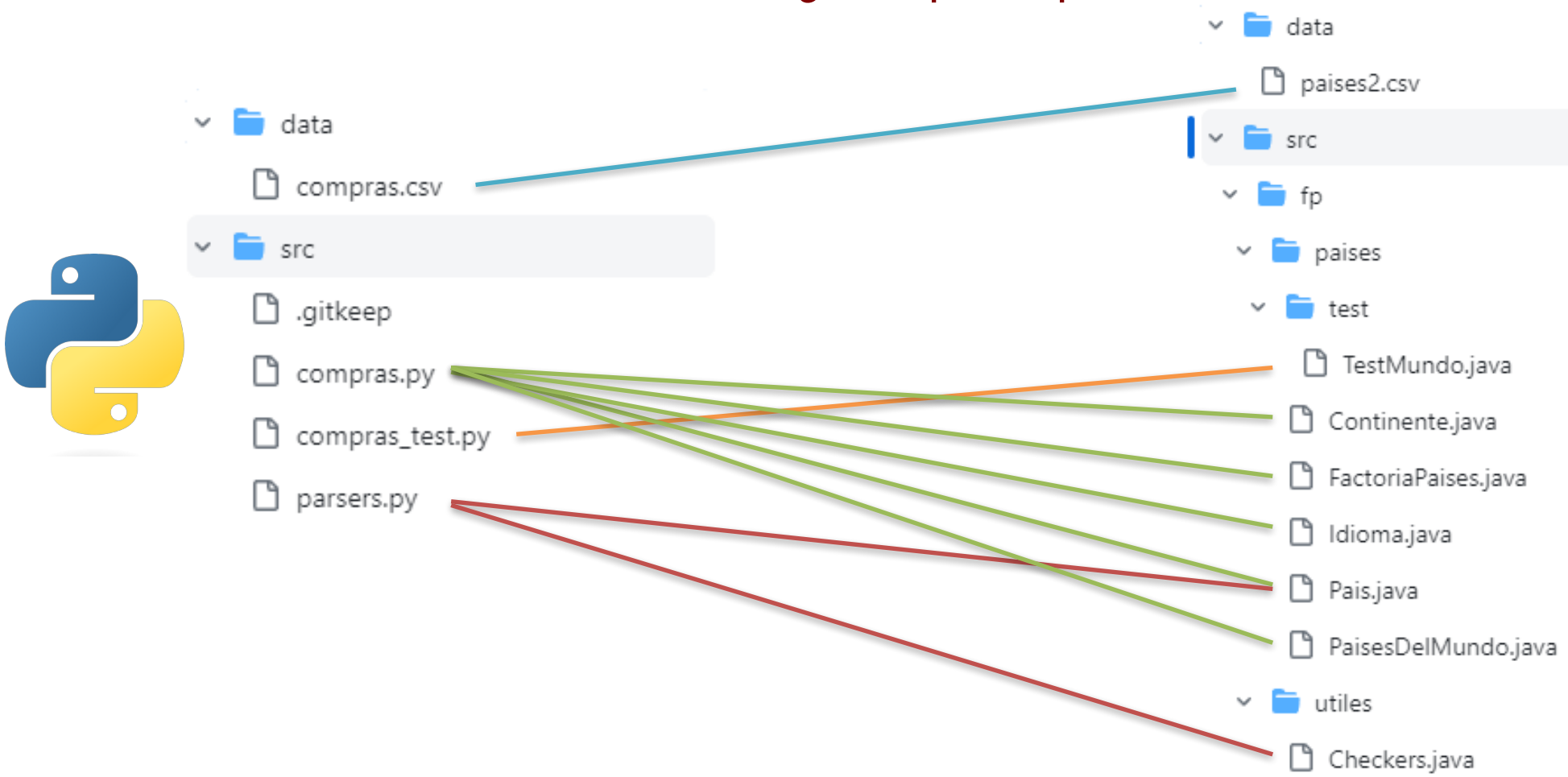
~~NamedTuple~~
~~parser_aux~~
función_1
función_2
…
función_n
función_aux1
…

tipo.py

ContenedorTipos.java

FactoriaTipos.java

1. Introducción
¿Y en qué se parecen?



ÍNDICE

2. Conceptos básicos de la P.O.O.

- Los elementos básicos de la POO son:
  - **Objeto**
  - **Interfaz**
  - **Clase**
    - **Atributos** (almacenan las propiedades)
    - **Métodos** (consultan o actualizan las propiedades)
  - **Paquete**

2. Conceptos básicos de la P.O.O.

Properties

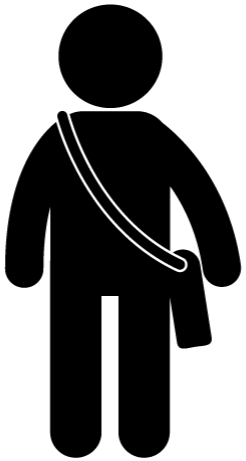
nombre
edad
dni
grupo
nota1
nota2
nota3

Methods

unirseAGrupo()
verNotas()
solicitarErasmus()
calcularNotaFinal()

Class

Estudiante



Objects

estudiante1
- "Pilar"
- 19
- 12345678A
- G1
- 4.25
- 7.5
- 9.75

estudiante2
- "Javier"
- 20
- 87654321Z
- G2
- 6.75
- 4.5
- 8.75

estudiante3
- "Alejandro"
- 18
- 18273645H
- G2
- 6.00
- 7.00
- 6.75

2. Conceptos básicos de la P.O.O.
Clase

- Las **clases** son las unidades de la POO que permiten definir los detalles del **estado interno** de un objeto (mediante los **atributos**), calcular las **propiedades** de los objetos a partir de los atributos e implementar las **funcionalidades** ofrecidas por los objetos (a través de los **métodos**).

```
[Modificadores] class NombreClase [extends ...] [implements ...]  {
          [atributos]
          [métodos]
}

public class Estudiante {

}
```

2. Conceptos básicos de la P.O.O.
Objeto

- Los objetos tienen una identidad, unas propiedades, un estado y una funcionalidad asociada:
  - **Identidad**: nombre que se le asigna al objeto para referirnos a él.
  - **Propiedades/Atributos:** características observables de un objeto desde el exterior del mismo. Pueden ser de diversos tipos (números enteros, reales, textos, booleanos, etc.). Las propiedades de un objeto pueden ser básicas y derivadas.
  - El **estado** indica cuál es el valor de sus propiedades en un momento dado.
  - La **funcionalidad** de un objeto se ofrece a través de un conjunto de **métodos**. Los métodos actúan sobre el estado del objeto (pueden consultar o modificar las propiedades) y son el mecanismo de comunicación del objeto con el exterior.

2. Conceptos básicos de la P.O.O.
Identificadores

- Son palabras que permiten referenciar los diversos elementos que constituyen el código. Los identificadores se construyen mediante una secuencia de letras, dígitos, o los símbolos _ y $. En cualquier caso, se debe observar que:
  - No pueden coincidir con palabras reservadas de Java
  - Deben comenzar por una letra, _ o $, aunque estos dos últimos no son aconsejables.
  - Pueden tener cualquier longitud.
  - Son sensibles a las mayúsculas, por ejemplo, el identificador min es distinto de MIN o de Min.

```
//Identificadores válidos
tiempo, distancia1, caso_A, PI, velocidad_de_la_luz
//Identificadores inválidos
1_valor, tiempo-total, dolares%, final
```

2. Conceptos básicos de la P.O.O.
Comentarios

- Los comentarios son un tipo especial de separadores que sirven para explicar o aclarar algunas sentencias del código, por parte del programador, y ayudar a su prueba y mantenimiento.
- Los comentarios son ignorados por el compilador.

```
//Este es un comentario de una línea

/* Este es un bloque de comentario
que ocupa varias líneas
nada de lo que haya aquí lo leerá el compilador
*/

/*
* Este también es un bloque de comentario que ocupa
* varias líneas nada de lo que haya
*/
```

ÍNDICE

3. Tipos de datos

- Java es un lenguaje compilado interpretado y fuertemente tipado. Los tipos básicos son:
  - **int, long**: representan valores de tipo entero y sus operaciones.
  - **float, double:** representan valores de tipo real y sus operaciones.
  - **boolean**: representa valores de tipo lógico y sus operaciones. Los valores de los tipos lógicos son dos, **true** y **false**
  - **char**: representa un caracter de un alfabeto determinado.
  - **String**: secuencia de caracteres.
  - **void**: es un tipo que no tiene ningún valor.

3. Tipos de datos

- Se pueden definir tipos nuevos mediante la cláusula **enum.** Un tipo enumerado puede tomar un conjunto determinado de valores, que se enumeran de forma explícita en su declaración.

```
public enum Color {
      ROJO, NARANJA, AMARILLO, VERDE, AZUL, VIOLETA
}
```

3. Tipos de datos

- Los tipos básicos son herencia de lenguajes de programación anteriores a Java. Por cada tipo básico se tiene disponible un tipo envoltura (*wrapper*):
  - **Byte** para byte
  - **Short** para short
  - **Integer** para int
  - **Long** para long
  - **Boolean** para boolean
  - **Float** para float
  - **Double** para double
  - **Character** para char
  - **Void** para void
- Los tipos envoltura añaden funcionalidad a los tipos primitivos. Esta funcionalidad añadida y los detalles que diferencian a uno y otros los iremos viendo más adelante.

ÍNDICE

4. Variables

- Las **variables** son elementos del lenguaje que permiten guardar y acceder a los datos que se manejan.
- Es necesario declararlas antes de usarlas en cualquier parte del código y, por convenio, se escriben en minúsculas.
- Mediante la declaración indicamos que la variable guardará un valor del tipo declarado.
- Mediante una asignación podemos dar un nuevo valor a la variable.

```
Integer edad = 38;
Double temperatura = 37.4;
boolean soltero = false;
char letra = 'T';
```

4. Constantes

- Las **CONSTANTES** son elementos del lenguaje que permiten guardar y referenciar datos que van a permanecer invariables durante la ejecución del código.
- La declaración de una constante comienza por la palabra reservada *final*.

```
final int DIAS_SEMANA = 7;
final Double PI = 3.1415926;
final String TITULO = "E.T.S. de Ingeniería Informática";
```

ÍNDICE

5. Expresiones

- Una **expresión** se forma con identificadores, valores constantes y operadores.
- Toda **expresión bien formada** tiene asociado un valor y un tipo.

```
edad >= 30                    //Esta expresión sería de tipo
boolean
(2 + peso) / 3                //Tipo Double
Color.ROJO                    //Tipo Color
```

5. Expresiones

- Mediante una **asignación** (representada por =) podemos dar nuevos valores a una variable.
- La asignación es un operador que da el valor de la expresión de la derecha a la variable que tiene a la izquierda.
- Una expresión formada con un operador de asignación tiene como tipo resultante el de la variable de la izquierda y como valor resultante el valor asignado a esa variable (la de la izquierda).

```
Double precio;
precio = 4.5 * peso + 34;
```

5. Operadores

- Los operadores más habituales en Java son los siguientes:
  - Operadores aritméticos: + (suma), - (resta), * (producto), / (división) y % (módulo)
  - Operadores lógicos: && (and), || (or) y ! (not)
  - Operadores relacionales: > (mayor que), < (menor que), >= (mayor o igual que), <= (menor o igual que), **== (¿igual que?)**, != (distinto de)
  - Operadores de asignación: =
    - y los abreviados: +=, -=, etc;
    - ++ (incremento) y -- (decremento)

ÍNDICE

6. Tipo String

- Como hemos visto antes, el tipo String representa una secuencia o cadena de caracteres.
- El tamaño de un String es **inmutable** y, por lo tanto, no cambia una vez creado el objeto. Se representa por el método *length()*.
- Cada carácter de la cadena ocupa una posición, comenzando por la posición 0 y terminando por la posición *length()-1*.
- Para acceder al carácter que ocupa una posición dada se utiliza el método *charAt(int i)*.

```
String nombre = "Amaia";
Integer lon = nombre.length() // El valor de lon será 5
Char inicial = nombre.charAt(0) // El valor de inicial será 'A'
Char ultima = nombre.charAt(lon - 1) // El valor de ultima será 'a'
```

6. Tipos para el manejo de fechas y horas

- Java dispone de un conjunto de tipos para trabajar con fechas, horas, instantes temporales y duraciones, incluidos en el paquete java.time.
  - LocalDate, tipo inmutable para representar fechas (sin zona horaria), es decir, solo con día, mes y año. Por ejemplo, 03-12-2015
  - LocalTime, tipo inmutable para representar horas sin fecha (ni zona horaria), solo con hora, minutos, segundos y nanosegundos (si son necesarios). Por ejemplo, 10:10:12.99.
  - Duration, tipo inmutable que representa una cantidad temporal que se mueve en el rango de nanosegundos, segundos, minutos, horas o días.
  - Period, tipo inmutable que representa una cantidad temporal que se mueve en el rango de años, meses o días.

```
LocalDate fecha2 = LocalDate.of(2014, 5, 23);
LocalTime hora1 = LocalTime.of(11, 00);
LocalDate fecha3 = LocalDate.now();
LocalDateTime hoy = LocalDateTime.now();
Integer dia = hoy.getDayOfMonth();
DayOfWeek diaSemana = hoy.getDayOfWeek();
```

6. Escritura de datos en pantalla

- Para escribir los datos por consola usaremos una llamada al método println. Así, por ejemplo, si escribimos la línea de código:

```
System.out.println("Hello World");
int n = 5;
System.out.println("El valor de la variable n es " + n + ".");
```

ÍNDICE

7. Sentencias de control selectivas

- Hay dos tipos de sentencias de control: las bifurcaciones y los bucles.
  - Las bifurcaciones permiten ejecutar un bloque de sentencias u otro, pero no ambos a la vez. En este tipo de sentencias de control se encuadran las sentencias if y switch.

```
Double salario = 3200.26;
Double impuesto = null;
if (salario >= 5000.0) {
impuesto = 20.0;
} else if (salario < 5000.0 && salario >= 2500.0) {
impuesto = 15.0;
} else if (salario < 2500.0 && salario >= 1500.0) {
impuesto = 10.0;
} else {
impuesto = 5.0;
}
```

7. Sentencia Switch

- Normalmente *switch* se utiliza cuando se requiere comparar una variable de un tipo discreto con una serie de valores diferentes.
- En la sentencia *switch*, se indican los posibles valores que puede tomar la variable y las sentencias que se tienen que ejecutar si la variable coincide con alguno de dichos valores.
- Es una sentencia muy indicada para comparar una variable de un tipo enumerado con cada uno de sus posibles valores.

```
int numero = 5;
switch (numero) {
        case 1:
        s = "Uno";
        break;
        case 2:
        s = "Dos";
        break;
        case 3:
        s = "Tres";
        break;
        default:
        s = "Otro";
}
```

ÍNDICE

8. Agregados - Listas

- Las listas representan colecciones de elementos de un mismo tipo en los que importa cuál es el primero, el segundo, etc.
- Cada elemento está referenciado mediante un índice; el índice del primer elemento es el 0.
- Las listas pueden contener elementos duplicados.

```
List<Double> temperaturas = new LinkedList<>();
temperaturas.add(27.5);
temperaturas.add(22.0);
temperaturas.add(25.3);

Double t1 = temperaturas.get(0);
Integer numeroElementos = temperaturas.size();
```

8. Agregados - Conjuntos

- El tipo *Set* de Java se corresponde con el concepto matemático de conjunto:
  - Agregado de elementos en el que no hay orden (no se puede decir cuál es el primero, el segundo, el tercero, etc.)
  - No puede haber elementos repetidos.

```
Set<Character> letras = new HashSet<>();
letras.add('A');
letras.add('V');
letras.add('E');
letras.contains('V’);

//Tipo de conjunto que sí establece un orden
SortedSet<Character> letrasOrdenadas = new TreeSet<>();
```

8. Agregados - Map

- El tipo de dato *Map* permite modelar el concepto de aplicación
  - Una relación entre los elementos de dos conjuntos de modo que a cada elemento del conjunto inicial le corresponde uno y solo un elemento del conjunto final.
  - Los elementos del conjunto inicial se denominan claves (*keys*) y los del conjunto final valores (*values*).
- Para crear un *Map* hay que indicar el tipo de las claves y el tipo de los valores.

```
Map<String, Double> temperaturasCiudad = new HashMap<>();
temperaturasCiudad.put("Córdoba", 19.1);
Double t = temperaturasCiudad.get("Córdoba");
```

ÍNDICE

9. Sentencias de control iterativas - While

- Los bucles son sentencias de control que ejecutan un bloque de sentencias un número determinado de veces.
- La sentencia while ejecuta el bloque de sentencias mientras la condición evaluada sea cierta.

```
//Suma los elementos entre 1 y n
Integer suma = 0;
int i = 1;
int n = 10;
while (i <= n) {
      suma = suma + i;
      i++;
}
```

9. Sentencias de control iterativas - for

- La sentencia *for* tiene la siguiente sintaxis, donde *inicialización* y *actualización* son sentencias y *condición* es una expresión lógica.

```
for (inicialización; condición; actualización) {
      sentencia-1;
      sentencia-2;
      ...
      sentencia-n;
}
```

```
Integer suma = 0;
for (int i = 0; i <= 10; i++) {
       suma = suma + i;
}
```

9. Sentencias de control iterativas - for

- Existe una variante del *for* que se utiliza para recorrer agregados, como listas o conjuntos.
- Supongamos por ejemplo que queremos calcular el valor medio de las temperaturas de la lista temperaturas. Lo podríamos hacer mediante un *for* clásico:

```
//Forma tradicional
Double suma = 0.0;
for (int i = 0; i < temperaturas.size(); i++) {
        suma = suma + temperaturas.get(i);
}
Double temperaturaMedia = suma / temperaturas.size();
```

```
//Forma extendida
Double suma = 0.0;
for (Double t: temperaturas) {
        suma = suma + t;
}
Double temperaturaMedia = suma / temperaturas.size();
```

9. Sentencias de control iterativas - for

- La sentencia **break** dentro de un bloque de sentencias que forma parte de una estructura iterativa hace que el flujo de programa salte fuera del bloque, dando por finalizada la ejecución de la sentencia iterativa.

```
Boolean temperaturaNegativa = false;
for (Double t: temperaturas) {
      if (t < 0) {
            temperaturaNegativa = true;
            break;
      }
}
```

ÍNDICE

10. Streams

- A partir de la versión 8, Java introduce una nueva y potente forma de realizar operaciones con agregados de datos. Para ello nos proporciona varios elementos que podemos combinar para realizar cualquier tratamiento que necesitemos sobre un agregado de datos.
- El elemento fundamental es el **Stream**. Un *stream* es un agregado de elementos (las canciones de un artista, las películas estrenadas el año pasado o las temperaturas diarias en una ciudad).
- Los *streams* se obtienen habitualmente a partir de una lista o conjunto, y podemos realizar sobre ellos diversos tratamientos.

10. Streams

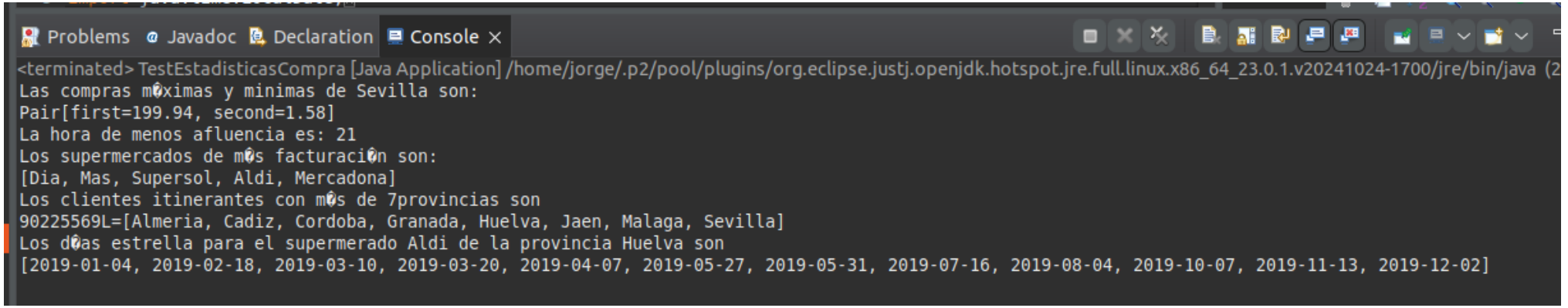

```
long bajoCero = 0;
for (Double t: temperaturas) {
        if (t < 0) {
                bajoCero++;
        }
}
System.out.println("Hay " + bajoCero + " temperaturas bajo cero.");
```

```
long bajoCero = temperaturas.stream()

.filter(x -> x < 0)

.count();
System.out.println("Hay " + bajoCero + " temperaturas bajo cero.");
```

Tarea

- Clonar repositorio Compras

- Ejecutar su programa principal y sacar por pantalla resultados



- Lectura comprensiva tema 2.