

# Sesión 07: Invocación de métodos

## Programación 2

---

Ángel Herranz

Febrero 2019

Universidad Politécnica de Madrid

# En capítulos anteriores

- ...
  - Objetos, referencias y variables (y primitivos)
  - Clases: plantilla para crear objetos
- Encapsular  
datos y comportamiento
- Terminología y ocultación

# En capítulos anteriores

- ...
- **Objetos**, **referencias** y variables (y **primitivos**)
- **Clases**: plantilla para crear objetos

**Encapsular**

datos y comportamiento

- Terminología y **ocultación**
- Modelización: racionales, puntos, naipes, ...



# Análisis sintáctico

Colorless green ideas sleep furiously

Noam Chomsky



# Análisis sintáctico

Colorless green ideas sleep furiously

Noam Chomsky



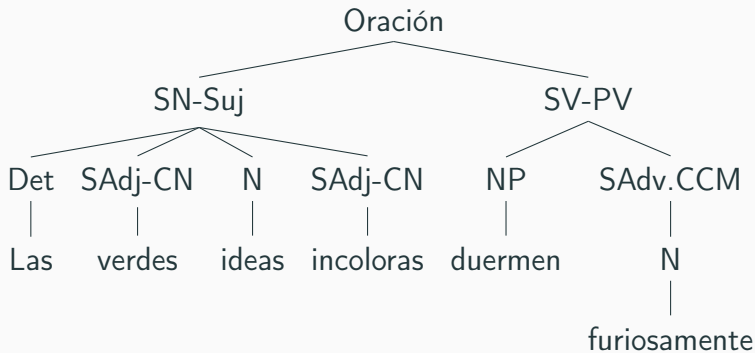
# Análisis sintáctico

Las verdes ideas incoloras duermen furiosamente

Noam Chomsky



# Análisis sintáctico

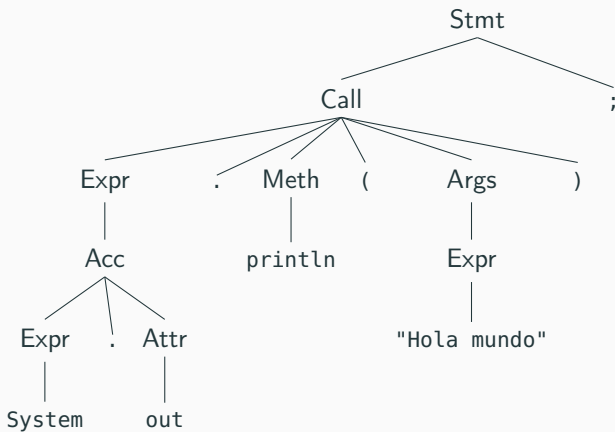


# Lo mismo en Java

```
System.out.println("Hola mundo");
```



# Lo mismo en Java



# ¿Algo más complejo?

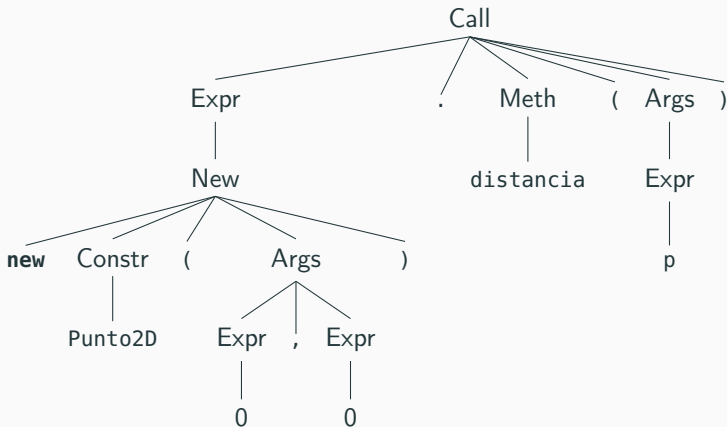
```
System.out.println(new Punto2D(0,0).distancia(p));
```

# ¿Algo más complejo?

```
new Punto2D(0,0).distancia(p)
```

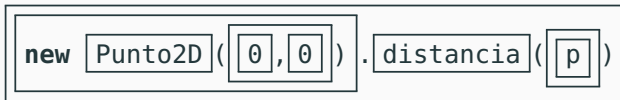
# ¿Algo más complejo?

`new Punto2D(0,0).distancia(p)`



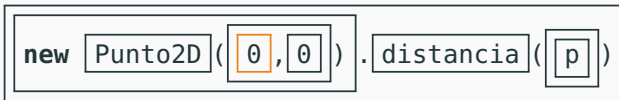
# ¿Algo más complejo?

**new** Punto2D(0,0).distancia(p)



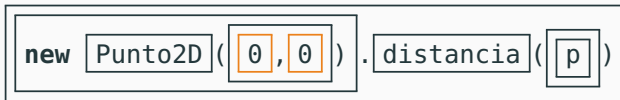
# ¿Algo más complejo?

**new** Punto2D(0,0).distancia(p)



# ¿Algo más complejo?

`new Punto2D(0,0).distancia(p)`



# ¿Algo más complejo?

`new Punto2D(0,0).distancia(p)`





# ¿Algo más complejo?

`new Punto2D(0,0).distancia(p)`



# ¿Algo más complejo?

`new Punto2D(0,0).distancia(p)`



# ¿Algo más complejo?

`new Punto2D(0,0).distancia(p)`



# ¿Algo más complejo?

`new Punto2D(0,0).distancia(p)`



# ¿Algo más complejo?

**new** Punto2D(0,0).distancia(p)



# ¿Algo más complejo?

**new** Punto2D(0,0).distancia(p)



# Pilas, pilas y pilas

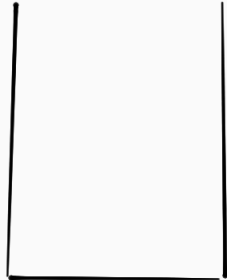


# Pilas, pilas y pilas

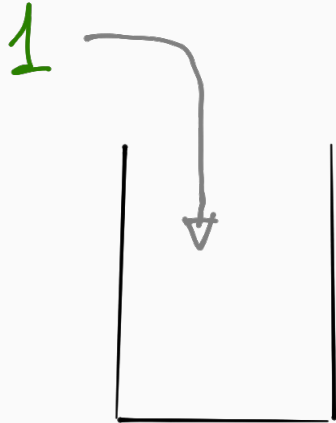




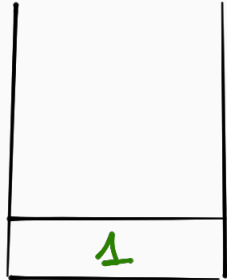
# Pilas, pilas y pilas (*Stack*)



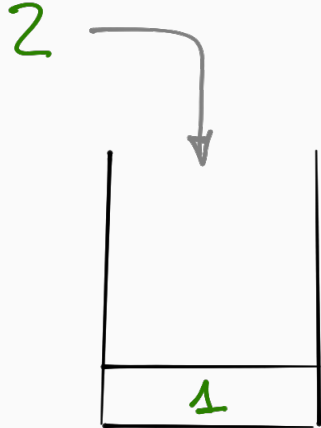
# Pilas, pilas y pilas (*Stack*)



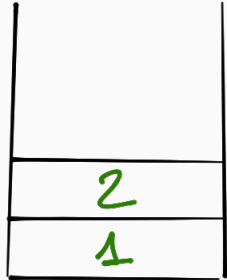
# Pilas, pilas y pilas (*Stack*)



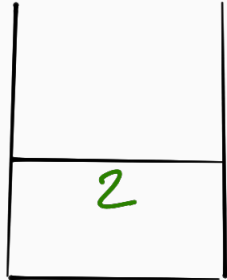
# Pilas, pilas y pilas (*Stack*)



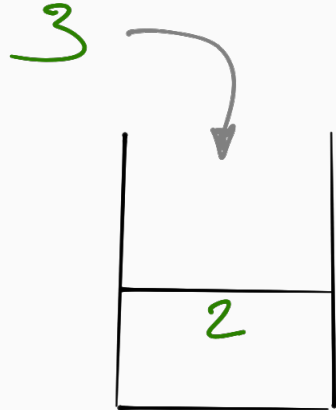
# Pilas, pilas y pilas (*Stack*)



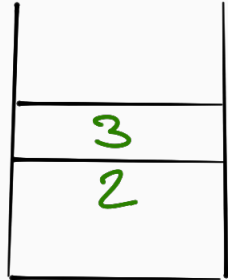
# Pilas, pilas y pilas (*Stack*)



# Pilas, pilas y pilas (*Stack*)

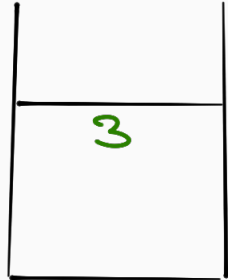


# Pilas, pilas y pilas (*Stack*)

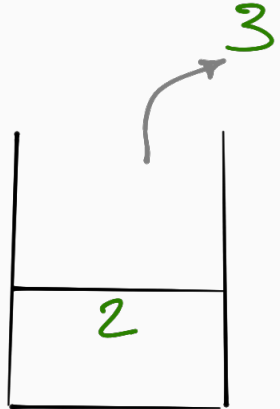




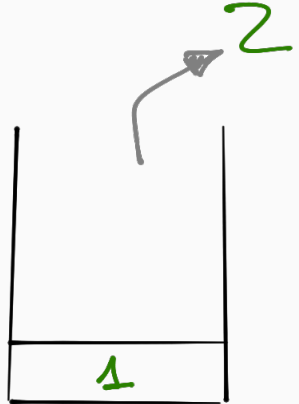
# Pilas, pilas y pilas (*Stack*)



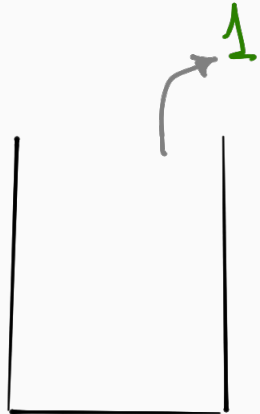
# Pilas, pilas y pilas (*Stack*)



# Pilas, pilas y pilas (*Stack*)



# Pilas, pilas y pilas (*Stack*)



# ¿Qué ocurre al invocar un método?

```
public static void  
    main(String[] args) {  
  
    Punto o =  
        new Punto2D();  
  
    Punto a =  
        new Punto2D(1,1);  
  
    System.out.println(  
        a.distancia(o)  
    );  
}
```

Q: ¿Cuál es el primer método que se invoca aquí?

# ¿Qué ocurre al invocar un método?

```
public static void  
    main(String[] args) {  
  
    Punto o =  
        new Punto2D();  
  
    Punto a =  
        new Punto2D(1,1);  
  
    System.out.println(  
        a.distancia(o)  
    );  
}
```

Herranz

Q: ¿Cuál es el primer método que se invoca aquí?

A: **¡main!**

Q: ¿Y después?

# ¿Qué ocurre al invocar un método?

```
public static void  
    main(String[] args) {  
  
    Punto o =  
        new Punto2D();  
  
    Punto a =  
        new Punto2D(1,1);  
  
    System.out.println(  
        a.distancia(o)  
    );  
}
```

Herranz

Q: ¿Cuál es el primer método que se invoca aquí?

A: **jmain!**

Q: ¿Y después?

A: **distancia**

Q: ¿Y después?

# ¿Qué ocurre al invocar un método?

```
public static void  
    main(String[] args) {  
  
    Punto o =  
        new Punto2D();  
  
    Punto a =  
        new Punto2D(1,1);  
  
    System.out.println(  
        a.distancia(o)  
    );  
}
```

Herranz

Q: ¿Cuál es el primer método que se invoca aquí?

A: `main!`

Q: ¿Y después?

A: `distancia`

Q: ¿Y después?

A: `println`



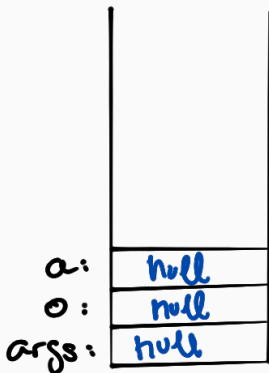
# Pila (*Stack*) de ejecución

Empieza la ejecución de main



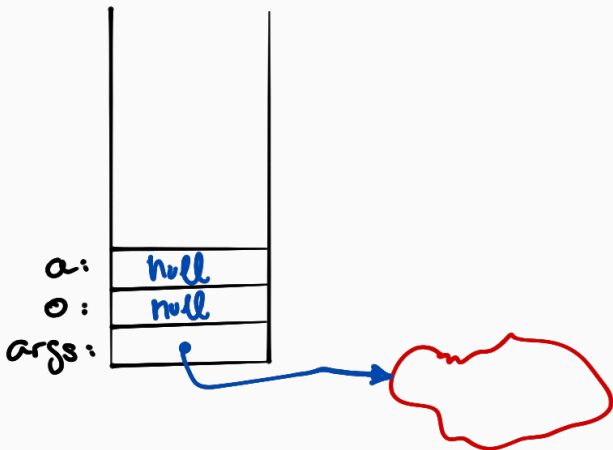
# Pila (*Stack*) de ejecución

Espacio para argumentos y variables locales



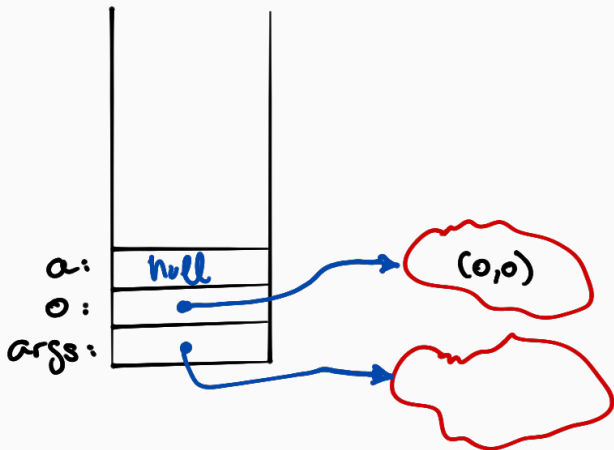
# Pila (*Stack*) de ejecución

Se cargan los argumentos



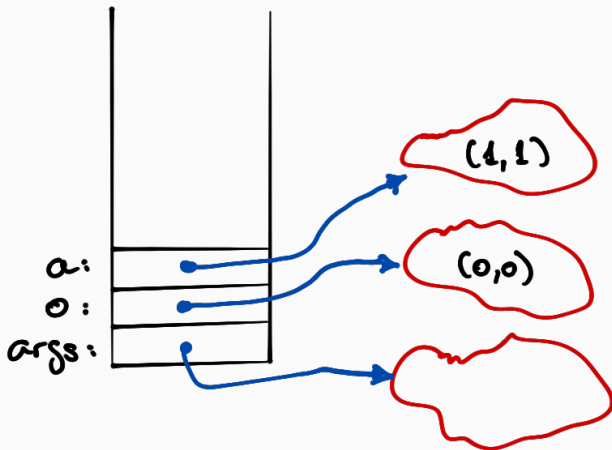
# Pila (*Stack*) de ejecución

Punto2D o = **new** Punto2D()

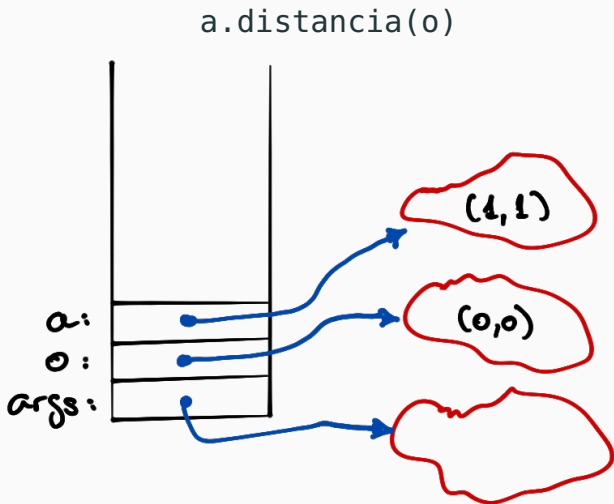


# Pila (*Stack*) de ejecución

```
Punto2D a = new Punto2D(1,1)
```

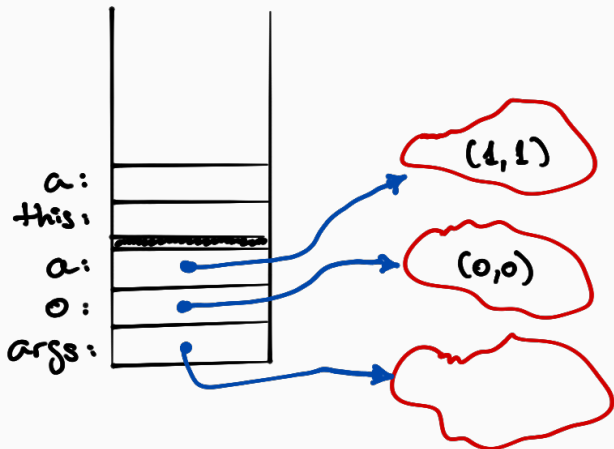


# Pila (*Stack*) de ejecución



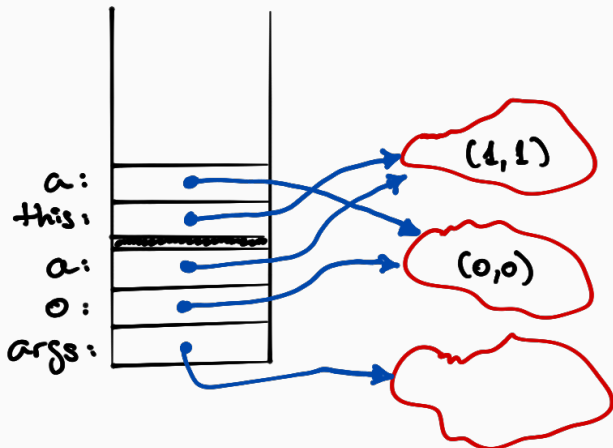
# Pila (*Stack*) de ejecución

`public double distancia(Punto2D a)` 🔔



# Pila (*Stack*) de ejecución

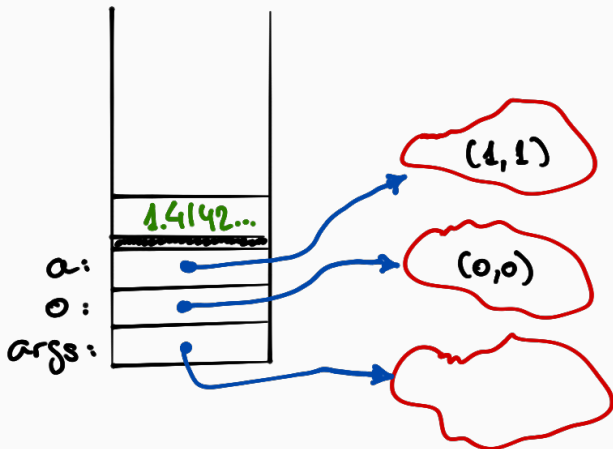
`a.distancia(o)`





# Pila (*Stack*) de ejecución

En distancia: **return** Math.sqrt(...);



# Paso de parámetros en Java

- El paso de parámetros en Java se hace *por valor*
- ¿Qué significa *por valor*<sup>1</sup>?

El valor se copia en el argumento

---

<sup>1</sup>También se usa el término *por copia*



# ¿Qué va a pasar?

```
class Intercambiar {  
    public static void  
        main(String args[]) {  
        int a, b;  
        a = 27;  
        b = 42;  
        intercambiar(a,b);  
        System.out.println(  
            a + " - " + b  
        );  
    }  
}
```

a) 27 - 42

b) 42 - 27

# ¿Qué va a pasar?

```
class Intercambiar {  
    public static void  
        main(String args[]) {  
        int a, b;  
        a = 27;  
        b = 42;  
        intercambiar(a,b);  
        System.out.println(  
            a + " - " + b  
        );  
    }  
}
```

a) 27 - 42

b) 42 - 27

 ¡Dibuja!


# ¿Qué va a pasar?

```
class Intercambiar {  
    public static void  
        main(String args[]) {  
        int a, b;  
        a = 27;  
        b = 42;  
        intercambiar(a,b);  
        System.out.println(  
            a + " - " + b  
        );  
    }  
}
```

a) 27 - 42

b) 42 - 27

 ¡Dibuja!

 Programa y ejecuta

# ¿Qué es **static**?

TODO

## Tema 2: Colecciones acotadas de objetos

---

Programa que lea de la entrada estándar e imprima en la salida estándar una *playlist*



## Mi *playlist*: entrada

- Lo primero que se lee es un entero que indica cuantas canciones hay
- Por cada canción tres líneas:
  1. Título
  2. Artísta
  3. Valoración (de 0 a 5)

## Mi *playlist*: ejemplo de entrada

3

Despacito

Luis Fonsi

2

The logical song

Supertramp

5

Wish you where here

Pink Floyd

4

## Mi *playlist*: salida

- Cada canción en una línea con este formato:

*Título:Artista:Valoración*

## Mi *playlist*: salida esperada

Despacito:Luis Fonsi:2

The logical song:Supertramp:5

Wish you where here:Pink Floyd:4

## ❓ Para leer de la entrada estándar

- Variable con un `java.util.Scanner`:

```
java.util.Scanner stdin =  
    new java.util.Scanner(System.in);
```

- Leer una línea:

```
String titulo;  
titulo = stdin.nextLine();
```

## ❓ Para leer de la entrada estándar

- Variable con un `java.util.Scanner`:

```
java.util.Scanner stdin =  
    new java.util.Scanner(System.in);
```

- Leer una línea:

```
String titulo;  
titulo = stdin.nextLine();
```

- `java.util.Scanner`

## ? Para probar más rápidamente

- Pon toda la entrada en un fichero, por ejemplo `canciones.txt` siguiendo el formato indicado
- Haz que tu programa lea de ese fichero en vez de usar la entrada estándar, para ello utiliza la siguiente línea de comandos:

```
C:\ Sesion07> java ImprimirPlaylist < canciones.txt
Despacito:Luis Fonsi:2
The logical song: Supertramp:5
Wish you where here:Pink Floyd:4
C:\ Sesion07> _
```