Sesión 03: Clases

Programación 2

1. Introducción a la programación orientada a objetos (POO)
2. Clases y Objetos

Ángel Herranz

2021-2022

Universidad Politécnica de Madrid

En capítulos anteriores. . .

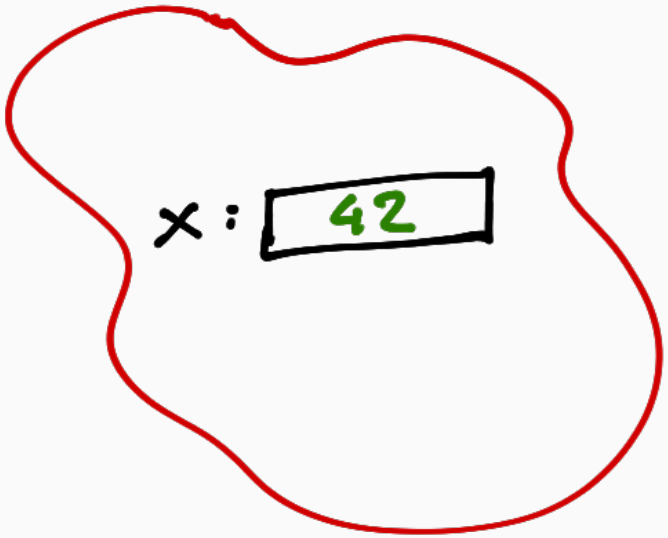
- Sobre los IDEs
- Clases y objetos (intro)
- Objetos, referencias y variables (y primitivos)

Objetos, referencias y variables

```
public class Sesion02 {
  public static void main(String args[]) {
    A a;
    a = new A();
  }
}
```

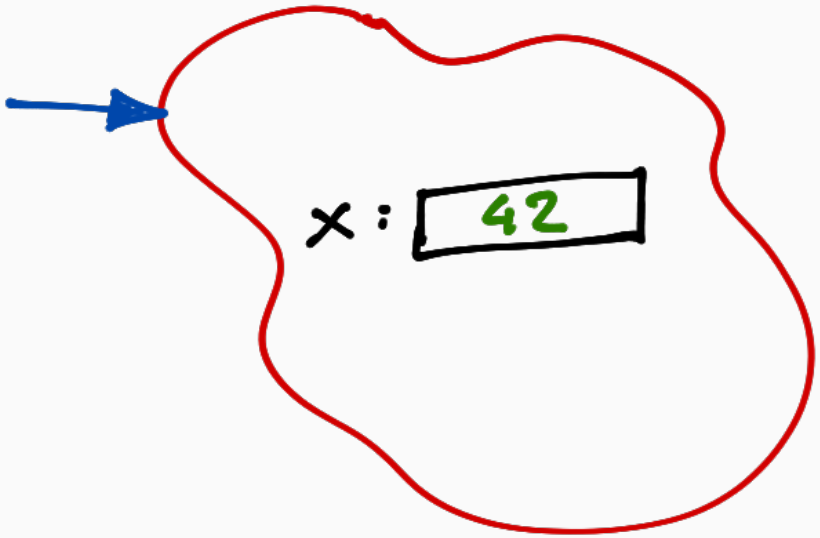
Objetos, referencias y variables

```
public class Sesion02 {
  public static void main(String args[]) {
    A a;
    a = new A();
  }
}
```



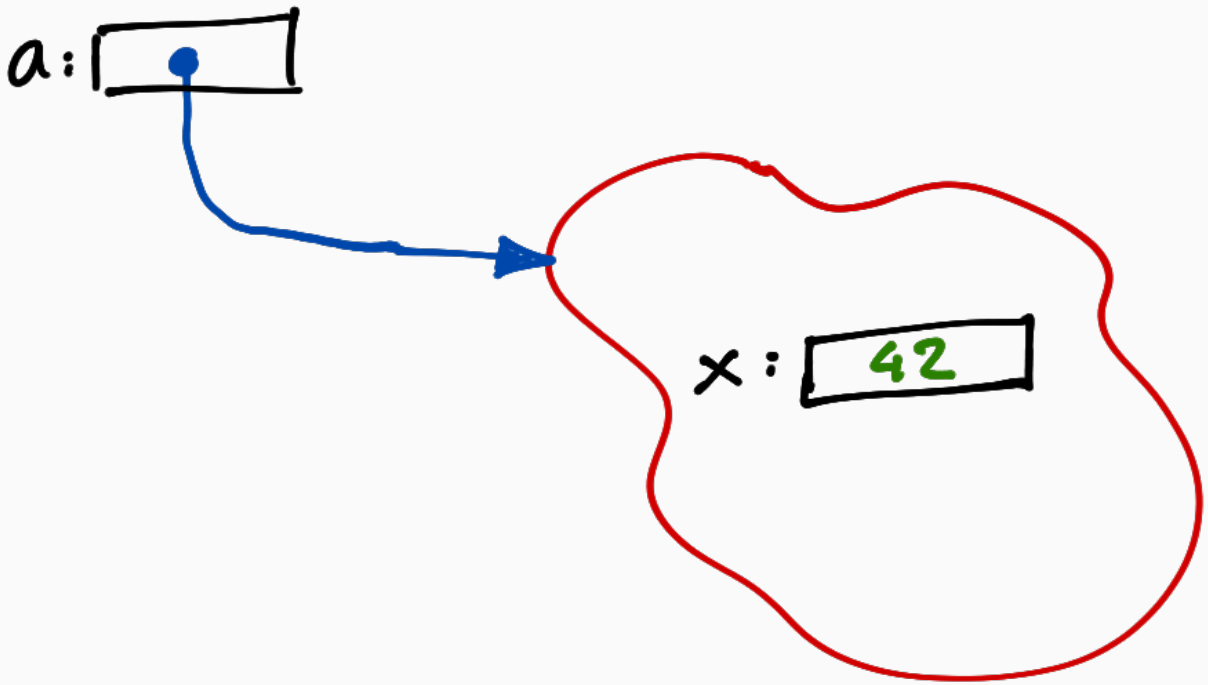
Objetos, referencias y variables

```
public class Sesion02 {
  public static void main(String args[]) {
    A a;
    a = new A();
  }
}
```



Objetos, referencias y variables

```
public class Sesion02 {
  public static void main(String args[]) {
    A a;
    a = new A();
  }
}
```



Spotify

There are only two hard things in Computer Science: cache invalidation and naming things.

Phil Karlton

explicit, nombre, título, artísta, interprete, autor, compositor, duración, álbum, audio, sonidos, notas, imagen, valoración[1]

[1] Esta lista de nombres representa una votación en clase

Punto de partida

Show me the code

 5’

Punto de partida

```
class Cancion {
  String titulo;
  String interprete;
  String compositor;
  String album;
  int duracion;
  String audio;
  String imagen;
  int valoracion;
  boolean explicit;
}
```

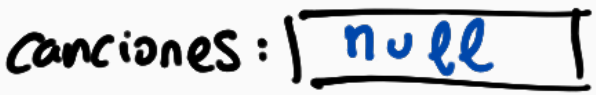
Lista de reproducción

Ya tenemos una forma de modelizar canciones

¿Cómo modelizar una lista de reproducción?

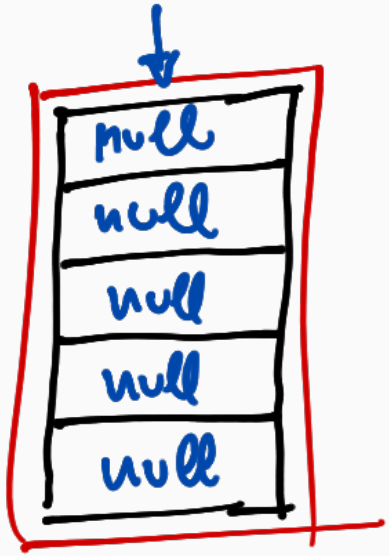
Lista de reproducción

```
class Sesion03 {
  public static void main(String args[]) {
    Cancion[] canciones = new Cancion[5];
    canciones[0] = new Cancion();
  }
}
```



Lista de reproducción

```
class Sesion03 {
  public static void main(String args[]) {
    Cancion[] canciones = new Cancion[5];
    canciones[0] = new Cancion();
  }
}
```

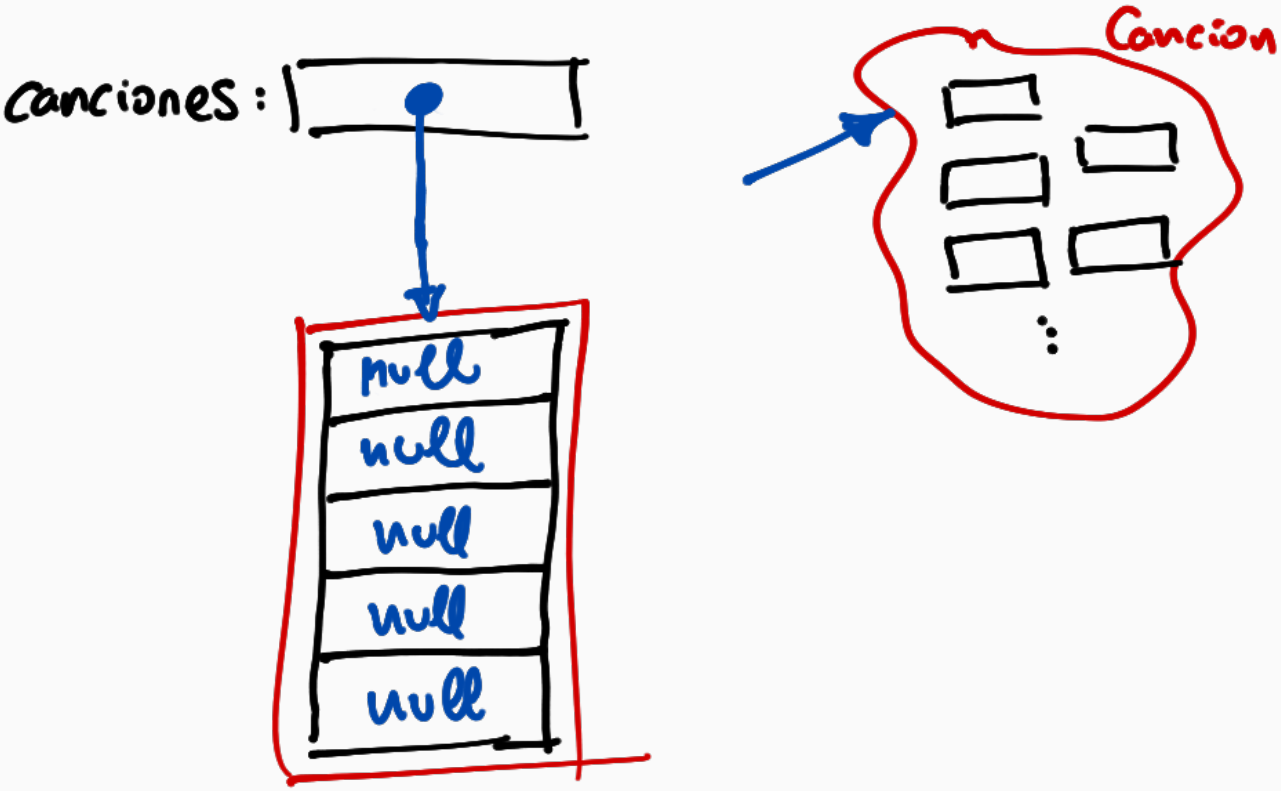


Lista de reproducción

```
class Sesion03 {
  public static void main(String args[]) {
    Cancion[] canciones = new Cancion[5];
    canciones[0] = new Cancion();
  }
}
```

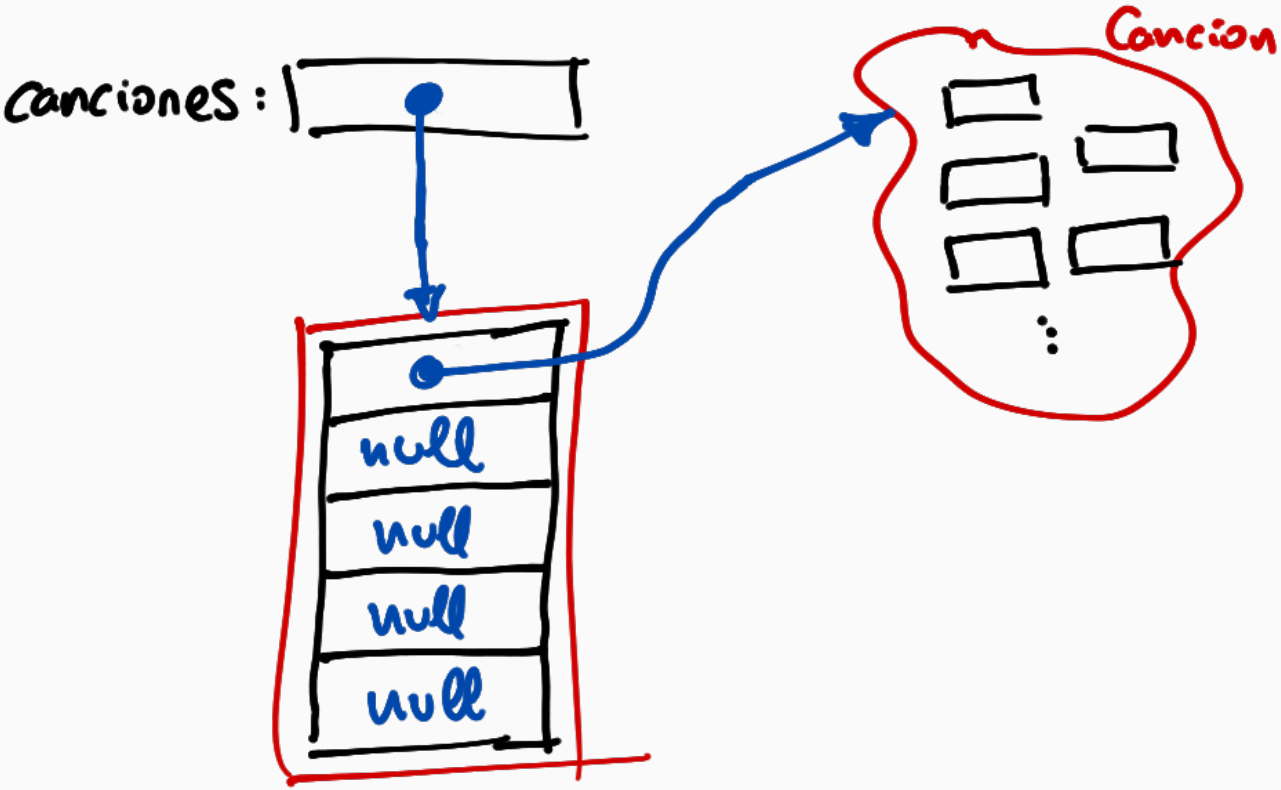
Lista de reproducción

```
class Sesion03 {
  public static void main(String args[]) {
    Cancion[] canciones = new Cancion[5];
    canciones[0] = new Cancion();
  }
}
```



Lista de reproducción

```
class Sesion03 {
  public static void main(String args[]) {
    Cancion[] canciones = new Cancion[5];
    canciones[0] = new Cancion();
  }
}
```



¡Los *arrays* de Java son objetos! `:o`

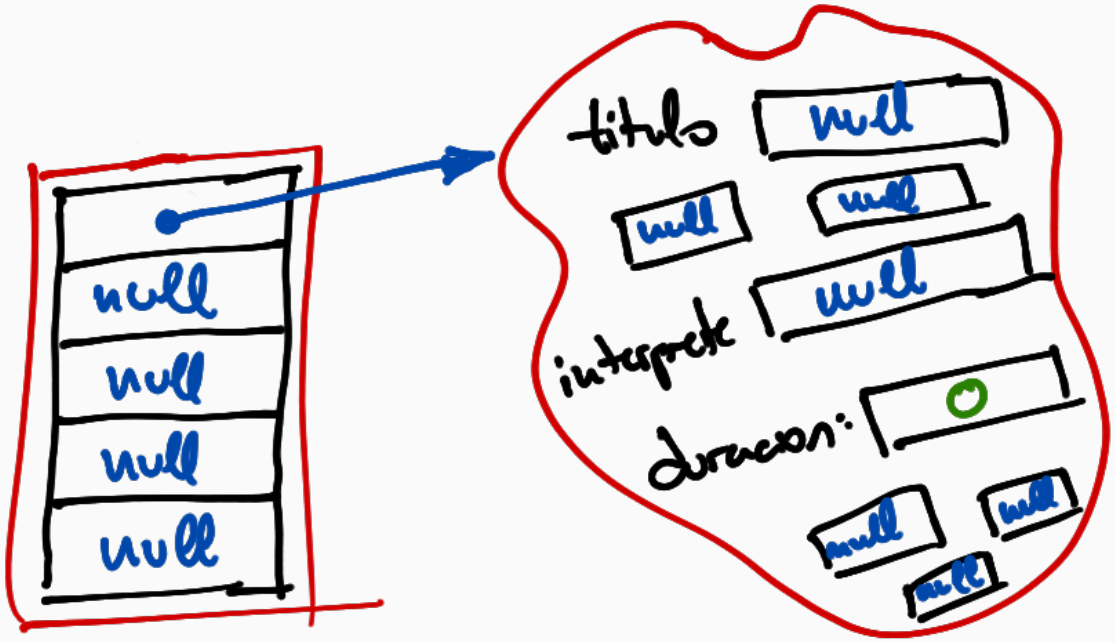
- `Cancion[]` es una *clase* (o **tipo**)
- Si *A* es un tipo entonces *A*`[]` es un tipo
- Es el tipo de los arrays de *A*s
- Declaramos variables poniendo el tipo delante delante: `Cancion[] canciones;`
- Las **variables array son referencias**
- El objeto array **no existe hasta hacer el `new`**
- **Sus elementos son variables** inicializadas a

  `null`

  `true`, `0`, `0.0`, etc.

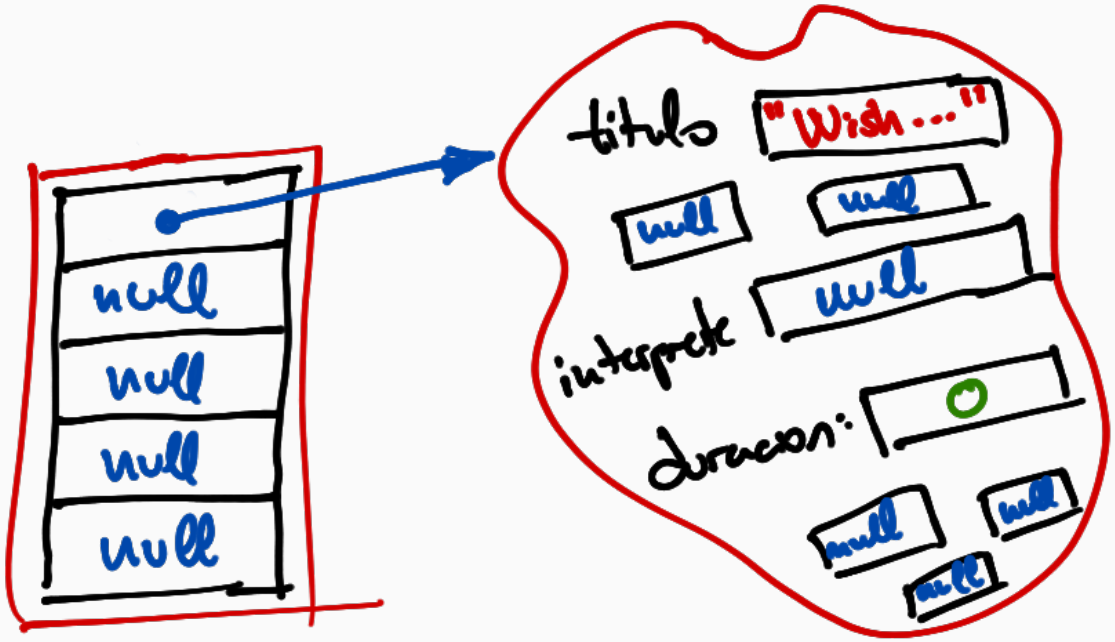
Primera canción

```
canciones[0] = new Cancion();
canciones[0].titulo = "Wish you were here";
canciones[0].interprete = "Pink Floyd";
canciones[0].duracion = 334;
```



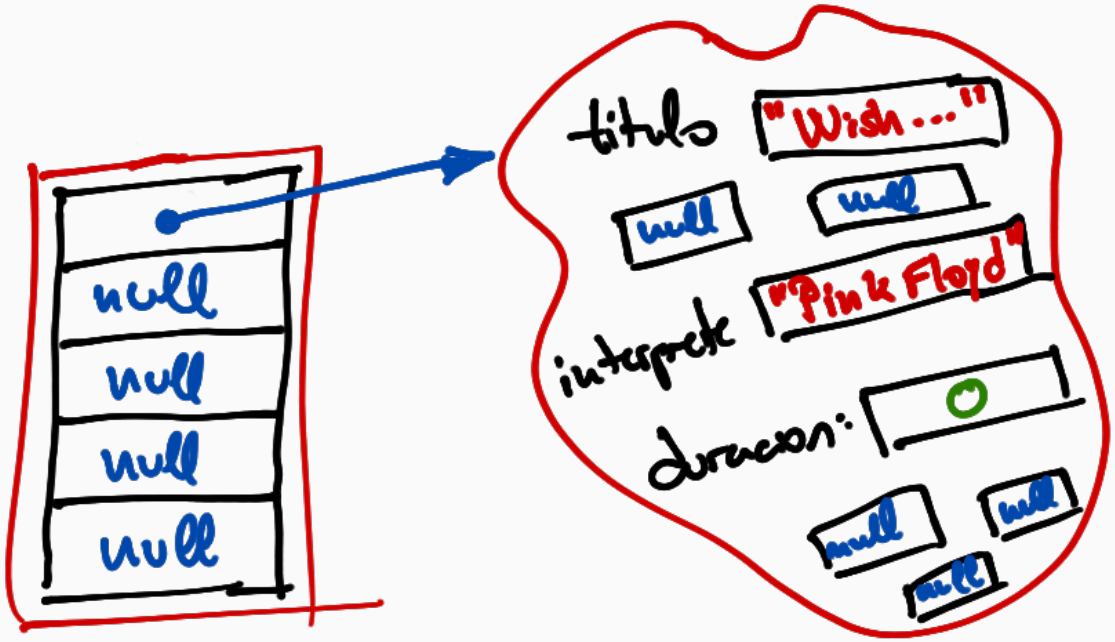
Primera canción

```
canciones[0] = new Cancion();
canciones[0].titulo = "Wish you were here";
canciones[0].interprete = "Pink Floyd";
canciones[0].duracion = 334;
```



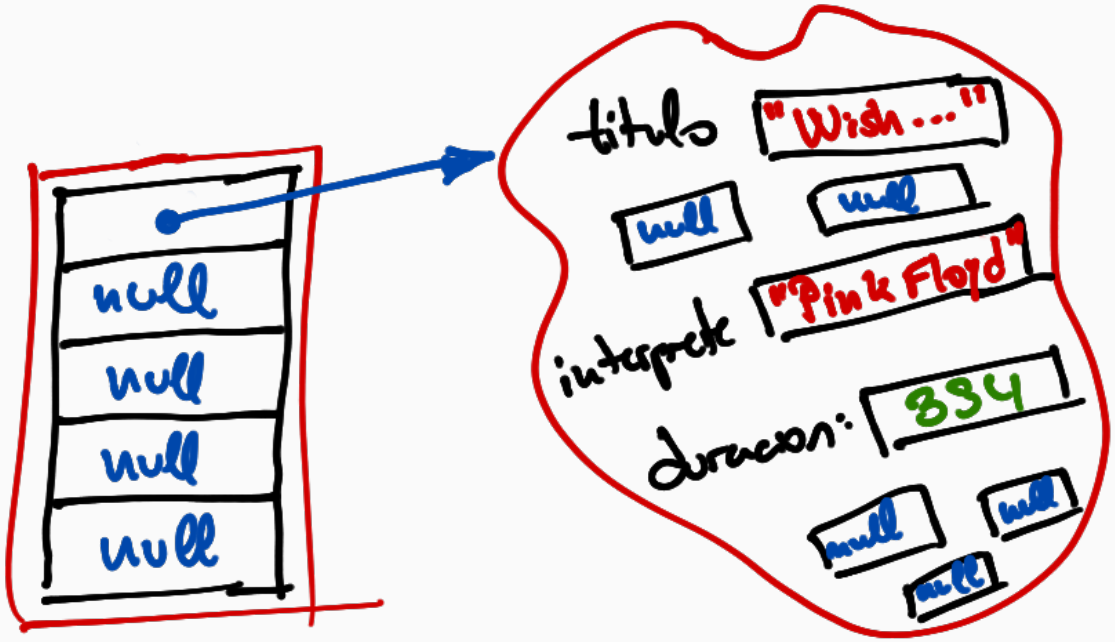
Primera canción

```
canciones[0] = new Cancion();
canciones[0].titulo = "Wish you were here";
canciones[0].interprete = "Pink Floyd";
canciones[0].duracion = 334;
```



Primera canción

```
canciones[0] = new Cancion();
canciones[0].titulo = "Wish you were here";
canciones[0].interprete = "Pink Floyd";
canciones[0].duracion = 334;
```



Cread un par de canciones i

Cread un par de canciones i

```
canciones[0] = new Cancion();
canciones[0].titulo = "Wish you were here";
canciones[0].interprete = "Pink Floyd";
canciones[0].compositor = "David Gilmour, Roger Waters";
canciones[0].album = "Wish you were here";
canciones[0].duracion = 334;
canciones[0].audio = "RocNidUrc6.mp3";
canciones[0].imagen = "RocNidUrc6.jpg";
canciones[0].valoracion = 5;
```

Cread un par de canciones ii

```
canciones[1] = new Cancion();
canciones[1].titulo = "The logical song";
canciones[1].interprete = "Supertramp";
canciones[1].compositor = "Rick Davies, Roger Hodgson";
canciones[1].album = "Breakfast in America";
canciones[1].duracion = 255;
canciones[1].audio = "wrewoig0.mp3";
canciones[1].imagen = "wrewoig0.jpg";
canciones[1].valoracion = 5;
```

Imprimir las canciones *bonitas*

```
System.out.println(
  "| " + canciones[0].titulo
  + " | " + canciones[0].interprete
  + " | " + canciones[0].duracion / 60 + "mins |"
);
System.out.println(
  "| " + canciones[1].titulo
  + " | " + canciones[1].interprete
  + " | " + canciones[1].duracion / 60 + "mins |"
);
```

```
| Wish you were here | Pink Floyd | 5mins |
| The logical song | Supertramp | 4mins |
```

Todos suspensos

Todos suspensos

Así no se hacen las cosas,

pero ya lo sabíamos...

Everything should be built top-down, except the first time.

Epigrams on Programming (Alan J. Perlis)

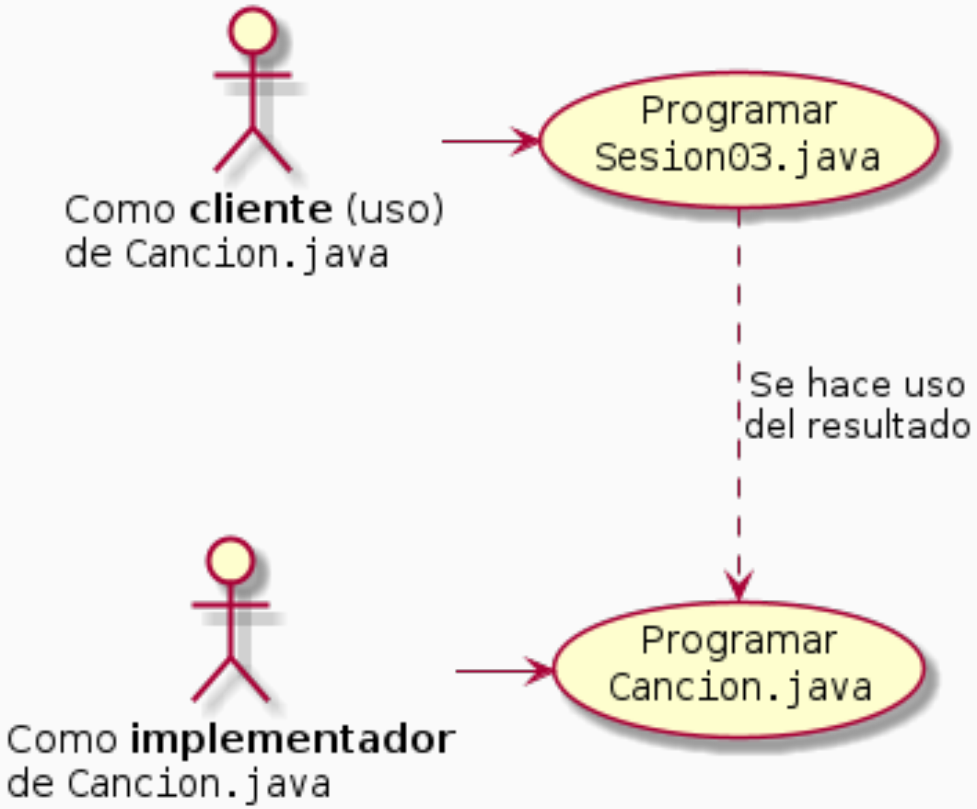
Dos roles para ver un código

Cuando lo implementas

y

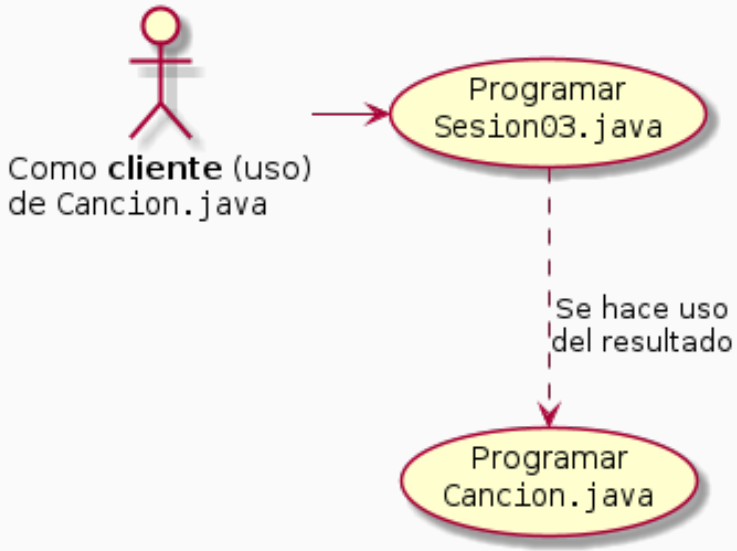
Cuando lo usas

Dos roles para ver un código



Cuando usas un código

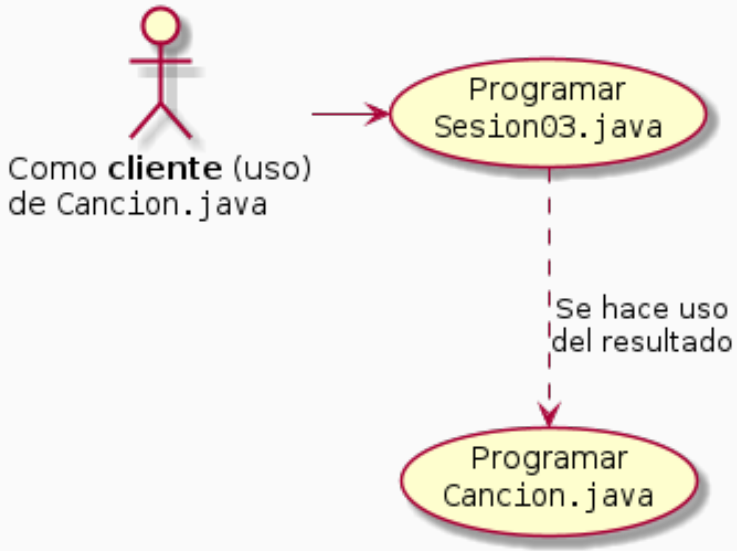
- No quieres saber cómo está hecho
- Sólo quieres saber qué hace
- No quieres repetir trabajo



Cuando usas un código

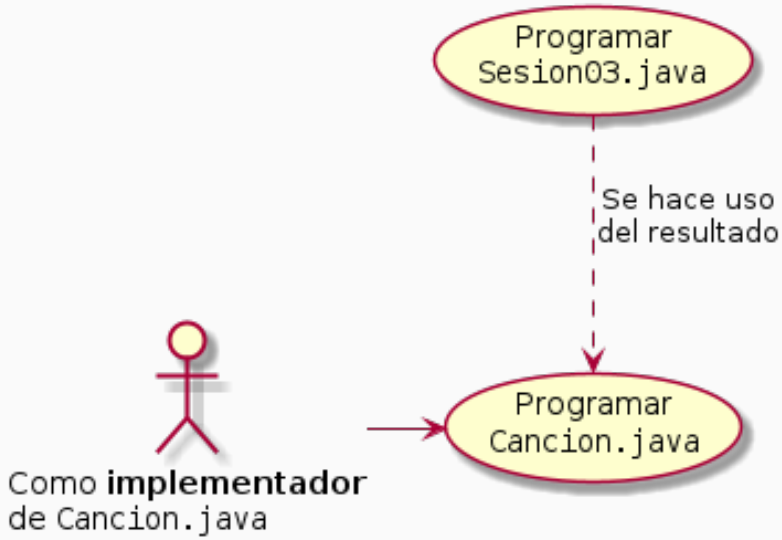
- No quieres saber cómo está hecho
- Sólo quieres saber qué hace
- No quieres repetir trabajo

Cuando programas `Sesion03.java`



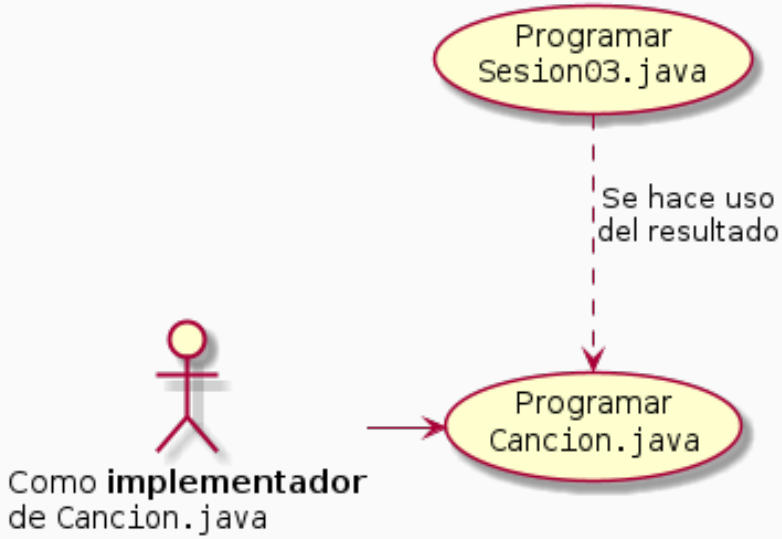
Cuando implementas un código

- Estás al servicio de quienes lo usan
- Que no necesiten mirar el cómo
- Explicar claramente el qué hace



Cuando implementas un código

- Estás al servicio de quienes lo usan
- Que no necesiten mirar el cómo
- Explicar claramente el qué hace



Cuando programas

`Cancion.java`

Objetivo: encapsular el comportamiento

- Escribimos una forma de crear canciones
- Escribimos una forma de sacar los minutos
- Escribimos una forma de formatear las canciones

Objetivo: encapsular el comportamiento

- Escribimos una forma de crear canciones
- Escribimos una forma de sacar los minutos
- Escribimos una forma de formatear las canciones

Encapsular

ese comportamiento

en la clase `Cancion`

(junto con los datos)

¿Imaginais?

```
canciones[0] =
  new Cancion("Wish you were here",
              "Pink Floyd",
              354);
```

¿Imaginais?

```
System.out.println(cancion[0].minutos());
```

¿Imaginais?

```
System.out.println( cancion[0] .minutos () );
```

Siga la flecha: `_._`

hacia un comportamiento: `minutos()`

¿Imaginais?

```
System.out.println( cancion[0] .minutos () );
```

5mins

Siga la flecha: `_._`

hacia un comportamiento: *`minutos()`*

¿Imaginais incluso?

```
System.out.println(cancion[0]);
```

```
| Wish you were here | Pink Floyd | 5mins |
```

¿Imaginais?

Yes, we can

¿Imaginais?

```
canciones[0] =
  new Cancion("Wish you were here",
              "Pink Floyd",
              354);
```

Comportamiento para crear

```
1  class Cancion {
2    String titulo;
3    String interprete;
4    int duration;
5    ...;
6    Cancion (String titulo,
7             String interprete,
8             int duration) {
9      titulo = titulo;
10     interprete = interprete;
11     duracion = duración;
12   }
13 }
```

- El comportamiento se añade al final de los datos (6-12)

Comportamiento para crear

```
1  class Cancion {
2    String titulo;
3    String interprete;
4    int duration;
5    ...;
6    Cancion (String titulo,
7             String interprete,
8             int duration) {
9      titulo = titulo;
10     interprete = interprete;
11     duracion = duración;
12   }
13 }
```

- El comportamiento se añade al final de los datos (6-12)
- Mismo nombre de la clase

+

Comportamiento para crear

```
1  class Cancion {
2    String titulo;
3    String interprete;
4    int duration;
5    ...;
6    Cancion (String titulo,
7             String interprete,
8             int duration) {
9      titulo = titulo;
10     interprete = interprete;
11     duracion = duración;
12   }
13 }
```

- El comportamiento se añade al final de los datos (6-12)
- Mismo nombre de la clase + parámetros *formales*

Comportamiento para crear

```
1  class Cancion {
2    String titulo;
3    String interprete;
4    int duration;
5    ...;
6    Cancion (String titulo,
7             String interprete,
8             int duration) {
9      titulo = titulo;
10     interprete = interprete;
11     duracion = duración;
12   }
13 }
```

- El comportamiento se añade al final de los datos (6-12)
- Mismo nombre de la clase + parámetros *formales*
- Toman el valor de la llamada **new**

Comportamiento para crear

```
1  class Cancion {
2    String titulo;
3    String interprete;
4    int duration;
5    ...;
6    Cancion (String titulo,
7             String interprete,
8             int duration) {
9      titulo = titulo;
10     interprete = interprete;
11     duracion = duración;
12   }
13 }
```

- El comportamiento se añade al final de los datos (6-12)
- Mismo nombre de la clase + parámetros *formales*
- Toman el valor de la llamada **new**
- Se copian a los datos

Comportamiento para crear

```
1  class Cancion {
2    String titulo;
3    String interprete;
4    int duration;
5    ...;
6    Cancion (String titulo,
7             String interprete,
8             int duration) {
9      titulo = titulo;
10     interprete = interprete;
11     duracion = duración;
12   }
13 }
```

- ⚠ ¡No funciona! ¿Cómo podemos resolverlo?
- El comportamiento se añade al final de los datos (6-12)
- Mismo nombre de la clase + parámetros *formales*
- Toman el valor de la llamada **new**
- Se copian a los datos

Ámbito

```
1  class Cancion {
2    String titulo;
3    String interprete;
4    int duration;
5    ...;
6    Cancion (String titulo,
7             String interprete,
8             int duration) {
9      titulo = titulo;
10     interprete = interprete;
11     duracion = duracion;
12   }
13 }
```

Ámbito

```
1  class Cancion {
2    String titulo;
3    String interprete;
4    int duration;
5    ...;
6    Cancion (String titulo,
7             String interprete,
8             int duration) {
9      titulo = titulo;
10     interprete = interprete;
11     duracion = duracion;
12   }
13 }
```

- ¿Por qué rosas?
- Java, como todos los lenguajes, resuelve el ámbito de *dentro a afuera*

Ámbito

```
1  class Cancion {
2    String titulo;
3    String interprete;
4    int duration;
5    ...;
6    Cancion (String titulo,
7             String interprete,
8             int duration) {
9      titulo = titulo;
10     interprete = interprete;
11     duracion = duracion;
12   }
13 }
```

- ¿Por qué rosas?
- Java, como todos los lenguajes, resuelve el ámbito de *dentro a afuera*
- 👎 Primero en el bloque `{...}`

Ámbito

```
1  class Cancion {
2    String titulo;
3    String interprete;
4    int duration;
5    ...;
6    Cancion (String titulo,
7             String interprete,
8             int duration) {
9      titulo = titulo;
10     interprete = interprete;
11     duracion = duracion;
12   }
13 }
```

- ¿Por qué rosas?
- Java, como todos los lenguajes, resuelve el ámbito de *dentro a afuera*

👎 Primero en el bloque `{...}`

👍 Después en los parámetros

Ámbito

```
1  class Cancion {
2    String titulo;
3    String interprete;
4    int duration;
5    ...;
6    Cancion (String titulo,
7             String interprete,
8             int duration) {
9      titulo = titulo;
10     interprete = interprete;
11     duracion = duracion;
12   }
13 }
```

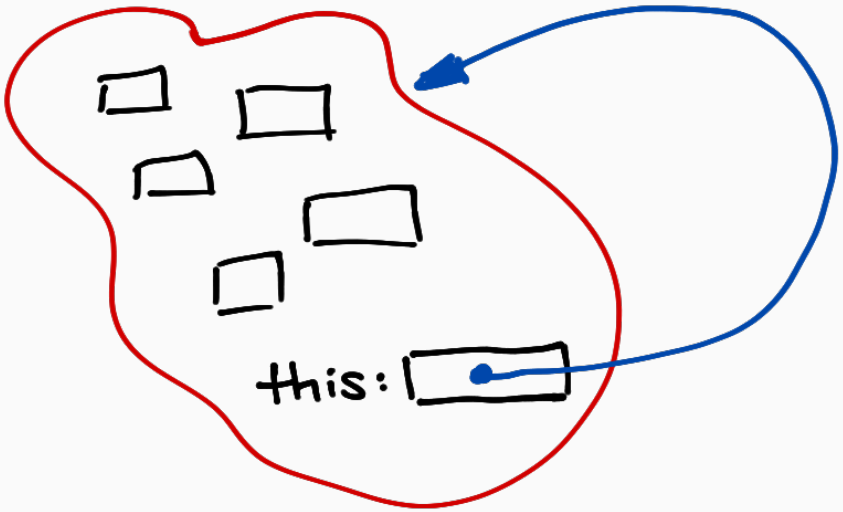
- ¿Por qué rosas?
- Java, como todos los lenguajes, resuelve el ámbito de *dentro a afuera*

👎 Primero en el bloque `{...}`

👍 Después en los parámetros

💬 ¿Cómo puedo referirme a los datos de este objeto usando el mismo nombre que el de un ámbito interno?

This



Existe una **variable** **implícita** llamada **`this`** que contiene una **referencia a *este*** objeto sobre el que se ejecuta el *comportamiento*

Comportamiento para crear

```
class Cancion {
  String titulo;
  String interprete;
  int duration;
  ...;
  Cancion (String titulo,
           String interprete,
           int duration) {
    this.titulo = titulo;
    this.interprete = interprete;
    this.duracion = duración;
  }
}
```

¿Imaginais?

```
canciones[0] =
  new Cancion("Wish you were here",
              "Pink Floyd",
              354);
```

¿Imaginais?

```
System.out.println(cancion[0].minutos());
```

¿Imaginais?

```
System.out.println( cancion[0] .minutos () );
```

Siga la flecha: `_._`

hacia un comportamiento: `minutos()`

¿Imaginais?

```
System.out.println( cancion[0] .minutos () );
```

5mins

Siga la flecha: `_._`

hacia un comportamiento: `minutos()`

Comportamiento para minutos

```
1  class Cancion {
2    ...
3    int duration;
4    ...;
5    Cancion (...) {
6      ...
7    }
8           minutos() {
9
10            ...;
11
12   }
13 }
```

- `minutos()` es el nuevo comportamiento (8-12)

Comportamiento para minutos

```
1  class Cancion {
2     ...
3     int duration;
4     ...;
5     Cancion (...) {
6        ...
7     }
8     String minutos() {
9
10                ...;
11
12    }
13 }
```

- `minutos()` es el nuevo comportamiento (8-12)
- Obligatorio indicar lo que *devuelve*: `String`

Comportamiento para minutos

```
1  class Cancion {
2    ...
3    int duration;
4    ...;
5    Cancion (...) {
6      ...
7    }
8    String minutos() {
9      return
10           duration / 60
11       + "mins";
12   }
13 }
```

- `minutos()` es el nuevo comportamiento (8-12)
- Obligatorio indicar lo que *devuelve*: `String`
- Programamos el comportamiento

Comportamiento para minutos

```
1  class Cancion {
2    ...
3    int duration;
4    ...;
5    Cancion (...) {
6      ...
7    }
8    String minutos() {
9      return
10       this.duration / 60
11       + "mins";
12   }
13 }
```

- `minutos()` es el nuevo comportamiento (8-12)
- Obligatorio indicar lo que *devuelve*: `String`
- Programamos el comportamiento
- Podemos usar **`this`** pero aquí no es necesario

¿Imaginais?

```
System.out.println(cancion[0]);
```

```
| Wish you were here | Pink Floyd | 5mins |
```

¿Imaginais? pero antes...

```
System.out.println(cancion[0].bonito());
```

```
| Wish you were here | Pink Floyd | 5mins |
```

Comportamiento para bonito

- Mismos pasos que lo que hemos hecho con `minutos()`
- De hecho podemos usar `minutos()` para programar `bonito()`

¿Imaginais incluso?

```
System.out.println(cancion[0]);
```

```
| Wish you were here | Pink Floyd | 5mins |
```

Comportamiento para bonito

¿Cómo hacemos la magia?

Comportamiento para bonito

¿Cómo hacemos la magia?

```
public String toString() {
  return "| " + titulo
    + " | " + interprete
    + " | " + minutos() + " |";
}
```

Comportamiento para bonito

¿Cómo hacemos la magia?

`toString()` es un método especial que existe en todas las clases

```
public String toString() {
  return "| " + titulo
    + " | " + interprete
    + " | " + minutos() + " |";
}
```

Comportamiento para duracion

```
System.out.println(cancion[2].duracion());
```

```
4:02
```

Comportamiento para duracion

```
System.out.println(cancion[2].duracion());
```

```
4:02
```

Cuidado cuando los segundos son < 10,

no queremos `"4:2"`

Comportamiento para duracion

```
System.out.println(cancion[2].duracion());
```

```
4:02
```

Cuidado cuando los segundos son < 10,
no queremos `"4:2"`

¿Puedo repetir el mismo nombre en un comportamiento que en un dato?

Comportamiento para duracion

```
System.out.println(cancion[2].duracion());
```

```
4:02
```

Cuidado cuando los segundos son < 10,
no queremos `"4:2"`

Puedo repetir el mismo nombre en un comportamiento que en un dato