

Introducción a la Programación Para Dispositivos Móviles

Sesión 1 Parte 1



¿Qué es lo que veremos?

- ◆ ¿Qué es un dispositivo móvil?
- ◆ ¿Es un diferente paradigma de programación?
- ◆ Un poco de historia
- ◆ Principales plataformas de desarrollo
- ◆ Patrón MVC

¿Qué es un dispositivo
móvil?

Todos conocemos...



Smartphone



Tablet



Lentes



Relojes



TVs

Pero también

Un dispositivo móvil es el artefacto que puedes llevar contigo en cualquier momento, ser utilizado casi instantáneamente, es personal y puede conectarse a una red

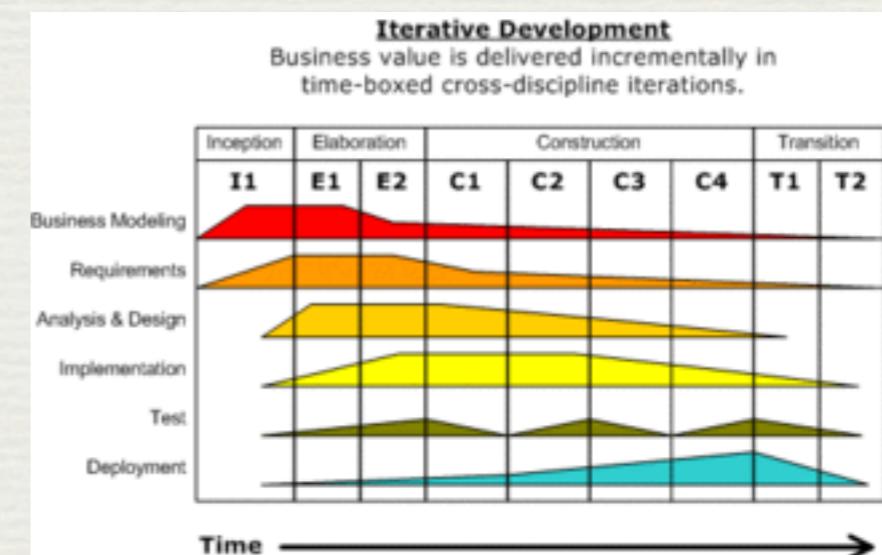
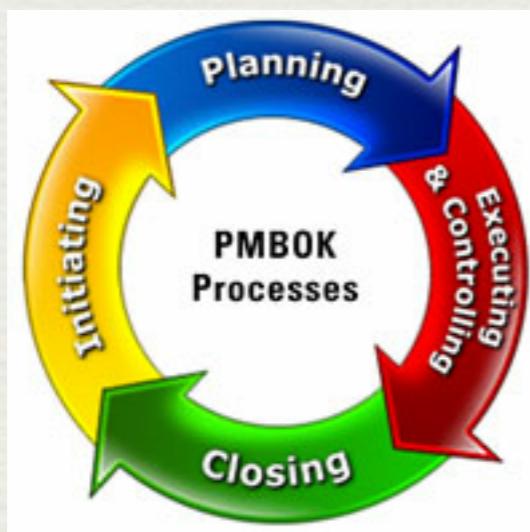
¿Es un diferente paradigma
de programación?

Diferencias

Desarrollo desktop / web

Nos enfrentamos a las reglas de la ingeniería de software:

- Metodologías de desarrollo
- Gestión de proyectos



Mientras más funciones tenga y se termine en el tiempo, mejor

Diferencias

Desarrollo móvil

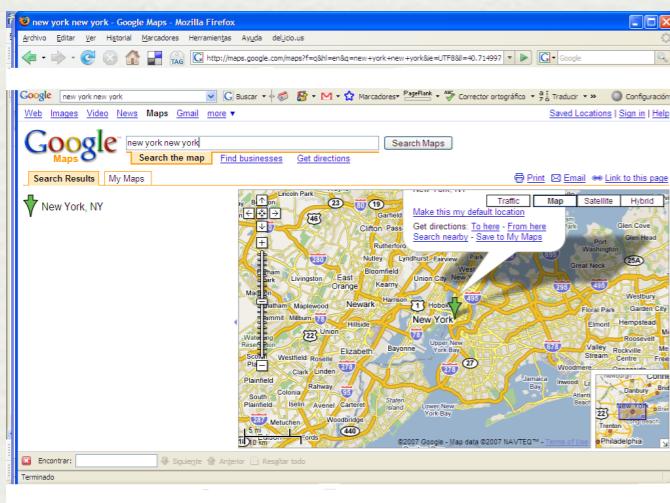
Nos enfrentamos contra la tiranía de la experiencia de usuario



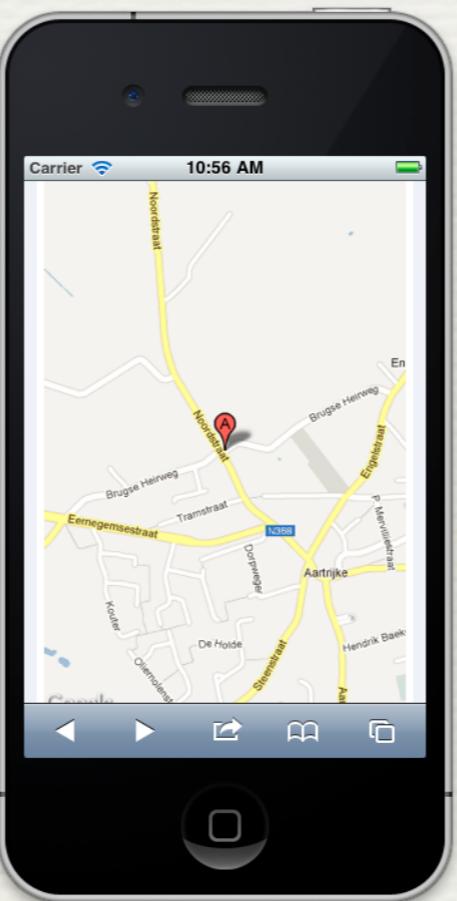
Experiencia de usuario

¿Qué es una solución móvil?

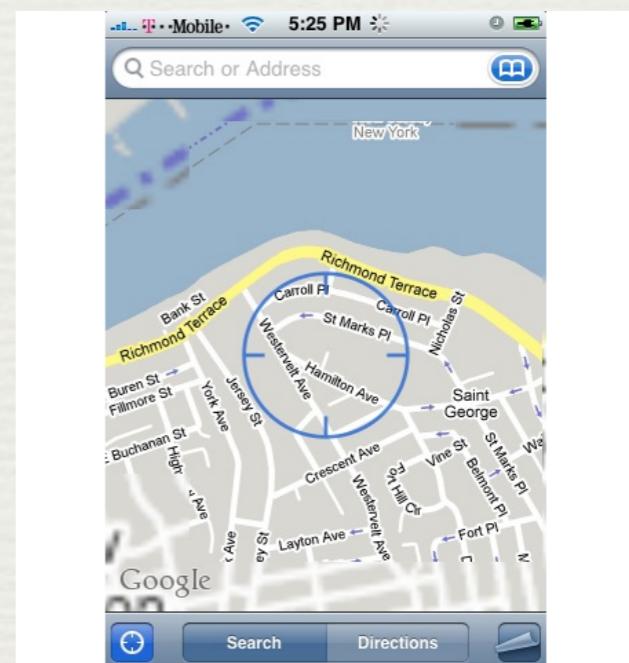
Solución Móvil



+



+



Web

Web Mobile

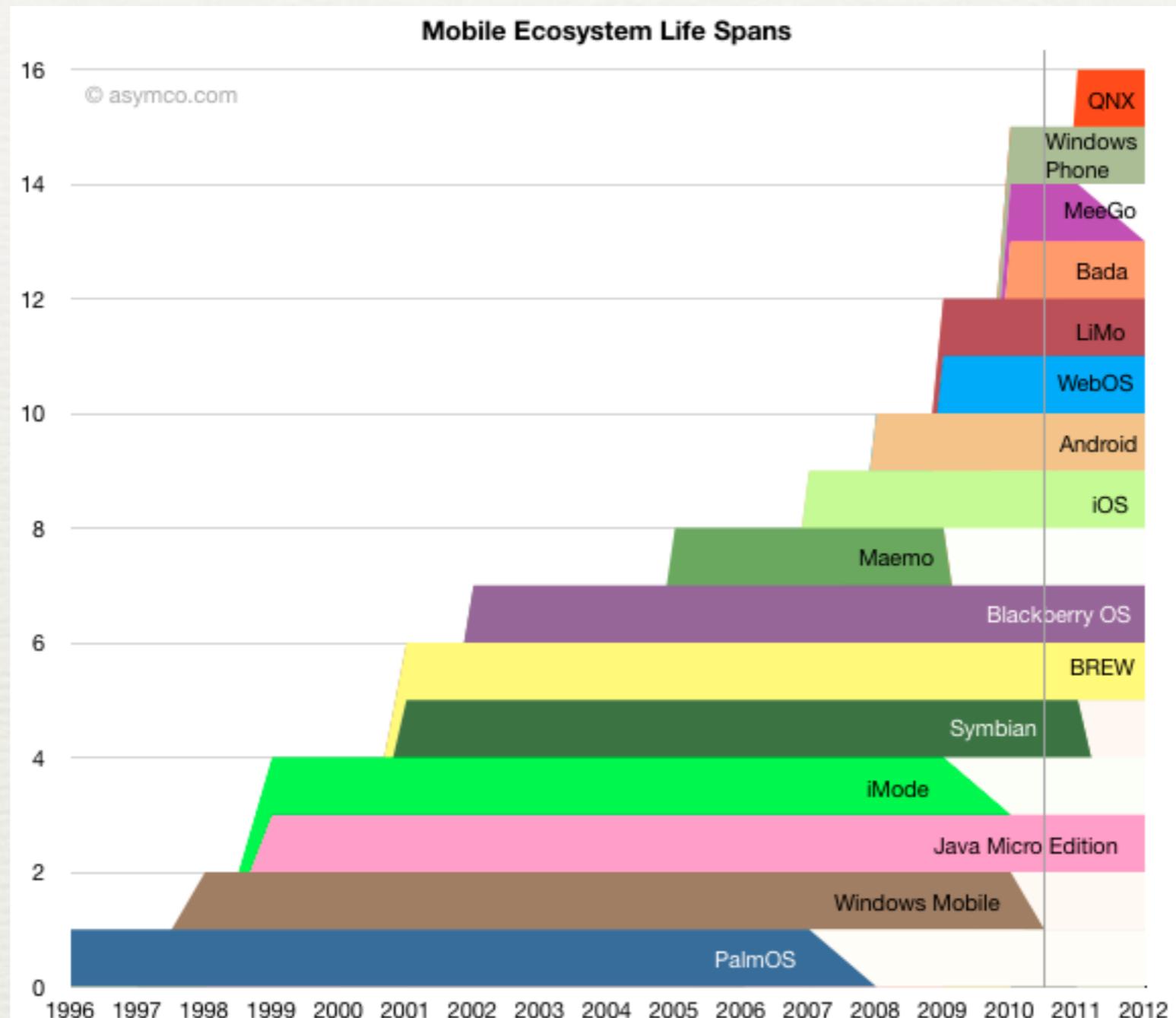
Nativa

Nuevos paradigmas

- ♦ Canales múltiples
 - ♦ No restringirse a solo un canal
 - ♦ Centrarse en capacidades de canal
 - ♦ Tener cuidado con escalabilidad
- ♦ Nuevas maneras de ofrecer un servicio
 - ♦ Llevar servicios ya existentes a los “dedos de tus clientes”
 - ♦ Proveer lo que el usuario quiere, en el momento y el lugar que lo necesita
- ♦ Simplificarle la vida a los clientes
 - ♦ Al ofrecer menor cantidad de información, la aplicación es más enfocada y por lo tanto más sencilla.

Un poco de historia

Evolución Histórica



Nuevos “jugadores”:

- Ubuntu OS, Firefox OS

Principales Plataformas

Pro

iOS

- Gran cantidad de dispositivos vendidos
- Un solo fabricante de hardware, desarrollo dirigido (baja fragmentación)
- Alta integración de software con hardware
- Market muy desarrollado (App Store)

Cons

- Lenguaje de programación Objective C
- Desarrollo solamente con hardware de Apple
- Dispositivos caros
- Licenciamiento necesario para desarrollo.



Windows phone

- Gran cantidad de dispositivos vendidos
- Lenguaje de programación Java
- Baja barrera de entrada
- Varios markets (muy desarrollados)
- Variedad de dispositivos

- Mediana fragmentación
- Diversidad de APIs (por fabricante)
- Bajo control del ecosistema *

- Tecnología .NET
- Facilidad de desarrollo

- Market no muy desarrollado
- Poca variedad de equipos
- Dispositivos caros
- Integración muy fuerte con Windows

Repaso de Java

Conceptos Java

- ◆ Clases y objetos (diferencia)
- ◆ Atributos y métodos
- ◆ Comunicación entre objetos
- ◆ Herencia
- ◆ Interfases
- ◆ Polimorfismo

Clases y objetos

Clase

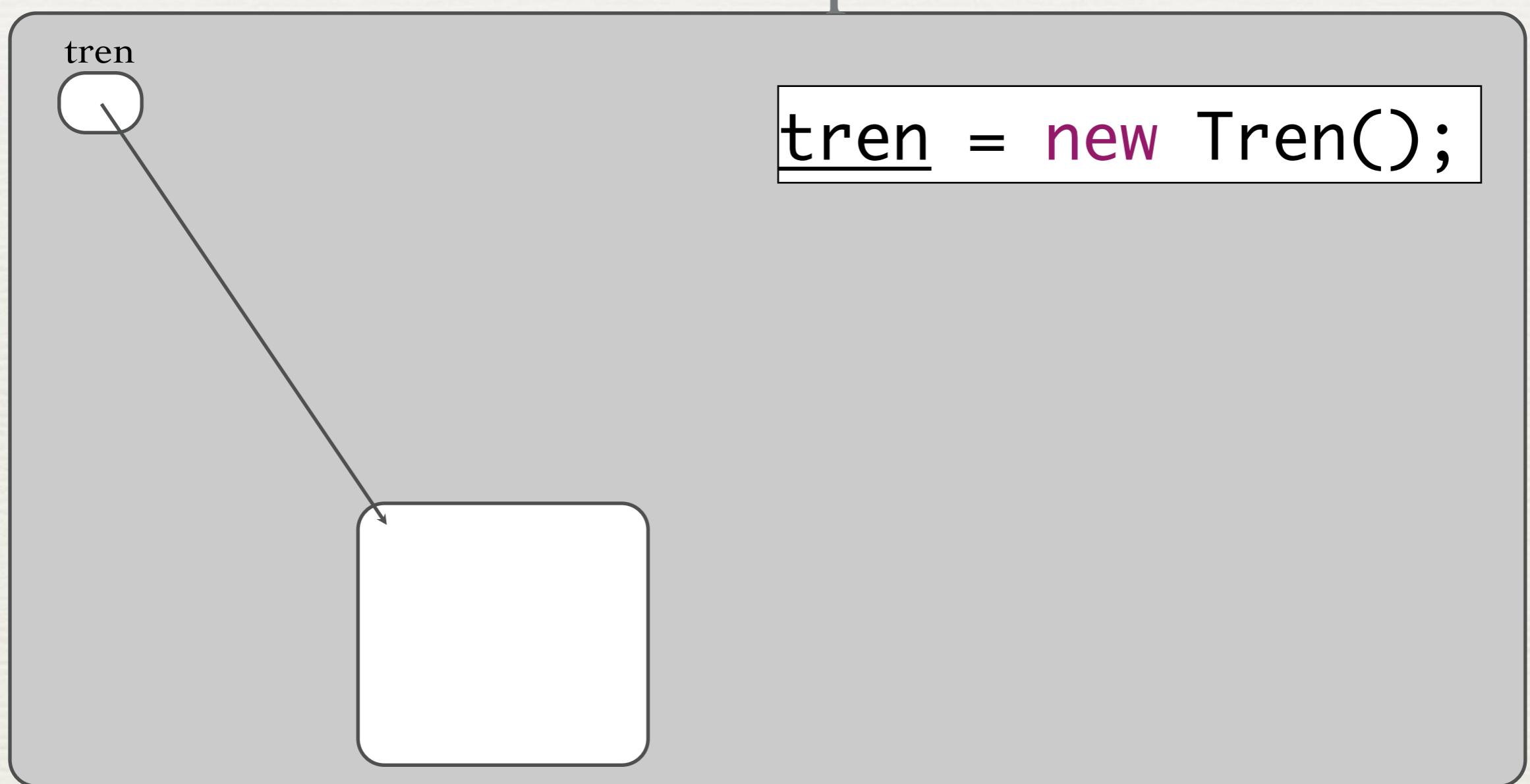
```
package com.javaejemplo.classes;  
public class Tren {  
}
```

Objeto

```
tren = new Tren();
```

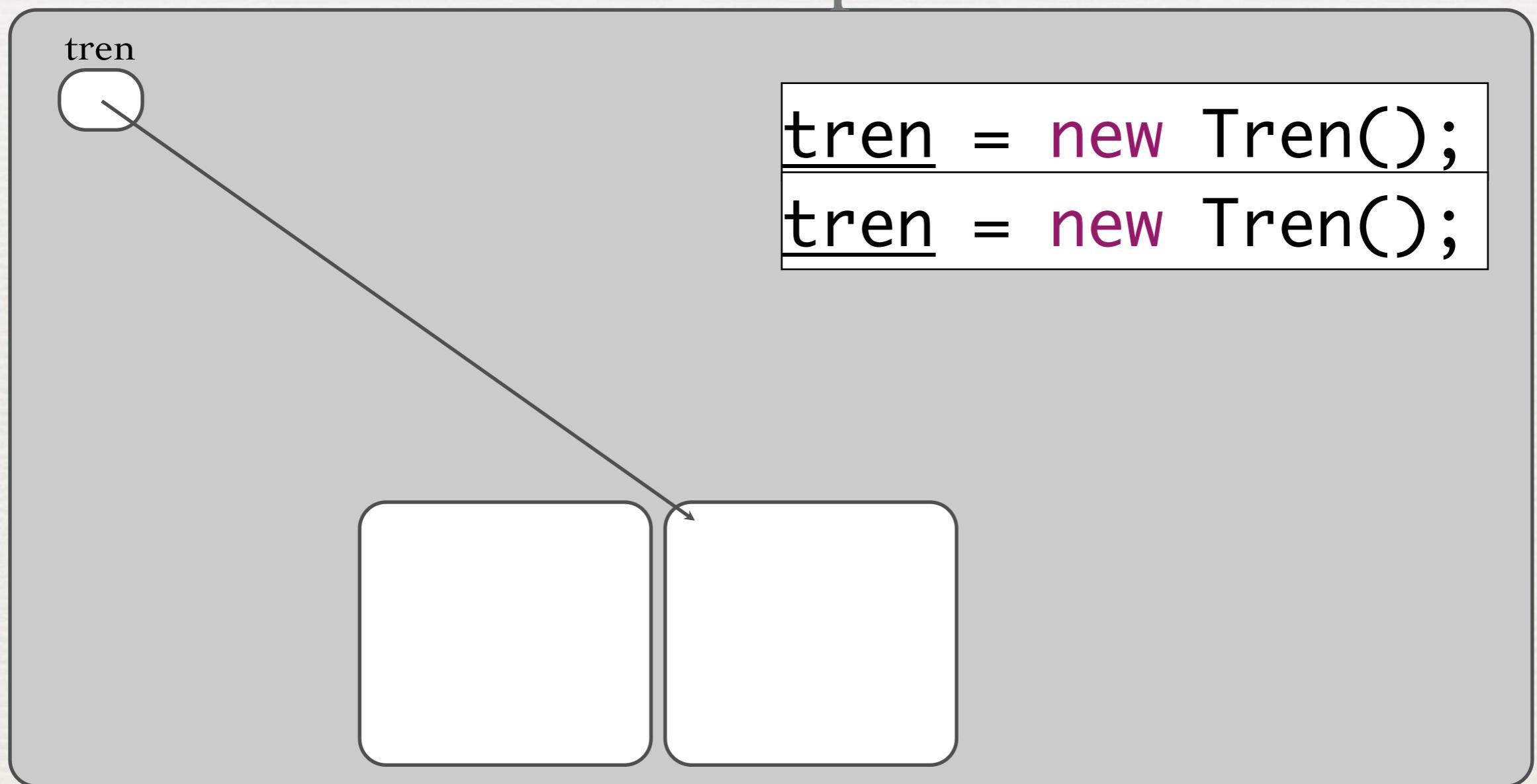
Un objeto es una instancia de una clase

¿Qué sucede en la memoria cuando instanciamos un objeto?

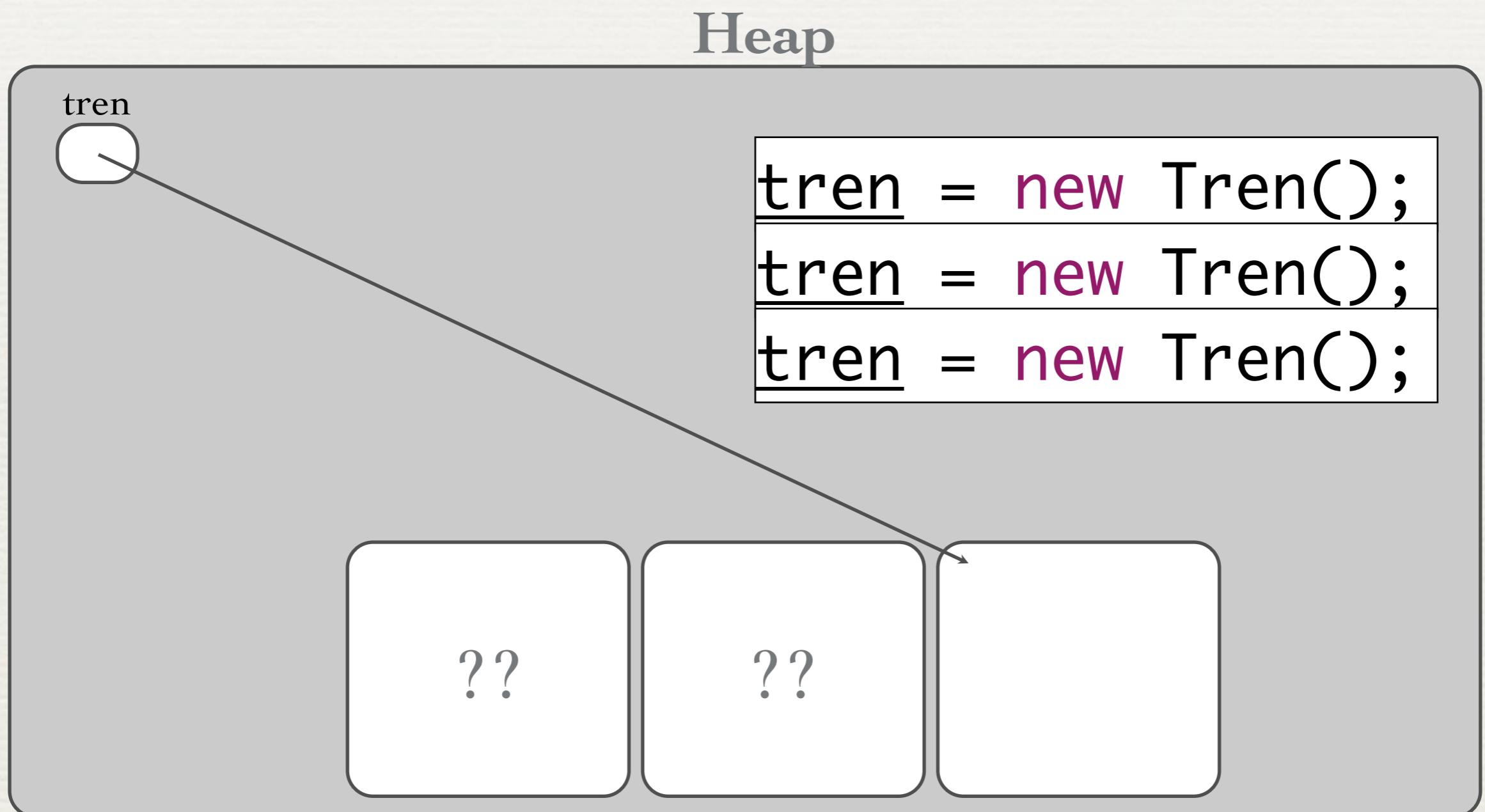


Así podemos instanciar varios

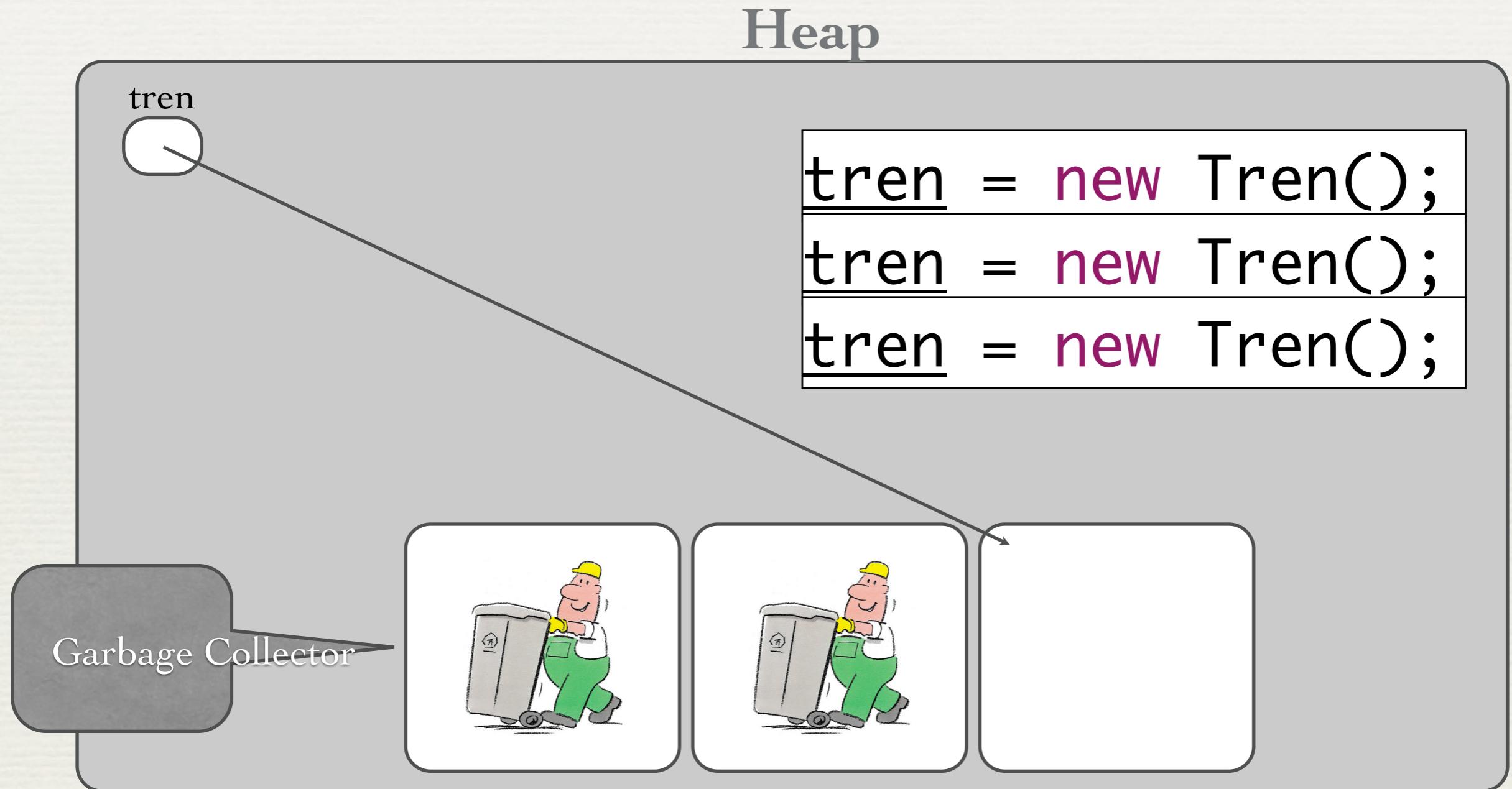
Heap



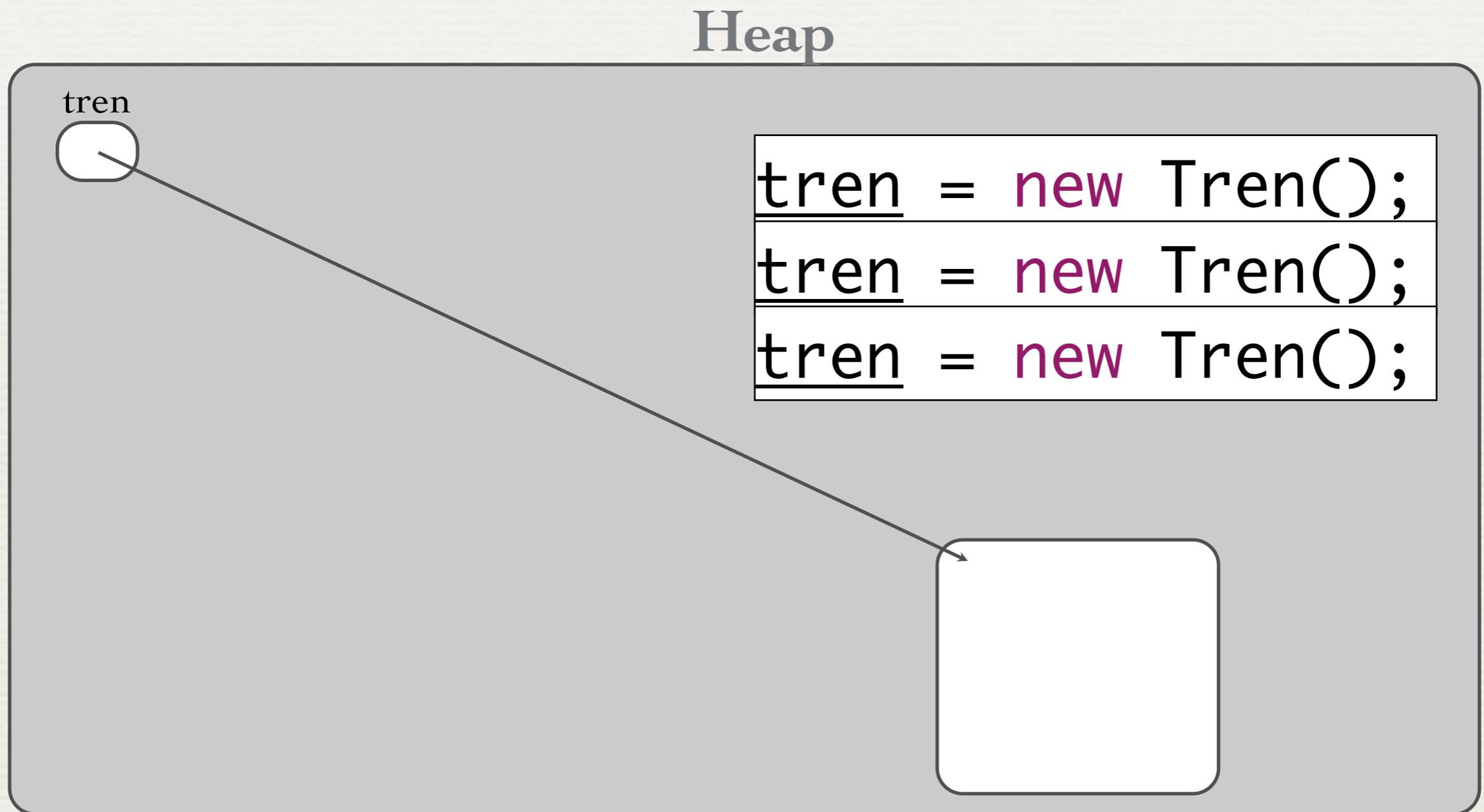
Así podemos instanciar varios



El sistema llama al proceso Garbage Collector



Así se puede reutilizar ese espacio



Atributos y métodos

```
public class Tren {  
    private String linea;  
    private int anhoFabricacion;  
    public double precio;  
  
    public String getLinea() {  
        return linea;  
    }  
    public void setLinea(String linea) {  
        this.linea = linea;  
    }  
    public int getAnhoFabricacion() {  
        return anhoFabricacion;  
    }  
    public void setAnhoFabricacion(int anhoFabricacion) {  
        this.anhoFabricacion = anhoFabricacion;  
    }  
}
```

Atributos

Métodos

Comunicación entre objetos

Persona



Automóvil



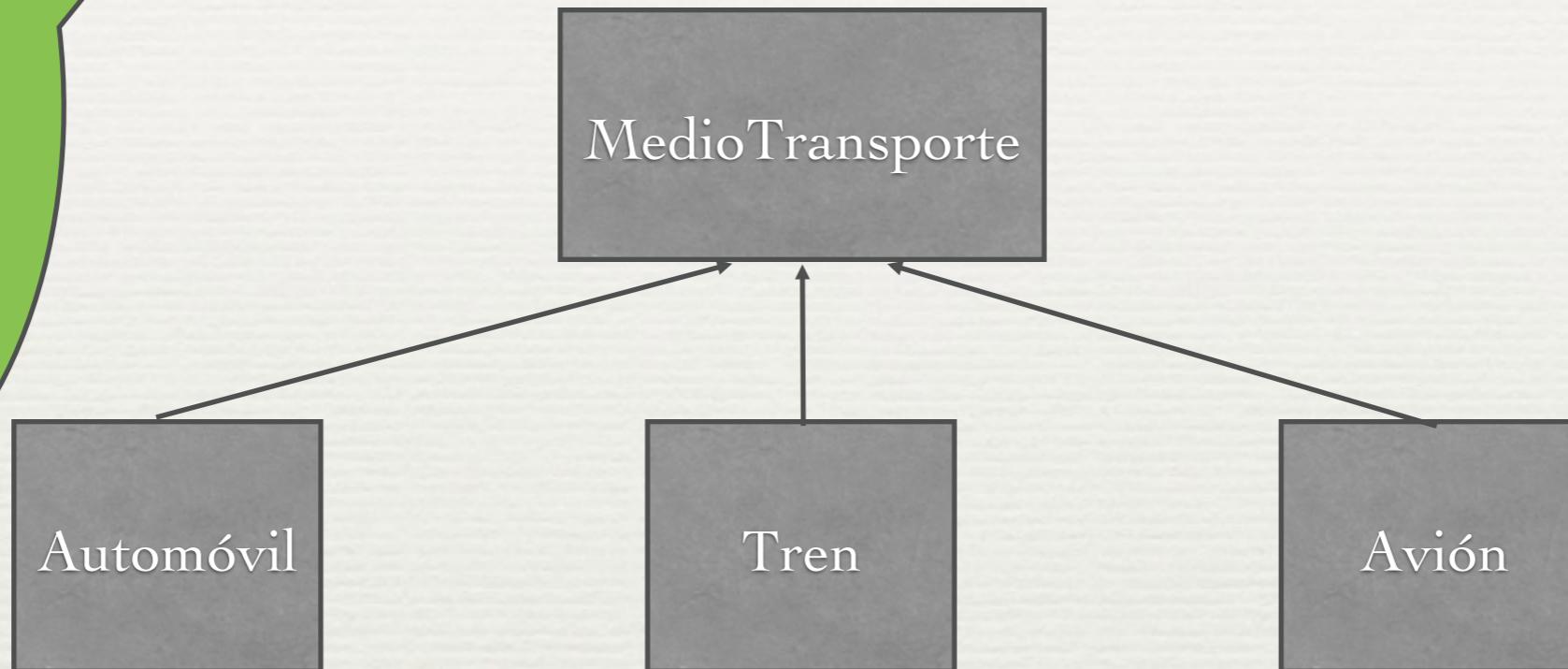
arrancar()

```
public class Persona {  
  
    public void irATrabajar(){  
        Automovil automovil = new Automovil();  
        automovil.arrancar();  
    }  
  
}
```

```
public class Automovil {  
  
    public void arrancar(){  
        // Código del método arrancar  
    }  
  
}
```

Herencia

En Java
no hay
herencia
múltiple



```
public class ClaseHijo extends ClasePadre{  
    //....  
}
```

Interfaces

- ◆ Solo declara (no implementa) métodos que deberán ser implementados por las clases que implementen la interface dada
- ◆ Una clase puede implementar **varias** interfaces

```
public interface IAccionesMedioTransporte {  
    public void arrancar();  
    public void frenar();  
}
```

```
public class Avion implements IAccionesMedioTransporte{  
  
    @Override  
    public void arrancar() {  
        // TODO Código para arrancar  
    }  
  
    @Override  
    public void frenar() {  
        // TODO Código para frenar  
    }  
}
```

Polimorfismo

En programación orientada a objetos el polimorfismo se refiere a la posibilidad de enviar un mensaje a un grupo de objetos cuya naturaleza puede ser heterogénea. El único requisito que deben cumplir los objetos que se utilizan de manera polimórfica es saber responder al mensaje que se les envía.

Wikipedia [http://es.wikipedia.org/wiki/Polimorfismo_\(informática\)](http://es.wikipedia.org/wiki/Polimorfismo_(informática))

```
public class MedioTransporte {  
    private String marca;  
  
    public String getMarca() {  
        return marca;  
    }  
  
    public void setMarca(String marca) {  
        this.marca = marca;  
    }  
}
```

```
public class Persona {  
    //...  
  
    public void iniciar(){  
        Automovil automovil = new Automovil();  
        String marca = automovil.getMarca();  
    }  
}
```

```
public class Automovil extends MedioTransporte{  
}
```

Métodos abstractos

```
public abstract class MedioTransporte {  
    //...  
    public abstract void limpiar();  
}
```

Creamos una clase que implemente los métodos abstractos

Implementamos el método dinámicamente durante la creación de la instancia

```
public class Automovil extends MedioTransporte{  
  
    @Override  
    public void limpiar() {  
        // TODO Código para limpiar  
    }  
}
```

```
public class Persona {  
  
    public void iniciar(){  
        MedioTransporte medioTransporte = new MedioTransporte() {  
  
            @Override  
            public void limpiar() {  
                // TODO Código implementado dinámicamente  
                // en tiempo de ejecución  
            }  
        };  
        medioTransporte.limpiar();  
    }  
}
```

Polimorfismo de interfaces

```
public interface IAccionesMedioTransporte {  
    public void arrancar();  
    public void frenar();  
}
```

```
public class Avion implements IAccionesMedioTransporte{  
  
    @Override  
    public void arrancar() {  
        // TODO Código para arrancar  
    }  
  
    @Override  
    public void frenar() {  
        // TODO Código para frenar  
    }  
}
```

Forma estática (en compilación)

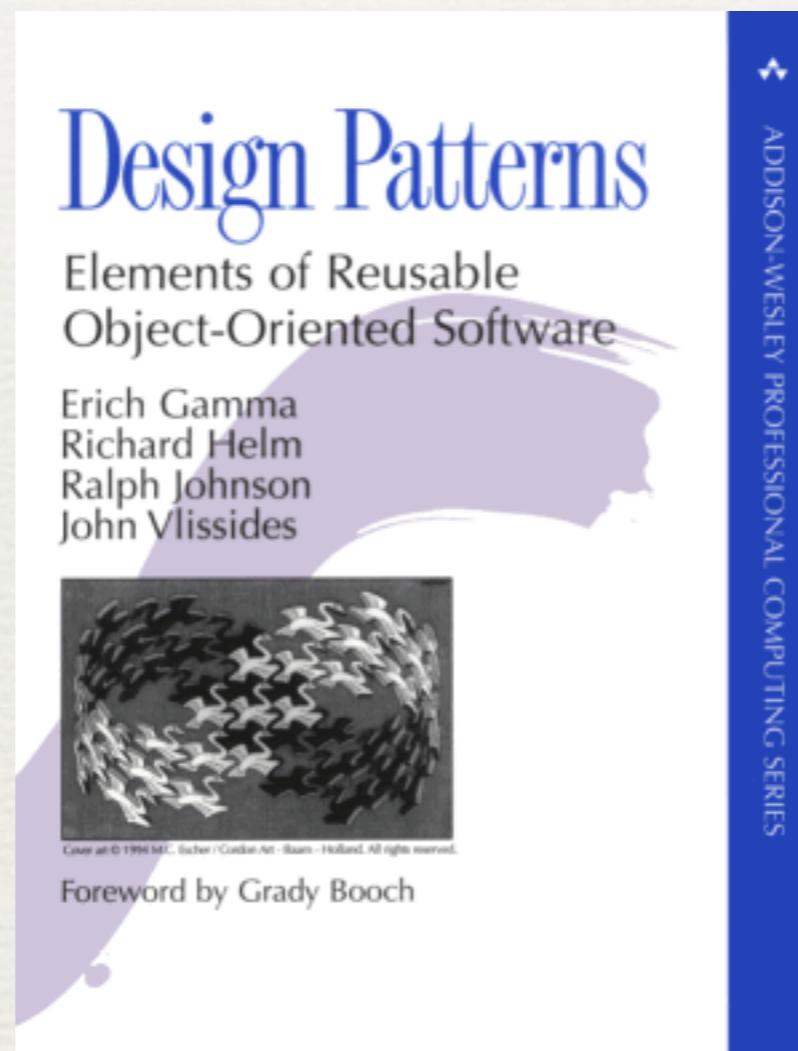
```
public class Avion implements IAccionesMedioTransporte{  
  
    @Override  
    public void arrancar() {  
        // TODO Código para arrancar  
    }  
  
    @Override  
    public void frenar() {  
        // TODO Código para frenar  
    }  
  
}
```

```
public class Persona {  
  
    public void iniciar(){  
        Avion avion = new Avion();  
        avion.arrancar();  
        avion.frenar();  
    }  
}
```

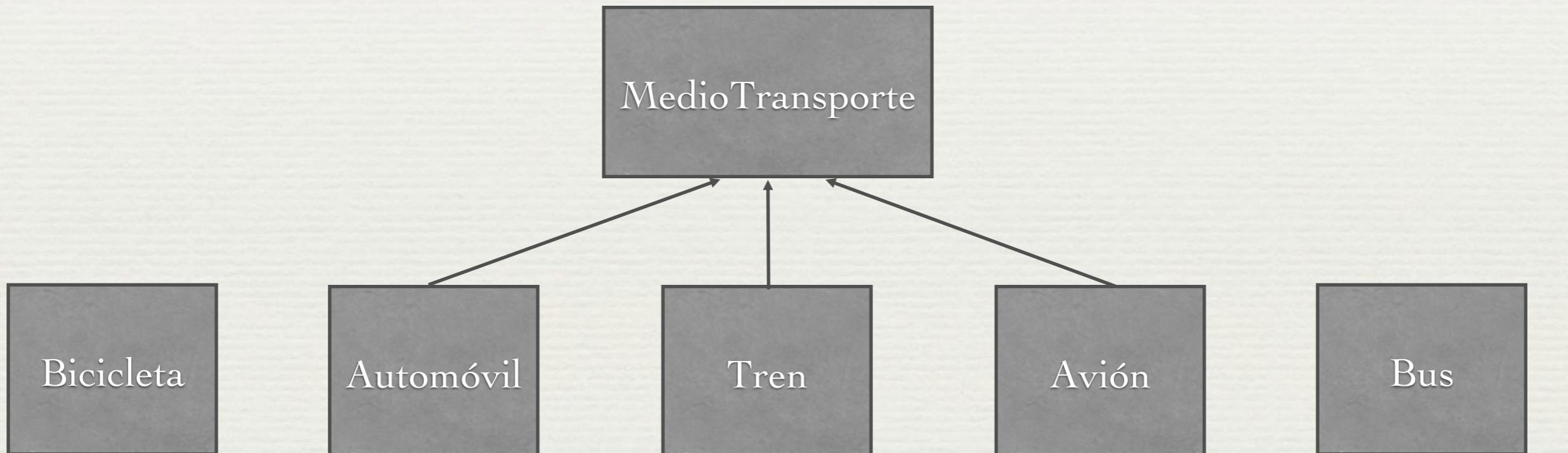
Forma dinámica (en ejecución)

```
public void iniciar(){  
    IAccionesMedioTransporte acciones = new IAccionesMedioTransporte() {  
  
        @Override  
        public void frenar() {  
            // TODO Código para frenar  
        }  
  
        @Override  
        public void arrancar() {  
            // TODO Código para arrancar  
        }  
    };  
    acciones.arrancar();  
    acciones.frenar();  
}
```

¿Qué utilizar, herencia o interfaces?



Estructura



No arranca ni
frena

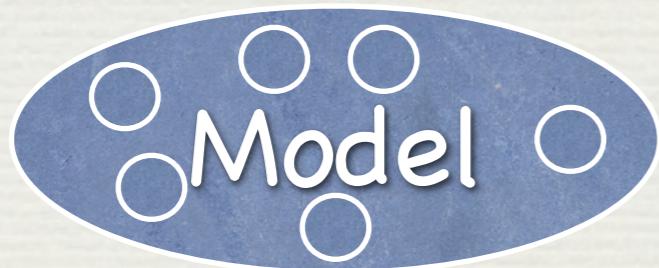
Carga
encomienda

Desacoplando...

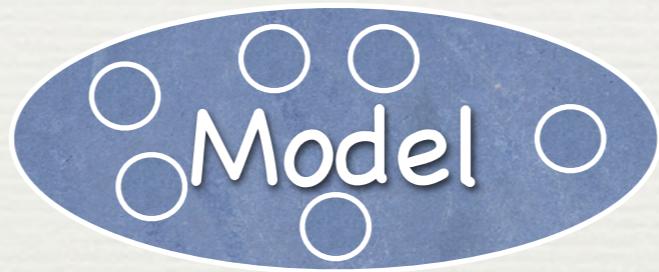
Patrón MVC

Controller

Dividemos objetos de nuestra app en 3 grupos



¿Qué es el model (modelo)?



Model: ¿Qué es lo que nuestra app es o hace?

Ojo! no es cómo se muestra en pantalla!

¿Qué es el controller (controlador)?

Controller

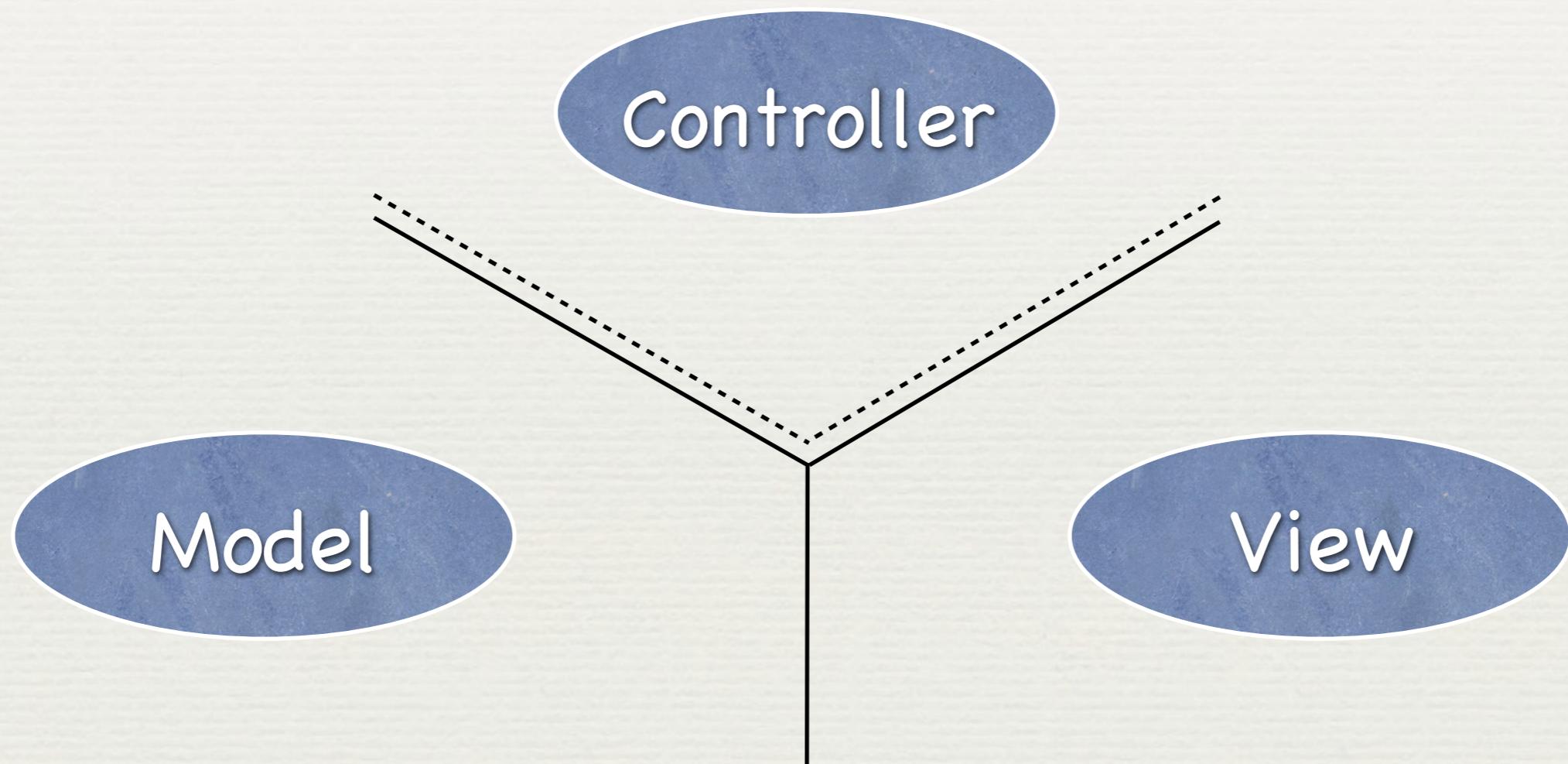
Es el que nos responde la siguiente pregunta: ¿Cómo nuestro modelo es presentado al usuario? (lógica de UI)

¿Qué son las views (vistas)?



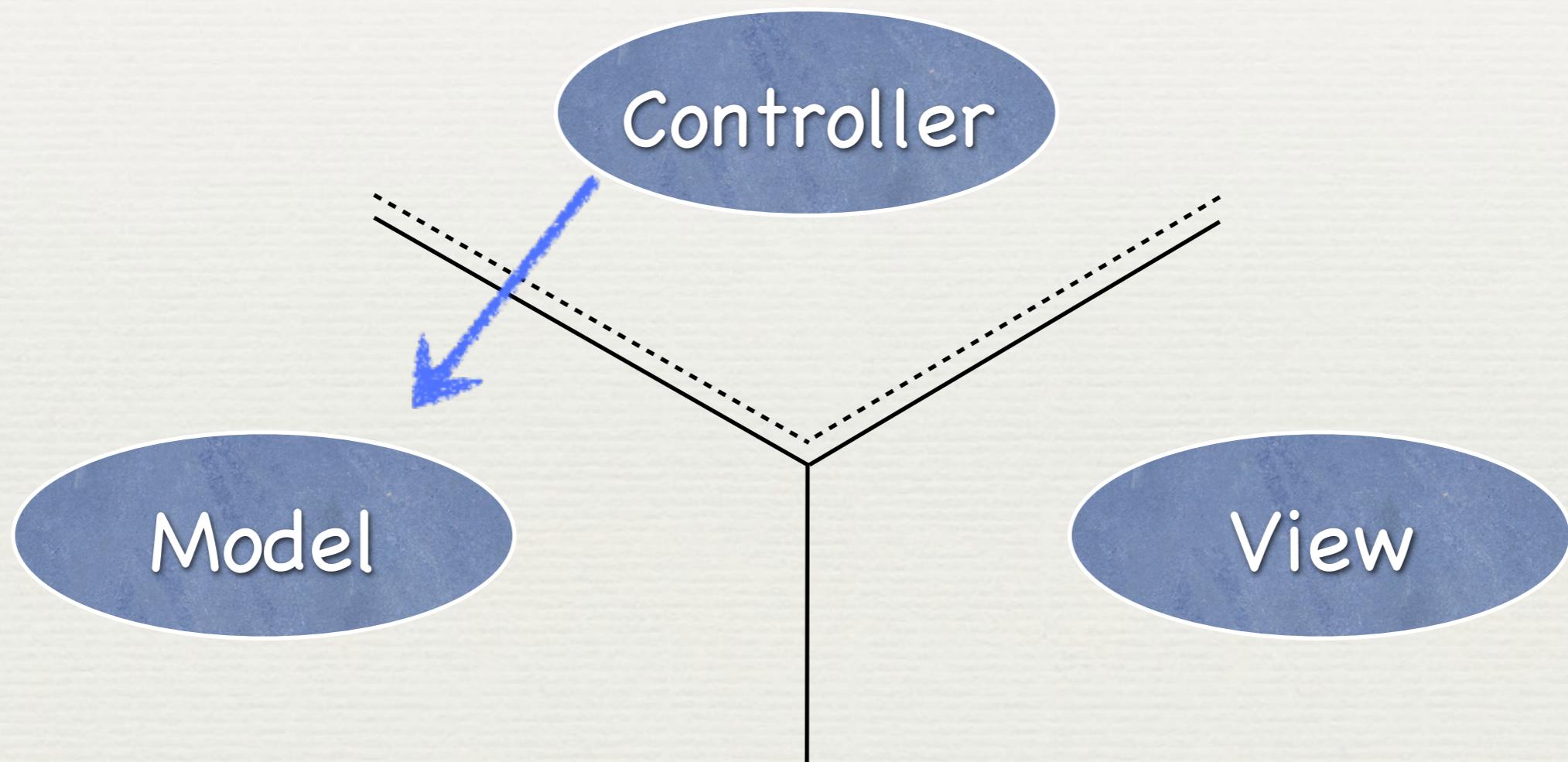
Son los subordinados del Controller

MVC



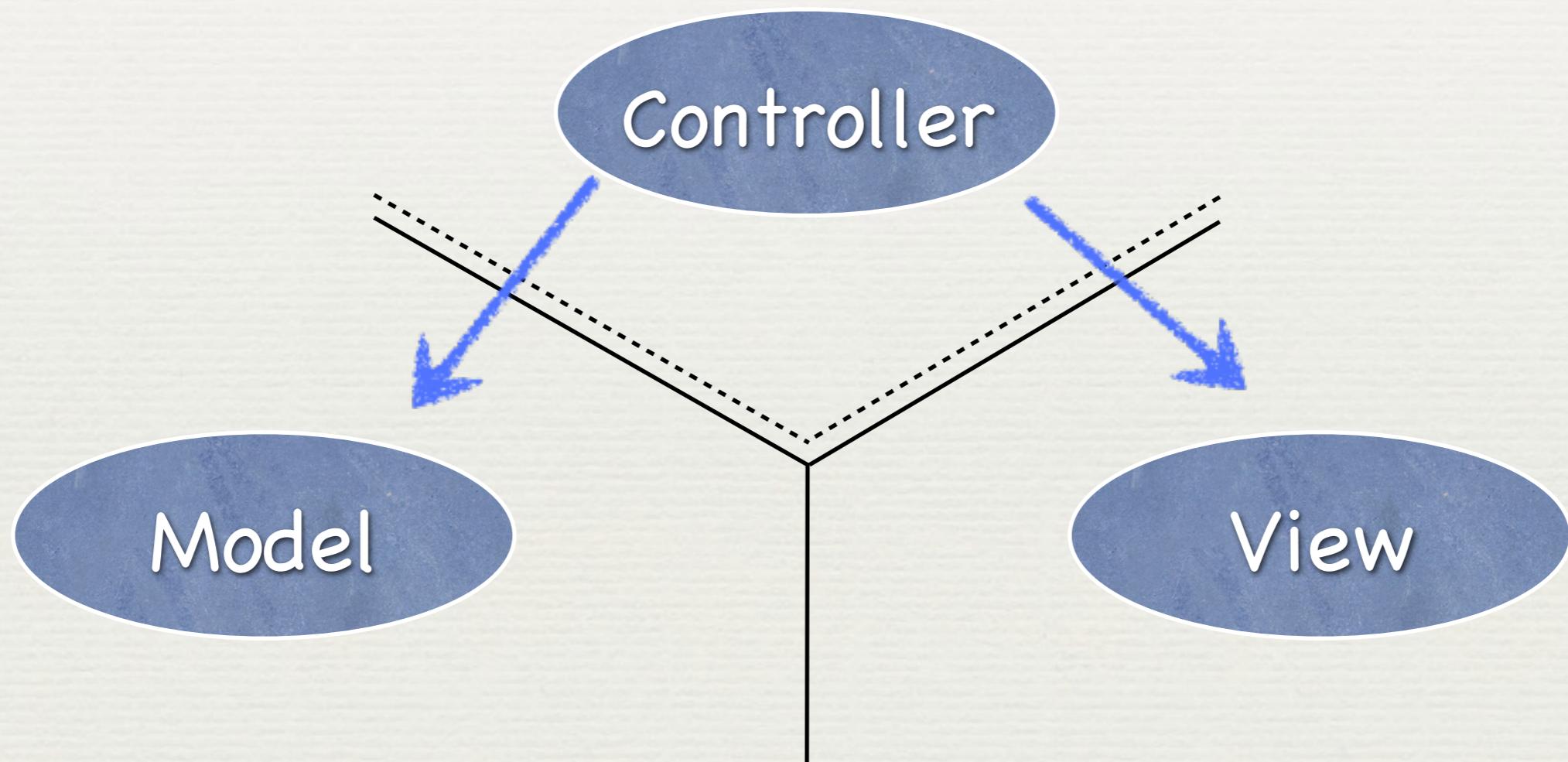
Intenta ordenar y organizar las clases y la comunicación entre ellas

MVC



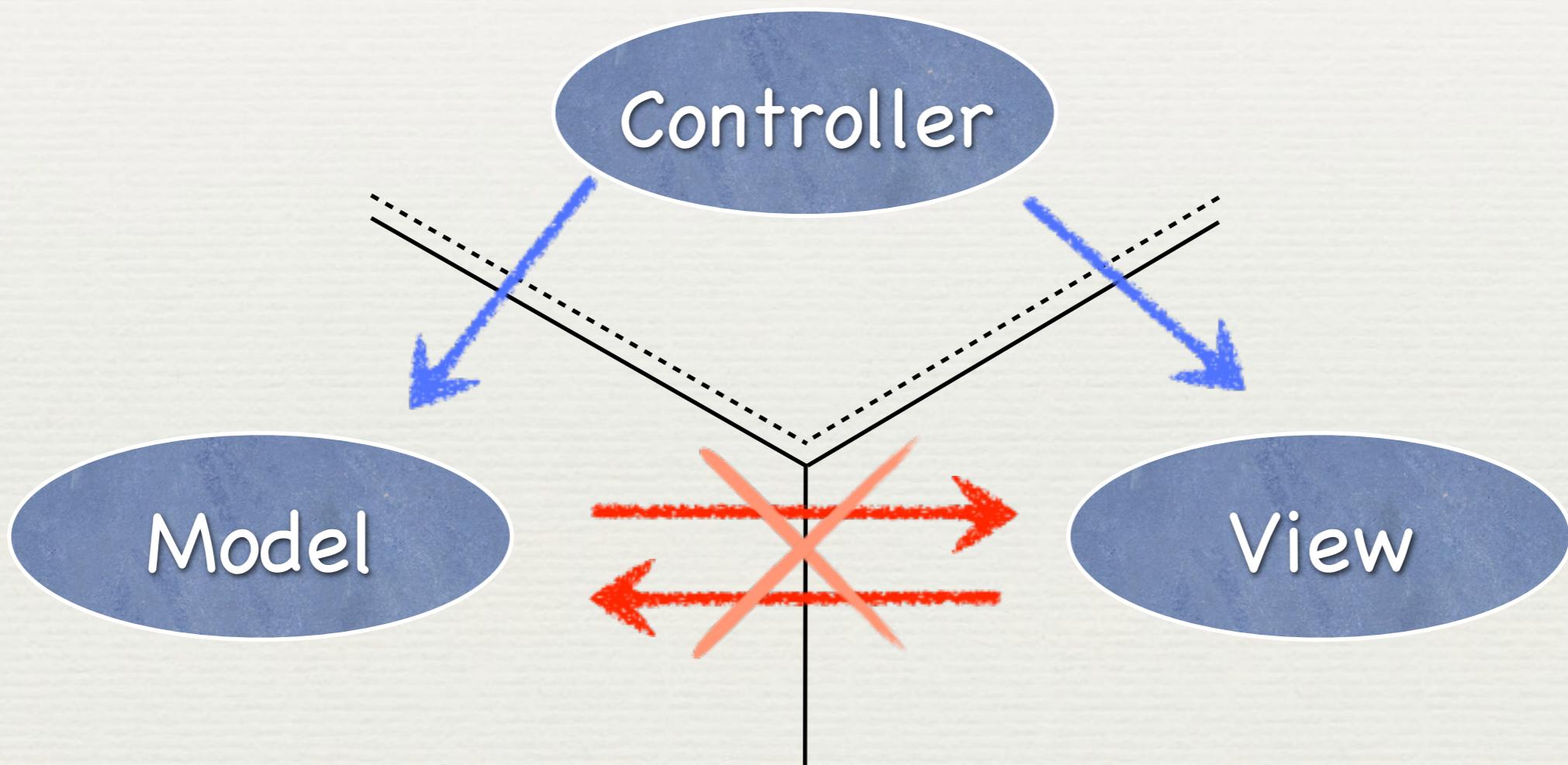
Los controller pueden siempre comunicarse directamente con los modelos

MVC



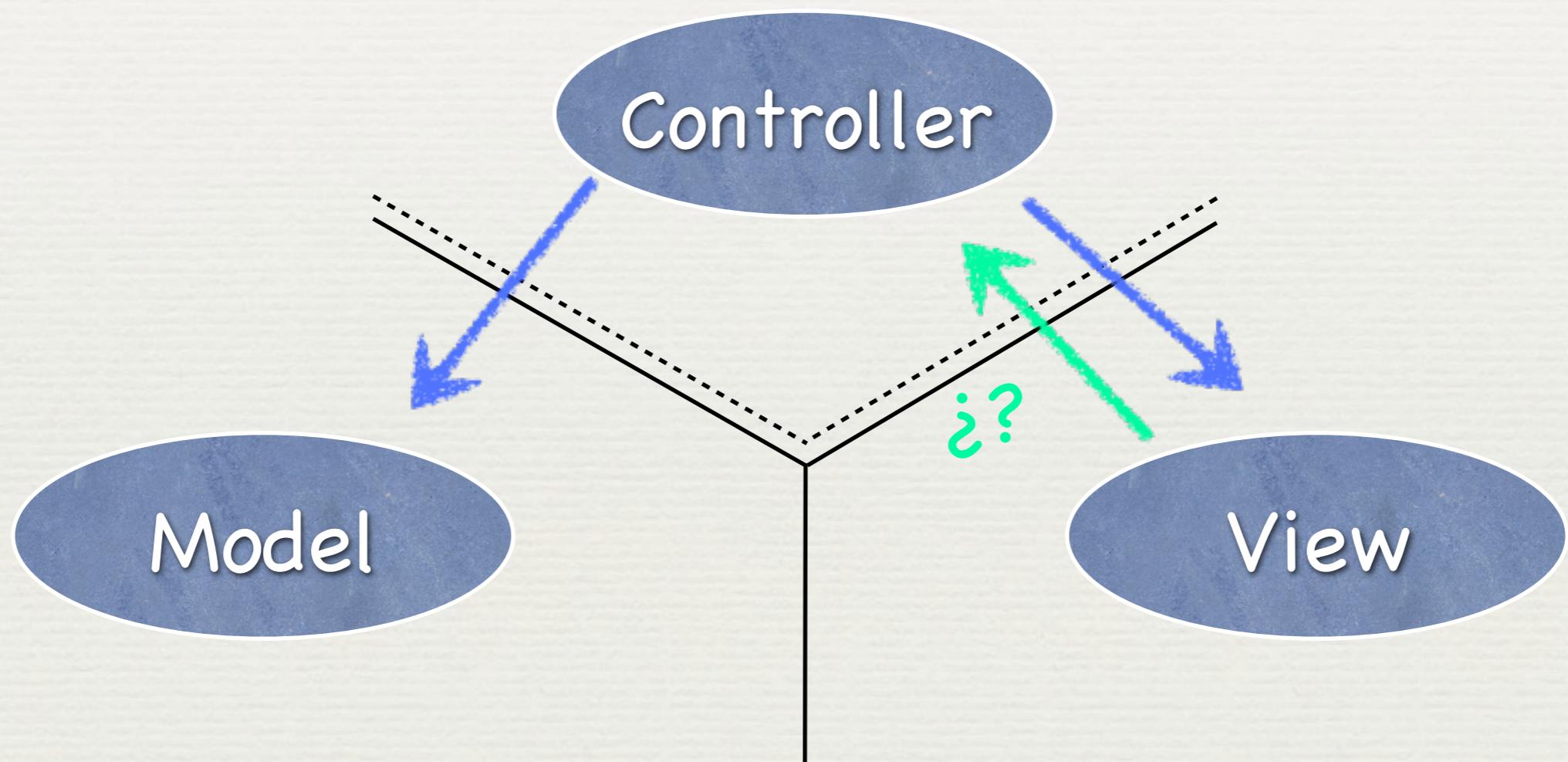
Los controller también pueden comunicarse
directamente con las vistas

MVC



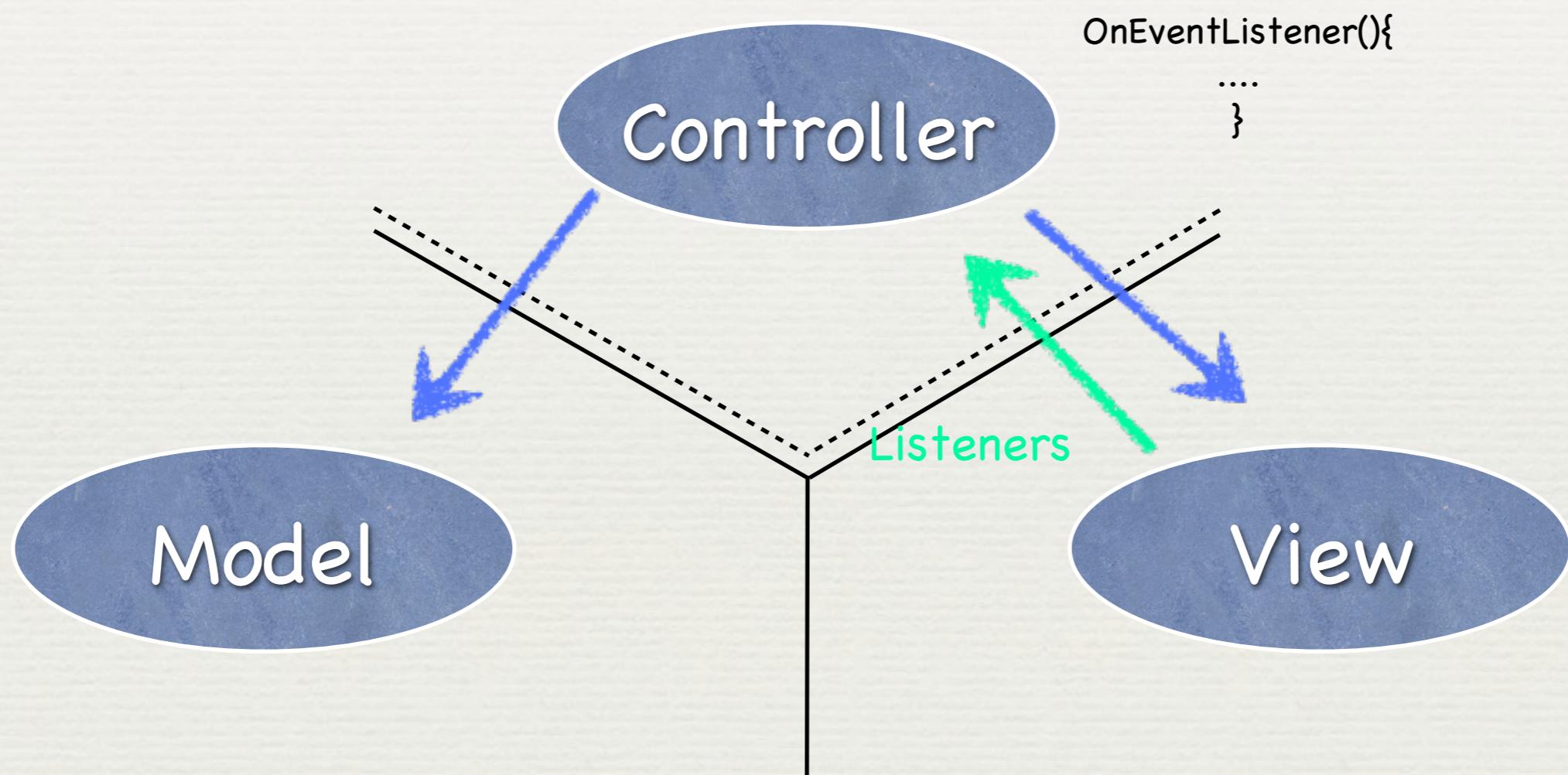
Los Modelos y Vistas nunca pueden comunicarse entre ellos

MVC



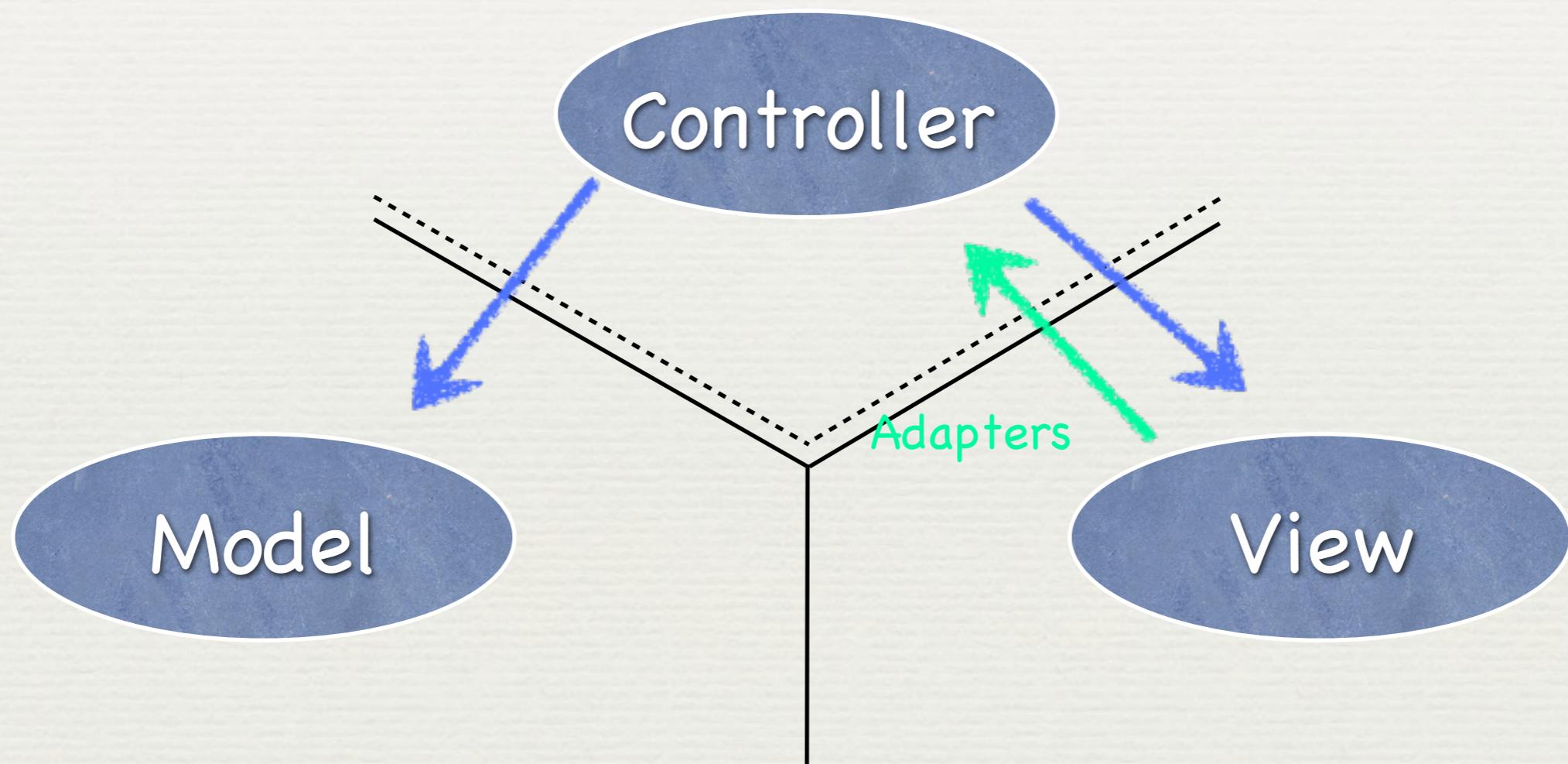
¿Puede la Vista comunicarse con el Controller?

MVC



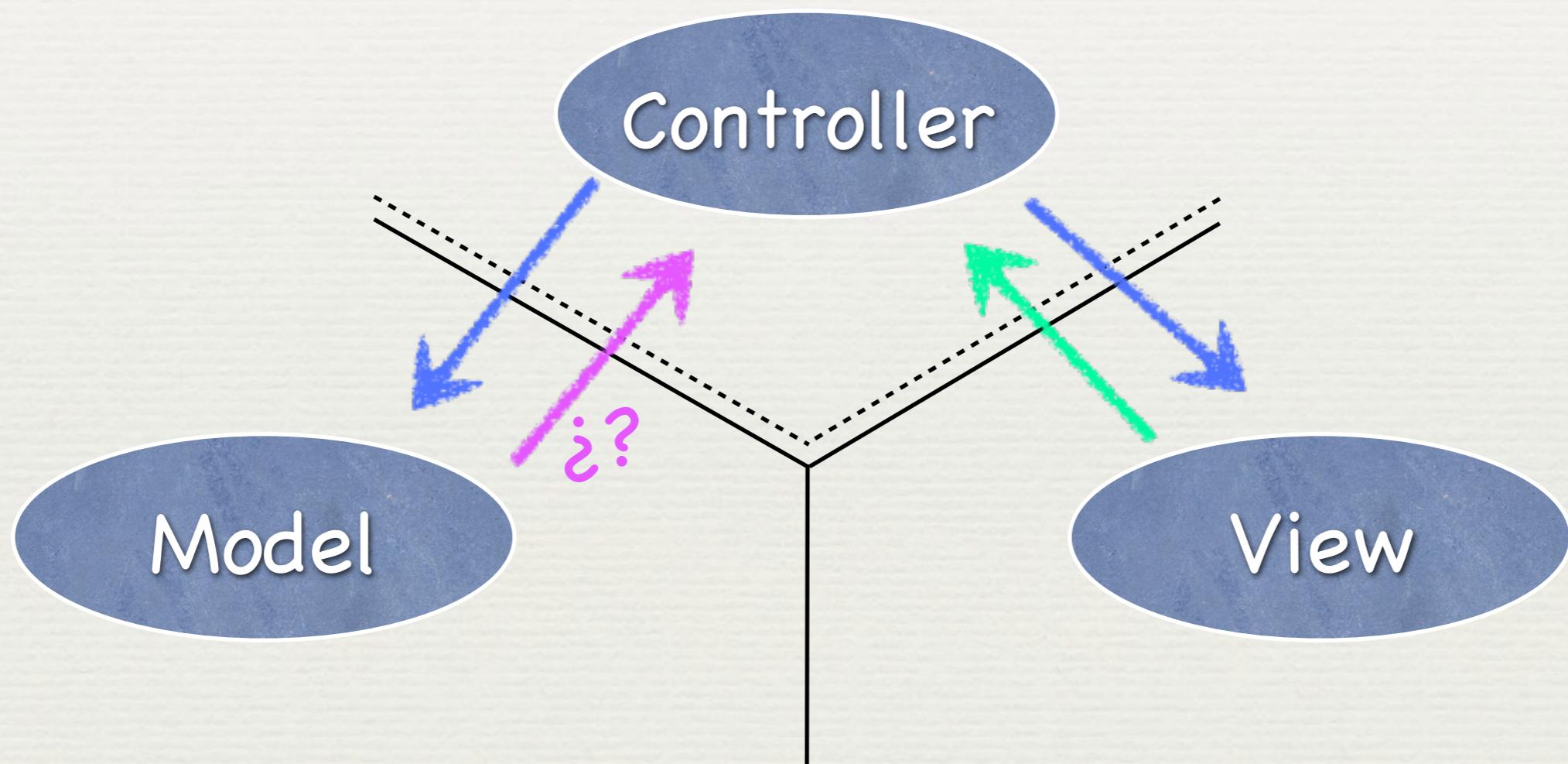
Si, indirectamente. Mediante Listeners

MVC



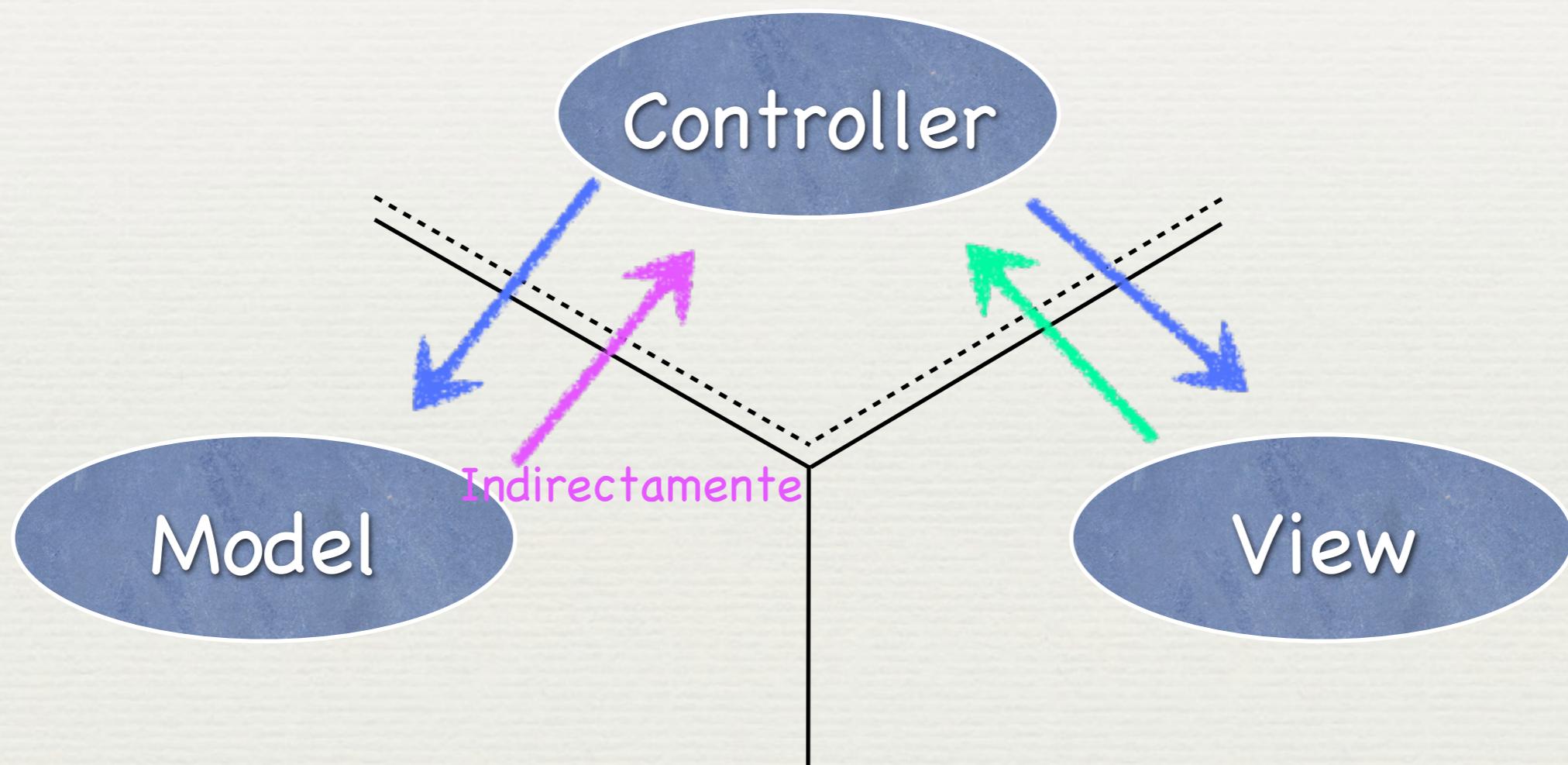
Existe otro método, por medio de Adapters

MVC

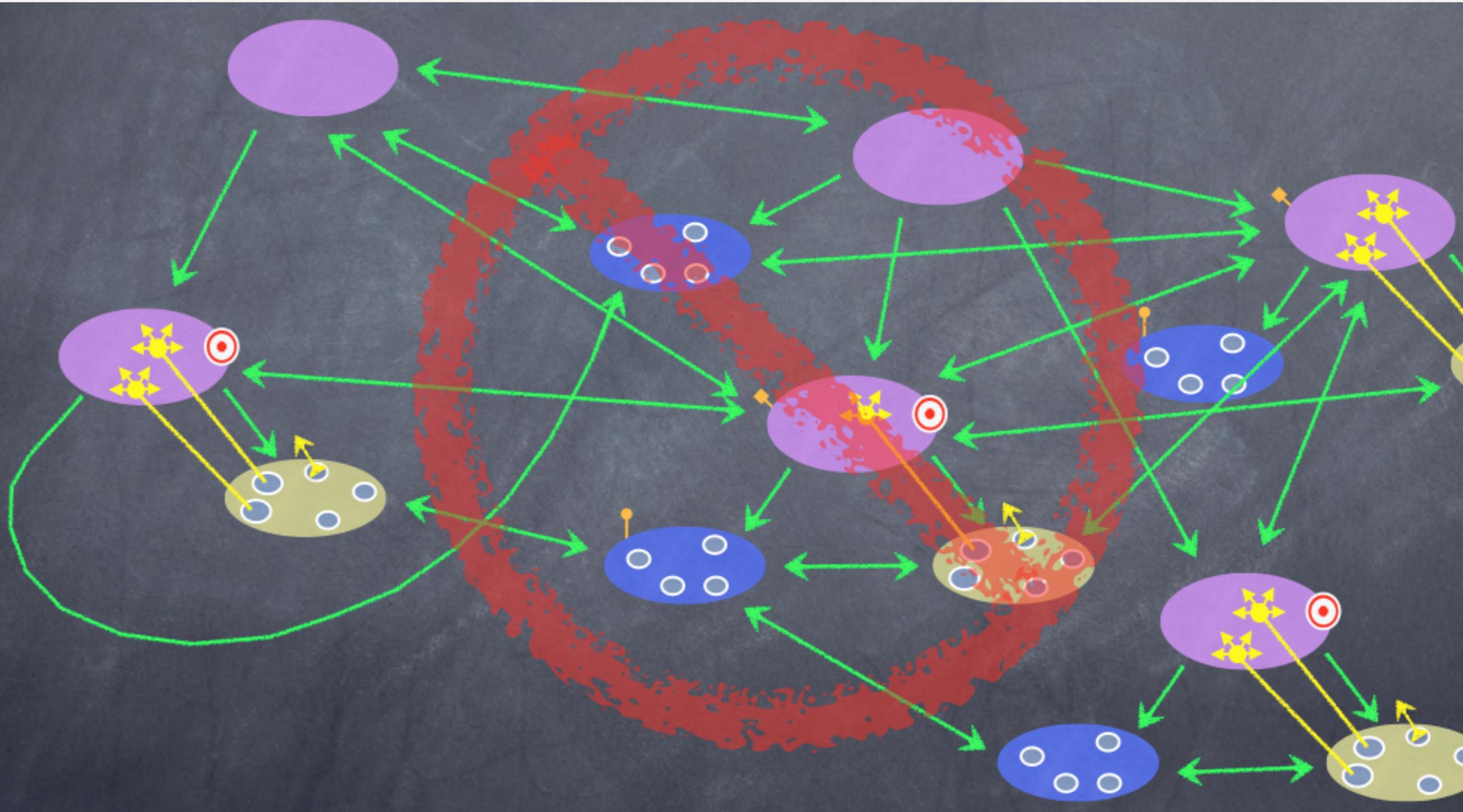


¿Puede el Modelo comunicarse con el Controller?

MVC



Indirectamente, si se podría



Uso de las reglas del patrón MVC

