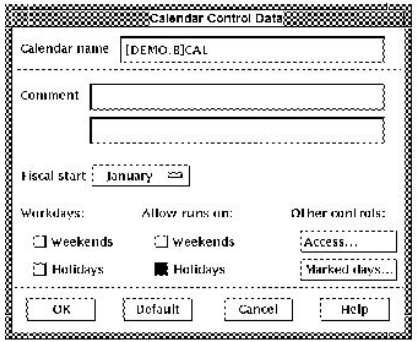
INVERSIÓN DE CONTROL E INYECCIÓN DE DEPENDENCIAS

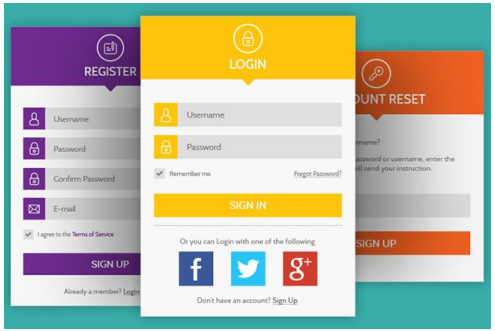
1.

INVERSIÓN DE CONTROL

INVERSIÓN DE CONTROL (IoC)

- Principio de diseño (o patrón)
- El objetivo es conseguir *desacoplar* objetos.





“

No nos llames.

Nosotros te llamaremos a tí.

Principio de Hollywood

INVERSIÓN DE CONTROL (IoC)

- Martin Fowler
- *Dejar que sea otro el que controle el flujo del programa* (por ejemplo, un framework)

```
#ruby
puts 'What is your name?'
name = gets
process_name(name)
puts 'What is your quest?'
quest = gets
process_quest(quest)
```

```
require 'tk'
root = TkRoot.new()
name_label = TkLabel.new() {text "What is Your Name?"}
name_label.pack
name = TkEntry.new(root).pack
name.bind("FocusOut") {process_name(name)}
quest_label = TkLabel.new() {text "What is Your Quest?"}
quest_label.pack
quest = TkEntry.new(root).pack
quest.bind("FocusOut") {process_quest(quest)}
Tk.mainloop()
```

Ejemplos propuestos por Martin Fowler

Ralph Johnson and Brian Foote

Journal of Object-Oriented Programming

Junio/Julio 1988

*Una característica importante de un **framework** es que los **métodos definidos por el usuario** para adaptar el mismo a menudo **serán llamados desde el framework**, en lugar de desde el código de aplicación del usuario. El framework a veces **desempeña el papel de programa principal** en la coordinación y secuenciación de actividad de la aplicación. Esta **inversión de control** proporciona al framework la posibilidad de servir como un **esqueleto extensible**. El usuario proporciona métodos que adaptan los algoritmos genéricos.*

ALGUNOS EJEMPLOS DE INVERSIÓN DE CONTROL

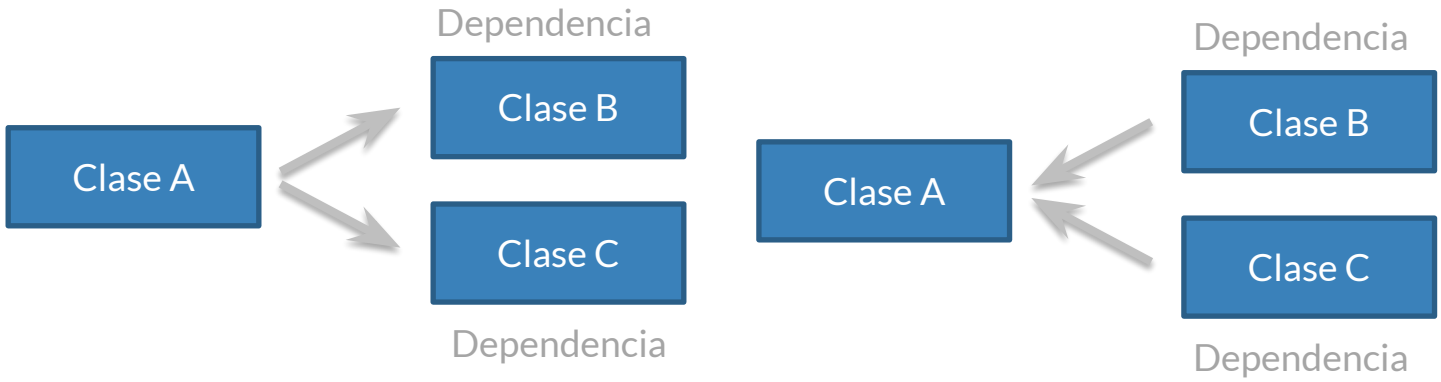
- Suscripción o manejo de eventos (.NET, Java, ...)
- Session Bean (EJB): `ejbRemove`, `ejbPassivate`, `ejbActivate`, ...
- JUnit: `setUp`, `tearDown`, ...
- Inyección de dependencias: es solo una forma de inversión de control.
- ....

2.

INYECCIÓN DE DEPENDENCIAS

INYECCIÓN DE DEPENDENCIAS

- Es una forma de inversión de control.



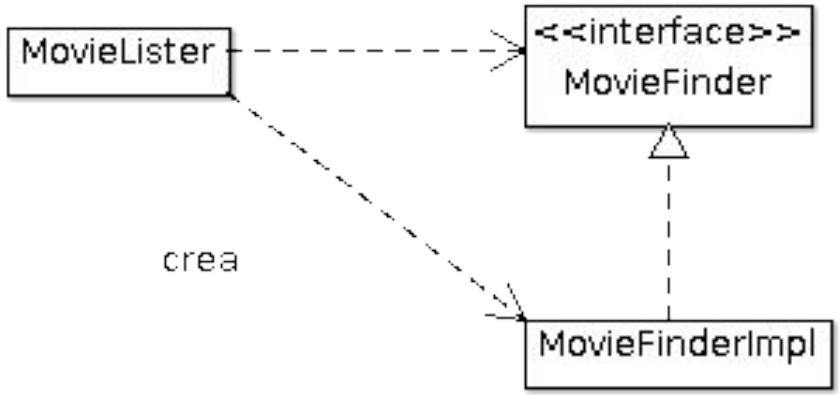
MOTIVACIÓN PARA EL USO DE INYECCIÓN DE DEPENDENCIAS

```
public class MovieLister {
  public Movie[] moviesDirectedBy(String arg)
  {
      List<Movie> allMovies = finder.findAll();

      for (Iterator it = allMovies.iterator(); it.hasNext();)
      {
        Movie movie = (Movie) it.next();
        if (!movie.getDirector().equals(arg)) it.remove();
      }

      return (Movie[]) allMovies.toArray(new
                                        Movie[allMovies.size()])
  }
}
```

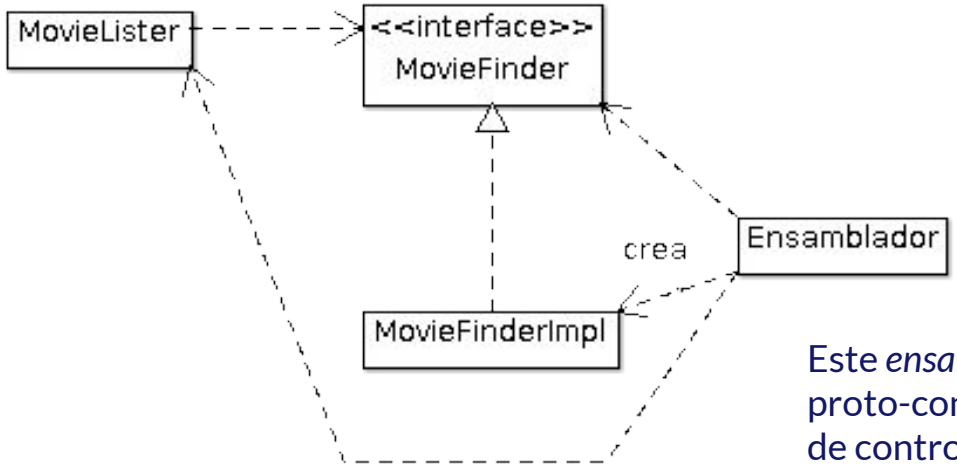
MOTIVACIÓN PARA EL USO DE INYECCIÓN DE DEPENDENCIAS

```
public interface MovieFinder
{
   List<Movie> findAll();
}

public class MovieLister {

  private MovieFinder finder;

  public MovieLister() {
    finder = new CSVMovieFinder("movies.txt");
  }
  //...
}
```

MOTIVACIÓN PARA EL USO DE INYECCIÓN DE DEPENDENCIAS



MOTIVACIÓN PARA EL USO DE INYECCIÓN DE DEPENDENCIAS



Este *ensamblador* es nuestro proto-contenedor de inversión de control.

EJEMPLO DE INYECCIÓN DE DEPENDENCIAS CON SPRING

```
class MovieLister {
    private MovieFinder finder;

    public void setFinder(MovieFinder finder) {
       this.finder = finder;
    }
}

class CSVMovieFinder implements MovieFinder {

    public void setFilename(String filename) {
       this.filename = filename;
    }
}
```

EJEMPLO DE INYECCIÓN DE DEPENDENCIAS CON SPRING

```
<beans>
    <bean id="MovieLister" class="spring.MovieLister">
        <property name="finder">
            <ref local="MovieFinder"/>
        </property>
    </bean>
    <bean id="MovieFinder" class="spring.CSVMovieFinder">
        <property name="filename">
            <value>movies.txt</value>
        </property>
    </bean>
</beans>
```