Fowler’s Movie Example

Find all movies by a given director

class MovieLister {
    private ColonDelimitedMovieFinder finder =
        new ColonDelimitedMovieFinder("movies1.txt");

    public Movie[] moviesDirectedBy(String arg) {
        List 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()]);
    }
}
MovieLister now depends on (uses) a particular low level service

What if we need to use a different low level service?

class MovieLister {
    private ColonDelimitedMovieFinder finder =
        new ColonDelimitedMovieFinder("movies1.txt");

    public Movie[] moviesDirectedBy(String arg) {
        List 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()]);
    }
}
Low level objects are building blocks for the applications
   Read files
   Interact with database
   Display data on screen
   Easy to reuse elsewhere

High level objects contain the business logic
   Main purpose of the application
   Hard to reuse elsewhere due to dependencies on low level details
public interface MovieFinder {
    List findAll();
}

Program to an Interface
With Factory Method

For each concrete finder need:
Concrete finder class
Subclass of MovieLister

class MovieLister {
    private MovieFinder finder;

    public MovieLister() {
        finder = createFinder();
    }

    public MovieFinder createFinder() {
        new ColonDelimitedMovieFinder("movies1.txt");
    }

    public Movie[] moviesDirectedBy(String arg) {
        // Same as before
    }
}
With Constructor

class MovieLister {
    private MovieFinder finder;

    public MovieLister(MovieFinder finder) {
        this.finder = finder;
    }

    public Movie[] moviesDirectedBy(String arg) {
        // Same as before
    }
}

class ColonDelimitedMovieFinder implements MovieFinder {
    private String filename;

    ColonDelimitedMovieFinder(String filename) { this.filename = filename;}

    public List findAll() {...}
}
public class Injector {
    public static void main(String[] args) {
        MovieFinder finder = new ColonDelimitedMovieFinder("movies1.txt");
        MovieLister lister = new MovieLister(finder);
        lister.moviesDirectedBy("Spielberg");
    }
}
So we replace

    MovieLister lister = new MovieLister();
lister.moviesDirectedBy("Spielberg");

With

    MovieFinder finder = new ColonDelimitedMovieFinder("movies1.txt");
    MovieLister lister = new MovieLister(finder);
lister.moviesDirectedBy("Spielberg");
Problems with Manual Injection

Scaling is hard

- Same dependency is needed in multiple places
- Multiple different dependencies in multiple places

Program is still dependent on the dependencies
Plugin Pattern

Links classes during configuration rather than compilation

Code runs in multiple runtime environments

Each environment requires different implementations of particular service

Plugin provides centralized runtime configuration
Plugin Pattern - How it works

Separated Interface

Define an interface in a separate package from its implementation

Program needs the interface at compile time

Program will load the implementation at runtime
Plugin Pattern - How it works

Plugin uses a factory to create the service

Plugin reads file to determine which implementation of service to create

With Reflection (Java)
  Plugin reads the class of the needed service from file
  Plugin factory creates instance of service class
  Plugin source code does not have reference class of the service

Without Reflection
  Plugin reads which service is needed from file
  Plugin factory uses conditional logic to create service instance
  Plugin source code needs to reference class of all service implementations
Dependency Injection & Plugin Pattern

Use the plugin pattern to provide
Central location to handle dependency injection
Configure the application from external data at runtime

Injector - add services to client
Also known as:
  assembler
  provider
  container
  factory
  builder
  spring
  construction code
Type of Dependency Injection

Constructor

Setter

Interface
Constructor Injection with PicoContainer

class MovieLister {
    public MovieLister(MovieFinder finder) { this.finder = finder;}
}

class ColonDelimitedMovieFinder implements MovieFinder {
    ColonDelimitedMovieFinder(String filename) { this.filename = filename;}
}

private MutablePicoContainer configureContainer() {
    MutablePicoContainer pico = new DefaultPicoContainer();
    Parameter[] finderParams = {new ConstantParameter("movies1.txt")};
    pico.registerComponentImplementation(MovieFinder.class, ColonMovieFinder.class, finderParams);
    pico.registerComponentImplementation(MovieLister.class);
    return pico;
}
pico.registerComponentImplementation(MovieFinder.class, ColonMovieFinder.class, finderParams);

When you need a MovieFinder instance return an instance of ColonMovieFinder

Use finderParams as argument for ColonMovieFinder constructor

Reflection is used to do this

pico.registerComponentImplementation(MovieLister.class);

Container can now create MovieLister instance

Its constructor needs a MovieFinder object,
Container already knows how to create a MovieFinder object
public void testWithPico() {
    MutablePicoContainer pico = configureContainer();
    MovieLister lister = (MovieLister) pico.getComponentInstance(MovieLister.class);
    Movie[] movies = lister.moviesDirectedBy("Sergio Leone");
    assertEquals("Once Upon a Time in the West", movies[0].getTitle());
}
MovieFinder finder = new ColonDelimitedMovieFinder("movies1.txt");
MovieLister lister = new MovieLister(finder);
lister.moviesDirectedBy("Spielberg");

private MutablePicoContainer configureContainer() {
    MutablePicoContainer pico = new DefaultPicoContainer();
    Parameter[] finderParams = {new ConstantParameter("movies1.txt")};
    pico.registerComponentImplementation(MovieFinder.class, ColonMovieFinder.class,
        finderParams);
    pico.registerComponentImplementation(MovieLister.class);
    return pico;
}

public void testWithPico() {
    MutablePicoContainer pico = configureContainer();
    MovieLister lister = (MovieLister) pico.getComponentInstance(MovieLister.class);
    Movie[] movies = lister.moviesDirectedBy("Sergio Leone");
    assertEquals("Once Upon a Time in the West", movies[0].getTitle());
}
How to configure from a file?

Class.forName("edu.sdsu.cs.whitney.BinarySearchTree")

Converts a string to the Class represented by the string
Setter Injection with Spring

Each class needs a setter method

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

class ColonMovieFinder...
    public void setFilename(String filename) {
        this.filename = filename;
    }
<beans>
    <bean id="MovieLister" class="spring.MovieLister">
        <property name="finder">
            <ref local="MovieFinder"/>
        </property>
    </bean>
    <bean id="MovieFinder" class="spring.ColonMovieFinder">
        <property name="filename">
            <value>movies1.txt</value>
        </property>
    </bean>
</beans>
Using the injector

ApplicationContext ctx = new FileSystemXmlApplicationContext("spring.xml");
MovieLister lister = (MovieLister) ctx.getBean("MovieLister");
Movie[] movies = lister.moviesDirectedBy("Sergio Leone");
assertEquals("Once Upon a Time in the West", movies[0].getTitle());
Interface Injection

Define an interface for doing the injection

```java
public interface InjectFinder {
    void injectFinder(MovieFinder finder);
}
```

```java
class MovieLister implements InjectFinder {
    public void injectFinder(MovieFinder finder) {
        this.finder = finder;
    }
}
```

The injector can be anything

Framework uses the interface to find & use the injector
Service Locator

Object that knows how to get all the services that an application needs
class MovieLister...
    MovieFinder finder = ServiceLocator.movieFinder();

class ServiceLocator...
    public static MovieFinder movieFinder() {
        return soleInstance.movieFinder;
    }
    private static ServiceLocator soleInstance;
    private MovieFinder movieFinder;
How to configure the service locator?

In code
From file
Service Locator vs Dependency Injection

Clients are dependent on Service Locator

Dependency Injection makes it easier to see component dependencies

If building an application dependency on Service Locator is ok

If providing component for others to use Dependency Injection is easier
SOLID
OO Design Principle by Robert Martin

**Single Responsibility Principle**

**Open Closed Principle**

**Liskov Substitution Principle**

**Interface Segregation Principle**

**Dependency Inversion Principle**

Single Responsibility Principle (SRP)

A class should have only one reason to change

Responsibility -> Reason to change

Simplest principle

Hardest to get right
public interface Modem {
    public void dial(String phoneNumber);
    public void hangup();
    public void send(char c);
    public char receive()
}

Two responsibilities
Connection management
Data communication

If need to change signature of connection functions then classes that call send and receive will have to be recompiled more often than needed

If app not changing in ways that cause the two responsibilities to change at different times then no need to separate them
An axis of change is only an axis of change if the changes actually occur
Interface Data Channel

- send(char)
- recv(): char

Interface Connection

- dial(String)
- hangup()

Modem Implementation
Separating Coupled Responsibilities

He kept both responsibilities in ModemImplementation class

Not desirable but may be necessary

By separating the interfaces we have decoupled them as far as the app is concerned

Nobody but main need to know it exists
The Open Closed Principle

You should be able to extend a classes behavior, without modifying it.

No significant program can be 100% closed

Designer must choose the kinds of changes against which to close the design
Liskov Substitution Principle

Child classes must be substitutable for their parent classes

Rectangle a = new Square();
class Rectangle {
    double width;
    double height;

    public double width()    {return width; }
    public double height()   {return height; }
    public void width(double w)    {width = w; }
    public void height(double h)   {height = h; }
    public double area()               {return height * width; }
}

public Square extends Rectangle {
    public void width(double w)    {
        super.width(w);
        super.height(w);
    }
    public void height(double h)   {
        super.width(h);
        super.height(h);
    }
}

public void foo(Rectangle r) {
    r.width(5);
    r.height(2);
    assert( r.area() == 10);
}
What Went Wrong?

Behavior of a square is not the same as the behavior of a rectangle

Behavior is what software is about

The ISA relationship pertains to behavior

View a design in terms of the reasonable assumptions made by users
Interface Segregation Principle

Make fine grained interfaces that are client specific

Diagram:
- **Interface Data Channel**
  - send(char)
  - recv(): char
- **Interface Connection**
  - dial(String)
  - hangup()

**Modem Implementation**
Bad Design

Rigidity
   Every change affects too many parts of the system

Fragility
   When you make a change, unexpected parts of the system break

Immobility
   Hard to reuse in another application because it can’t be disentangled from the current application
Causes of Bad Design

Interdependence of the modules
Dependency Inversion Principle

High level modules should not depend upon low level modules. Both should depend upon abstractions.

Abstractions should not depend upon details. Details should depend upon abstractions.
Violation

MovieLister depends on ColonDelimitedMovieFinder

class MovieLister {
    private ColonDelimitedMovieFinder finder =
        new ColonDelimitedMovieFinder("movies1.txt");

    public Movie[] moviesDirectedBy(String arg) {
        List 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()]);
    }
}
public interface MovieFinder {
    List findAll();
}

Program to an Interface
Copy Program

A simple example may help to explain the ideas in this section. Consider a simple program that is charged with copying characters typed on a keyboard to a printer. Assume, furthermore, that the implementation platform does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence.

A simple example may help to explain the ideas in this section. Consider a simple program that is charged with copying characters typed on a keyboard to a printer. Assume, furthermore, that the implementation platform does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence.

A simple example may help to explain the ideas in this section. Consider a simple program that is charged with copying characters typed on a keyboard to a printer. Assume, furthermore, that the implementation platform does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence.

A simple example may help to explain the ideas in this section. Consider a simple program that is charged with copying characters typed on a keyboard to a printer. Assume, furthermore, that the implementation platform does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence.

A simple example may help to explain the ideas in this section. Consider a simple program that is charged with copying characters typed on a keyboard to a printer. Assume, furthermore, that the implementation platform does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence. We might conceive of a structure for this program that does not have an operating system that supports device independence.