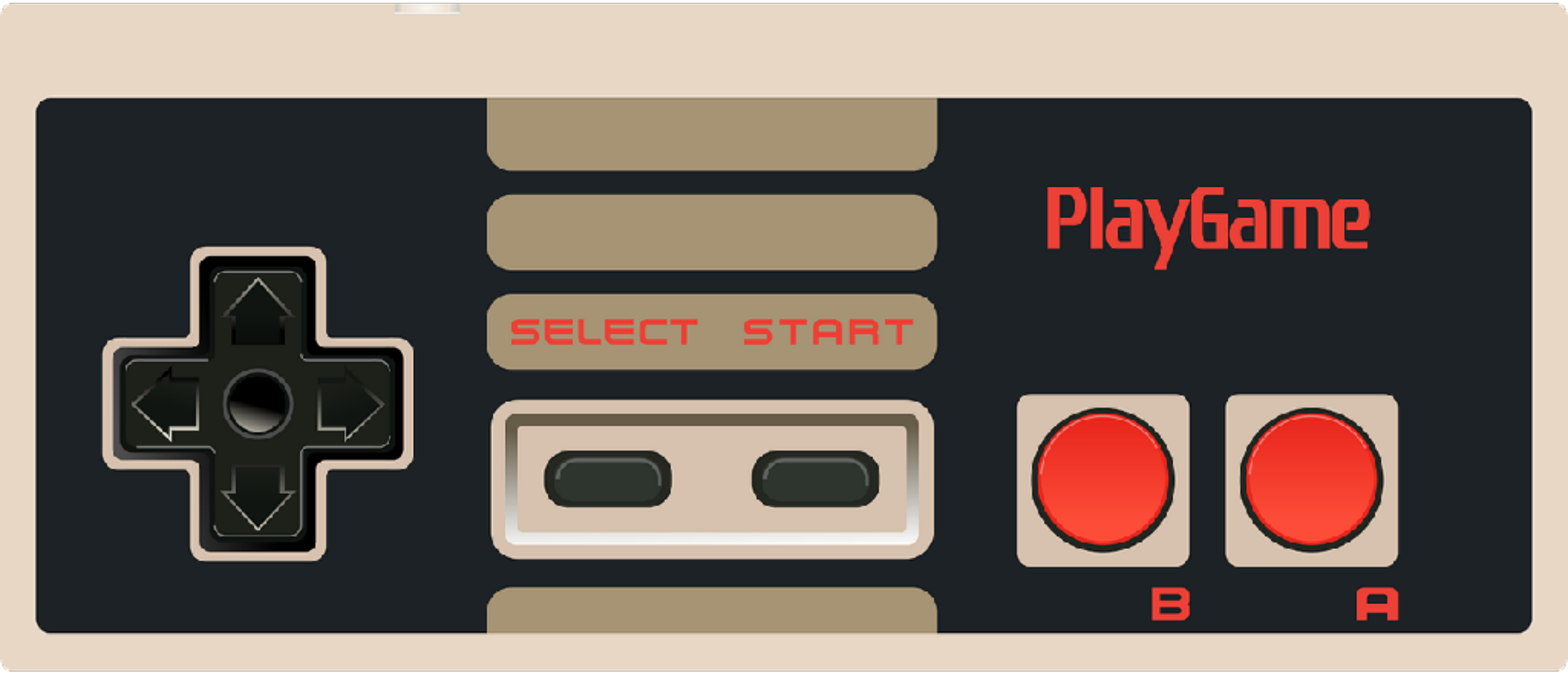
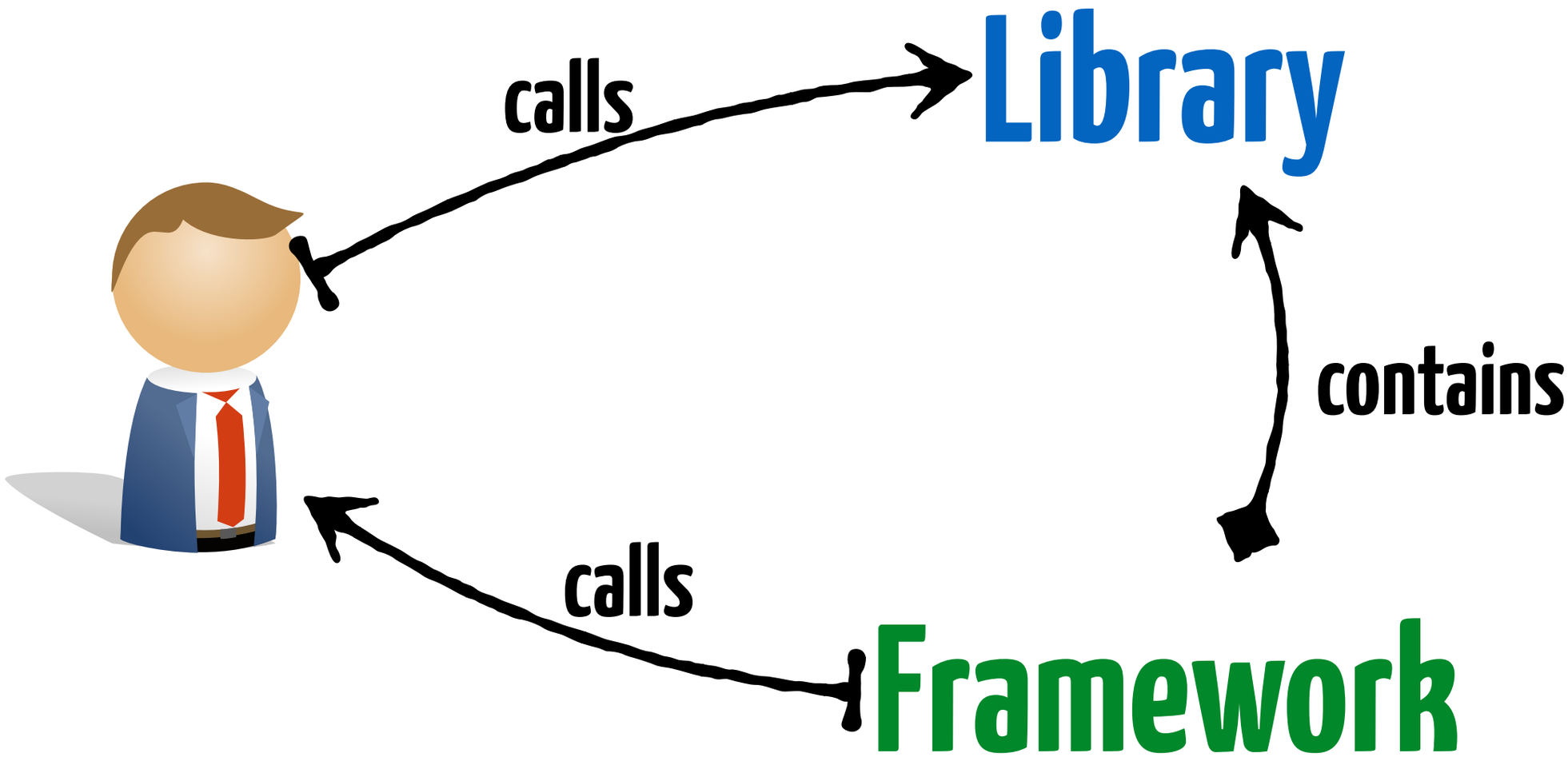
Inversion of Control



Library versus Framework?



Inversion of Control

Your **code** reuses a **library**

A **framework** reuses your **code**

Dependency Injection

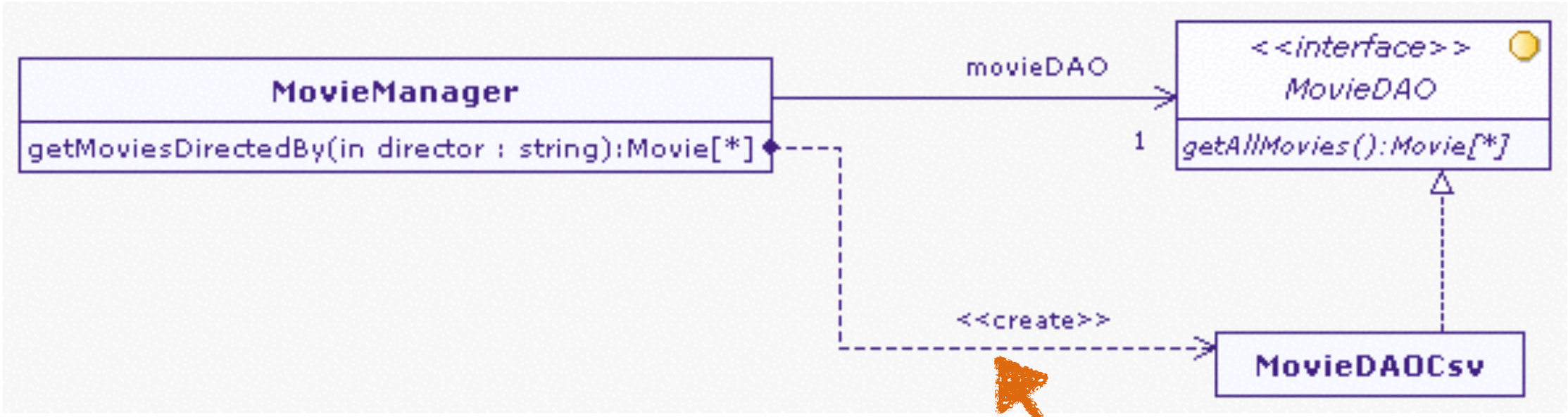
Illustration

Based in part on Martin Fowler's reference article on dependency injection


```java
public class MovieManager {

    private MovieDAOCsv movieDAOCsv;

    public MovieManager() {
        movieDAOCsv = new MovieDAOCsv("mymovies.txt");
    }

    public List<Movie> getMoviesDirectedBy(String director) {
        List<Movie> allMovies = movieDAOCsv.getAllMovies();
        // ...
```

strong coupling

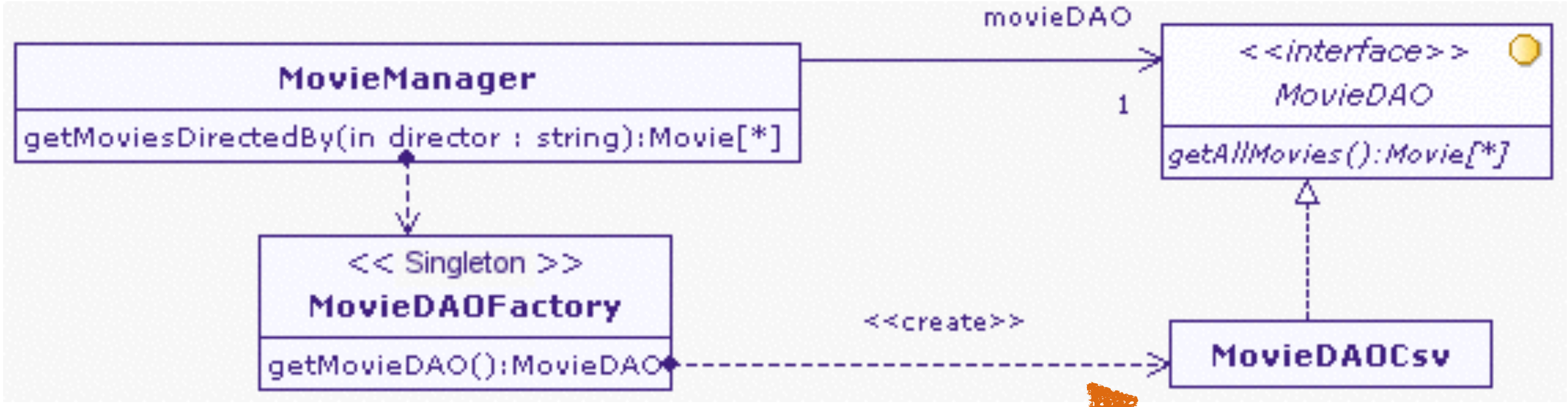
Using an interface




```java
public class MovieManager {

    private MovieDAO movieDAO;

    public MovieManager() {
        movieDAO = new MovieDAOCsv("mymovies.txt");
    }


    public List<Movie> getMoviesDirectedBy(String director) {
        List<Movie> allMovies = movieDAO.getAllMovies();
        // ...
```

weaker coupling

Using a factory




```
public class MovieManager {

    private MovieDAO movieDAO;

    public MovieManager() {
        movieDAO = MovieDAOFactory.getInstance().getMovieDAO();
    }

    public List<Movie> getMoviesDirectedBy(String director) {
        List<Movie> allMovies = movieDAO.getAllMovies();
```

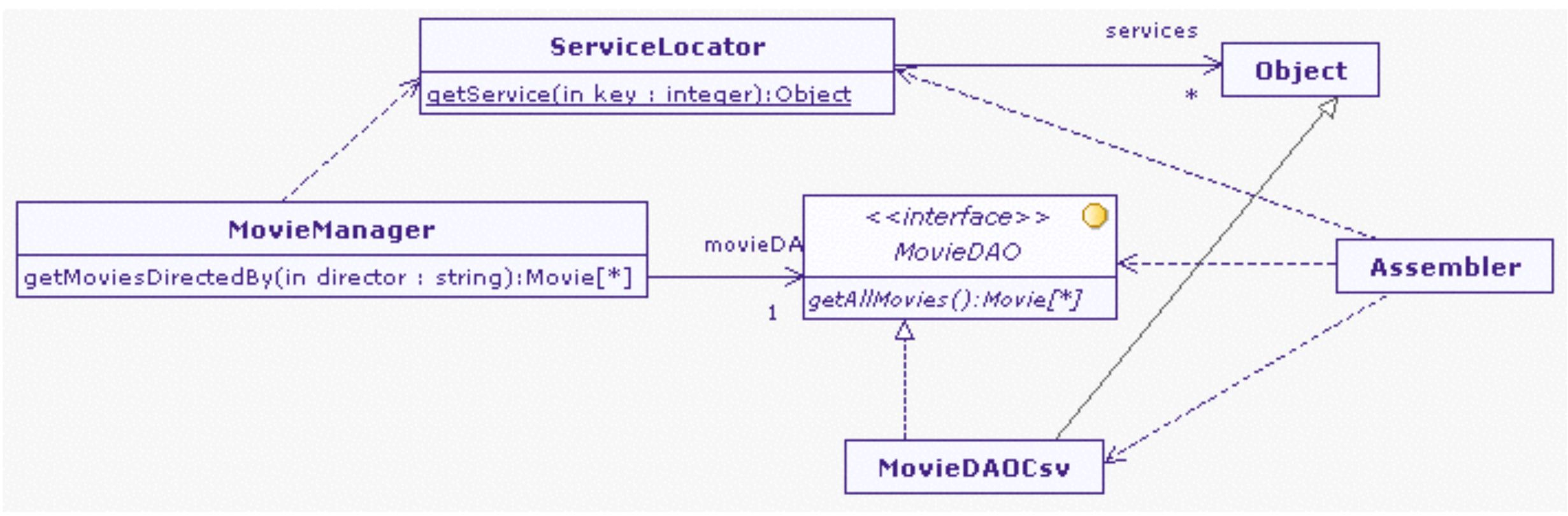
decoupling

But the factory is a class in our application...

Looking for dependencies

An external object is responsible for creating instances and add them to a list of services

These services are managed by a provider by associating them with key (string)



Looking for dependencies

```
public class MovieManager {

    private MovieDAO movieDAO;

    public MovieManager() {
        movieDAO = (MovieDAO) ServiceLocator.getService("movieDao");
    }
```

Only the interface declaration

Lookup

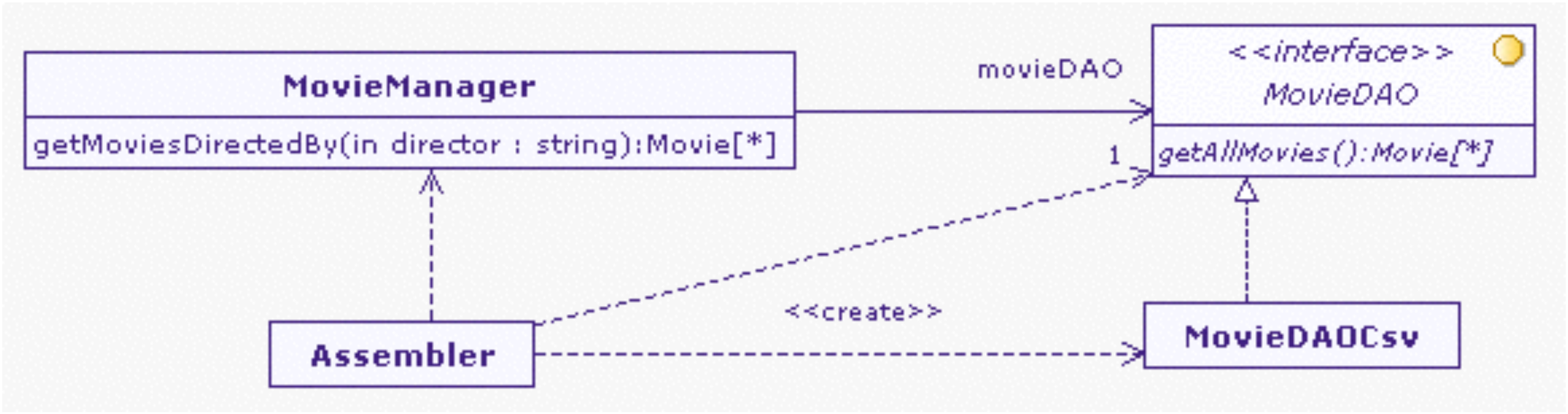
ServiceLocators is known as an Architectural Pattern

Dependency Injection

An external object is responsible for creating the application

It creates instances

It injects them in the classes it uses



Dependency Injection

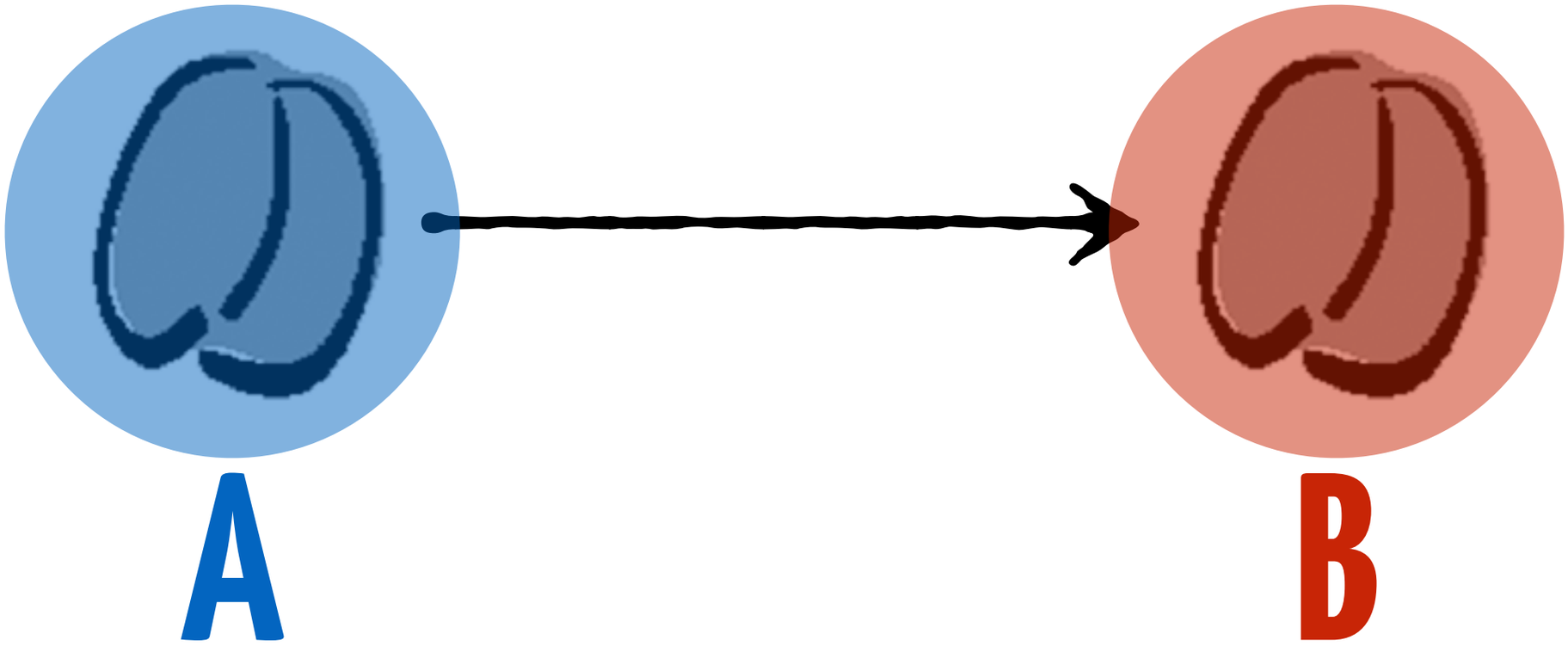
```
public class MovieManager {

    private MovieDAO movieDAO;

    public MovieManager() {
    }

    public void setMovieDAO(MovieDAO movieDAO) {
        this.movieDAO = movieDAO;
    }
```

Only the interface declaration

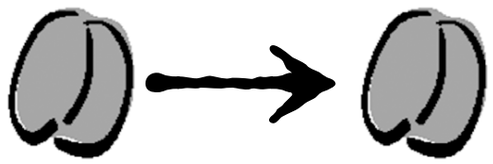
creates movieDAO

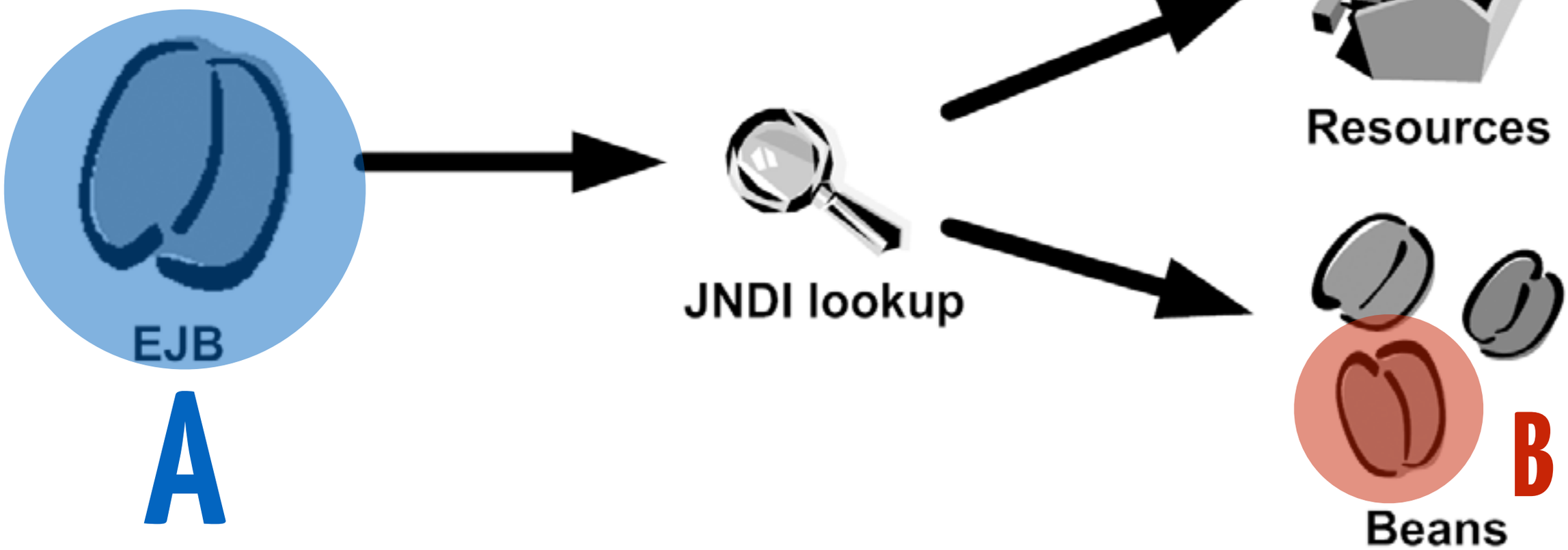
calls setMovieDAO

Problem: Bean Dependencies



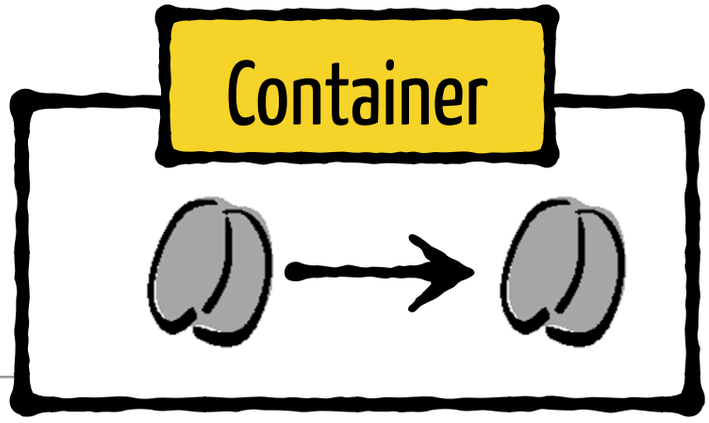
The good-old method: Lookup

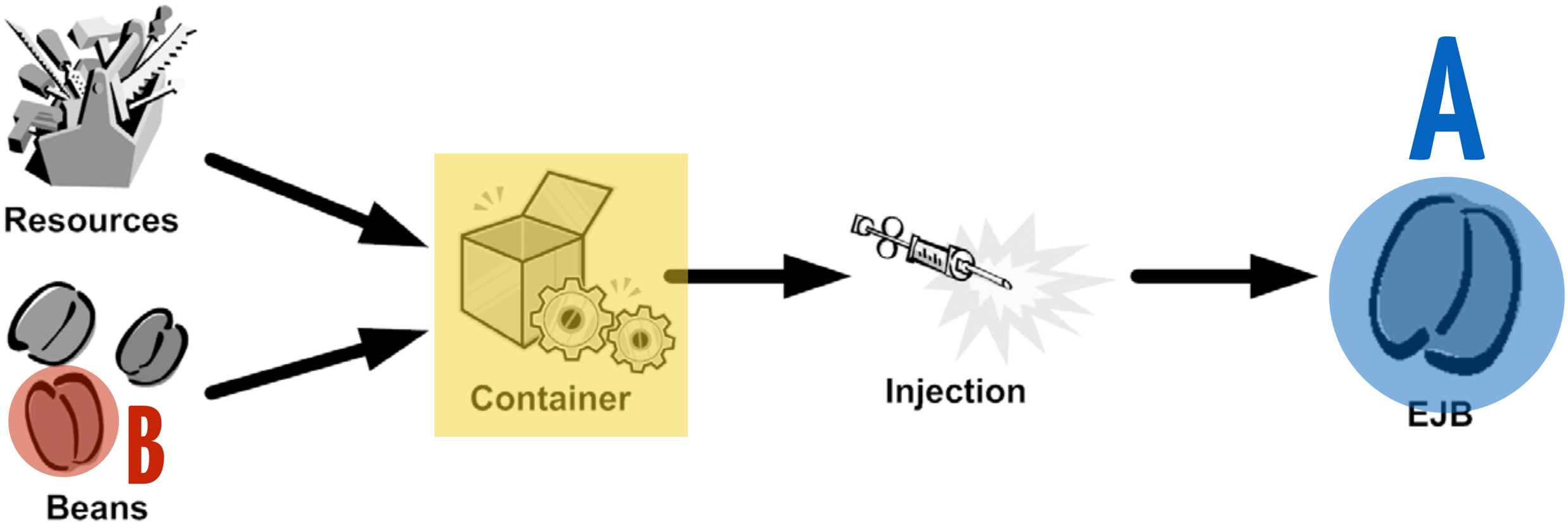





```
Object ejbHome = new InitialContext().lookup("java:comp/env/PlaceBid");
PlaceBidHome placeBidHome = (PlaceBidHome)
    PortableRemoteObject.narrow(ejbHome, PlaceBidHome.class);
PlaceBid placeBid = placeBidHome.create();
```

Steroids: Dependency Injection



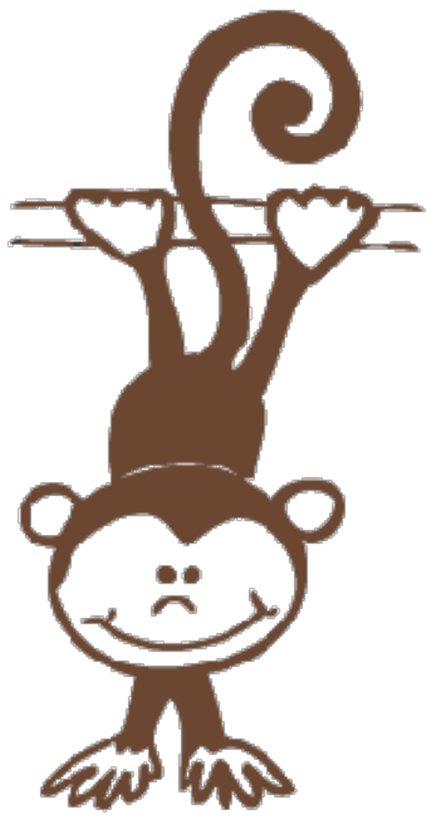



```
public class PlaceOrderTestClient {
    @EJB
    private static PlaceOrder placeOrder;
```

Injects an instance of EJB

https://github.com/collet/4A_ISA_TheCookieFactory

monkey see



monkey do