Dependency Injection

CSCI-4448 - Boese

University of Colorado **Boulder**

Overview

Goals

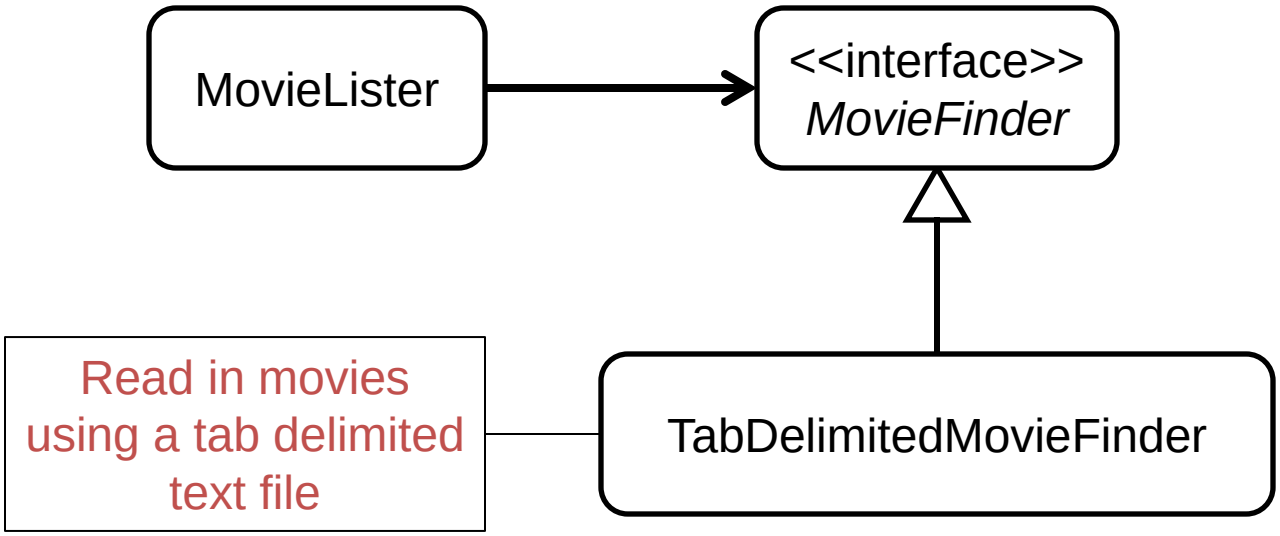
- Introduce dependency injection
- Provide examples using various Frameworks
  - Spring (Java)

Definition

- Dependency Injection
  - At its simplest, a *design pattern* which implements *Inversion of Control*
    - ***Decouples client code from concrete implementation***
    - ***Provide objects with dependencies at run-time***
  - A technique for assembling applications
    - ***From a set of concrete classes***
    - ***Concrete classes do not know about each other***
  - Goal: A loosely coupled system
    - ***References* wired *together using generic interfaces***
    - ***Ideally, allow a system to be* configured, *rather than compiled***

Example – Movie Lister

- Consider a simple movie listing class
  - ***MovieLister lists movies with certain characteristics***
  - ***Movies are loaded using some MovieFinder instance***

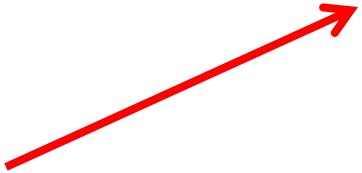


Example – Movie Lister

- Even in this simple example, want to avoid depending on concrete classes

```
public class MovieLister {
    private MovieFinder finder;

    public MovieLister() {
        this.finder = new TabDelimitedMovieFinder("movies.txt");
    }
}
```



Depencency on a Concrete Class

Depencency on Hard-Coded String

Example – Movie Lister

- The code has two concrete dependencies
  - A reference to a concrete MovieFinder
  - A reference to a hard-coded string
- Both references are brittle
  - Cannot change the filename of the list of movies without changing MovieLister and recompiling
  - Cannot change the format of the data without changing the concrete MovieFinder
    - ***e.g., use an SQLite database instead of flatfile***
    - ***Recompile MovieLister after resetting class***

Target

- For loose coupling to be achieved, need *no* concrete references

```
public class MovieLister {
    private MovieFinder finder;

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

- Furthermore, nowhere in our system's code should the strings "TabDelimitedMovieFinder" or "movies.txt" appear

Two types of dependency injection

- **Constructor Injection**

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

  - MovieLister indicates dependency via constructor

- **Setter Injection**

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

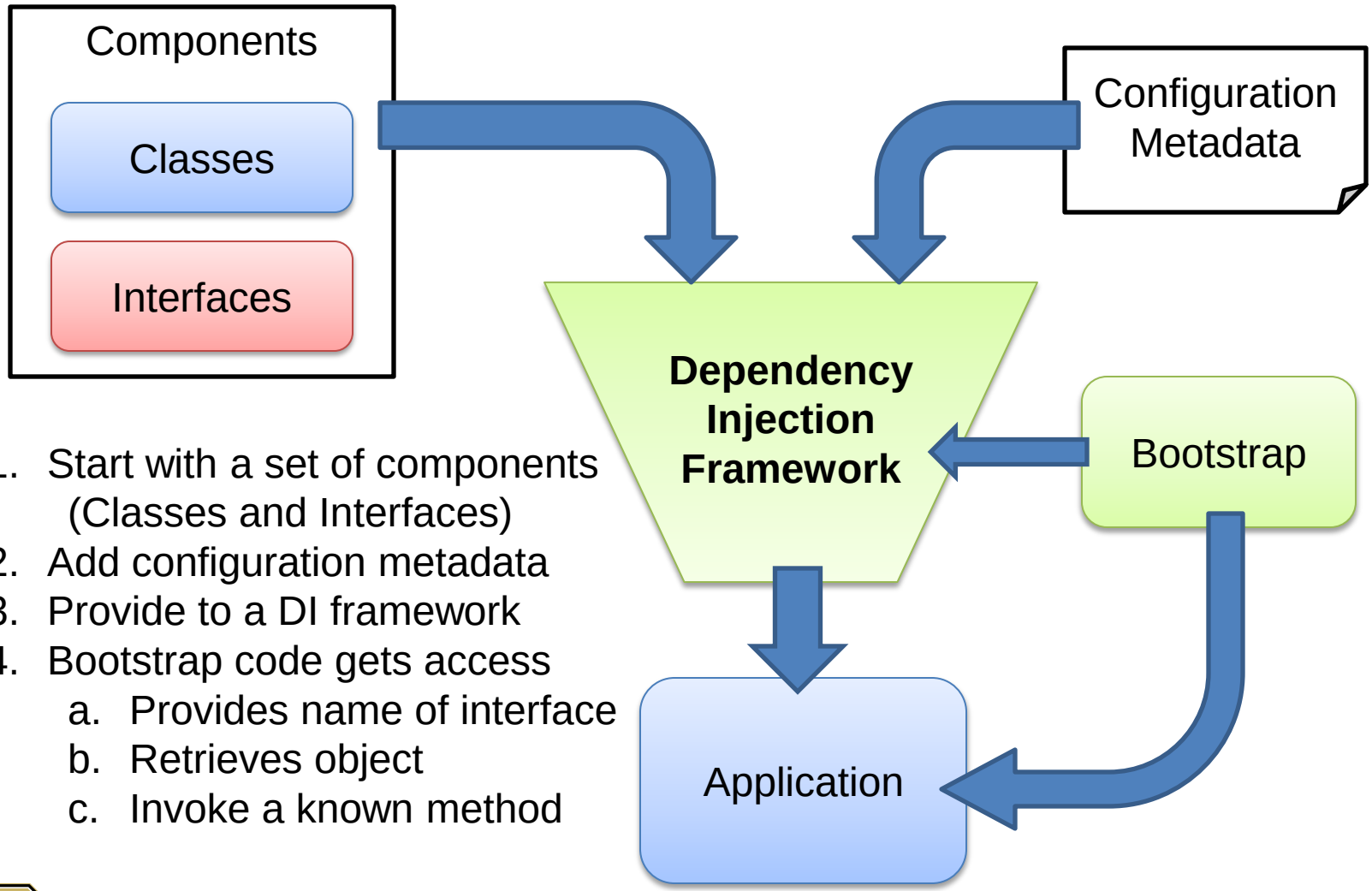
  - MovieLister indicates dependency via setter

Dependency Injection

- What is dependency injection?
  - The idea is that classes indicate dependencies in abstract ways
    - ***MovieLister indicates it NEEDS-A MovieFinder***
    - ***Main indicates it NEEDS-A MovieLister***
  - A third party injects a class that will meet that dependency at run-time
  - The “third party” is known as an “Inversion of Control container” or “dependency injection framework”

Dependency Injection Framework

Concept



1. Start with a set of components (Classes and Interfaces)
2. Add configuration metadata
3. Provide to a DI framework
4. Bootstrap code gets access
   a. Provides name of interface
   b. Retrieves object
   c. Invoke a known method

Example – Movie Lister with Spring

- The Spring framework can be used to create an instance of MovieLister

```
public class MyApp {
   public void runApp() throws Exception {
      ApplicationContext context =
            new FileSystemXmlApplicationContext("spring.xml");
      MovieLister lister =
            (MovieLister) context.getBean("MovieLister");
      Movie[] movies = listener.moviesDirectedBy("Ang Lee");
      …
   }
}
```

  - ***Note: no direct reference to either "movies.txt" or TabDelimitedMoveFinder***

Example – Movie Lister with Spring

```
public class MyApp {
   public void runApp() throws Exception {
      ApplicationContext context =
            new FileSystemXmlApplicationContext("spring.xml");
      MovieLister lister =
            (MovieLister) context.getBean("MovieLister");
      Movie[] movies = listener.moviesDirectedBy("Ang Lee");
      …
   }
}
```

- "spring.xml"
  - Contains metadata about the application
    - ***MovieLister needs a TabDelimitedMovieFinder***
    - ***Database is in a file called "movies.txt"***
  - These dependencies are inserted when `context.getBean()` is invoked

Example – Movie Lister with Spring

```
public class MyApp {
   public void runApp() throws Exception {
      ApplicationContext context =
            new FileSystemXmlApplicationContext("spring.xml");
      MovieLister lister =
            (MovieLister) context.getBean("MovieLister");
      Movie[] movies = listener.moviesDirectedBy("Ang Lee");
      …
   }
}
```

- In Spring, "Beans" = POJOs (plain old Java objects)
  - A Java class which follows certain conventions
    - ***Access property "foo" of type String with***
      - public String getFoo();
      - public void setFoo(String foo);
  - Once objects are specified in configuration file, pull instances of those objects out of the Spring container via the getBean method

Example

- Create an interface for MovieFinders

```
public interface MovieFinder {
   public Movie[] findWithDirector(String director);
}
```

- Create concrete implementations

```
public interface TabDelimitedMovieFinder {
   private String filename;
   public String getFilename() { return this.filename; }
   public void setFilename(String filename) {
      this.filename = filename;
      // Load the file as needed
   }
   public Movie[] findWithDirector(String director) { … }
}
```

Example

- Set up application configuration (“spring.xml”)

```
<?xml version="1.0" encoding="UTF-8" ?>

<beans xmlns=http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

  <bean id="movieFinder" class="TabDelimitedMovieFinder">
    <property name="filename" value="movies.txt">
  </bean>
</beans>
```

Which file contains the movies

Which MovieFinder class to use

Example – Movie Lister with Spring

- Create the application using Spring framework

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;

public class MyApp {
   public static void main(String[] args) {
      // Load up the Spring container from "spring.xml"
      ApplicationContext context =
            new FileSystemXmlApplicationContext("spring.xml");

      // Get an instance of a MovieFinder (as defined in spring.xml)
      MovieFinder finder =
            (MovieFinder) context.getBean("movieFinder");

      // Create the lister using our finder
      MovieLister lister = new MovieLister(finder);
      …
   }
}
```

Example – Change to SQLite database

- What if we wanted to change to an SQLite DB?
  - ***Create SQLiteMovieFinder class***
  - ***Change the Spring configuration XML file***

```
<?xml version="1.0" encoding="UTF-8" ?>

<beans xmlns=http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

  <bean id="movieFinder" class="SQLiteMovieFinder">
    <property name="databaseURI" value="file:data/movies.db?mode=ro">
  </bean>
</beans>
```

Which MovieFinder class to use

Property of SQLiteMovieFinder class

The URI of the database

Simple Example