Full Stack Development
Containers, Microservices and UI

7b. Spring Framework

Application Architecture

- A "bean" is
  - A class definition that has been placed under the control of the Spring Context
  - We mark a class with a "@Component" annotation to identify it as a bean
  - The bean class must implement a Java Interface

- The bean in implemented as:
  - An instance of the POJO
  - Wrapped inside a class that manages the POJO
  - The wrapper class is the bean and implements the same interface that the POJO class implements
  - This is referred to a "Aspect Oriented Programming"
  - The bean is an "interceptor" that intercepts calls to the POJO, does management operations, then forwards the call to the POJO and relays its response to the client
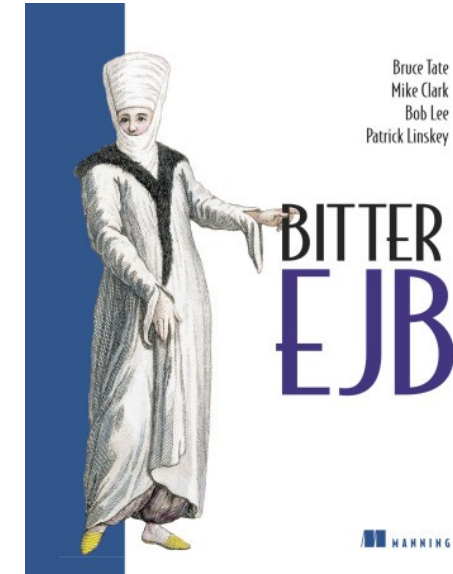
J2EE

- SUN tried to solve the problem by introducing an "enterprise architecture" called J2EE

  - It was based on application servers like Tomcat into which runtime containers were deployed

  - The containers contained a deployable form of a Java application called an Enterprise Java Bean (EJB)

  - These were wrappers around one of more POJOs to make them deployable onto an application server

  - Very popular due to the Web components like Java Server Pages and Servlets

- EJBs were a failure

  - They required a massive amount of configuration and boilerplate code

 - And they performed horribly

J2EE
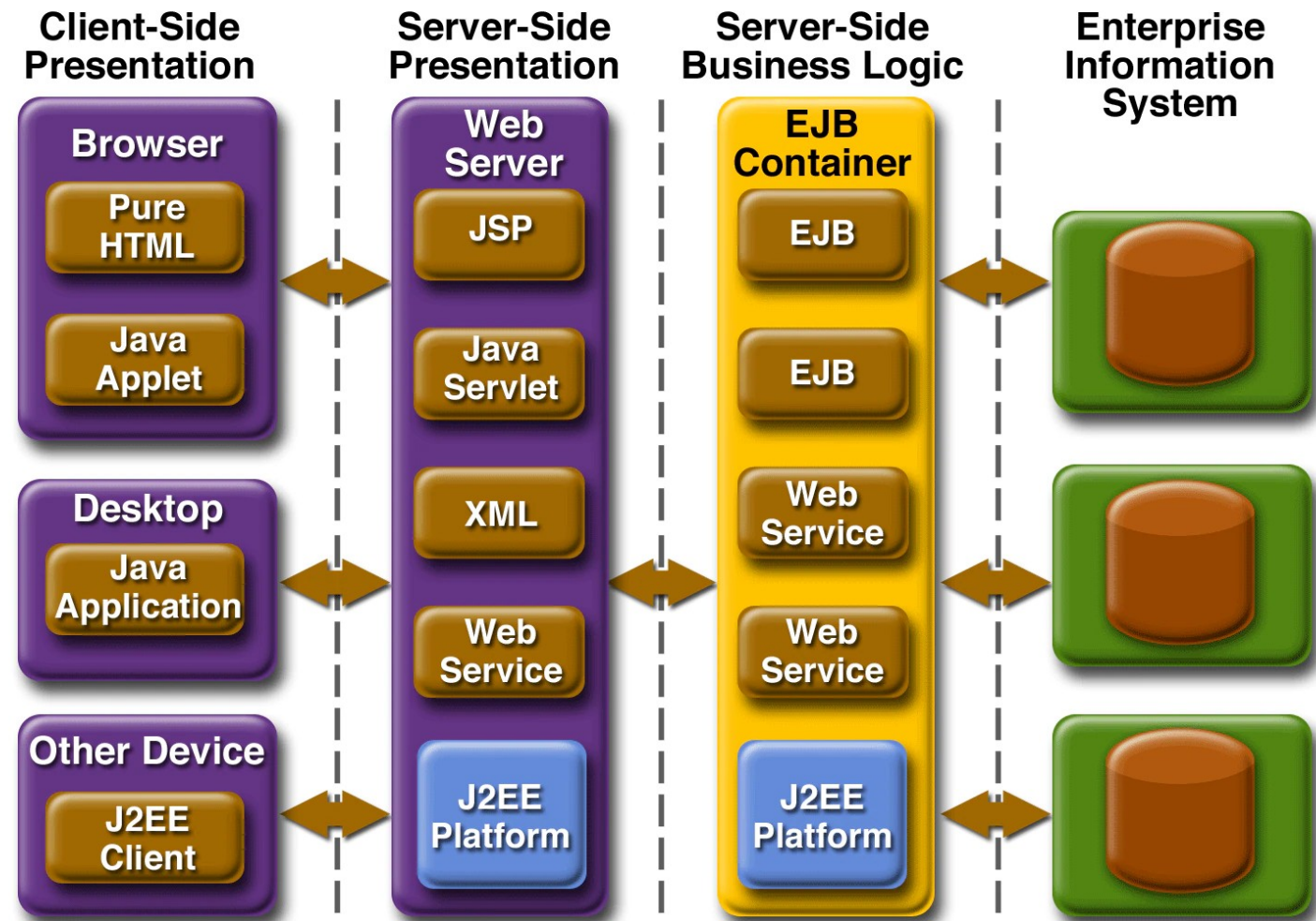
The Spring Insurgency

- J2EE was so problematic to use that Rod Johnson created the first version of Spring as an alternative

    - The first version was in 2004 when frustration with J2EE was at its peak

    - Some of the developers of EJBs wrote a book describing how to work aroundthe design problems of EJBs

- Spring became the go-to alternative to J2EE

The Spring Timeline
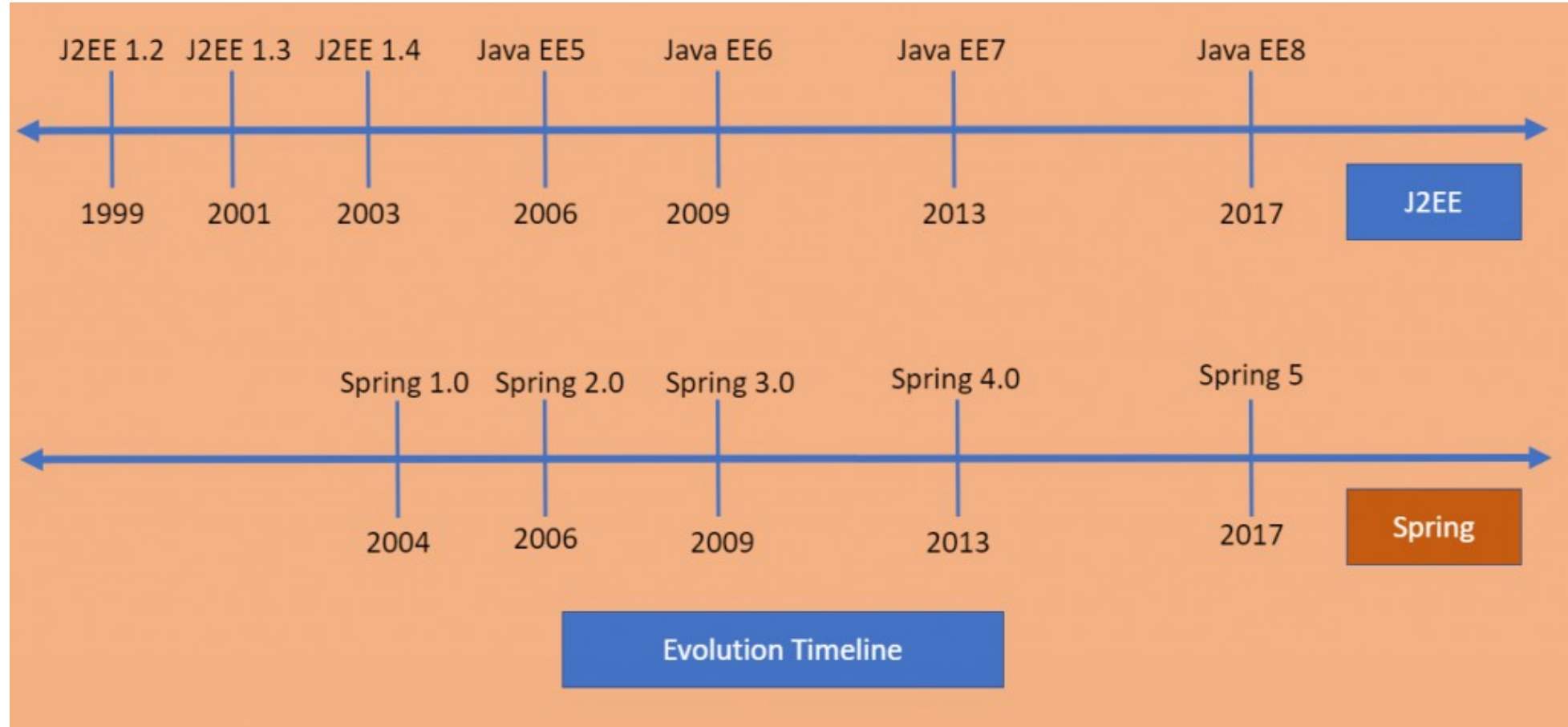
The Spring Impact

- After J2EE was rebranded as Java EE

  - Many of the features of Spring were adopted by Java EE

  - Especially the techniques of Inversion of Control (IoC) and Dependency Injection (DI)

- Spring has evolved into an ecosystem

  - Supports a number of projects that cover a wide range of development needs

  - Similar to the Apache project ecosystem

- This collection of projects is referred to as the Spring Platform

  - More info at http://spring.io

The Spring Platform
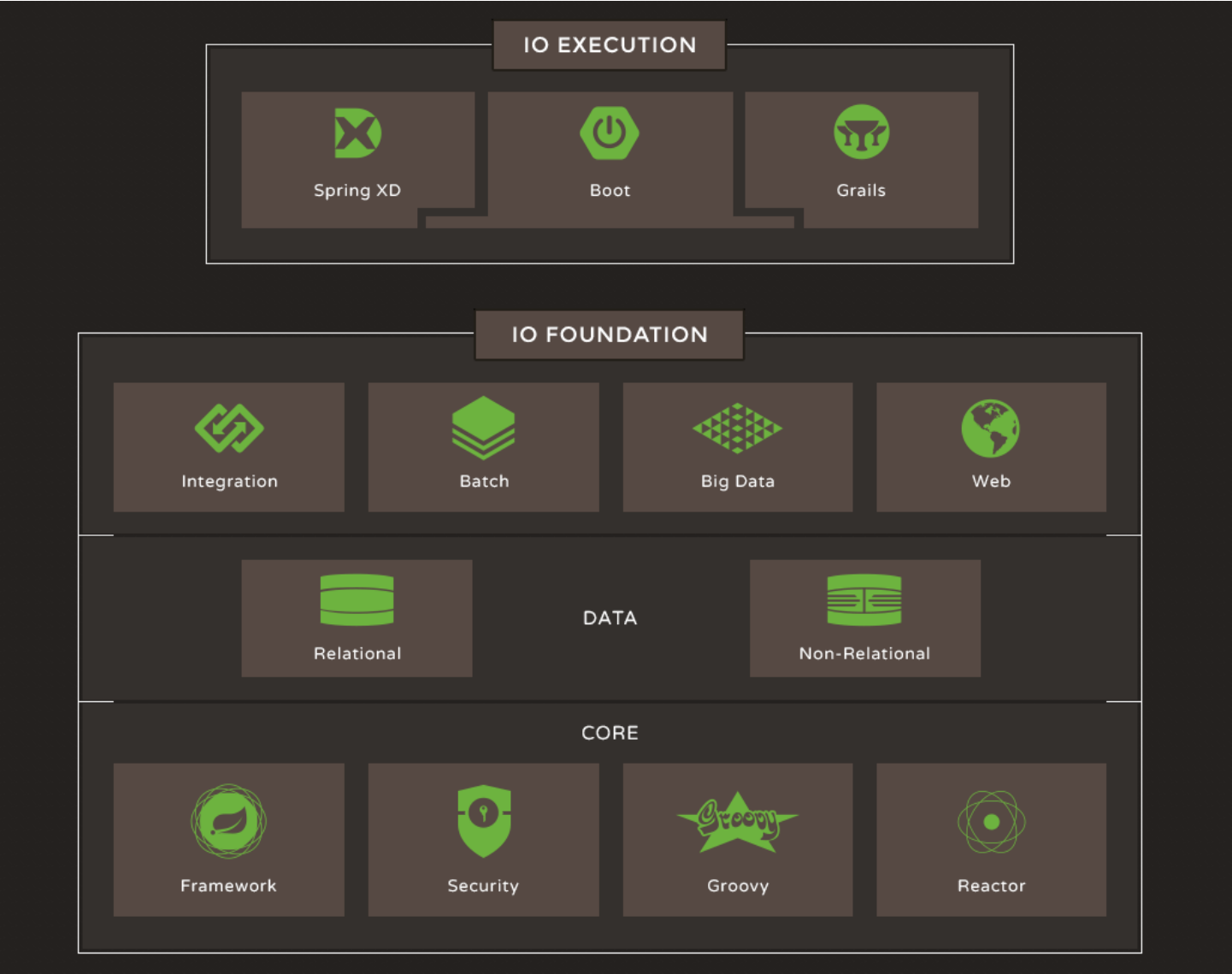
Inversion of Control

- Objects define their dependencies only through

    - Arguments to the object constructor

    - Arguments to a factory method

    - Properties set on an instance by a factory method

- Implementing the dependencies is done via DI

    - Commonly done in constructors or setter methods

    - Dependencies are "injected" during or after the instance is created

- IoC means that the instance controls the creation of its dependencies or connections to existing instances it depends on

Spring Container

- Responsible for:
  - Creating, configuring and wiring beans together
  - Metadata is read from an XML configuration file
  - Configuration file may direct the container to scan the code for additional metadata contained in Java annotations

- The container is made up of
  - A bean factory that manages the beans
  - A Spring context that provides additional functionality

- The context used depends on the application type
  - Application contexts are for stand alone applications
  - Web contexts are for web-based applications

Spring Container
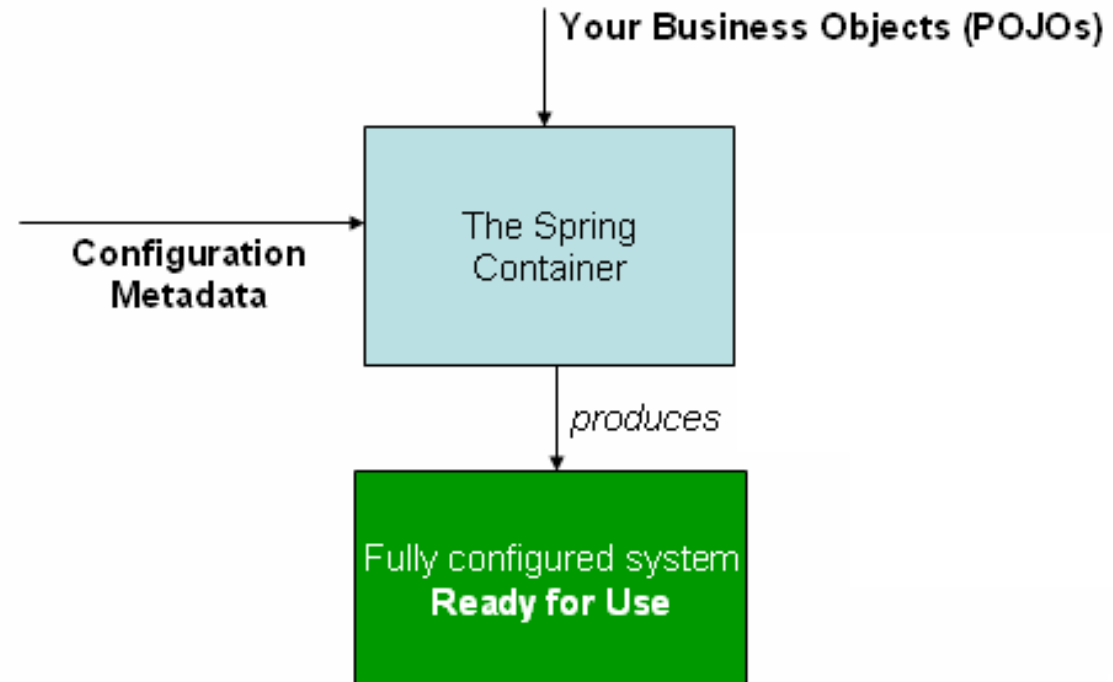
Creating a Container

```java
public static void main(String[] args) {
ClassPathXmlApplicationContext context =
            new ClassPathXmlApplicationContext("config.xml");
// working with beans
context.close();
```

Spring Beans

- A Spring bean is an instance of a POJO created the container that has been configured using the supplied metadata

- A "Bean Definition" consists of:
    - The class name for the POJO to be implemented
    - The scope and lifecycle of the bean
    - Reference to other beans needed to satisfy dependencies
    - Runtime settings like size limits

- Beans have a unique name
    - Can be defined in the metadata
    - Defaults to the name of the Java class

- Beans are a wrapper around a POJO
    - The bean implements the same interface as the POJO

Defining a Named Bean

- The code below defines a bean named "SpinDoctor" using the PRWhiz Java class.

```java
@Component("SpinDoctor")
public class PRWhiz implements Consultant {

    @Override
    public String getAdvice() {
        return "Don't let them see you sweat";
    }
}
```

Default Name for a Bean

- The code below defines a bean named "ITGuru" by using the Java class name by default

```java
@Component
public class ITGuru implements Consultant {

    @Override
    public String getAdvice() {
        return "Turn it off and on again";
    }
}
```

Interfaces

- A bean may be implemented by different Java classes

    - For example, different versions of the class

- In order to decouple the client logic from the container internals

    - Beans are referenced in the client code by a Java Interface reference

    - Every class that is used for making a bean must implement an interface known to the container

- This is consistent with programming best practices

- The bean wrapper is an interceptor

    - Takes client messages, does the management overhead

    - Forwards the message to the POJO and returns the result

Getting a Bean

- We get a reference to a bean by asking the container for a bean by referencing a specific name
    - It may provide a reference to an existing instance
    - It may create a new instance

```java
public static void main(String[] args) {
ClassPathXmlApplicationContext context =
            new ClassPathXmlApplicationContext("config.xml");


Consultant dilbert = context.getBean("ITGuru",Consultant.class);
Consultant ratbert =  context.getBean("SpinDoctor",Consultant.class);
```

Bean Lifecycle

- By default, a bean is a singleton

  - That means if we ask for a reference to a bean and an instance already exists, we get a reference to that bean

  - At any time, there exists only one copy of the bean

  - One reason for this is to avoid creating multiple copies of a resource during dependency resolution – use existing resources instead

- We can specify other lifecycles

  - If the "Prototype" scope is declared, ever request for a reference to a bean creates a new instance of the bean

  - The discussion of other aspects of lifecycle management is beyond the scope of the course

Bean Scope

- In this definition, the @Scope() annotation is used to ensure a new copy of the bean is created each time a bean is requested.

```java
@Component
@Scope("prototype")
public class ITGuru implements Consultant {

    @Override
    public String getAdvice() {
        return "Turn it off and on again";
    }
}
```

Dependency Injection

- A bean may be dependent on other beans

  - This dependencies are identified by the @Autowired annotation

  - Once dependence is identified by Spring, it scans for a bean that implements the interface in the dependency

  - One a bean is instantiated, Spring either creates or gets a reference to an existing bean that satisfies that dependency

- Two common kinds of DI

  - Constructor injection: the dependency is identified in the constructor and resolved when the Java object is created

  - Setter injection: The dependency is identified in a setter method so allow for a looser coupling between instances

Constructor Injection

- In this example, we have a "Manual" bean which is used by an ITGuru bean

```java
@Component
public class TechGuide implements Manual {

    @Override
    public String lookup() {
        return "Just a sec ... Googling it.";
    }

}
```

Constructor Injection

- In this example, we have a "Manual" bean which is used by an ITGuru bean

  In the ITGuru bean, we tell Spring we need an instance of a bean that implements the "Manual" interface

```java
@Component
public class ITGuru implements Consultant {

    private Manual myManual;

    @Autowired
    public ITGuru(Manual m ) {
        this.myManual = m;
    }

    @Override
    public String getAdvice() {
        return this.myManual.lookup();
    }
}
```

Dependency Injection

- The dependency is defined in terms of an interface

    - This decouples the client bean from any specific implementation of a manual

- In this example, only one TechManual object needs to be present that all the ITGuru objects can share

    - When a bean is used in this way, it is often called a service

    - We generally only need a single copy of a service

Creating a Spring Application

Demo

Spring Lab

End Module