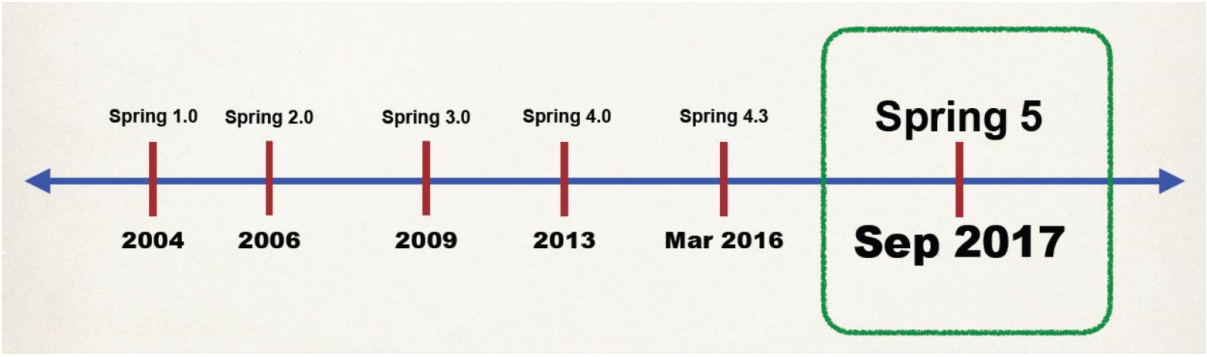
Spring Framework Overview

Spring in a Nutshell

- Very popular framework for building Java applications
- Initially a simpler and lightweight alternative to J2EE
- Provides a large number of helper classes … makes things easier

Spring Release Timeline



Spring Website - Official

www.spring.io

Why Spring?

Simplify Java Enterprise Development

Goals of Spring

- Lightweight development with Java POJOs (Plain-Old-Java-Objects)
- Dependency injection to promote loose coupling
- Declarative programming with Aspect-Oriented-Programming (AOP)
- Minimize boilerplate Java code

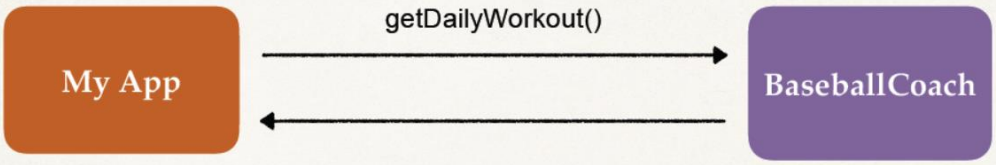
To Do List

1. Create Eclipse Project
2. Download Spring JAR Files
3. Add JAR files to Eclipse Project … *Build Path*

Inversion of Control (IoC)

The approach of outsourcing the construction and management of objects.

Coding Scenario

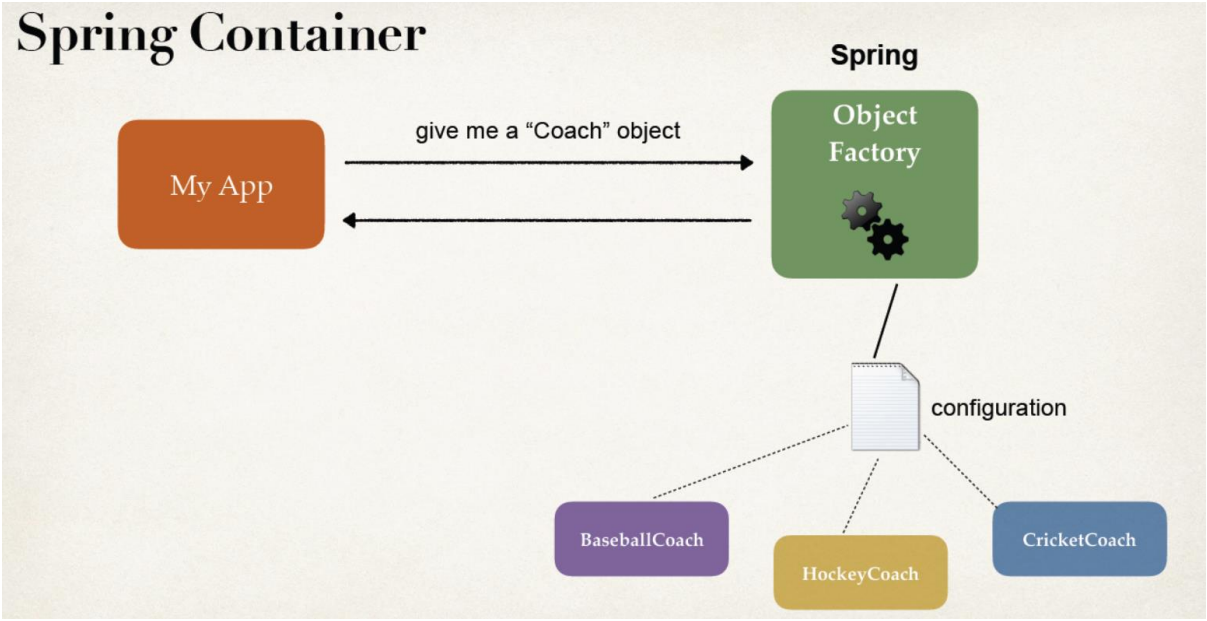


- App should be configurable
- Easily change the coach for another sport
  - Hockey, Cricket, Tennis, Gymnastics etc …

Code Demo

- **MyApp.java**: main method
- **BaseballCoach.java**
- **Coach.java**: interface after refactoring
- **TrackCoach.java**

Spring Container



Spring Container

- Primary functions
  - Create and manage objects *(Inversion of Control)*
  - Inject object's dependencies *(Dependency Injection)*

Configuring Spring Container

- XML configuration file *(legacy, but most legacy apps still use this)*
- Java Annotations *(modern)*
- Java Source Code *(modern)*

Spring Development Process

1. Configure your Spring Beans
2. Create a Spring Container
3. Retrieve Beans from Spring Container

Step 1: Configure your Spring Beans

```
<bean id="mycoach"
class="com.ford.springdemo.BaseballCoach" />
```

Step 2: Create a Spring Container

- Spring container is generically known as **ApplicationContext**

- Specialized implementations
  - ClassPathXmlApplicationContext
  - AnnotationConfigApplicationContext
  - GenericWebApplicationContext
  - others …


```
ClassPathXmlApplicationContext context =
    new ClassPathXmlApplicationContext("applicationContext.xml");
```

Step 3: Retrieve Beans from Container

```
// create a spring container
ClassPathXmlApplicationContext context =
    new ClassPathXmlApplicationContext("applicationContext.xml");

// retrieve bean from spring container
Coach theCoach = context.getBean("myCoach", Coach.class);
```

Dependency Injection

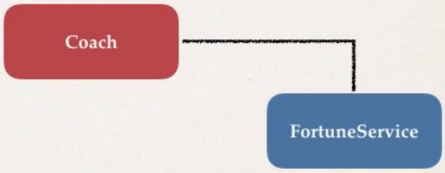
The dependency inversion principle.

The client delegates to calls to another object the responsibility of providing its dependencies.

Demo Example

dependency = helper

- Our **Coach** already provides daily workouts
- Now will also provide daily fortunes
  - New helper: **FortuneService**
  - This is a ***dependency***



Injection Types

- We will cover the two most common
  - Constructor Injection
  - Setter Injection
- Will talk about "auto-wiring" in the Annotations section later

Development Process - Constructor Injection

1. Define the dependency interface and class
2. Create a constructor in your class for injections
3. Configure the dependency injection in Spring config file

Step 1: Define the dependency interface and class

File: FortuneService.java

```
public interface FortuneService {

  public String getFortune();

}
```

File: HappyFortuneService.java

```
public class HappyFortuneService implements FortuneService {

  public String getFortune() {
    return "Today is your lucky day!";
  }
}
```

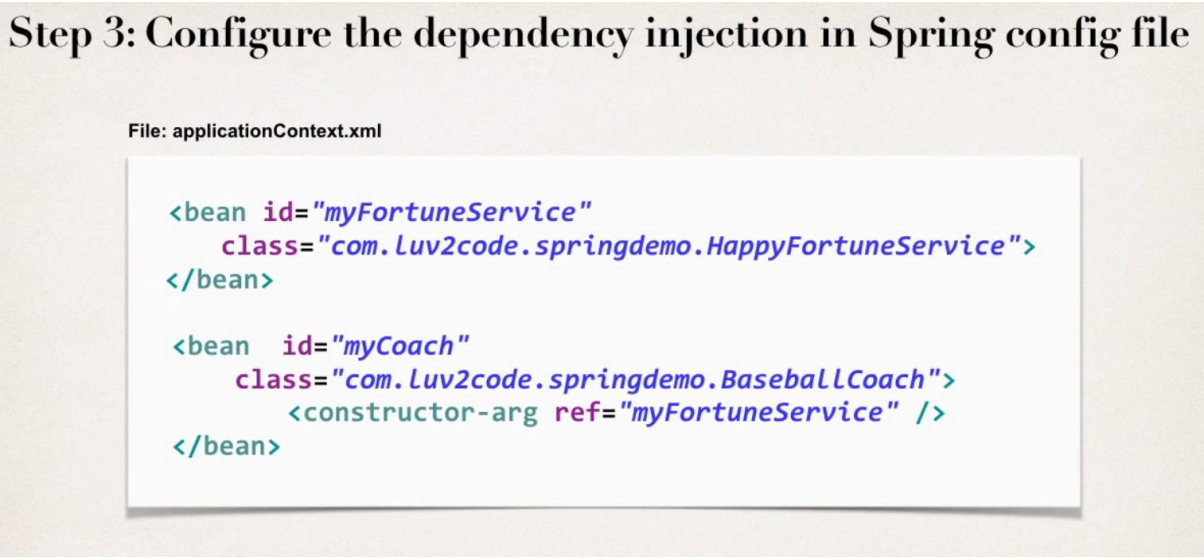
Step 2: Create a constructor in your class for injections

File: BaseballCoach.java

```
public class BaseballCoach implements Coach {

  private FortuneService fortuneService;

  public BaseballCoach(FortuneService theFortuneService) {
    fortuneService = theFortuneService;
  }
   ...
}
```



Setter Injection

Literal Value

Bean Scopes

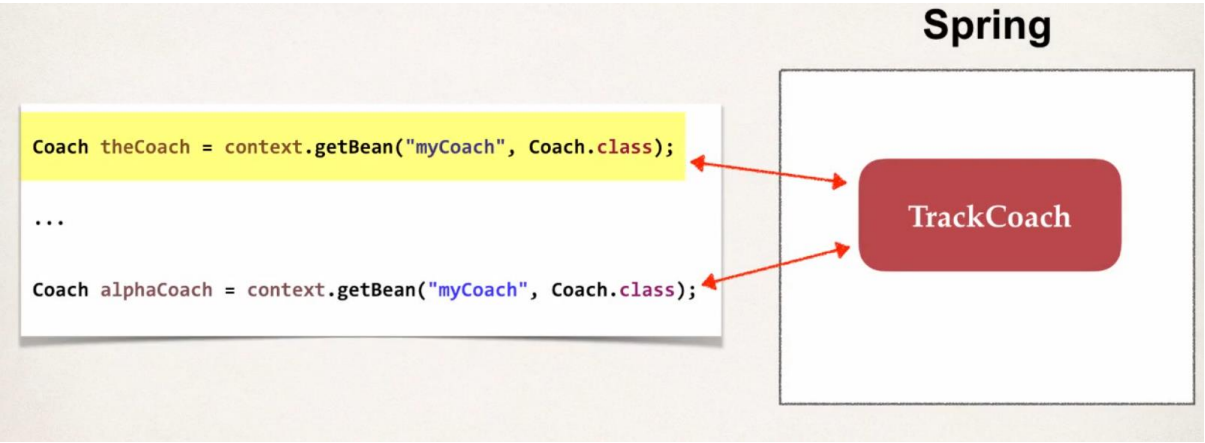
- Scope refers to the lifecycle of a bean
- How long does the bean live?
- How many instances are created?
- How is the bean shared?

Additional Spring Bean Scopes

Scope	Description
singleton	Create a single shared instance of the bean. Default scope.
prototype	Creates a new bean instance for each container request.
request	Scoped to an HTTP web request. Only used for web apps.
session	Scoped to an HTTP web session. Only used for web apps.
global-session	Scoped to a global HTTP web session. Only used for web apps.

What Is a Singleton?

- Spring Container creates only one instance of the bean, by default
- It is cached in memory
- All requests for the bean
  - will return a SHARED reference to the SAME bean




```
<bean id="mycoach" class="com.ford.springdemo.BaseballCoach"
 scope="singleton"  />
```

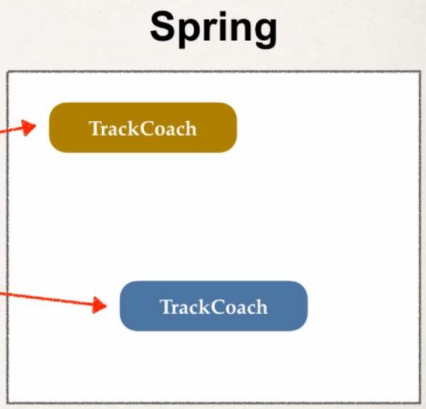
Prototype Scope Example

Prototype scope: new object for each request

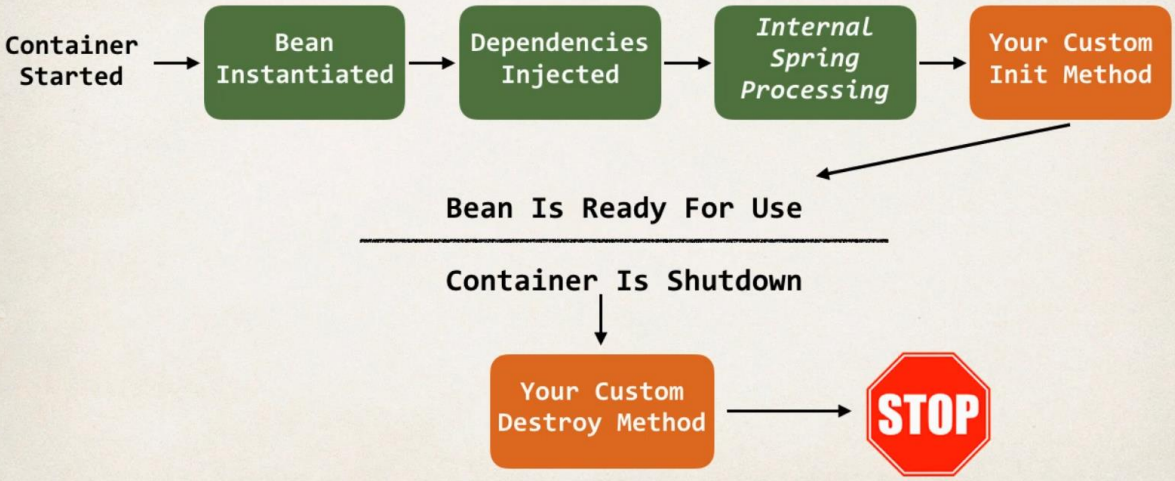
```
Coach theCoach = context.getBean("myCoach", Coach.class);

...

Coach alphaCoach = context.getBean("myCoach", Coach.class);
```



Bean Lifecycle



Bean Lifecycle Methods / Hooks

- You can add custom code during **bean initialization**
  - Calling custom business logic methods
  - Setting up handles to resources (db, sockets, file etc)
- You can add custom code during **bean destruction**
  - Calling custom business logic method
  - Clean up handles to resources (db, sockets, files etc)

```
<bean id="mycoach" class="com.ford.springdemo.BaseballCoach"
 init-method="doMyStartupStuff"  destroy-method="doMyCleanupStuff"  />
```