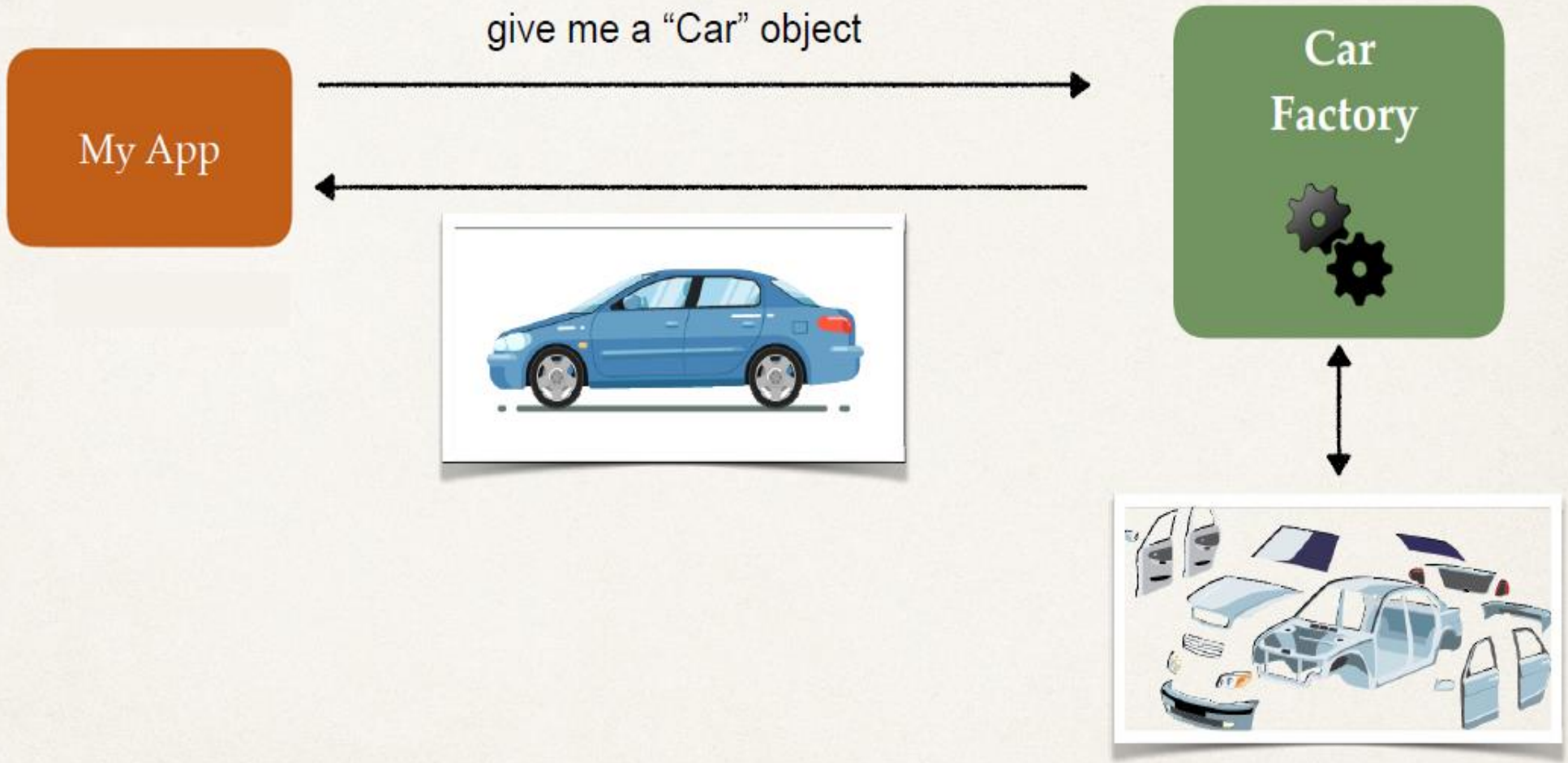
Spring Dependency Injection

Dependency Injection

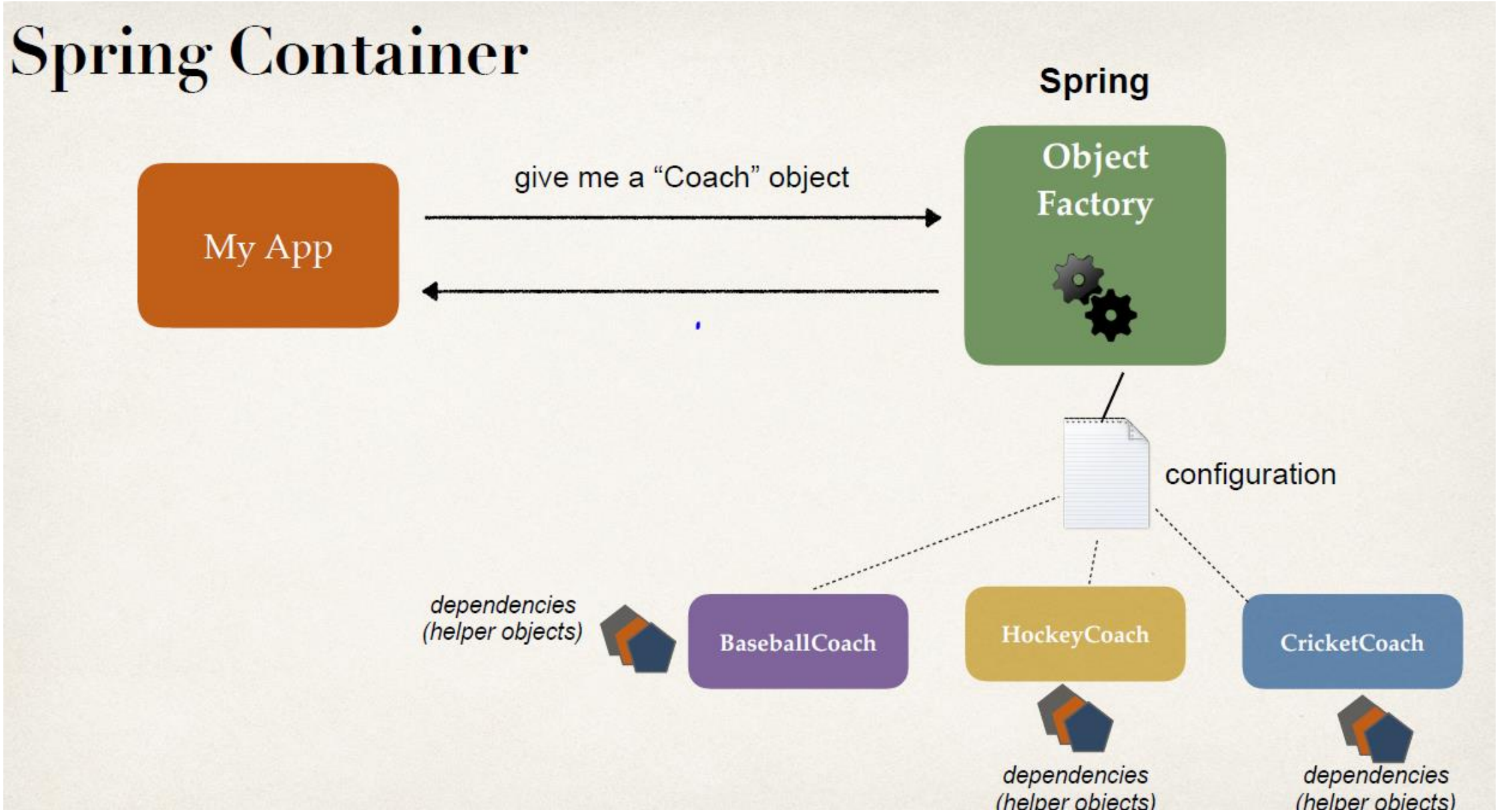
The dependency inversion principle.

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

Car Factory



Spring Container



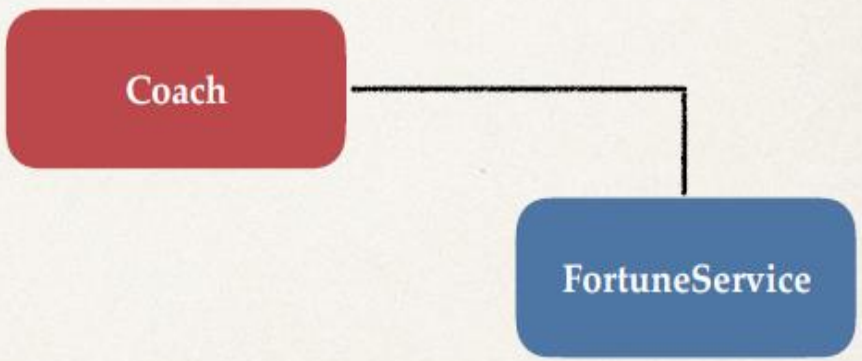
Spring Container

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

Demo Example

- Our **Coach** already provides daily workouts

- Now will also provide daily fortunes
  - New helper: **FortuneService**
  - This is a *dependency*



Injection Types

- There are many types of injection with Spring
- 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

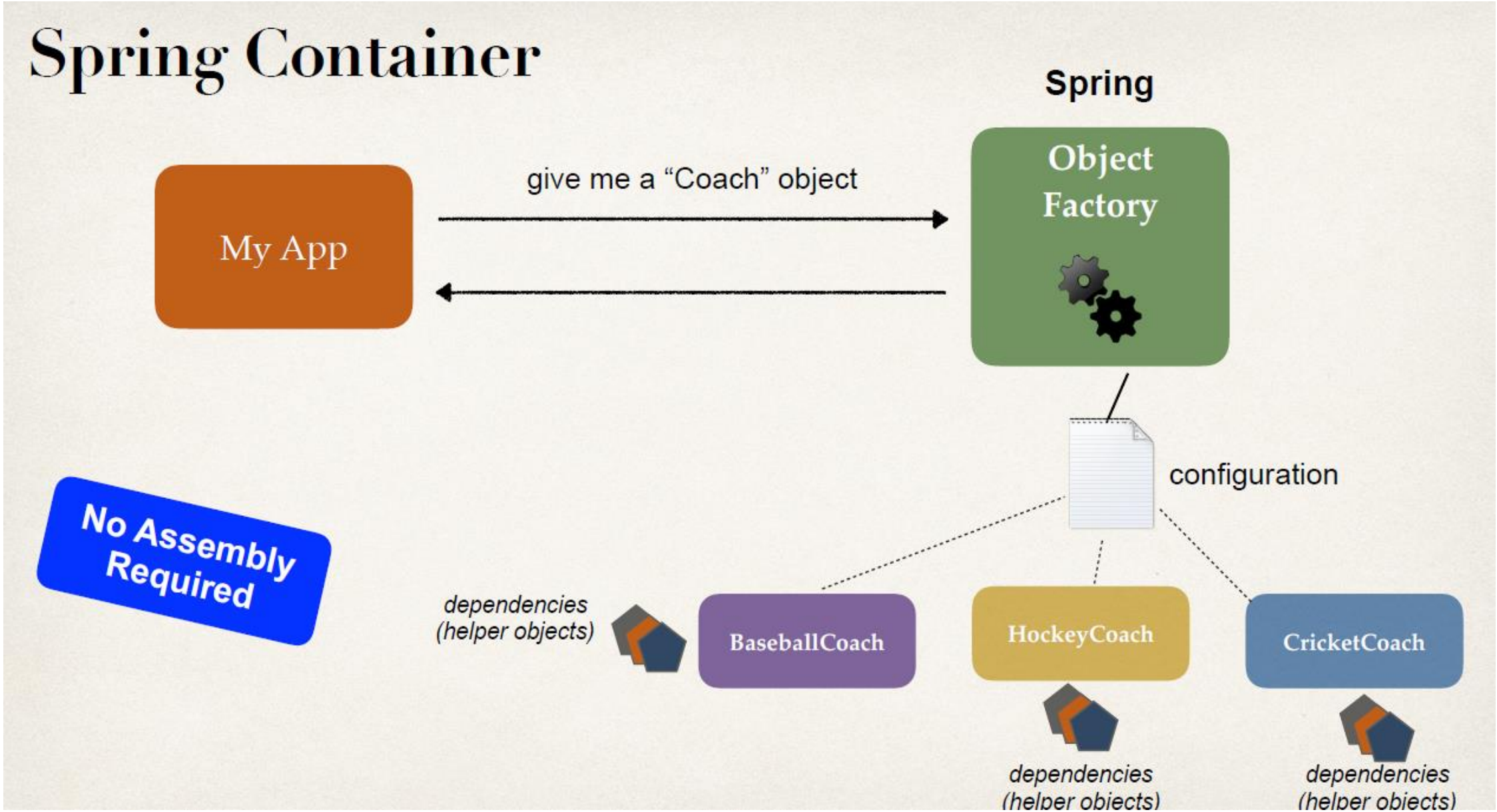
```
public class BaseballCoach implements Coach {

    private FortuneService fortuneService;

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

Step 3: Configure the dependency injection in Spring config file

File: applicationContext.xml

```
<!-- define the dependency -->
<bean id="myFortuneService"
        class="com.springdemo.HappyFortuneService">
    </bean>

<bean id="myCoach"
    class="com.springdemo.TrackCoach">

    <!-- set up constructor injection -->
    <constructor-arg ref="myFortuneService" />
</bean>
```

Spring Container



Constructor Injection - Behind the Scenes

How Spring Processes your Config File

File: applicationContext.xml

```
<!-- define the dependency -->
<bean id="myFortuneService"
        class="com.springdemo.HappyFortuneService">
    </bean>

<bean id="myCoach"
    class="com.springdemo.TrackCoach">

    <!-- set up constructor injection -->
    <constructor-arg ref="myFortuneService" />
</bean>
```

The Spring Framework will perform operations

behind the scenes for you :-)

How Spring Processes your Config File

```xml
<!-- define the dependency -->
<bean id="myFortuneService"
        class="com.springdemo.HappyFortuneService">
    </bean>

<bean id="myCoach"
    class="com.springdemo.TrackCoach">

    <!-- set up constructor injection -->
    <constructor-arg ref="myFortuneService" />
</bean>
```

Spring Framework

```
HappyFortuneService myFortuneService=new HappyFortuneService();

TrackCoach myTrackCoach=new TrackCoach(myFortuneService);
```

Setter Injection

Spring Injection Types

- Constructor Injection
- Setter Injection

Inject dependencies by calling

setter method(s) on your class

Development Process - Setter Injection

1. Create setter method(s) in your class for injections

2. Configure the dependency injection in Spring config file

Step1: Create setter method(s) in your class for injections

File: CricketCoach.java

```
public class CricketCoach implements Coach {

   private FortuneService fortuneService;

   public CricketCoach() {
   }

   public void setFortuneService(FortuneService fortuneService) {
      this.fortuneService = fortuneService;
   }
  ...
}
```

Step 2: Configure the dependency injection in Spring config file

File: applicationContext.xml

```
    <!-- define the dependency -->
<bean id="myFortuneService"
        class="com.springdemo.HappyFortuneService">
    </bean>

<bean id="myCricketCoach"
    class="com.springdemo.CricketCoach">
        <!-- set up setter injection -->
    <property name="fortuneService" ref="myFortuneService" />

</bean>
```

Call setter method on Java class

```
<property name="fortuneService" ref="myFortuneService" />
```

```
public void setFortuneService(...)
```

How Spring Processes your Config File

```
    <!-- define the dependency -->
<bean id="myFortuneService"
        class="com.springdemo.HappyFortuneService">
    </bean>
```

Spring Framework

```
HappyFortuneService myFortuneService =
                        new HappyFortuneService();
```

How Spring Processes your Config File

```
    <!-- define the dependency -->
 <bean id="myFortuneService"
        class="com.springdemo.HappyFortuneService">
    </bean>

<bean id="myCricketCoach"
    class="com.springdemo.CricketCoach">
        <!-- set up setter injection -->
    <property name="fortuneService" ref="myFortuneService" />

</bean>
```

Spring Framework

```
HappyFortuneService myFortuneService =
                    new HappyFortuneService();
```

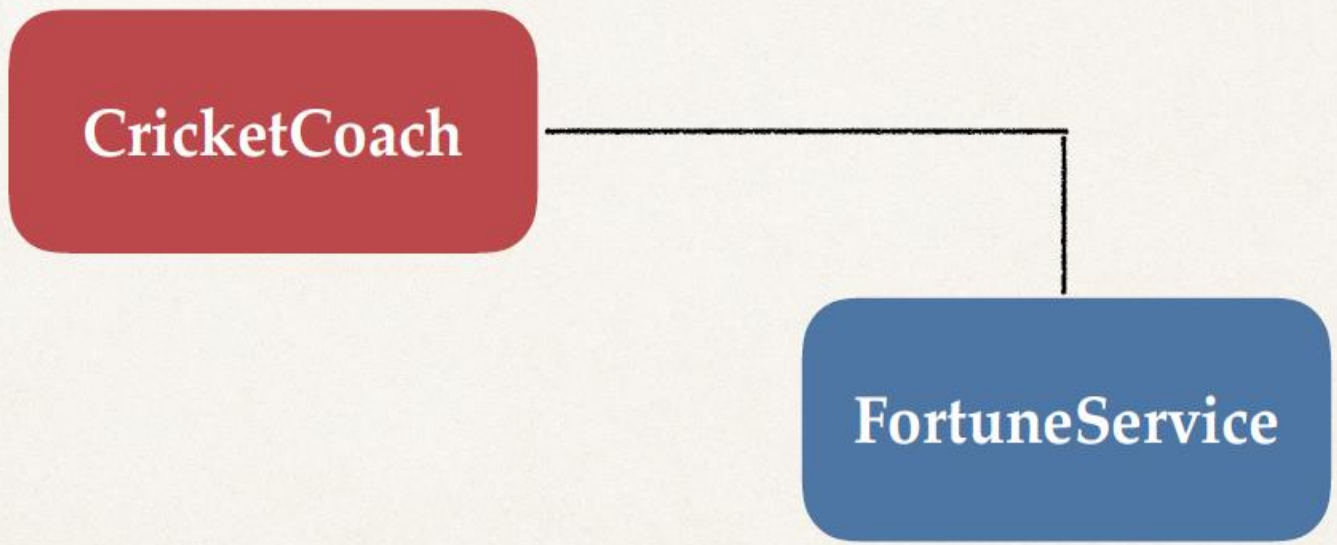
Spring Framework

```
CricketCoach myCricketCoach =
        new CricketCoach();

myCricketCoach.setFortuneService(myFortuneService);
```

Injecting Literal Values

Injecting Literal Values

```
emailAddress:thebestcoach@com
team : Sunrisers Hyderabad
```



Development Process

1. Create setter method(s) in your class for injections

2. Configure the injection in Spring config file

Step1: Create setter method(s) in your class for injections

File: CricketCoach.java

```
public class CricketCoach implements Coach {

   private String emailAddress;
   private String team;

   public void setEmailAddress(String emailAddress) ...

   public void setTeam(String team) ...

   ...
}
```

Step 2: Configure the injection in Spring config file

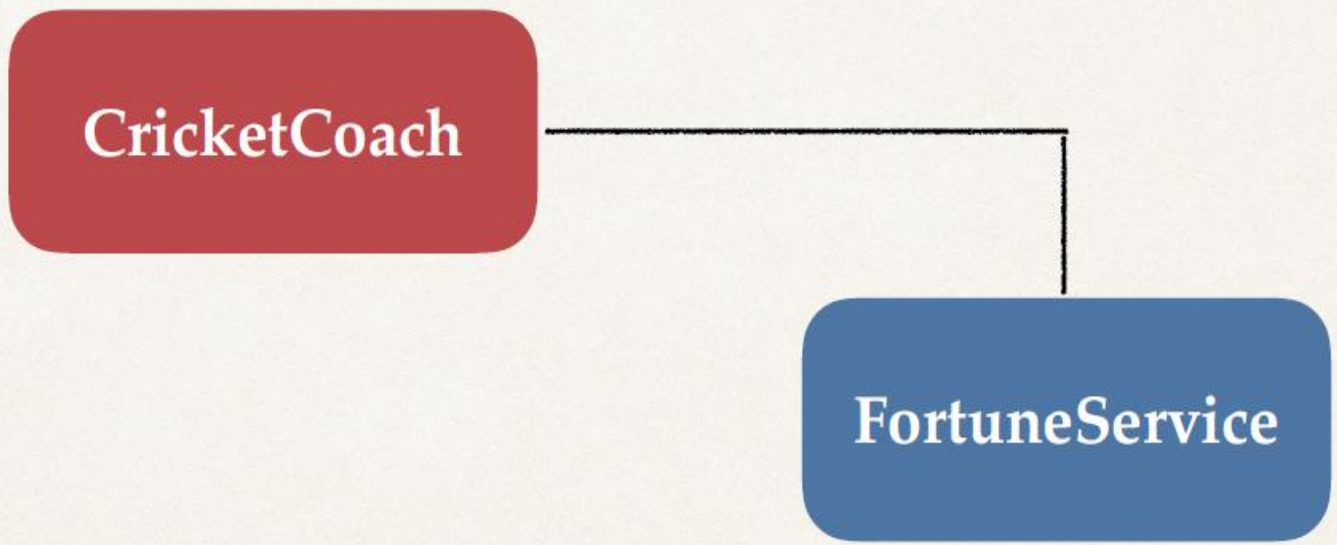
File: applicationContext.xml

```
<bean id="myCricketCoach"
    class="com.springdemo.CricketCoach">

    <!-- set up setter injection -->
    <property name="fortuneService" ref="myFortuneService" />

    <!-- inject literal values -->
    <property name="emailAddress" value="thebestcoach@coach.com" />
    <property name="team" value="Sunrisers Hyderabad" />

</bean>
```

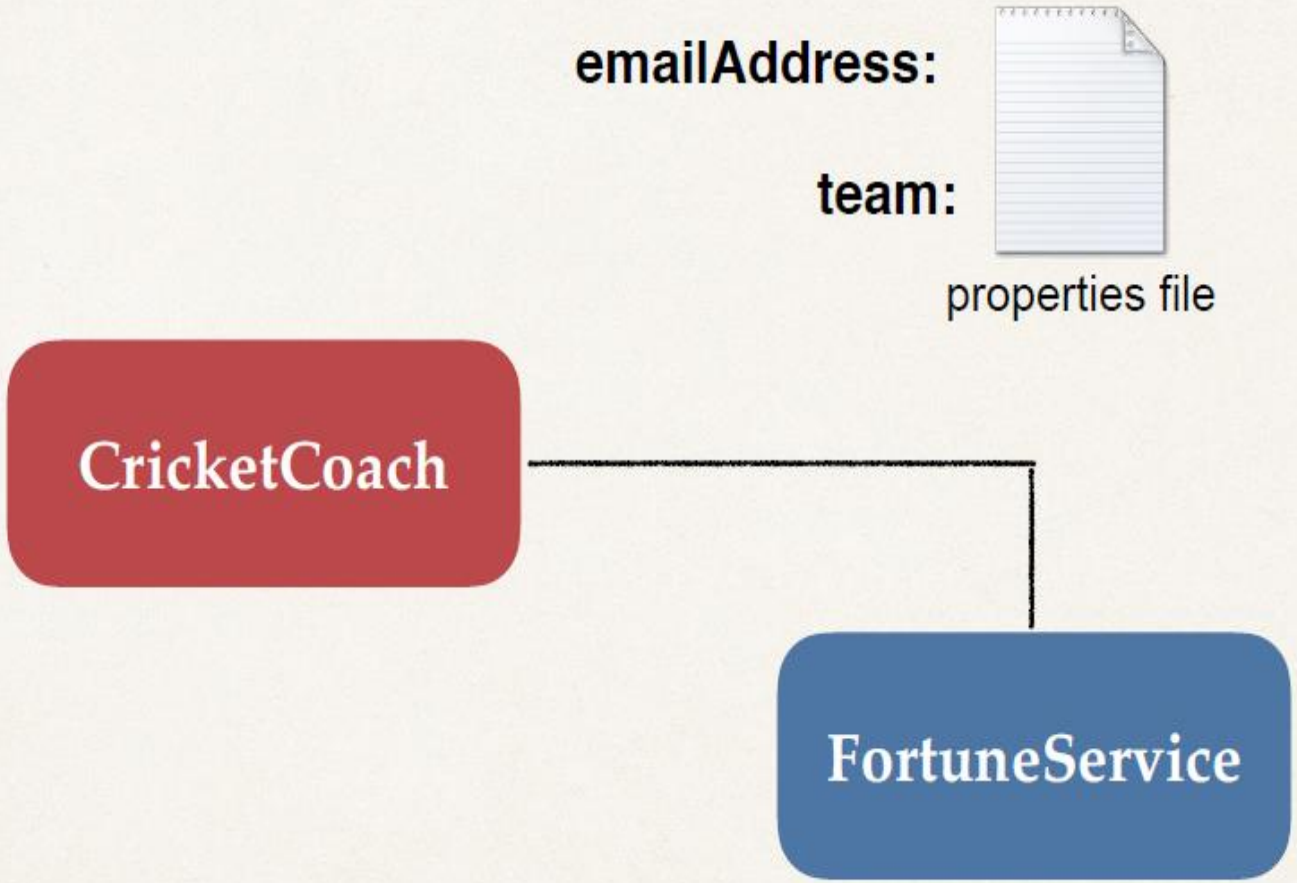
Injecting Values from Properties File

Injecting Literal Values

```
emailAddress:thebestcoach@com
team : Sunrisers Hyderabad
```



Read from a Properties File



Development Process

1. Create Properties File
2. Load Properties File in Spring config file
3. Reference values from Properties File

Step 1: Create Properties File

File: sport.properties

```
foo.email=myeasycoach@coach.com
foo.team=Royal Challengers Bangalore
```

Step 2: Load Properties file in Spring config file

File: applicationContext.xml

```
<context:property-placeholder location="classpath:sport.properties"/>
```

Step 2: Load Properties file in Spring config file

File: applicationContext.xml

```
<context:property-placeholder location="classpath:sport.properties"/>
```

Step 3: Reference Values from Properties File

File: applicationContext.xml

```
<bean id="myCricketCoach"
        class="com.springdemo.CricketCoach">

    <!-- set up setter injection -->
    <property name="fortuneService" ref="myFortuneService" />

    <!-- inject literal values -->
    <property name="emailAddress" value="${foo.email}" />
    <property name="team" value="${foo.team}" />

    </bean>
```

```
foo.email=myeasycoach@luv2code.com
foo.team=Royal Challengers Bangalore
```