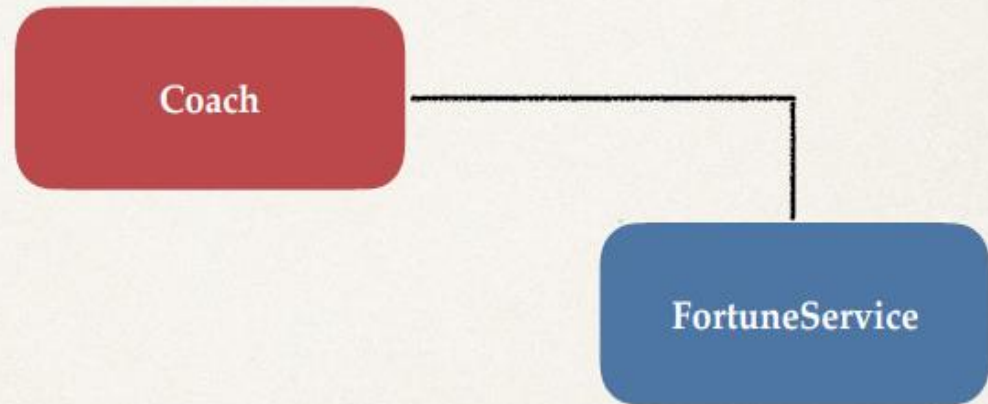


Spring Dependency Injection with Annotations and Autowiring

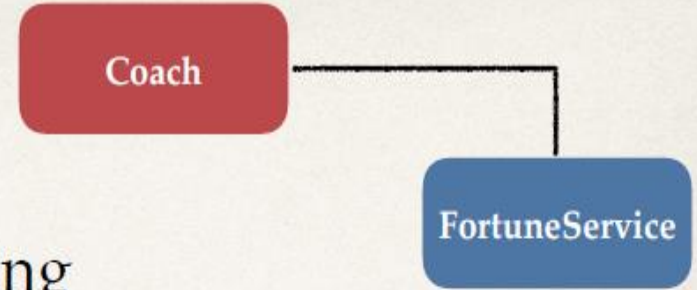


Demo Example

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

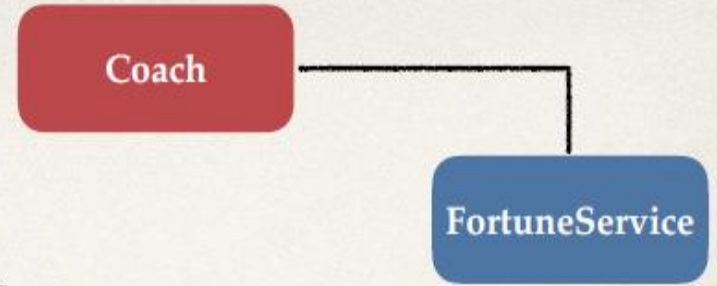


What is Spring AutoWiring?



- For dependency injection, Spring can use auto wiring
- Spring will look for a class that *matches* the property
 - *matches by type*: class or interface
- Spring will inject it automatically ... hence it is autowired

Autowiring Example



- Injecting FortuneService into a Coach implementation
- Spring will scan @Components
- Any one implements FortuneService interface???
- If so, let's inject them. For example: *HappyFortuneService*

Autowiring Injection Types

- Constructor Injection
- Setter Injection
- Field Injections

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 with **@Autowired** Annotation

Step-By-Step

Step 1: Define the dependency interface and class

File: FortuneService.java

```
public interface FortuneService {  
  
    public String getFortune();  
  
}
```

File: HappyFortuneService.java

```
@Component  
public class HappyFortuneService implements FortuneService {  
  
    public String getFortune() {  
        return "Today is your lucky day!";  
    }  
}
```

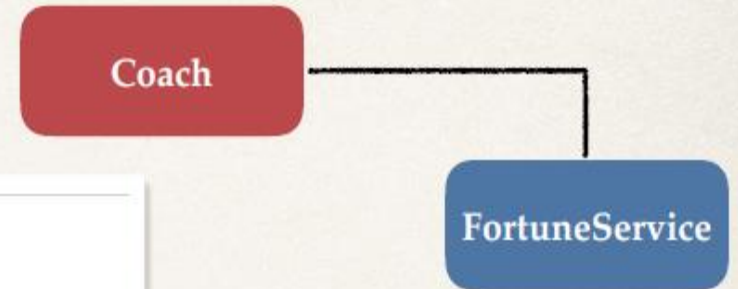
Step 2: Create a constructor in your class for injections

File: TennisCoach.java

```
@Component
public class TennisCoach implements Coach {

    private FortuneService fortuneService;

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



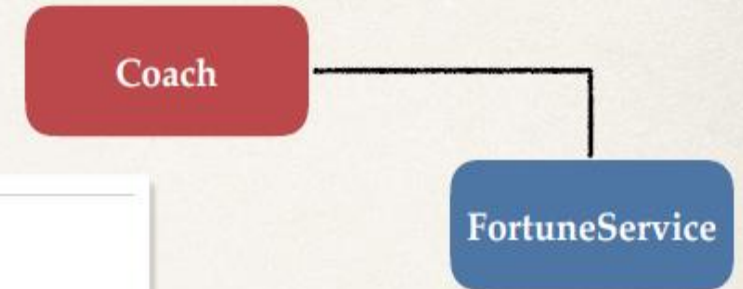
Step 2: Create a constructor in your class for injections

File: TennisCoach.java

```
@Component
public class TennisCoach implements Coach {

    private FortuneService fortuneService;

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



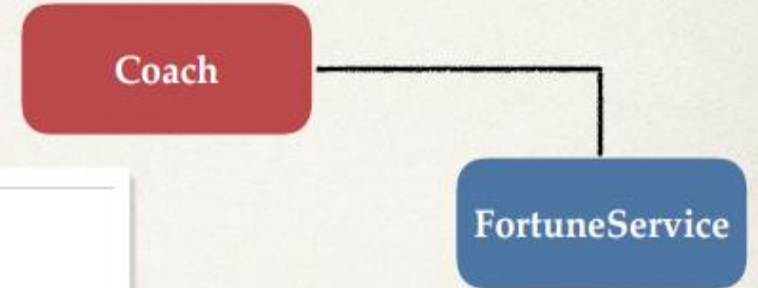
Step 3: Configure the dependency injection @Autowired annotation

File: TennisCoach.java

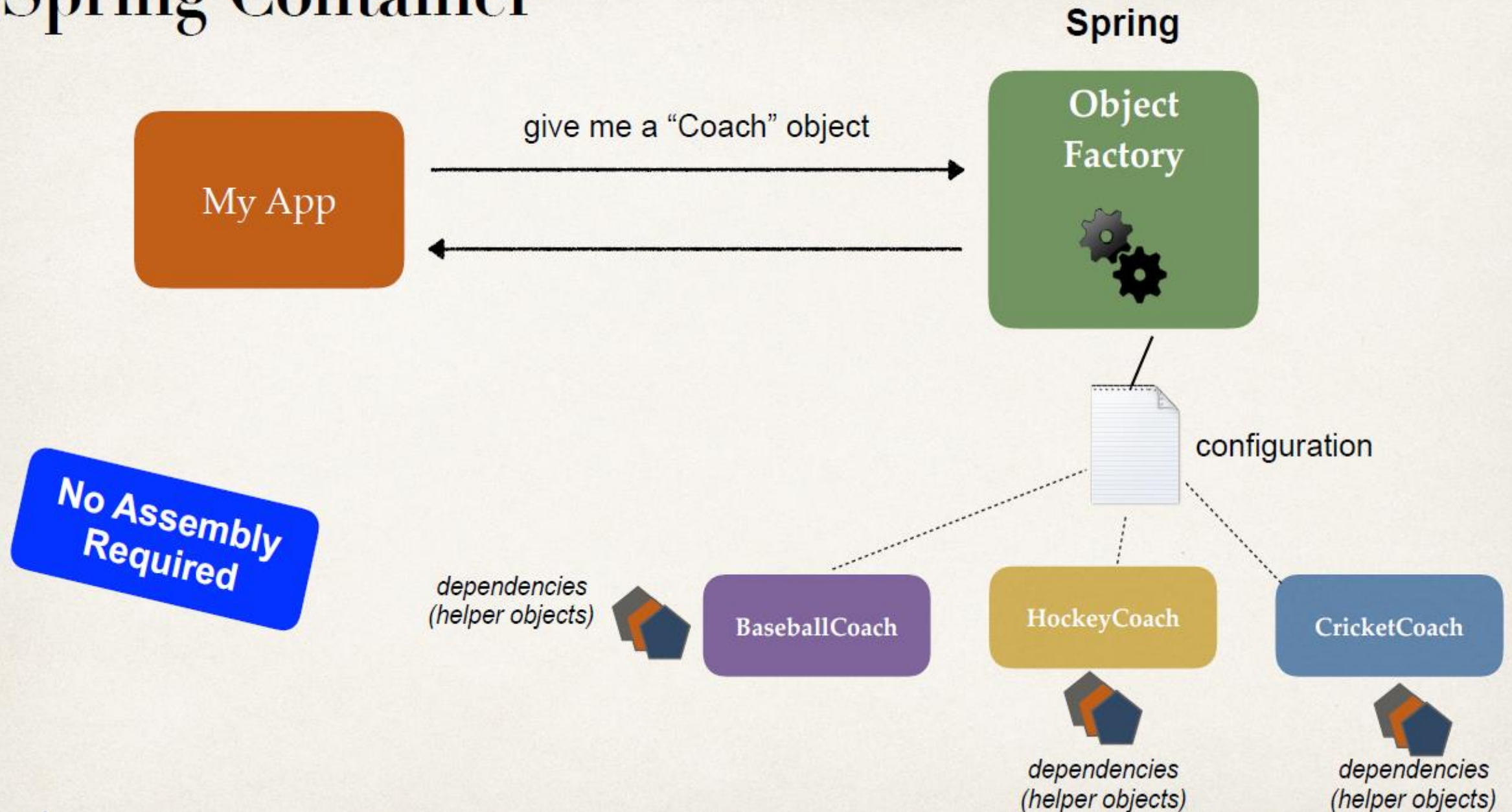
```
@Component
public class TennisCoach implements Coach {

    private FortuneService fortuneService;

    @Autowired
    public TennisCoach(FortuneService theFortuneService) {
        fortuneService = theFortuneService;
    }
    ...
}
```



Spring Container



Setter Injection with Annotations and Autowiring

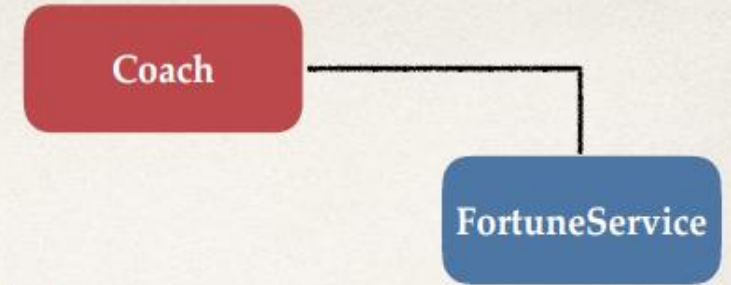


Spring Injection Types

- Constructor Injection
- Setter Injection
- Field Injection

**Inject dependencies by calling
setter method(s) on your class**

Autowiring Example



- Injecting FortuneService into a Coach implementation
- Spring will scan @Components
- Any one implements FortuneService interface???
- If so, let's inject them. For example: *HappyFortuneService*

Development Process - Setter Injection

Step-By-Step

1. Create setter method(s) in your class for injections
2. Configure the dependency injection with **@Autowired** Annotation

Setter Injection with Annotations and Autowiring

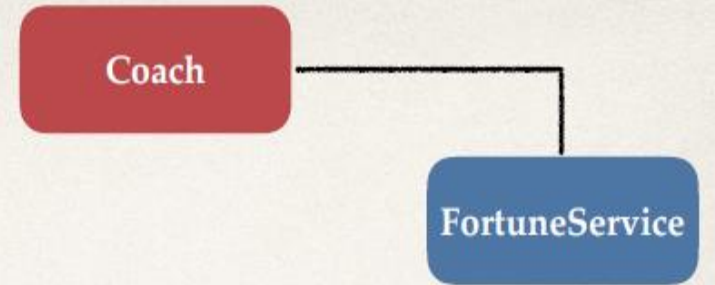


Spring Injection Types

- Constructor Injection
- Setter Injection
- Field Injection

**Inject dependencies by calling
setter method(s) on your class**

Autowiring Example



- Injecting FortuneService into a Coach implementation
- Spring will scan @Components
- Any one implements FortuneService interface???
- If so, let's inject them. For example: *HappyFortuneService*

Development Process - Setter Injection

Step-By-Step

1. Create setter method(s) in your class for injections
2. Configure the dependency injection with **@Autowired** Annotation

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

File: TennisCoach.java

```
@Component
public class TennisCoach implements Coach {

    private FortuneService fortuneService;

    public TennisCoach() {
    }

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

Step 2: Configure the dependency injection with Autowired Annotation

File: TennisCoach.java

```
@Component
public class TennisCoach implements Coach {

    private FortuneService fortuneService;

    public TennisCoach() {
    }

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


Step 2: Configure the dependency injection with Autowired Annotation

File: TennisCoach.java

```
@Component
public class TennisCoach implements Coach {

    private FortuneService fortuneService;

    public TennisCoach() {
    }

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

Inject dependencies by calling

ANY method on your class

Simply give: @Autowired

Step 2: Configure the dependency injection with *Autowired* Annotation

File: TennisCoach.java

@Component

public class TennisCoach **implements** Coach {

private FortuneService fortuneService;

public TennisCoach() {
 }

@Autowired

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

 ...
}

Field Injection with Annotations and Autowiring



Spring Injection Types

- Constructor Injection
- Setter Injection
- Field Injection

**Inject dependencies by setting field values
on your class directly
(even private fields)**

Accomplished by using Java Reflection

Development Process - Field Injection



1. Configure the dependency injection with Autowired Annotation

- ❖ Applied directly to the field
- ❖ No need for setter methods

Step 1: Configure the dependency injection with Autowired Annotation

File: TennisCoach.java

```
public class TennisCoach implements Coach {  
  
    @Autowired  
    private FortuneService fortuneService;  
  
    public TennisCoach() {  
    }  
  
    // no need for setter methods  
    ...  
}
```

Spring Injection Types

- Constructor Injection
- Setter Injection
- Field Injection

Spring Injection Types

- Constructor Injection
- Setter Injection
- Field Injection

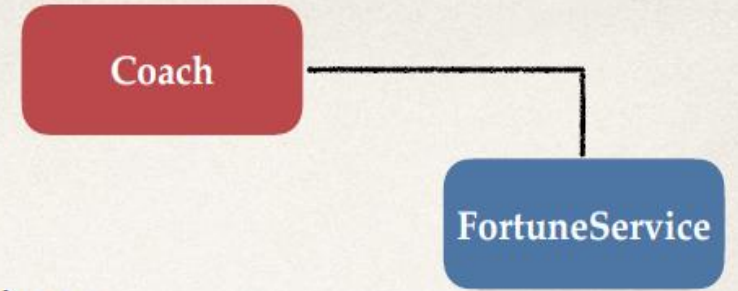
Choose a style

Stay consistent in your project

Annotation Autowiring and Qualifiers

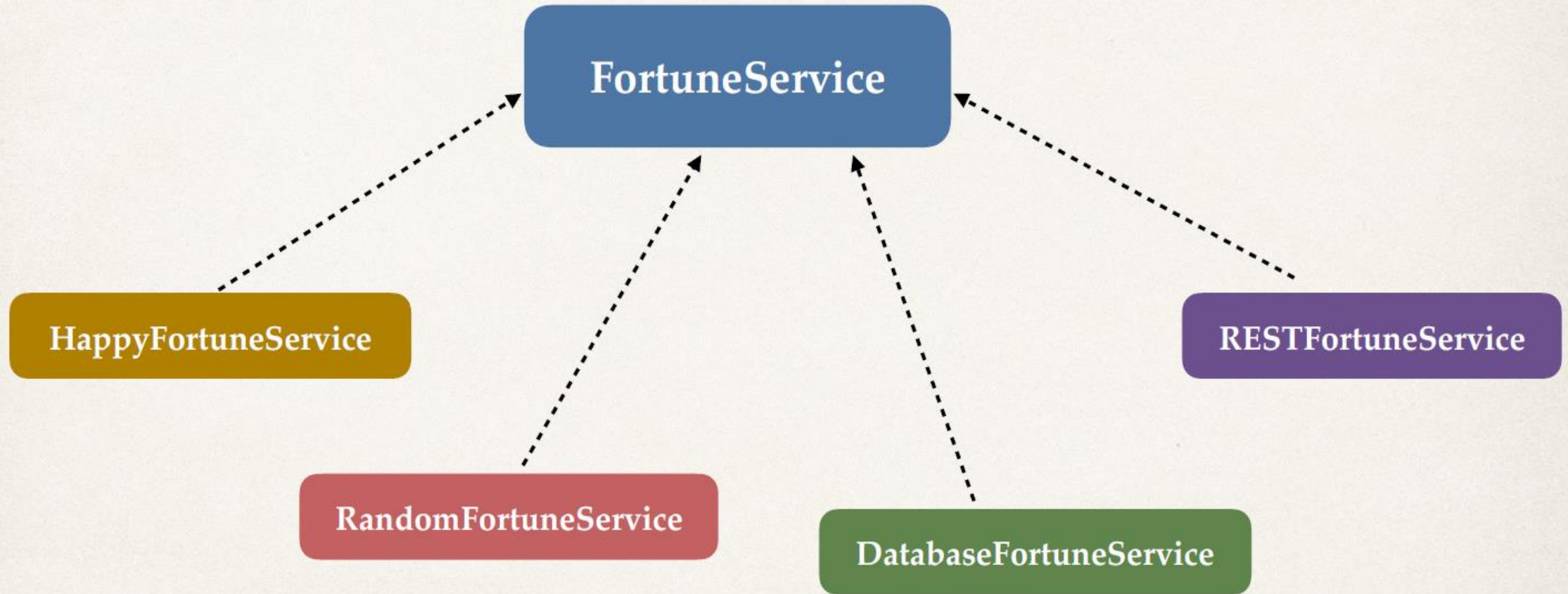


Autowiring



- Injecting FortuneService into a Coach implementation
- Spring will scan @Components
- Any one implements FortuneService interface???
- If so, let's inject them ... *oops which one?*

Multiple FortuneService Implementations



Umm, we have a little problem

[> TennisCoach: inside default constructor

May 13, 2019 5:47:53 AM org.springframework.context.support.AbstractApplicationContext.refresh

WARNING: Exception encountered during bootstrap in 'DefaultAnnotationConfigApplicationContext': [org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'TennisCoach': Unsatisfied dependency exception](#)

Exception in thread "main" org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'TennisCoach': Unsatisfied dependency exception

at org.springframework.beans.factory.annotation.AutowiredAnnotationMethodPostProcessor.inject(AutowiredAnnotationMethodPostProcessor.java:111)

at org.springframework.beans.factory.annotation.InjectionMetadata.inject(InjectionMetadata.java:90)

at org.springframework.beans.factory.annotation.AutowiredAnnotationMethodPostProcessor.postProcessDependencies(AutowiredAnnotationMethodPostProcessor.java:424)

at org.springframework.beans.factory.support.AbstractAutowiredCapableBeanFactory.populateBean(AbstractAutowiredCapableBeanFactory.java:1411)

at org.springframework.beans.factory.support.AbstractAutowiredCapableBeanFactory.doInstantiateBean(AbstractAutowiredCapableBeanFactory.java:553)

at org.springframework.beans.factory.support.AbstractAutowiredCapableBeanFactory.createBean(AbstractAutowiredCapableBeanFactory.java:515)

at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:229)

at org.springframework.beans.factory.support.DefaultSingletonBeanRegistry.getSingleton(DefaultSingletonBeanRegistry.java:222)

at org.springframework.beans.factory.support.AbstractBeanFactory.doGetBean(AbstractBeanFactory.java:416)

at org.springframework.beans.factory.support.AbstractBeanFactory.getBean(AbstractBeanFactory.java:144)

at org.springframework.beans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(DefaultListableBeanFactory.java:643)

at org.springframework.context.support.AbstractApplicationContext.finishBeanFactoryInitialization(AbstractApplicationContext.java:877)

at org.springframework.context.support.AbstractApplicationContext.refresh(AbstractApplicationContext.java:549)

at org.springframework.boot.loader.launch.ClassPathApplicationContext\$Initializer.<init>(ClassPathApplicationContext.java:144)

at org.springframework.boot.loader.launch.ClassPathApplicationContext\$Initializer.<init>(ClassPathApplicationContext.java:85)

at com.springdemo.AnnotationDemoApp.main(AnnotationDemoApp.java:11)

Caused by: org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type 'com.springdemo.FortuneService' available: expected single m

at org.springframework.beans.factory.config.DependencyDescriptor.resolveAtUnique(DependencyDescriptor.java:221)

at org.springframework.beans.factory.support.DefaultListableBeanFactory.doResolveDependency(DefaultListableBeanFactory.java:1225)

at org.springframework.beans.factory.support.DefaultListableBeanFactory.resolveDependency(DefaultListableBeanFactory.java:1167)

at org.springframework.beans.factory.annotation.AutowiredAnnotationMethodPostProcessor.inject(AutowiredAnnotationMethodPostProcessor.java:111)

... 15 more

Solution: Be specific! - @Qualifier

```
@Component
public class TennisCoach implements Coach {

    @Autowired
    @Qualifier("happyFortuneService")
    private FortuneService fortuneService;

    ...

}
```


Solution: Be specific! - @Qualifier

```
@Component  
public class TennisCoach implements Coach {  
  
    @Autowired  
    @Qualifier("happyFortuneService")  
    private FortuneService fortuneService;  
  
    ...  
  
}
```

Injection Types

- Can apply **@Qualifier** annotation to
 - Constructor injection
 - Setter injection methods
 - Field injection