**8. Spring Configuration with Java Annotations - Dependency Injection**

Here we inject the dependency using annotations and AutoWiring.

**AutoWiring**:

Autowiring is a mechanism of resolving dependencies in Spring IoC container. So instead of directly specifying the dependency (in XML or in Java configuration), you can depend on container itself to provide you with candidate(s). Spring itself should abort if you find more than one matching dependency (unless you are looking for a collection of beans).

For dependency injection, Spring can automatically ware up our object together. Spring will look for a class that matches a given property. And it will actually match by type, so the type could be either the class or the interface. Once spring find the match then it will automatically injected. Hence it call autowired.

**AutoWiring Example**:

* Inject 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**:

There are three different types of injection

1. Constructor injection
2. Setter injection
3. Field injection

**Development Process - Constructor injection**:

* Define the dependency interface and class
* Create a constructor in your class for injections
* Configure the dependency injection with @Autowired Annotation

**1) Define the dependency interface and class**:

**File: FortuneService.java**:

public interface FortuneService {

public String getFortune();

}

**File: HappyFortuneService.jav**:

@Component

public class HappyFortuneService implements FortuneService {

public String getFortune() {

return "Today is your lucky day!";

}

}

**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;

}

...

}

**3) Configure the dependency injection with @Autowired Annotation**:

**File: TenniseCoach.java**:

@Component

public class TennisCoach implements Coach {

private FortuneService fortuneService;

@Autowired

public TennisCoach(FortuneService theFortuneService) {

fortuneService = theFortuneService;

}

...

}

**Example**:

**File: Coach.java (interface)**:

**package** com.ruhul.odduu;

**public** **interface** Coach {

**public** String getDailyWorout();

**public** String getDailyFortune();

}

**File: FortuneService.java (interface)**:

**package** com.ruhul.odduu;

**public** **interface** FortuneService {

**public** String getFortune();

}

**File: HappyFortuneService.java (clsaa)**:

**package** com.ruhul.odduu;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** HappyFortuneService **implements** FortuneService {

@Override

**public** String getFortune() {

**return** "Today is your lucky day!!!";

}

}

**File: TennisCoach.java(class)**:

**package** com.ruhul.odduu;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** TennisCoach **implements** Coach {

**private** FortuneService fortuneService;

@Autowired

**public** TennisCoach(FortuneService fortuneService) {

**this**.fortuneService = fortuneService;

}

@Override

**public** String getDailyWorout() {

**return** "Practice your backhand volley";

}

@Override

**public** String getDailyFortune() {

**return** fortuneService.getFortune();

}

}

**File: AnnotationDemoApp.java (class)**:

**package** com.ruhul.odduu;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** AnnotationDemoApp {

**public** **static** **void** main(String[] args) {

// read spring congig file

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

// get the bean from container

Coach theCoach = context.getBean("tennisCoach", Coach.**class**);

// call a method on the bean

System.***out***.println(theCoach.getDailyWorout());

// call method to get the daily fortune

System.***out***.println(theCoach.getDailyFortune());

// close the context

context.close();

}

}

**File: applicationContext.java**:

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

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- add entry to enable component scanning -->

<context:component-scan base-package=*"com.ruhul.odduu"* />

</beans>

/\*

Output:

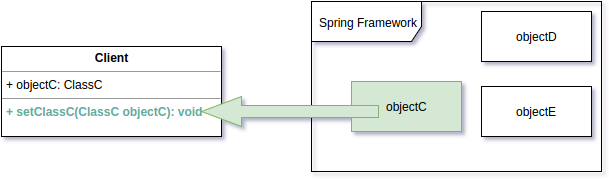
Practice your backhand volley

Today is your lucky day!!!

\*/

**Setter Injection**:

Setter Injection in Spring is a type of dependency injection in which the framework injects the dependent objects into the client using a setter method. The container first calls the no argument constructor and then calls the setters.



Spring Configuration with Java Annotations - Dependency Injection