Bộ môn: Kỹ Thuật Phần Mềm

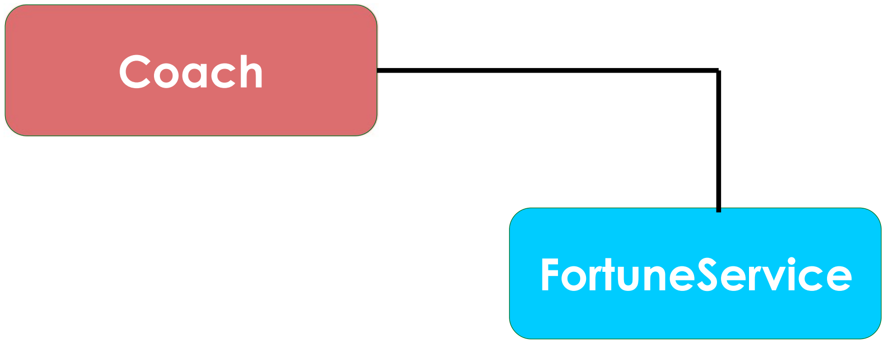
Lập trình WWW *Java*

.

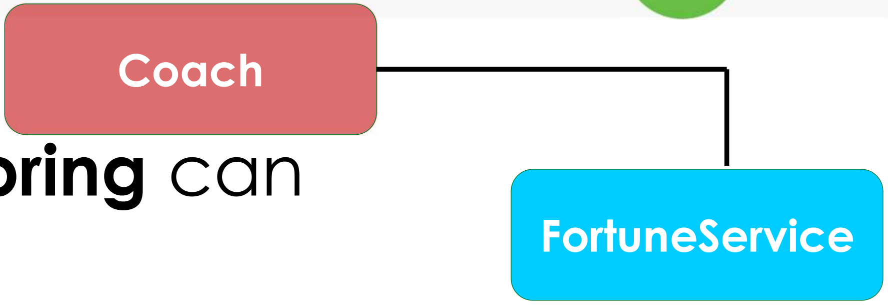
Spring DI (Dependency Injection)

Coding Scenario

- » **Our Coach** already provide **daily workouts**
- » Now will also provide **daily fortune → FortuneService** (dependency)



Spring Auto Wiring



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

Autowiring Injection Types

- Constructor Injection
- Setter Injection
- Field Injection

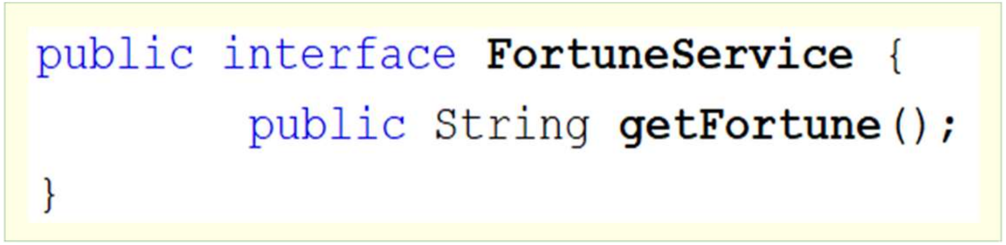
Constructor Injection

Development Process - Constructor Injection

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

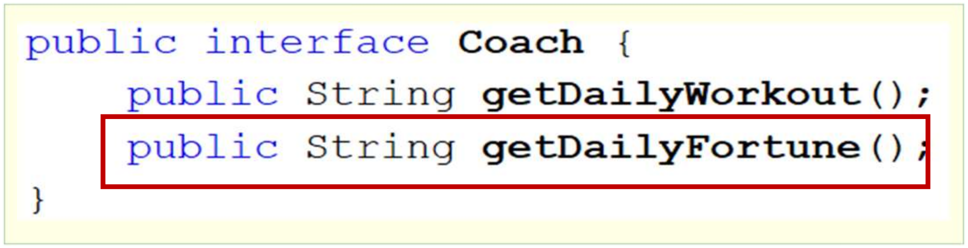
Step 1: Define dependency interface and class

File: *FortuneService.java*



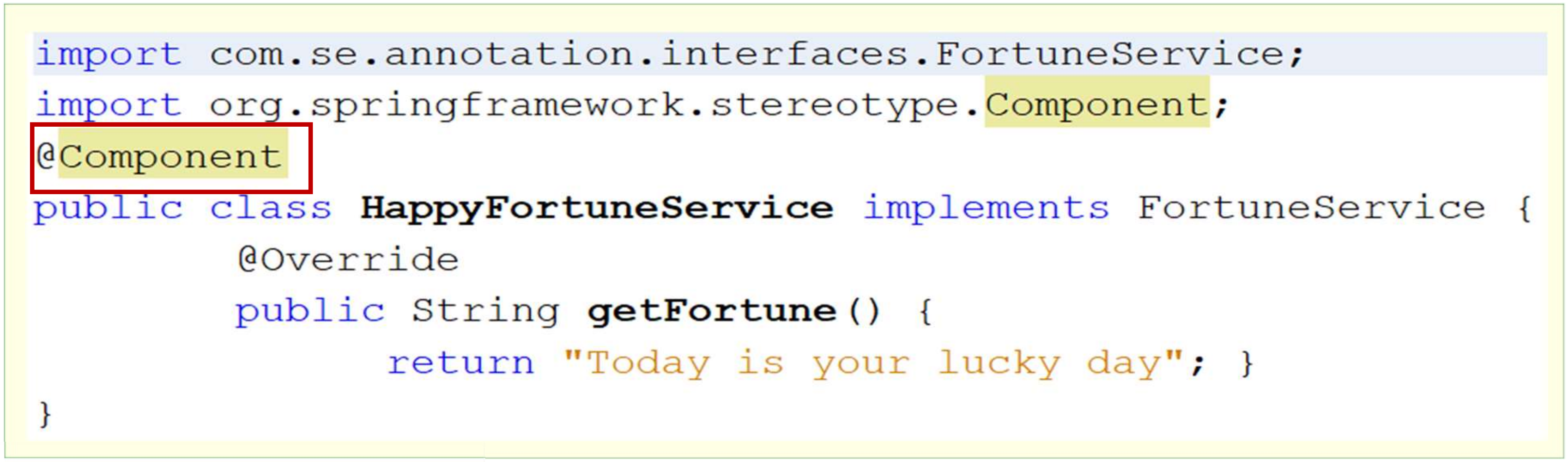
```
public interface FortuneService {
        public String getFortune();
}
```

File: *Coach.java*



```
public interface Coach {
     public String getDailyWorkout();
     public String getDailyFortune();
}
```

File: *HappyFortuneService.java*



```
import com.se.annotation.interfaces.FortuneService;
import org.springframework.stereotype.Component;
@Component
public class HappyFortuneService implements FortuneService {
        @Override
        public String getFortune() {
                return "Today is your lucky day"; }
}
```

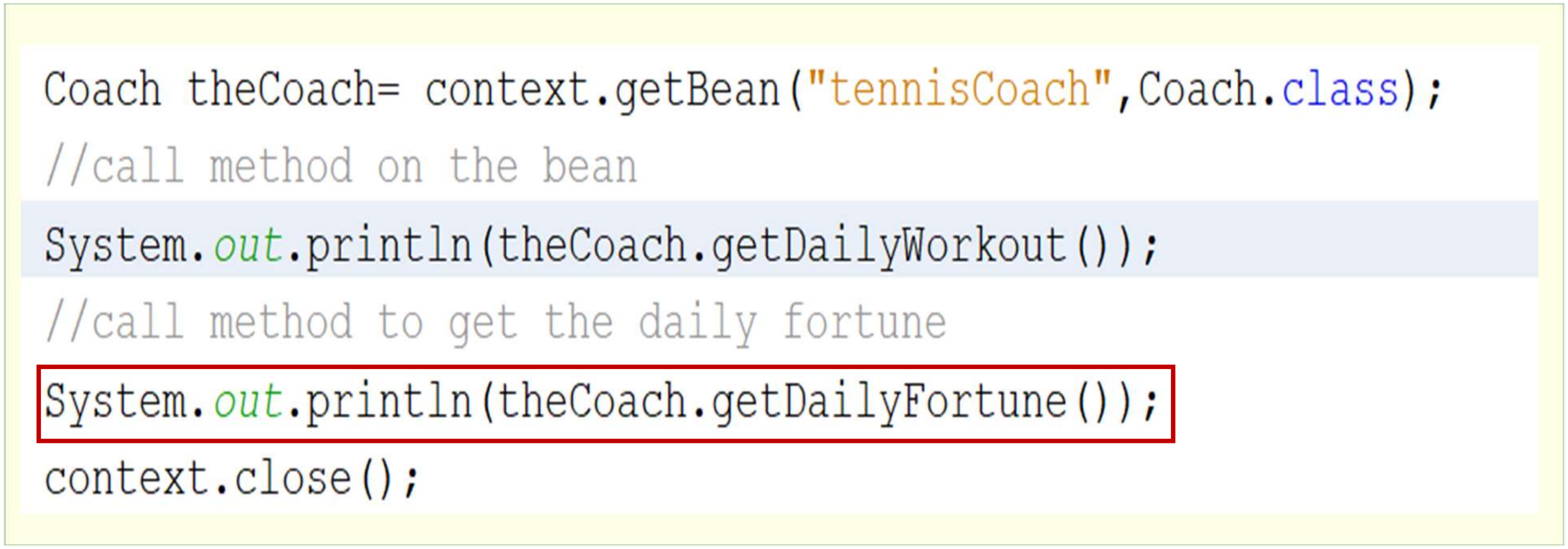
Step 2, 3:
Create a constructor in your class for injection
Configure the dependency injection with @Autowired annotation

File*: TennisCoach.java*

```java
@Component
public class TennisCoach implements Coach {
    private FortuneService fortuneService;
    @Autowired
    public  TennisCoach(FortuneService theFortuneService)
                fortuneService = theFortuneService;}
    @Override
    public String getDailyFortune() {
        return fortuneService.getFortune();}
```

Test Demo Application

File: *AnnotationDemoApp.java*



```java
Coach theCoach= context.getBean("tennisCoach",Coach.class);
//call method on the bean
System.out.println(theCoach.getDailyWorkout());
//call method to get the daily fortune
System.out.println(theCoach.getDailyFortune());
context.close();
```

Setter Injection

Setter Injection

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

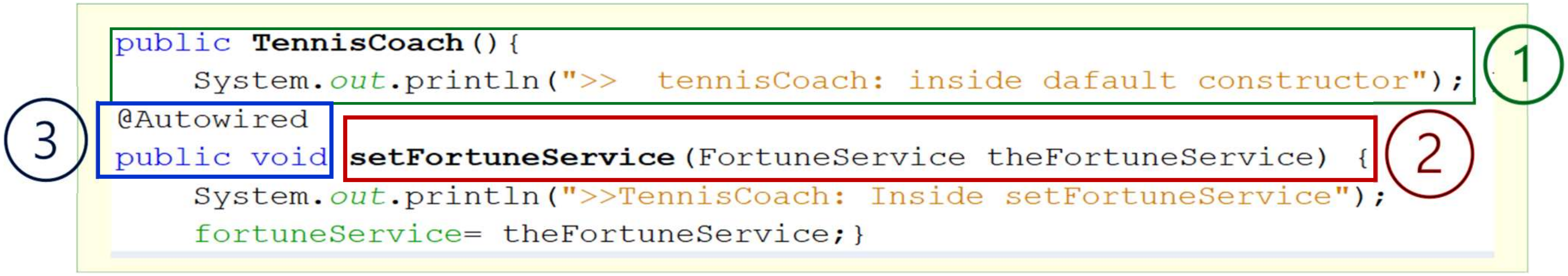
Development Process - Setter Injection

» Create setter method(s) in your class for **injection**

» Configure the **dependency injection** with **@Autowired** annotation

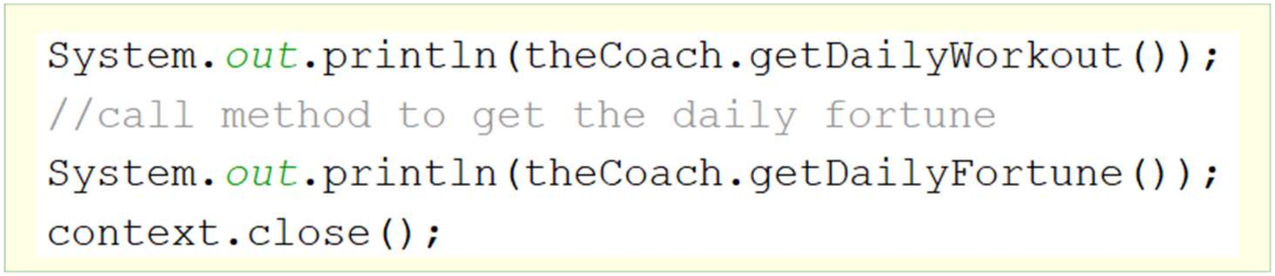
Step 1: Create setter method(s) in your class for injection
Step 2: Configure the dependency injection with @Autowired annotation

File*: TennisCoach.java*

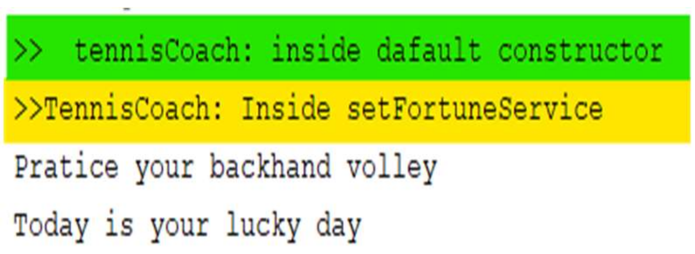


```java
public TennisCoach(){
    System.out.println(">>  tennisCoach: inside dafault constructor");
@Autowired
public void setFortuneService(FortuneService theFortuneService)  {
    System.out.println(">>TennisCoach: Inside setFortuneService");
    fortuneService= theFortuneService;}
```

File*: AnnotationDemoApp.java*



```java
System.out.println(theCoach.getDailyWorkout());
//call method to get the daily fortune
System.out.println(theCoach.getDailyFortune());
context.close();
```



```
>>  tennisCoach: inside dafault constructor
>>TennisCoach: Inside setFortuneService
Pratice your backhand volley
Today is your lucky day
```

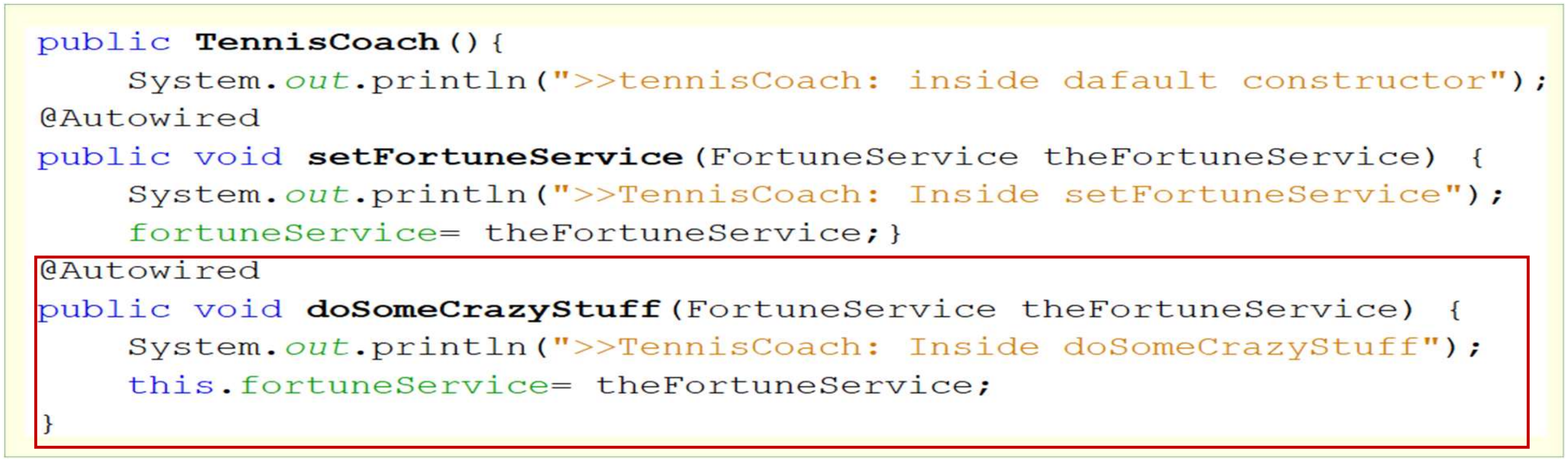
Method Injection

Inject dependencies by calling ANY method(s) on your class

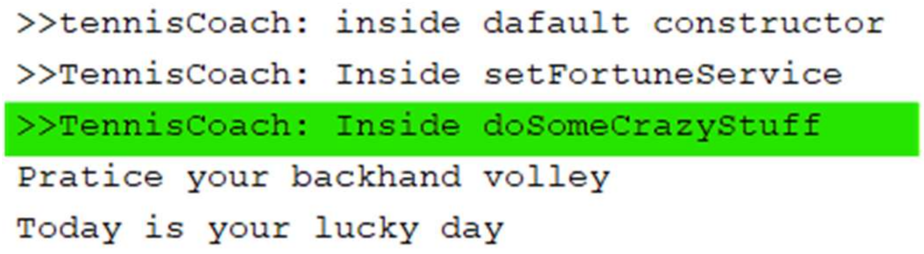
Simply give @autowire

Development Process - Method Injection

File: *TennisCoach.java*



```
public TennisCoach(){
     System.out.println(">>tennisCoach: inside dafault constructor");
@Autowired
public void setFortuneService(FortuneService theFortuneService) {
     System.out.println(">>TennisCoach: Inside setFortuneService");
     fortuneService= theFortuneService;}
@Autowired
public void doSomeCrazyStuff(FortuneService theFortuneService) {
     System.out.println(">>TennisCoach: Inside doSomeCrazyStuff");
     this.fortuneService= theFortuneService;
}
```



```
>>tennisCoach: inside dafault constructor
>>TennisCoach: Inside setFortuneService
>>TennisCoach: Inside doSomeCrazyStuff
Pratice your backhand volley
Today is your lucky day
```

Field Injection

Field Injection

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

Accomplished by using Java Reflection

Development Process - Field Injection

Config the **dependency injection** with **Autowired annotation**

- Applied **directly to the field**
- **No need** setter method/constructor

Development Process - Field Injection

File: *TennisCoach.java*

```
@Autowired
private FortuneService fortuneService;
//no need for setter method or constructor
```

```
>>tennisCoach: inside dafault constructor
Pratice your backhand volley
Today is your lucky day
```

Which one should I use?

- Constructor Injection
- Setter Injection
- Field Injection

Choose a style and stay consistency in your project

QUESTIONS