Bộ môn: Kỹ Thuật Phần Mềm

Lập trình WWW *Java*

.

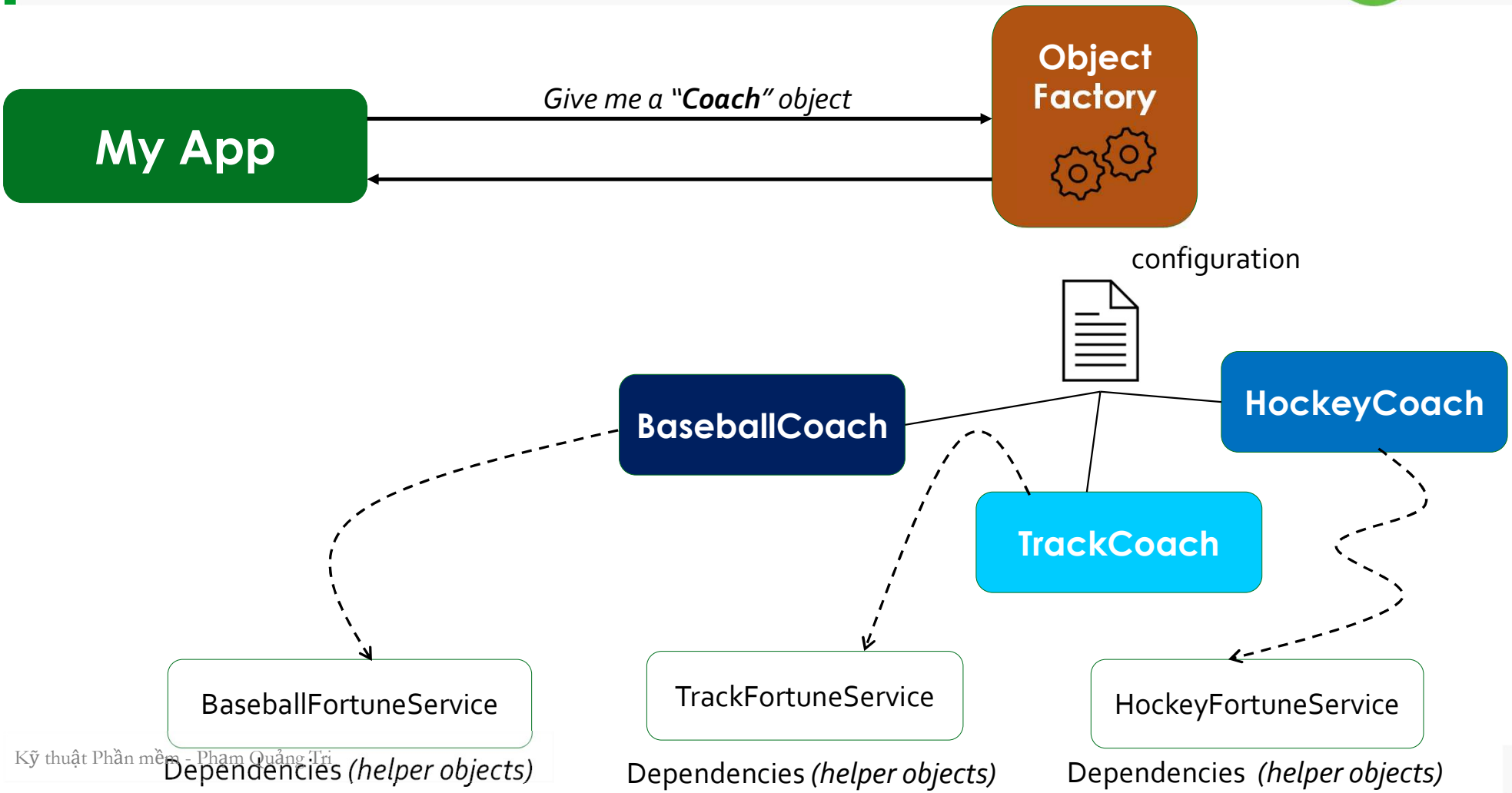
Spring DI (Dependency Injection)

Dependency Injection (DI)

The dependency inversion principle

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

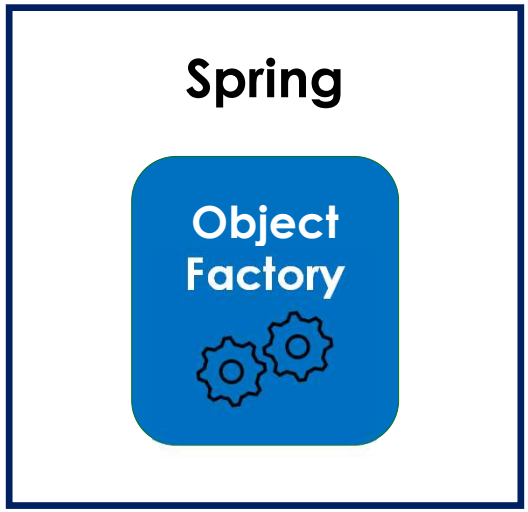
Coding Scenario



Coding Scenario

Primary functions of Spring Container:

- **Create** and **manage** objects (IoC)
- **Inject** object's **dependencies** (DI)



Can we use IoC to create dependence object?

Injection Types

There are many types of injections with Spring. We will cover two most common:

- **Constructor Injection**
- **Setter Injection**

Development Process – Constructor Injection

- **Define** the **dependency interface** and **Class**
- **Create** a **constructor** in **your class** for injection
- **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 {
        @Override
        public String getFortune(){
                return "Today is your lucky day!";}
}
```

Step 2: Create a constructor in your class for injection

File*: BaseballCoach.java*

```
public class BaseballCoach implements Coach{
    //define a private field for dependency
    private FortuneService fortuneSevice;
    //define a constructor for denpendency injection
    public BaseballCoach(FortuneService theFortuneService ) {
            fortuneSevice= theFortuneService;
    }
      ...
```

Define field

Define constructor

Step 3: Configure the dependency injection in Spring config file

File: *ApplicationContext.xml*

```
<bean id="myFortune"
    class="com.se.springdemo.libs.HappyFortuneService">
</bean>
<bean id="myCoach"
    class="com.se.springdemo.libs.BaseballCoach">
    <constructor-arg ref = "myFortune" />
</bean>
```

Define dependency/helper

inject the dependency/helper using constructor injection

Setter Injection

Inject dependencies by calling **setter methods** on your class

Development Process – Setter Injection

- **Create setter method(s)** in your class for **injection**
- **Configure** the **dependency** injection in Spring config file

Step 1: Create setter method(s) in your class for injection

File: *CricketCoach.java*

```
public class CricketCoach implements Coach {

    private FortuneService fortuneService;

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

called by Spring during setter injection

Step 2: Configure the dependency injection in Spring config file

File: *applicationContext.xml*

```
<bean id="myFortune"
    class="com.se.springdemo.libs.HappyFortuneService">
</bean>
<bean id= "myCricketCoach"
      class="com.se.springdemo.libs.CricketCoach"    >
      <property name="fortuneService" ref="myFortune" />
</bean>
```

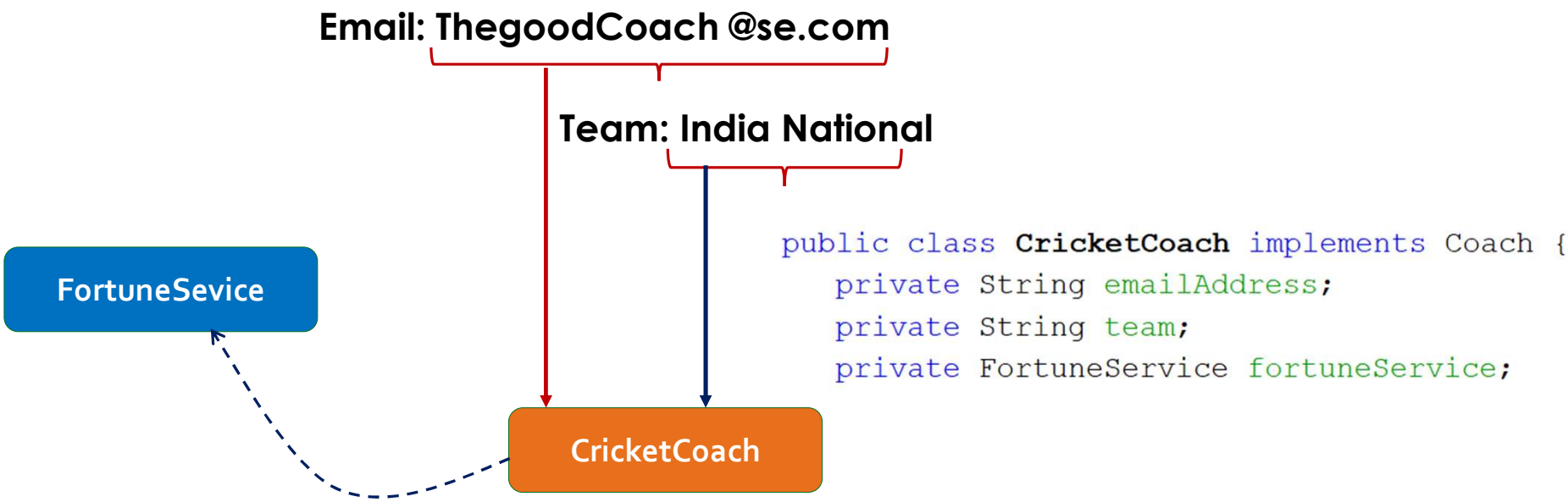
File: *CricketCoach.java*

```
public class CricketCoach implements Coach {

   private FortuneService fortuneService;

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

Injecting Literal Values



Development Process – Literal Values Injection

- **Create setter method(s)** in your class for **injection**
- **Configure** the **dependency** injection in Spring config file

Step 1: Create setter method(s) in your class for injection

File: *CricketCoach.java*

```
public class CricketCoach implements Coach {
    private String emailAddress;
    private String team;
    public void setEmailAddress(String mailAddress) {...4 lines }
    public void setTeam(String team) {...4 lines }
```

Create private fields

Create setter methods

Step 2: Configure the dependency injection in Spring config file

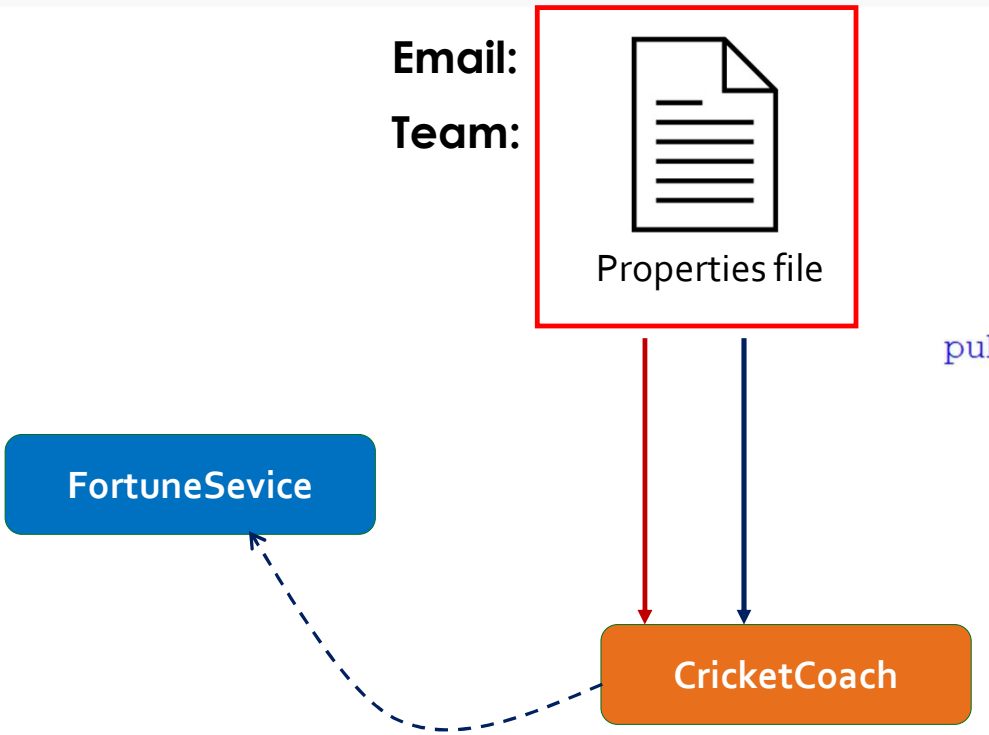
File: *applicationContext.xml*

```
<bean id="myFortune"
    class="com.se.springdemo.libs.HappyFortuneService">
</bean>
<bean id= "myCricketCoach"
      class="com.se.springdemo.libs.CricketCoach"    >
      <property name="fortuneService" ref="myFortune" />
      <property name="emailAddress" value="ThegoodCoach@se.com " />
      <property name="team" value="India National" />
</bean>
```

File: *CricketCoach.java*

```
public class CricketCoach implements Coach {
   private String emailAddress;
   private String team;
   public void setEmailAddress(String mailAddress) {...4 lines }
   public void setTeam(String team) {...4 lines }
```

Injecting Literal Values from Properties File

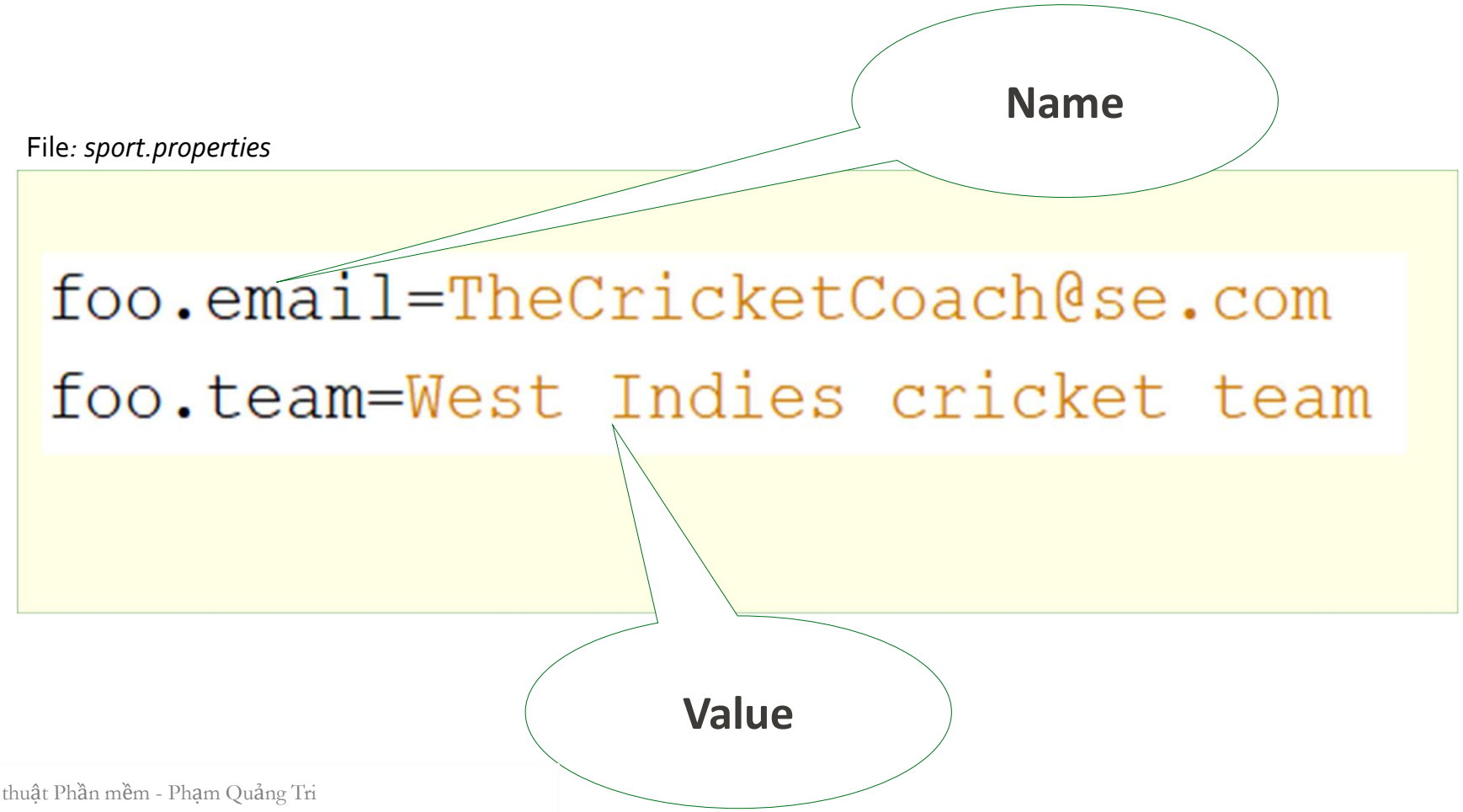



```
public class CricketCoach implements Coach {
    private String emailAddress;
    private String team;
    private FortuneService fortuneService;
```

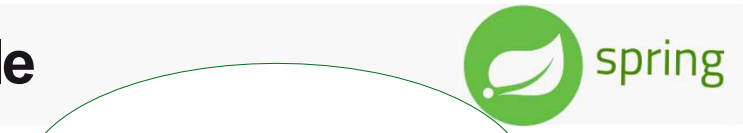
Development Process – Literal Values Injection (properties file)

- Create properties file
- Load Properties File in Spring config file
- Reference values from Properties File

Step 1: Create Properties File



Step 2: Load Properties File in Spring config file



File name

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.se.spring...o.libs.CricketCoach"
      <property name="fortuneService" ref="myFortune" />
      <property name="emailAddress" value="${foo.email}" />
      <property name="team" value="${foo.team}" />
 <!-- <property name="emailAddress" value="TheGoodCoach@se.com " />
      <property name="team" value="India National" />-->
</bean>
```

```
foo.email=TheCricketCoach@se.com
foo.team=West Indies cricket team
```

QUESTIONS

Bean Scopes

- Scope refers to the lifecycle of a bean
- How long the bean will live
- How many instances are created
- How is the bean shared in the spring environment