Spring Framework Dependency Injection (DI) Basics

"Code with Passion!"

Topics

- What is and Why Spring Bean?
- What is and Why Dependency Injection (DI)?
- Two DI variants
- Reading configuration
- Bean configuration
- Bean parameter types
- Auto-wiring of beans
- Auto-scanning of beans
- Bean naming

What is and Why Spring Bean?

What is a Spring Bean?

- In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans
  - A bean is an object that is instantiated, assembled, and managed by a Spring IoC container
- Any normal Java POJO class can be a Spring Bean if it's configured to be initialized via container by providing configuration metadata information

Why Spring Bean?

- Many features of Spring such as persistence, security, transaction, controller functionality, view selection, etc. are possible only because Spring IoC container can intercept the calls to these beans the container manages these beans

What is and Why Dependency Injection (DI)?

What is Dependency Injection (DI)?

- Also know as Inversion of Control (IoC) – technically DI is one of IoC schemes but for all practical purpose, you can think of DI and IoC same thing

- "Hollywood Principle"
  - Don't call me, I'll call you ("DI Container" is the agent)
- "DI Container" resolves dependencies of components by wiring/injecting dependency objects (push)
  - As opposed to a component looks for and instantiates dependency objects (pull)
- Termed by Martin Fowler

Why Dependency Injection?

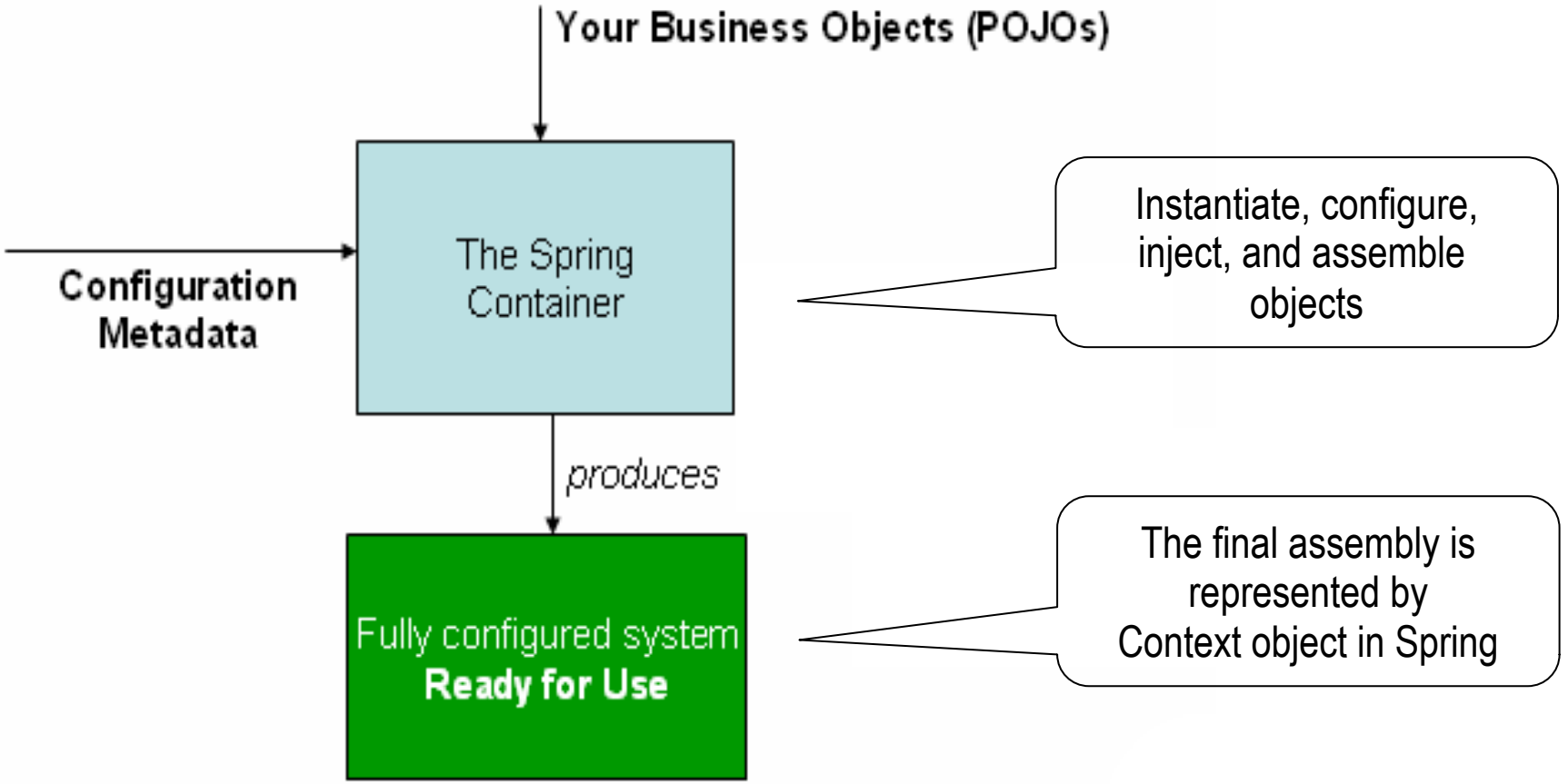
- Target component and dependencies are loosely coupled
  - ✓ No need to have dependency creation/lookup code in the target component
- Target components are testable as POJO's
  - During testing, mock dependency objects can be injected by testing framework
  - During production environment, real dependency objects are injected by spring DI container
- Target component and dependencies can be configured
  - Instantiation and wiring of target component and dependencies can be done through configuration (as in the case of Spring framework)

DI Configuration

- DI container gets its instructions on what objects to instantiate, configure, inject, and assemble by reading configuration metadata

- The configuration metadata is represented in XML and/or Java configuration
  - XML is no longer supported from Spring 5

Spring DI Container



Two Dependency Injection Variants

Two Dependency Injection Variants

- Constructor dependency Injection
  - Dependencies are injected through constructors of a component
- Setter dependency injection
  - Dependencies are injected through setter methods of a component

Constructor Dependency Injection

```
public class ConstructorInjection {

    private Dependency dep;

    public ConstructorInjection(Dependency dep) {
        this.dep = dep;
    }
}
```

Setter Dependency Injection

```
public class SetterInjection {

    private Dependency dependency;

    public void setDependency(Dependency dependency) {
        this.dependency = dependency;
    }
}
```

Reading Configuration

DI Container Java Interfaces in Spring

- *org.springframework.beans.factory.BeanFactory*
  - Root interface for accessing a Spring bean container
- *org.springframework.context.ApplicationContext*
  - Sub-interface of BeanFactory
  - Adds easier integration with Spring's AOP features; message resource handling (for use in internationalization), event publication; and application-layer specific contexts such as the WebApplicationContext for use in web applications

Reading Configuration

- #1: Reading Java configuration file
- #2: Reading Java configuration file with Spring Boot class

#1: Reading Java Configuration via AnnotationConfigApplicationContext class

```
@Configuration
@Import(BeanConfiguration.class)
public class MainApplication {

    public static void main(String[] args) {

        // AnnotationConfigApplicationContext accepts annotated classes
        // as input - in particular @Configuration-annotated classes
        ApplicationContext context
                = new AnnotationConfigApplicationContext(MainApplication.class);

        Person person = context.getBean(Person.class);
        System.out.println(person.getName());
    }
}
```

#2: Reading Java Configuration via SpringApplication class

```
import org.springframework.boot.SpringApplication;

@Configuration
@Import(BeanConfiguration.class)
public class MainApplication {

    public static void main(String[] args) {

        // SpringApplication bootstraps and launches a Spring application
        // from a Java main method
        ApplicationContext context
                = SpringApplication.run(MainApplication.class, args);

        Person person = context.getBean(Person.class);
        System.out.println(person.getName());
    }
}
```

Bean Configuration

Beans

- The term “bean” is used to refer any component managed by Spring
- Properties of beans may be simple values or more likely references to other beans
- Beans can have multiple names

DI Parameter Types

Injection Parameter Types

- Spring supports various kinds of injection parameters
  1. Simple values
  2. Beans
  3. Collections
- You can use these types for both setter or constructor injections

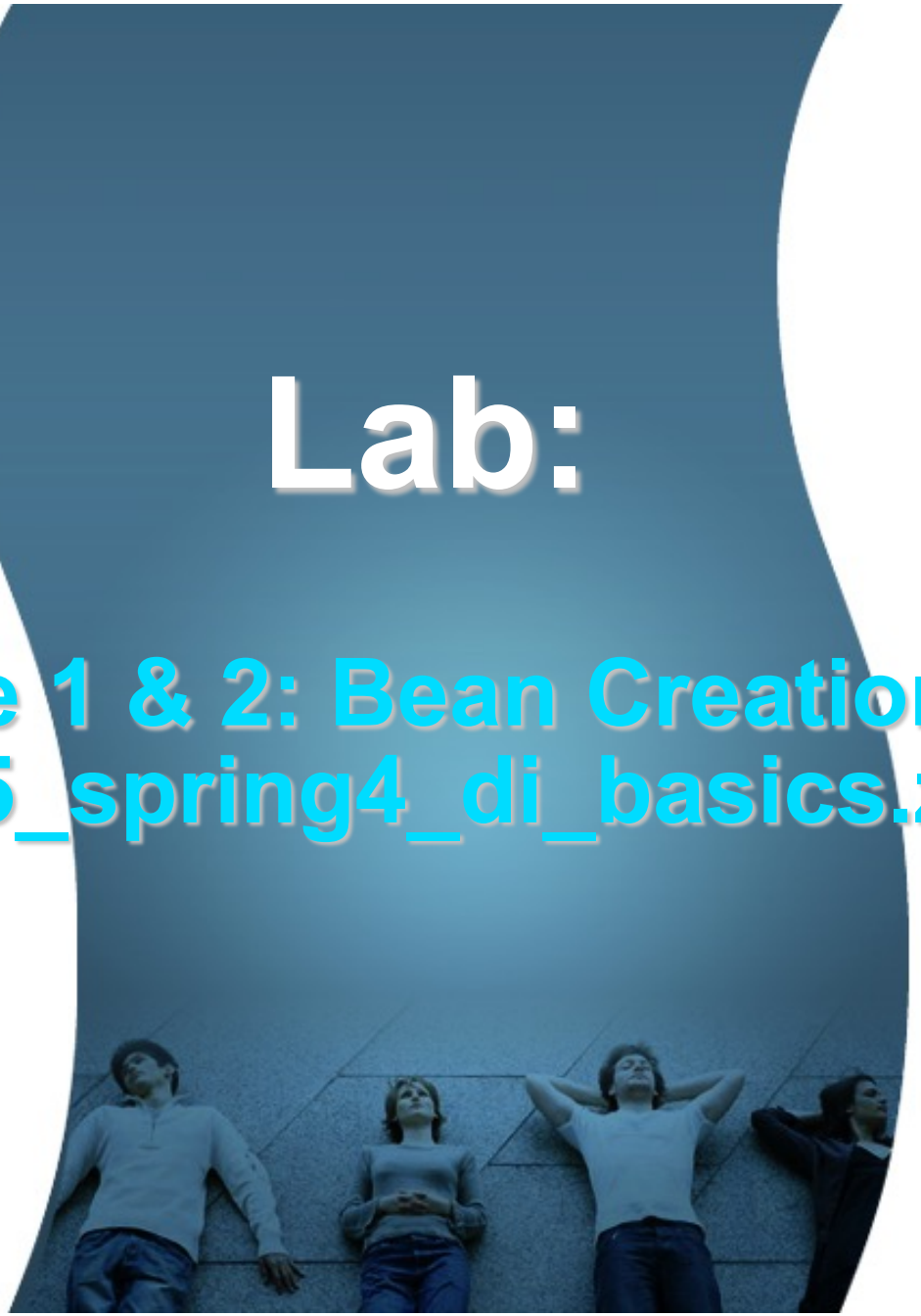
1.b Injecting Simple Values (Java)

```
@Configuration
public class BeanConfiguration {

    @Bean
    public Person getPerson() {
        Person person = new Person();
        person.setName("John Smith");
        person.setAge(85);
        person.setHeight(1.99F);
        person.setIsProgrammer(true);
        return person;
    }

}
```

(We are going to learn @Configuration and @Bean annotations in detail in next presentation: Spring DI annotation)

Lab:

Exercise 1 & 2: Bean Creation 1 & 2
4935_spring4_di_basics.zip



Injecting Dependency Beans

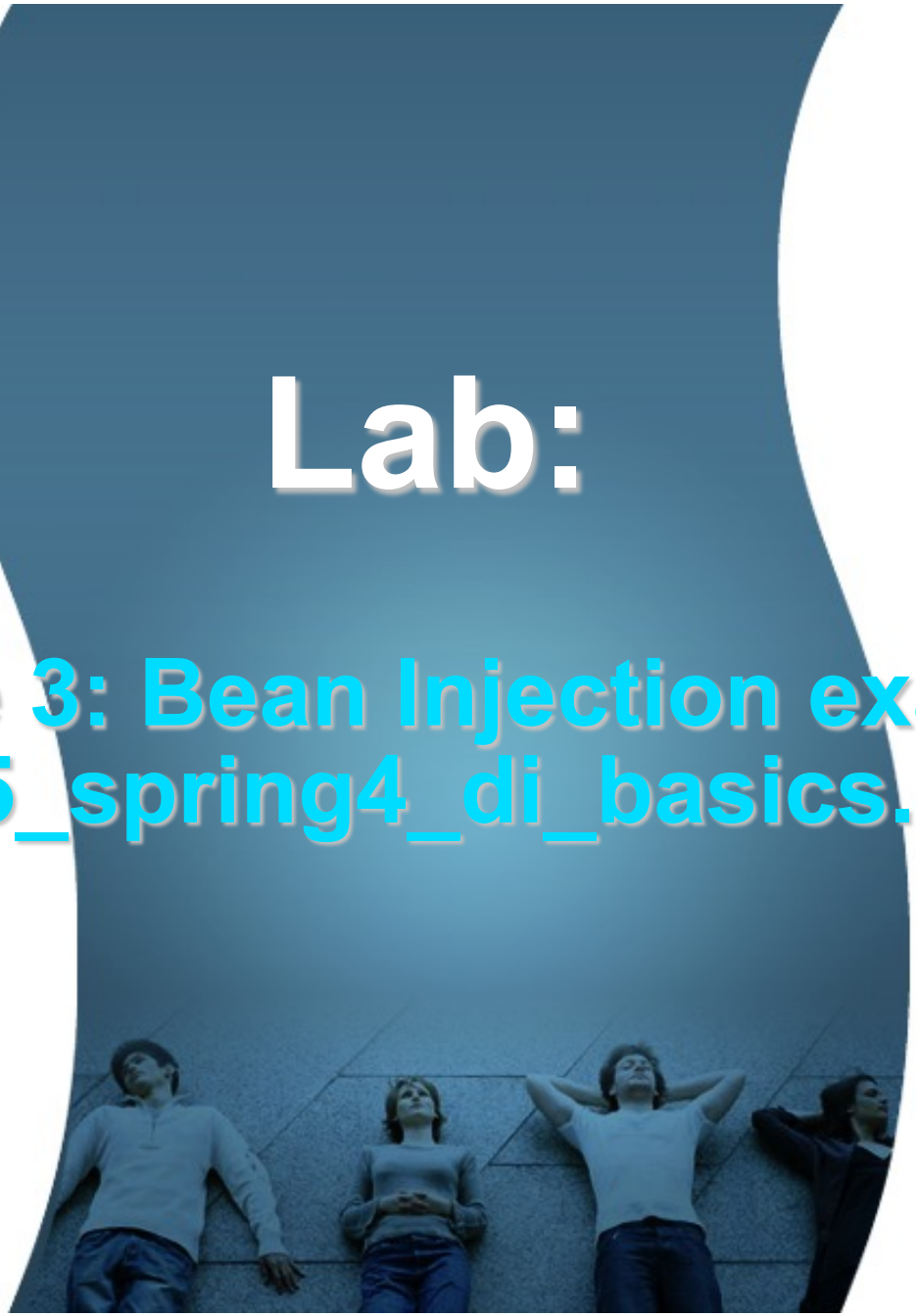
- Used when you need to inject a dependency bean into another (target bean)

2.b Injecting Beans: Example (Java)

```
@Configuration
public class MyConfiguration {

    @Bean
    public Address getAddress(){
        Address address = new Address();
        return address;
    }

    @Bean
    public Person getPerson(Address address) {
        Person person = new Person(address);
        return person;
    }

}
```

Lab:

Exercise 3: Bean Injection examples
4935_spring4_di_basics.zip



Autowiring of Beans

@Autowired

- Can be used in the target bean's Java source code for specifying DI requirement
- (We will cover @Autowired a bit more detail in next presentation – Spring DI annotation)

Lab:

Exercise 4: Autowiring
4935_spring4_di_basics.zip

Autoscanning of Beans

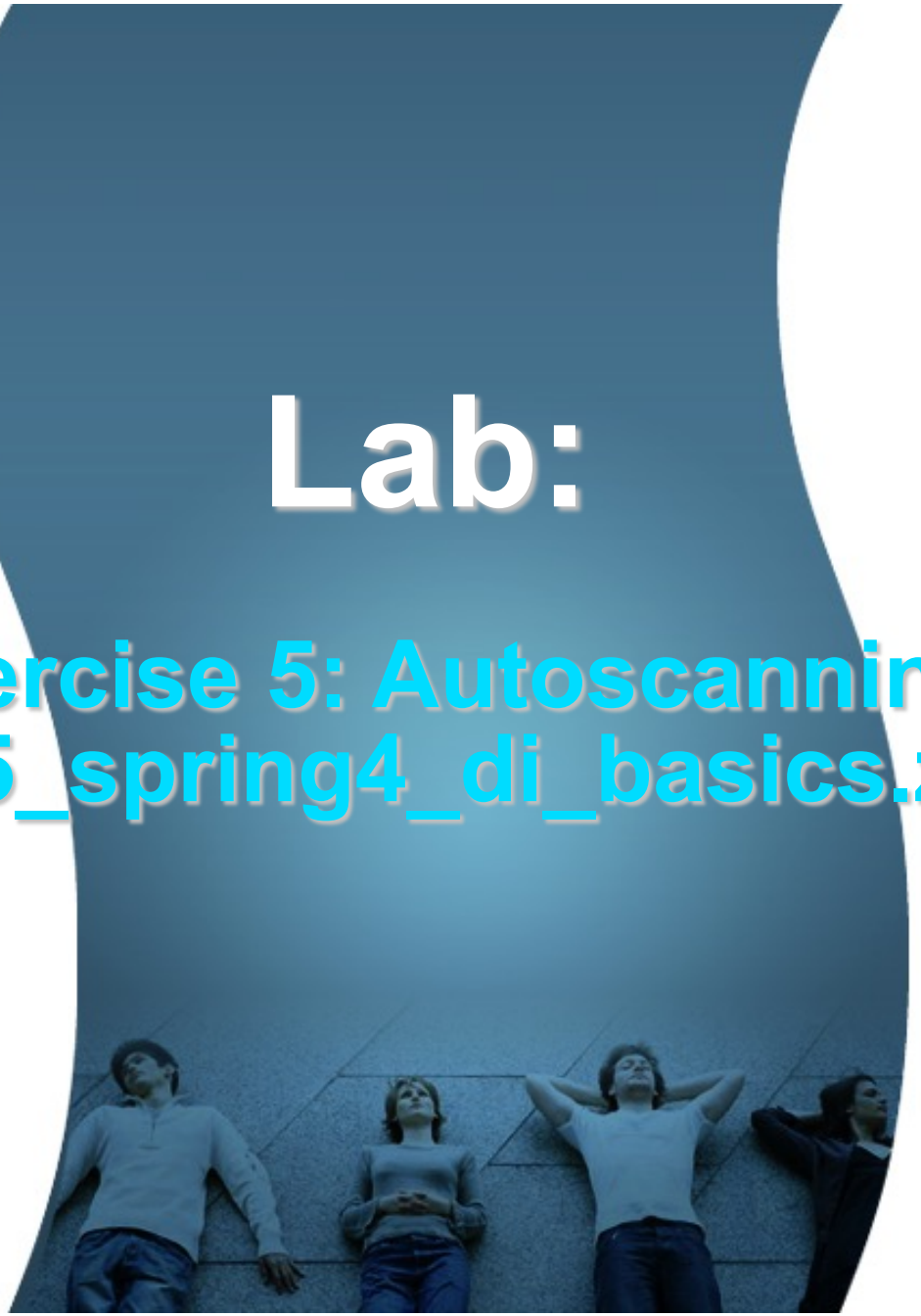
Autoscanning configuration in (Java)

- Any bean annotated with @Component under the "com.jpassion.di" package will be auto-detected and their instances will be created by the Spring framework
- No need to have @Bean configuration for beans
- We will cover autoscanning in more detail in next presentation (Spring DI annotation)

```
@Configuration
@ComponentScan(basePackages = {"com.jpassion.di"})
public class MainApplication {

    public static void main(String[] args) {

        ApplicationContext context = SpringApplication.run(MainApplication.class,
                args);

        Person person = context.getBean(Person.class);
        System.out.println(getPersonInfo(person));
    }
}
```

Lab:

Exercise 5: Autoscanning 4935_spring4_di_basics.zip



Bean Naming

Bean Naming

- Each bean must have at least one name that is unique within the containing BeanFactory
- Name resolution procedure
  - If a *bean*has an *id* attribute, the value of the *id* attribute is used as the name
  - If there is no *id* attribute, Spring looks for *name* attribute
  - If neither *id* nor *name* attribute are defined, Spring use the *class* name as the name
- A bean can have multiple names
  - Specify comma or semicolon-separated list of names in the name attribute

Bean Naming Example (Java)

```
@Configuration
public class BeanConfiguration {

    @Bean(name={"name1", "name2", "name3", "name4"})
    public Person getPerson() {
        Person person = new Person();
        return person;
    }

}
```

Lab:

Exercise 6: Bean naming
4935_spring4_di_basics.zip

Code with Passion!