Bài 13: Spring Basic

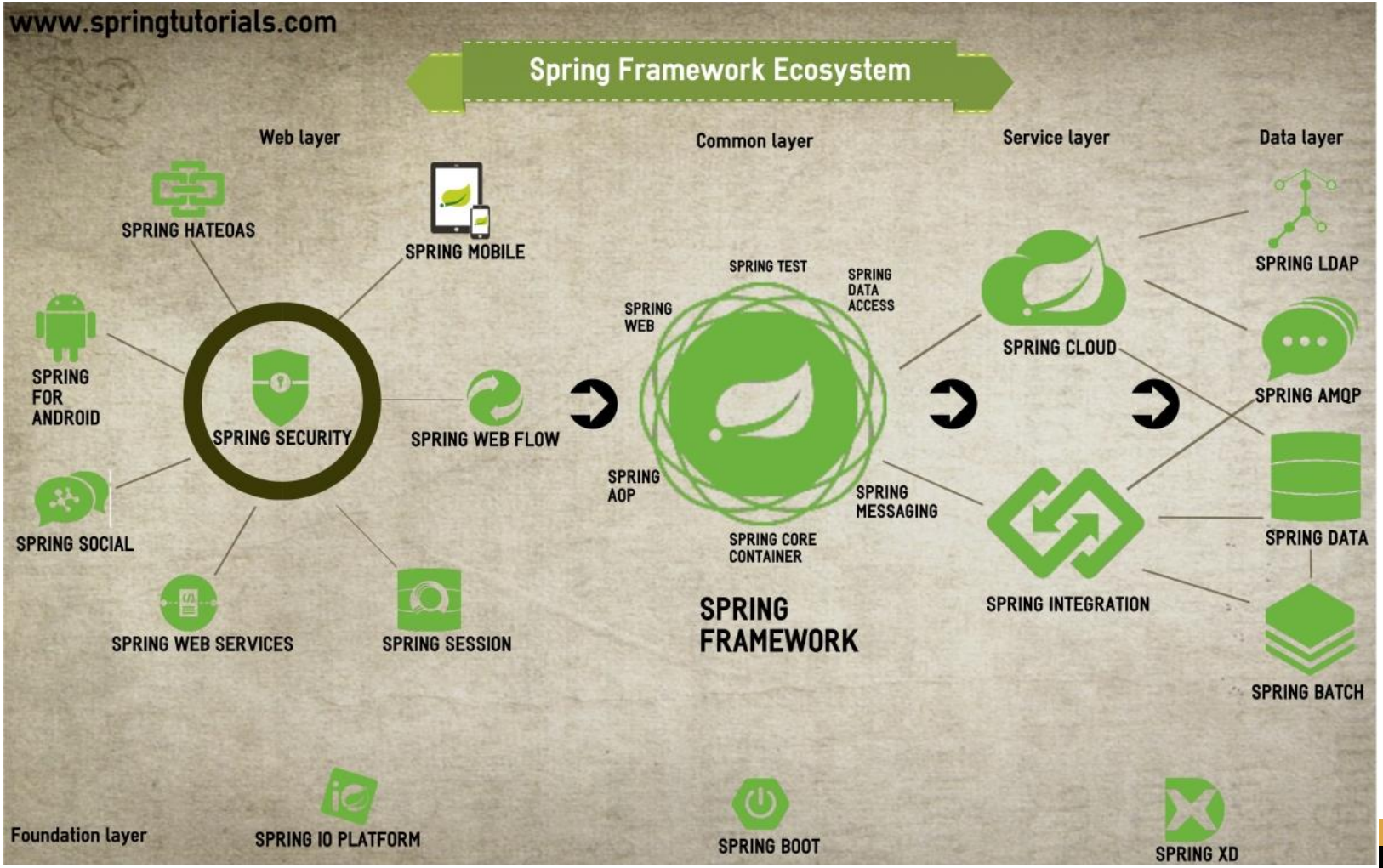
Giảng viên: Nguyễn Quang Huy
Email: HuyNQ12@topica.edu.vn

Spring - Overview

- "Spring" means different things in different contexts
- The Spring Framework is divided into modules, applications can choose which modules they need.
- The Spring Framework provides foundational support for different application architectures
  - messaging
  - transactional data and persistence
  - web
  - servlet-based Spring MVC web framework
  - Spring WebFlux reactive web framework
- Came into being in 2003 as a response to the complexity of the early J2EE specifications
- Integrates with carefully selected individual specifications from the EE umbrella: Servlet API (JSR 340), WebSocket API (JSR 356), Concurrency Utilities (JSR 236), JSON Binding API (JSR 367), Bean Validation (JSR 303), JPA (JSR 338), JMS (JSR 914), Dependency Injection (JSR 330), Common Annotations (JSR 250)
- As of Spring Framework 5.0, Spring requires JDK 8+ (Java SE 8+), Spring requires the Java EE 7 level (e.g. Servlet 3.1+, JPA 2.1+) as a minimum

Spring - Design Philosophy

- ❑ Provide choice at every level
- ❑ Accommodate diverse perspectives
- ❑ Maintain strong backward compatibility
- ❑ Care about API design
- ❑ Set high standards for code quality



Spring - The IoC container - Container overview

- IoC (Inversion of Control) is also known as dependency injection (DI), it is a process whereby
  - **objects** define their dependencies (the other objects they work with)
    - through constructor arguments
    - arguments to a factory method
    - or properties that are set on the object instance after it is constructed
    - or returned from a factory method
  - the container then **injects** those dependencies when it creates the bean
- In Spring, the **objects** that form the backbone of your application and that are managed by the Spring IoC **container** are called **beans**
- The interface *org.springframework.context.ApplicationContext* represents the Spring IoC container and is responsible for instantiating, configuring, and assembling the aforementioned beans

XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- services -->

    <bean id="petStore"
class="org.springframework.samples.jpetstore.services.PetStoreServiceImpl">
        <property name="accountDao" ref="accountDao"/>
        <property name="itemDao" ref="itemDao"/>
        <!-- additional collaborators and configuration for this bean go here -->
    </bean>

    <!-- more bean definitions for services go here -->

</beans>
```

XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="accountDao"
        class="org.springframework.samples.jpetstore.dao.jpa.JpaAccountDao">
        <!-- additional collaborators and configuration for this bean go here -->
    </bean>

    <bean id="itemDao" class="org.springframework.samples.jpetstore.dao.jpa.JpaItemDao">
        <!-- additional collaborators and configuration for this bean go here -->
    </bean>

    <!-- more bean definitions for data access objects go here -->

</beans>
```

```
// create and configure beans
ApplicationContext context = new ClassPathXmlApplicationContext("services.xml", "daos.xml");

// retrieve configured instance
PetStoreService service = context.getBean("petStore", PetStoreService.class);

// use configured instance
List<String> userList = service.getUsernameList();
```

Spring - The IoC container - Bean overview

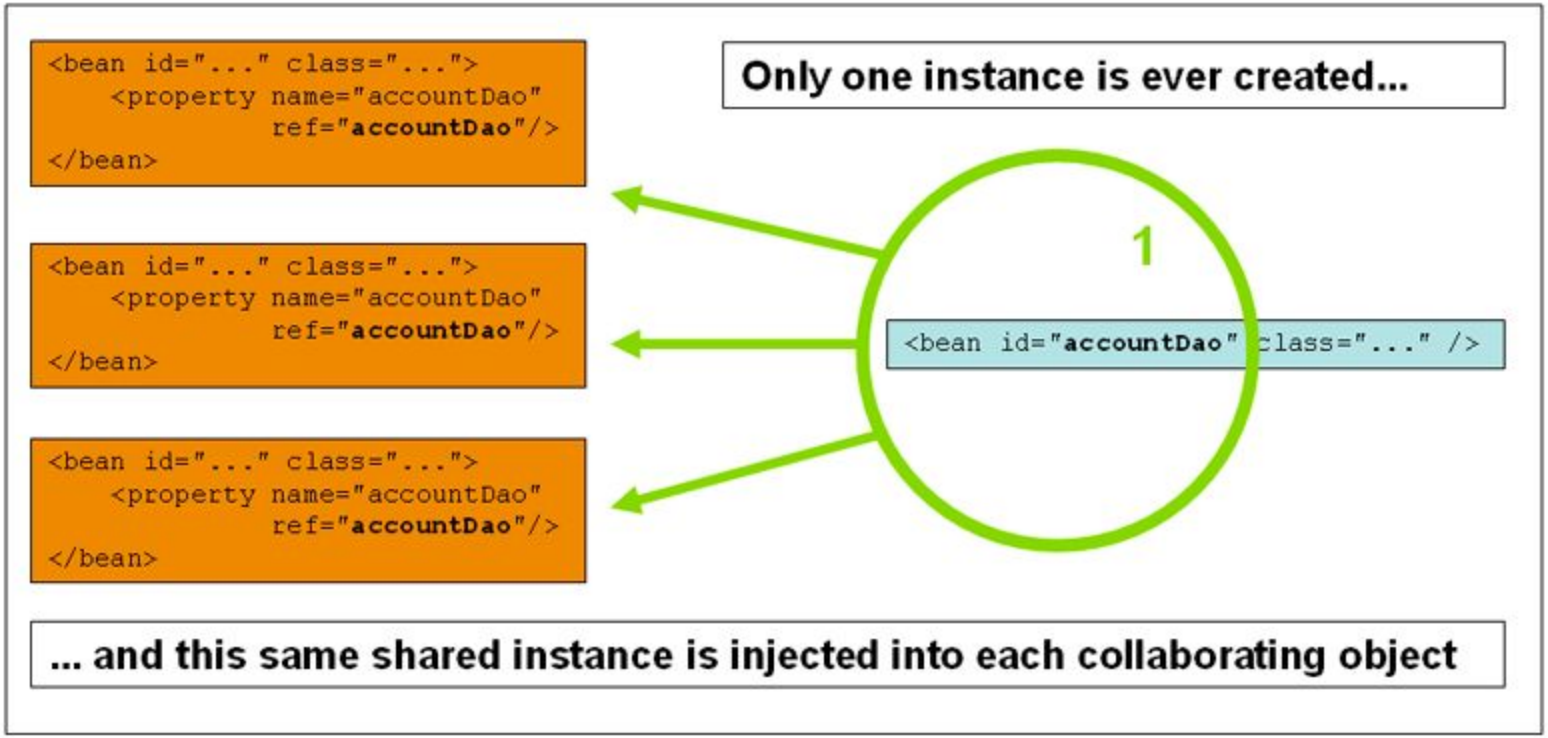
- Bean definitions are represented as *BeanDefinition* objects, which contain (among other information) the following metadata
  - *A package-qualified class name:* typically the actual implementation class of the bean being defined
  - Bean behavioral configuration elements, which state how the bean should behave in the container (scope, lifecycle callbacks, and so forth)
  - References to other beans that are needed for the bean to do its work; these references are also called *collaborators* or *dependencies*
  - Other configuration settings to set in the newly created object, for example, the number of connections to use in a bean that manages a connection pool, or the size limit of the pool
- Every bean has one or more identifiers. These identifiers must be unique within the container that hosts the bean

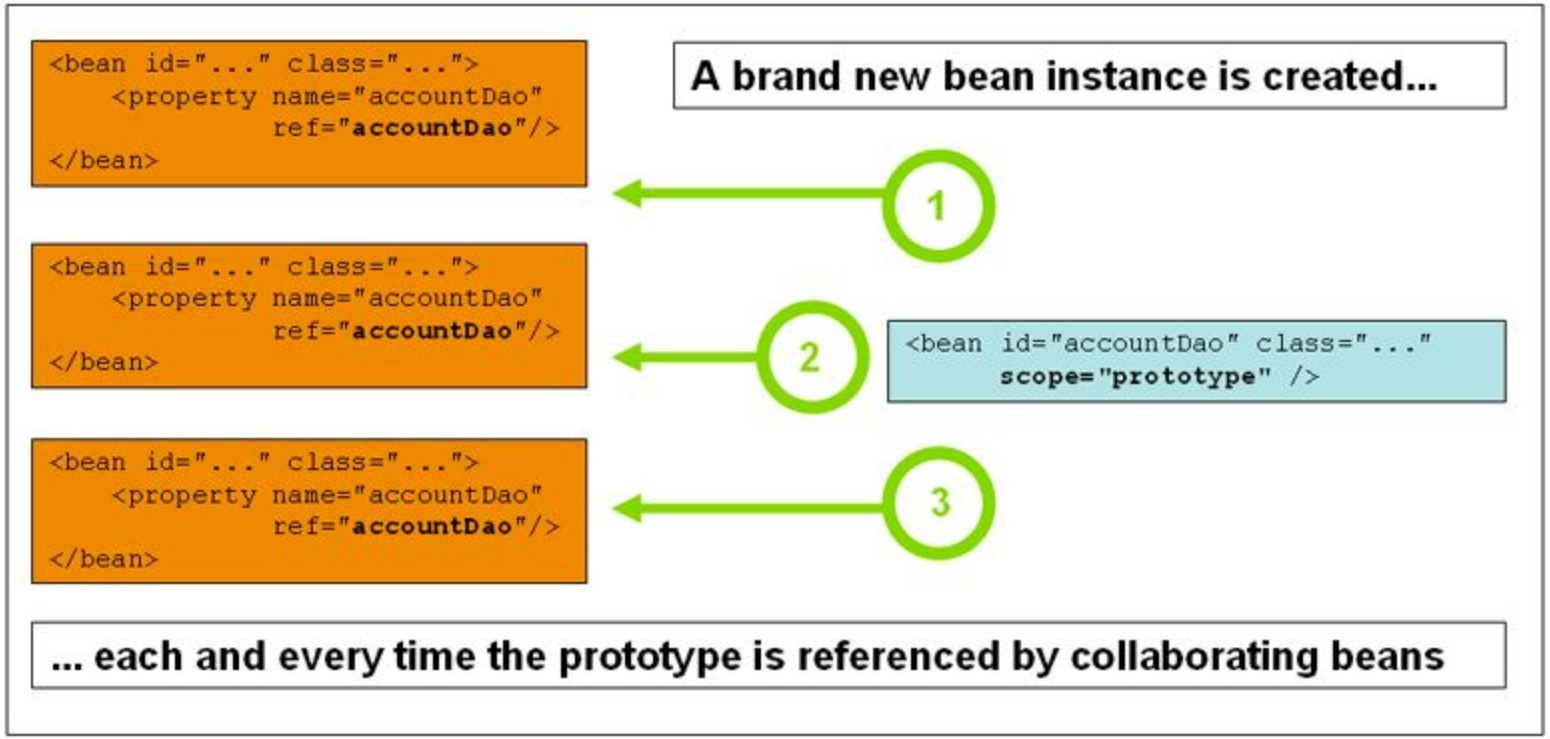
Table 1. The bean definition

Property	Explained in…
class	Instantiating beans
name	Naming beans
scope	Bean scopes
constructor arguments	Dependency Injection
properties	Dependency Injection
autowiring mode	Autowiring collaborators
lazy-initialization mode	Lazy-initialized beans
initialization method	Initialization callbacks
destruction method	Destruction callbacks

Table 3. Bean scopes

Scope	Description
singleton	(Default) Scopes a single bean definition to a single object instance per Spring IoC container.
prototype	Scopes a single bean definition to any number of object instances.
request	Scopes a single bean definition to the lifecycle of a single HTTP request; that is, each HTTP request has its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring `ApplicationContext`.
session	Scopes a single bean definition to the lifecycle of an HTTP `Session`. Only valid in the context of a web-aware Spring `ApplicationContext`.
application	Scopes a single bean definition to the lifecycle of a `ServletContext`. Only valid in the context of a web-aware Spring `ApplicationContext`.
websocket	Scopes a single bean definition to the lifecycle of a `WebSocket`. Only valid in the context of a web-aware Spring `ApplicationContext`.





Spring - The IoC container - Dependency Injection

- Code is cleaner with the DI principle and decoupling is more effective when objects are provided with their dependencies
- The object does not look up its dependencies, and does not know the location or class of the dependencies
- DI exists in two major variants
  - Constructor-based dependency injection
  - Setter-based dependency injection
- The Spring container can autowire relationships between collaborating beans


DEMO
delete

Spring - Resources

- Built-in Resource implements
  - UrlResource
  - ClassPathResource
  - FileSystemResource
  - ServletContextResource
  - InputStreamResource
  - ByteArrayResource

On the other hand, you may also force `ClassPathResource` to be used, regardless of the application context type, by specifying the special `classpath:` prefix:

```
Resource template = ctx.getResource("classpath:some/resource/path/myTemplate.txt");
```

Similarly, one can force a `UrlResource` to be used by specifying any of the standard `java.net.URL` prefixes:

```
Resource template = ctx.getResource("file:///some/resource/path/myTemplate.txt");
```

```
Resource template = ctx.getResource("http://myhost.com/resource/path/myTemplate.txt");
```

Table 10. Resource strings

Prefix	Example	Explanation
classpath:	`classpath:com/myapp/config.xml`	Loaded from the classpath.
file:	`file:///data/config.xml`	Loaded as a `URL`, from the filesystem. [3]
http:	`http://myserver/logo.png`	Loaded as a `URL`.
(none)	`/data/config.xml`	Depends on the underlying `ApplicationContext`.

Spring - Validation, Data Binding, and Type Conversion

```
public class PersonForm {
    private String name;
    private int age;
}
```

JSR-303 allows you to define declarative validation constraints against such properties:

```
public class PersonForm {

    @NotNull
    @Size(max=64)
    private String name;

    @Min(0)
    private int age;

}
```

```java
public class PersonValidator implements Validator {

    /**
     * This Validator validates *just* Person instances
     */
    public boolean supports(Class clazz) {
        return Person.class.equals(clazz);
    }

    public void validate(Object obj, Errors e) {
        ValidationUtils.rejectIfEmpty(e, "name", "name.empty");
        Person p = (Person) obj;
        if (p.getAge() < 0) {
            e.rejectValue("age", "negativevalue");
        } else if (p.getAge() > 110) {
            e.rejectValue("age", "too.darn.old");
        }
    }
}
```

Spring - Spring Expression Language (SpEL)

- A powerful expression language that supports querying and manipulating an object graph at runtime
- DEMO

Spring - Aspect Oriented Programming with Spring

- Next course

Spring - Null-safety

- Next course

Spring - Data Buffers and Codecs

- Next course


Q&A

Task

- Build an application use Spring Framework 5.x (Sring boot 2.x)
  - Business requirements:
    - Allow user search by type in Vietnamese
    - Allow search by “tiếng Việt có dấu hoặc không dấu”
  - Technical requirement
    - Use IoC, Resources, Validation, Type Conversion, SpEL
  - Bonus: build application use **Spring MVC** and use **Spring MVC Validation**