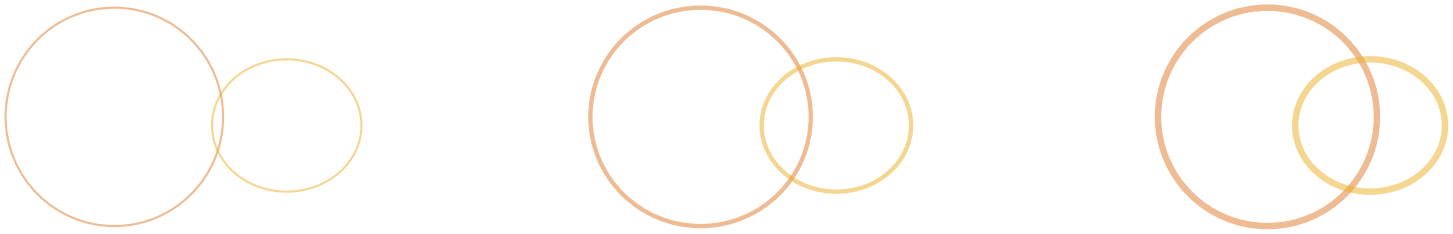
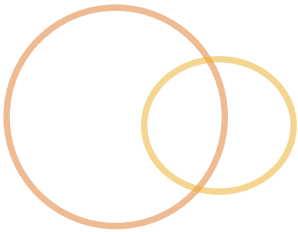


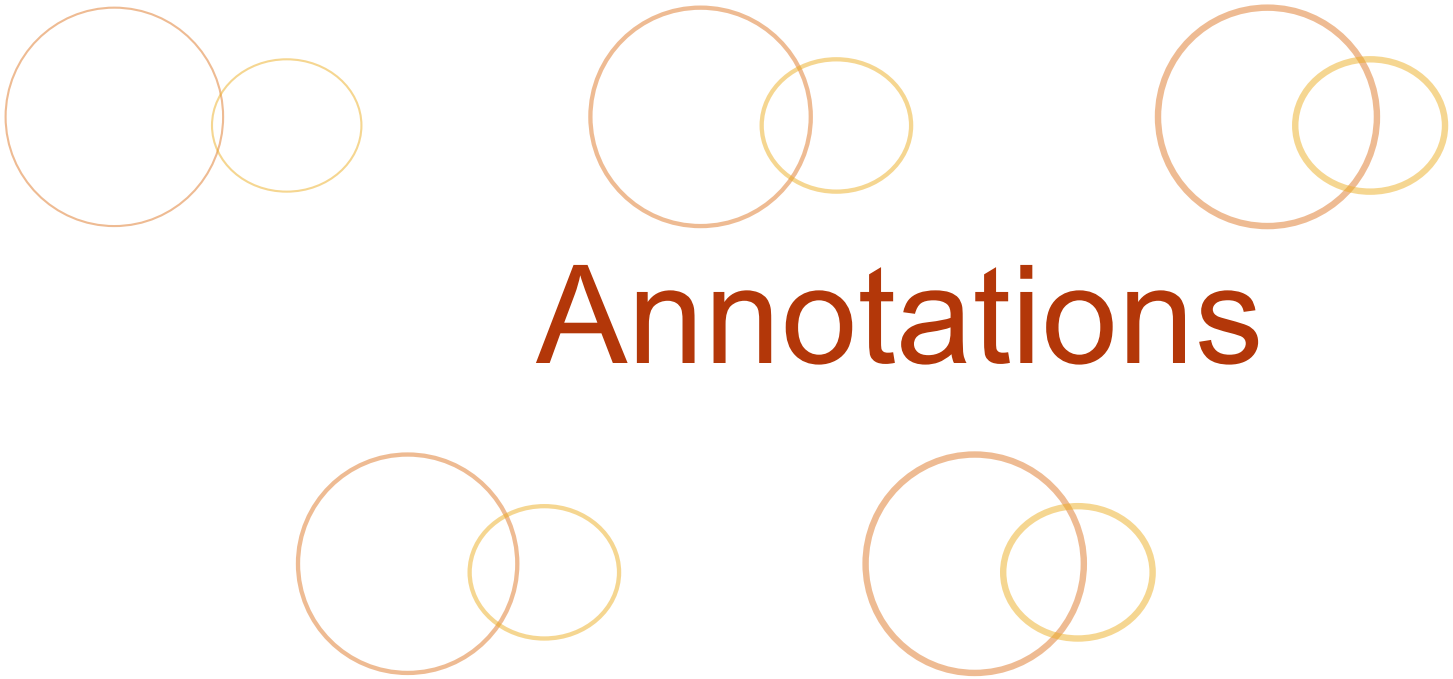
Annotations in Spring

Objectives



When we are done, you should be able to:

- Configure Spring Beans using Annotations
- Know when to use annotations
- Understand what JSR-330 is



Annotations

A Different Configuration Mechanism

- Can be used instead of or in addition to XML
  - Added in Spring 2.5 to compliment XML configuration
  - By Spring 3.s and 4.x, mostly 'replaced' by Java configuration which uses annotations as well

What is still needed in 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"
    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.x

    <context:component-scan base-package="bookshelf"/>
</beans>
```

- `<context:component-scan>` needed or annotations are ignored
- `base-package` limits the scope of where the runtime looks for the components

Configuration Annotations

- `@Component`
  - Defines this as a Spring Bean
  - Name of bean is name of class with a lower-case first letter
- `@Autowired`
  - Defines how to wire the beans
  - Is a required dependency
  - On constructor, defines that the arguments passed into the constructor should be retrieved from the context
  - On method, defines that the parameters should be retrieved from the context
  - On field, uses reflection to set values

Using Annotations - Constructor

```
import org.springframework.stereotype.Component;
import org.springframework.beans.factory.annotation.Autowired;

@Component
public class LibraryServiceImpl implements LibraryService {

  private BookSource bookSource;

  @Autowired
  public LibraryServiceImpl(BookSource source){
    bookSource = source;
  }
}
```

Using Annotations – Set Method

```
import org.springframework.stereotype.Component;
import org.springframework.beans.factory.annotation.Autowired;
import javax.sql.DataSource;

@Component
public class BookSourceImpl {
  private DataSource data;

  public BookSourceImpl(){ }

  @Autowired
  public void setDataSource(DataSource dataSource){
    data = dataSource;
  }
}
```

Using XML to Configure

```
<bean id="library" class=”com.example.LibraryServiceImpl ”>
  <constructor-arg ref="bookSource"/>
</bean>
<bean id="bookSource" class="com.example.BookSourceImpl">
  <property name="dataSource" ref="dataSource"/>
</bean>
```

<context:annotation-config>

- Looks for any dependency injection annotation
- Requires that the bean still be defined
  - <property> and <constructor-config> not needed

```
<?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">

      <context:annotation-config/>
      <bean id="library" class="java.example.LibraryServiceImpl"/>
</beans>
```

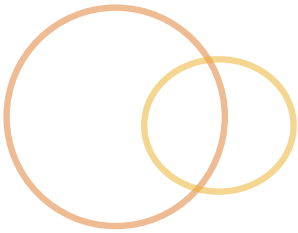
<context:component-scan>

- Looks for @Component (or subsets) and automatically loads them
- Does not require the bean to be defined
  - Can limit what is scanned by using `base-package`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ...>

      <context:component-scan base-package="..."/>
</beans>
```

@ComponentScan



- In `JavaConfig` file, we can define component-scanning
  - Will scan application and automatically load all classes annotated with `@Component` or its subsets
  - Can limit scanning to specific packages
  - Will also pick up other configuration files that are annotated with `@Configuration`

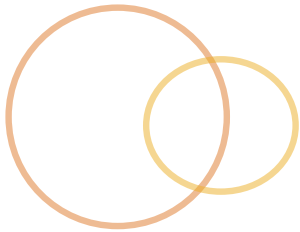
```
@ComponentScan(basePackages={"package.one", "package.two"})
@Configuration
public class JavaConfig {

}
```

@PostConstruct / @PreDestroy

- Add behavior to startup and shutdown of application
  - `@PostConstruct` runs after all dependency injections and before bean is ready to be used
  - `@PreDestroy` runs when object is being removed from the container
  - Both can annotate methods that
    - Take no arguments
    - Return void
    - Are not static
    - Do not throw checked exceptions
- Mark life-cycle methods

More @Bean



- Can define initialization and destruction methods

```
@Configuration
public class JavaConfig {

  @Bean(initMethod="setup" destroyMethod="shutdown")
  public LibraryService libraryService(){
    return new LibraryServiceImpl(bookSource());
  }

  private void setup(){
    // setup something
  }

  private void shutdown() {
    // shutdown something
  }
}
```

JSR-330

- Attempt to standardize many frameworks' annotations
- Also called `@Inject`
  - Package is `javax.inject`
- Often replaces the Spring annotations mentioned
  - `@Inject` – most of `@Autowired`
  - `@Named` – `@Component`
  - `@Scope` – defines new scope
  - `@Singleton` – similar to Spring's prototype scope

Note: Generally, use one or the other unless one gives distinct features that the other doesn't

Lab 4 – Update Phone Service