

Spring

Miroslav Blaško, Bogdan Kostov, Martin Ledvinka

KBSS FEL CTU in Prague

Winter Term 2022



Contents

- 1 Introduction
- 2 Dependency Injection - Revisited
- 3 Spring Beans
- 4 Spring Transaction Management
 - Proxy Design Pattern
- 5 Other Commonly Used Spring Features
 - Demo E-Shop Application
- 6 Tasks



Introduction



Seminar Topic

In this seminar we will learn to use Spring and its main features:

- Dependency Injection (DI)
- Transaction management



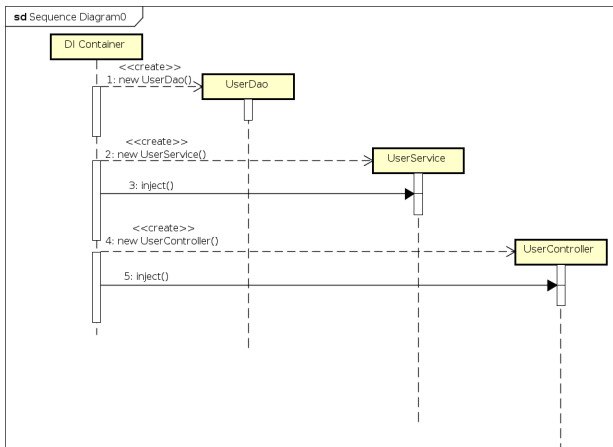
Dependency Injection - Revisited



Definition and Sequence Example

Dependency Injection

Component lifecycle is controlled by the *container* which is responsible for delivering correct implementation of the given dependency.



Plain Java code vs DI

```
package cz.cvut.kbss.ear.spring_example;
import ...

public class SchoolInformationSystem {

    private CourseRepository repository
        = new InMemoryCourseRepository();

    public static void main(String[] args) {
        SchoolInformationSystem main = new SchoolInformationSystem();
        System.out.println(main.repository.getName());
    }
}
```

The client code (`SchoolInformationSystem`) itself decides which repository implementation to use

- change in **implementation** requires *client code* change.
- change in **configuration** requires *client code* change.



DI using Annotations

SchoolInformationSystem.java

```
package cz.cvut.kbss.ear.spring_example;
import ...

@Component
public class SchoolInformationSystem {
    @Autowired
    private CourseRepository repository;
}
```

CourseRepository.java

```
package cz.cvut.kbss.ear.spring_example;
public interface CourseRepository {
    public String getName() { return name; }
}
```

InMemoryCourseRepository.java

```
package cz.cvut.kbss.ear.spring_example;
import ...

@Component
public class InMemoryCourseRepository
    implements CourseRepository {
    public String getName() { return
        "In-memory course repository"; }
}
```



Spring Beans



Bean Declaration

Bean declaration tells Spring from which classes to create beans. We will learn about two ways of bean declaration:

- Bean creation through annotated classes
- Bean creation with a factory method

Spring needs to know where to look for bean declarations. With Spring Boot, it scans the package of the main application class and all its sub-packages.



Bean Declaration - Annotated Class

A bean can be declared using an annotation on a class. Annotations used for declaration of beans in this way are:

- @Component
- @Configuration
- @Repository
- @Service
- etc.

Code example:

```
package cz.cvut.kbss.ear.spring_example;
import ...

@Component
public class InMemoryCourseRepository implements CourseRepository {
    public String getName() { return "In-memory course repository"; }
}
```



Bean Declaration - Factory method

A bean factory method should be implemented in a configuration bean. The method should be annotated with the `@Bean` annotation and it should return the bean. The method's parameters will be injected if possible. Code example:

```
@Configuration // is this a bean? Yes it is.
public class RepositoryConfiguration{
    @Bean
    public InMemoryCourseRepository createInMemoryRepository() {
        return new InMemoryCourseRepository();
    }
}
```



Spring Bean Scopes

Scopes define the life cycle of a bean

singleton a single bean instance per a Spring IoC container (the default scope)

prototype a new bean instance every time when requested (e.g., injected during creation of another bean)

request a single bean instance per an HTTP request

session a single bean instance per an HTTP session

globalSession a single bean instance per a global HTTP session

Code example specifying the scope of a bean:

```
@Component
@Scope("singleton")
public class SchoolInformationSystem {
    @Autowired
    private CourseRepository repository;
}
```

Spring allows custom scope definition (e.g. JSF 2 Flash scope)



Bean Injection

During creation, beans are injected with the declared dependencies (must be also beans). To declare a dependency in Spring use the annotation:

- `@Autowired`

Code example:

```
package cz.cvut.kbss.ear.spring_example;
import ...

@Component
public class SchoolInformationSystem {
    @Autowired
    private CourseRepository repository;
}
```



Injecting Entity Manager

The entity manager is injected using the `@PersistenceContext` annotation (JPA).

Code example:

```
@Repository  
public class CartDao {  
    @PersistenceContext  
    private EntityManager em;  
    ...  
}
```



Spring Transaction Management



Transaction Management Annotations

Spring provides means to declaratively manage business transactions. The following annotations work in conjunction with JPA.

- `@Transactional` - on methods and classes, wraps a method in a transaction
- `@EnableTransactionManagement` - enabling declarative transaction support, enabled by default in Spring Boot

Code example:

```
@Service
public class CartService {
    @Transactional
    public void addItem(Cart cart, CartItem toAdd) {
        ...
    }
}
```



Proxy Design Pattern

Question Is the class of `CartService` **bean** `CartService`?



Proxy Design Pattern

Question Is the class of `CartService` **bean** `CartService`?

Answer No, it is not. It is a subclass of `CartService`.



Proxy Design Pattern

Question Is the class of `CartService` bean `CartService`?

Answer No, it is not. It is a subclass of `CartService`.

Proxy Design Pattern

Spring implements transaction management using the Proxy Design Pattern on beans. The `CartService` is sub-classed in the background to enable wrapping transactional method calls with code managing the transactions.



Proxy Design Pattern - Java Code Example

Calculator.java

```
public class Calculator{
    public int add(int a, int b){
        return a + b;
    }

    public int subtract(int a, int b){
        return a - b;
    }
}
```

CalculatorLoggerProxy.java

```
public class CalculatorLoggerProxy extends
    Calculator{
    private static final Logger LOG ...
    @Override
    public int add(int a, int b){
        int ret = super.add(a,b);
        LOG.debug("{} + {} = {}", a, b, ret);
        return ret;
    }

    @Override
    public int subtract(int a, int b){
        int ret = super.subtract(a,b);
        LOG.debug("{} - {} = {}", a, b, ret);
        return ret;
    }
}
```

Some observations:

- CalculatorLoggerProxy is also a Calculator
- Extends execution by adding pre- and post-processing code



Other Commonly Used Spring Features



Annotation based Spring Configuration

- `@ComponentScan` searching for Spring beans among classes in a given package
 - `@ComponentScan` without argument scans the current package and all its sub-packages
- `@Import` composing Spring configuration

Searching for beans:

```
@Configuration
@ComponentScan(basePackages = "cz.cvut.kbss.ear.eshop.dao")
public class PersistenceConfig {
    ...
}
```

Importing configuration:

```
@Configuration
@Import({PersistenceConfig.class})
public class AppConfig {
    ...
}
```



Spring Configuration in E-shop

The E-Shop application is implemented using Spring Boot, which has many benefits over plain Spring:

- Provides a single Maven dependency which includes profiles of different spring packages commonly used together
- Provides a simpler Maven build configuration
 - A Maven plugin which builds the application into a JAR with an embedded application server for rapid deployment
- Requires minimal application configuration compared to Spring
 - Contains a lot of sensible defaults which *just work* in most cases



Tasks



Syncing Your Fork

- 1 Ensure you have no uncommitted changes in the working tree
 - `git status` → your branch is up to date, nothing to commit
- 2 Fetch branches and commits from the upstream repository (EAR/B221-eshop)
 - `git fetch upstream`
- 3 Check out the task branch from **upstream**
 - `git checkout upstream/b221-seminar-05-task`
- 4 You are now in a detached state, create a branch for this task
 - `git branch b221-seminar-05-task`
- 5 Do the task
- 6 Commit and push the solution to **your** fork
 - `git push -u origin b221-seminar-05-task`



Task 1 – Configuration of Persistence Layer (1 point)

- 1 Declare missing bean declarations and injections.
 - Some of the classes in the `dao` package should be declared as beans but they are not. Declare them properly.
 - In the `dao` package, there is also one dependency injection which is not declared properly. Fix it.
 - **Hint:** `@Repository`, `@PersistenceContext`
- 2 Create a prototype bean of type `java.util.Date`.
 - **Hint:** `@Configuration`, `@Bean`
 - **Hint:** Use tests to help you debug the issues.
 - **Acceptance criteria:** All enabled tests are passing.



Task 2 – Implementation of a Service (1 point)

- 1 Remove the `@Disabled` annotation from `CartServiceTest` and verify that tests are now failing
- 2 Implement `CartService` that allows to
 - Add specific items to a cart
 - Remove specific items from a cart
 - Amount of products available in stock is correctly adjusted after every add/remove operation
 - `CartService` class must be declared as a Spring bean
 - Inject beans necessary to implement the service methods (DAO)
- 3 Make sure that service methods are transactional
- 4 **Hint:** `@Service`
- 5 **Acceptance criteria:** Transactional processing is configured properly and all tests are passing.



The End



The End

Thank You



Resources

- <https://docs.spring.io/spring-boot/docs/current/reference/html/>

