

Spring

Miroslav Blaško, Bogdan Kostov

miroslav.blasko@fel.cvut.cz

bogdan.kostov@fel.cvut.cz

Winter Term 2019



Contents

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



Introduction



Seminar Topic

In this seminar we will learn to use Spring's:

- Dependency Injection (DI) functionality,
- Transactional management



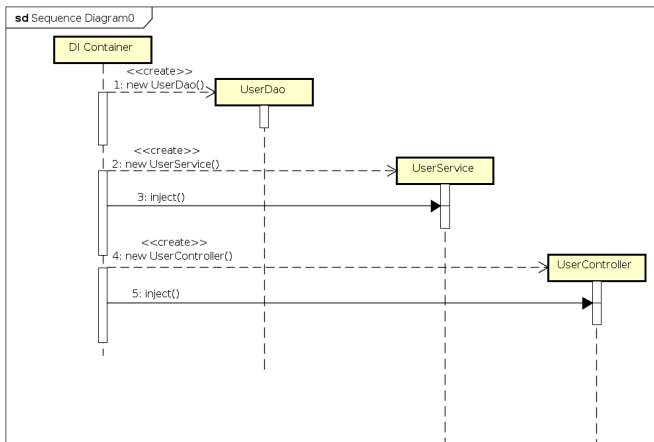
Dependency Injection - Revisited



Definition and Sequence Example

Dependency Injection

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



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"; }
}
```



Proxy Design Pattern

Question Is the class of
`SchoolInformationSystem.repository`
`InMemoryCourseRepository`?



Proxy Design Pattern

Question Is the class of
`SchoolInformationSystem.repository`
`InMemoryCourseRepository`?

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



Proxy Design Pattern

Question Is the class of
`SchoolInformationSystem.repository`
`InMemoryCourseRepository`?

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

Proxy Design Pattern

Spring implements Dependency Injection using the Proxy Design Pattern on Beans. The `InMemoryCourseRepository` is inherited in the background to enable DI.



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



Spring beans



Bean Declaration

Bean declaration tells Spring how to create a bean. We will learn about two ways of bean creation:

- Bean creation through class constructor
- Bean creation with a factory method



Bean Declaration - Class Constructor

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

- @Component
- @Configuration
- @Repository
- @Service

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 methods input 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();
    }
}
```



Bean Injection

During creation beans are injected with declared dependencies. 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;
}
```



Spring Bean Scopes

Scopes define the life cycle of a bean

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

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

request a single bean instance per HTTP request

session a single bean instance per HTTP session

globalSession a single bean instance per 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 Disambiguation

To resolve a bean dependency spring looks for beans in the application context which will satisfy that dependency.

SchoolInformationSystem.java

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

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

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 Transactional Management



Transaction Management Annotations

The following annotations work with conjunction with JPA.

- `@Transactional` - on methods and classes, wraps a method in a transaction
- `@EnableTransactionManagement` - enabling transactional support, not needed in Spring Boot

Code example:

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



Injecting Entity Manager

The following annotations work with conjunction with JPA.

- `@Transactional` - on methods and classes, wraps a method in a transaction
- `@EnableTransactionManagement` - enabling transactional support, not needed in Spring Boot

Code example:

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



Other Commonly Used Spring Features



Annotation based Spring Configuration

- `@ComponentScan` searching for spring beans among classes in a given package
- `@Import` composing spring configuration

Code example searching for beans:

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

Code example importing configuration.

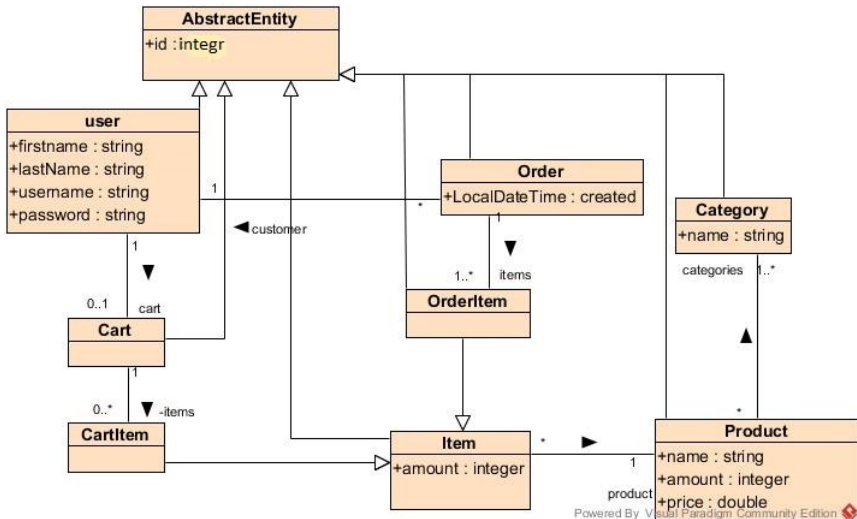
```
@Configuration
@Import({PersistenceConfig.class})
public class AppConfig {
    ...
}
```



Demo E-Shop Application



JPA Model



Powered By Visual Paradigm Community Edition



Tasks



Syncing Your Fork

- 1 Make sure your local copy git repository is configured correctly¹ and all changes are committed (git status - your branch is up to date, nothing to commit).
- 2 Fetch branches and commits from the upstream repository (EAR/B191-eshop)
 - `git fetch upstream`
- 3 Check out local branch corresponding to the task branch
 - `git checkout -b b191-seminar-04-task`
- 4 Merge changes from the corresponding upstream branch
 - `git merge upstream/b191-seminar-04-task`
- 5 Do your task
- 6 Push the solution to your fork
 - `git push origin b191-seminar-04-task`

¹see seminar page 18 (Task for 1 point) and page 19 (Syncing Your Fork)



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 with class `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 `@Ignore` 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 should be correctly adjusted during each add/remove operation
- 3 Make sure that service methods are transactional
- 4 **Acceptance criteria:** Transactional processing is configured properly and all tests are passing.



The End Thank You

