POLITECHNIKA GDAŃSKA

Spring Platform

Internet Services Architectures

Michał Wójcik, Waldemar Korłub

Spring Platform

Spring refers to an entire family of projects built on top of a Spring Framework project:

- Spring Boot,
- Spring Framework,
- Spring Data,
- Spring Cloud,
- Spring Cloud Data Flow,
- Spring Security,
- Spring Session,
- Spring Integration,
- Spring HATEOAS,
- Spring REST Docs,
- Spring Batch,
- Spring AMQP,
- Spring for Android,
- Spring CredHub,
- Spring Flo,
- Spring for Apache Kafka,
- Spring LDAP,
- Spring Mobile,
- Spring Roo,
- Spring Shell,
- Spring Statemachine,
- Spring Vault,
- Spring Web Flow,
- Spring Web Services.

Spring Framework

Spring Framework - a framework that allows to create complex web and enterprise-class applications running on a Java virtual machine:

- supports container based dependency injection (DI),
- supports container based Inversion of Control (IoC),
- can be use to create desktop applications,
- name suggests fresh approach to business applications development,
- considered as competition to Java EE.

Spring Framework

Spring Framework history:

- 2002 *Expert One-on-One J2EE Design and Development* book by *Rod Johnson*,
- 2003 Spring first release,
- 2004 Spring 1.0,
- 2007 Spring 2.5,
- 2012 Spring 3.2,
- 2014 Spring 4.0,
- 2017 Spring 5.0.

Spring Framework

Spring Framework is developed in a modular architecture:

- each modules is responsible for different scope of haviours,
- some of them can be used independently.

Some of the modules:

- `spring-core` - contains mainly core utilities and common stuff,
- `spring-context` - provides Application Context, that is Spring's DI container,
- `spring-mvc` - model-view-controller framework with requests to handlers dispatcher,
- `spring-data` - access to various data sources (SQL databases, NoSQL databases, etc.),
- `spring-security` - authorization and authentication mechanisms.

Spring Boot

Spring Boot - a mechanism facilitating the development of applications based on the Spring platform:

- simplifies projects configuration,
- matches compatible platform (and not only) elements (controls libraries versions),
- simplifies distribution package preparation:
  - web application require to be launched within web server in order to support HTTP requests,
  - allows to uses embedded Tomcat server configured on the project level.

Spring Boot Application

Instead of creating new projects from scratch and adding dependencies manually we can:

1. use Spring Initializr: http://start.spring.io/,
2. set GroupId i ArtifactId,
3. select desired modules (dependencies):
    - Web,
    - JPA,
    - Derby (database),
    - ...
4. generate projects,
5. download ZIP archive and unpack,
6. open project in favorite IDE.

Spring Boot Application

Main entry point:

```
@SpringBootApplication
public class SimpleRpgApplication {

    public static void main(String[] args) {
        SpringApplication.run(SimpleRpgApplication.class, args);
    }

}
```

Application can be started simply by:

```
java -jar simple-rpg.jar
```

It starts embedded Tomcat server and deploys the application.

Spring Context

Basic component types managed by the Spring container:

- `@Component` - most basic, generic component managed by the container,
- `@Controller` - specialized component to be used as controller in MVC framework,
- `@Repository` - DAO (data access object) in persistence layer,
- `@Service` - specialized component responsible for business logic.

By default all of those are realized as singletons (only one global instance in the container).

How to provide dependencies

There are three methods for dependency injection:

- using constructor arguments,
- using setters,
- directly into class field.

Injection with constructor

```
public class UserService {

    private UserRepository repository;

    public UserService(UserRepository repository) {
        this.repository = repository;
    }

}
```

```
UserRepository repository = //create notifier
UserService service = new UserService(repository);
```

Injection with constructor:

- immutable objects (no setters and final fields) can be created,
- construction and injection in single step.

Injection with setters

```
public class UserService {

    private UserRepository repository;

    public void setRepository(UserRepository repository) {
        this.repository = repository;
    }

}
```

```
UserService service = new UserService();
UserRepository repository = //create notifier
service.setRepository(repository);
```

Injection with setters:

- the default constructor is present,
- dependency can be replaced after injection,
- setters is required (no immutable objects with final fields).

Field injection

```
public class UserService {

    private UserRepository repository;

}
```

```
UserService service = new UserService();
Field field = service.getClass().getDeclaredField("repository");
field.setAccessible(true);
UserRepository repository = //create notifier
field.set(service, repository);
```

Field injection:

- no additional methods,
- requires reflection mechanism.

DI in Spring

Dependency Injection in Spring Container:

- supports all three methods,
- declared with `@Autowired` annotation.

DI in Spring

```java
public class UserService {

    private UserRepository repository;

    @Autowired
    public UserService(UserRepository repository) {
        this.repository = repository;
    }

}
```

```java
public class UserService {

    private UserRepository repository;

    @Autowired
    public void setRepository(UserRepository repository) {
        this.repository = repository;
    }

}
```

```java
public class UserService {

    @Autowired
    private UserRepository repository;

}
```

Controlled bean life-cycle

In case of container controlled beans there is possibility to act on creation or destroy event.

```
@Component
public class DataStoreComponent {

    @PostConstruct
    public void init() throws Exception {
    //..
    }

    @PreDestroy
    public void clean() throws Exception {
    //..
    }

}
```

Initialization:

- constructor should be used only for creating object, no logic initialization,
- `@PostConstruct` methods are called after object creation and after all dependencies injection.

Command Line Application

Command line applications in Spring Boot can be achieved with components implementing `CommandLineRunner` interface:

```
@Component
public class CommandLine implements CommandLineRunner {

    private UserService service;

    @Autowired
    public CommandLine(UserService service) {
        this.service = service;
    }

    @Override
    public void run(String... args) throws Exception {
        service.findAll().forEach(System.out::println);
    }

}
```

Helpful resources

- Spring Team, *Spring Quickstart Guide*, https://spring.io/quickstart.
- Spring Team, *Spring Guides*, https://spring.io/guides.
- Baeldung, *Spring Tutorial*, https://www.baeldung.com/spring-tutorial.
- Baeldung, *Learn Spring Boot*, https://www.baeldung.com/spring-boot.
- Baeldung, *Spring Tutorials*, https://www.baeldung.com/category/spring/.
- Baeldung, *Spring Boot Tutorials*, https://www.baeldung.com/category/spring/spring-boot/.