The Spring Framework introduction

by

Martin Podolinský

(ByteSourceTechnologies Consulting GmbH)

Email: martin.podolinsky@bytesource.net
Twitter: @mpodolinsky

Agenda

- What is the Spring Framework?
- Dependency injection
- Spring configuration
- Spring AOP
- Testing support

java/j2ee Application Framework

What is the Spring Framework?

What is the spring framework?

[bytesource]

- Spring is a **Lightweight** Application Framework

  ...and is not an application server

- Springs adresses all tiers of an application - optionally

- Easy integrates 3rd party products and interfaces (JPA, Hibernate, web frameworks, WS, ...)

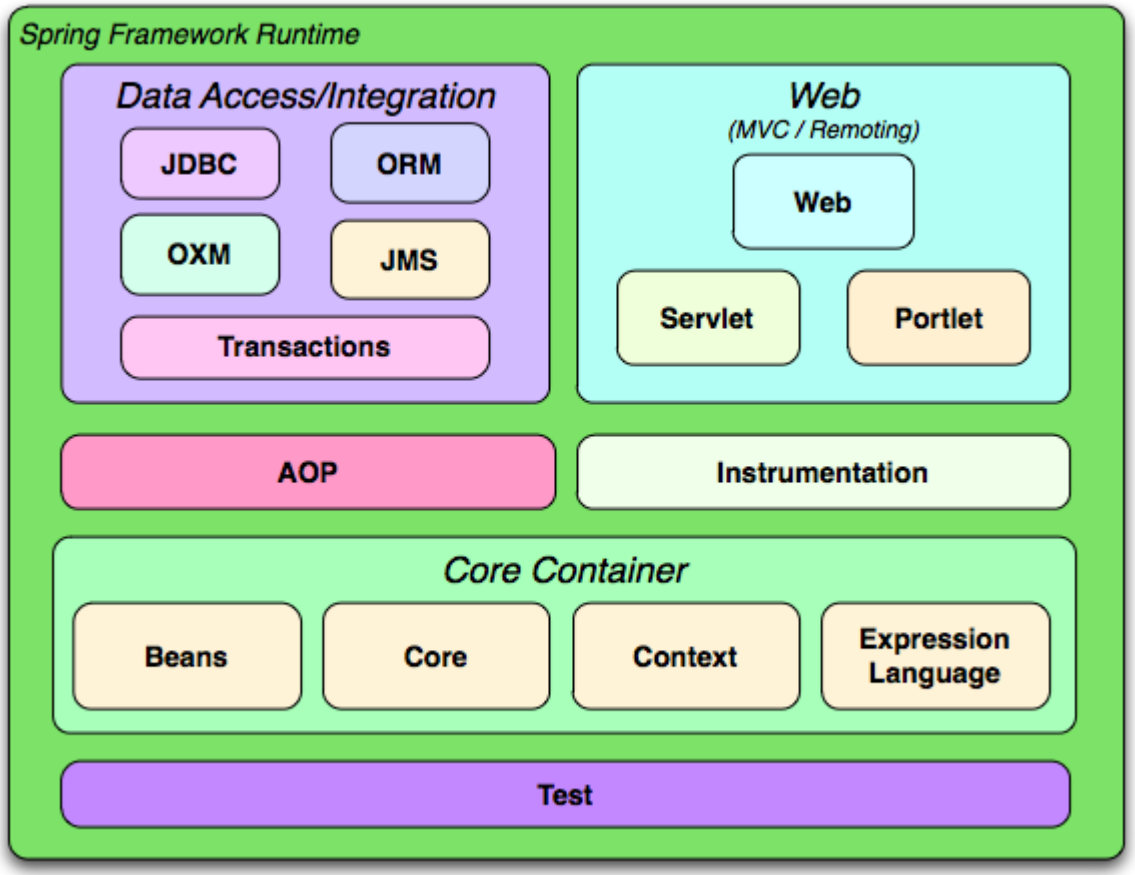
The Spring Framework history

- The first version by Rod Johnson as a framework around his book „Expert One-on-One J2EE Design and Development“ in October 2002
- The Spring 1.0 was released in March 2004 under the Apache 2.0 license
- 2005 become framework very popular and is emerged as a leading J2EE application framework
- 2006 won a Jolt productivity award and a JAX Innovation Award
- The current version is 3.0.x

Overview of the Spring Framework

[bytesource]

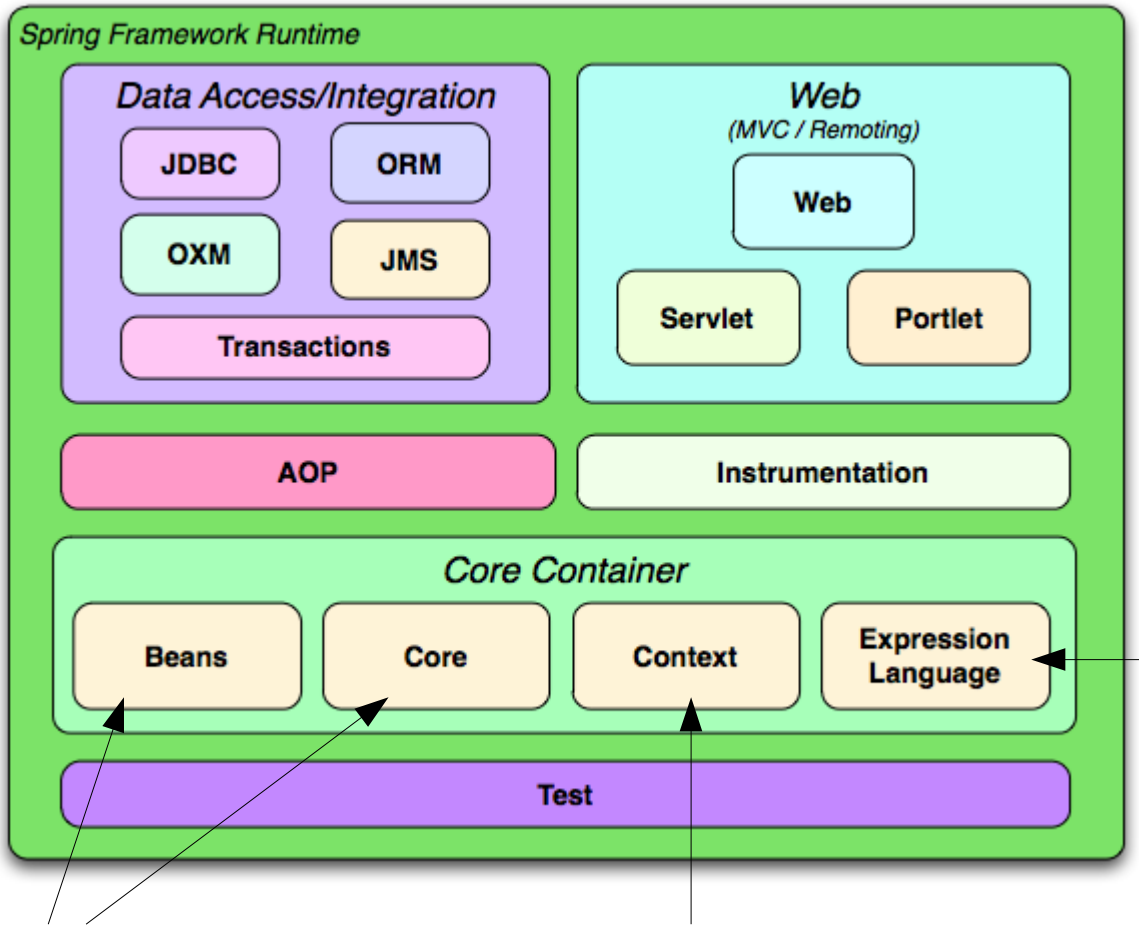
diagram from springsource.org



Overview of the Spring Framework

[bytesource]

diagram from springsource.org



Powerful language for querying and manipulating an object graph at runtime

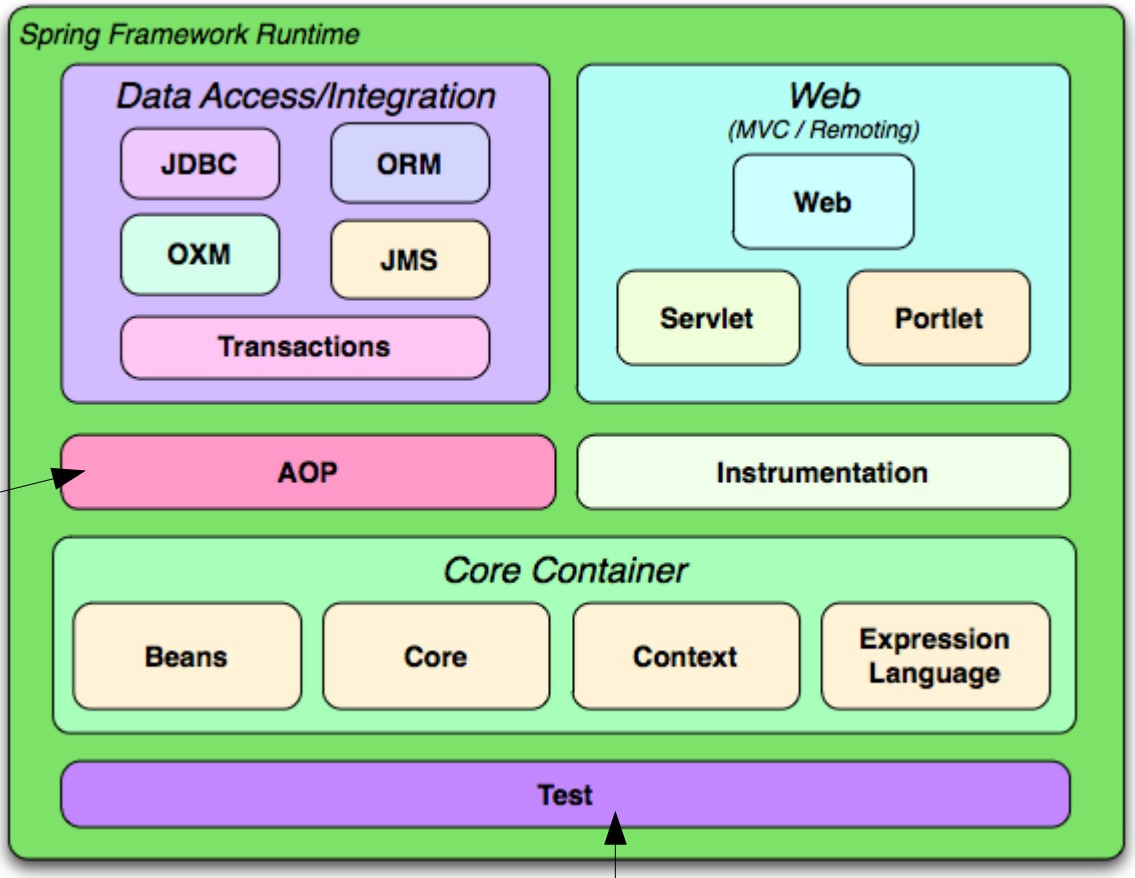
IoC and DI features based on the **BeanFactory** container concept

Build on the base of Beans and Core. Provide way to access objects, support for i18n, resource loading,...

Overview of the Spring Framework

[bytesource]

diagram from springsource.org



AOP support provided by Spring. Also possibility to use AspectJ

Support for testing Spring components using TestNG or Junit. Provides loading of Spring ApplicationContexts.

Overview of the Spring Framework

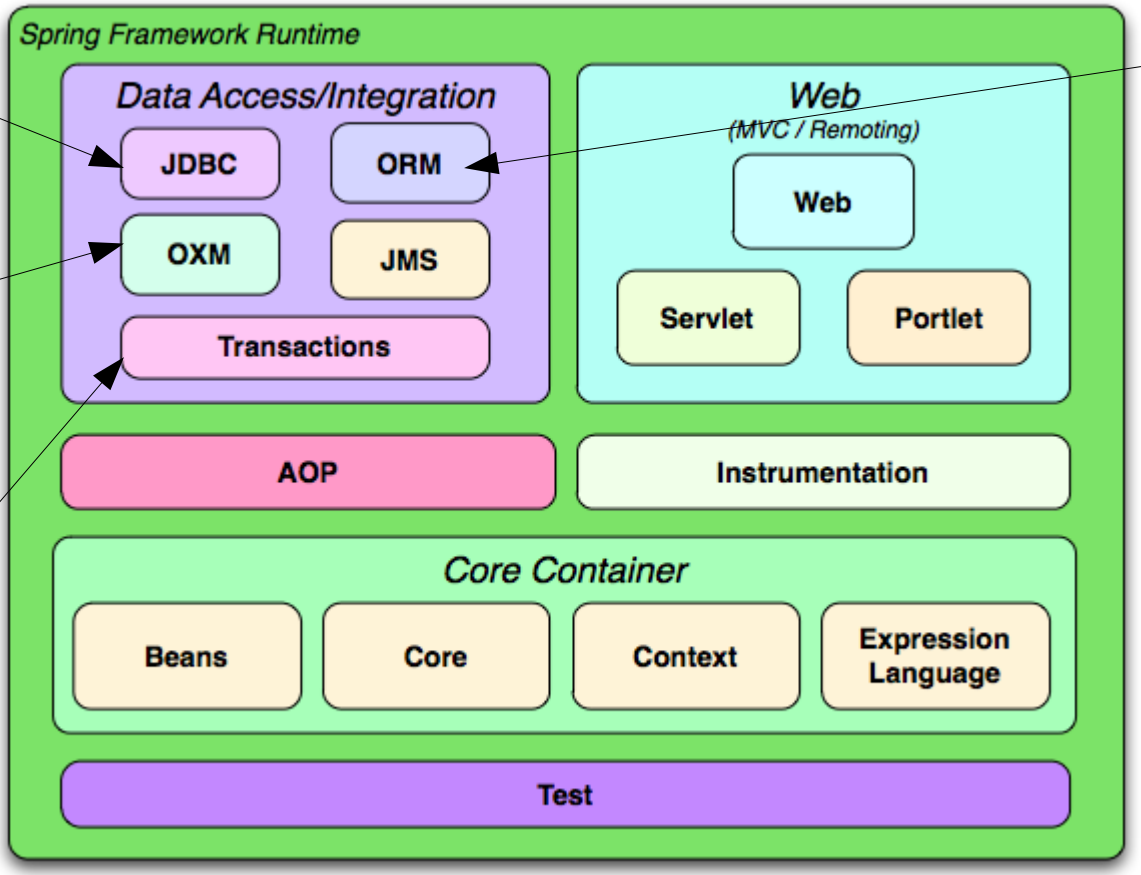
[bytesource]

diagram from springsource.org

JDBC – provides an abstraction layer

Object/XML mapping implementations like JAXB or XStream

Programatic or declarative **transaction management**.



ORM – provides integration layers for popular ORM APIs like JPA, Hibernate or iBatis. Support of declarative transaction management.

Spring is non-invasive

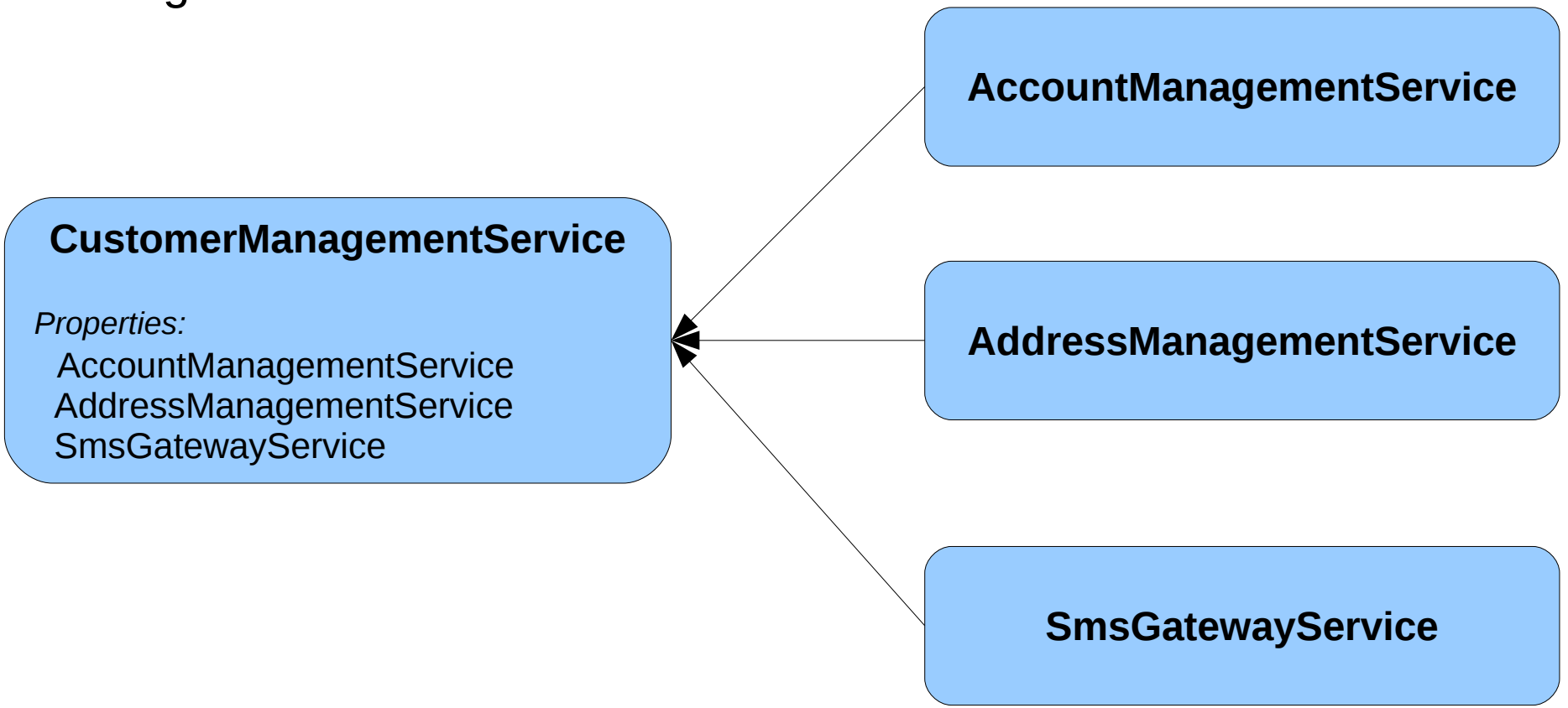
- Developer is not forced to introduce any framework-specific code into business/domain model.
  - Beans are JavaBean or POJOs and dependecies are „injected“ using setters or constructors
- Independence on a concrete framework gives better possibilities to reuse or test our components.

Dependency injection

Dependency injection

[bytesource]

- Implementation of the **Inversion of Control pattern**
- **BeanFactory** responsible for instantiating all components based on a configuration



Dependency injection

- No need for a component lookup code in our services
- Enables good reusability of our code
- Promotes good OOP eg. using **interfaces**
- Provides an easy way to support different environments during the project lifecycle (dev, QA, prod)
- Application will be extremly testable on a unit or component level (**testability is essential!**)

So again, what is really Spring?

So far we know,

- A Dependency Injection Container
- An AOP Framework
- A Service Abstraction Layer
  - Consistent integration with various standard and 3rd party APIs

Spring is a way how to write powerful, scalable and testable applications using POJOs

Spring configuration

Spring managed beans configuration

[bytesource]

Interfaces

```
public interface AuthenticationService { ... }
public interface CryptoService { ... }
```

Implementation

```
public class AuthenticationServiceImpl implements AuthenticationService {
    private CryptoService cryptoService;
    private String editModeGroupName;

    public void setCryptoService(CryptoService cryptoService) {
        this.cryptoService = cryptoService;
    }
    public void setEditModeGroupName(String editModeGroupName) {
        this.editModeGroupName = editModeGroupName;
    }
...
```

Spring managed beans configuration

[bytesource]

XML Configuration

```
<bean id="service.AuthenticationService"
    class="net.bytesource.rcrsdocreport.infrastructure.service.impl.AuthenticationServic
    eImpl">

   <property name="cryptoService" ref="cryptoService"/>

   <property name="editModeGroupName">

      <value>${group.edit}</value>

   </property>

</bean>

<bean id="cryptoService"
    class="net.bytesource.corporate.crypto.impl.CryptoServiceSymetricImpl">

      <constructor-arg index="0"><value>${cryptoSecret}</value></constructor-arg>

</bean>
```

Spring configuration options (I.)

[bytesource]

- **XML-based** configuration
- **Annotation-based** configuration
  - Classpath scanning and autodetecting of *@Component* (*@Repository*, *@Service*, *@Controller*)
  - Injecting via
    - *@Autowired* – spring annotation
    - *@Resource(name="componentName")* - JSR-250

Spring configuration options (II.)

[bytesource]

- Writing **configurations using Java**
  - *@Configuration* - class annotation indicates a source of beans definitions
  - @Bean is a method-level annotation and a direct analog of the XML <bean/> element. Provides equivalent parameters as the xml configuration (scope, qualifier, ...)
  - Offers a type safety and more flexibiliy than pure xml configuration

All configuration options can be combined so already existing spring code can be integrated without refactoring.

Beans wiring methods

[bytesource]

- Let Spring decide – **autowiring**
  - **byType** – there must be only one bean of the type ...

    ```
    public void setCryptoService(CryptoService cryptoService)
    ```
  - **byName** – there must be a bean with exact name ...

    ```
    public void setCryptoService(CryptoService cryptoService)
    ```
  - **Constructor** – analogous to byType, aplies to constructor args.

    ```
    public AuthenticationServiceImpl(CryptoService cryptoService)
    ```

- Wiring by defining bean's names

  recommended for larger deployments

  ```
  @Resource(name="myCryptoService")
  private CryptoService cryptoService;
  ```

Bean scopes

[bytesource]

Singleton – a single object instance per Spring IoC container

- Usually stateless beans like DAOs

Prototype – each request creates a new instance

- Statefull beans

Valid in the context of web-aware Spring ApplicationContext

Request - a single HTTP request

Session - HTTP Session

Global session - global HTTP Session

Spring configuration demo ...

Spring AOP

Aspect Oriented Programming

[bytesource]

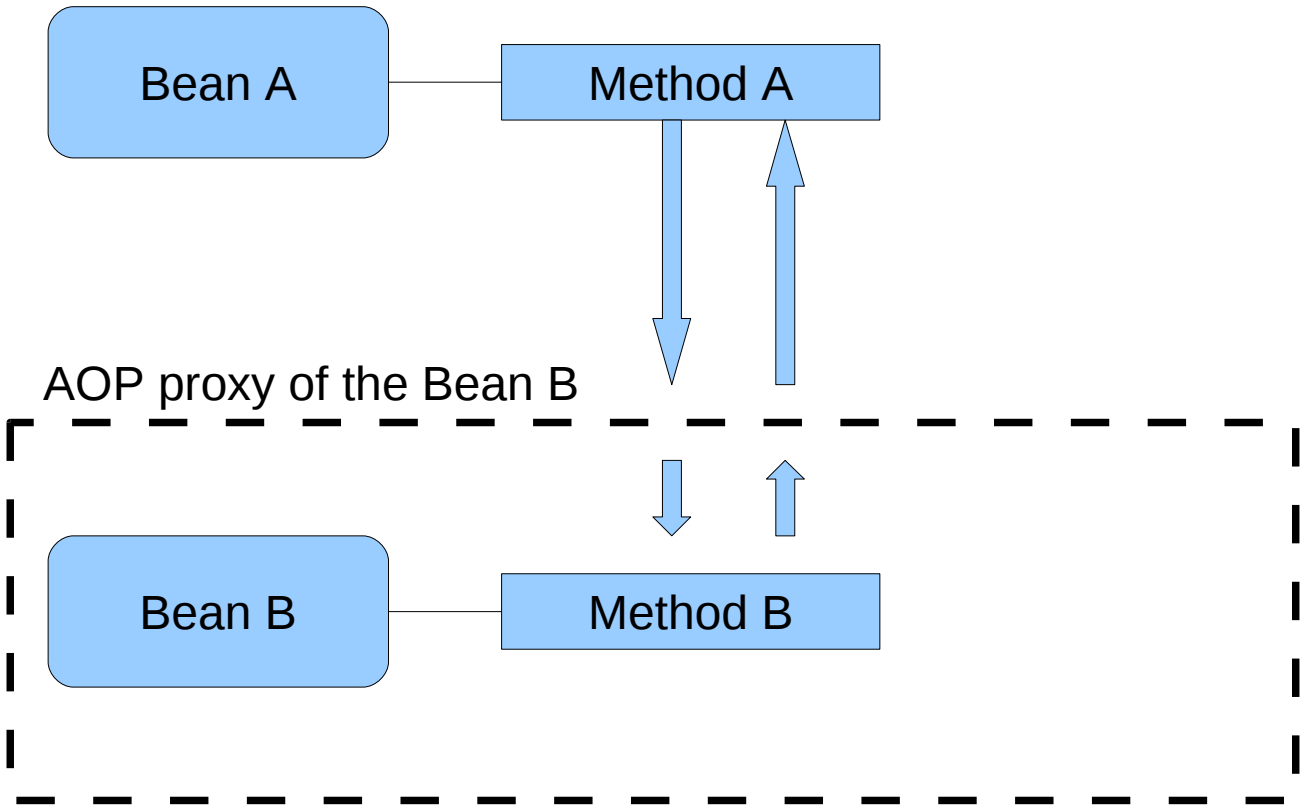
- AOP complements OOP by providing another way of thinking about program structure

- Decomposes a system into concerns

  (TX, logging, audit, caching, security, ...)

- Solves an issue of **cross-cutting concerns** which are difficult to modularize

Spring AOP



Advice runs

@**Before**,
@**AfterReturning**,
@**AfterThrowing**,
@**After** (finally)
or
@**Around**

method executions matched by the **Pointcut**

AOP Terminology

Aspect – a modularization of a concern. In Spring are implemented as regular classes

Joint point – a point during the execution of a program. In Apring AOP always represents a method execution

Pointcut – a predicate that matches the point

Advice – action taken by an aspect at a particular join point

Spring TX support via AOP configration demo ...

Spring AOP demo ...

Testing support in Spring

Introduction to testing

- IoC principle is a benefit and increases productivity
- **Unit testing**
  - JUnit, TestNG support
  - Easy to instantiate object simply using the new operator or use Mock objects
  - Spring provides set of utilities, eg ReflectionTestUtils
- **Integration testing**
  - Spring IoC container caching between test execution
  - Dependency injection
  - Transaction Management
  - Support abstract classes providing eg. application context

Test specific annotations

[bytesource]

```
@ContextConfiguration(locations={"example/test-context.xml"})
@DirtiesContext(classMode = ClassMode.AFTER_EACH_TEST_METHOD)
@TestExecutionListeners({ TransactionalTestExecutionListener.class})
@TransactionConfiguration(transactionManager="txMgr", defaultRollback=false)
@Rollback(false)
@BeforeTransaction
@AfterTransaction
@Timed(millis=1000)
@Repeat(10)
```

Non-test-specific annotations

[bytesource]

@Autowired

@Qualifier

@Resource (javax.annotation) if JSR-250 is present

@Inject (javax.inject) if JSR-330 is present

@Named (javax.inject) if JSR-330 is present

@Provider (javax.inject) if JSR-330 is present

@PersistenceContext (javax.persistence) if JPA is present

@PersistenceUnit (javax.persistence) if JPA is present

@Required

@Transactional

Test demo ...