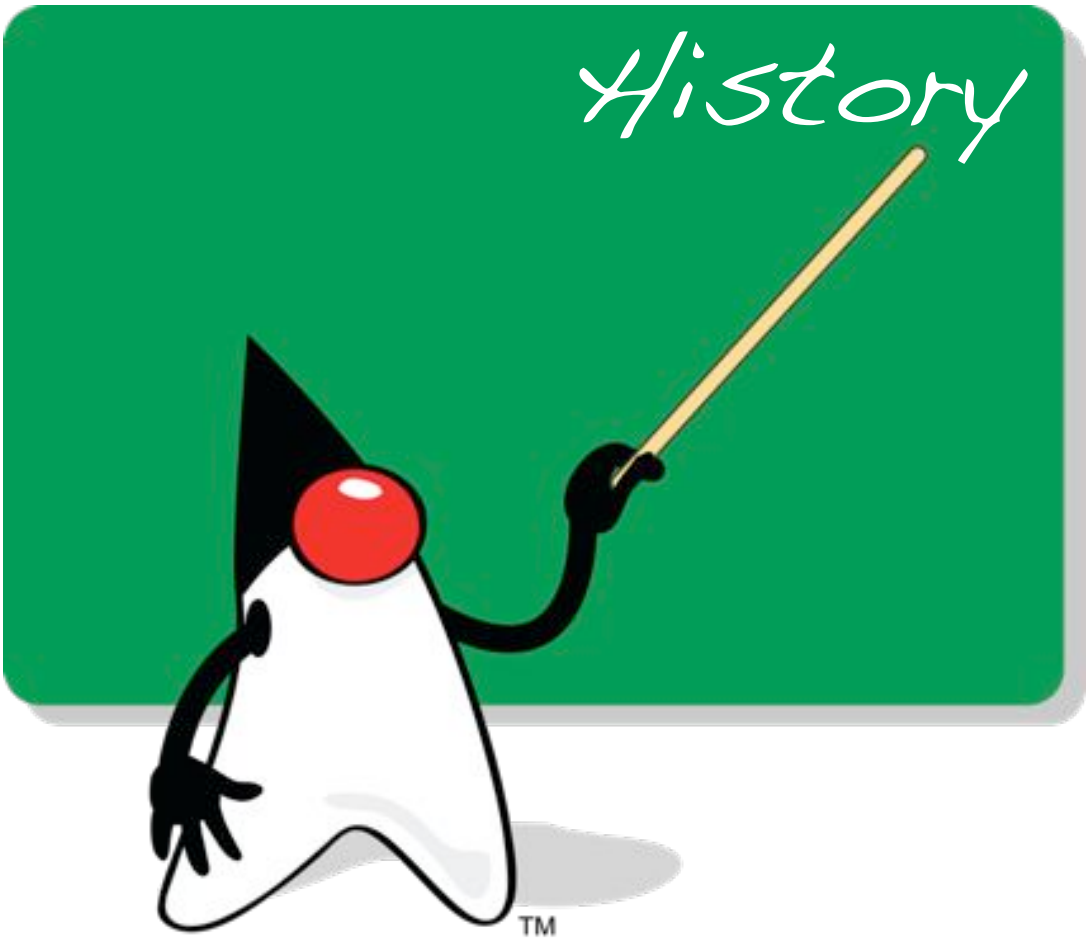
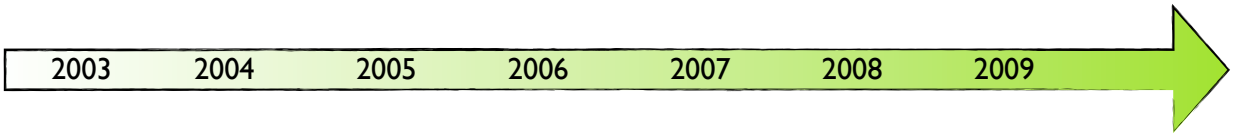
Spring

Paul Bakker

Course contents

- Spring XML configuration
- Spring Annotations
- AOP
- Data Access & Transaction management
- JDBC
- JPA Integration
- Web MVC
- RESTful web services
- Spring Security
- JMS
- Remoting
- SOAP web services
- Tasks


History
TM


2003
2004
2005
2006
2007
2008
2009

Founded

Rebranded

2003 2004 2005 2006 2007 2008 2009

Open Source Project created

Funding raised

Founded

Rebranded

2003 2004 2005 2006 2007 2008 2009

Open Source Project created

Funding raised

- Apache technology
  - HTTP Server
  - Tomcat

Founded

Rebranded

2003 2004 2005 2006 2007 2008 2009

Open Source Project created

Funding raised

Groovy & Grails

Founded

Rebranded

2003 2004 2005 2006 2007 2008 2009

Open Source Project created

Funding raised

Tomcat monitoring & administration

Founded

Rebranded

2003 2004 2005 2006 2007 2008 2009

Open Source Project created

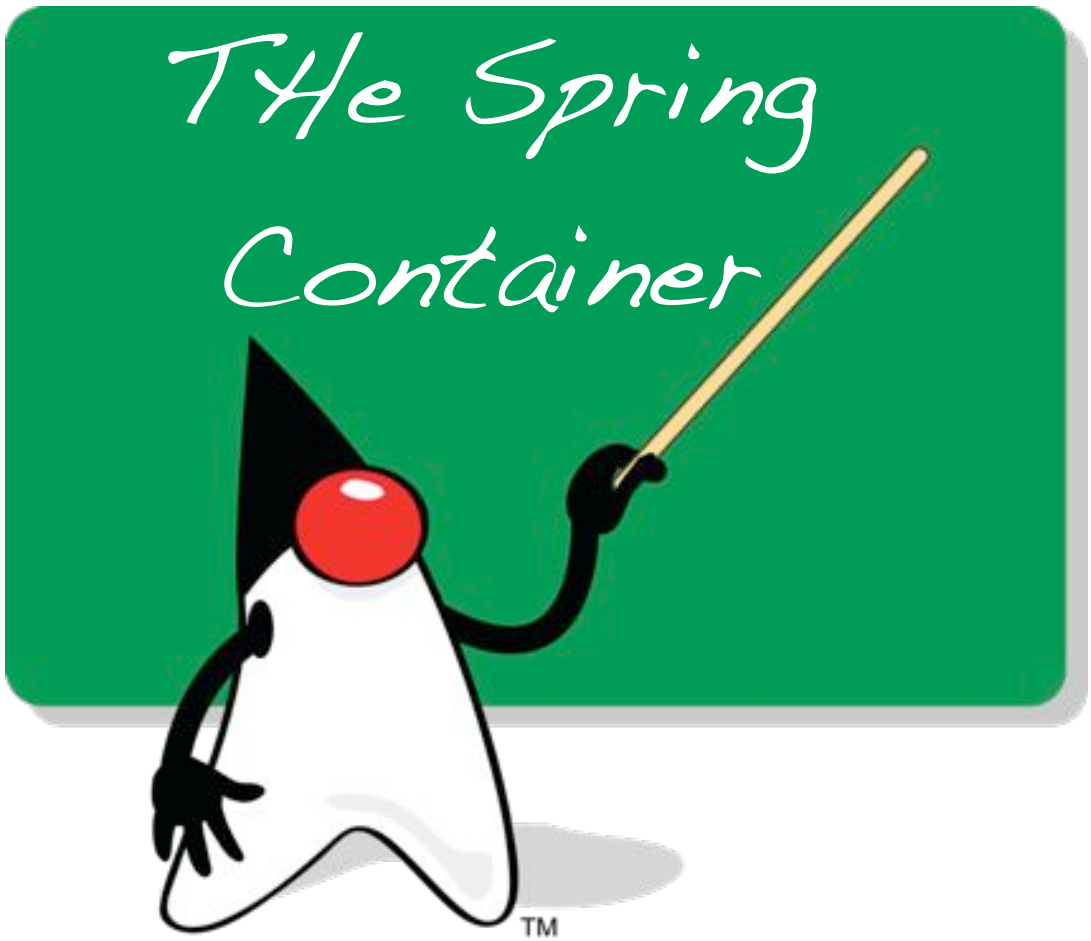
Funding raised

Amazon Elastic Cloud integration

vmware®

vmware®

Virtualization of Enterprise Java applications



The ApplicationContext

- Each Spring application has an ApplicationContext
- Defined in a XML file
- Should be on the classpath of the application

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:util="http://www.springframework.org/schema/util"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd" >
```

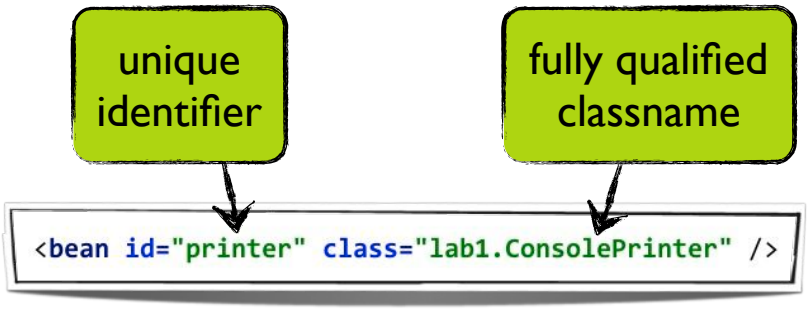

Spring beans

- Any Java class configured in applicationContext.xml
- A Spring Bean is singleton by default

```
<bean id="printer" class="lab1.ConsolePrinter" />
```

Spring beans

- Any Java class configured in applicationContext.xml
- A Spring Bean is singleton by default



Spring bean example

- Plain POJO class
  - optionally implement an interface
- Must have no-arg constructor

```
public class ConsolePrinter implements PrinterService {

    public void print(String message) {
        System.out.println(message);

    }

}
```

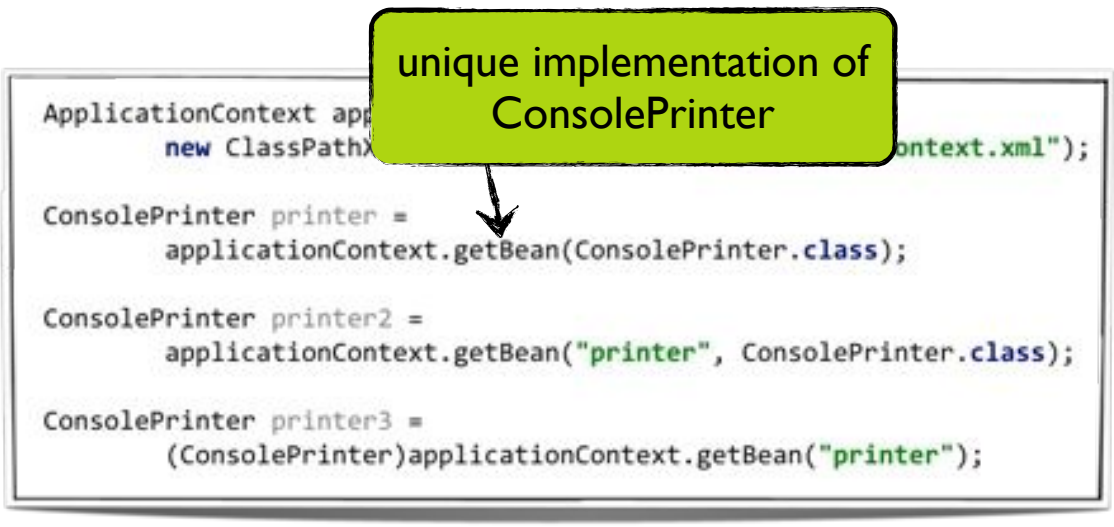
Starting a Spring container

- Just for stand-alone applications

```
ApplicationContext applicationContext =
        new ClassPathXmlApplicationContext("applicationContext.xml");

ConsolePrinter printer =
        applicationContext.getBean(ConsolePrinter.class);

ConsolePrinter printer2 =
        applicationContext.getBean("printer", ConsolePrinter.class);

ConsolePrinter printer3 =
        (ConsolePrinter)applicationContext.getBean("printer");
```


Starting a Spring container

- Just for stand-alone applications



14

Starting a Spring container

- Just for stand-alone applications

```
ApplicationContext appl
        new ClassPathX                    ntext.xml");

ConsolePrinter printer
        applicationCont                   ),

ConsolePrinter printer2 =
        applicationContext.getBean("printer", ConsolePrinter.class);

ConsolePrinter printer3 =
        (ConsolePrinter)applicationContext.getBean("printer");
```

Get by ID with type-parameter

Starting a Spring container

- Just for stand-alone applications

```
ApplicationContext applicationContext =
        new ClassPathXmlApplicationContext("applicationContext.xml");

ConsolePrinter printer =
        applicationContext.getBean(Cons

ConsolePrinter printer2 =
        applicationContext.getBean("pri

ConsolePrinter printer3 =
        (ConsolePrinter)applicationContext.getBean("printer");
```

Get by ID (needs type cast)

Dependency Injection

- Only **define** a dependency, don't instantiate or lookup dependencies
- Spring wires dependencies
  - Setter injection
  - Constructor injection
- Dependencies are injected at bean creation time

Constructor Injection

```
public class ConstructorDI {
    private PrinterService printer;

    public ConstructorDI(PrinterService printer) {

        this.printer = printer;
    }
```

```
<bean id="printer" class="lab1.ConsolePrinter" />

<bean id="constructorDI" class="lab1.ConstructorDI">
    <constructor-arg ref="printer"/>
</bean>
```

Constructor Injection

- Arguments are resolved by type
  - argument order cannot be determined

```
public ConstructorDI(PrinterService printer,
                     OtherDependency otherDependency) {
    this.printer = printer;
    this.otherDependency = otherDependency;
}
```

```
<bean id="constructorDI" class="lab1.ConstructorDI">
    <constructor-arg ref="printer"/>
    <constructor-arg ref="otherDependency"/>
</bean>
```


Constructor type ambiguity

```
public ConstructorDI(String myString, int myInt) {
    this.myString = myString;
    this.myInt = myInt;
}
```

```
<bean id="constructorDI" class="lab1.ConstructorDI">
    <constructor-arg type="java.lang.String" value="Hello"/>
    <constructor-arg type="int" value="1"/>
</bean>
```

Remove ambiguity for simple types using explicit types

Constructor type ambiguity

```
public ConstructorDI(String myString, int myInt) {
    this.myString = myString;
    this.myInt = myInt;
}
```

```
<bean id="constructorDI" class="lab1.ConstructorDI">
    <constructor-arg index="0" value="Hello"/>
    <constructor-arg index="1" value="1"/>
</bean>
```

Remove ambiguity for simple types using argument index

Setter Injection

```
public class SetterDI {
    private PrinterService printer;

    public void setPrinter(PrinterService printer) {
        this.printer = printer;
    }
```

```
<bean id="printer" class="lab1.ConsolePrinter"/>

<bean id="setterDI" class="lab1.SetterDI">
    <property name="printer" ref="printer"/>
</bean>
```

AutoWiring

- Wire dependencies automatically
  - don't specify dependencies using `<property/>` or `<constructor-arg/>`
- Less XML configuration
- Add / remove dependencies more easily
- Turn on globally or per bean

AutoWiring Types

Mode	Explanation
no	No autowiring (default)
byName	Property name must match bean id
byType	Property type must match **exactly one** bean. Must have default constructor.
constructor	Injection by type using a constructor. All arguments must be resolvable.
autodetect (deprecated)	Chooses between *byType* and *constructor*.

AutoWiring byType

```
<bean id="autowiringDI" class="lab1.AutoWireDI" autowire="byType"/>
```

```
public class AutoWireDI {
    private PrinterService printer;

    public void setPrinter(PrinterService printer) {
        this.printer = printer;
    }
```


Checking dependencies

Mode	Explanation
none	No dependency checking (default)
simple	Only primitive types and collections
object	Only collaborators
all	Primitive types and collaborators

Deprecated in Spring 3

AutoWiring constructor

```
<bean id="autowiringDI" class="lab1.AutoWireDI" autowire="constructor"/>
```

```
public class AutoWireDI {
    private PrinterService printer;

    public AutoWireDI(PrinterService printer) {
        this.printer = printer;
    }
```


AutoWire byName

```
<bean id="printer" class="lab1.ConsolePrinter"/>

<bean id="autowiringDI" class="lab1.AutoWireDI" autowire="byName"/>
```

```
public class AutoWireDI {
    private PrinterService printer;

    public void setPrinter(PrinterService printer) {
        this.printer = printer;
    }
```

AutoWire byName

```
<bean id="printer" class="lab1.ConsolePrinter"/>

<bean id="autowiringDI" class="lab1.AutoWireDI" autowire="byName"/>
```

must match

```
public class AutoWireDI {
    private PrinterService printer;

    public void setPrinter(PrinterService printer) {
        this.printer = printer;
    }
```


Global AutoWiring

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd"
       default-autowire="byType">
```

Autowire-candidate

- When multiple implementations of an interface exists Spring can't autowire
  - choose the implementation to inject

Don't use for injection

```
<bean id="otherDependency" class="lab1.OtherDependency"
      autowire-candidate="false"/>
```

Always use for injection

```
<bean id="otherDependency" class="lab1.OtherDependency"
      primary="true"/>
```


Factory method

- Construct beans using a static factory method

```
public class FactoryMethodInstantiation {
    private FactoryMethodInstantiation() {
    }

    public static FactoryMethodInstantiation createInstance() {
        return new FactoryMethodInstantiation();
    }
```

```
<bean id="factoryMethodInstantiation"
      class="lab1.FactoryMethodInstantiation"
      factory-method="createInstance">
</bean>
```

Factory method arguments

```
<bean id="factoryMethodInstantiation"
      class="lab1.FactoryMethodInstantiation"
      factory-method="createInstance">
    <constructor-arg ref="printer"/>
    <constructor-arg value="MyTest"/>
</bean>
```

```
public class FactoryMethodInstantiation {
    private PrinterService printer;
    private String value;

    private FactoryMethodInstantiation() {
    }

    public static FactoryMethodInstantiation createInstance(
            PrinterService printer, String value) {
        FactoryMethodInstantiation instance =
                new FactoryMethodInstantiation();
        instance.printer = printer;
        instance.value = value;
        return instance;
    }
```


Factory Bean

- Construct and initialize beans that you don't control

```
<bean id="nameFactory" class="lab1.NameFactory"/>

<bean id="names" factory-bean="nameFactory"
      factory-method="createNameList"/>
```

```
public class NameFactory {
    public List<String> createNameList() {
        return Arrays.asList("Paul", "Bert", "Joris");
    }
}
```

Method injection

- Normally injection is done at bean creation time
- Method injection allows injection at each call to a bean
- Spring injects an abstract method implementation

Method injection

```
public abstract class NormalSpringBean {
    public int doSomething() {
        return getDynamicDependency().doSomething();
    }

    public abstract MyDynamicDependency getDynamicDependency();
}
```

```
<bean id="methodInjectionExample" class="lab1.NormalSpringBean">
    <lookup-method name="getDynamicDependency" bean="dynamicBean"/>
</bean>
```

Generates a method that re-inject dynamicBean

Scopes

Scope	Explanation
singleton (default)	A single instance of a the bean. Dependencies to the bean are shared.
prototype	New instance for each injection point. Each injection point creates instance once however.
prototype + method injection	New instance for each call to the injected method.
request (web only)	Instance per HTTP request
session (web only)	Instance per HTTP session

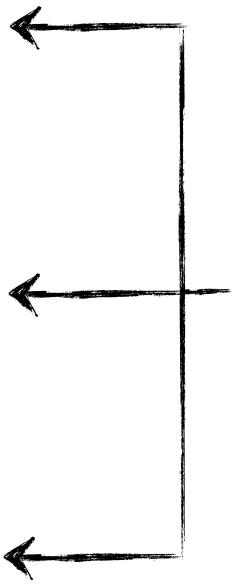
```
<bean id="hitCounter" class="lab1.HitCounter"
      scope="prototype"/>
```


Singleton scope

```
<bean id="hitter1"
      class="lab1.Hitter">
    <property
            name="counter"
            ref="hitCounter"/>
</bean>
```

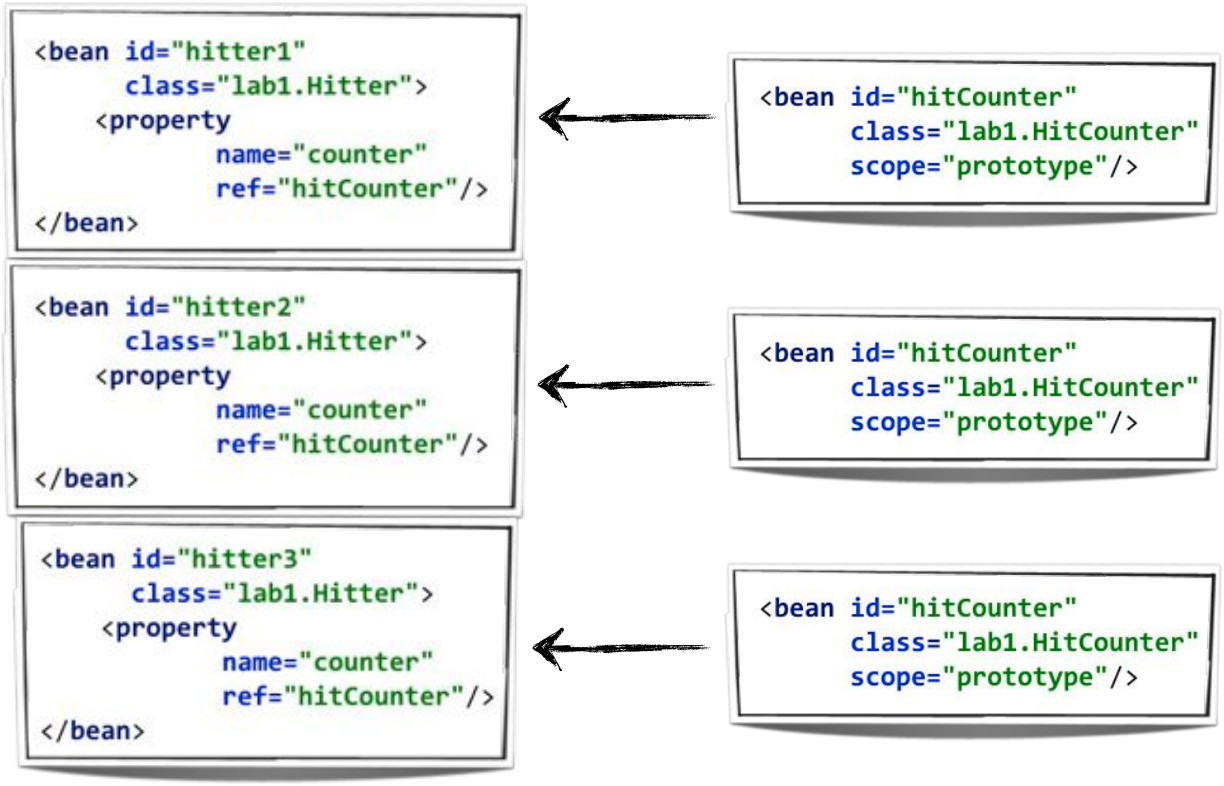
```
<bean id="hitter2"
      class="lab1.Hitter">
    <property
            name="counter"
            ref="hitCounter"/>
</bean>
```

```
<bean id="hitter3"
      class="lab1.Hitter">
    <property
            name="counter"
            ref="hitCounter"/>
</bean>
```



```
<bean id="hitCounter"
      class="lab1.HitCounter"
      scope="singleton"/>
```

Prototype scope



Life-cycle callbacks

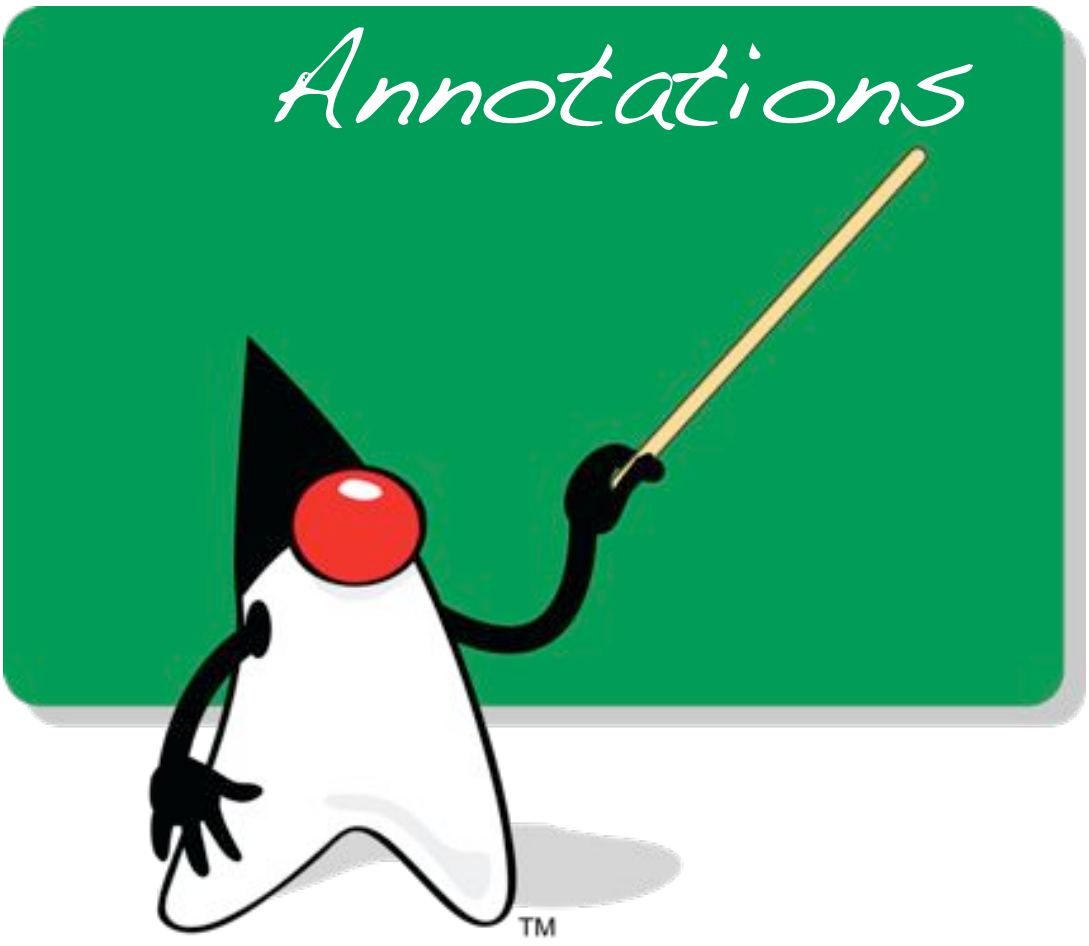
```
public class LifecycleCallbacks {
    public void init() {
        System.out.println("Creating bean");
    }

    public void destroy() {
        System.out.println("Destroying bean");
    }
}
```

```
<bean id="lifecycleCallbacks" class="lab1.LifecycleCallbacks"
      init-method="init" destroy-method="destroy"/>
```

Bean definition inheritance

No instance will be created, just acts as a template

```
<bean id="parentBean" abstract="true" class="lab1.ParentBean">
    <property name="name" value="parent"/>
    <property name="age" value="1"/>
</bean>

<bean id="inheritsWithDifferentName" parent="parentBean">
    <property name="name" value="override"/>
</bean>
```

Configuring annotations

```
<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-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <context:annotation-config />

    <context:component-scan base-package="lab2"/>
</beans>
```

Turns on annotation based dependency injection

Turns on scanning for @Component etc.

Defining Spring beans

```
@Component
public class SpringComponent {
```

<->

```
<bean id="springComponent"
      class="lab2.SpringComponent"/>
```

```
@Repository
public class SpringDao {
```

```
@Service
public class SpringService {
```

```
@Controller
public class SpringController {
```

Extensions of @Component

@Autowired

Field injection

```
@Autowired
private BookCatalog bookCatalog;
```

Setter injection

```
@Autowired
public void setBookCatalog(BookCatalog bookCatalog) {
    this.bookCatalog = bookCatalog;
}
```

Constructor injection

```
@Autowired
public BookController(BookCatalog bookCatalog) {
    this.bookCatalog = bookCatalog;
}
```


@Inject

- Same semantics as @Autowired
- Standardized in JSR-330
- Needs JSR-330 on the classpath

```
@Inject
private BookCatalog bookCatalog;
```

```
<dependency>
    <groupId>javax.inject</groupId>
    <artifactId>javax.inject</artifactId>
    <version>1</version>
</dependency>
```

@Required

- Works together with XML-based autowired setter injection
- Makes sure a dependency is injected

```
@Required
public void setBookCatalog(BookCatalog bookCatalog) {
    this.bookCatalog = bookCatalog;
}
```


Qualifiers

- Qualifiers bind an injection point to a specific implementation of an interface
  - runtime binding, not compile time
  - unit testing still possible
- Necessary when multiple implementations exist

String based qualifiers

```
@Component
@Qualifier("general")
public class GeneralMovieCatalog implements MovieCatalog {
```

```
@Autowired @Qualifier("general")
private MovieCatalog generalMovies;
```


Qualifier annotations

```
@Component
@Action
public class ActionMovieCatalog implements MovieCatalog{
```

```
@Autowired @Action
private MovieCatalog movieCatalog;
```

```
@Target({ElementType.TYPE,
         ElementType.FIELD,
         ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
@Qualifier
public @interface Action {
}
```

custom qualifier annotation

Injecting multiple implementations

All MovieCatalog implementations

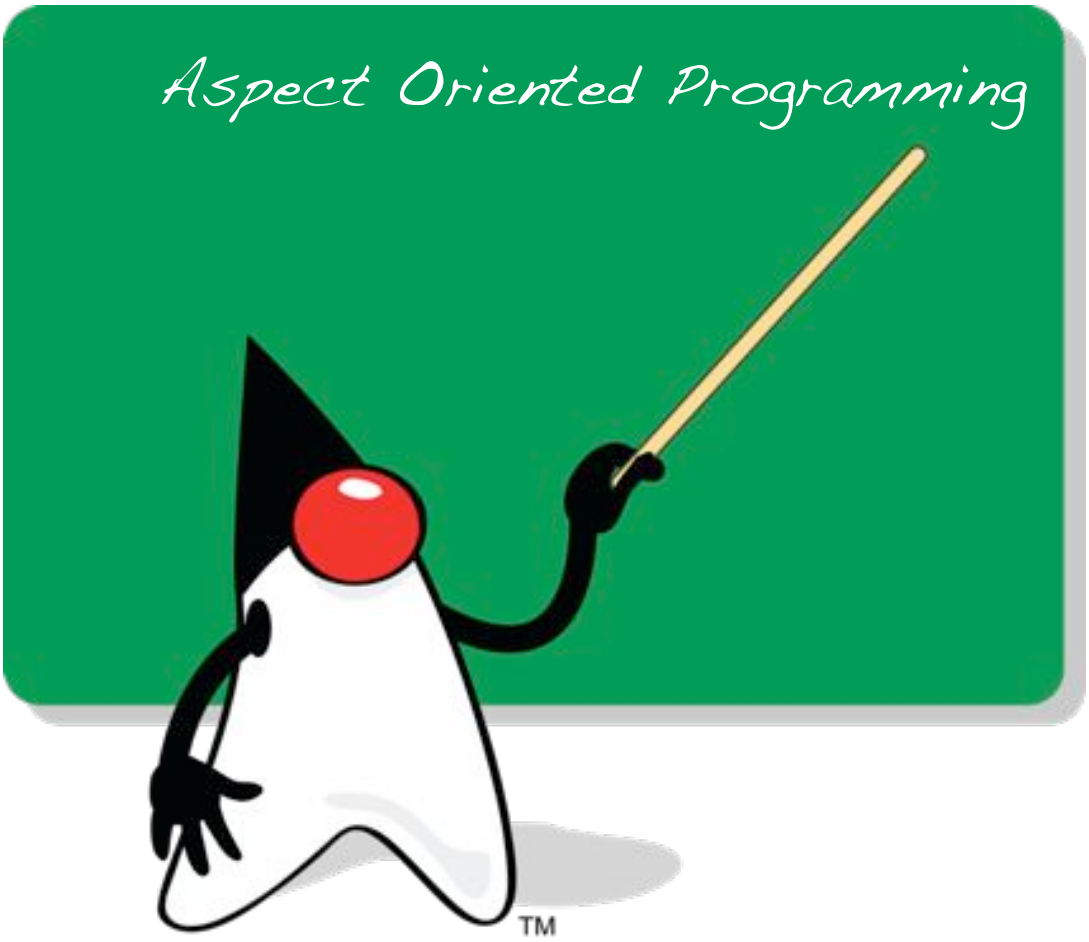
```
@Autowired
List<MovieCatalog> allCatalogs;
```

All MovieCatalog implementations with bean id

```
@Autowired
Map<String, MovieCatalog> allCatalogs;
```


Life-cycle callbacks

```
@PostConstruct
public void init() {

}

@PreDestroy
public void destroy() {

}
```



AOP

- Some *concerns* in an application are *crosscutting*
  - the same code would be repeated at many places
  - e.g. security, logging, retry-on-error etc.
- AOP complements the Object Oriented paradigm

AOP in Java

- AspectJ is the most used AOP implementation
- Needs an additional *weaving* compiler after the Java compiler
  - the build process becomes more complicated
- Spring AOP is easier but less powerful

AOP definitions

- Join point
  - a point during execution of code (a method execution)
- Advice
  - Action taken at a join point
  - e.g. "around", "before" or "after"
  - this is where you implement your code
- Pointcut
  - predicate that matches join points
  - e.g. "all methods in a certain package"

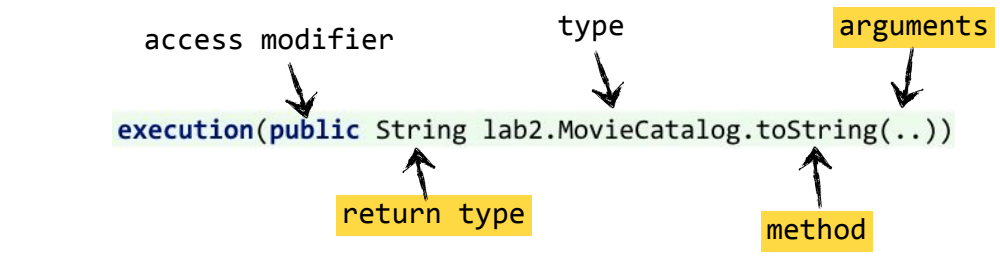
Declaring Aspects

Combined pointcut and advice definition

```
@Aspect
@Component
public class TraceLogAspect {

    @Before("execution(public * *(..))")
    public void before() {
        System.out.println("Called before each public method!");
    }
```

```
<context:component-scan base-package="lab2"/>
<aop:aspectj-autoproxy/>
```


Pointcut definitions



Pointcut examples

Pointcut	Explanation
execution(public * *(..))	All public methods
execution(String *(..))	All methods returning a String
execution(public * save(..))	All public save methods
execution(public * a.b.dao.* (..))	All public methods in the dao package
execution(public String *(String, Integer))	All public methods returning a String and arguments of type String and Integer
@annotation(a.b.MyAnnotation)	All methods annotated a.b.MyAnnotation

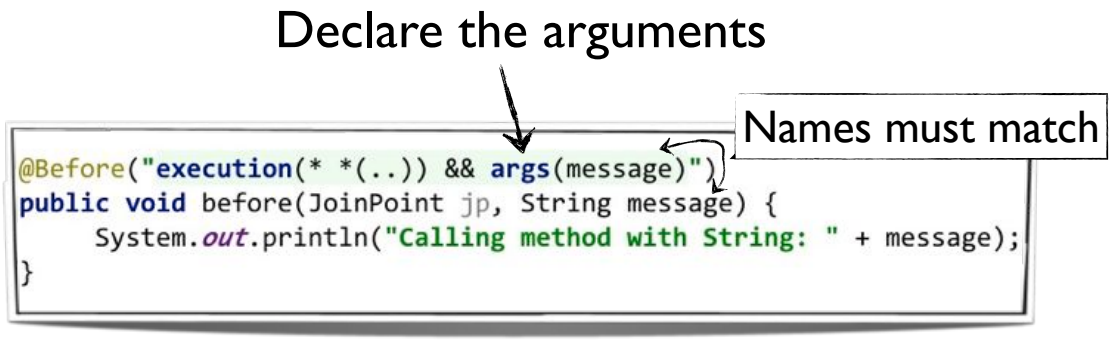
Before advice

```
@Before("execution(* *(..)) ")
public void before() {
}
```

Declare a `org.aspectj.lang.ProceedingJoinPoint` as first parameter

```
@Before("execution(* *(..))")
public void before(JoinPoint jp) {
     System.out.println("Before " + jp.getStaticPart().toString());
}
```

Passing arguments



After advices

```
@AfterReturning(value = "execution(* * (..))", returning = "retVal")
public void afterSuccess(Object retVal) {
    System.out.println("I returned successfully: "
            + retVal);
}

@AfterThrowing(value = "execution(* * (..))", throwing = "ex")
public void afterException(Exception ex) {
    System.out.println("I executed with an exception: "
            + ex.getMessage());
}

@After("execution(* *(..))")
public void after() {
    System.out.println("I'm done!");
}
```

Around advice

Must call *proceed* to continue method call

```
@Around("execution(public * *(..))")
public Object trace(ProceedingJoinPoint jp) throws Throwable {
    long curTime = System.currentTimeMillis();

    Object result = jp.proceed();

    long time = System.currentTimeMillis() - curTime;
    System.out.println(jp.getStaticPart().
                              getSignature().
                              getName() + " " + time);

    return result;
}
```


Retry example

```
@Around("execution(public * *(..))")
public Object retry(ProceedingJoinPoint jp) throws Throwable {
    Object result = null;

    try {
        result = jp.proceed();
    } catch (Exception ex) {
        System.out.println("Sleep 2 seconds before retry");
        TimeUnit.SECONDS.sleep(2);
        result = jp.proceed();
    }

    return result;
}
```

Re-usable pointcut definitions

```java
@Aspect
@Component
public class MyPointcuts {
    @Pointcut("execution(* a.b.dao(..))")
    public void inDaoLayer() {}
}
```

Pointcut name is method name

```java
@After("lab2.aop.MyPointcuts.inDaoLayer()")
public void after() {
    System.out.println("I'm done!");
}
```


AOP using XML

```xml
<aop:config>
    <aop:pointcut expression="execution(* *(..))" id="allMethods"/>
    <aop:aspect ref="traceLogAspect">
        <aop:before method="trace" pointcut-ref="allMethods"/>
    </aop:aspect>
</aop:config>
```

Introductions

- Introduce new methods and interfaces to a class
- How to make a car fly and shoot?
  - mix-in behavior

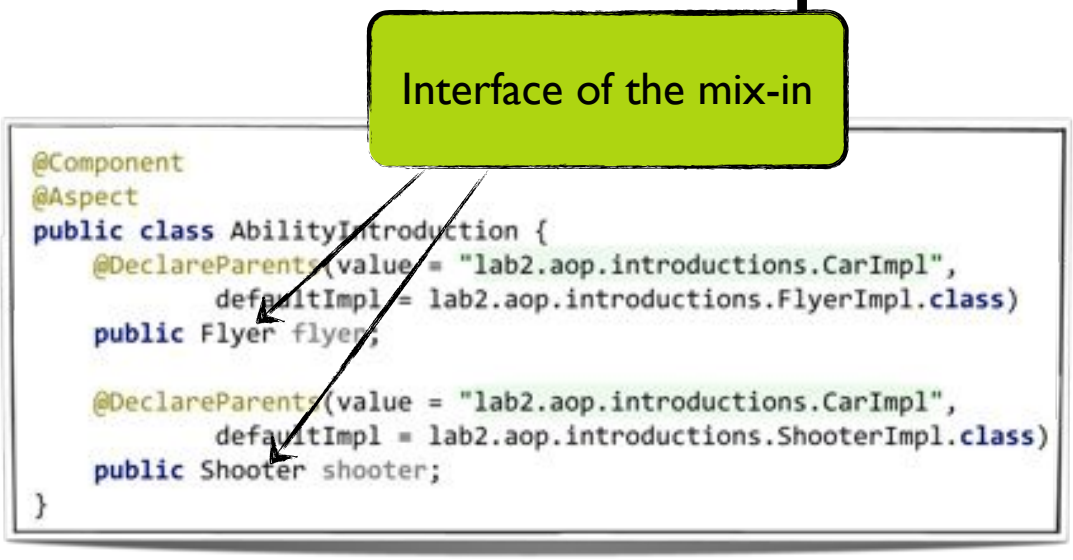
The plain car

```
@Component
public class CarImpl implements Car {
    @Override
    public void drive() {
        System.out.println("Driving...");
    }
}
```

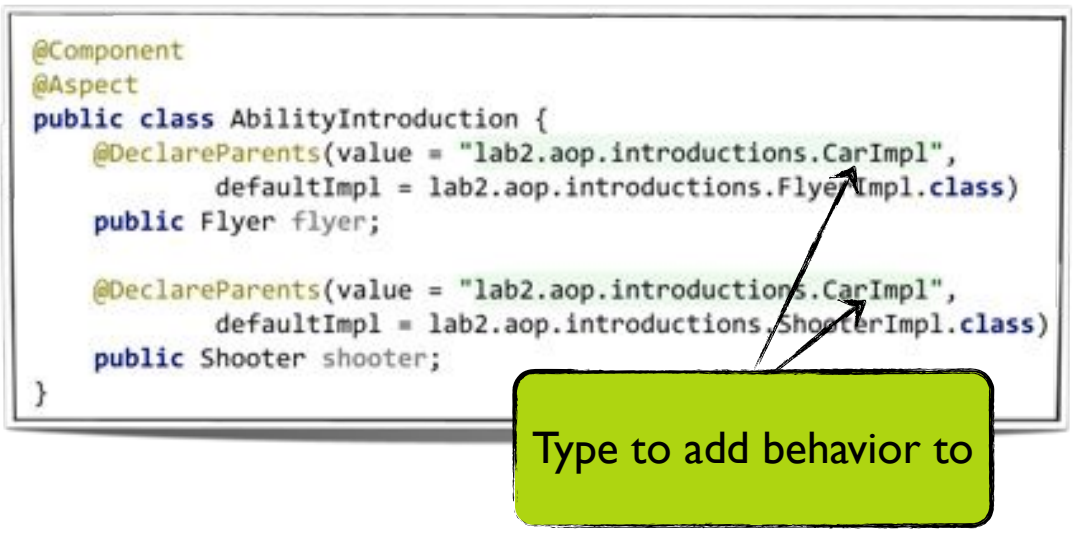

Introduction example

```
@Component
@Aspect
public class AbilityIntroduction {
    @DeclareParents(value = "lab2.aop.introductions.CarImpl",
            defaultImpl = lab2.aop.introductions.FlyerImpl.class)
    public Flyer flyer;

    @DeclareParents(value = "lab2.aop.introductions.CarImpl",
            defaultImpl = lab2.aop.introductions.ShooterImpl.class)
    public Shooter shooter;
}
```

Introduction example



Introduction example



Introduction example

Mix-in implementation

```
@Component
@Aspect
public class AbilityIntroduction {
    @DeclareParents(value = "lab2.aop.introductions.CarImpl",
            defaultImpl = lab2.aop.introductions.FlyerImpl.class)
    public Flyer flyer;

    @DeclareParents(value = "lab2.aop.introductions.CarImpl",
            defaultImpl = lab2.aop.introductions.ShooterImpl.class)
    public Shooter shooter;
}
```

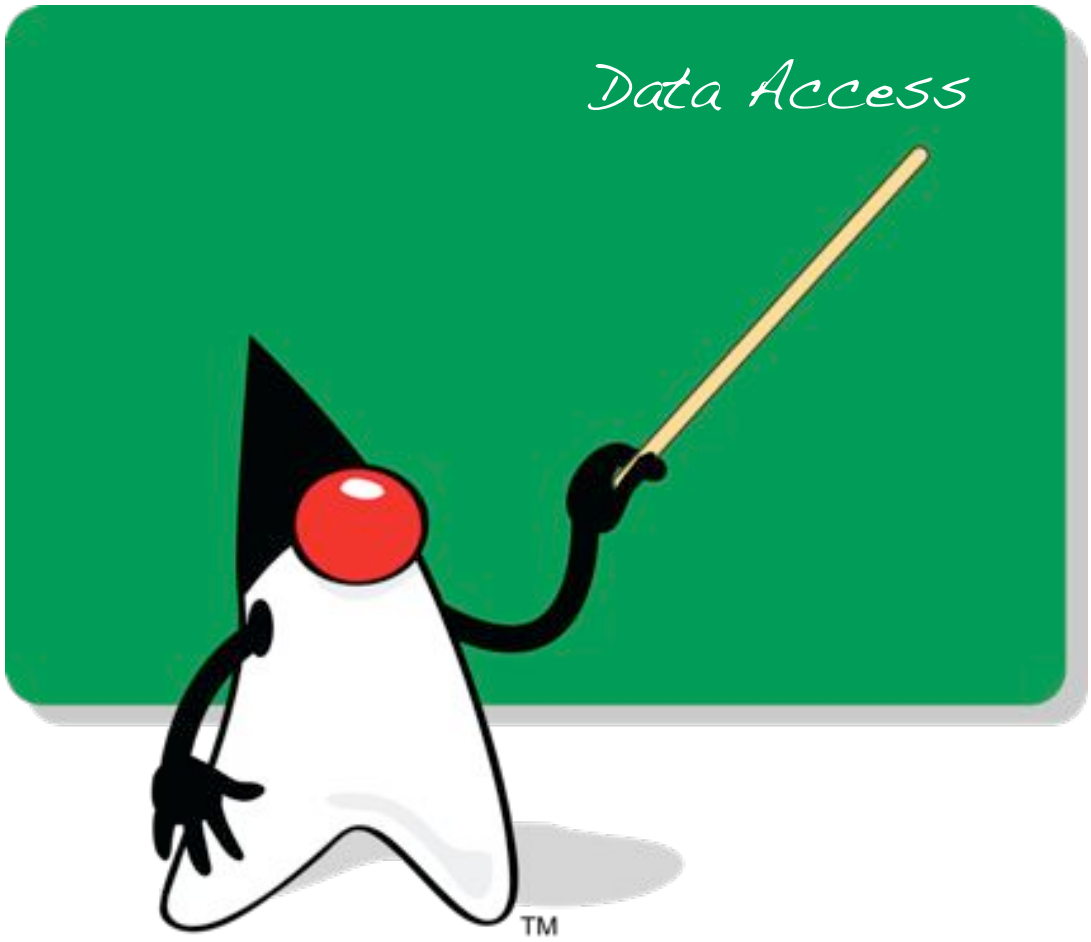

Introduction example

The car now implements Flyer and Shooter

```
Car car = ctx.getBean(Car.class);
car.drive();

Flyer flyer = (Flyer)car;
flyer.fly();

Shooter shooter = (Shooter)car;
shooter.shoot();
```

```
public class FlyerImpl implements Flyer {
    @Override
    public void fly() {
        System.out.println("Flying!");
    }
}
```

```
public class ShooterImpl implements Shooter {
    @Override
    public void shoot() {
        System.out.println("Shooting!");
    }
}
```



Data Access overview

- Declarative transaction management
- Exception handling
- Simplified JDBC support
- ORM integration

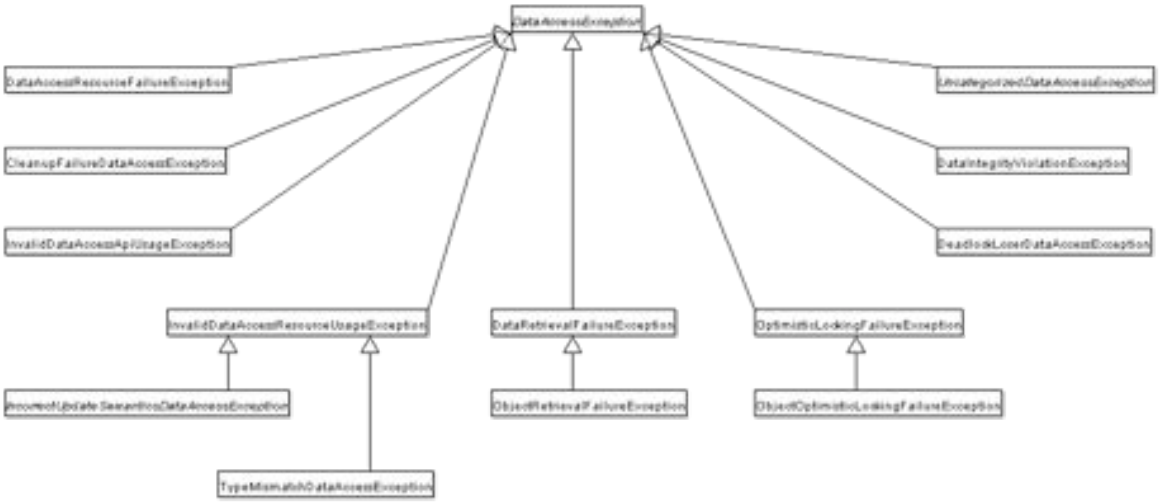
Things wrong with JDBC

- A lot of mandatory checked exception handling
  - most are not recoverable
- SQLException is very generic
  - have to parse the sql error-code yourself
- Clumsy API
  - simple things require a lot of code

Spring JDBC support

- Exception wrapping
  - all exceptions are translated to unchecked exceptions
- Consistent exceptions
  - native sql error codes are translated to consistent exception types
- Simplified APIs

Consistent Exception Hierarchy



Configuring a connection

Create a data source

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
      destroy-method="close">
    <property name="driverClassName" value="${jdbc.driverClassName}"/>
    <property name="url" value="${jdbc.url}"/>
    <property name="username" value="${jdbc.username}"/>
    <property name="password" value=""/>
</bean>

<context:property-placeholder location="classpath:jdbc.properties"/>
```

Lookup a data source

```
<jee:jndi-lookup jndi-name="myJndiDS"/>
```

Simple JDBC Template

- Creating a SimpleJdbcTemplate

```
private SimpleJdbcTemplate jdbcTemplate;

@Autowired
public void setDataSource(DataSource ds) {
    jdbcTemplate = new SimpleJdbcTemplate(ds);
}
```


Queries and mapping

Called for each row

```
public List<Movie> listMovies() {
    return jdbcTemplate.query("select * from movies",
            new RowMapper<Movie>() {

        @Override
        public Movie mapRow(ResultSet rs, int rowNum)
                throws SQLException {
            Movie movie = new Movie();
            movie.setTitle(rs.getString("title"));
            movie.setGenre(rs.getString("genre"));
            movie.setReleaseDate(rs.getDate("releaseDate"));

            return movie;
        }
    });
}
```

QueryForObject

```
public Movie findMovie(int id) {
    return jdbcTemplate.queryForObject("select * from movies where id = ?",
            new RowMapper<Movie>() {
                @Override
                public Movie mapRow(ResultSet rs, int i) throws SQLException {
                    Movie movie = new Movie();
                    movie.setTitle(rs.getString("title"));
                    movie.setGenre(rs.getString("genre"));
                    movie.setReleaseDate(rs.getDate("releaseDate"));

                    return movie;
                }
            }, id);
}
```


QueryFor...

- Convenience methods for several types

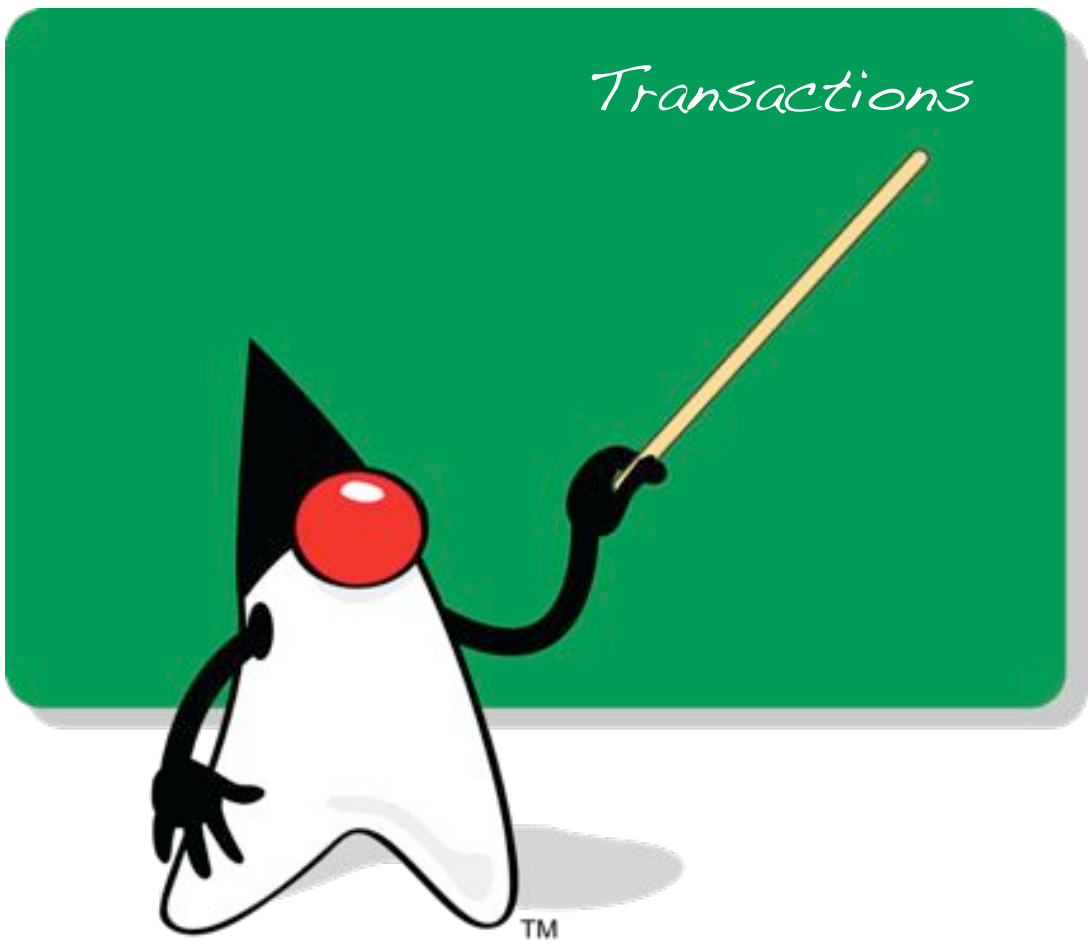
```
int nrOfRows = jdbcTemplate.queryForInt("select count(*) from movies");
```

Simple JDBC Insert

- Creating a SimpleJdbcInsert

```
private SimpleJdbcInsert simpleJdbcInsert;

@Autowired
public void setDataSource(DataSource ds) {
    simpleJdbcInsert = new SimpleJdbcInsert(ds).withTableName("movies")
            .usingGeneratedKeyColumns("id");
}
```

```
Map<String, Object> params = new HashMap<String, Object>();
params.put("title", movie.getTitle());
params.put("genre", movie.getGenre());
params.put("releasedate", movie.getReleaseDate());

return simpleJdbcInsert.executeAndReturnKey(params);
```

Transaction Management

- Consistent programming model across different technologies
  - JDBC, JPA, Hibernate etc.
  - Run JDBC and JPA code in the same transaction
- Declarative tx-management
- Simplified programmatic tx-management API

Creating a transaction manager

- Different transaction managers for different persistence solutions
  - DataSourceTransactionManager
  - HibernateTransactionManager
  - JpaTransactionManager
  - WebLogicJtaTransactionManager
  - WebSphereUowTransactionManager

Creating a transaction manager

JDBC transaction manager

```
<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
```

JPA transaction manager

```
<bean id="transactionManager"
      class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="myEmf"/>
</bean>
```


Declarative Transaction Management

Turn on annotation driven tx-management

```
<tx:annotation-driven transaction-manager="transactionManager"/>
```

Make each method transactional

Read only transaction

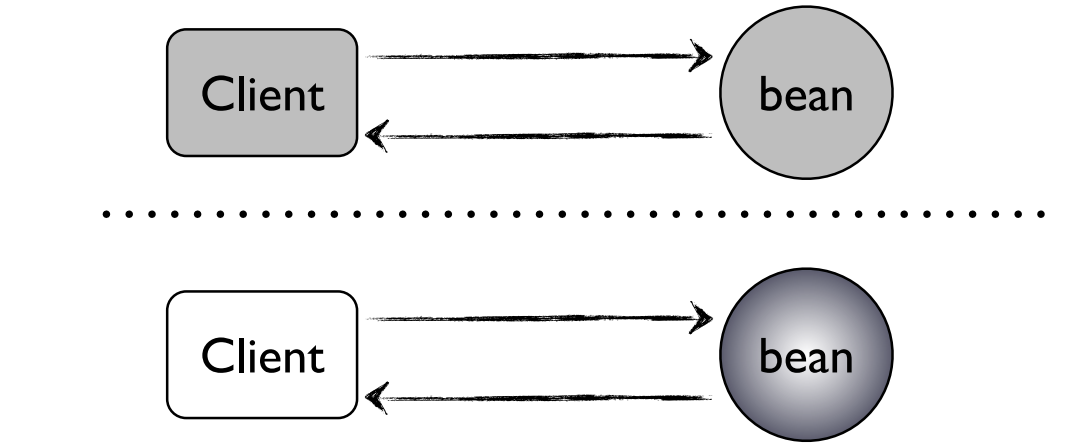
```
@Repository
@Transactional
public class ExampleDAO {
    public void saveContact(String name) {
        //Insert contact
    }

    @Transactional(readOnly = true)
    public List<String> listContacts() {
        //Query contact table
        return null;
    }
}
```

@Transactional

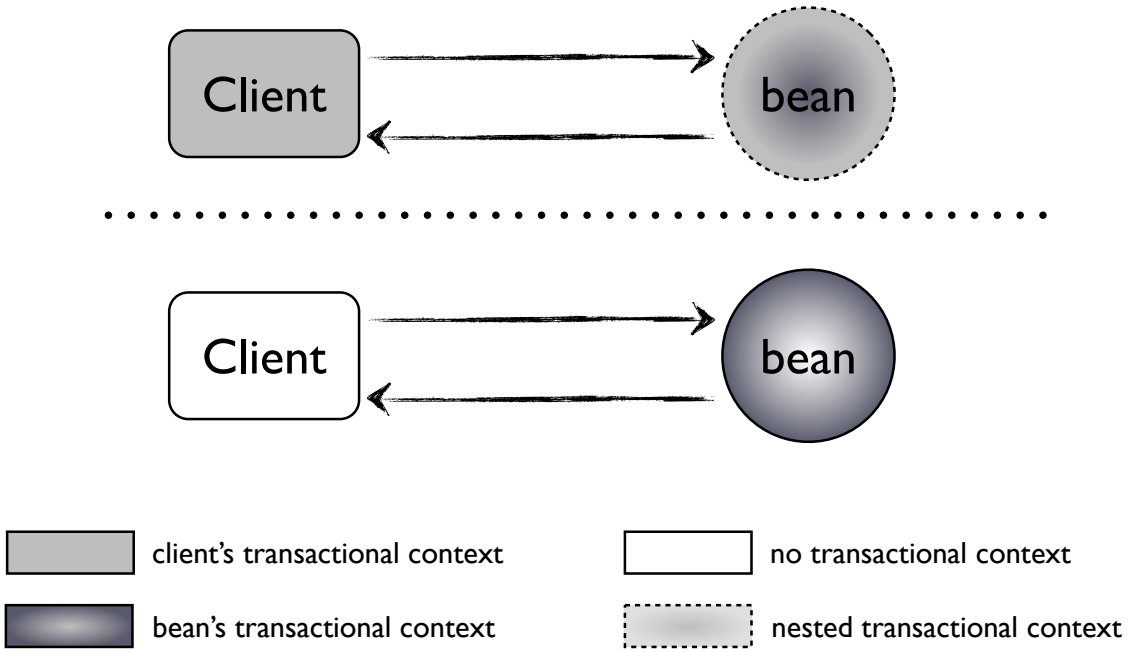
Attribute	Explanation
value	Optional qualifier specifying the tx-manager to use
propagation	Transaction propagation
isolation	Transaction isolation
read-only	Read/write or read-only transaction
rollbackFor	Array of Class objects that extends Throwable and should result in a rollback
noRollbackFor	Array of Class objects that extends Throwable and should not result in a rollback

Propagation

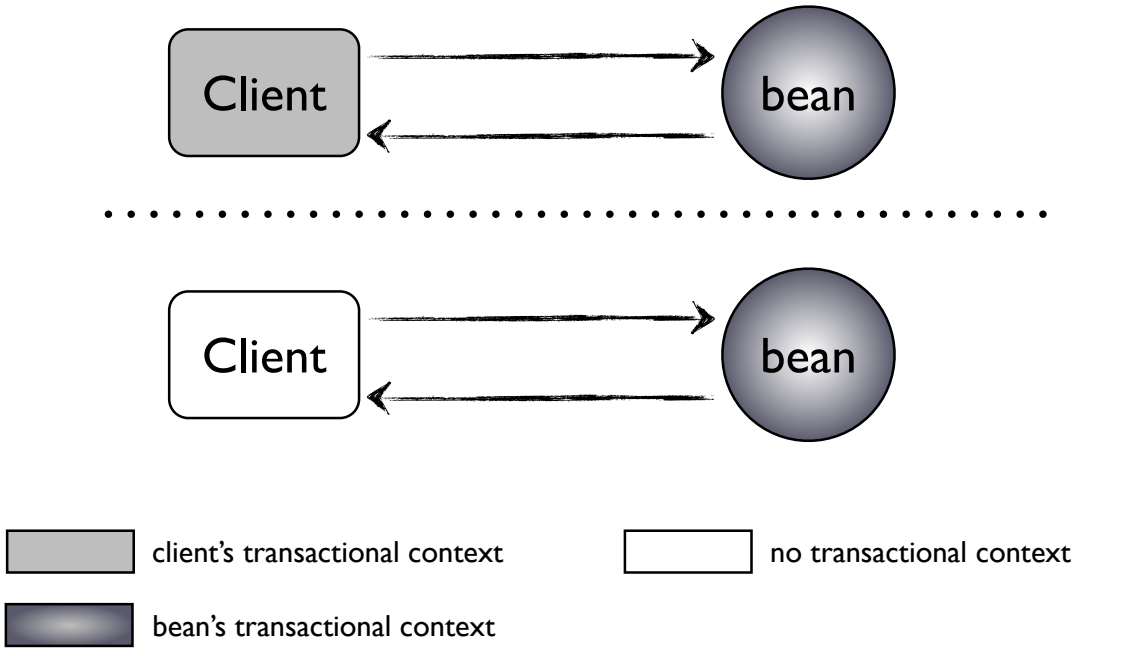
REQUIRED



Propagation
NESTED

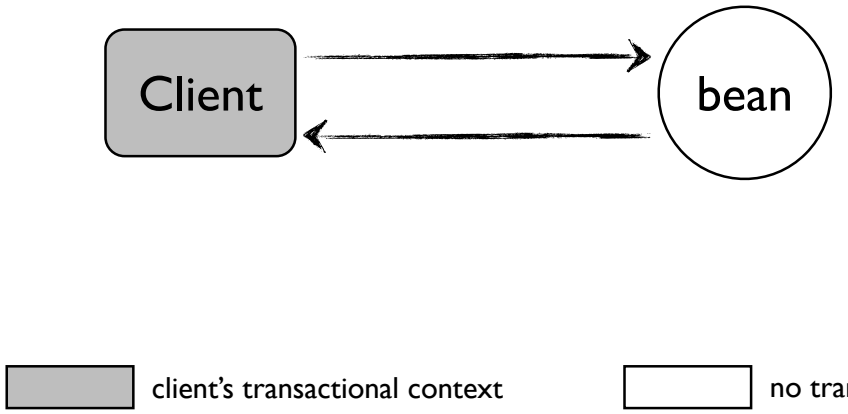


Propagation
REQUIRES_NEW



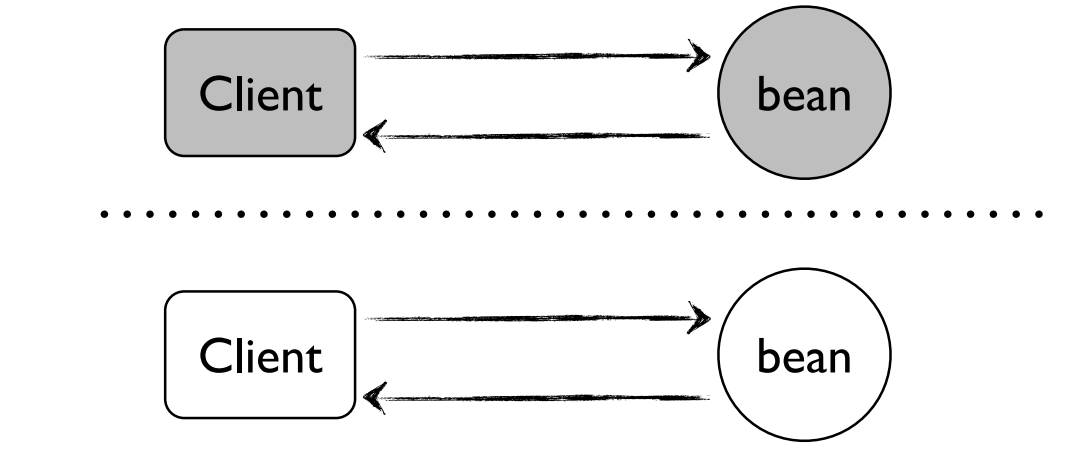
Propagation

NOT_SUPPORTED



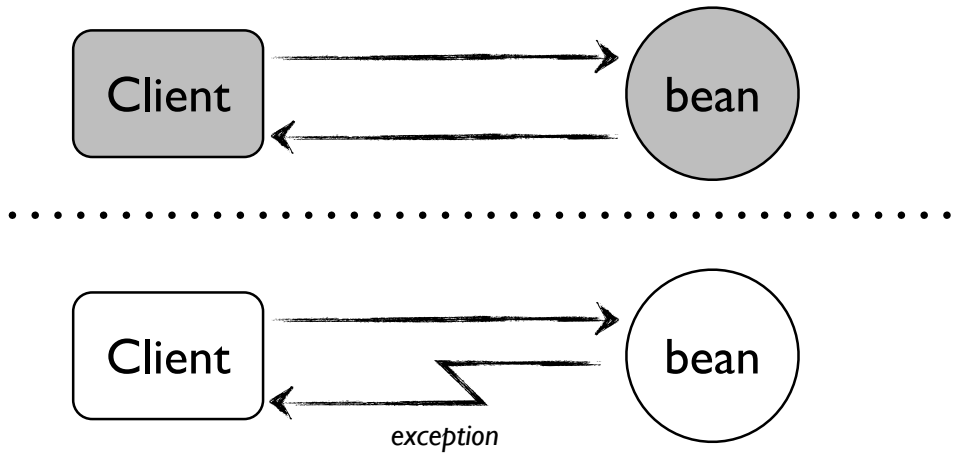
Propagation

SUPPORTS



Propagation

MANDATORY



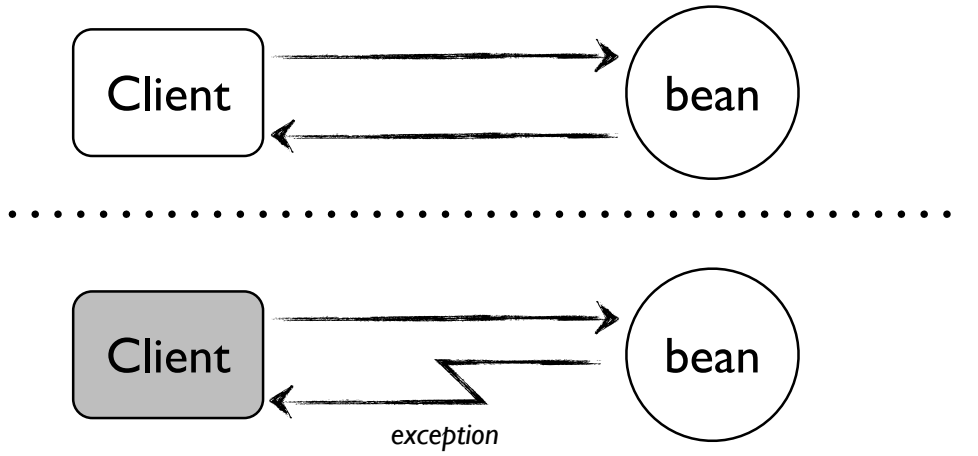
client's transactional context

no transactional context

bean's transactional context

Propagation

NEVER



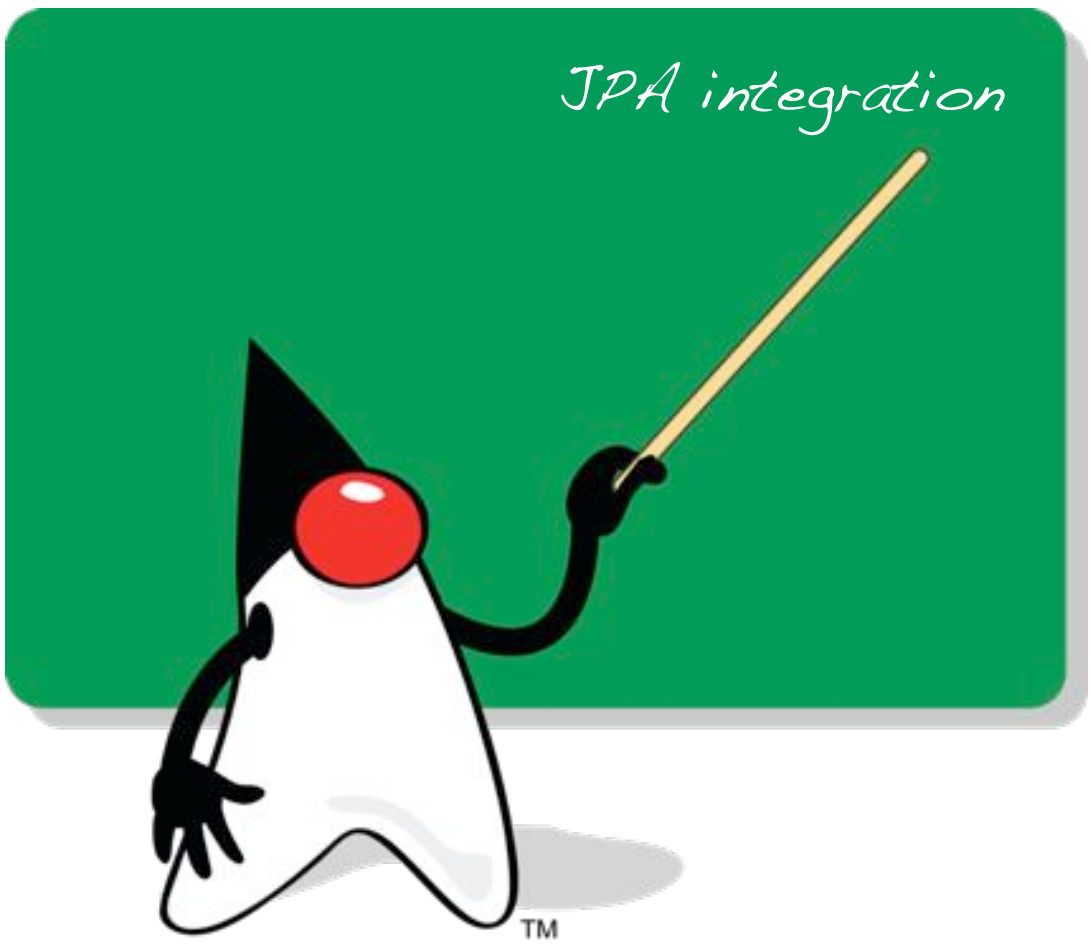
client's transactional context

no transactional context

bean's transactional context

Rollback behavior

- Exceptions bubbled up the stack are handled by Spring
- Transaction is rolled back for unchecked exceptions
  - probably an unrecoverable error
- Transaction is not rolled back for check exceptions
  - probably recoverable situation



Setting up JPA

- Create a META-INF/persistence.xml file
- Configure an EntityManagerFactory
- Configure a JpaTransactionManager
- Configure the JPA dialect in Spring

persistence.xml

```
<persistence-unit name="testPU" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <properties>
        <property name="hibernate.show_sql" value="true" />
        <property name="hibernate.format_sql" value="true" />
        <property name="hibernate.dialect"
                  value="org.hibernate.dialect.MySQL5InnoDBDialect" />
        <property name="hibernate.hbm2ddl.auto" value="update" />
    </properties>
</persistence-unit>
```

Configuring JPA

```
<bean id="myEmf"
      class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="jpaDialect" ref="jpaDialect"/>
</bean>

<bean id="jpaDialect"
      class="org.springframework.orm.jpa.vendor.HibernateJpaDialect"/>

<bean id="transactionManager"
      class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="myEmf"/>
</bean>
```


Using JPA

Inject an EntityManager

```
@PersistenceContext
EntityManager em;
```

Plain JPA usage with declarative transactions

```
@SuppressWarnings("unchecked")
@Transactional(readOnly=true)
public List<Book> listBooks() {
    Query q = em.createQuery("select b from Book b");
    return q.getResultList();
}
```

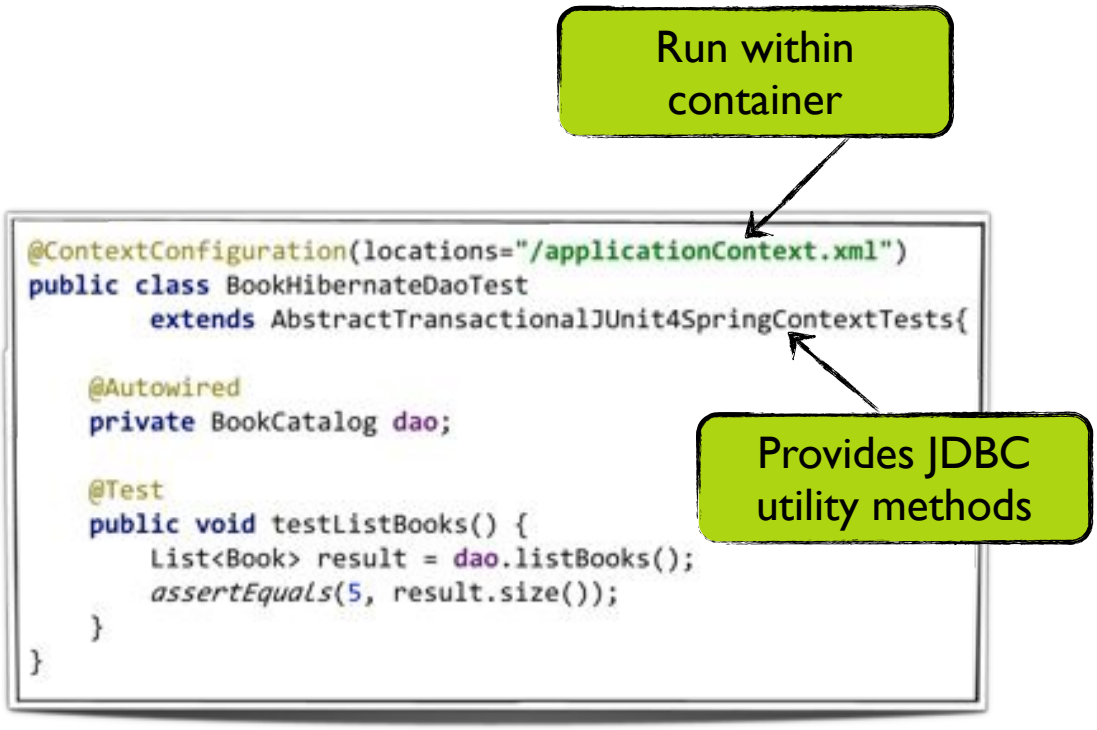
Templates and DaoSupport

- Several Template and DaoSupport classes exists
  - similar to JdbcTemplate
- Not necessary any more because JPA API is already easy to use
- Prefer plain JPA API

Testing DAOs

- Unit testing doesn't make sense
  - queries can only be tested with a real database
- Run automated tests within a Spring container

Testing DAOs



Testing DAOs

- Transactions are rolled back after each test by default
  - no need to re-insert test data for each test
- Test JPA code using JDBC queries

```
@Test
public void testSaveBook() {
    Book book = new Book();
    book.setTitle("Harry Potter");
    int books = countRowsInTable("Book");
    dao.saveBook(book);
    assertEquals(books + 1, countRowsInTable("Book"));
}
```

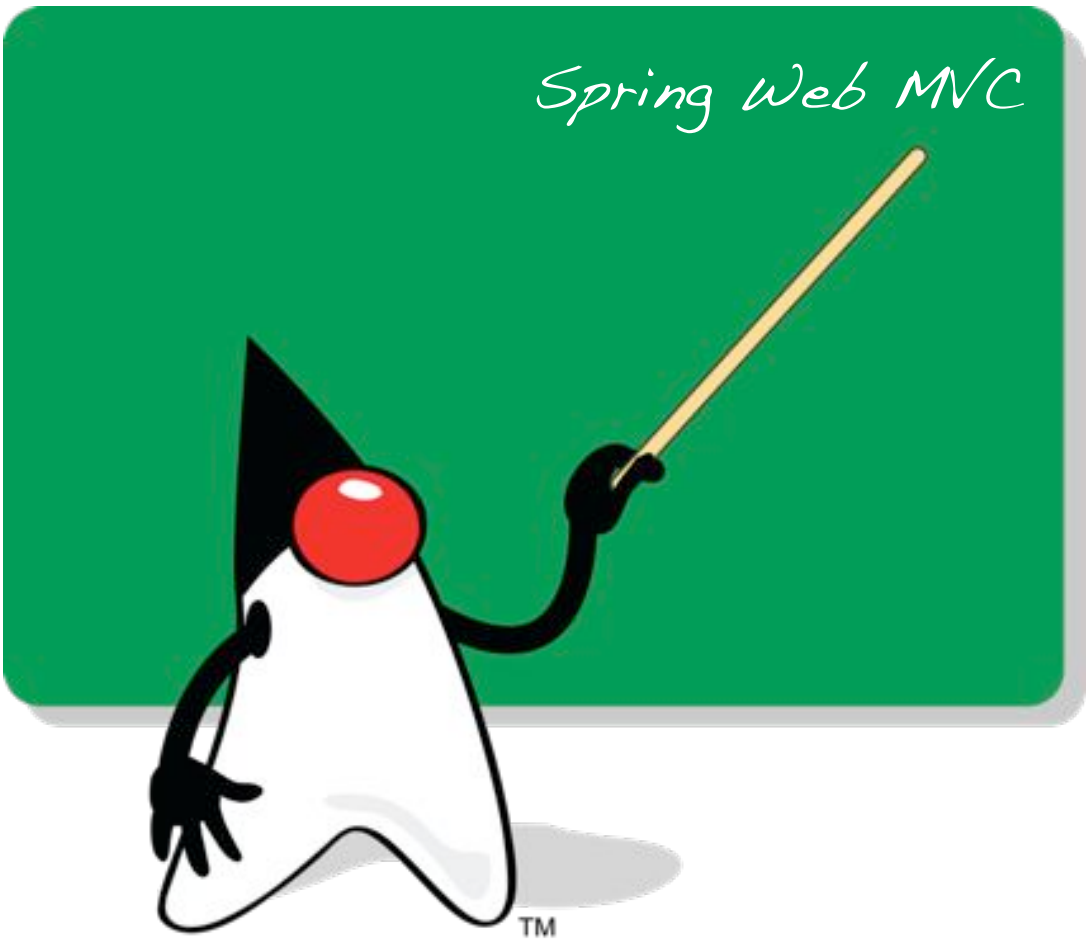
Disable default rollback behavior

Don't roll back transactions for any method

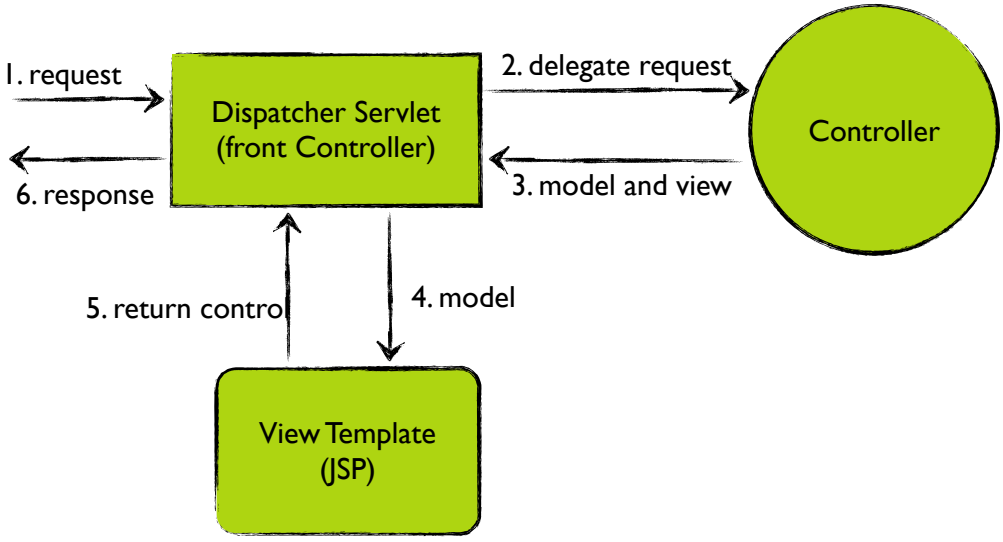
```
@ContextConfiguration(locations="/applicationContext.xml")
@TransactionConfiguration(defaultRollback = false)
public class BookHibernateDaoTest
        extends AbstractTransactionalJUnit4SpringContextTests{
```

Don't roll back transactions for this method

```
@Test @Rollback(false)
public void testSaveBook() {
```

Handling requests



Configuring Web MVC

Add the Dispatcher Servlet to web.xml

Needs a `WEB-INF/spring-servlet.xml` config file

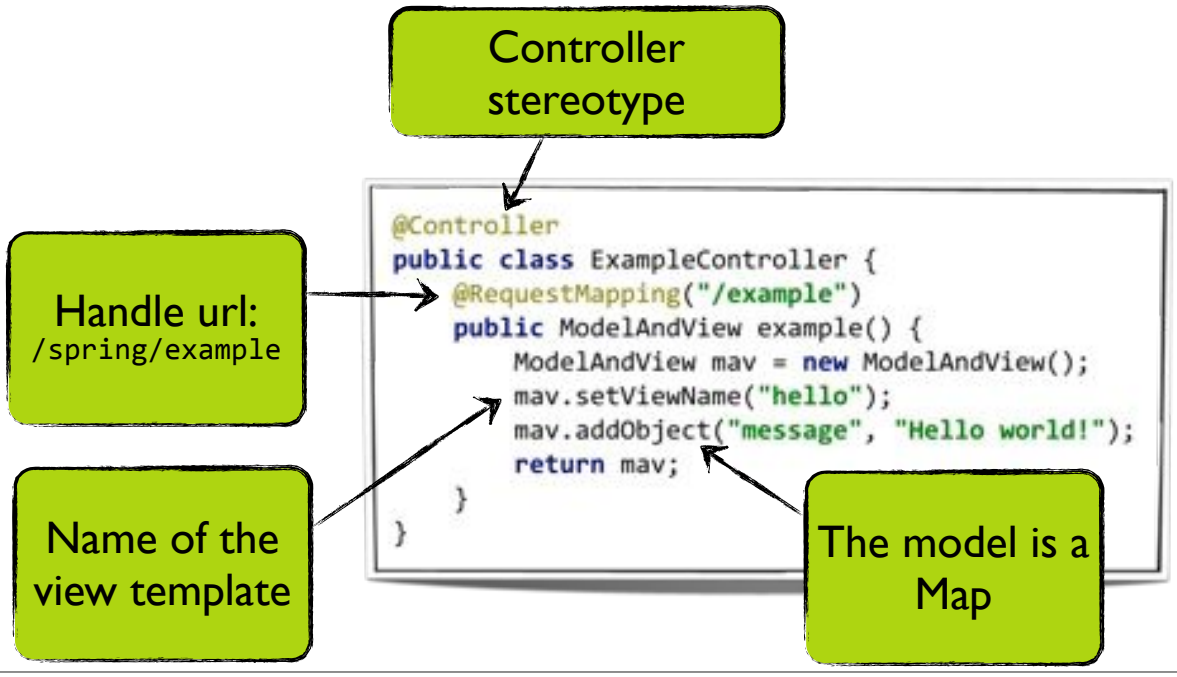
```
<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/spring/*</url-pattern>
</servlet-mapping>
```

Additional configuration files

web.xml

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext.xml
        /WEB-INF/applicationContext-security.xml
    </param-value>
</context-param>

<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
```


Implementing controllers



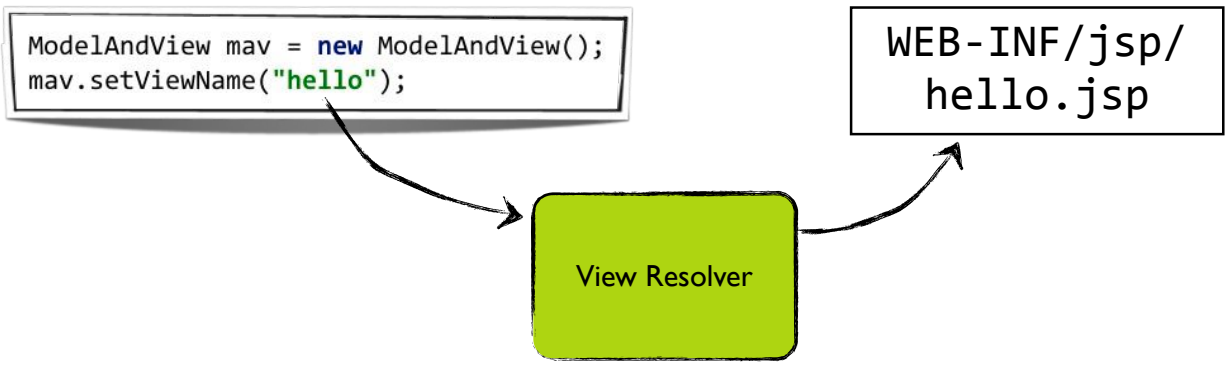
JSP page

- Use Expression Language to render model values
- Use JSTL and Spring tags to make pages dynamic

```
<html>
    <body>
        ${message}
    </body>
</html>
```


View resolvers

- View names are resolved to views using view resolvers



Most useful View resolvers

View resolver	Explanation
`InternalResourceViewResolver`	Servlet and JSP view resolver. View names are prefixed and suffixed to generate file name
`ResourceBundleViewResolver`	Views are defined in views.properties. Supports multiple kinds of views working together (e.g. JSP and Excel)
`ContentNegotiatingViewResolver`	Resolve views based on the *accept* header in the HTTP request. Useful for RESTful web services

There are more view resolvers, but are less likely to be used

Resolving JSP views

```
<bean id="jspViewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass"
              value="org.springframework.web.servlet.view.JstlView"/>
    <property name="prefix" value="/WEB-INF/jsp/"/>
    <property name="suffix" value=".jsp"/>
</bean>
```

Creating PDF documents

```
public class MyPdfView extends AbstractPdfView{
    @Override
    protected void buildPdfDocument(
            Map<String, Object> model,
            com.lowagie.text.Document document,
            com.lowagie.text.pdf.PdfWriter pdfWriter,
            HttpServletRequest httpServletRequest,
            HttpServletResponse httpServletResponse) throws Exception {
        document.add(new Paragraph((String)model.get("message")));
    }
}
```


Resolving PDF views

```
<bean id="pdfViewResolver"
      class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
    <property name="order" value="1"/>
    <property name="basename" value="views"/>
</bean>
```

views.properties

```
mypdf.(class)=dvdstore.views.MyPdfView
```

controller

```
ModelAndView mav = new ModelAndView();
mav.setViewName("mypdf");
```

URI Templates

- URLs can contain variable data
  - `/products/{productId}`
  - `/products/{category}/all`
- Use parameter in request handling

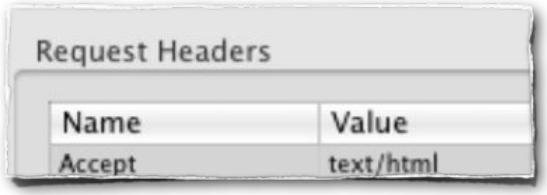
```
@RequestMapping("products/{productId}")
public ModelAndView productDetails(@PathVariable int productId) {
    ModelMap model = new ModelMap();
    model.addAttribute("productId", productId);
    return new ModelAndView("product", model);
}
```


Accessing request parameters

- `/search?query=Spring`

```
@RequestMapping("search")
public void search(@RequestParam(required = true) String query) {
    System.out.println("Query: " + query);
}
```

Accessing request headers




```
@RequestMapping("/example")
public ModelAndView example(@RequestHeader String accept) {
    System.out.println("Accept header: " + accept);
```


Accessing request cookies

```
@RequestMapping("/example")
public ModelAndView example(@CookieValue String userId) {
    System.out.println("Cookie value: " + userId);
```

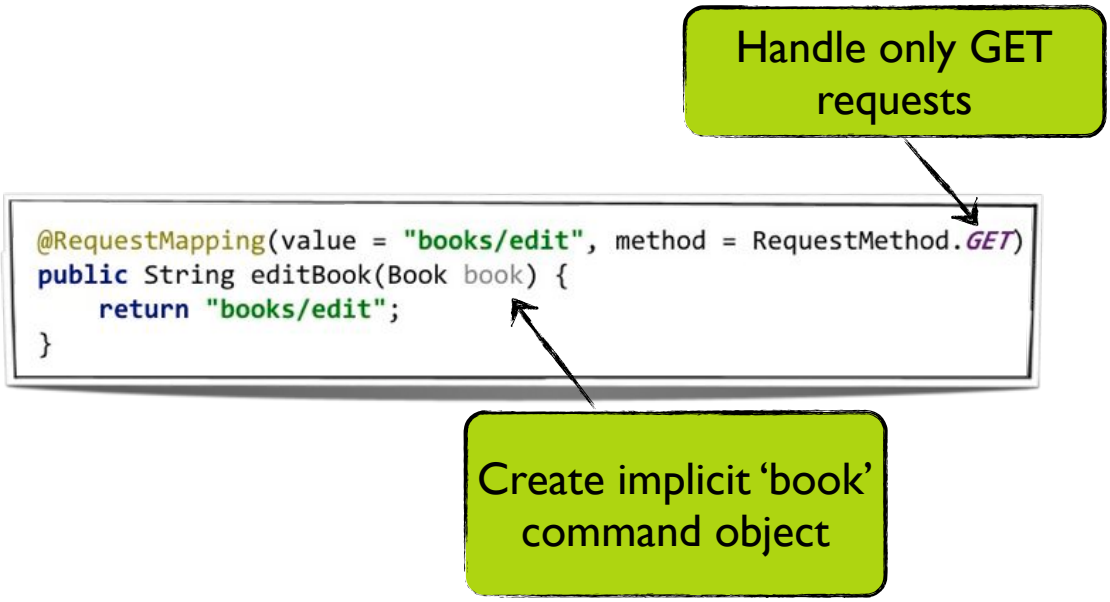
Request method arguments

View resolver	Explanation
`@RequestBody`	The body of the request mapped by a HttpMessageConverter (for RESTful web services)
`HttpServletRequest / HttpServletResponse`	Plain Servlet API request and response
`HttpSession`	The Servlet API session object (think about thread safety)
`Locale`	The current request's locale
`InputStream / Reader`	Raw inputstream to access the request's content
`OutputStream / Write`	Raw outputstream to write response content
`Map / Model / ModelMap`	Implicit model that's exposed to the web view
`Command objects`	Command objects available in the model
`Errors / BindingResult`	Validation results for the preceding command object
`SessionStatus`	Status handle for marking form processing complete, triggering session cleanup

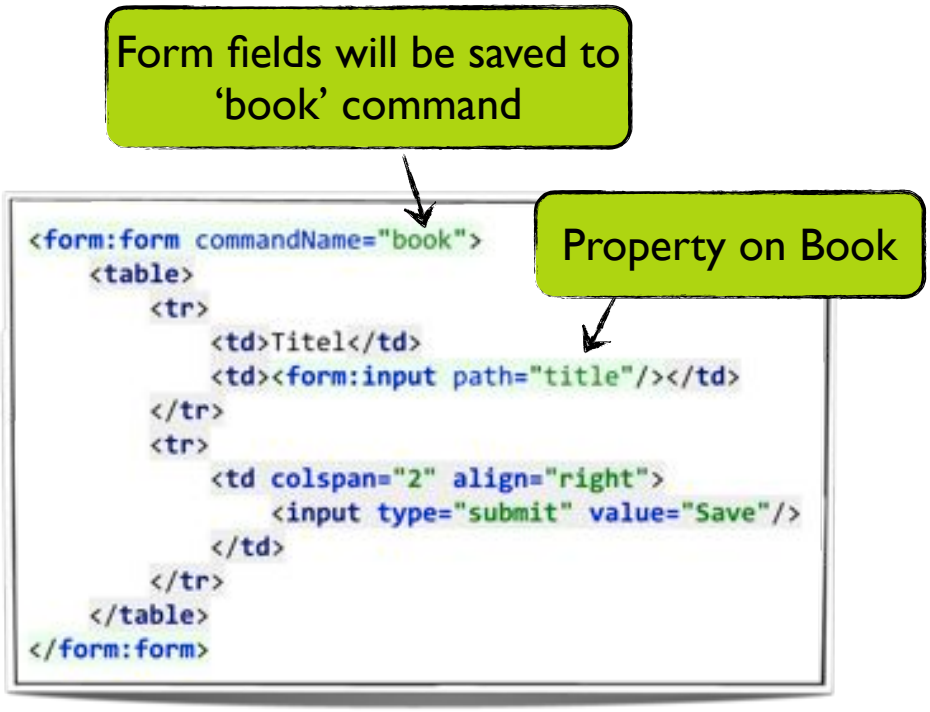
Request method return types

View resolver	Explanation
`ModelAndView`	Combination of explicit model and view object
`Model / Map`	View name is resolved using the `RequestToViewNameTranslator`
`View`	Model resolved using implicit model command objects
`String`	View name. Model is resolved using implicit model command objects
`Void`	Model resolved using implicit model command objects. View name is resolved using the `RequestToViewNameTranslator`
`@ResponseBody`	Object converted using a HttpMessageConverter. Useful for RESTful web services

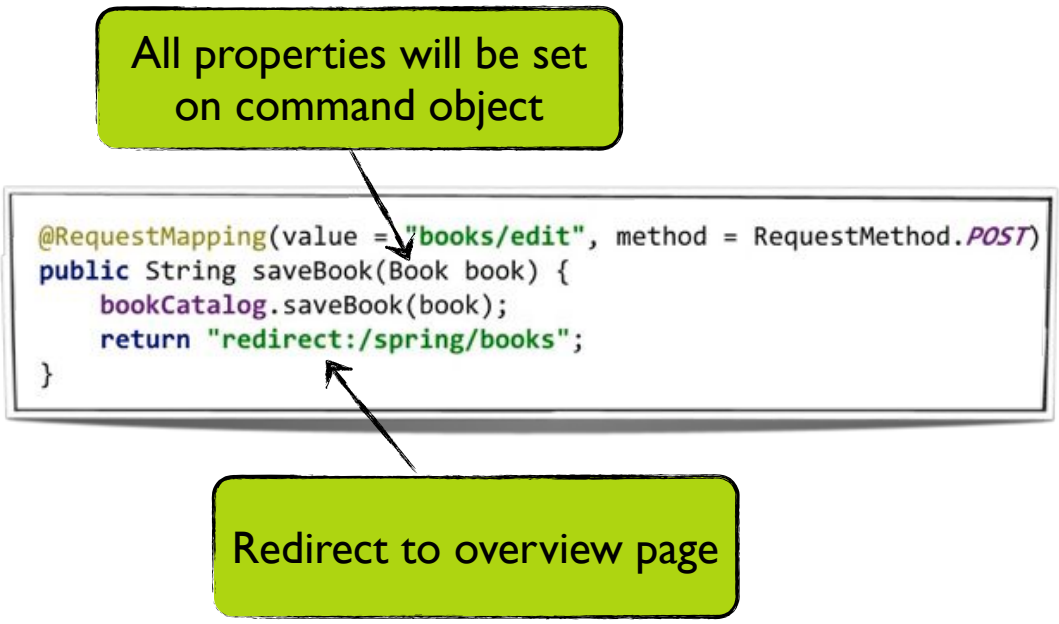
Showing a form



HTML Form



Submitting a form



Bean Validation

- Bean Validation API (JSR-303)
- Define field constraints on Java classes
- Integrates with other frameworks
  - e.g. JSF 2.0, Spring 3, JPA 2

Validation Example

```
@Entity
public class Employee {
    @Id
    private Long id;

    @NotNull @Size(min = 2, max = 20)
    private String name;

    @Past
    private Date birthDate;

    @Min(value = 1000)
    private double salary;
```

- @Null
- @NotNull
- AssertTrue
- AssertFalse
- @Min
- @Max
- @DecimalMin
- @DecimalMax
- @Size
- @Digits
- @Past
- @Future
- @Pattern

Triggering validation

Trigger bean validation

```
@RequestMapping(value = "employees/edit", method = RequestMethod.POST)
public String saveBook(@Valid Employee employee,
                       BindingResult result) {
    if (result.hasErrors()) {
        return "employee/addForm";
    } else {
        //Save employee
        System.out.println("Employee: " + employee.getFirstname());
        return "redirect:/spring/employees";
    }
}
```

Contains error information

Rendering validation errors

```
<form:form commandName="employee">

    <spring:hasBindErrors name="employee">
            <span style="color:red;">
            Er zijn validatiefouten.
            </span>
    </spring:hasBindErrors>

    <table>
        <tr>
            <td>Titel</td>
            <td><form:input path="firstname"/>
                    <span style="color:red;">
                      <form:errors path="firstname"/>
                    </span>
            </td>
        </tr>
```

Check if there are binding errors

Render error for specific property

Custom bean validation invocation

```
validator.validate(Object instance, Class<?>... groups)
```

```
ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
Validator validator = factory.getValidator();

Set<ConstraintViolation<Employee>> violations =
        validator.validate(emp1, Communicating.class);

for (ConstraintViolation<Employee> violation : violations) {
    System.out.println("Error: " + violation.getMessage());
}
```

Creating constraints

- Create annotation
- Implement validator class

```
@ElementCollection
@NonEmptyCollection
private Set<String> emailAddresses;
```

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD, ElementType.FIELD})
@Constraint(validatedBy = NonEmptyCollectionValidator.class)
public @interface NonEmptyCollection {
    String message() default "Collection may not be empty";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
```


Creating constraints

```
public class NonEmptyCollectionValidator implements
        ConstraintValidator<NonEmptyCollection, Collection<?>> {
    @Override
    public void initialize(NonEmptyCollection nonEmptyCollection) {

    }

    @Override
    public boolean isValid(Collection<?> objects,
                           ConstraintValidatorContext
                                   constraintValidatorContext) {
        return objects != null && objects.size() > 0;
    }
}
```

Validation Groups

- Define different sets of constraints for different situations
- Trigger validation only for a certain constraints

```
@ElementCollection
@NonEmptyCollection(groups = Communicating.class)
private Set<String> emailAddresses;
```

```
public interface Communicating {
}
```

```
validator.validate(emp1, Communicating.class);
```


Checkbox tag

```
<form:checkbox path="fulltime" />
```

```
private boolean fulltime;

public boolean getFulltime() {
    return fulltime;
}

public void setFulltime(boolean fulltime) {
    this.fulltime = fulltime;
}
```

Select & options tags

Render from an Enum

```
<form:select path="gender">
    <form:options/>
</form:select>
```

```
private Gender gender;

public Gender getGender() {
    return gender;
}

public void setGender(Gender gender) {
    this.gender = gender;
}
```

```
public enum Gender {
    MALE, FEMALE
}
```


Select & options tags

```
<form:select path="sex">
    <form:option value="M" label="Male"/>
    <form:option value="F" label="Female"/>
</form:select>
```

```
private String sex;

public String getSex() {
    return sex;
}

public void setSex(String sex) {
    this.sex = sex;
}
```

Radiobuttons tag

```
<form:radiobuttons path="role" items="${roles}"/>
```

```
private String role;

public String getRole() {
    return role;
}

public void setRole(String role) {
    this.role = role;
}
```

Method to produce model attributes

```
@ModelAttribute("roles")
public List<String> listRoles() {
    return Arrays.asList("Developer", "Sales", "Management");
}
```


Binders

```
@InitBinder
public void initBinder(WebDataBinder binder) {
    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
    dateFormat.setLenient(false);
    binder.registerCustomEditor(
            Date.class,
            new CustomDateEditor(dateFormat, false));
}
```

```
private Date birthdate;

public Date getBirthdate() {
    return birthdate;
}

public void setBirthdate(Date birthdate) {
    this.birthdate = birthdate;
}
```

```
<form:input path="birthdate"/>
```

Uploading files

```
<form:form commandName="employee"
           enctype="multipart/form-data">
    <input type="file" name="icon"/>
```

```
<bean id="multipartResolver"
      class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <property name="maxUploadSize" value="100000"/>
</bean>
```

```
@RequestMapping(value = "employees/edit", method = RequestMethod.POST)
public String saveBook(@Valid Employee employee,
                       BindingResult result,
                       @RequestParam("icon") MultipartFile file) {

    System.out.println("File: " + file.getOriginalFilename());
```


Handling exceptions

Map exceptions to custom error pages

```
<bean class="...SimpleMappingExceptionResolver">
    <property name="defaultErrorView" value="error"/>
    <property name="exceptionMappings">
        <util:map>
            <entry key="dvdstore.controllers.MyCustomException"
                   value="customerror"/>
        </util:map>
    </property>
</bean>
```

Handling exceptions

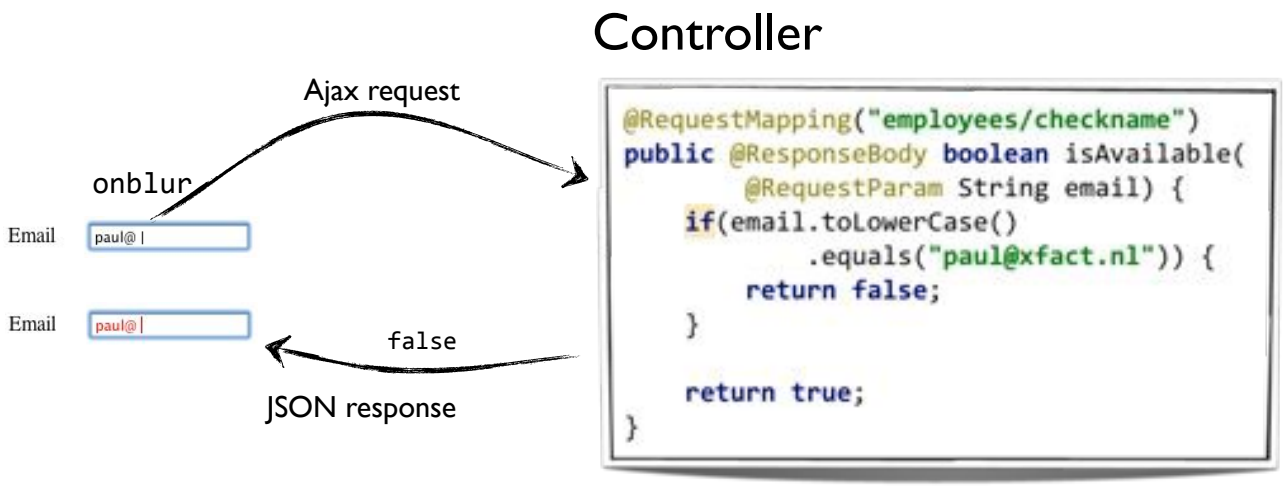
Handle exceptions from code

```
@Component
public class CustomErrorResolver
        implements HandlerExceptionResolver {
    @Override
    public ModelAndView resolveException(HttpServletRequest request,
                                         HttpServletResponse response,
                                         Object handler,
                                         Exception ex) {
        if (ex instanceof MyCustomException) {
            return new ModelAndView("customerror");
        } else {
            return new ModelAndView("error");
        }
    }
}
```


Two types of Ajax

- Data centric
  - server sends back JSON
  - client does all the rendering
- Content centric
  - server sends back HTML
  - client just places the new content in the page

Data centric Ajax



jQuery example

```
<script type="text/javascript">
    $(document).ready(function() {
        $("#email").blur(function() {
            $.getJSON("checkname", {
                email: $("#email").val()
            },
            function(availability) {
                if(!availability) {
                    $("#email").addClass('error');
                }

            });

        });
    });

</script>
```

Interceptors

- Intercept requests before handling
- Similar to Servlet Filters but within the Spring context
- Implement *`HandlerInterceptor`* interface
  - or extend *`HandlerInterceptorAdaptor`* convenience class

Interceptor example

```
public class TimeBasedAccessInterceptor extends
        HandlerInterceptorAdapter {

    @Override
    public boolean preHandle(HttpServletRequest request,
                             HttpServletResponse response,
                             Object handler) throws Exception {
        //implement intercepter logic here
        if (1 > 2) {
            return true;
        } else {
            response.setStatus(403);
            return false;
        }

    }
}
```

Configuring interceptors

```
<mvc:interceptors>
    <bean class="...TimeBasedAccessInterceptor"/>

    <bean class="...ThemeChangeInterceptor">
        <property name="paramName" value="theme"/>
    </bean>

    <bean class="...LocaleChangeInterceptor">
        <property name="paramName" value="lang"/>
    </bean>
</mvc:interceptors>
```


Localization

- Retrieve locale from browser

```
<bean id="localeResolver"
      class="...AcceptHeaderLocaleResolver"/>
```

- Retrieve locale from cookie

```
<bean id="localeResolver"
      class="...CookieLocaleResolver"/>
```

Localization

- Set locale from request parameter
  - e.g. /products?lang=nl
- Can't be used with Accept Header

```
<mvc:interceptors>
    <bean class="...LocaleChangeInterceptor">
      <property name="paramName" value="lang"/>
    </bean>
</mvc:interceptors>
```


Themes

- Themes allow retrieving stylesheet names etc. from property files

```
<link rel="stylesheet" href="<spring:theme code='css'/>" type="text/css"/>
```

black.properties

```
css=/css/black.css
```

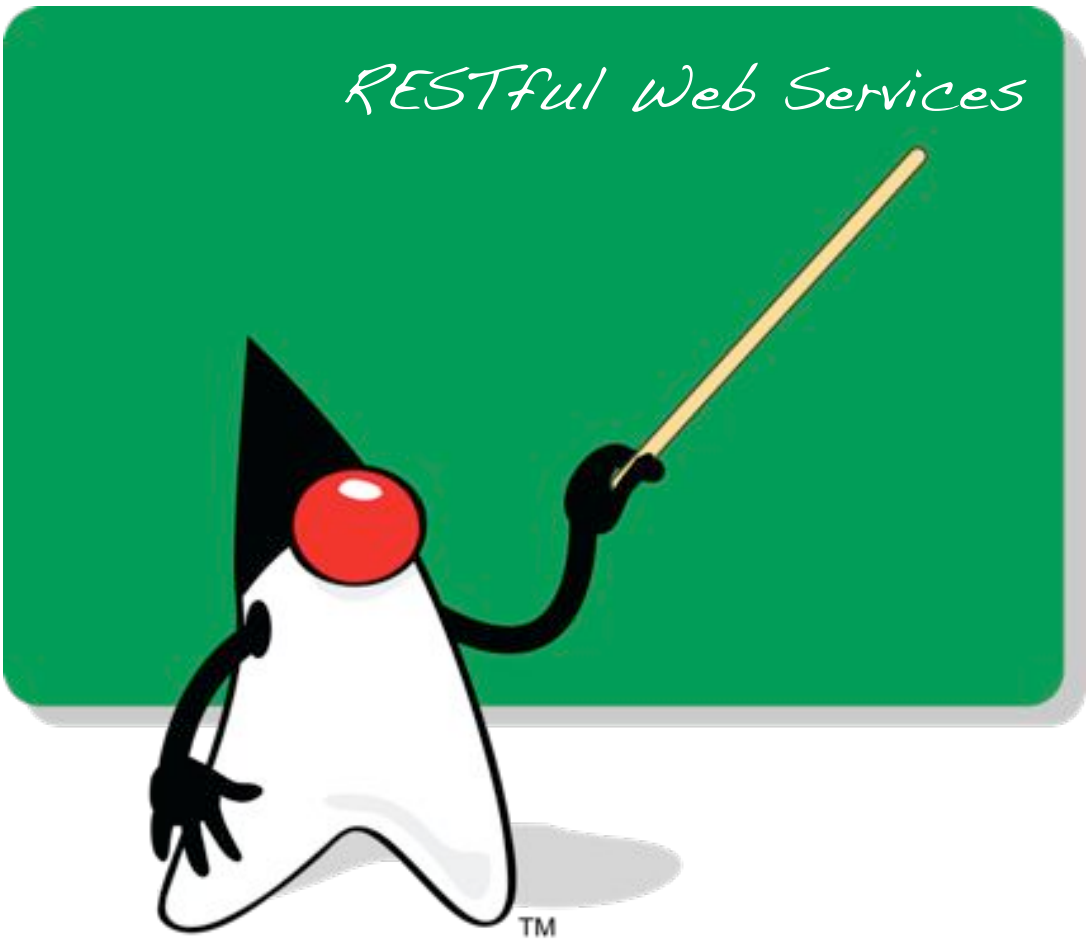
white.properties

```
css=/css/white.css
```

Resolving themes

```
<bean name="themeResolver" class="...CookieThemeResolver">
    <property name="defaultThemeName" value="white"/>
</bean>
```

```
<mvc:interceptors>
    <bean class="...ThemeChangeInterceptor">
        <property name="paramName" value="theme"/>
    </bean>
</mvc:interceptors>
```

REST fact sheet

- **RE**presentational **S**tate **T**ransfer
- Introduced by Roy Fielding
  - Dissertation in 2000
  - An architectural style for distributed systems
- HTTP is an example of REST

RESTful web services

- Services implemented conform the REST principles
- Mostly based at HTTP

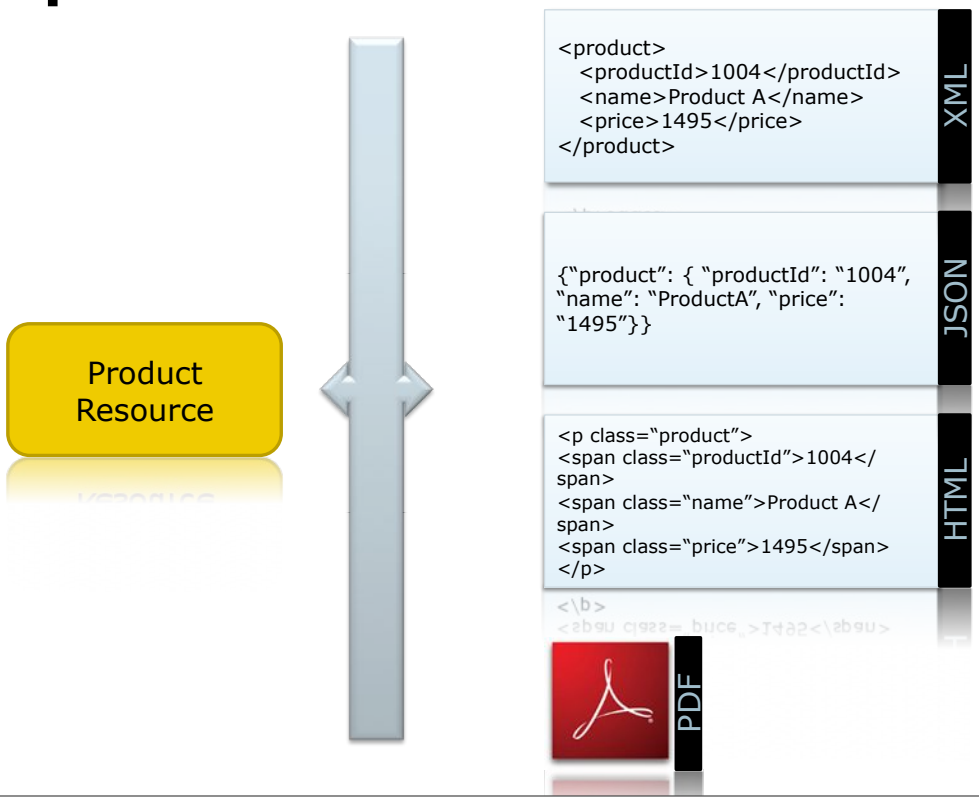
The REST hype

- More public web APIs
  - Google, Amazon, Flickr etc.
- Popularity of lightweight web frameworks
  - Rails / Grails
- People are tired of WSDL
- XML is not always the best format

Everything is a resource

- A list of books
- A product
- A list of search results
- An order

Representations



HTTP content negotiation

- A client can ask for specific formats
- The accept header
  - Accept: "application/xml"

Dynamic resources

- A resource can be ‘static’
  - A record in your database
  - A file
- A resource can be ‘dynamic’
  - Calculated results
  - Generated data

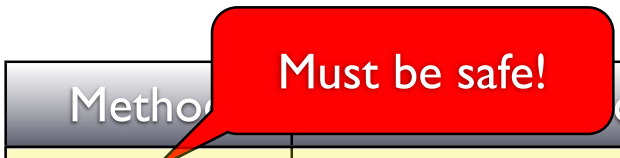
RESTful properties

- Uniform Interface
- Addressability
- Connectedness
- Statelessness

Uniform Interface

Method	Description
GET	Retrieve a resource representation
PUT	Add or modify a resource with a specified URI
DELETE	Delete a resource
HEAD	GET without body: "Does this resource exists?"
POST	Overloaded: implementation may vary. Might generate a new URI

Uniform Interface



Metho	cription
GET	Retrieve a resource representation
PUT	Add or modify a resource with a specified URI
DELETE	Delete a resource
HEAD	GET without body: "Does this resource exists?"
POST	Overloaded: implementation may vary. Might generate a new URI

Uniform Interface

Method	Description
GET	Retrieve a resource representation
PUT	Add or modify a resource with a specified URI
DELETE	Delete a resource
HEAD	GET without body: "Does this resource exists?"
POST	Overloaded: implementation may vary. Might generate a new URI

Uniform Interface

Method	Description
GET	Retrieve a resource representation
PUT	Add or modify a resource with a specified URI
DELETE	Delete a resource
HEAD	GET without body: "Does this resource exists?"
POST	Overloaded: implementation may vary. Might generate a new URI

Uniform Interface

Method	Description
GET	Retrieve a resource representation
PUT	Add or modify a resource with a specified URI
DELETE	Delete a resource
HEAD	GET without body: "Does this resource exists?"
POST	Overloaded: implementation may vary. Might generate a new URI

Uniform Interface

Method	Description
GET	Retrieve a resource representation
PUT	Add or modify a resource with a specified URI
DELETE	Delete a resource
HEAD	GET without body: "Does this resource exists?"
POST	Overloaded: implementation may vary. Might generate a new URI

Addressability

Each resource has a Unique Resource Identifier (URI)

- /products
- /product/{id} => /product/10
- /products?color=red
- /search?q=jax-rs

Connectedness

- Navigate from one resource to another
- Clients do not generate URIs
- One of the most important WEB concepts
  - Hyperlinks

Not connected: How do I get product information?

```
<searchresult>
   <product name="Product 1"/>
   <product name="Product 2"/>
   <product name="Product 3"/>
   <product name="Product 4"/>
</searchresult>
```


Connected: Linked to more information

```
<searchresult>
   <product name="Product 1" url="http://myservice/product/1"/>
   <product name="Product 2" url="http://myservice/product/2"/>
   <product name="Product 3" url="http://myservice/product/3"/>
   <product name="Product 4" url="http://myservice/product/4"/>
</searchresult>
```

Back to the SOAP age

A SOAP product service

/ProductService

- API
  - saveProduct
  - getProduct
  - deleteProduct
  - getAllProducts
  - searchProducts


```
<definitions targetNamespace="http://demo/" name="ProductServiceService" xmlns="http://
schemas.xmlsoap.org/wsdl/" xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:tns="http://demo/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://
schemas.xmlsoap.org/wsdl/soap/" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd">
<types>
  <xsd:schema>
    <xsd:import namespace="http://demo/" schemaLocation="ProductServiceService_schema1.xsd"/>
  </xsd:schema>
</types>
<message name="addProduct">
  <part name="parameters" element="tns:addProduct"/>
</message>
<message name="addProductResponse">
  <part name="parameters" element="tns:addProductResponse"/>
</message>
<portType name="ProductService">
  <operation name="addProduct">
    <input message="tns:addProduct"/>
    <output message="tns:addProductResponse"/>
  </operation>
</portType>
<binding name="ProductServicePortBinding" type="tns:ProductService">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  <operation name="addProduct">
    <soap:operation soapAction=""/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
<service name="ProductServiceService">
  <port name="ProductServicePort" binding="tns:ProductServicePortBinding">
    <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
  </port>
</service>
```

Does this feel like the web?

The RESTful alternative

Resources

/products
/products/{id}
/products/search

The interface

GET (a product)
PUT (a product)
DELETE (a product)

Discussion

Design a RESTful API for a web shop

- ☐ Browse products
- ☐ Search products
- ☐ Order products
- ☐ Track a customer's order history

Don't we need a service description?

- WSDL is a technical interface that lists methods
- WSDL is **NOT** service documentation
- XML Schema is still an option

Don't we need a service description?

- WSDL is a technical interface that lists methods
- WSDL is **NOT** service documentation
- XML Schema is still an option

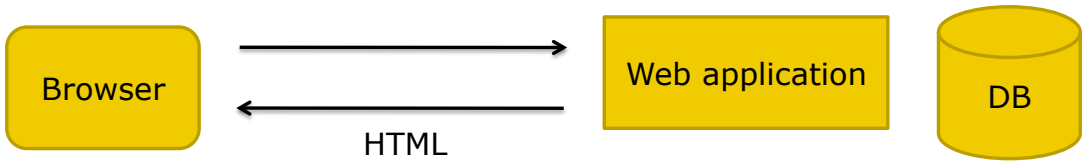
We DO need service documentation

We DON'T need a technical interface description

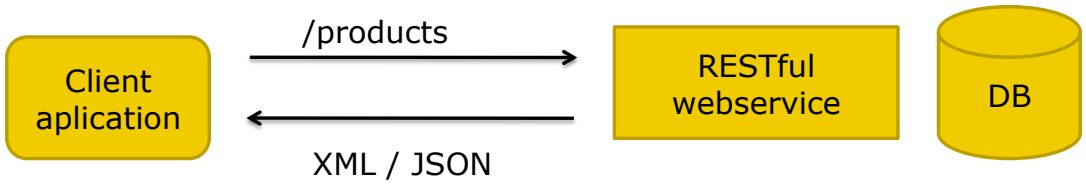
We MIGHT need to provide XML schemas

166

Implementing a web application



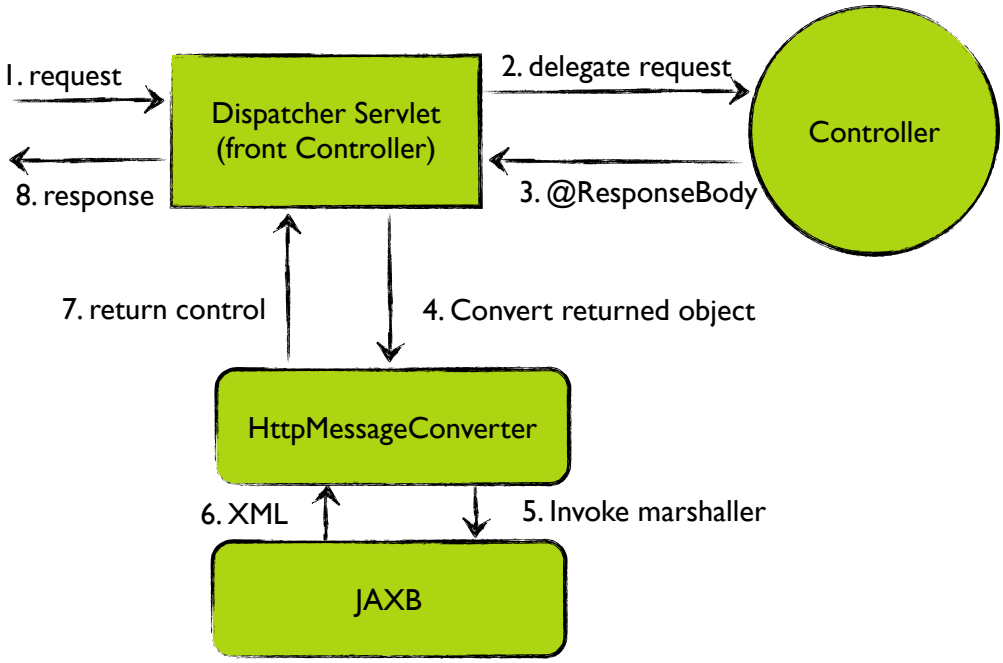
Implementing a web service



RESTful web services in Spring

- Web Services are implemented using controllers
- Familiar Spring MVC programming model

Handling web service requests



@ResponseBody

- The object returned is converted using a HttpMessageConverter
  - Jaxb2RootElementHttpMessageConverter
  - MappingJacksonHttpMessageConverter
  - StringHttpMessageConverter
  - etc.

```
@RequestMapping(method = RequestMethod.GET,
        value = "books",
        headers = "accept=application/xml")
public @ResponseBody BookList listBooksXml() {
    List<Book> books = bookCatalog.listBooks();
    return new BookList(books);
}
```

```
@XmlRootElement
public class Book {
```


Choosing handlers

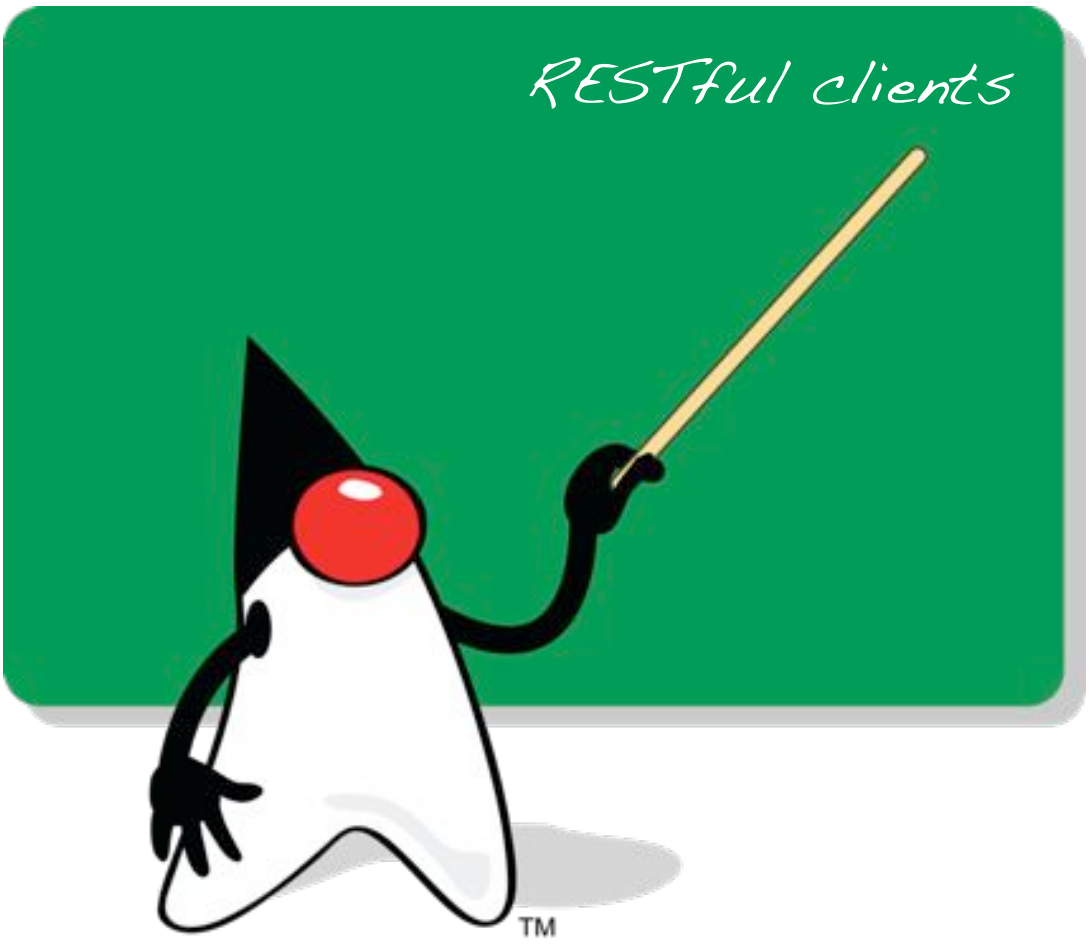
- How to offer data both as XML and HTML?
  - use the HTTP accept header
  - use a different extension
  - use a request parameter
  - use content negotiation

Choosing handlers

```
@RequestMapping(method = RequestMethod.GET,
        value = "books.xml")
```

```
@RequestMapping(method = RequestMethod.GET,
        value = "books",
        headers = "accept=application/xml")
```

```
@RequestMapping(method = RequestMethod.GET,
        value = "books",
        params = "contentType=application/xml")
```

RESTful clients

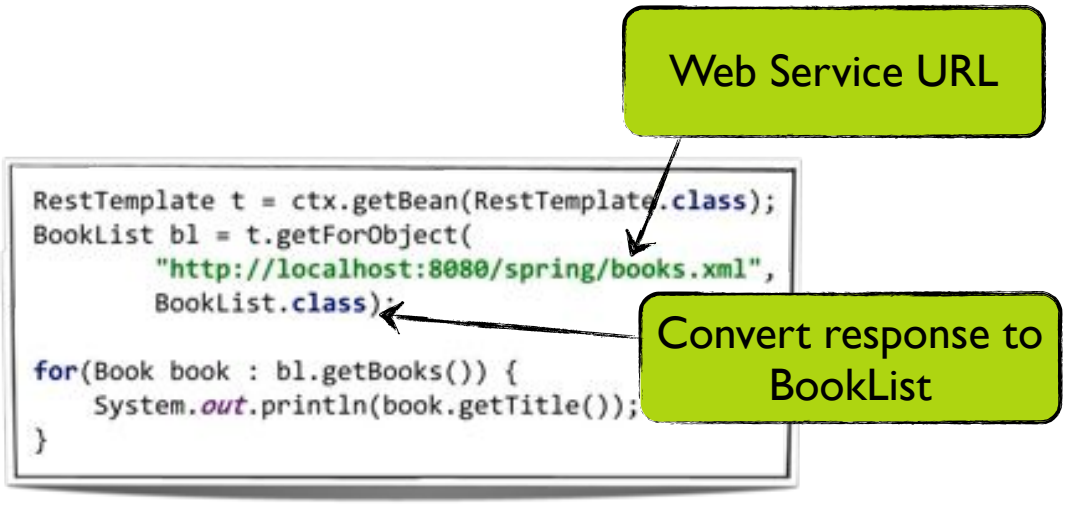
- Use RESTful web services using a template
- Configure template to convert messages

```
<bean id="restTemplate"
      class="org.springframework.web.client.RestTemplate">
    <property name="messageConverters">
        <array>
            <ref bean="bookConverter"/>
        </array>
    </property>
</bean>
```


Message converters

```
<bean id="bookConverter"
      class="...MarshallingHttpMessageConverter">
    <property name="marshaller" ref="marshaller"/>
    <property name="unmarshaller" ref="marshaller"/>
</bean>

<oxm:jaxb2-marshaller id="marshaller">
    <oxm:class-to-be-bound name="dvdstore.domain.Book"/>
    <oxm:class-to-be-bound name="dvdstore.domain.BookList"/>
</oxm:jaxb2-marshaller>
```

Calling a service



Putting data

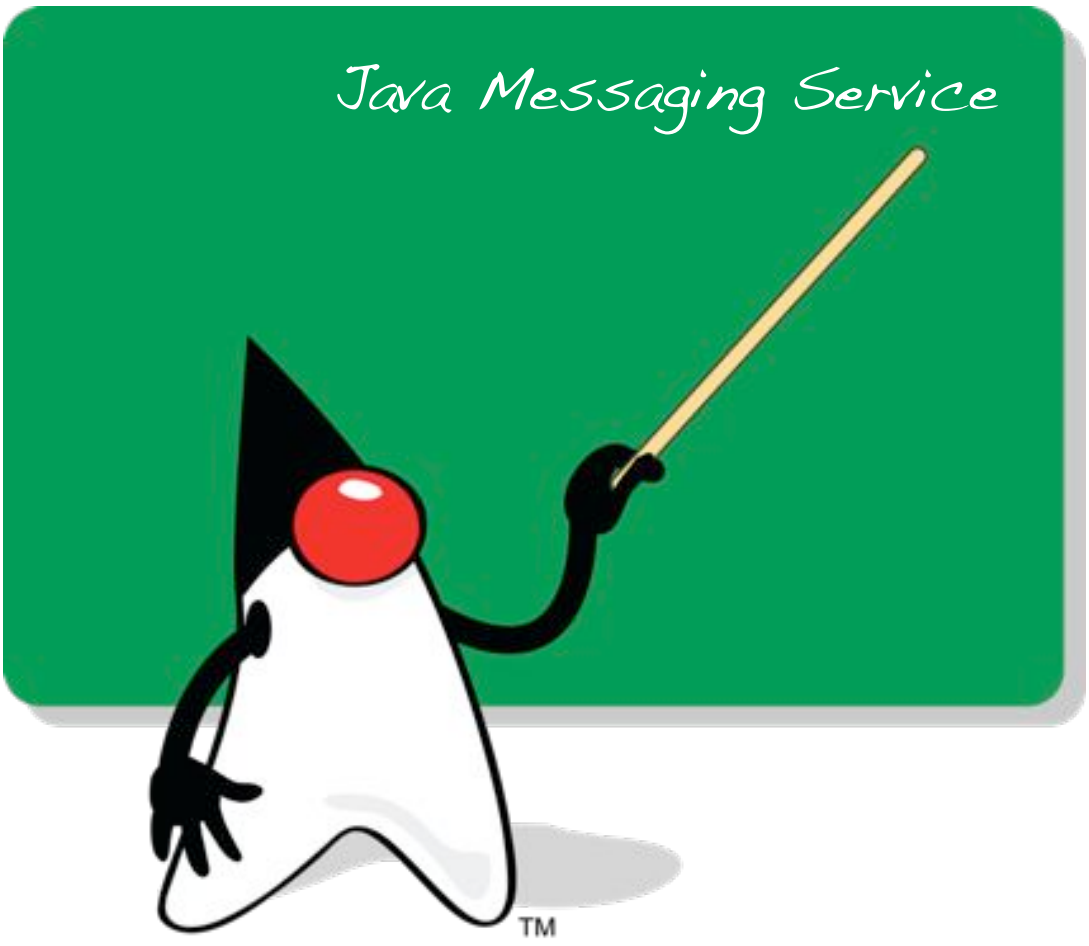
```
@RequestMapping(value = "books/{bookId}", method = RequestMethod.PUT)
public void updateBookFromXML(@PathVariable long bookId,
                              @RequestBody Book book,
                              HttpServletResponse response) {

    System.out.println("Saving book (" + bookId +"): " + book.getTitle());

    book.setId(bookId);
    bookCatalog.saveBook(book);
    response.setStatus(200);
}
```

```
RestTemplate t = ctx.getBean(RestTemplate.class);
final Book book = new Book();
book.setTitle("jQuery in Action");
t.put("http://localhost:8080/spring/books/1", book);
```

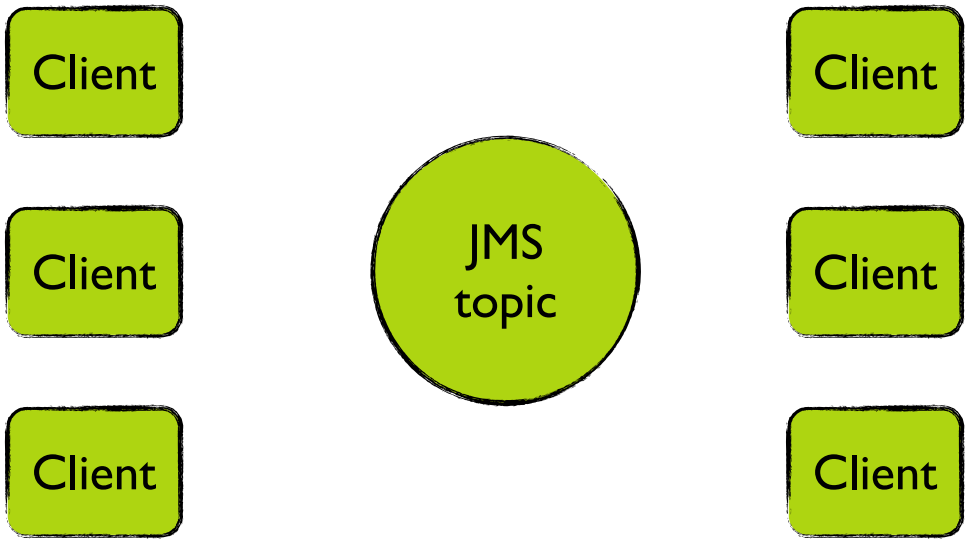
Posting data

```
@RequestMapping(method = RequestMethod.POST)
public
@ResponseBody
Book saveBook(@RequestBody Book bookXML) {
    System.out.println("Saving book" + bookXML.getTitle());
    Book book = bookCatalog.saveBook(bookXML);

    return book;
}
```

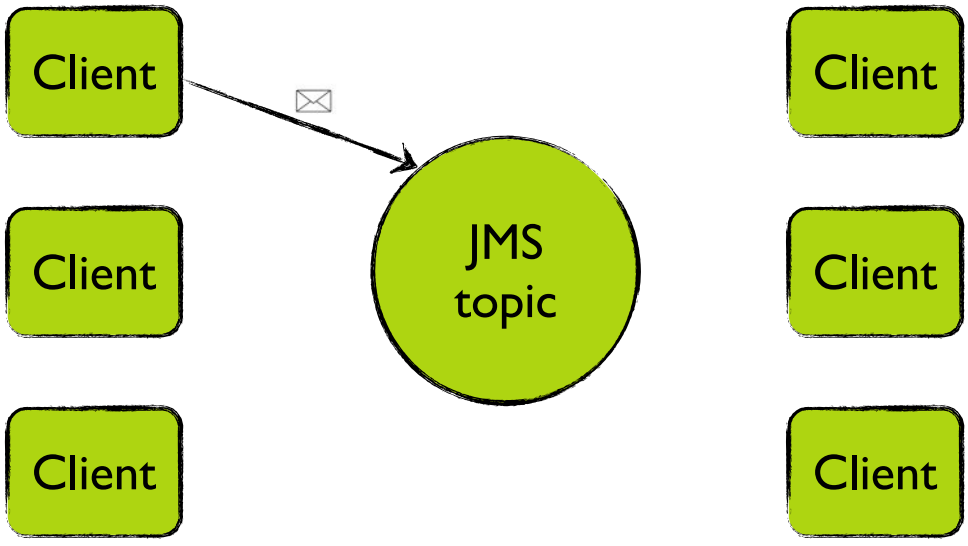
```
RestTemplate t = ctx.getBean(RestTemplate.class);
final Book book = new Book();
book.setTitle("jQuery in Action");
final URI newUrl = t.postForLocation(
        "http://localhost:8080/spring/books",
        book);
System.out.println("Book saved to: " + newUrl);
```

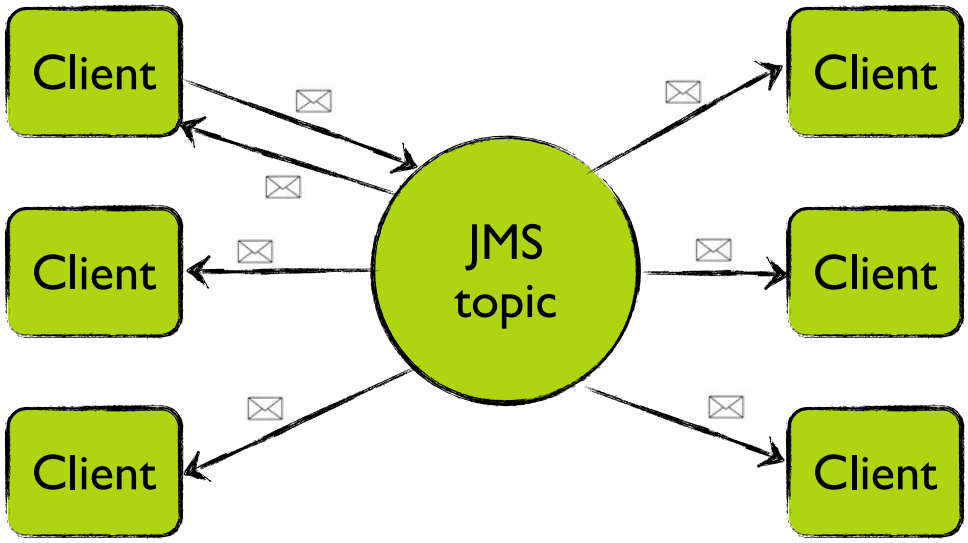
JMS Topics



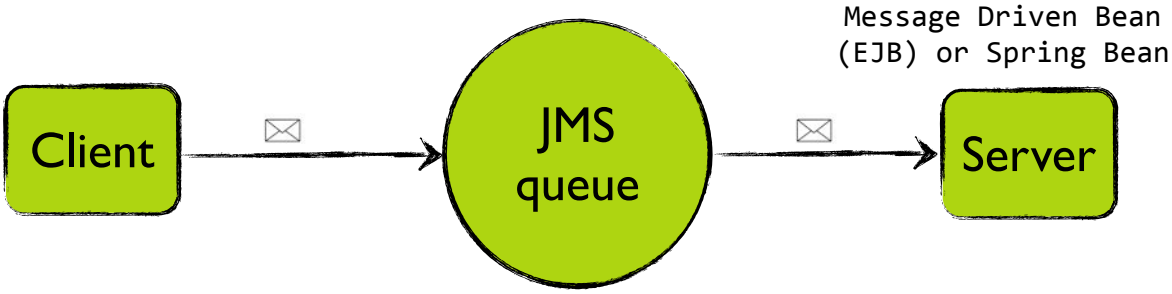
JMS Topics



JMS Topics

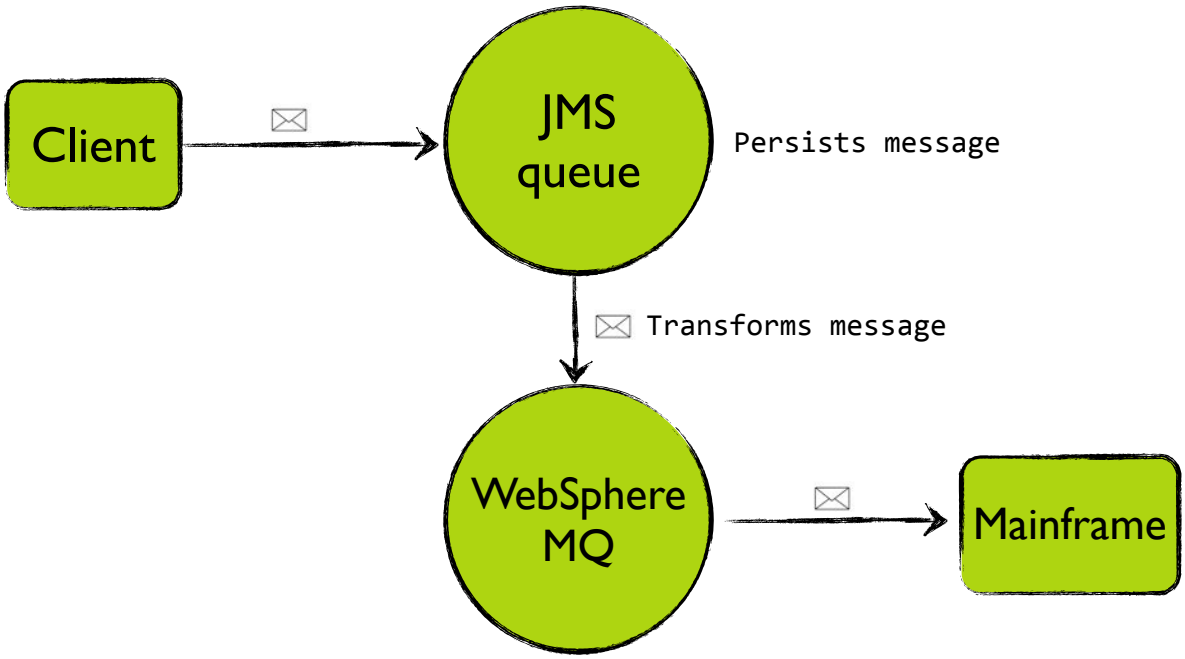


JMS Queues



Persists message

JMS Queues



Message Listener

```
@Component
public class BookJmsListener implements MessageListener{
    @Override
    public void onMessage(Message message) {
        if(message instanceof TextMessage) {
            TextMessage txt = (TextMessage)message;
            try {
                System.out.println("MSG: " + txt.getText());
            } catch (JMSException e) {
                e.printStackTrace();
            }
        } else {
            System.out.println("Received unknown message");
        }

    }
}
```

JMS connection factory

```
<bean id="connectionFactory"
      class="...PooledConnectionFactory"
      destroy-method="stop">
    <property name="connectionFactory">
        <bean class="org.apache.activemq.ActiveMQConnectionFactory">
            <property name="brokerURL">
                <value>tcp://localhost:61616</value>
            </property>
        </bean>
    </property>
</bean>
```

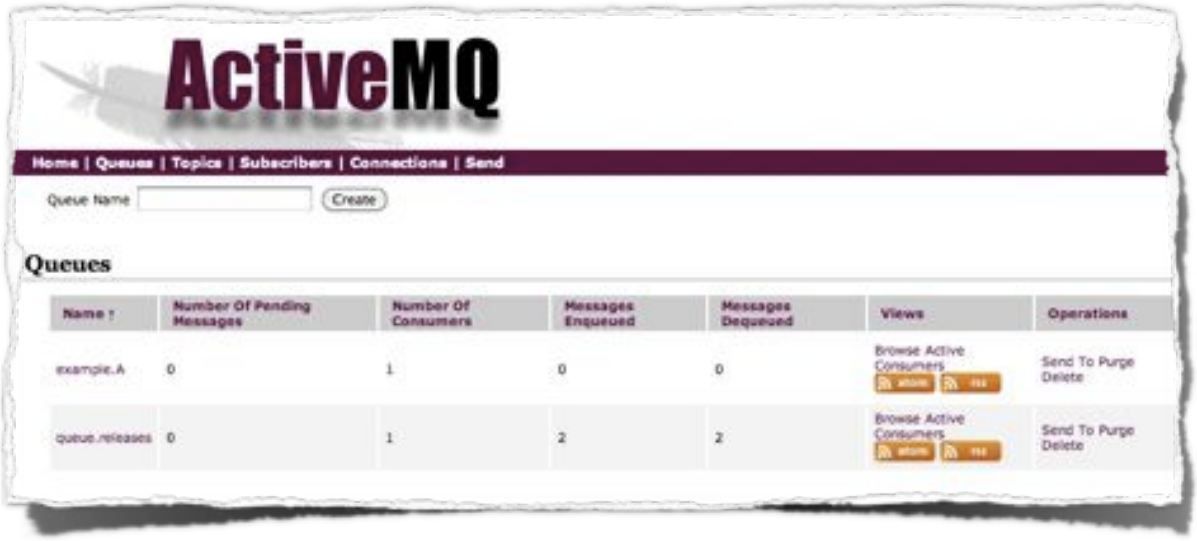

JMS listener

```
<jms:listener-container>
    <jms:listener destination="queue.releases"
                  ref="bookJmsListener"/>
</jms:listener-container>
```

Active MQ installation

- Start `bin/activemq.bat`
- Open `http://localhost:8161`
- Create new topic / queue

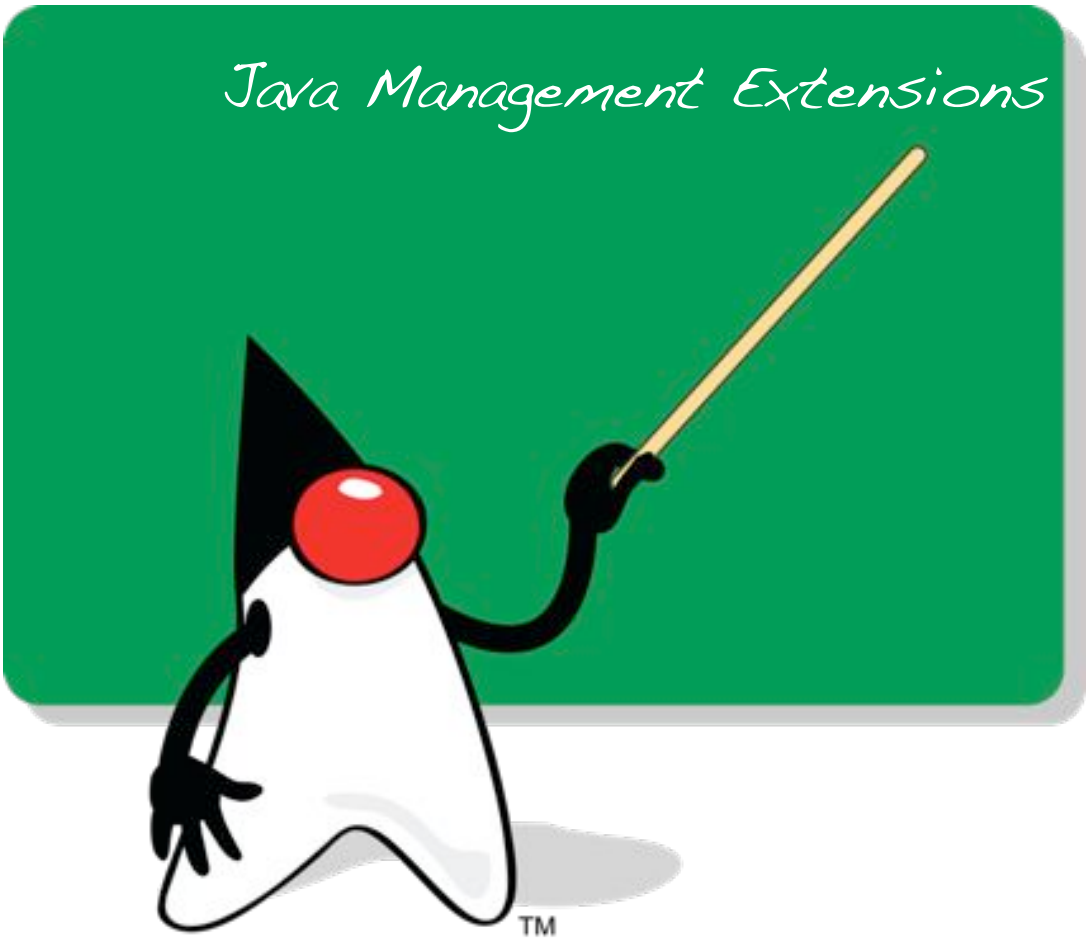
Active MQ installation



Sending test messages

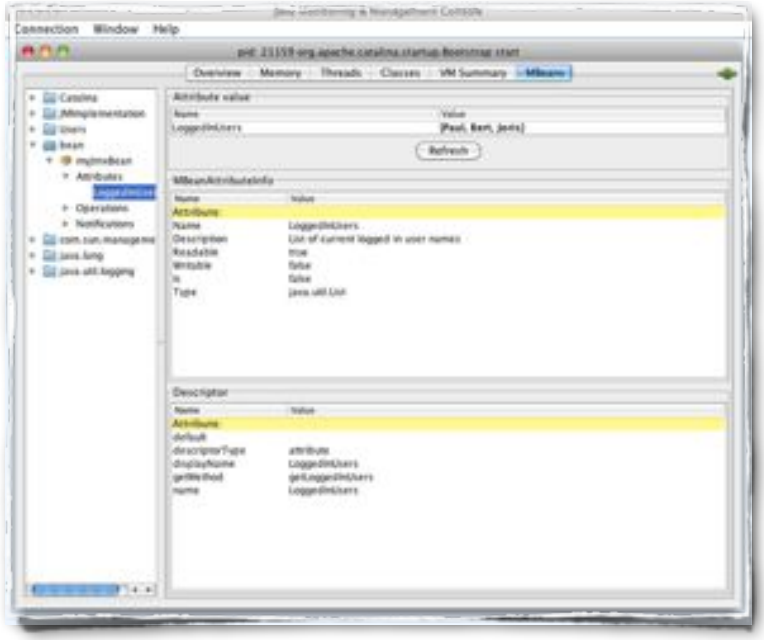
Sending messages

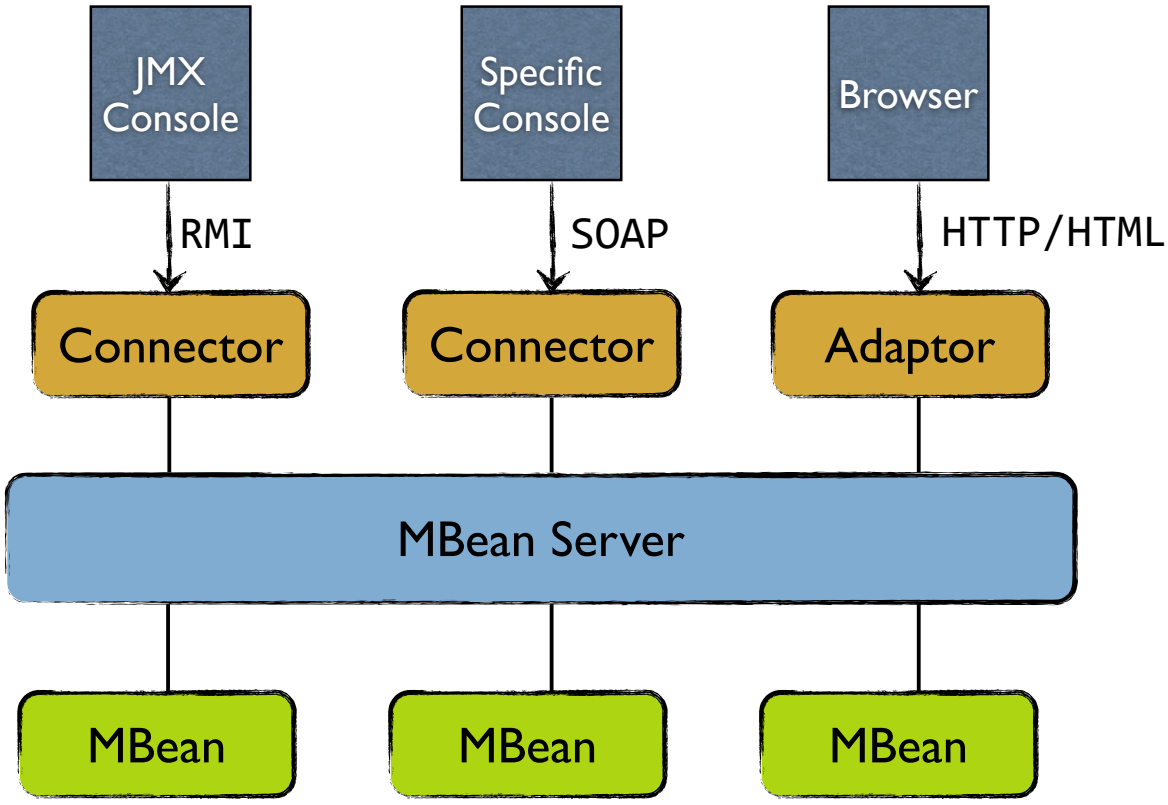
```
@Component
public class BookReleaseSender {
    private JmsTemplate jmsTemplate;

    @Autowired
    public void setConnectionFactory(ConnectionFactory cf) {
        this.jmsTemplate = new JmsTemplate(cf);
    }

    public void sendMessage(final Book book) {
        jmsTemplate.send("queue.releases", new MessageCreator() {
            @Override
            public Message createMessage(Session session) throws JMSException {
                return session.createTextMessage("New book: " + book.getTitle());
            }
        });
    }
}
```



JMX overview

- Manage and monitor Java applications using standard tools





Creating MBeans

Just a standard Spring bean

All public properties are exported

```
@Component("jmxBookManager")
public class JMXBookManager {
    @Autowired
    private BookCatalog bookCatalog;

    public int getNumberOfBooks() {
        return bookCatalog.listBooks().size();
    }

    public String[] getBookNames() {
        ...

        return books.toArray(new String[books.size()]);
    }
}
```

Exporting MBeans

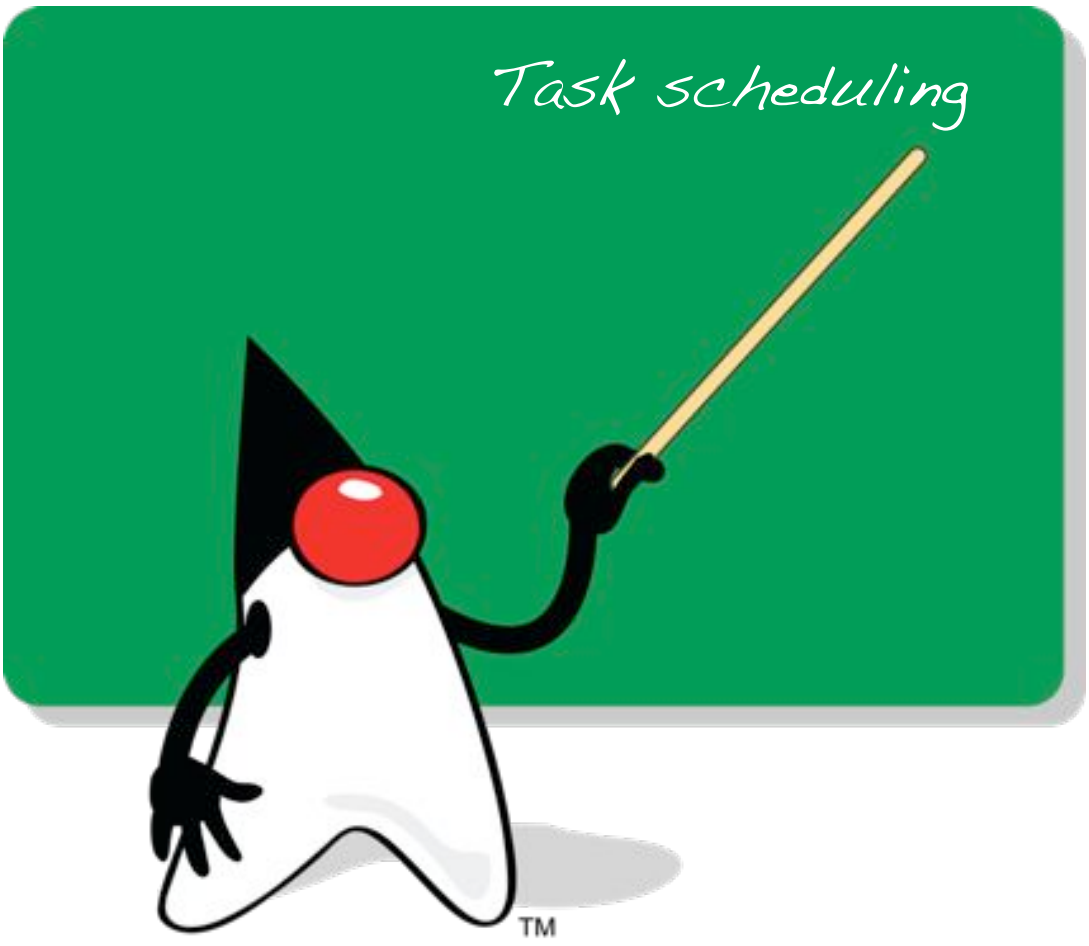
```
<bean id="exporter" class="...MBeanExporter" lazy-init="false">
    <property name="beans">
        <map>
            <entry key="bean:name=bookManager"
                   value-ref="jmxBookManager"/>
        </map>
    </property>
</bean>
```


Configuring what to export

```
@Component
@ManagedResource(objectName = "bean:name=myJmxBean",
        description = "My Managed Bean")
public class JMXExample {
    public String shouldNotBeExported() {
        return "not good";
    }

    @ManagedAttribute(description = "List of ...")
    public List<String> getLoggedInUsers() {
        return Arrays.asList("Paul", "Bert", "Joris");
    }
}
```

Using annotations to export

```
<bean id="exporter" class="...MBeanExporter">
    <property name="assembler" ref="assembler"/>
    <property name="namingStrategy" ref="namingStrategy"/>
    <property name="autodetect" value="true"/>
</bean>

<bean id="jmxAttributeSource"
      class="...AnnotationJmxAttributeSource"/>

<bean id="assembler"
      class="...MetadataMBeanInfoAssembler">
    <property name="attributeSource" ref="jmxAttributeSource"/>
</bean>

<bean id="namingStrategy"
      class="...MetadataNamingStrategy">
    <property name="attributeSource" ref="jmxAttributeSource"/>
</bean>
```

Scheduling tasks

```
<task:scheduler id="scheduler" pool-size="10"/>

<task:scheduled-tasks scheduler="scheduler">
    <task:scheduled ref="pingJob" method="ping"
                    cron="0 0/15 * * * *"/>
</task:scheduled-tasks>
```

```
@Component
public class PingJob {
    public void ping() {
        System.out.println("Ping!");
    }
}
```


Cron syntax

1. Seconds
2. Minutes
3. Hours
4. Day-of-Month
5. Month
6. Day-of-Week
7. Year (optional field)

`0 0 12 * * ?`	Fire at 12pm (noon) every day
`0 * 14 * * ?`	Fire every minute starting at 2pm and ending at 2:59pm, every day
`0 0-5 14 * * ?`	Fire every minute starting at 2pm and ending at 2:05pm, every day
`0 15 10 ? * MON-FRI`	Fire at 10:15am every Monday, Tuesday, Wednesday, Thursday and Friday
`0 0/5 14 * * ?`	Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day

Annotation-driven tasks

```
<task:scheduler id="scheduler" pool-size="10"/>

<task:annotation-driven scheduler="scheduler"/>
```

```
@Component
public class AnnotationJob {

    //Schedule every 5 seconds
    @Scheduled(fixedDelay = 5000)
    public void doSomething() {
        System.out.println("I'm an annotated task");
    }
}
```