Spring/REST

Learn to develop Enterprise Applications using the Spring Framework

SAP Development University

Resources

Spring Framework Reference Documentation

Organization

Prerequisites:

Java Basics

Method:

Presentation, discussion, exercises (> 30%)

Tools:

Java 8

Spring 5.x

STS/Eclipse/IntelliJ

Apache Maven

JUnit

Duration:

16 lessons, 90’ each

1

SPRING BASICS

1.1

SETUP

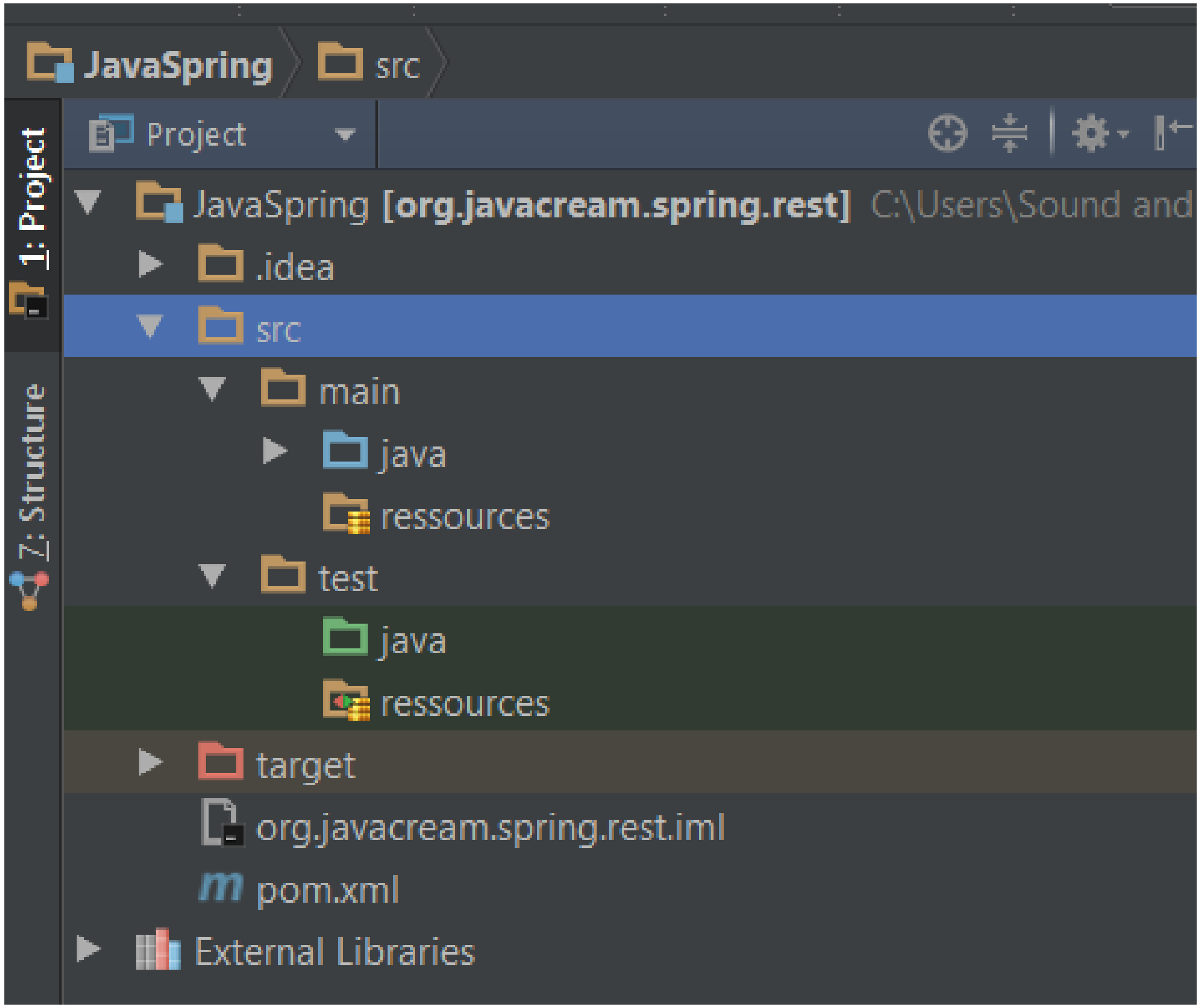
Dependencies

- Spring is a huge framework
    - a lot of dependencies
- Spring Boot provides predefined Maven POMs
    - Parent POM
    - "Starter" POMs declaring dependencies matching a technology stack
        - Web applications
        - JPA
        - …

POM

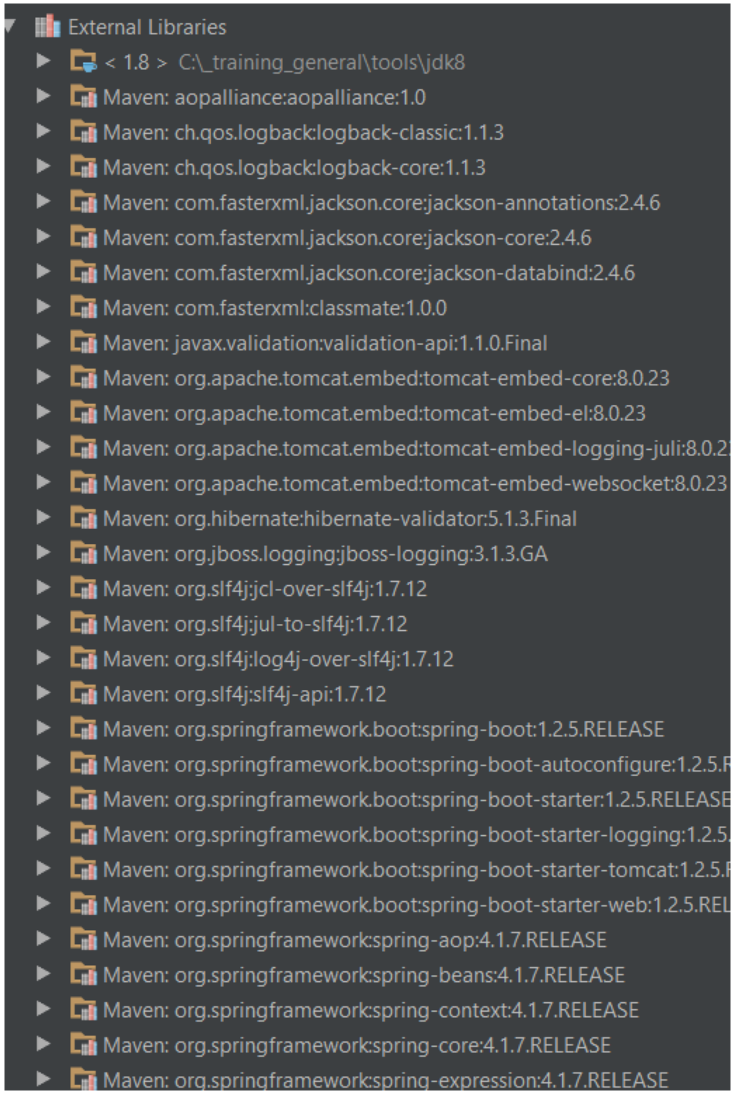
```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.javacream.</groupId>
<artifactId>myproject</artifactId>
<version>0.0.1-SNAPSHOT</version>
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>1.2.5.RELEASE</version>
</parent>
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
</dependencies>
</project>
```

IntelliJ-Project



```
▼ JavaSpring [org.javacream.spring.rest] C:\Users\Sound and
    ▶ .idea
    ▼ src
        ▼ main
            ▶ java
              ressources
        ▼ test
              java
              ressources
    ▶ target
      org.javacream.spring.rest.iml
      pom.xml
▶ External Libraries
```

Dependencies



```
▼ External Libraries
  ▶ < 1.8 > C:\_training_general\tools\jdk8
  ▶ Maven: aopalliance:aopalliance:1.0
  ▶ Maven: ch.qos.logback:logback-classic:1.1.3
  ▶ Maven: ch.qos.logback:logback-core:1.1.3
  ▶ Maven: com.fasterxml.jackson.core:jackson-annotations:2.4.6
  ▶ Maven: com.fasterxml.jackson.core:jackson-core:2.4.6
  ▶ Maven: com.fasterxml.jackson.core:jackson-databind:2.4.6
  ▶ Maven: com.fasterxml:classmate:1.0.0
  ▶ Maven: javax.validation:validation-api:1.1.0.Final
  ▶ Maven: org.apache.tomcat.embed:tomcat-embed-core:8.0.23
  ▶ Maven: org.apache.tomcat.embed:tomcat-embed-el:8.0.23
  ▶ Maven: org.apache.tomcat.embed:tomcat-embed-logging-juli:8.0.2
  ▶ Maven: org.apache.tomcat.embed:tomcat-embed-websocket:8.0.23
  ▶ Maven: org.hibernate:hibernate-validator:5.1.3.Final
  ▶ Maven: org.jboss.logging:jboss-logging:3.1.3.GA
  ▶ Maven: org.slf4j:jcl-over-slf4j:1.7.12
  ▶ Maven: org.slf4j:jul-to-slf4j:1.7.12
  ▶ Maven: org.slf4j:log4j-over-slf4j:1.7.12
  ▶ Maven: org.slf4j:slf4j-api:1.7.12
  ▶ Maven: org.springframework.boot:spring-boot:1.2.5.RELEASE
  ▶ Maven: org.springframework.boot:spring-boot-autoconfigure:1.2.5.
  ▶ Maven: org.springframework.boot:spring-boot-starter:1.2.5.RELEASE
  ▶ Maven: org.springframework.boot:spring-boot-starter-logging:1.2.5.
  ▶ Maven: org.springframework.boot:spring-boot-starter-tomcat:1.2.5.
  ▶ Maven: org.springframework.boot:spring-boot-starter-web:1.2.5.REL
  ▶ Maven: org.springframework:spring-aop:4.1.7.RELEASE
  ▶ Maven: org.springframework:spring-beans:4.1.7.RELEASE
  ▶ Maven: org.springframework:spring-context:4.1.7.RELEASE
  ▶ Maven: org.springframework:spring-core:4.1.7.RELEASE
  ▶ Maven: org.springframework:spring-expression:4.1.7.RELEASE
```

1.2

A FIRST EXAMPLE

A Spring Boot Rest Application

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class RestApplication {

    @RequestMapping("/hello")
    public String doHello(){
        return "Hello!";
    }
    @RequestMapping("/exit")
    public void doExit(){
        System.exit(0);
    }

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

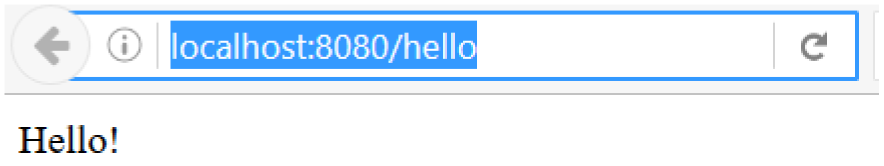
Running the application

Simply start the main method

It's really that easy!

Open a browser and enter

http://localhost:8080/hello


localhost:8080/hello

```
Hello!
```

2

CONTEXT AND DEPENDENCY INJECTION

2.1

OVERVIEW

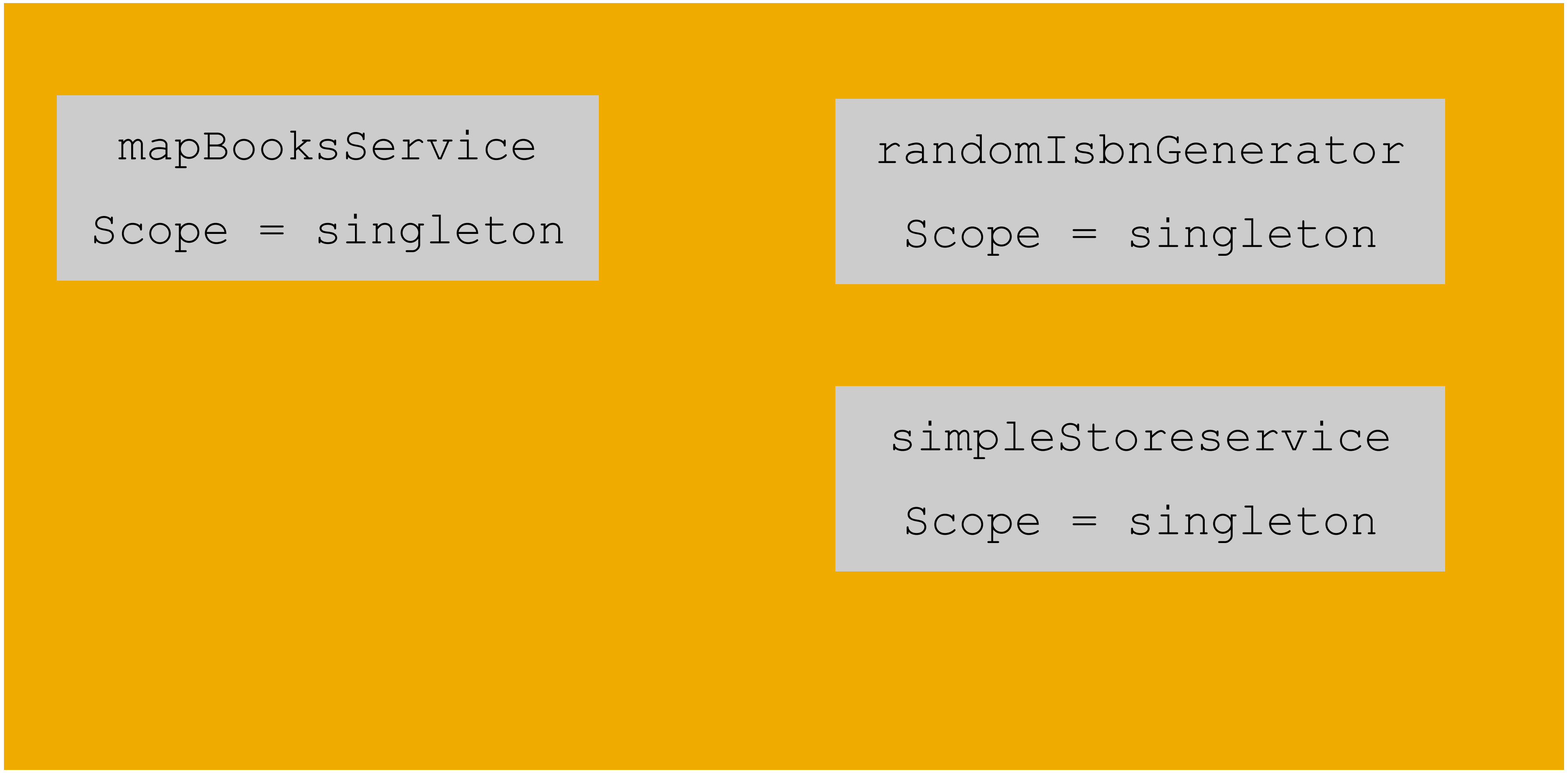
Context

A Context is responsible to create business objects depending on the declared "scope"

- singleton/application
- prototype/request
- session
- …

Basically a context is a (very) smart Map

A "Live" Context

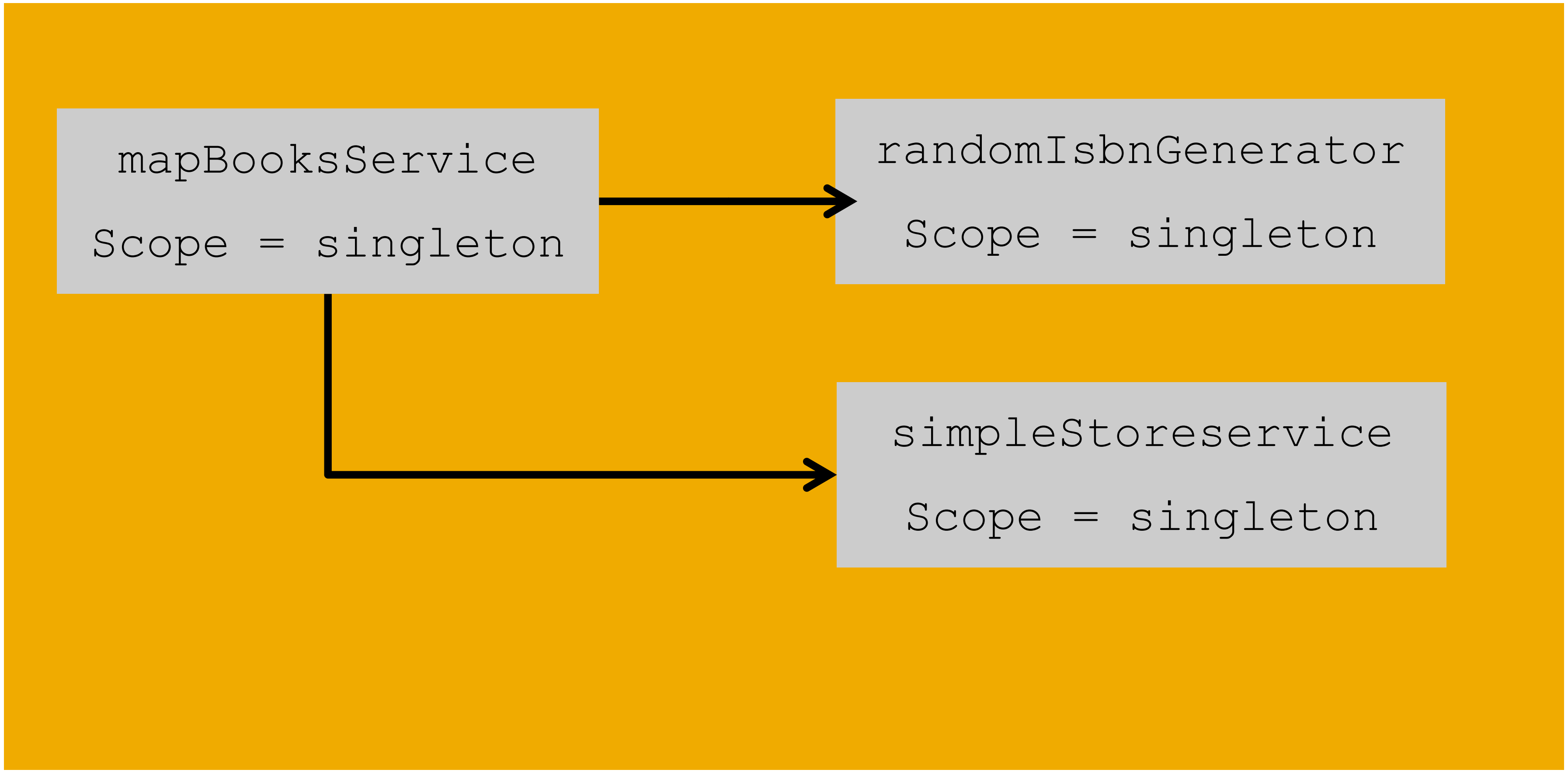


Dependency Injection

Every business class declares dependencies
this ist a simple OOP association

A Context is responsible to
identify and
inject or set these dependencies

A "Live" Context after dependency injection

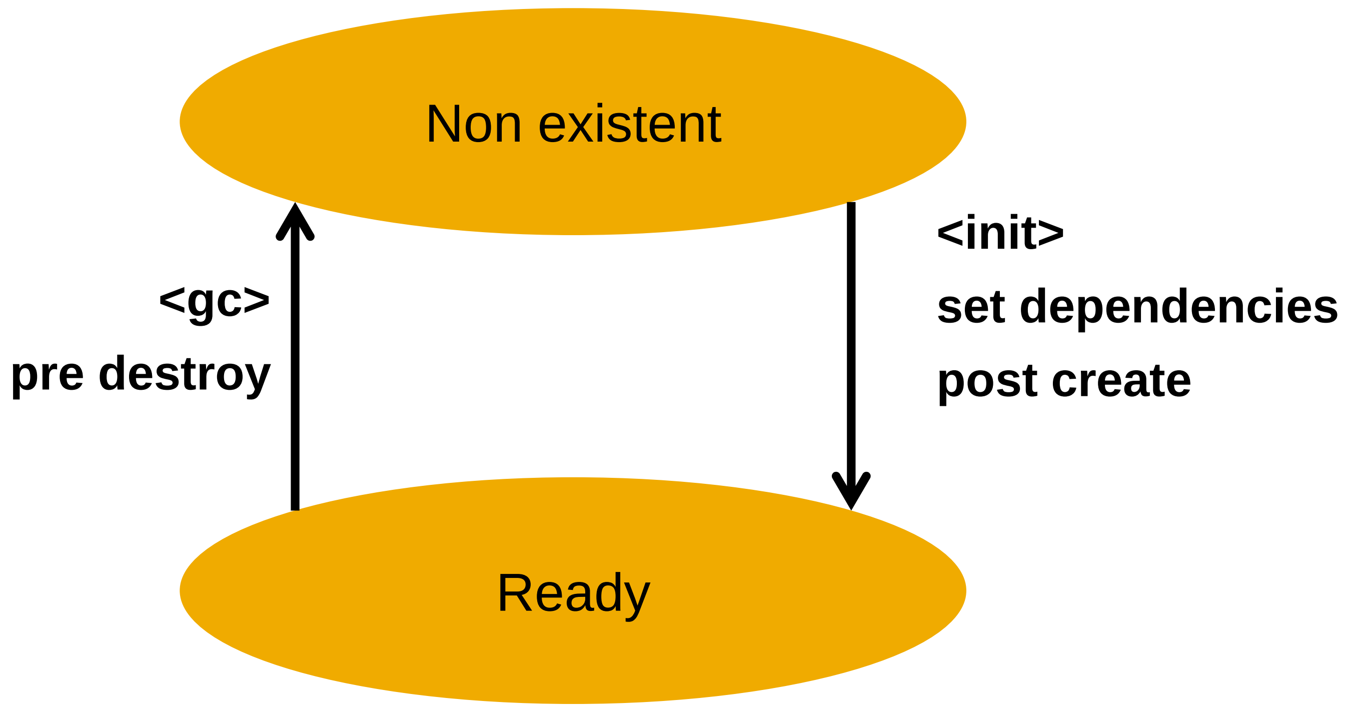


Lifecycle

Every business object has a defined lifecycle

- instantiation
- dependency injection
- post create
- pre destroy
- destroy (Garbage collection)

State diagram



2.2

SPRING CORE

The Spring Context

Scope and lifecycle are defined using

XML

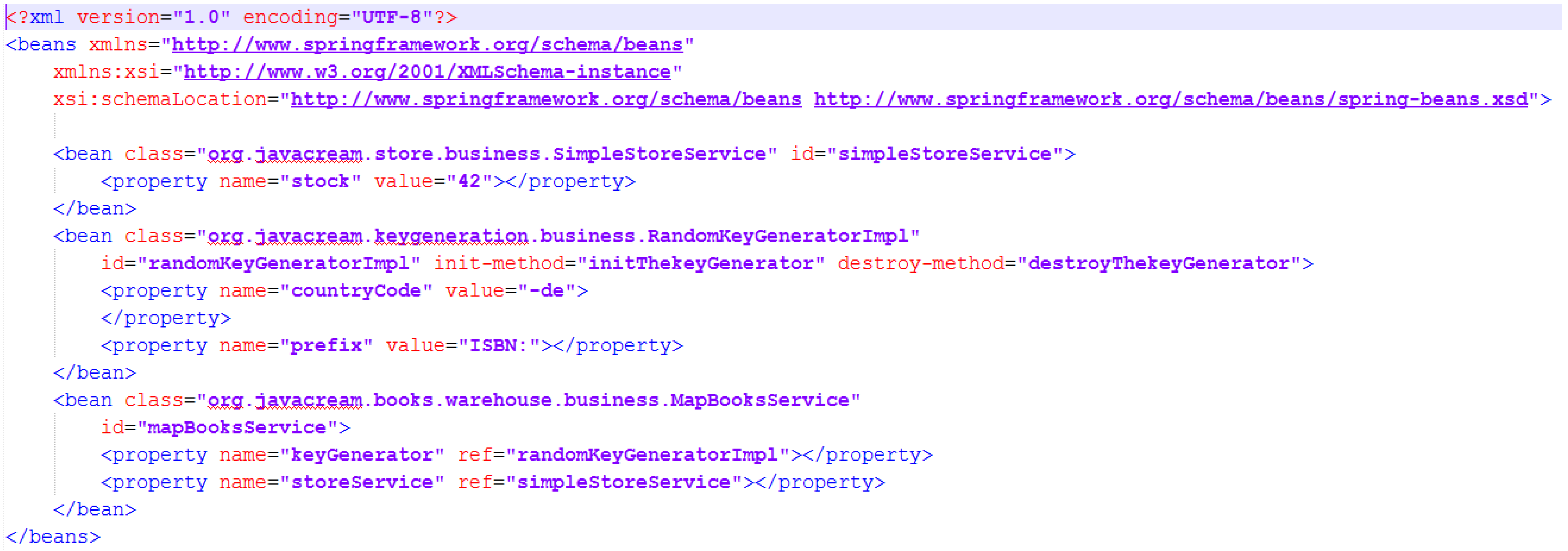
Java Annotations

JavaConfig

- a kind of factory

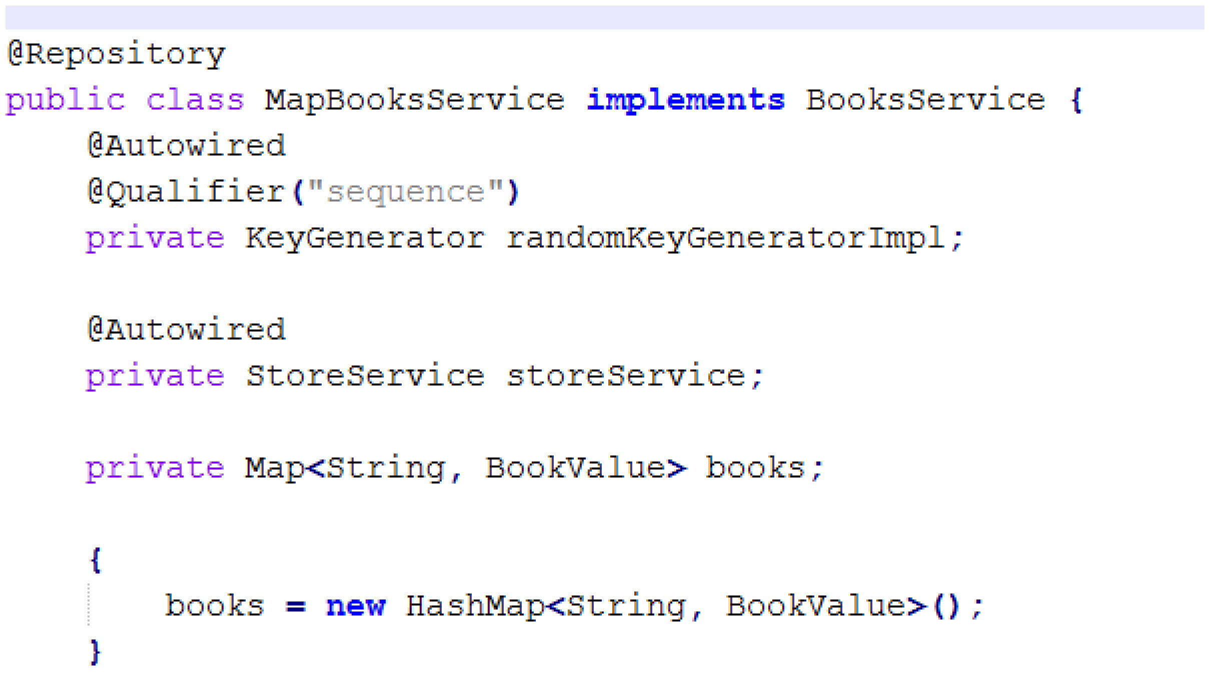
a mixture of all above

Spring XML



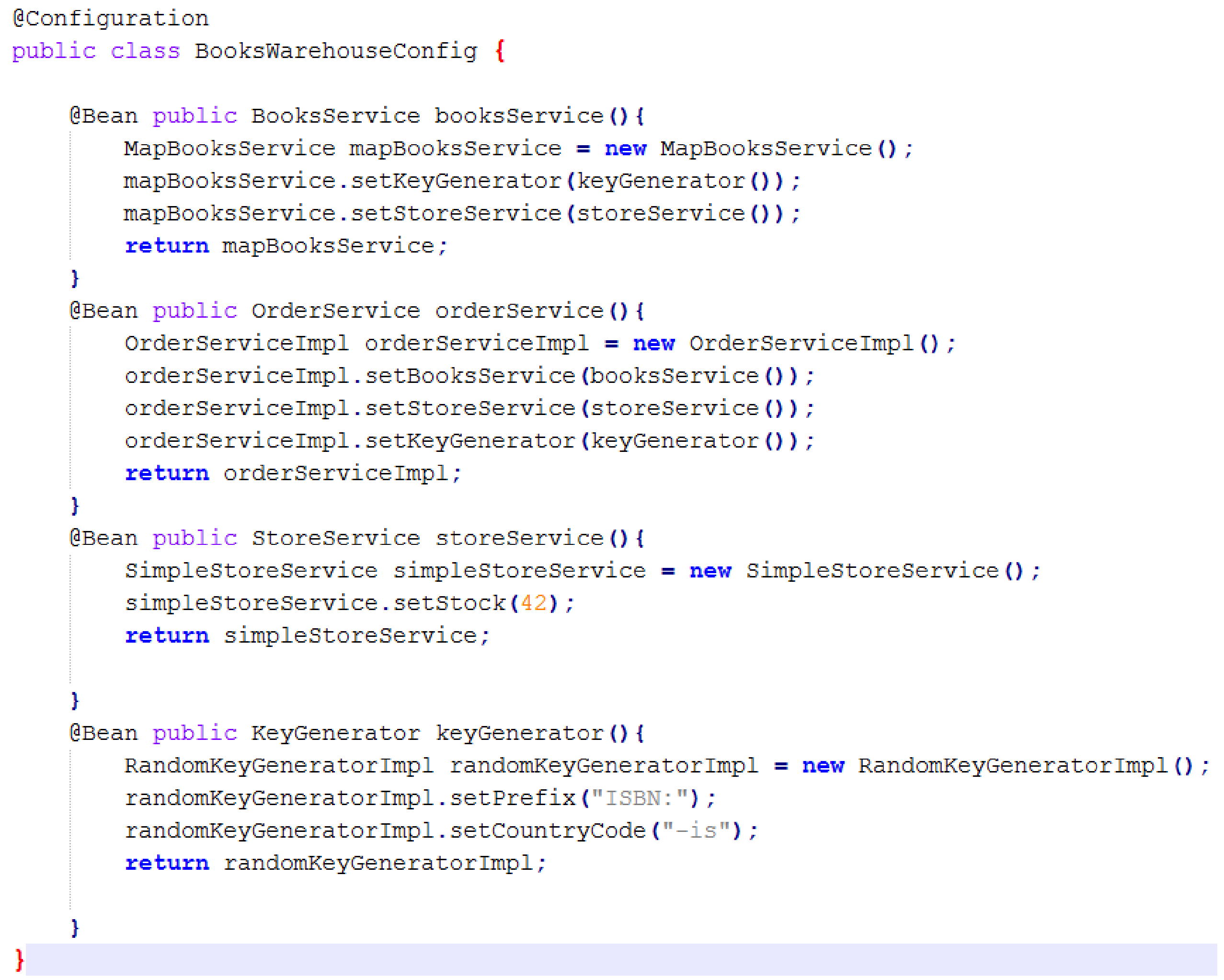
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean class="org.javacream.store.business.SimpleStoreService" id="simpleStoreService">
        <property name="stock" value="42"></property>
    </bean>
    <bean class="org.javacream.keygeneration.business.RandomKeyGeneratorImpl"
        id="randomKeyGeneratorImpl" init-method="initThekeyGenerator" destroy-method="destroyThekeyGenerator">
        <property name="countryCode" value="-de">
        </property>
        <property name="prefix" value="ISBN:"></property>
    </bean>
    <bean class="org.javacream.books.warehouse.business.MapBooksService"
        id="mapBooksService">
        <property name="keyGenerator" ref="randomKeyGeneratorImpl"></property>
        <property name="storeService" ref="simpleStoreService"></property>
    </bean>
</beans>
```

Spring Annotations



```
@Repository
public class MapBooksService implements BooksService {
    @Autowired
    @Qualifier("sequence")
    private KeyGenerator randomKeyGeneratorImpl;

    @Autowired
    private StoreService storeService;

    private Map<String, BookValue> books;

    {
        books = new HashMap<String, BookValue>();
    }
```

Spring JavaConfig

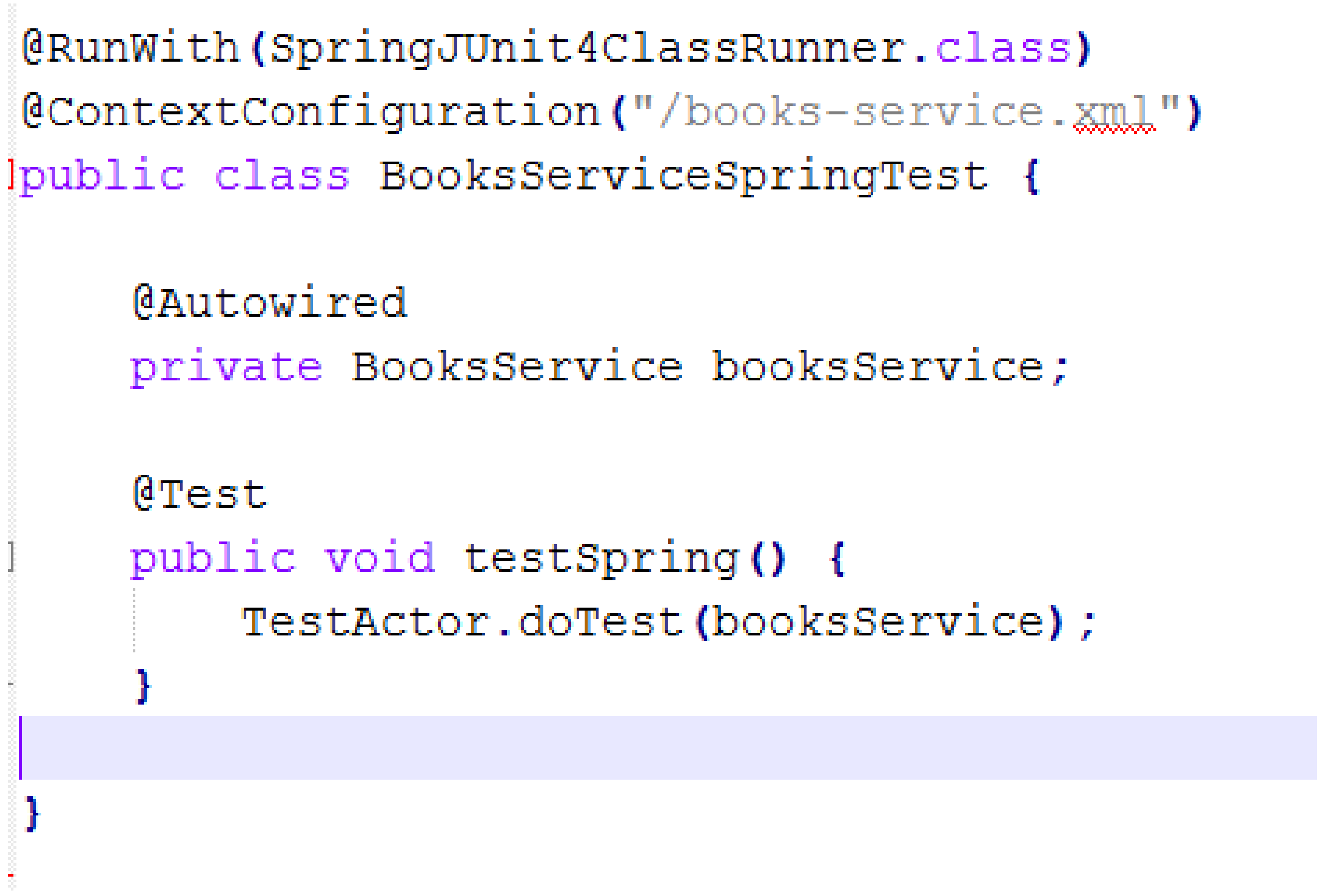


```java
@Configuration
public class BooksWarehouseConfig {

    @Bean public BooksService booksService(){
        MapBooksService mapBooksService = new MapBooksService();
        mapBooksService.setKeyGenerator(keyGenerator());
        mapBooksService.setStoreService(storeService());
        return mapBooksService;
    }
    @Bean public OrderService orderService(){
        OrderServiceImpl orderServiceImpl = new OrderServiceImpl();
        orderServiceImpl.setBooksService(booksService());
        orderServiceImpl.setStoreService(storeService());
        orderServiceImpl.setKeyGenerator(keyGenerator());
        return orderServiceImpl;
    }
    @Bean public StoreService storeService(){
        SimpleStoreService simpleStoreService = new SimpleStoreService();
        simpleStoreService.setStock(42);
        return simpleStoreService;

    }
    @Bean public KeyGenerator keyGenerator(){
        RandomKeyGeneratorImpl randomKeyGeneratorImpl = new RandomKeyGeneratorImpl();
        randomKeyGeneratorImpl.setPrefix("ISBN:");
        randomKeyGeneratorImpl.setCountryCode("-is");
        return randomKeyGeneratorImpl;

    }
}
```

2.3

RUNNING SPRING APPLICATIONS

Unit Tests



```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("/books-service.xml")
public class BooksServiceSpringTest {

    @Autowired
    private BooksService booksService;

    @Test
    public void testSpring() {
        TestActor.doTest(booksService);
    }

}
```

Simple main

```
public static void main(String[] args){

ClasspathXmlApplicationContext context = new
ClasspathXmlApplicationContext("/books-service.xml");
BooksService bs context.getBean(BooksService.class);
}
```

Spring Boot

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

2.4

MORE FEATURES

Spring AOP

Aspect Oriented Programmming introduces "Cross Cutting Concerns"
e.g. "every business method call must be audited"

More examples:
Authentication
Transaction Management
Profiling during tests
Tracing

Spring provides us with a sophisticated AOP framework
XML configuration (`aop`-namespace)
Annotations (`@AspectJ` and `@Around`)

Configuration

Configuring Spring Applications is easy

Reading properties files and system properties

Use Spring Expression Language to access properties

in most cases a simple expressions are sufficient

- `${propertyKey}`

advanced expressions

- `${(bean.list[i5] + bean2.foo.goo) > 42}`

Property values will be injected

`value`-Attribut using XML

`@Value`

Utilities

Spring distribution contains a lot of utilities

JdbcTemplate for database access

ServiceExporters and proxies to run client server applications

A full blown web framework, Spring MVC

JMX support using annotations

And even more!

Beside Spring core we have a complete ecosystem of established Spring projects

Spring Data

Spring Security

Spring Batch

Spring Integration

Spring Web Services

…

3

RESTFUL WEBSERVICES

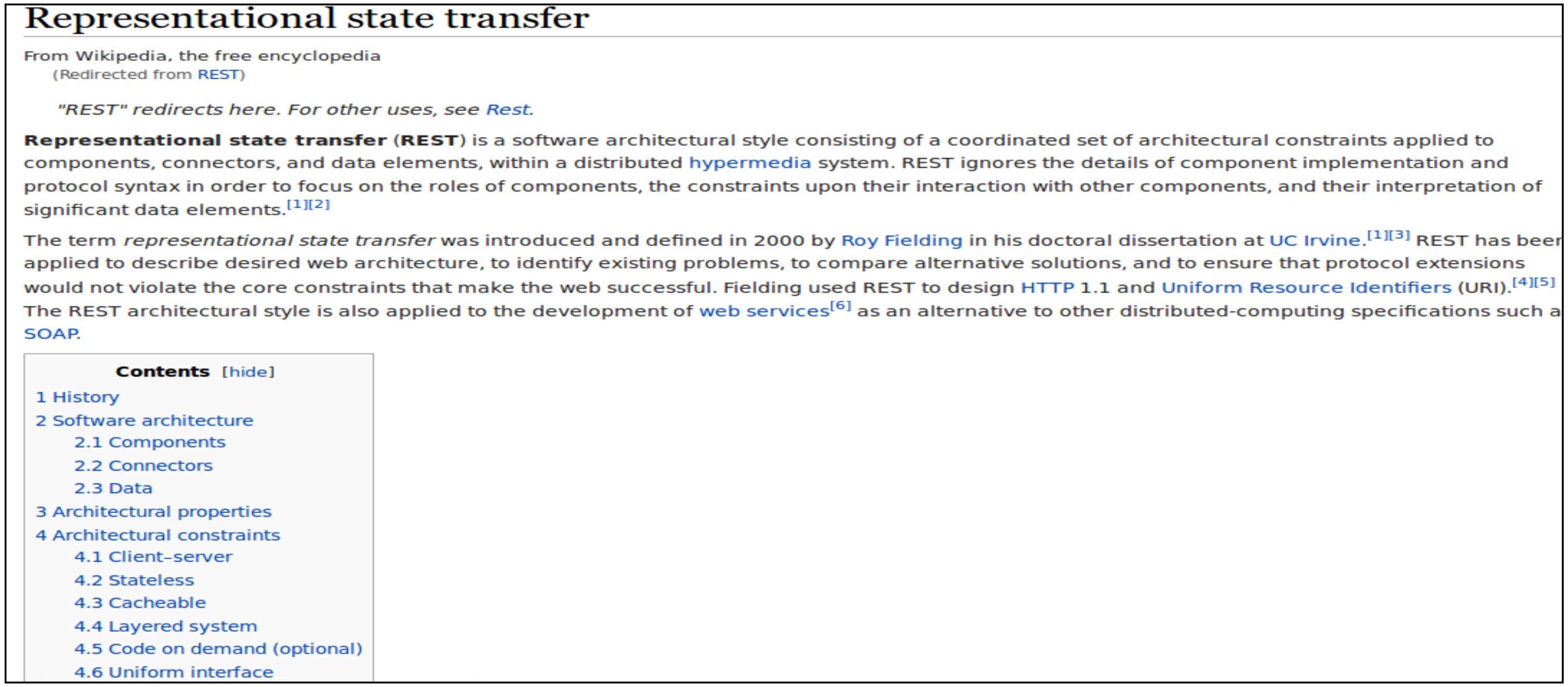
3.1

OVERVIEW

Representional State Transfer

Doctoral dissertation of Roy Fielding, 2000

http://en.wikipedia.org/wiki/REST



Representational state transfer

From Wikipedia, the free encyclopedia
(Redirected from REST)

"REST" redirects here. For other uses, see Rest.

Representational state transfer (**REST**) is a software architectural style consisting of a coordinated set of architectural constraints applied to components, connectors, and data elements, within a distributed hypermedia system. REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements.[1][2]

The term *representational state transfer* was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation at UC Irvine.[1][3] REST has beer applied to describe desired web architecture, to identify existing problems, to compare alternative solutions, and to ensure that protocol extensions would not violate the core constraints that make the web successful. Fielding used REST to design HTTP 1.1 and Uniform Resource Identifiers (URI).[4][5] The REST architectural style is also applied to the development of web services[6] as an alternative to other distributed-computing specifications such a SOAP.

Contents [hide]

Basic Concepts

Starting point: How is it possible to abstract the world wide web?

REST architecture:

What are resources?

Resource identifiers?

Resource operations?

What are "stateless" operations?

Implementing caches

Resources are linked together

Not really surprising: „WWW follows the REST style"

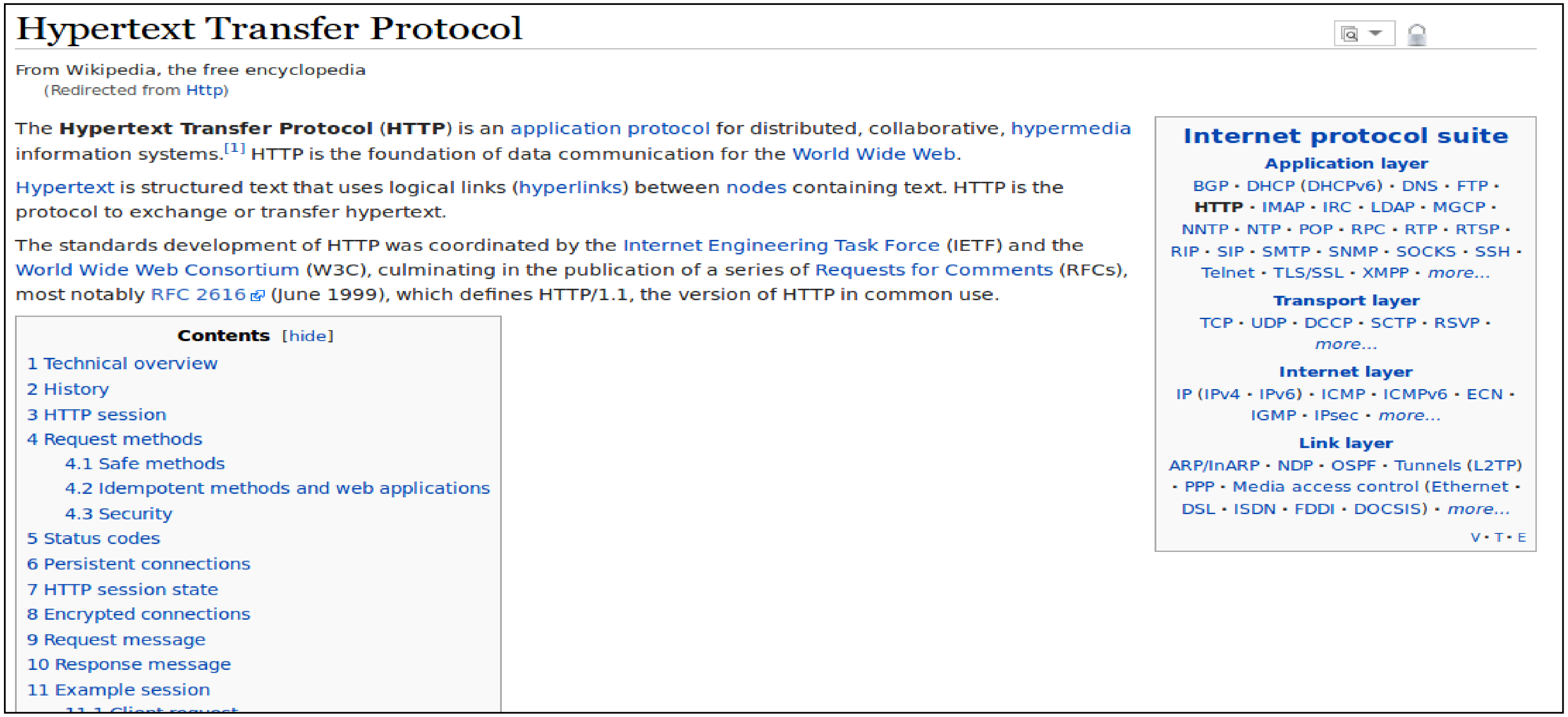
3.2

REST AND HTTP

http-protocol

A w3w specification

http://en.wikipedia.org/wiki/Http



Hypertext Transfer Protocol

From Wikipedia, the free encyclopedia
(Redirected from Http)

The **Hypertext Transfer Protocol** (**HTTP**) is an application protocol for distributed, collaborative, hypermedia information systems.[1] HTTP is the foundation of data communication for the World Wide Web.

Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext.

The standards development of HTTP was coordinated by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C), culminating in the publication of a series of Requests for Comments (RFCs), most notably RFC 2616 (June 1999), which defines HTTP/1.1, the version of HTTP in common use.

Contents [hide]

Internet protocol suite

Application layer

BGP · DHCP (DHCPv6) · DNS · FTP · **HTTP** · IMAP · IRC · LDAP · MGCP · NNTP · NTP · POP · RPC · RTP · RTSP · RIP · SIP · SMTP · SNMP · SOCKS · SSH · Telnet · TLS/SSL · XMPP · *more...*

Transport layer

TCP · UDP · DCCP · SCTP · RSVP · *more...*

Internet layer

IP (IPv4 · IPv6) · ICMP · ICMPv6 · ECN · IGMP · IPsec · *more...*

Link layer

ARP/InARP · NDP · OSPF · Tunnels (L2TP) · PPP · Media access control (Ethernet · DSL · ISDN · FDDI · DOCSIS) · *more...*

V · T · E

http specification

URIs

Path

Parameters

http-Request and http-Response

Data-Container with header and body

Encoding

Predefinded header properties

Content-Length

Accepts

Content-Type

http specification II

http-methods

PUT

GET

POST

DELETE

OPTIONS

HEAD

Status codes

404: „Not found“

204: „Created“

…

MIME Types

Data types of WWW

Not a XML Schema!

A MIME type is a structured string

- text/html
- application/xml
- text/html; charset=UTF-8

custom mime types are possible

REST and http

REST and http are NOT the same!

REST is an archtitectural style

http is a network protocole

But

http is a natural candidate to implement a RESTful architecture

Mapping REST - http

PUT

- create a resource
- may be used to update a resource

GET

- Read resources

POST

- update a resource
- may be used to create a resource

DELETE

- delete a resource

Creating Resources

Using PUT

Client provides a unique resourcen id

Status code „201: Created“

Using POST

Server decides to create or update

if create:

- Status code „201: Created“
- `Location`-Header contains the generated resource id

Update

Using PUT

Statuscode „200: OK“ or „204: No content

PUT is idempotent

Using POST

POST is NOT idempotent

Delete

Using DELETE

Statuscode „200: OK“ or „204: No content

PUT is idempotent

Caution:

DELETE deletes the resource

Maybe an UPDATE is what you really need

- Cancelling an order is a kind of "delete", but the order must not be deleted
- "overloading" DELETE
  - `DELETE order/ISBN42?cancel=true`

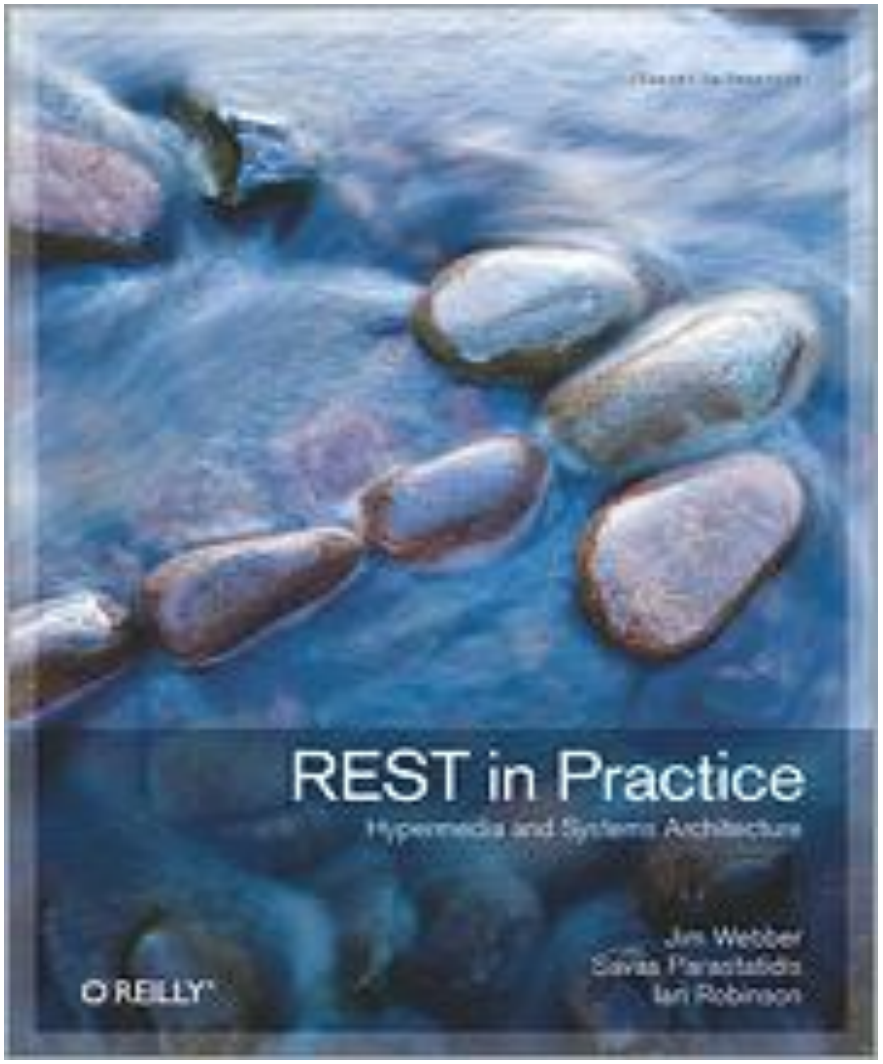
4

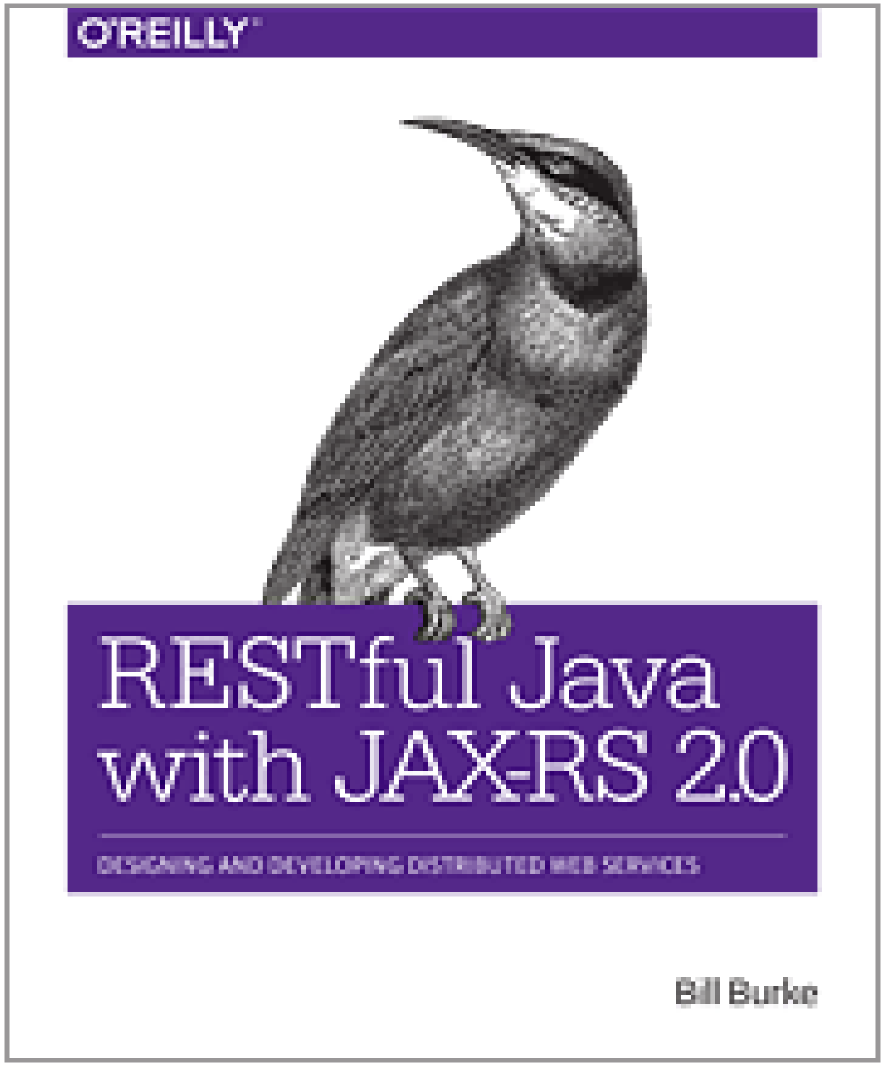
JAX-RS

4.1

OVERVIEW

Resources


REST in Practice
Hypermedia and Systems Architecture
Jim Webber
Savas Parastatidis
Ian Robinson
O'REILLY

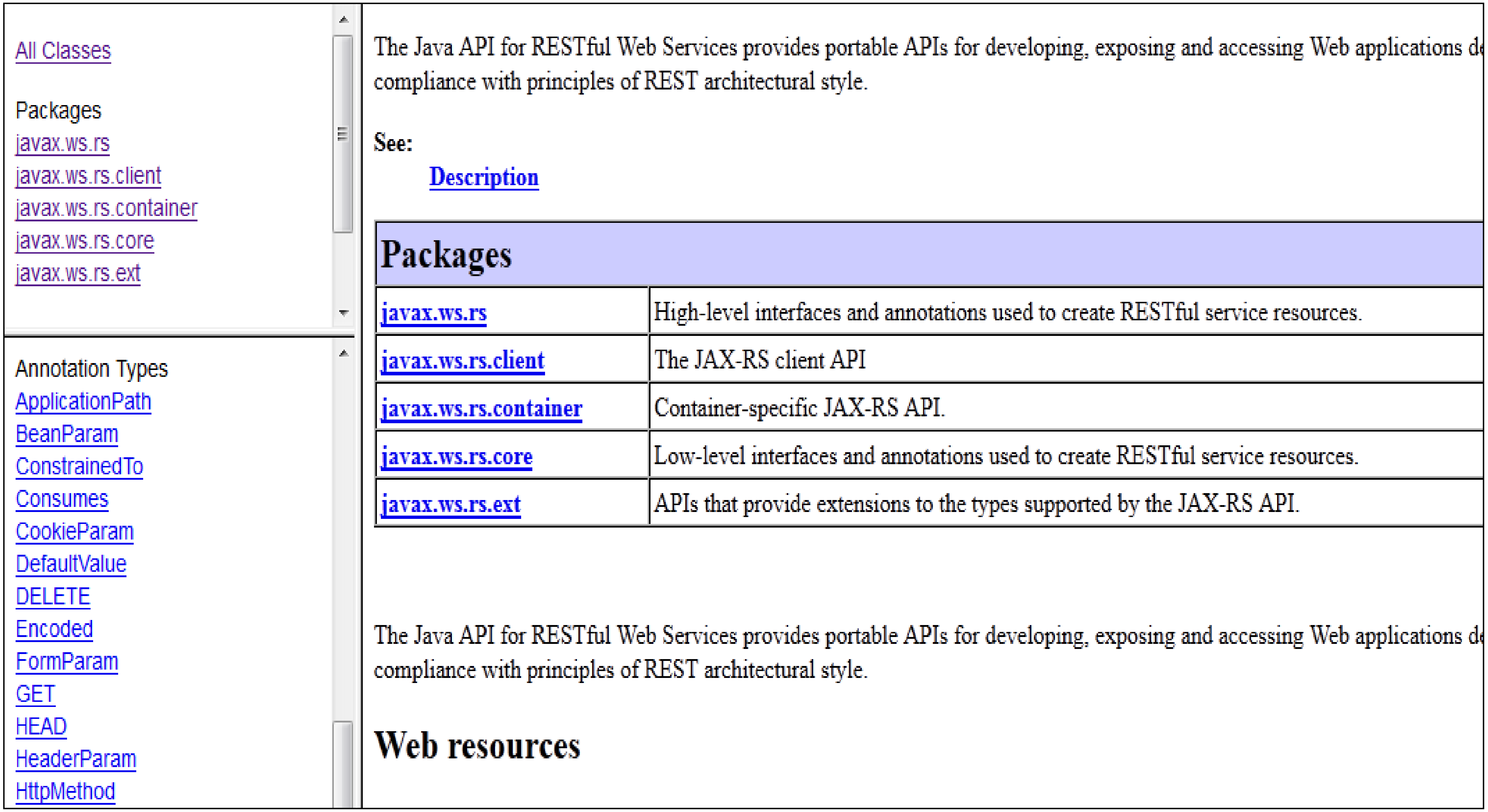

O'REILLY
RESTful Java with JAX-RS 2.0
DESIGNING AND DEVELOPING DISTRIBUTED WEB SERVICES
Bill Burke

JAX-RS

Part of the Java Enterprise Edition

- a specification maintained by Oracle
- Application servers implement JEE
- every application server must provide a JAX-RS runtime

JAX-RS-API



All Classes

Packages
javax.ws.rs
javax.ws.rs.client
javax.ws.rs.container
javax.ws.rs.core
javax.ws.rs.ext

Annotation Types
ApplicationPath
BeanParam
ConstrainedTo
Consumes
CookieParam
DefaultValue
DELETE
Encoded
FormParam
GET
HEAD
HeaderParam
HttpMethod

The Java API for RESTful Web Services provides portable APIs for developing, exposing and accessing Web applications d
compliance with principles of REST architectural style.

See:
Description

Packages	
javax.ws.rs	High-level interfaces and annotations used to create RESTful service resources.
javax.ws.rs.client	The JAX-RS client API
javax.ws.rs.container	Container-specific JAX-RS API.
javax.ws.rs.core	Low-level interfaces and annotations used to create RESTful service resources.
javax.ws.rs.ext	APIs that provide extensions to the types supported by the JAX-RS API.

The Java API for RESTful Web Services provides portable APIs for developing, exposing and accessing Web applications d
compliance with principles of REST architectural style.

Web resources

4.2

COMPONENTS

Component Annotations

`@Provider`

`@Path`

- Path is a special Provider

JAX-RS provides a CDI-Framework

Lifecycle

Dependency Injection

Spring Integration

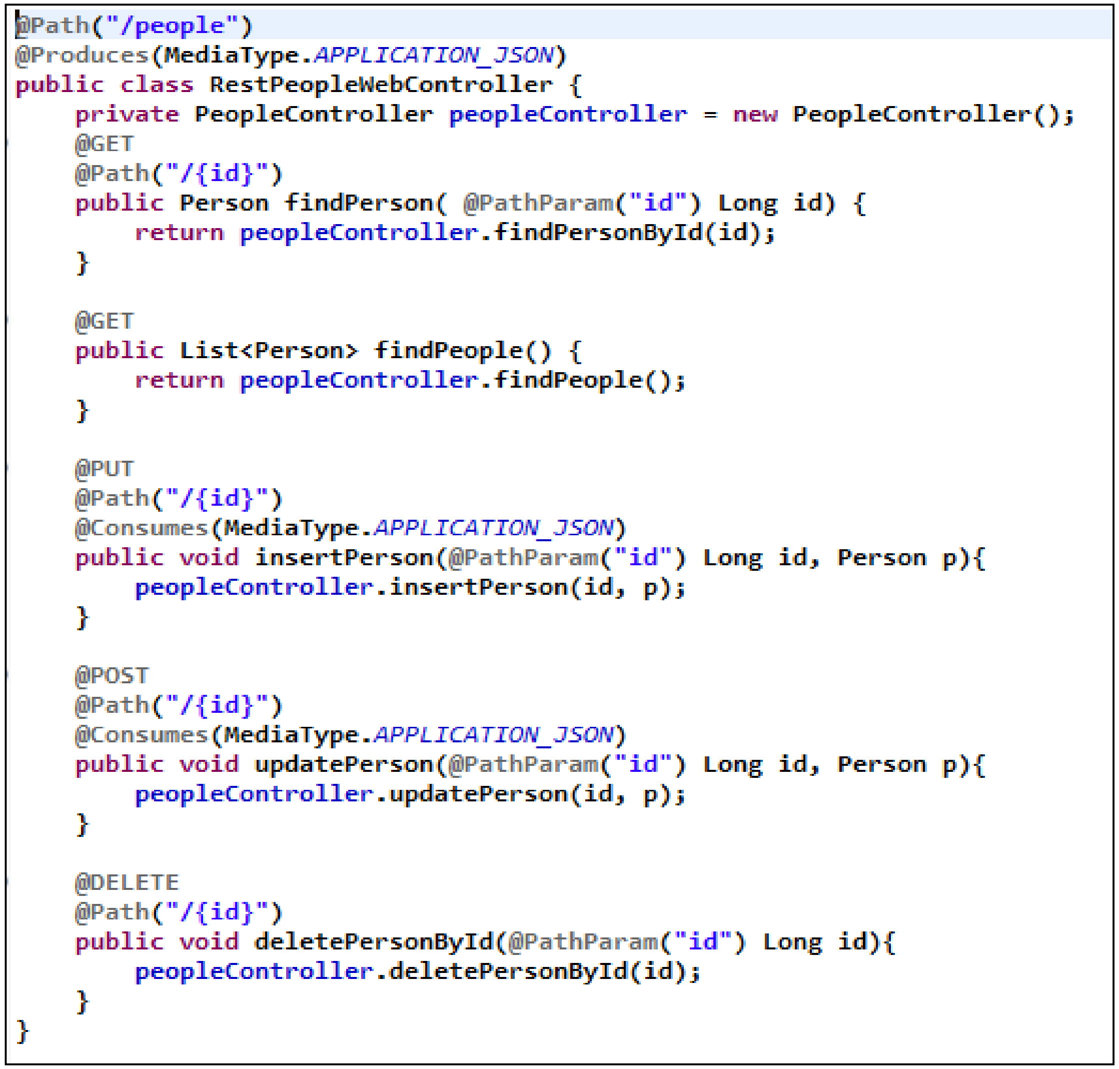
- Just add a Spring Stereotype Annotation like @Service
- or use CDI-Framework

Dependency Injection

Method-parameters use `@Context`-Annotation to access

- `Request`
- `HttpHeader`
- `UriInfo`
- `Configurable`

Example



```
@Path("/people")
@Produces(MediaType.APPLICATION_JSON)
public class RestPeopleWebController {
    private PeopleController peopleController = new PeopleController();
    @GET
    @Path("/{id}")
    public Person findPerson( @PathParam("id") Long id) {
        return peopleController.findPersonById(id);
    }

    @GET
    public List<Person> findPeople() {
        return peopleController.findPeople();
    }

    @PUT
    @Path("/{id}")
    @Consumes(MediaType.APPLICATION_JSON)
    public void insertPerson(@PathParam("id") Long id, Person p){
        peopleController.insertPerson(id, p);
    }

    @POST
    @Path("/{id}")
    @Consumes(MediaType.APPLICATION_JSON)
    public void updatePerson(@PathParam("id") Long id, Person p){
        peopleController.updatePerson(id, p);
    }

    @DELETE
    @Path("/{id}")
    public void deletePersonById(@PathParam("id") Long id){
        peopleController.deletePersonById(id);
    }
}
```

4.3

DEPLOYMENT

Web Archive

RESTful Web Services will be deployed as a Web Archive, .war

Contents:

- `WEB-INF`
- `WEB-INF/classes`
- `WEB-INF/lib`

Descriptor `web.xml`

JAX-RS runtime is a JEE Servlet

- reuse existing ServletFilters

Application

```
@ApplicationPath("/rest")
public class RestApplication extends Application{
 private HashSet<Class<?>> classes;
 {
      classes = new HashSet<>();
      classes.add(RestBooksServiceFacade.class);
 }
 @Override
 public Set<Class<?>> getClasses() {
      return classes;
 }
}
```

Annotation Scanning

Use an "empty" `Application`

```
@ApplicationPath("/rs")
public class AnnotationScannerApplication extends Application {
public Set<Class<?>> getClasses() {
return Collections.emptySet();
}
@Override
public Set<Object> getSingletons() {
return Collections.emptySet();
}
}
```

4.4

REQUEST

URI-Mapping

http-Requests is mapped to a Java signature using the `@Path`-annotationen

Mappings may overlap, but must be unambigious

the more specific mapping wins

- `@Path("books/isbn-{isbn}")`
- `@Path("books/{isbn})`

- `@Path("books/{isbn})`
- `@Path("books/{title})`

Passing parameters

`@PathParam`

`@QueryParam`

`@FormParam`

`@CookieParam`

`@HeaderParam`

`@MatrixParam`

`@BeanParam`

- Class name with annotated fields

`@DefaultValue`

- sets a default value

Example

```
public String javaService(@QueryParam("order") String order,
@PathParam Long id)
```

4.5

CONTENT HANDLER

Content Handler

A Content Handler encodes the Java result object to a MIME type representation

- Low-Level Content Handler must be part of a JAXRS runtime

Content Type is defined using the `@Produces`-annotation

`javax.ws.rs.core.MediaType` contains some constants

- `MediaType.APPLICATION_JSON`
- `MediaType.TEXT_PLAIN`

Supported Standard Java Types

`java.io.InputStream` and `java.io.Reader`

- read request bodies

`java.io.File`

- returned value is the file content
- Content Type must match!

`byte[]`, `char[]` and `String`

`javax.xml.transform.Source`

- reading/writing XML documents

JAX-RS-Types

`javax.ws.rs.core.StreamingOutput`

- Streaming API
- Wraps `java.io.OutputStream`

```
public void write(OutputStream output) throws IOException,
WebApplicationException
```

- return type of a Component method

`javax.ws.rs.core.MultivaluedMap`

- Form input

JAXB

Java-to-XML Mapping

- using a lot of annotations

every JAXRS provider must provide a matching Content Handler

MIME type

- `@Produces(MediaType.APPLICATION_XML)`

JSON

JSON-API defined in package `javax.json`

`JsonObject`

`JsonArray`

`JsonObjectBuilder und JsonArrayBuilder`

`JsonParser`

Libraries like "Jackson" provide ready-to-use Content Handlers

Custom Content Handlers

A custom content handler implements

```
javax.ws.rs.ext.MessageBodyWriter
javax.ws.rs.ext.MessageBodyReader
```

Annotations:

```
@Provider
@Produces("application/xml")
public class MyJAXBMarshaller implements MessageBodyWriter{…}
```

or

```
@Provider
@Consumes("application/xml")
public class MyJAXBUnmarshaller implements MessageBodyReader{…}
```

4.6

CONTENT NEGOTIATION

`@Produces`

Sets the Content-Type of response

used to resolve the corresponding Java method

- `Accept-Header`
- works like `@Path`

4.7

RESPONSE AND EXCEPTIONS

Standard Response Codes

w3w maintaines a mapping between REST and http

http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

Status Codes and Exceptions

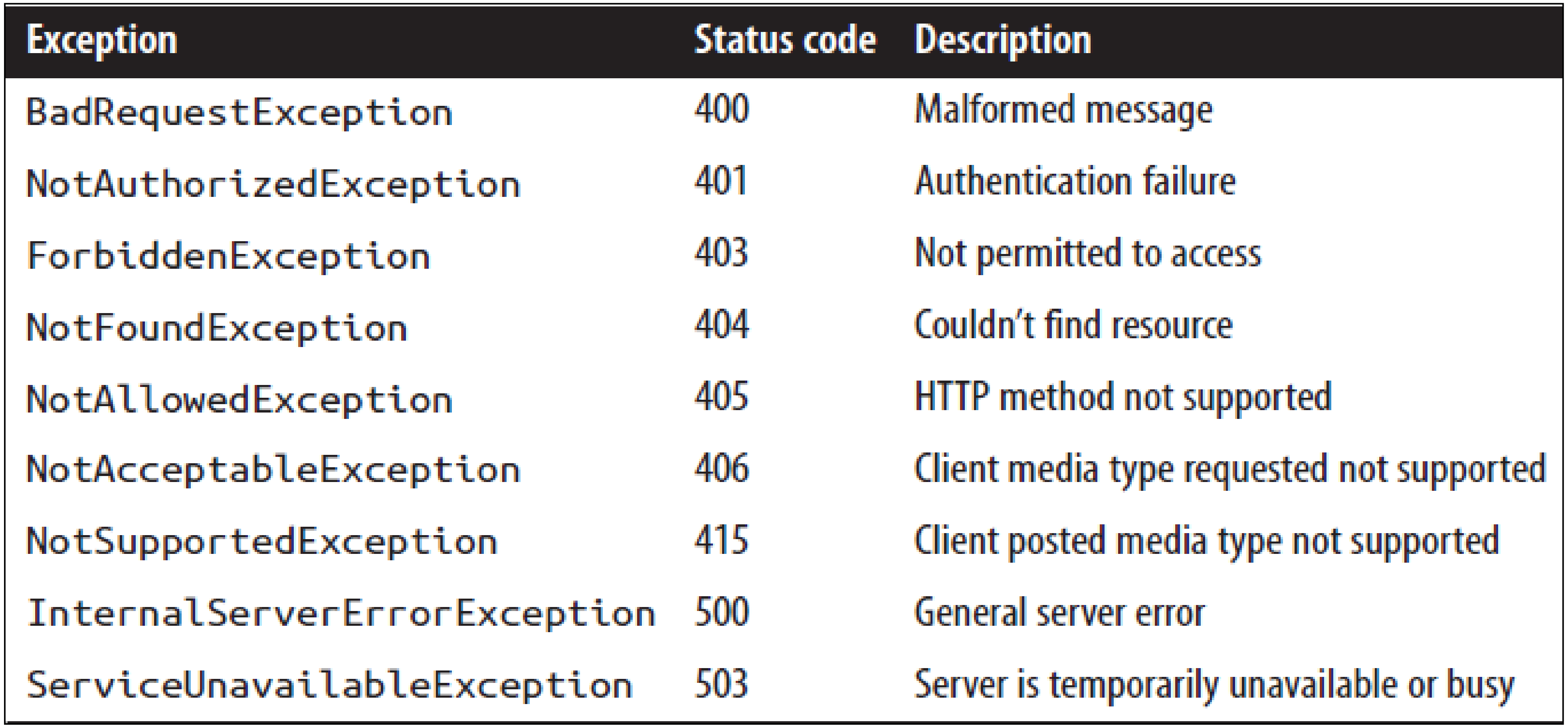
`javax.ws.rs.WebApplicationException`

a special RuntimeException

Caution

- Developers must create a `WebApplicationException` with valid status codes!
  - PUT must not return „404: Not Found“

Exceptions



Exception	Status code	Description
`BadRequestException`	400	Malformed message
`NotAuthorizedException`	401	Authentication failure
`ForbiddenException`	403	Not permitted to access
`NotFoundException`	404	Couldn't find resource
`NotAllowedException`	405	HTTP method not supported
`NotAcceptableException`	406	Client media type requested not supported
`NotSupportedException`	415	Client posted media type not supported
`InternalServerErrorException`	500	General server error
`ServiceUnavailableException`	503	Server is temporarily unavailable or busy

Response and ResponseBuilder

```
public Response getBook() {
String book = ...;
ResponseBuilder builder = Response.ok(book);
builder.language("fr")
.header("Some-Header", "some value");
return builder.build();
}
```

`javax.ws.rs.core.Response` holds

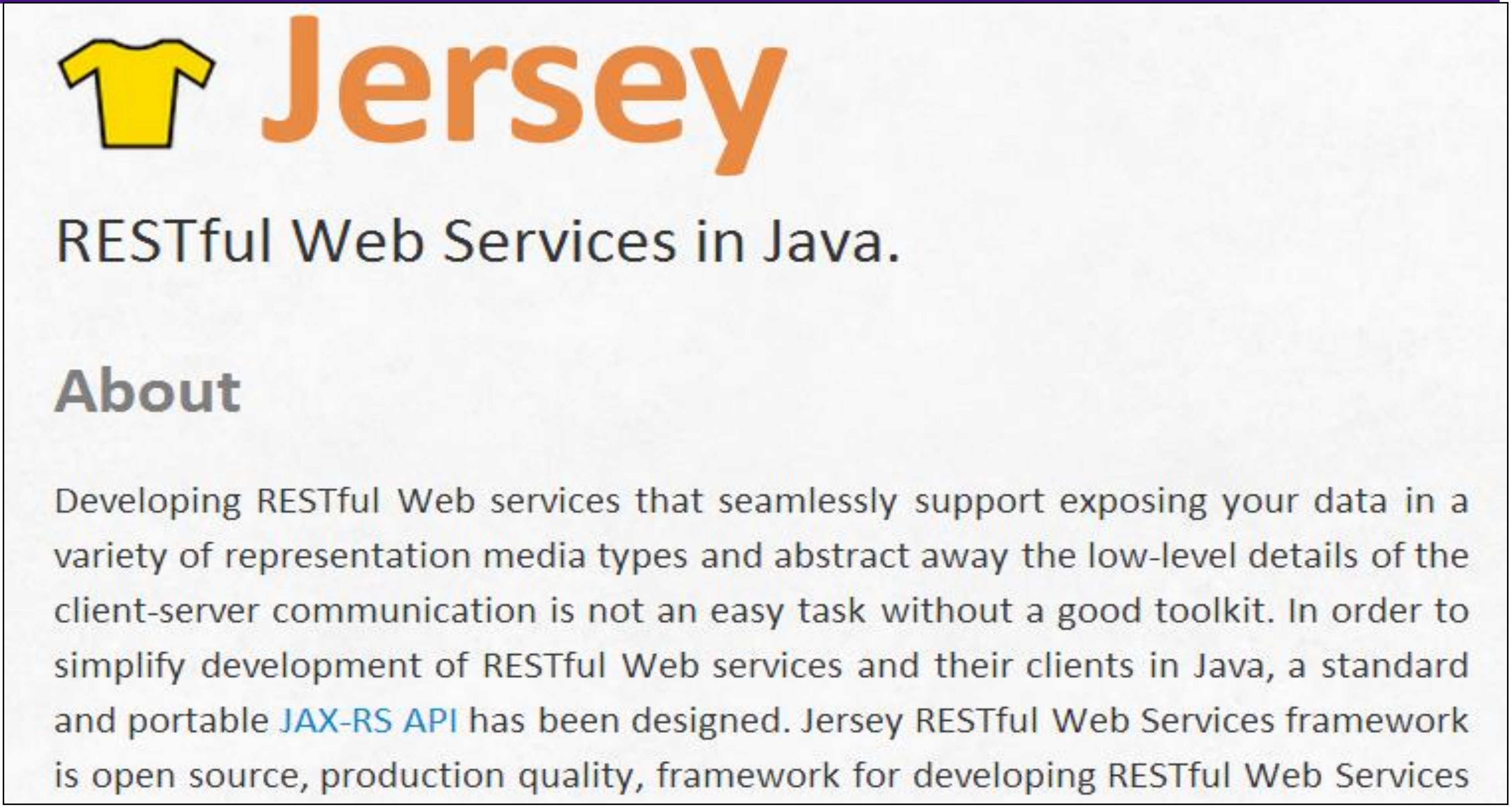
the converted Java object

- An „Entity“
- Status code
- Header as `MultivaluedMap`

4.8

PROVIDER

Jersey


Jersey

RESTful Web Services in Java.

About

Developing RESTful Web services that seamlessly support exposing your data in a variety of representation media types and abstract away the low-level details of the client-server communication is not an easy task without a good toolkit. In order to simplify development of RESTful Web services and their clients in Java, a standard and portable JAX-RS API has been designed. Jersey RESTful Web Services framework is open source, production quality, framework for developing RESTful Web Services

RESTEasy

RESTEasy

JBoss/RedHat

Apache CXF

Every JEE ApplicationServer

Glassfish

Wildfly

…

5

SPRING REST

5.1

OVERVIEW

Resources

THE EXPERT'S VOICE® IN SPRING
Spring REST
REST AND WEB SERVICES DEVELOPMENT USING SPRING
Balaji Varanasi and Sudha Belida
Apress®
www.allitebooks.com

Basic Concepts

Spring REST uses Spring MVC

A Model-View-Controller framework

Spring MVC introduces

- A central dispatcher servlet serving all http requests
- Mapping annotations to let the dispatcher servlet know which methods to invoke
- A rendering engine using plugins, e.g. for JSP, XML or JSON

From a developers view REST and HTML Web Applications look the same

Spring Rest and JAX-RS

"Quite" the same

both frameworks use an annotation based approach

Spring integrates JAX-RS provider

spring-boot-starter-jersey

Apache CXF is based on Spring

5.2

SOME DETAILS

API

`@RequestMapping`

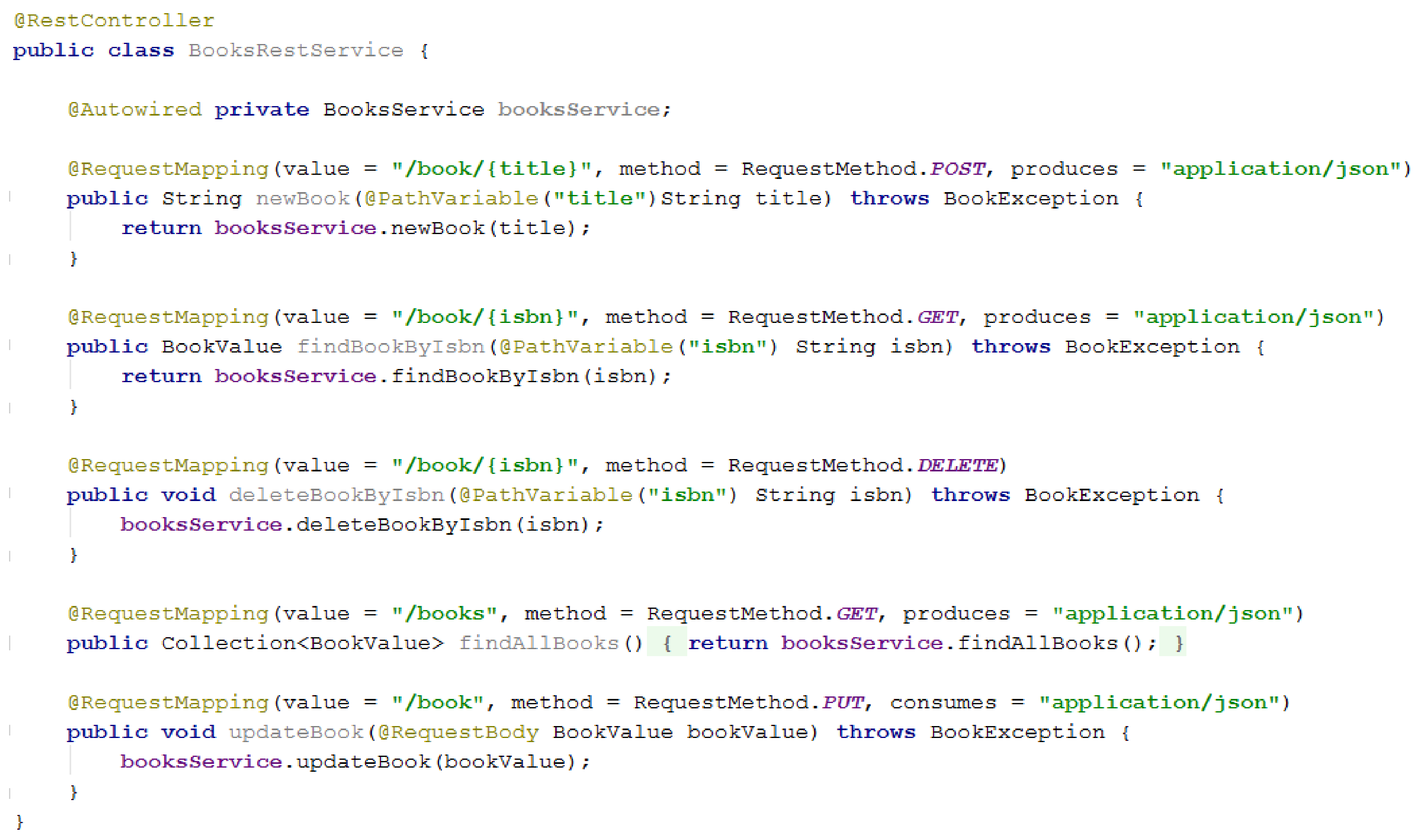
`@PathVariable, @RequestParam`

`@RequestBody, …`

`RequestEntity<T>, ResponseEntity<T>`

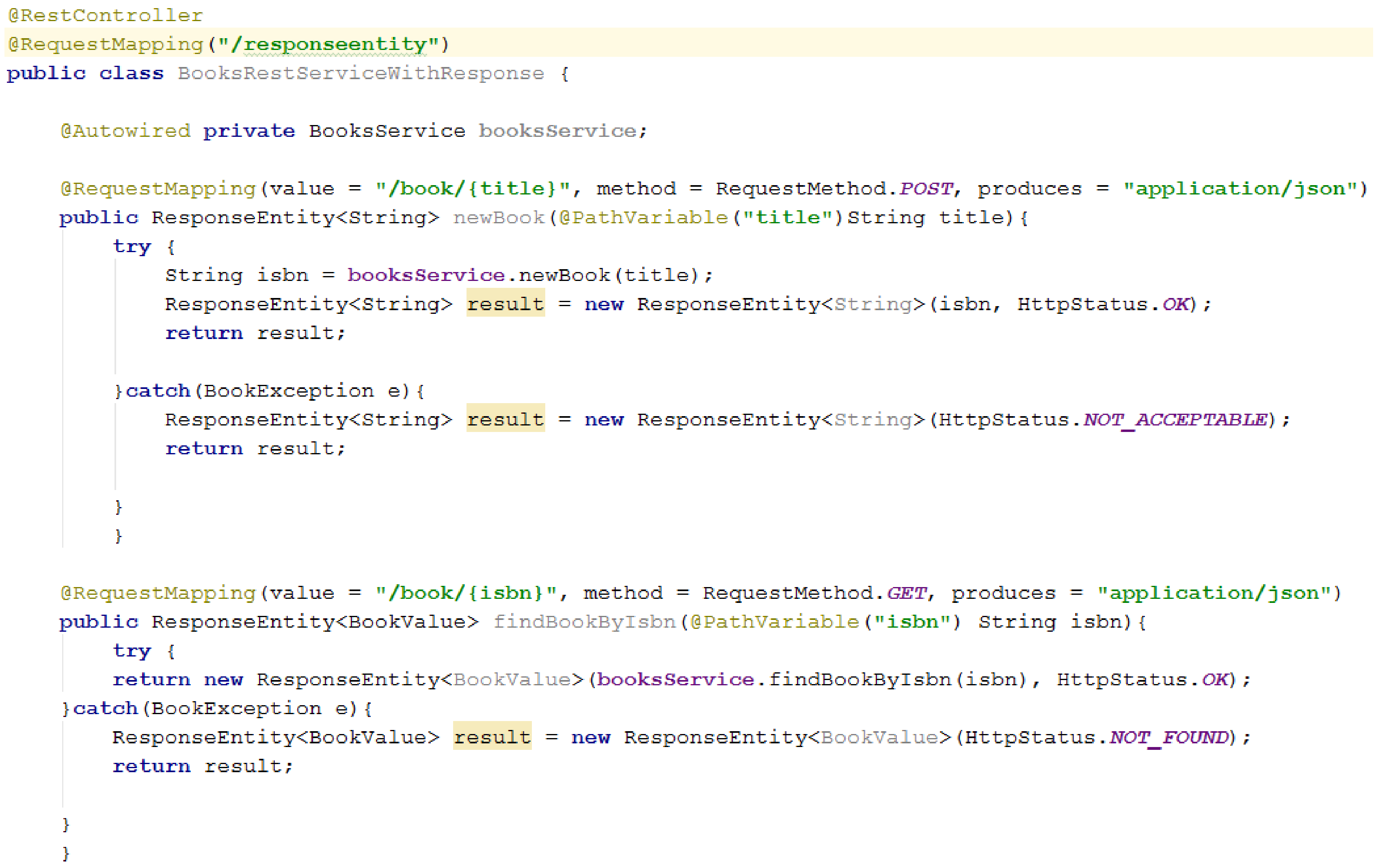
`@RestController`

Example 1: Annotations



```java
@RestController
public class BooksRestService {

    @Autowired private BooksService booksService;

    @RequestMapping(value = "/book/{title}", method = RequestMethod.POST, produces = "application/json")
    public String newBook(@PathVariable("title")String title) throws BookException {
        return booksService.newBook(title);
    }

    @RequestMapping(value = "/book/{isbn}", method = RequestMethod.GET, produces = "application/json")
    public BookValue findBookByIsbn(@PathVariable("isbn") String isbn) throws BookException {
        return booksService.findBookByIsbn(isbn);
    }

    @RequestMapping(value = "/book/{isbn}", method = RequestMethod.DELETE)
    public void deleteBookByIsbn(@PathVariable("isbn") String isbn) throws BookException {
        booksService.deleteBookByIsbn(isbn);
    }

    @RequestMapping(value = "/books", method = RequestMethod.GET, produces = "application/json")
    public Collection<BookValue> findAllBooks() { return booksService.findAllBooks(); }

    @RequestMapping(value = "/book", method = RequestMethod.PUT, consumes = "application/json")
    public void updateBook(@RequestBody BookValue bookValue) throws BookException {
        booksService.updateBook(bookValue);
    }
}
```

Example 2: ResponseEntity



```
@RestController
@RequestMapping("/responseentity")
public class BooksRestServiceWithResponse {

    @Autowired private BooksService booksService;

    @RequestMapping(value = "/book/{title}", method = RequestMethod.POST, produces = "application/json")
    public ResponseEntity<String> newBook(@PathVariable("title")String title){
        try {
            String isbn = booksService.newBook(title);
            ResponseEntity<String> result = new ResponseEntity<String>(isbn, HttpStatus.OK);
            return result;

        }catch(BookException e){
            ResponseEntity<String> result = new ResponseEntity<String>(HttpStatus.NOT_ACCEPTABLE);
            return result;

        }
        }

    @RequestMapping(value = "/book/{isbn}", method = RequestMethod.GET, produces = "application/json")
    public ResponseEntity<BookValue> findBookByIsbn(@PathVariable("isbn") String isbn){
        try {
        return new ResponseEntity<BookValue>(booksService.findBookByIsbn(isbn), HttpStatus.OK);
    }catch(BookException e){
        ResponseEntity<BookValue> result = new ResponseEntity<BookValue>(HttpStatus.NOT_FOUND);
        return result;

    }
    }
```

6

CLIENTS

Direct http

REST is HTTP

`java.net.URL`

Apache HttpClient

Open Source Library

Android devices use this library

RESTEasy

Open Source Library provided by JBoss/RedHat

REST services will be accesses using an interface

JAX RS

Generic Client-API

```
Client client = ClientBuilder.newClient();
```

Invoking a REST service using a fluent API`Response res =`

```
client.target("http://foo.org/hello").request("text/plain").get();
```

```
WebTarget messages = client.target("http://foo.org/messages/{id}");
WebTarget msg123 = messages.resolveTemplate("id", 123);
WebTarget msg456 = messages.resolveTemplate("id", 456);
```

Spring Rest: RestTemplate

```
import org.javacream.books.warehouse.BookValue;
import org.junit.Test;
import org.springframework.web.client.RestTemplate;

public class RestClientTest {

    @Test public void testRestService(){
        RestTemplate restTemplate = new RestTemplate();
        BookValue book =
restTemplate.getForObject("http://localhost:8080/book/ISBN1",
BookValue.class);
        System.out.println(book);
    }
}
```

Naturally: JavaScript

JavaScript clients and REST are a perfect match if you use JSON

JavaScript libraries like jQuery, Angular make it very easy to invoke a REST service

- `service.get("path_to_service", function(data){…})`