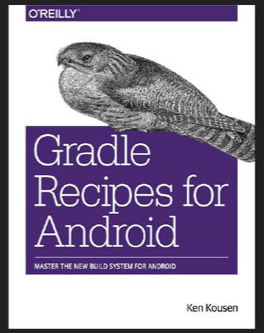
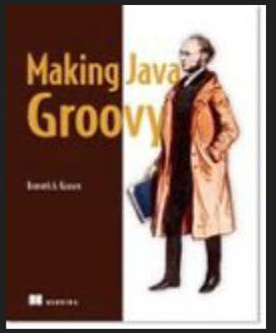
Reactive Spring

An Exercise-Driven Approach

Contact Info

Ken Kousen
Kousen IT, Inc.
ken.kousen@kousenit.com
http://www.kousenit.com
http://kousenit.org (blog)
@kenkousen





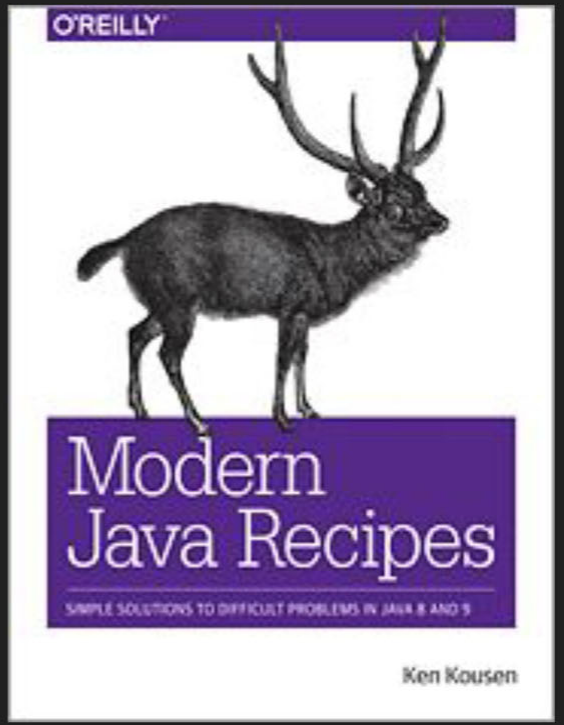
Modern Java Recipes

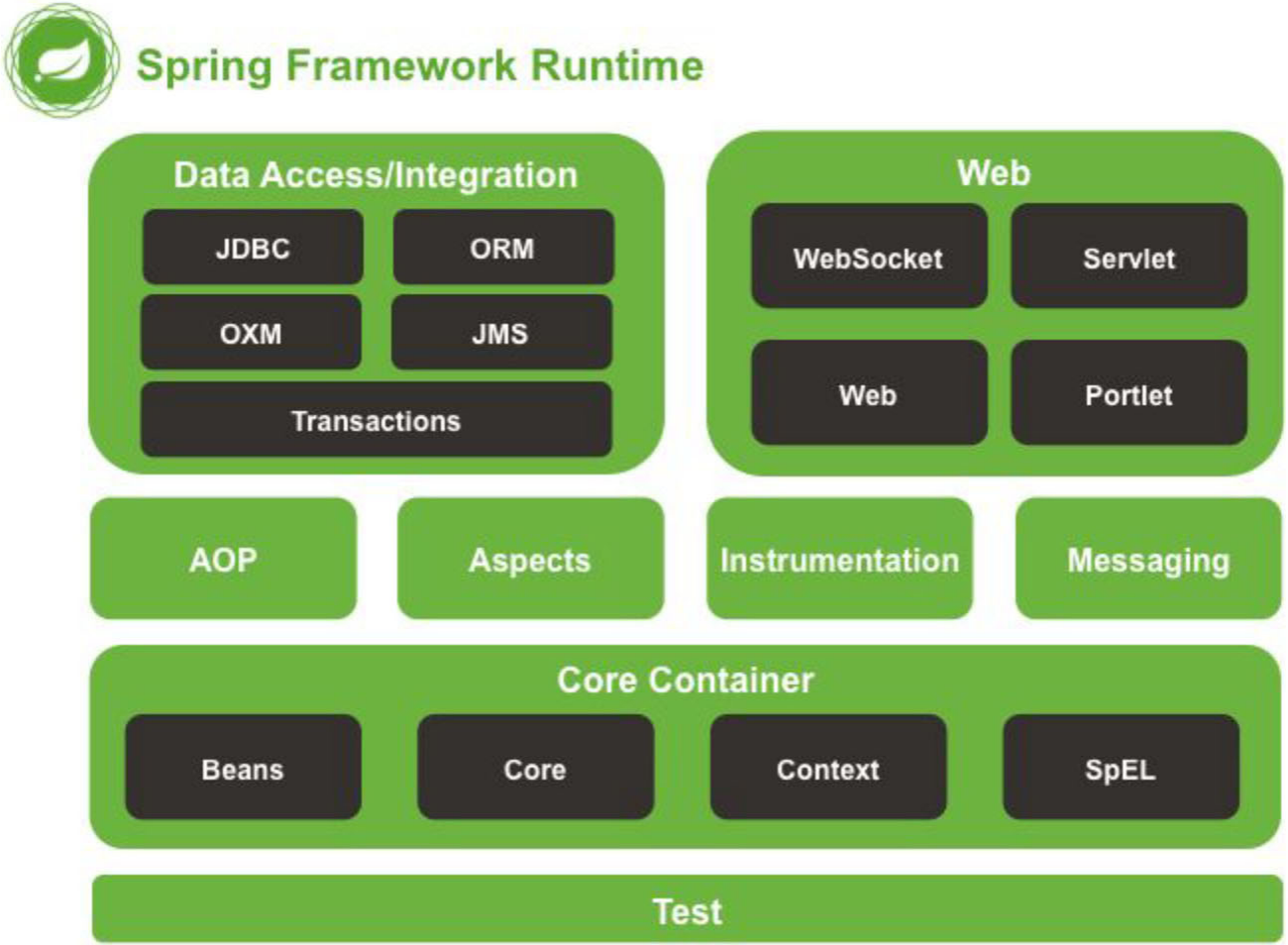
Source code:

https://github.com/kousen/java_8_recipes

https://github.com/kousen/cfboxscores

https://github.com/kousen/java_9_recipes




Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
WebSocket
Servlet
Web
Portlet
AOP
Aspects
Instrumentation
Messaging
Core Container
Beans
Core
Context
SpEL
Test

Spring

Project infrastructure

Spring

- Lifecycle management of "beans"
  - Any POJO with getters/setters

Spring

- Provides "services"
  - transactions, security, persistence, …

Spring

- Library of beans available
  - transaction managers
  - rest client
  - DB connection pools
  - testing mechanisms

Spring

- Code to interfaces
  - Library has many interfaces, each with many implementations

Spring

- Need "metadata"
  - Tells Spring what to instantiate and configure
    - XML → old style
    - Annotations → better
    - JavaConfig → preferred
  - All still supported

Spring

Application Context

- Collection of managed beans
- the "lightweight" Spring container

Spring Boot

- Easy creation and configuration for Spring apps
  - Many "starters"
  - Gradle or Maven based
- Automatic configuration based on classpath
  - If you add JDBC driver, it adds DataSource bean

Dependency Injection

- Spring adds dependencies on request
  - Annotate field, or setter, or constructor
  - `@Autowired` → autowiring by type
  - `@Resource` (from Java EE) → autowiring by (bean) name, then by type if necessary

Spring Initializr

Website for creating new Spring (Boot) apps

http://start.spring.io

Incorporated into major IDEs

Select features you want

Download zip containing build file

Spring Boot

Application with main method created automatically

Annotated with `@SpringBootApplication`

Gradle or Maven build produces executable jar in build/libs folder

```
$ java -jar appname.jar
```

Or use gradle task `bootRun`

Spring MVC

Annotation based MVC framework

`@Controller` → controllers

`@GetMapping` → annotations for HTTP methods

`@RequestParam` and more for model parameters

Rest Client

Spring includes a class called `RestTemplate`

- Access RESTful web services
- Set HTTP methods, headers, query string, templates
- Use `RestTemplateBuilder` to create one
- Use content negotiation to return JSON or XML
- Convenient `getForObject(url, class)` method

Rest Client

Spring 5 includes a new class called `WebClient`

- Can do async processing
- Understands Flux and Mono

Logging

Spring libraries include SLF4J automatically

- Use `LoggerFactory.getLogger(... class name ...)`
- Returns an `org.slf4j.Logger` instance
- Invoke logging methods as usual

Testing

Spring uses special JUnit runner

- `@RunWith(SpringRunner.class)`
- Annotate test class with `@SpringBootTest`
- Annotate tests with `@Test`
- Use normal asserts as usual

Testing

Special annotations for web integration tests

`@WebMvcTest(... controller class ...)`

`MockMvc` package

`MockMvcRequestBuilders`

`MockMvcRequestMatchers`

Parsing JSON

- Several options, but one is the `Jackson JSON 2` library
  - Create classes that map to JSON response
  - `restTemplate.getForObject(url, ... your class ...)`
- Maps JSON to Java objects

Component Scan

Spring detects annotated classes in the expected folders

`@Component` → Spring bean

`@Controller`, `@Service`, `@Repository` → based on `@Component`

Application properties

Two options for file name

Default folder is `src/main/resources`

- `application.properties` → standard Java properties file
- `application.yml` → YAML format

Persistence

- Spring provides `JdbcTemplate`
  - Easy to access and use relational databases
  - Best if you already have the SQL you want to use

Persistence

More conventions:

Two standard files in `src/main/resources`

`schema.sql` → create test database

`data.sql` → populate test database

Both executed on startup, using DB connection pool

H2 Database

- Add the H2 dependency
  - `runtime('com.h2database:h2')`
  - Automatically adds DataSource for it

If you add the web starter and the dev-tools dependency,

you also get the H2 console

http://localhost:8080/h2-console

DB URL (by default) is **`jdbc:h2:mem:testdb`**

JdbcTemplate

Standard practice:

- Create DAO class
- Autowire `DataSource` into constructor
- Instantiate `JdbcTemplate` from `DataSource`

JdbcTemplate

Use `queryForObject` to map DB row to Java class

(`query` method does the same for all rows)

In Java 7, uses inner class that implements `RowMapper<MyClass>`

In Java 8, can use lambda expression

SimpleJdbcInsert

- Specify table name and generated key columns
- Create a `SqlParameterSource`
- Run `executeAndReturnKey(parameters)`

Transactions

- Spring transactions configured with `@Transactional`
- Spring uses `TransactionManager` to talk to resource
  - usually a relational DB, but other options available

@Transactional

Each method wrapped in a REQUIRED tx by default

Propagation levels:

REQUIRED, REQUIRES_NEW, SUPPORTS, NOT_SUPPORTED

In tests, transactions in test methods roll back by default

Can configure isolation levels:

READ_UNCOMMITTED, READ_COMMITTED,

REPEATABLE_READ, SERIALIZABLE

JPA

Java Persistence API

- Uses a "provider" → Hibernate most common

Annotate entity classes

- `@Entity, @Table, @Column, @Id, @GeneratedValue`

use in Spring `@Repository` → exception translation

- `@PersistenceContext` → `EntityManager`

Spring Data

Large, powerful API

Create interface that extends a given one

`CrudRepository, PagingAndSortingRepository`

We'll use `JpaRepository<class, serializable>`

Add your own finder method declarations

All SQL generated automatically

HAL Browser

Browser used to access RESTful web services

Executes HTTP methods

Parses JSON responses

Handles hypermedia

Reactive Spring

Spring 5 → requires Java SE8

WebFlux module

Web on Reactive Stack

Spring Boot 2

Spring WebFlux Framework

WebFlux

Two approaches:

- Annotation-based → Similar to MVC
- Functional → Uses a "routing configuration"

Reactive Streams

- Industry specification with wide adoption
  - http://www.reactive-streams.org/
- Supported in Java 9
  - java.util.concurrent.Flow

Reactive Streams

- Four interfaces
  - Publisher
  - Subscriber
  - Subscription
  - Processor

Publisher

Provides a sequence of elements

Signals emitted:

- onSubscribe → always signaled
- onNext → possibly unbounded number of signals
- onError → only if there is a failure
- onComplete → no more elements available

Publisher

```
public interface Publisher<T> {
    void subscribe(Subscriber<? super T> s);
}
```

Subscriber

Receives signals

```
public interface Subscriber<T> {
    void onSubscribe(Subscription s);
    void onNext(T t);
    void onError(Throwable t);
    void onComplete();
}
```

Subscription

Sent from Publisher to Subscriber

```
public interface Subscription {
        void request(long n);
        void cancel();
}
```

Processor

A combination Publisher/Subscriber

```
public interface Processor<T,R>
        extends Subscriber<T>, Publisher<R> {
}
```

No additional methods

Project Reactor

Reactive library for the JVM based on Reactive Streams

Reactive Core → fully non-blocking

Typed sequences → Flux, Mono

Non-blocking IO with backpressure

WebFlux

- Annotated Controllers
  - Support reactive return types
    - Reactor, RxJava 2
- Functional Endpoints
  - Lambda-based, functional programming model
  - Library of utilities to route and handle requests

WebFlux

Uses Netty by default

Others: Undertow, Tomcat (Servlet 3.1+), Jetty (Servlet 3.1+)

@EnableWebFlux in Java config

WebClient

Reactive, non-blocking client for HTTP requests

Functional API with Java 8 lambdas

Both synchronous and asynchronous

Use WebTestClient for testing → mock request and response objects

WebFlux

- Functional Endpoints
  - HandlerFunction
  - RouterFunction

Exercises

HTML docs: http://www.kousenit.com/reactivespring/

Solutions:

https://github.com/kousen/reactive-officers

https://github.com/kousen/rest-service

https://github.com/kousen/spring-and-spring-boot (MVC, non-reactive)

Docs

- Spring API JavaDocs:
  - http://docs.spring.io/spring/docs/current/javadoc-api/
- Spring Reference Guide:
  - https://spring.io/docs/reference
- Spring Boot Reference Guide
  - http://docs.spring.io/spring-boot/docs/1.5.1.RELEASE/reference/htmlsingle/
- Spring Initializr
  - https://start.spring.io/

Docs

Wait, you're all on Safari, so…

- Learning Path: Learn Spring and Spring Boot
  - Includes Spring Framework Essentials
- Spring in Action, 4th Edition
- Spring Boot in Action
- … lots and lots more …