

# Spring Data Repositories Intro & Best Practices

Thomas Darimont  
MAJUG 2016



**Thomas Darimont**  
**Software Architect** >eurodata AG

**Spring Data Contributor**  
Former Member of Spring Team @ Pivotal

**Java User Group**  
**Saarland**



**thomas.darimont@gmail.com**  
**@thomasdarimont**

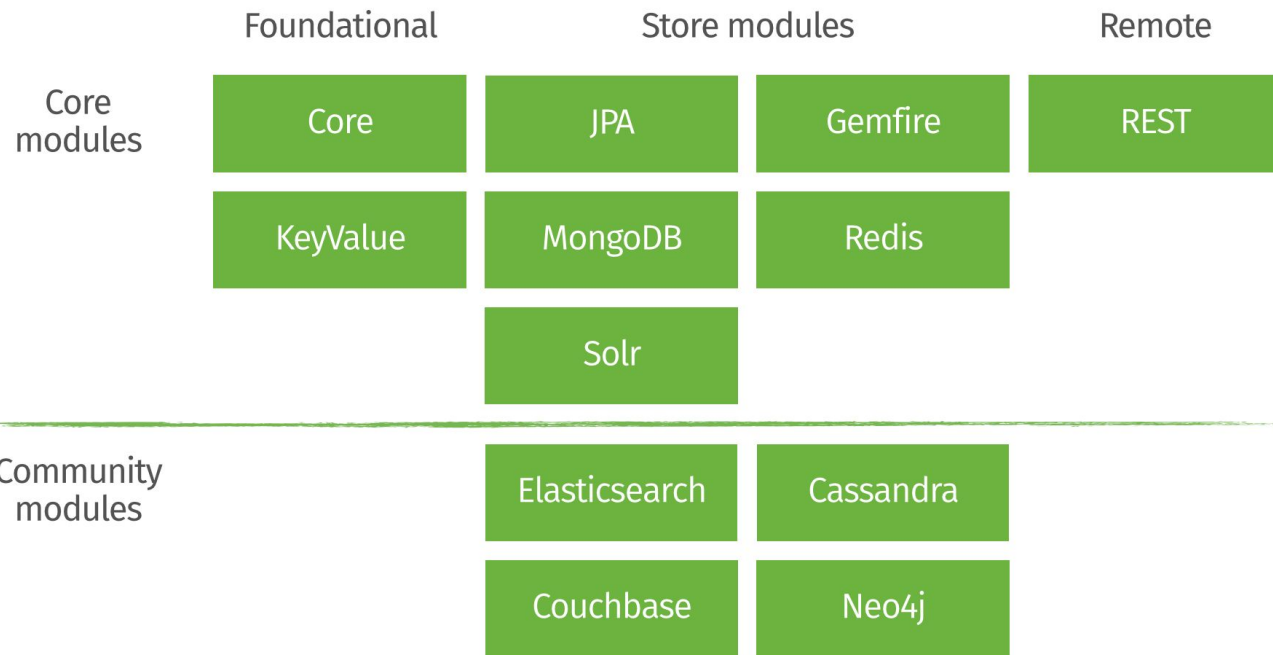
# Spring Data

*“ Provides a consistent approach to data access for several persistence models ”*

*with support for*

*Configuration, Templates, Exception Translation, Object Mapping,  
**Repository Abstraction** and various Datastores*

# Spring Data Modules & Datastore Support

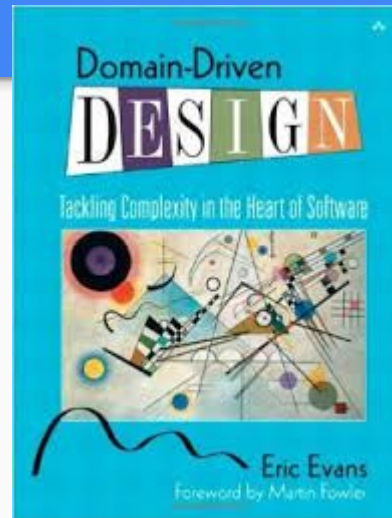


Concept from *Domain Driven Design*

*A Repository...*

*“...mediates between the **domain** and **data mapping layers** using a **collection-like** interface for **accessing domain objects**.”*

<http://martinfowler.com/eaCatalog/repository.html>



- Pragmatic Data Access API's
- Interface based Programming Model
- Provides useful Base Abstractions
- Automatic Query Generation
- ... and much much more - let's find out!

# Hands On

All code on github!

Steps as individual commits :)



[jugsaar/visit-majug-20170413-sd-repositories-bp](https://github.com/jugsaar/visit-majug-20170413-sd-repositories-bp)

# Step 0 - Project Setup

## *Requirement*

*“Setup Data Access Infrastructure with Spring, JPA, H2”*

1. Spring Source Toolsuite (STS) / IntelliJ
2. Spring Boot Starter to create Maven Project
3. Add JPA + H2 Dependency
4. Profit!



# Step 0 - Summary

- Spring Boot + Spring Data = Easy!
  - Easily configure dependencies
  - Defaults application config from classpath
- Spring Data Infrastructure Ready to use!

# Step 1 - Setup Domain Model

## *Requirement*

*“Define Domain Model, Populate & Test Database”*

- Define basic JPA entities
- Populate Database with a Script
- Test Population

# Step 1 - Summary

- We defined
  - Customer & Address Entities
  - EmailAddress ValueObject
- Populated Database via `data.sql`
- Tested Population via Integration Test

# Step 2 - Enable JPA Repositories

## *Requirement*

*“Customers can be **saved, looked up** by their **id** and **email address**.”*

- Define Customer Repository
- Basic CRUD functionality
- Based on Spring Data Repositories

## Step 2 - Summary

- Interface-based programming model
- No implementation required
- Proxy with Implementation provided by Spring Data
- Queries derived from method names

## Step 3 - Extended CRUD Methods

### *Requirement*

“Customers can be ***deleted*** and ***listed!***”

## Step 3 - Summary

- Switched to `CrudRepository`
- `CustomerRepository` is now package protected
- Exposed CRUD methods
- Broad API exposed

## Step 4 - Add Pagination Support

### *Requirement*

“A User

... can ***pagewise*** browse through a ***Customer list***.

... wants to ***browse*** through the ***list*** of Customers ***sorted by lastname in desc order.***”



## Step 4 - Summary

- Switched to `PagingAndSortingRepository`
- Exposed CRUD methods + Pagination
- Broad API exposed
- Tip: Use `Slice` instead of `Page` if possible
  - `Slice` as a Return type
  - Avoids count queries

## Step 5 - Redeclare existing methods

### *Requirement*

*“CustomerRepository.findAll()  
should return a List.”*

*“The **transaction timeout** for **save(...)** should be  
**customized to 10 seconds**”*

## Step 5 - Summary

- Re-declared methods
- To customize return types
- Customize Behaviour *e.g. TX, Locking, Query, Hints, Fetching*
- Customize Query

# Step 6 - Def. your own Abstractions

*Requirement*

***“Products\* shall be **accessible** only **in read-only mode.**”***

## Step 6 - Summary

- We crafted our own *custom abstraction*
  - `ReadOnlyRepository`
- Applied by implementing base interface
- Customized return types
- Narrowed down the API to the necessary parts

# Step 7 - Using custom queries

## *Requirement*

“As a user, I want to ***look up products by custom attributes.***”

# Step 7 - Summary

- You can customize the queries
- Via `@Query` annotation
- JPA named queries
  - `@NamedQuery`
  - `<named-query>` in `orm.xml`
- Spring Data named queries properties file
  - e.g. `jpa-named-queries.properties`

# Step 8 - Flexible Predicate execution

## *Requirement*

“As a user, I want to **search** for **customers by first name, last name, email address** and **any combination** of them”



## Step 8 - Summary

- Querydsl - type safe queries for Java
- extend QuerydslPredicateExecutor
- Introduced flexible query model

# Step 9 - Custom Implementations

## *Requirement*

“As an admin user, I'd like to *use custom code* to *raise* all prices before winter sale.”

## Step 9 - Summary

- Provide custom implementation
- Support for Dependency Injection
- Base class support  
(Querydsl, Hibernate, Jdbc-DaoSupport)

# Step 10 - Custom base class

## *Requirement*

*“I'd like to use a custom base class with new methods for all repositories.”*

# Step 10 - Summary

- Provide custom base class
- via `@EnableJpaRepository(repositoryBaseClass=MyRepoBaseClass.class)`
- Reuse existing functionality and add your own

# Step 11 - Predicates from MVC Request

## *Requirement*

*"I'd like to create flexible query predicates based on http request/URL parameters"*

## Step 11 - Summary

- Bind Request Parameters to a QueryDSL Predicate
- Predicate as Parameter in your MVC Controller method

# Stuff on top

- Spring MVC integration
  - Native support for Pagination
  - Inject Domain Instances into your MVC handlers
  - Expose parts of Domain Model via Projections
- Spring Data REST
- Spring Boot Integration
- Spring Security Integration



# Spring Data Repositories

## Summary

# Spring Data Repositories

Interface-based programming model

# Spring Data Repositories

Start simple, get more sophisticated

# Spring Data Repositories

## Declarative Query Execution

# Spring Data Repositories

## Flexible Predicate Execution

# Spring Data Repositories

## Custom Implementations

# Spring Data Repositories

## CDI Integration

# Spring Data Repositories

## Spring Security Integration



# Spring Data Examples



[spring-projects/spring-data-examples](https://github.com/spring-projects/spring-data-examples)

## Questions



### CONTENTS INCLUDE:

- » About the Spring Data Project
- » Configuration Support
- » Object Mapping
- » Template APIs
- » Repositories
- » Advanced Features... and more!

## Core Spring Data

By: Oliver Gierke

### ABOUT THE SPRING DATA PROJECT

The Spring Data project is part of the ecosystem surrounding the Spring Framework and constitutes an umbrella project for advanced data access related topics. It contains modules to support traditional relational data stores (based on plain JDBC or JPA), NoSQL ones (like MongoDB, Neo4j or Redis), and big data technologies like Apache Hadoop. The core mission of the project is to provide a familiar and consistent Spring-based programming model for various data access technologies while retaining store-specific features and capabilities.

#### General Themes

##### Infrastructure Configuration Support

A core theme of all the Spring Data projects is support for configuring resources to access the underlying technology. This support is implemented using XML namespaces and support classes for Spring JavaConfig allowing you to easily set up access to a Mongo database, an embedded Neo4j instance, and the like. Also, integration with core Spring functionality like JMX is provided, meaning that some stores will expose statistics through their native API, which will be exposed to JMX via Spring Data.

##### Object Mapping Framework

Most of the NoSQL Java APIs do not provide support to map domain objects onto the stores' data model (e.g., documents in MongoDB, or nodes and relationships for Neo4j). So, when working with the native Java drivers, you would usually have to write a significant amount of code to map data onto the domain objects of your application when reading, and vice versa on writing. Thus, a core part of the Spring Data project is a mapping and conversion API that allows obtaining metadata about domain classes to be persisted and enables the conversion of arbitrary domain

### JPA

XML element	Description
<jpa:repositories />	Enables Spring Data repositories support for repository interfaces underneath the package configured in the base-package attribute. JavaConfig equivalent is @EnableJpaRepositories.
<jpa:auditing />	Enables transparent auditing of JPA managed entities. Note that this requires the AuditingEntityListener applied to the entity (either globally through a declaration in orm.xml or through @EntityListener on the entity class).

### MongoDB

For Spring Data MongoDB XML namespace elements not mentioning a dedicated @Enable annotation alternative, you usually declare an @Bean-annotated method and use the plain Java APIs of the classes that would have otherwise been set up by the XML element. Alternatively, you can use the JavaConfig base class AbstractMongoConfiguration that Spring Data MongoDB ships for convenience.

XML element	Description
<mongo:db-factory />	One stop shop to set up a Mongo instance pointing to a particular database instance. For advanced-use cases define a <mongo:mongo /> externally and refer to it using a mongo-ref attribute.
<mongo:mongo />	Configures a Mongo instance. Supports basic attributes like host, port, write concern etc. Configure