Data Access 2.0?

...please welcome...

Spring Data!

Oliver Gierke

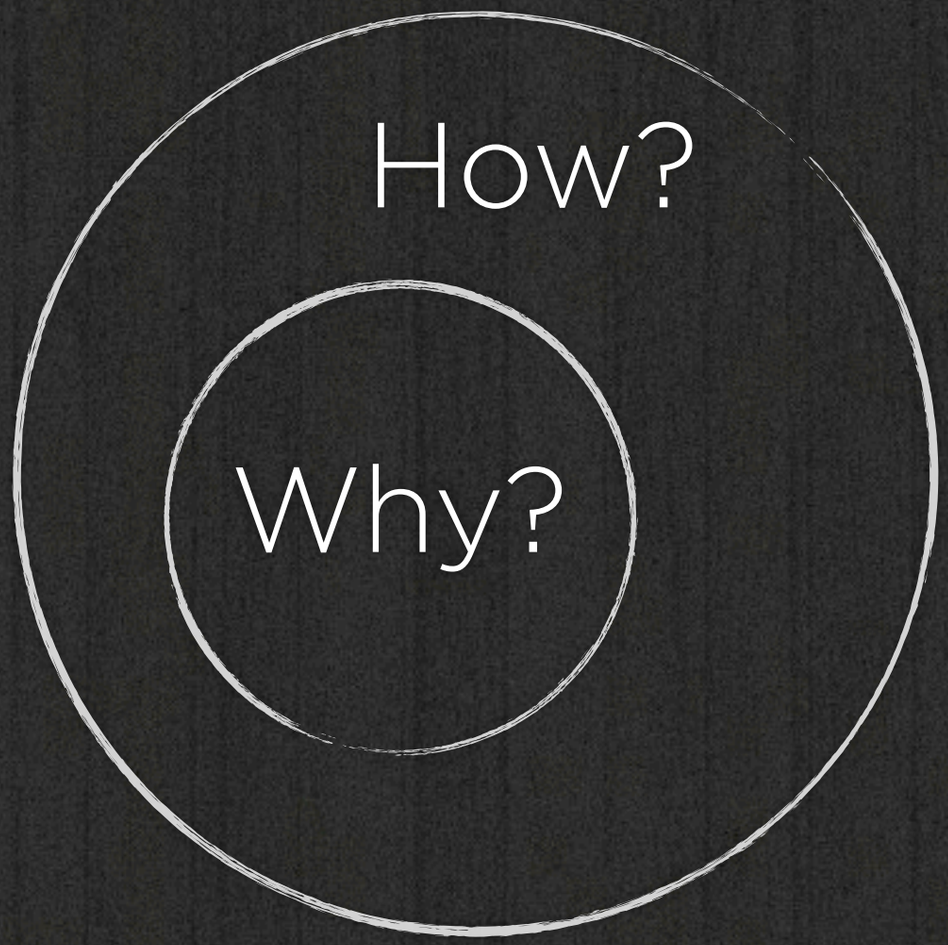
Oliver Gierke

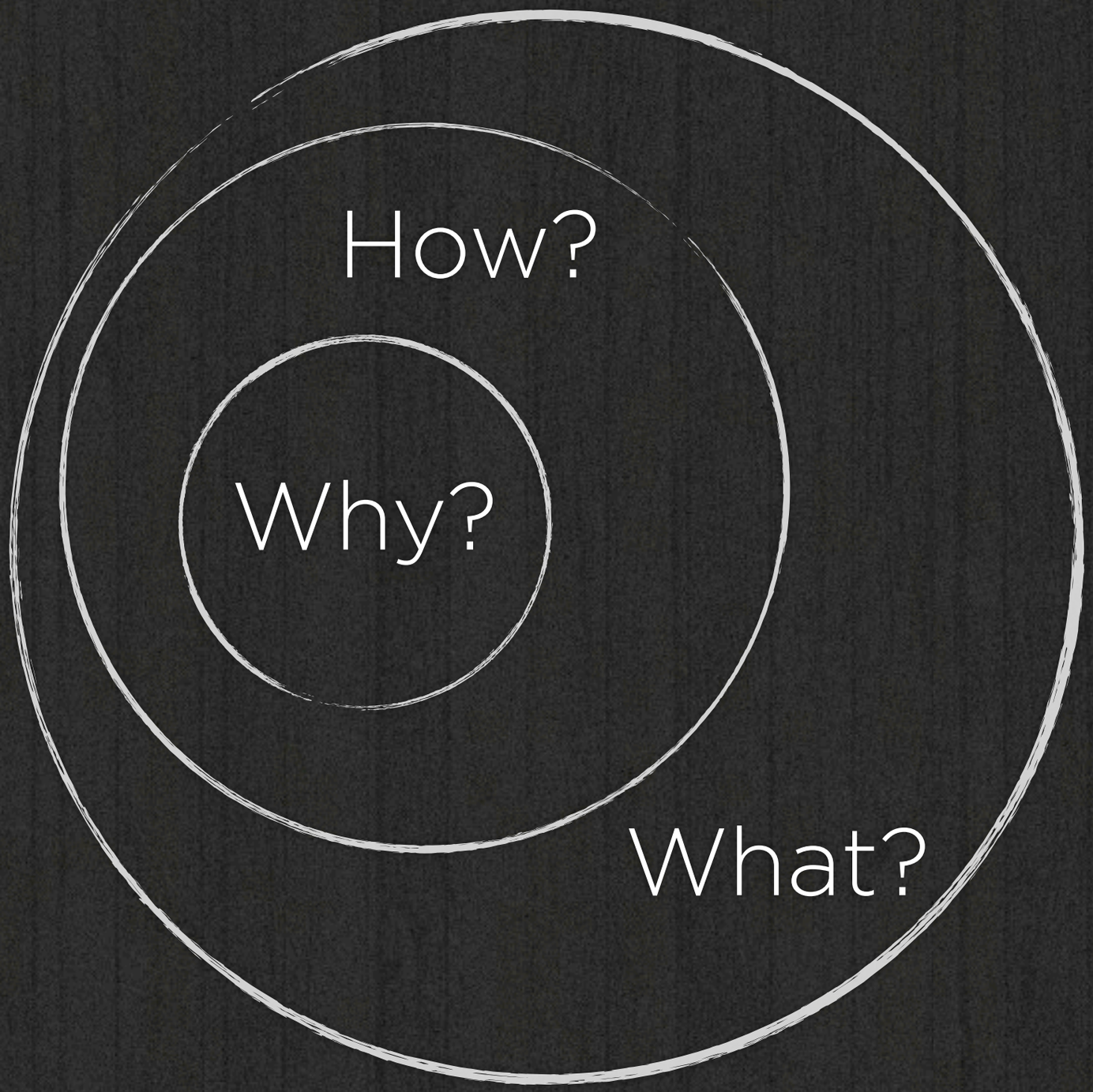
Spring Data
Core/JPA/MongoDB
JPA 2.1 Expert Group

ogierke@gopivotal.com
www.olivergierke.de
olivergierke

What to expect?


Why?


How?
Why?


How?
Why?
What?

A Developer‘s View

What to expect?

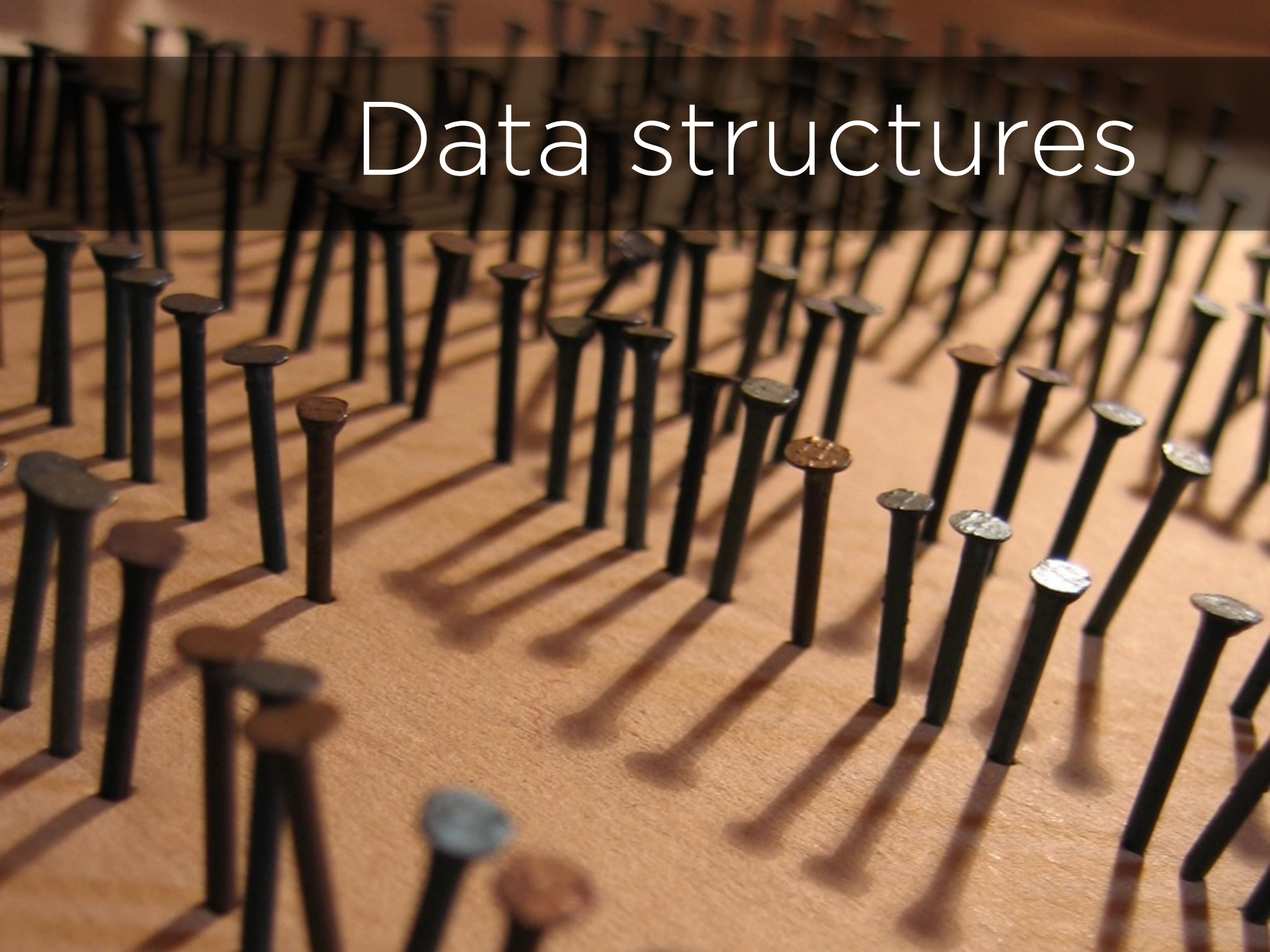
NOT!


ZIMA
NONALCOHOLIC BEER
COLD BEER
FROZEN PIZZAS
FROZEN DINNERS
FROZEN VEGETABLES
WATER
BUD ICE LIGHT
BLACK TAN
BUD LIGHT
BUSCH
BUSCH LIGHT
Budweiser
Natural Light

Retrospect

Relational databases

Data structures



Querying

Hibari

Voldemort

Membase

Riak

Cassandra

Redis

(No)SQL

SimpleDB

MongoDB

OrientDB

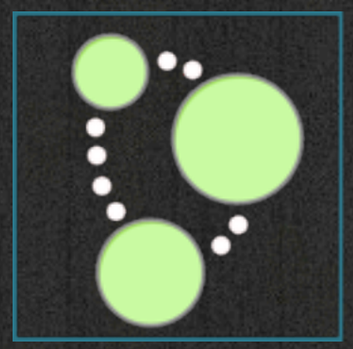
CouchDB

HBase

Couchbase

Neo4J

mongoDB



Neo4j

the graph database

Document database

JSON documents

JSON queries

MongoDB Infrastructure API

```
Mongo mongo = new Mongo(...);
DB db = mongo.getDB("myDatabase");
Collection collection = db.getCollection("myCollection");

DBObject address = new BasicDBObject();
address.put("city", "London");

DBObject person = new BasicDBObject();
person.put("firstname", "Dave");
person.put("lastname", "Matthews");
person.put("address", address);

collection.save(person);
```

MongoDB Query API

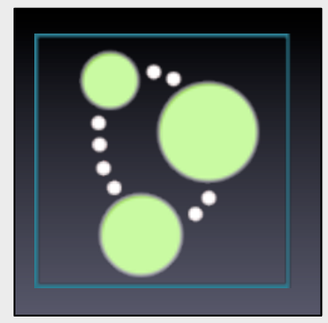
```
Mongo mongo = new Mongo(...);
DB db = mongo.getDB("myDatabase");
Collection collection = db.getCollection("myCollection");

DBObject query = new BasicDBObject();
query.put("address.city", "London");

DBCursor cursor = collection.find(query);

for (DBObject element : cursor) {
  // Map data onto object
}
```

Neo4j
the graph database

Graph database
Nodes / Relationships
Traversals / Cypher / Gremlin

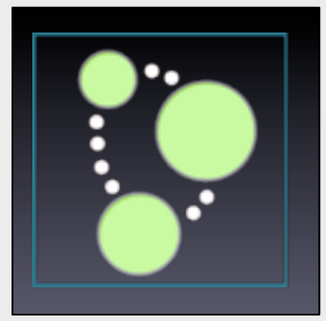
Neo4J Infrastructure API

```
GraphDatabaseService database = new EmbeddedGraphDatabase(...);
Transaction tx = database.beginTx();

try {
    Node mrAnderson = database.createNode();
    mrAnderson.setProperty("name", "Thomas Anderson");
    Node morpheus = database.createNode();
    morpheus.setProperty("name", "Morpheus");

    Relationship friendship = mrAnderson.createRelationshipTo(
        morpheus, FriendTypes.KNOWS);

    tx.success();
} finally {
    tx.finish();
}
```



Neo4J Query API

```
GraphDatabaseService database = new EmbeddedGraphDatabase(...);

CypherParser parser = new CypherParser();
Query query = parser.parse("start person = Person(id = *) match " +
    "person-[:colleagues]->colleague where colleague.firstname = {name}");

Map<String, Object> parameters = new HashMap<String, Object>();
parameters.put("name", "Dave");

ExecutionEngine engine = new ExecutionEngine(database);
ExecutionResult result = engine.execute(query, parameters);

for (EntrySet<String, Object> element : result) {
    // Map data onto object
}
```



Forest for the woods?

JPA?

"

This document is the specification of the Java API for the management of persistence and object/relational mapping with Java EE and Java SE. The technical objective of this work is to provide an object/relational mapping facility for the Java application developer using a Java domain model to man- age a relational database.

"

This document is the specification of the Java API for the management of persistence and **object/relational mapping** with Java EE and Java SE. The technical objective of this work is to provide an **object/relational mapping** facility for the Java application developer using a Java domain model to man- age a relational database.

JPA

JPA

@Id
@Entity

JPA

@Table
@Column

TX?

@Id
@Entity

JPA

@Table
@Column

TX?

@Id
@Entity

@Index
@Field
@RelationshipType

GeoSpatial
Graph traversals

JPA

Store-specific
functionality

@Table
@Column

TX?

@Id
@Entity

@Index
@Field
@RelationshipType

GeoSpatial
Graph traversals

JPA

Store-specific
functionality

JPA?

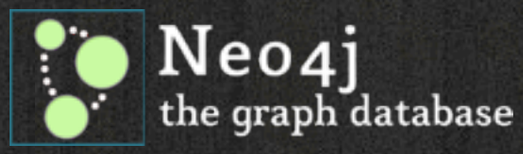
~~JPA?~~

There‘s some Spring for that!

Spring Data

> ... provide a familiar and consistent Spring-based programming model while retaining store-specific features and capabilities.

Spring Data



JDBC / JPA

Spring Data

riak

mongoDB

Neo4j
the graph database

redis

Couchbase

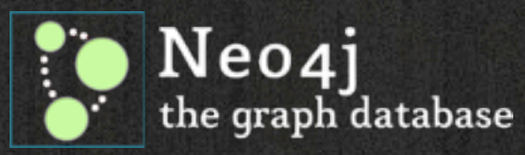
hadoop

Apache Solr

elasticsearch.

JDBC / JPA

Spring Data



JDBC / JPA

Spring Data

JDBC / JPA

Spring Data

riak

mongoDB

Neo4j
the graph database

redis

Couchbase

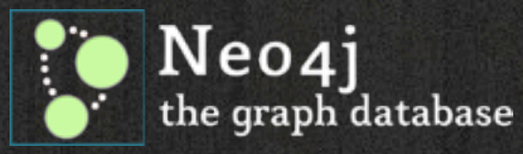
hadoop

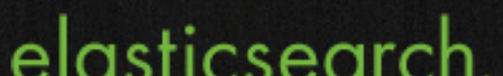
Apache Solr

elasticsearch.

JDBC / JPA

Spring Data





JDBC / JPA

Spring Data

riak

mongoDB

Neo4j
the graph database

redis

Couchbase

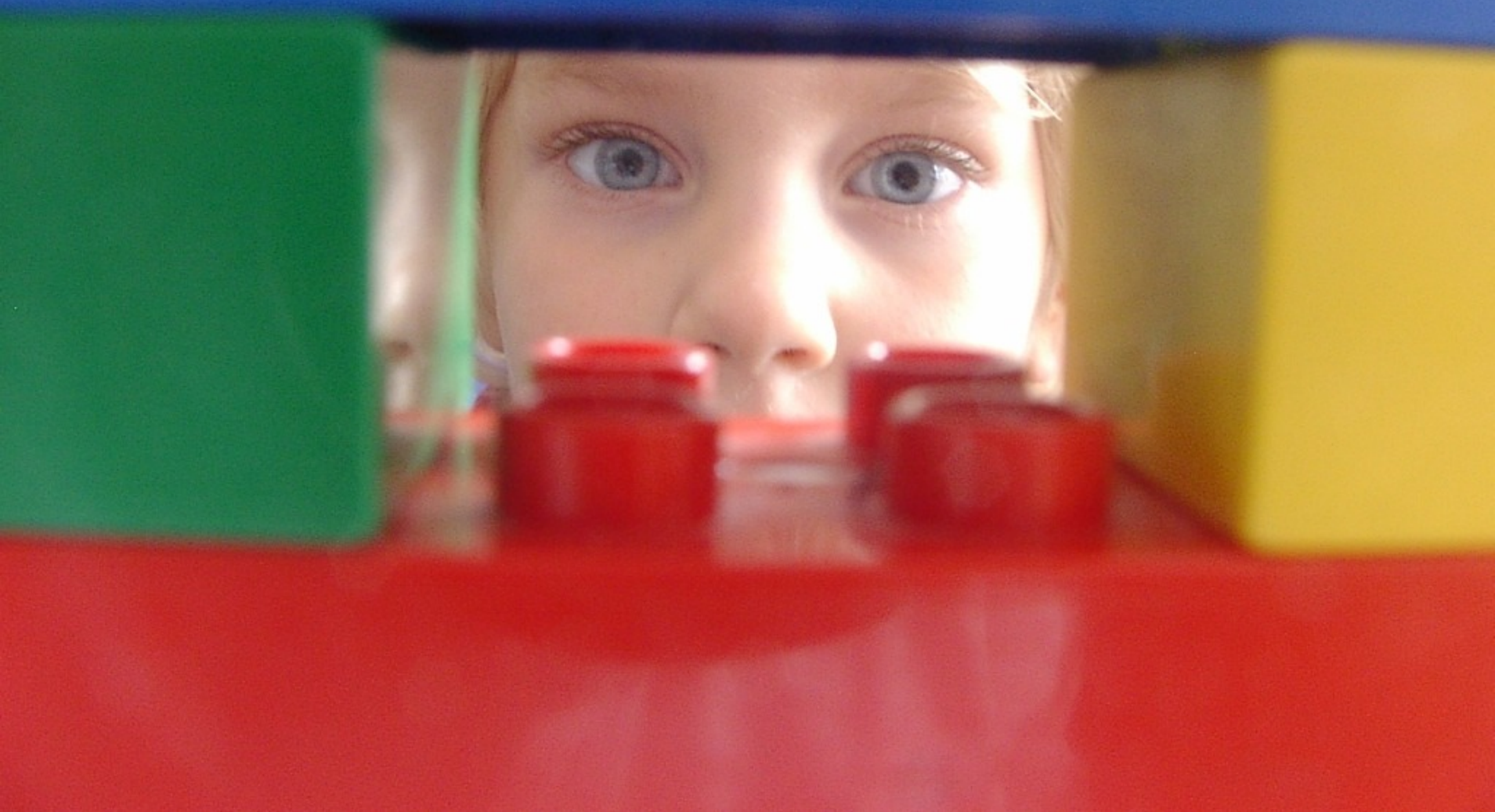
hadoop

Apache Solr

elasticsearch.

JDBC / JPA

Building blocks



Spring

Mapping

JPA entity mapping

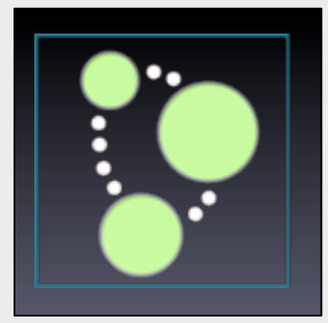
```
@Entity
class Person {

   @Id
   @GeneratedValue(strategy=GenerationType.AUTO)
   private BigInteger id;
   private String firstname, lastname;

   @Column(name="email")
   private String emailAddress;

   @OneToMany
   private Set<Person> colleagues;
}
```

Entity mapping - MongoDB

```
@Document
class Person {

  @Id private BigInteger id;
  @Indexed private String firstname, lastname;
  @Field("email") private String emailAddress;
  @DBRef private Set<Person> colleagues;

  public Person(String firstname) { ... }

  @PersistenceConstructor
  public Person(String firstname, String lastname) { ... }

  ...
}
```

Entity mapping - Neo4J

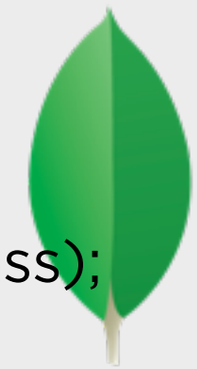
```
@NodeEntity
class Person {

    @GraphId private long id;
    @Indexed private String firstname, lastname;
    @RelatedTo(direction = Direction.INCOMING)
    private Set<Person> colleagues;

    public Person(String firstname) { ... }

    @PersistenceConstructor
    public Person(String firstname, String lastname) { ... }

    ...
}
```



Templates

MongoOperations / -Template

```
public interface MongoOperations {

    // Generic callback-accepting methods
    <T> T execute(DbCallback<T> action);
    <T> T execute(Class<?> entityClass, CollectionCallback<T> action);
    <T> T execute(String collectionName, CollectionCallback<T> action);

    // Higher level access methods
    <T> List<T> find(Query query, Class<T> entityClass);
    void save(Object objectToSave, String collectionName);
    WriteResult updateFirst(Query query, Update update, Class<?>
        entityClass);

    // Geo API
    <T> GeoResults<T> geoNear(NearQuery near, Class<T> entityClass);
}
```



MongoTemplate usage

```
// Setup infrastructure
Mongo mongo = new Mongo();
MongoDbFactory factory = new SimpleMongoDbFactory(mongo, „foo“);
MongoTemplate template = new MongoTemplate(factory);

// Create and save entity
Person dave = new Person("Dave", "Matthews");
dave.setEmailAddress("dave@dmband.com");
template.save(person);

// Query entity
Query query = new Query(new Criteria("emailAddress")
                                        .is("dave@dmband.com"));
assertThat(template.findOne(query, Person.class), is(dave));
```

Repositories

Repositories - JPA

```
<jpa:repositories base-package="com.acme.repositories" />

public interface PersonRepository extends Repository<Person, BigInteger>
{
   // Finder for a single entity
   Person findByEmailAddress(String emailAddress);

   // Finder for multiple entities
   List<Person> findByLastnameLike(String lastname);

   // Finder with pagination
   Page<Person> findByFirstnameLike(String firstname, Pageable page);
}
```

Repositories - MongoDB

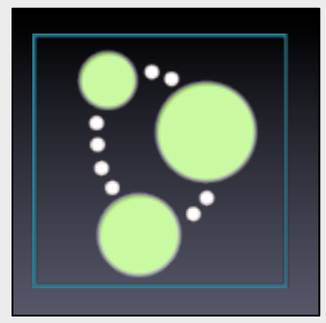
```
public interface PersonRepository extends Repository<Person, BigInteger>
{
   // Finder for a single entity
   Person findByEmailAddress(String emailAddress);

   // Finder for multiple entities
   List<Person> findByLastnameLike(String lastname);

   // Finder with pagination
   Page<Person> findByFirstnameLike(String firstname, Pageable page);

   // Geospatial queries
   List<Person> findByLocationNear(Point location, Distance distance);
   GeoResults<Person> findByLocationNear(Point location);
}
```

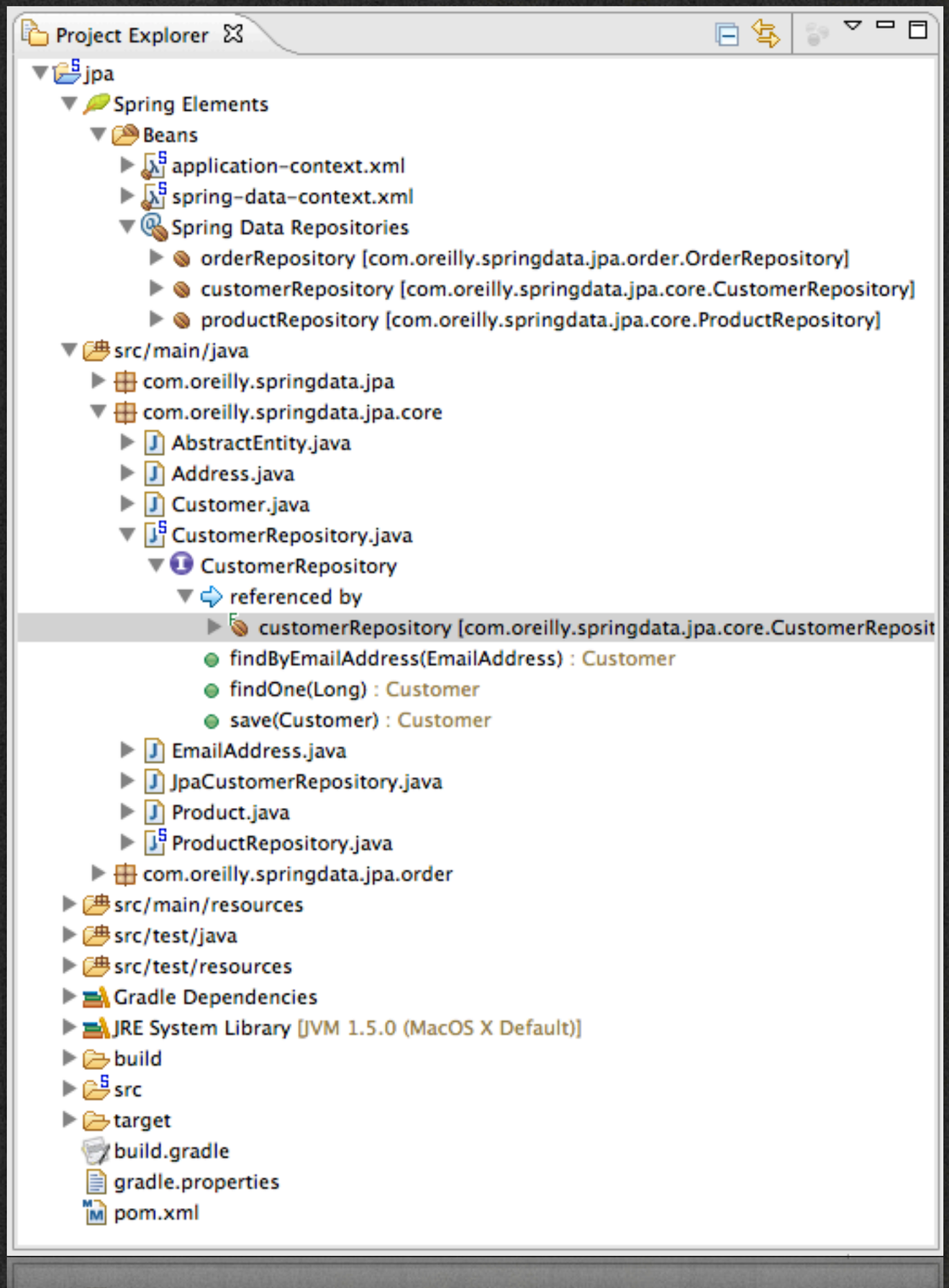
Repositories - MongoDB

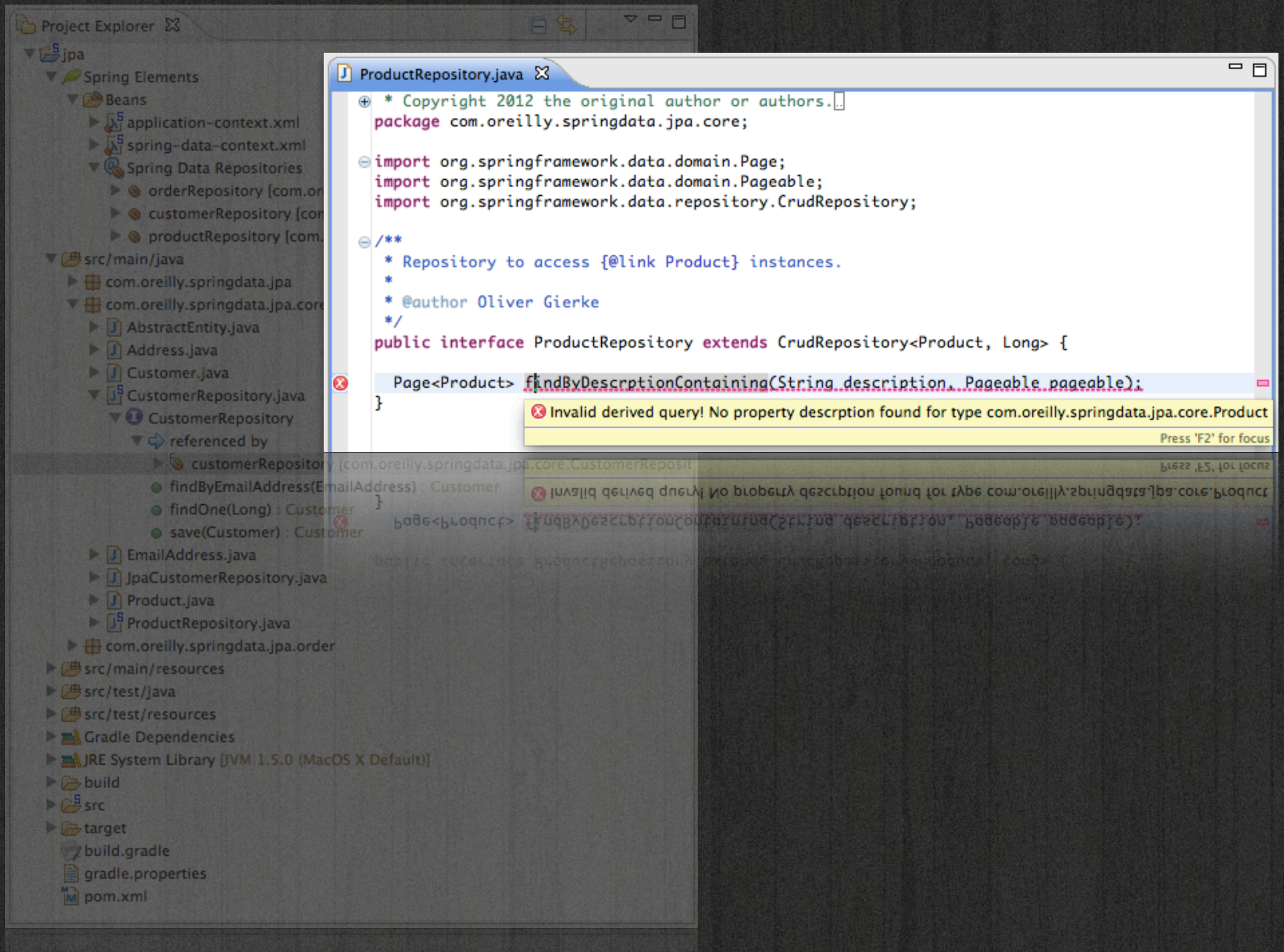
```
<mongo:repositories base-package="com.acme.repositories" />

@Component
public class MyClient {

   @Autowired
   private PersonRepository repository;

   public List<Person> doSomething() {

      Point point = new Point(43.7, 48.8);
      Distance distance = new Distance(200, Metrics.KILOMETERS);
      return repository.findByLocationNear(point, distance);
   }
}
```

Repositories - Neo4J

```
interface PersonRepository extends GraphRepository<Person, Long>
    // Finder for a single entity
    Person findByEmailAddress(String emailAddress);

    // Finder for multiple entities
    List<Person> findByLastnameLike(String lastname);

    // Finder with pagination
    Page<Person> findByFirstnameLike(String firstname, Pageable page);

    @Query("start person = Person(id = *) " +
                "match person-[:colleagues]->colleague where " +
                "colleague.firstname = {name}")
    List<Person> getPersonsWithColleaguesName(
                                        @Param("name") Movie m);
}
```




Project Explorer
jpa
Spring Elements
Beans
application-context.xml
spring-data-context.xml
Spring Data Repositories
orderRepository [com.oreilly.springdata.jpa.order.OrderRepository]
customerRepository [com.oreilly.springdata.jpa.core.CustomerRepository]
productRepository [com.oreilly.springdata.jpa.core.ProductRepository]
src/main/java
com.oreilly.springdata.jpa
com.oreilly.springdata.jpa.core
AbstractEntity.java
Address.java
Customer.java
CustomerRepository.java
CustomerRepository
referenced by
customerRepository [com.oreilly.springdata.jpa.core.CustomerReposit
findByEmailAddress(EmailAddress) : Customer
findOne(Long) : Customer
save(Customer) : Customer
EmailAddress.java
JpaCustomerRepository.java
Product.java
ProductRepository.java
com.oreilly.springdata.jpa.order
src/main/resources
src/test/java
src/test/resources
Gradle Dependencies
JRE System Library [JVM 1.5.0 (MacOS X Default)]
build
src
target
build.gradle
gradle.properties
pom.xml


Project Explorer
jpa
Spring Elements
Beans
application-context.xml
spring-data-context.xml
Spring Data Repositories
orderRepository [com.or
customerRepository [cor
productRepository [com.
src/main/java
com.oreilly.springdata.jpa
com.oreilly.springdata.jpa.core
AbstractEntity.java
Address.java
Customer.java
CustomerRepository.java
CustomerRepository
referenced by
customerRepository [com.oreilly.springdata.jpa.core.CustomerReposit
findByEmailAddress(EmailAddress) : Customer
findOne(Long) : Customer
save(Customer) : Customer
EmailAddress.java
JpaCustomerRepository.java
Product.java
ProductRepository.java
com.oreilly.springdata.jpa.order
src/main/resources
src/test/java
src/test/resources
Gradle Dependencies
JRE System Library [JVM 1.5.0 (MacOS X Default)]
build
src
target
build.gradle
gradle.properties
pom.xml
ProductRepository.java
* Copyright 2012 the original author or authors.
package com.oreilly.springdata.jpa.core;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.repository.CrudRepository;
/**
* Repository to access {@link Product} instances.
*
* @author Oliver Gierke
*/
public interface ProductRepository extends CrudRepository<Product, Long> {
Page<Product> findByDescrptionContaining(String description, Pageable pageable);
}
Invalid derived query! No property descrption found for type com.oreilly.springdata.jpa.core.Product
Press 'F2' for focus

Project Explorer

- jpa
  - Spring Elements
    - Beans
      - application-context.xml
      - spring-data-context.xml
      - Spring Data Repositories
        - orderRepository [com.oreilly.springdata.jpa.order.OrderRepository]
        - customerRepository [com.oreilly.springdata.jpa.core.CustomerRepository]
  - Gradle Dependencies
  - JRE System Library [JVM 1.5.0 (MacOS X Default)]
  - build
  - src
  - target
  - build.gradle
  - gradle.properties
  - pom.xml

ProductRepository.java

```
 * Copyright 2012 the original author or authors.
package com.oreilly.springdata.jpa.core;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.repository.CrudRepository;

} instances.

ends CrudRepository<Product, Long> {

ning(String description, Pageable pageable);
```

property descrption found for type com.oreilly.springdata.jpa.core.Product

Press 'F2' for focus

*Order.java

```
/**
 * An order.
 *
 * @author Oliver Gierke
 */
@Entity
@Table(name = "Orders")
public class Order extends AbstractEntity {

  @ManyToOne
  private Customer customer;
  @ManyToOne
  private Address billingAddress;
  @ManyToOne
  private Address shippingAddress;

  @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
  @JoinColumn(name = "order_id")
  private Set<LineItem> lineItems = new HashSet<LineItem>();

  // …
```

Project Explorer

- jpa
  - Spring Elements
    - Beans
      - application-context.xml
      - spring-data-context.xml
    - Spring Data Repositories
      - orderRepository [com.oreilly.springdata.jpa.order.OrderRepository]
      - customerRepository [com.oreilly.springdata.jpa.core.CustomerRepository]
  - Gradle Dependencies
  - JRE System Library [JVM 1.5.0 (MacOS X Default)]
  - build
  - src
  - target
  - build.gradle
  - gradle.properties
  - pom.xml

ProductRepository.java

```
 * Copyright 2012 the original author or authors.
package com.oreilly.springdata.jpa.core;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.repository.CrudRepository;

} instances.

ends CrudRepository<Product, Long> {

ning(String description, Pageable pageable);
```

*Order.java

```
/**
 * An order.
 *
 * @author Oliver Gierke
 */
@Entity
@Table(name = "Orders")
public class Order extends AbstractEntity

  @ManyToOne
  private Customer customer;
  @ManyToOne
  private Address billingAddress;
  @ManyToOne
  private Address shippingAddress;

  @OneToMany(cascade = CascadeType.ALL, or
  @JoinColumn(name = "order_id")
  private Set<LineItem> lineItems = new Ha

  // …
```

*OrderRepository.java

```
/**
 * Repository to access {@link Order}s.
 *
 * @author Oliver Gierke
 */
public interface OrderRepository extends PagingAndSortingRepository<Order, Long> {

  List<Order> findBy(Customer customer);
}
```

billingAddress
customer
lineItems
shippingAddress

Project Explorer

- jpa
  - Spring Elements
    - Beans
      - application-context.xml
      - spring-data-context.xml
    - Spring Data Repositories
      - orderRepository [com.oreilly.springdata.jpa.order.OrderRepository]
      - customerRepository [com.oreilly.springdata.jpa.core.CustomerRepository]
  - Gradle Dependencies
  - JRE System Library [JVM 1.5.0 (MacOS X Default)]
  - build
  - src
  - target
  - build.gradle
  - gradle.properties
  - pom.xml

ProductRepository.java

```
 * Copyright 2012 the original author or authors.
package com.oreilly.springdata.jpa.core;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.repository.CrudRepository;

} instances.

ends CrudRepository<Product, Long> {

ning(String description, Pageable pageable);
```

*Order.java

```
/**
 * An order.
 *
 * @author Oliver Gierke
 */
@Entity
@Table(name = "Orders")
public class Order extends AbstractEntity

  @ManyToOne
  private Customer customer;
  @ManyToOne
  private Address billingAddress;
  @ManyToOne
  private Address shippingAddress;

  @OneToMany(cascade = CascadeType.ALL, or
  @JoinColumn(name = "order_id")
  private Set<LineItem> lineItems = new Ha

  // …
```

*OrderRepository.java

```
/**
 * Repository to access {@link Order}s.
 *
 * @author Oliver Gierke
 */
public interface OrderRepository extends PagingAndSortingRepository<Order, Long> {

  List<Order> findByBillingAddress(Customer customer);
}
```

- city
- country
- street
- And
- Equals
- Exists
- In
- Is
- IsIn

Querydsl

Querydsl

```
QPerson $ = QPerson.person;
BooleanExpression left = $.lastname.contains("eth");
BooleanExpression right = $.firstname.is("Carter");

public interface QueryDslPredicateExecutor<T> {
   T findOne(Predicate predicate);
   List<T> findAll(Predicate predicate);
}

public interface PersonRepository extends Repository<Person, BigInteger>,
   QueryDslPredicateExecutor { ... }

List<Person> result = repository.findAll(left.or(right));
assertThat(result.size(), is(2));
assertThat(result, hasItems(dave, carter));
```

Wrap up

Mapping support

Templates

Repositories

Store-specific functionality

Querydsl

CDI Integration

IDE support

#184

DZone Refcardz

CONTENTS INCLUDE:

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 objects into store-specific data types.

Template APIs

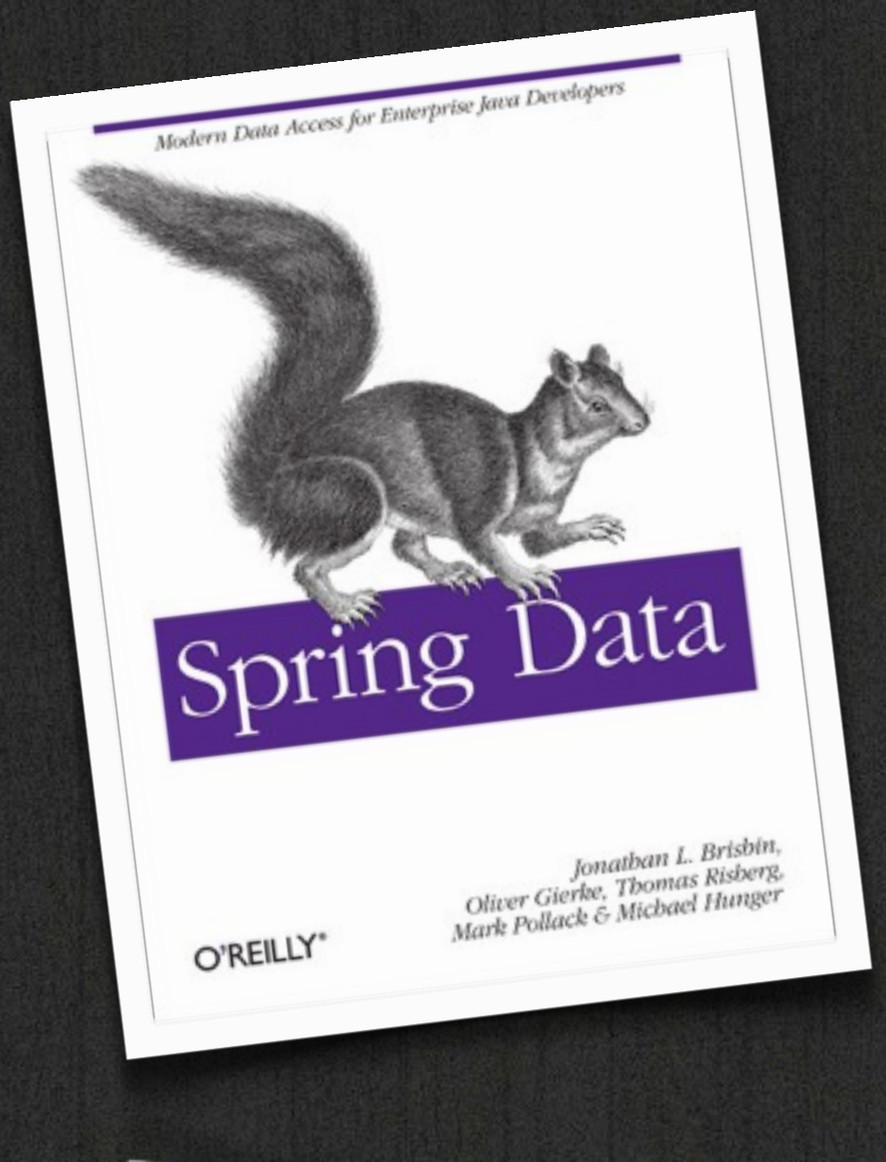
JPA

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 /> extenally 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 more advanced options through the nested <mongo:options /> element. In JavaConfig simply declare an @Bean



Spring Data

Modern Data Access For Enterprise Java

Hive

Pig

JDBC

JPA

HBase

Splunk

NoSQL

Big Data

Hadoop

Redis

Roo

Gemfire

MongoDB

Neo4j

REST exporter

Querydsl

Repositories

Questions?

Resources

- springio.org/spring-data
- github.com/spring-projects/spring-data-jpa
- github.com/spring-projects/spring-data-mongodb
- github.com/spring-projects/spring-data-neo4j
- se-radio.net/2010/07/episode-165-nosql-and-mongodb-with-dwight-merriman
- kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis
- github.com/spring-projects/spring-data-jpa-examples