Spring Data JPA

brief introduction

Mario Kaufmann

b UNIVERSITÄT BERN

Persisting users

We want to save users in a database:

- First name
- Last name
- Email address
- Address

Traditional way to connect to a database (1)

Creating the schema:

```
create table users
(
  id int,
  lastName varchar(255),
  firstName varchar(255),
  email varchar(255)
);
```

Inserting data:

```
insert into user values (5, 'John', 'Doe',
'john@doe.com');
```

Traditional way to connect to a database (2)

```
public class DBHandler {

 public void saveNewUser() {
   try {
     Class.forName("com.mysql.jdbc.Driver");
     Connection connection = DriverManager.
       getConnection("jdbc:mysql://server/db?user=us&password=pwd");
     PreparedStatement statement = connection.
       prepareStatement("insert into users values (?, ?, ?, ?)");
     statement.setLong(1, 5);
     statement.setString(2, "John");
     statement.setString(3, "Doe");
     statement.setString(4, "john@doe.com");
     statement.executeUpdate();
   } catch (ClassNotFoundException | SQLException e) {
     e.printStackTrace();
   }
 }
}
```

Load the driver

Create a connection

Create an SQL statement

Fill in the statement parameters

Execute the statement

Spring Data JPA

- Java Persistence API
- Use Java beans (special, simple classes)

```
public class User {
  private String firstName;
  private String lastName;
  private String email;
  private Address address;

  // getters and setters
}
```

```
public class Address {
  private String address;
  private int number;

  // getters and setters
}
```

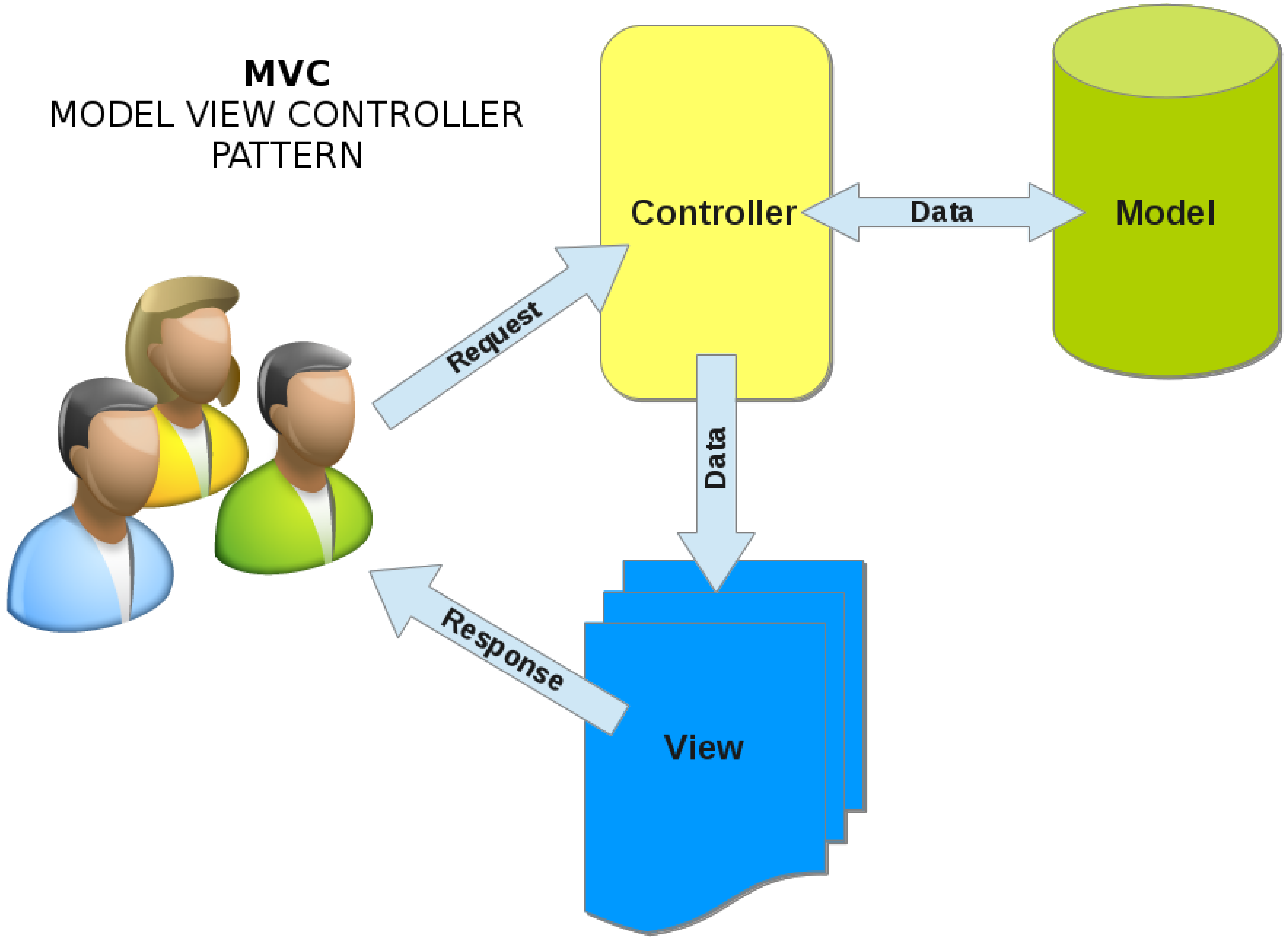
Spring Data JPA

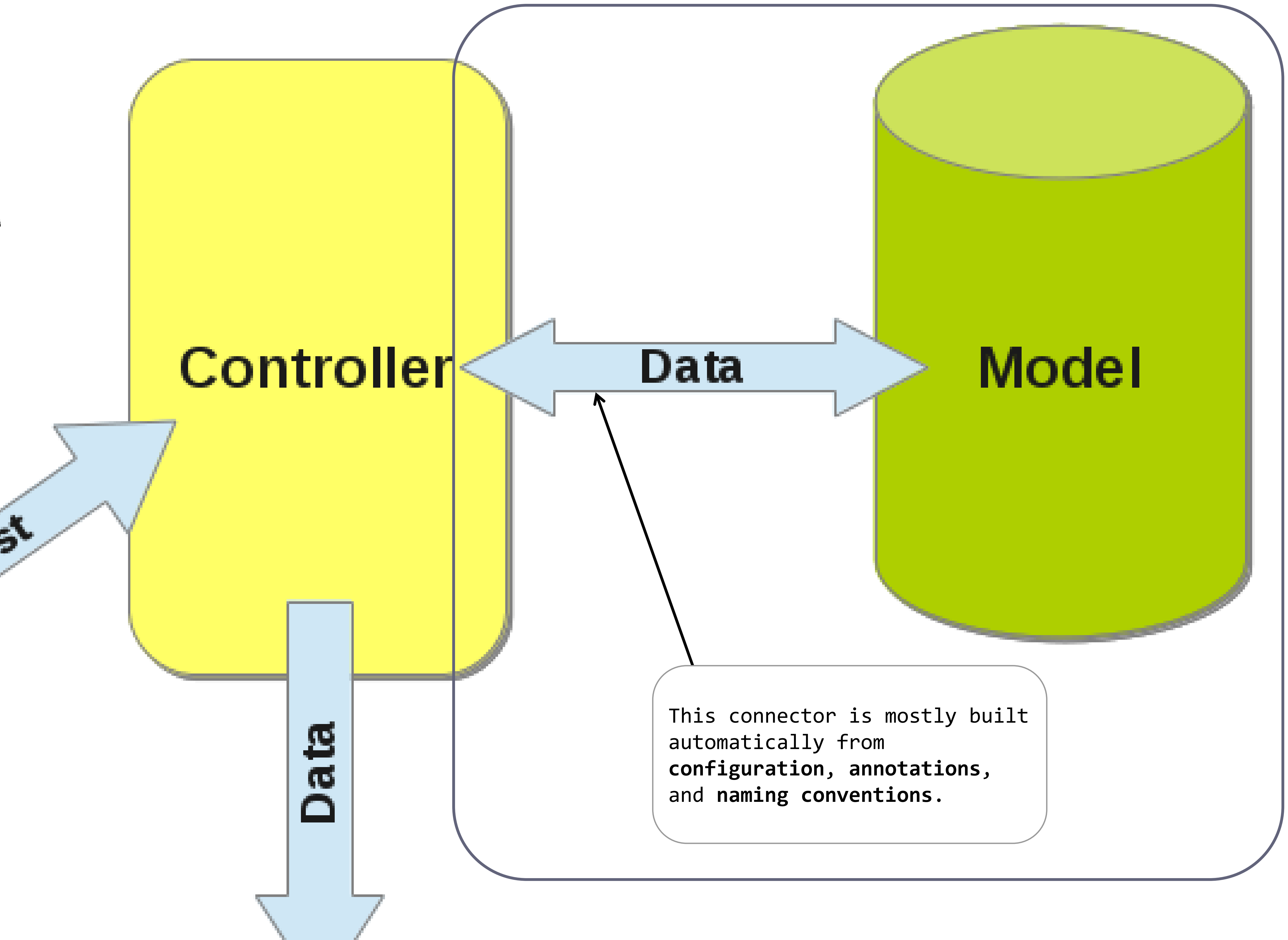
- Annotate beans

```
@Entity
public class User {
   @Id
   @GeneratedValue
   private Long id;

   private String firstName;
   private String lastName;
   private String email;

   @OneToOne
   private Address address;
   // getters and setters
}
```

```
@Entity
public class Address {
   @Id
   @GeneratedValue
   private Long id;

   private String street;
   private int number;

   // getters and setters
}
```

MVC
MODEL VIEW CONTROLLER PATTERN





This connector is mostly built automatically from **configuration, annotations,** and **naming conventions.**

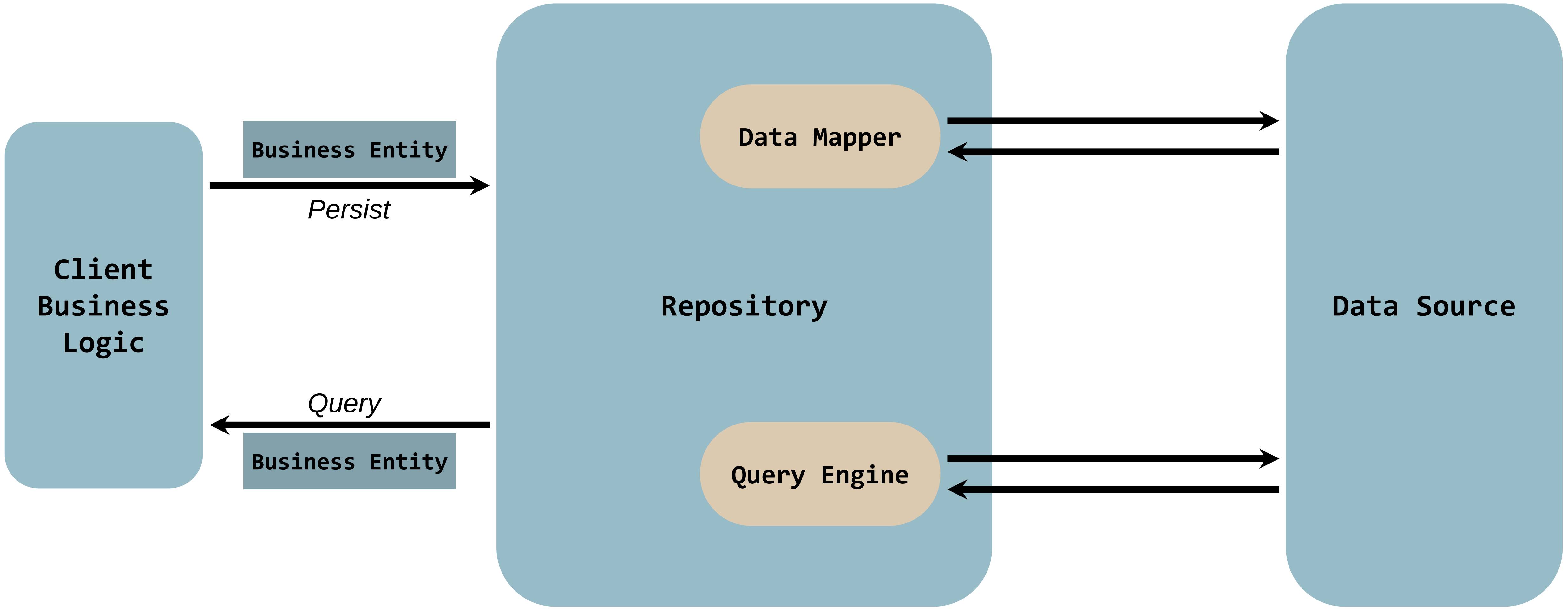


diagram from “*Patterns of Enterprise Application Architecture*”, by *Martin Fowler*

Spring Style to Connect to DB (1)

```
@Entity
public class User {
   @Id
   @GeneratedValue
   private Long id;

   private String firstName;
   private String lastName;
   private String email;

   @OneToOne
   private Address address;
   // getters and setters
}
```

```
@Entity
public class Address {
   @Id
   @GeneratedValue
   private Long id;

   private String street;
   private int number;

   // getters and setters
}
```

Business Logic

Repository

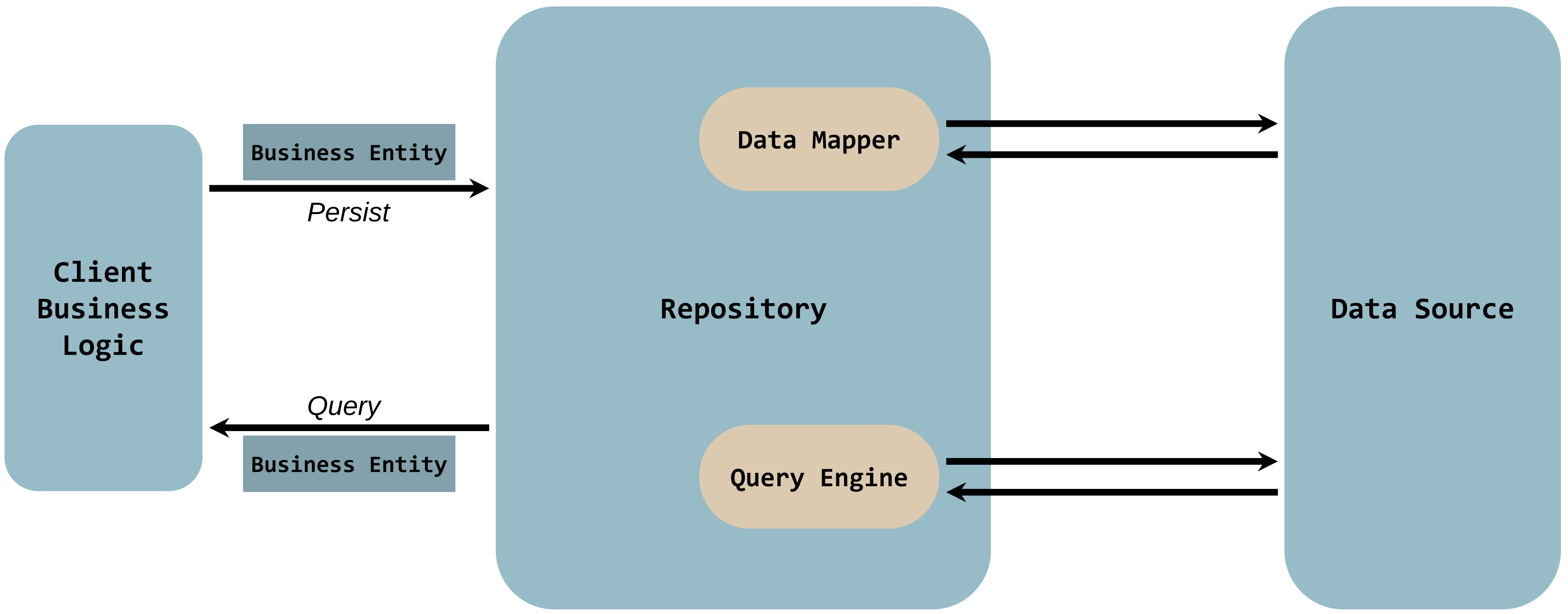
```
public interface UserDao
extends CrudRepository<User, Long>{


}
```

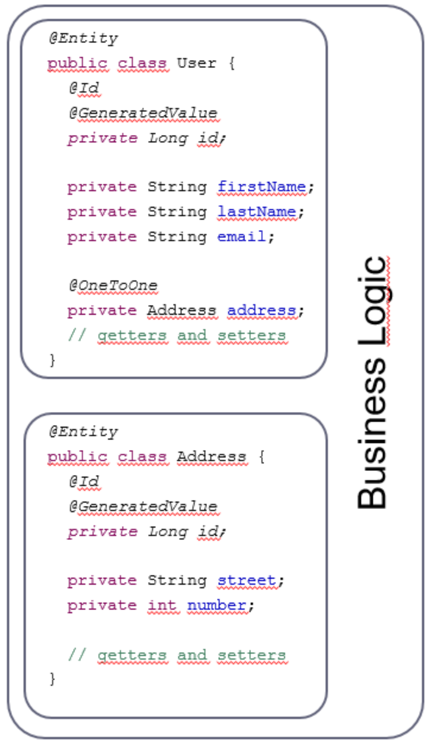
```
public interface AddressDao extends
CrudRepository<Address, Long>{


}
```

Spring Style to Connect to DB (2)

Data Source

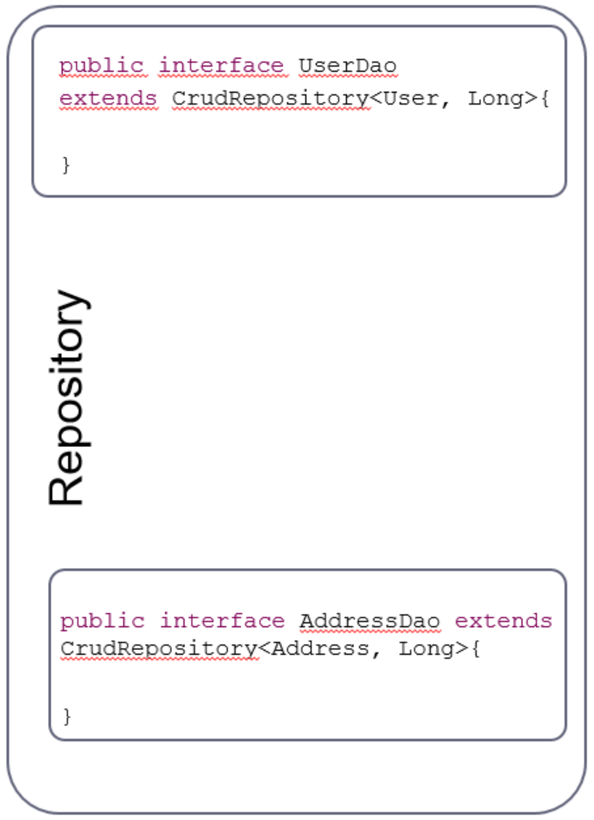
```
<bean id="mainDataSource" class="com.jolbox.bonecp.BoneCPDataSource" destroy-method="close">
   <property name="driverClass" value="com.mysql.jdbc.Driver" />
   <property name="jdbcUrl" value="jdbc:mysql://localhost/ESE?
                                              autoReconnect=true&amp;createDatabaseIfNotExist=true&amp;
                                              useUnicode=true&amp;characterEncoding=utf-8" />
   <property name="username" value="root"/>
   <property name="password" value=""/>
   <property name="idleConnectionTestPeriodInMinutes" value="60"/>
   <property name="idleMaxAgeInMinutes" value="240"/>
   <property name="maxConnectionsPerPartition" value="30"/>
   <property name="minConnectionsPerPartition" value="10"/>
   <property name="partitionCount" value="3"/>
   <property name="acquireIncrement" value="5"/>
   <property name="statementsCacheSize" value="100"/>
   <property name="releaseHelperThreads" value="3"/>
</bean>
```





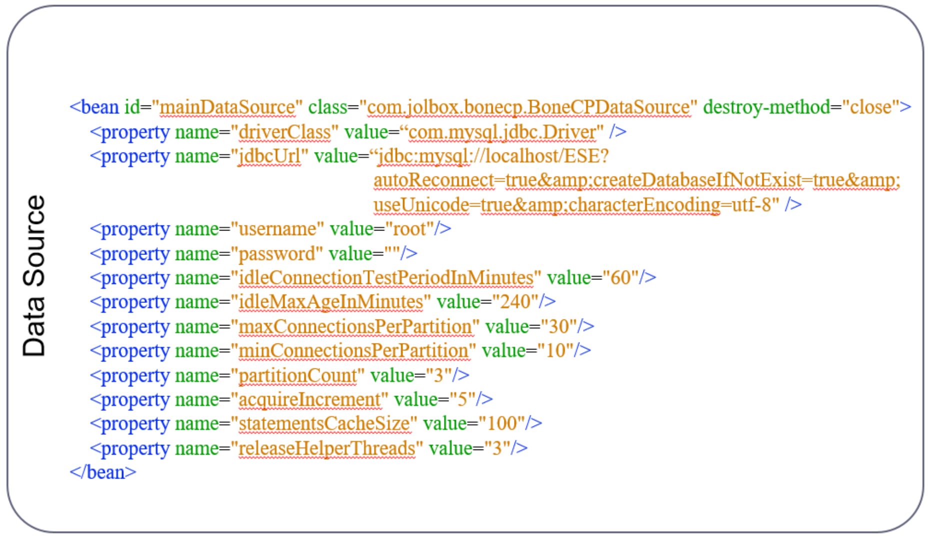
```java
@Entity
public class User {
  @Id
  @GeneratedValue
  private Long id;

  private String firstName;
  private String lastName;
  private String email;

  @OneToOne
  private Address address;
  // getters and setters
}
```

```java
@Entity
public class Address {
  @Id
  @GeneratedValue
  private Long id;

  private String street;
  private int number;

  // getters and setters
}
```



```java
public interface UserDao
extends CrudRepository<User, Long>{

}
```

```java
public interface AddressDao extends
CrudRepository<Address, Long>{

}
```



```xml
<bean id="mainDataSource" class="com.jolbox.bonecp.BoneCPDataSource" destroy-method="close">
   <property name="driverClass" value="com.mysql.jdbc.Driver" />
   <property name="jdbcUrl" value="jdbc:mysql://localhost/ESE?
                                   autoReconnect=true&amp;createDatabaseIfNotExist=true&amp;
                                   useUnicode=true&amp;characterEncoding=utf-8" />
   <property name="username" value="root"/>
   <property name="password" value=""/>
   <property name="idleConnectionTestPeriodInMinutes" value="60"/>
   <property name="idleMaxAgeInMinutes" value="240"/>
   <property name="maxConnectionsPerPartition" value="30"/>
   <property name="minConnectionsPerPartition" value="10"/>
   <property name="partitionCount" value="3"/>
   <property name="acquireIncrement" value="5"/>
   <property name="statementsCacheSize" value="100"/>
   <property name="releaseHelperThreads" value="3"/>
</bean>
```

Spring Style to Connect to DB (3)

```
@Service
public class SampleServiceImpl
implements SampleService {

  @Autowired
  UserDao userDao;
  @Autowired
  AddressDao addDao;

  @Transactional
  public int saveUser(){
    Address address = new Address();
        address.setStreet("TestStreet");

        User user = new User();
        user.setFirstName("John");
        user.setEmail(“john@doe.com”);
        user.setLastName("Doe");
        user.setAddress(address);

        address = addDao.save(address);
        user = userDao.save(user);
  }
}
```

```
public interface UserDao
extends CrudRepository<User, Long>{

}
```

Available methods:

public void **deleteAll**();
public void **delete**(Iterable<? extends User> itrbl);
public void **delete**(User t);
public void **delete**(Long id);
public long **count**();
public Iterable<User> **findAll**(Iterable<Long> itrbl);
public Iterable<User> **findAll**();
public boolean **exists**(Long id);
public User **findOne**(Long id);
public <S extends User> Iterable<S> **save**(Iterable<S> itrbl);
public <S extends User> S **save**(S s);

Extend the data access objects

```
public interface UserDao extends CrudRepository<User, Long> {

public Iterable<User> findByEmail(String email);

public Iterable<User>
findByFirstNameNotOrderByLastNameDesc(String firstName);

}
```

Query Method Keywords

Keyword	Sample
And	findByLastnameAndFirstname
Or	findByLastnameOrFirstname
Is,Equals	findByFirstname,findByFirstnameIs,findByFirstnameEquals
Between	findByStartDateBetween
LessThan	findByAgeLessThan
LessThanEqual	findByAgeLessThanEqual
GreaterThan	findByAgeGreaterThan
GreaterThanEqual	findByAgeGreaterThanEqual
After	findByStartDateAfter
Before	findByStartDateBefore
IsNull	findByAgeIsNull
IsNotNull,NotNull	findByAge(Is)NotNull
Like	findByFirstnameLike
NotLike	findByFirstnameNotLike
StartingWith	findByFirstnameStartingWith
EndingWith	findByFirstnameEndingWith
Containing	findByFirstnameContaining
OrderBy	findByAgeOrderByLastnameDesc
Not	findByLastnameNot
In	findByAgeIn(Collection<Age> ages)
NotIn	findByAgeNotIn(Collection<Age> age)
True	findByActiveTrue()
False	findByActiveFalse()
IgnoreCase	findByFirstnameIgnoreCase

https://docs.spring.io/spring-data/jpa/docs/current/reference/html/

Useful Links

- **Spring Data JPA Quick Start**
  - http://spring.io/guides/gs/accessing-data-jpa/
- **Spring Data JPA Quick Start**
  - http://spring.io/guides/gs/accessing-data-jpa/
- **Spring Data JPA Reference Documentation**
  - http://docs.spring.io/spring-data/jpa/docs/current/reference/html/
- **Spring Data JPA Tutorial**
  - http://spring.io/guides/tutorials/data/