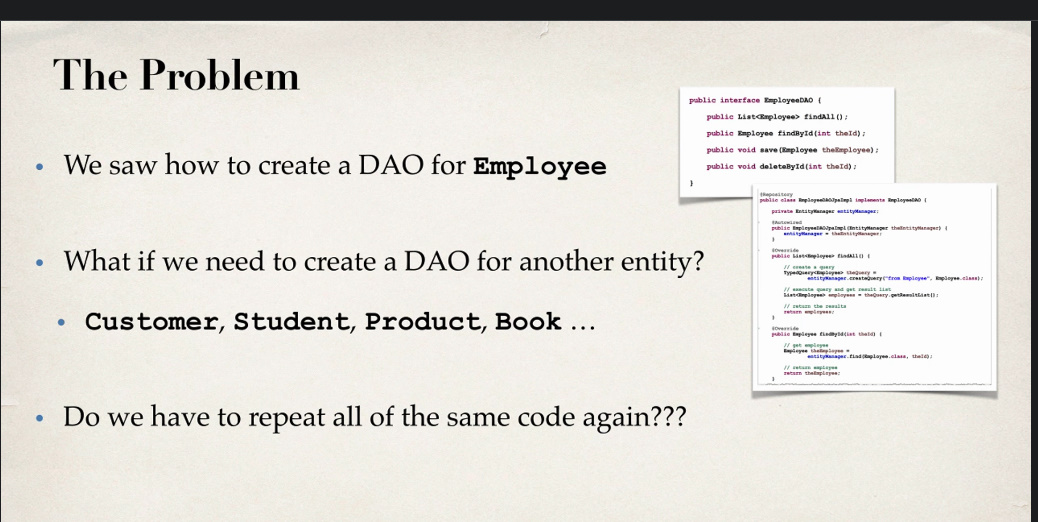
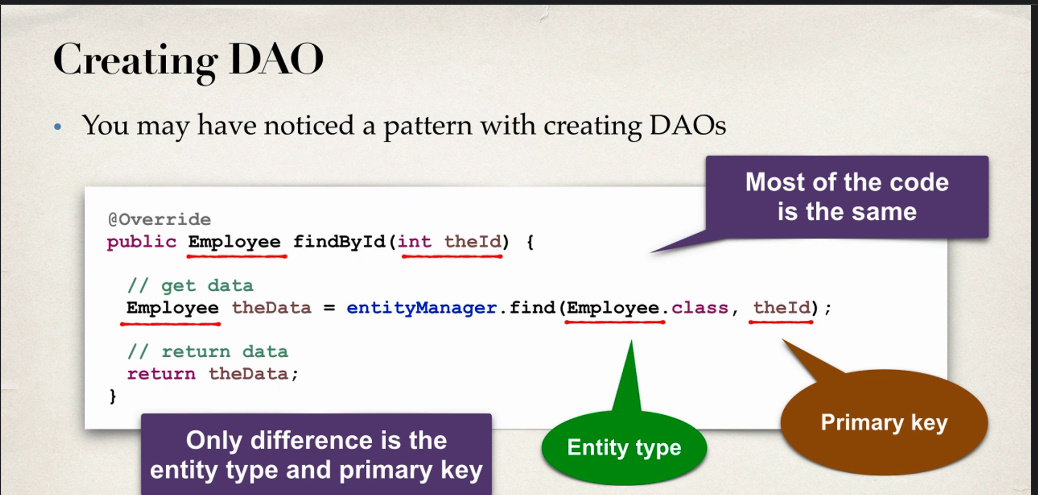
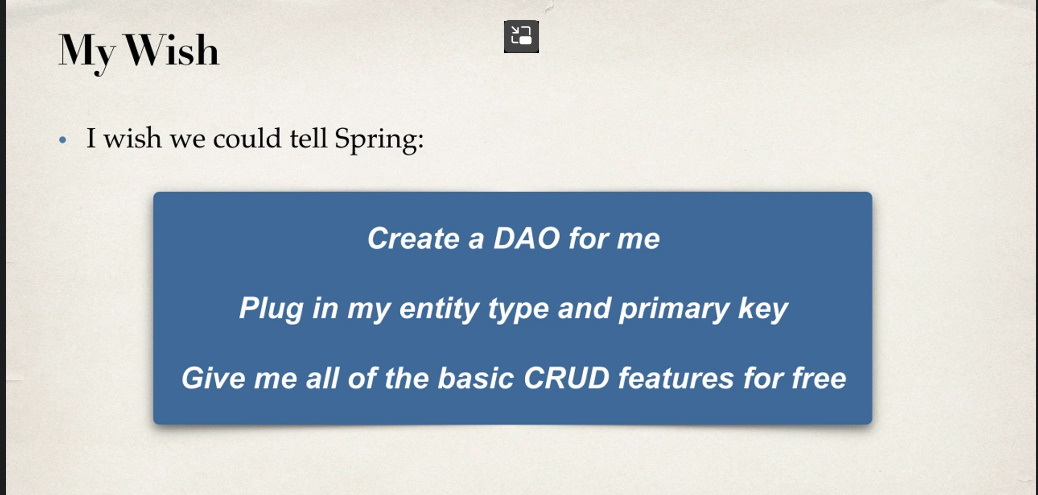
**Metoda 3 Spring Data JPA**

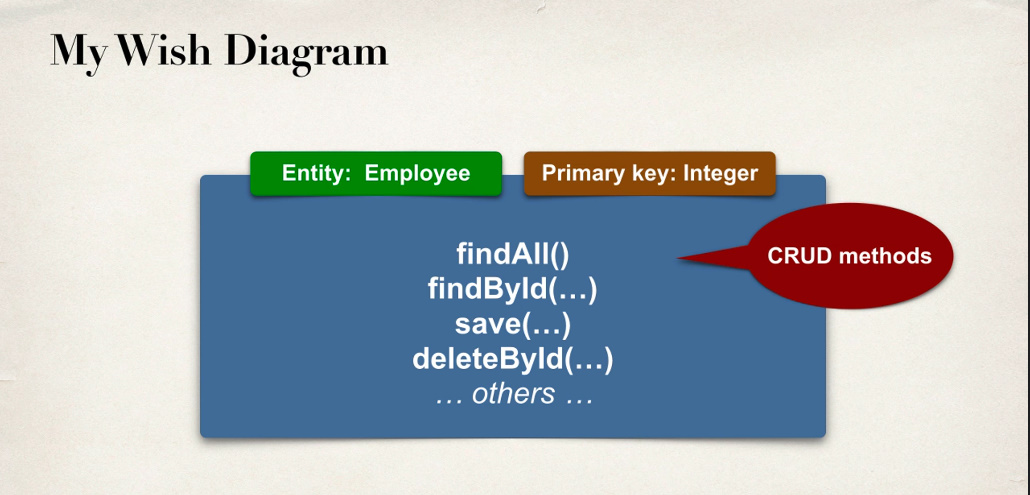


Daca mai vrem sa cream DAO pentru alte entitati, trebuie iarasi sa repetam o gramada de cod. Asta e foarte incomod.



Vedem ca cam toate DAO ar avea aceleasi metode, doar ca logic ca ar fi alt obiect, de ex nu Employee, dar Book, Customer etc.

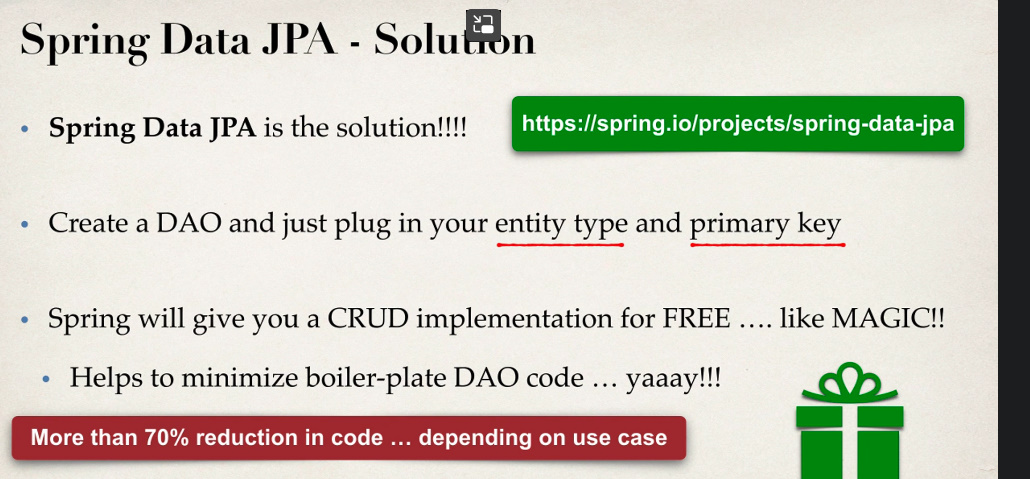
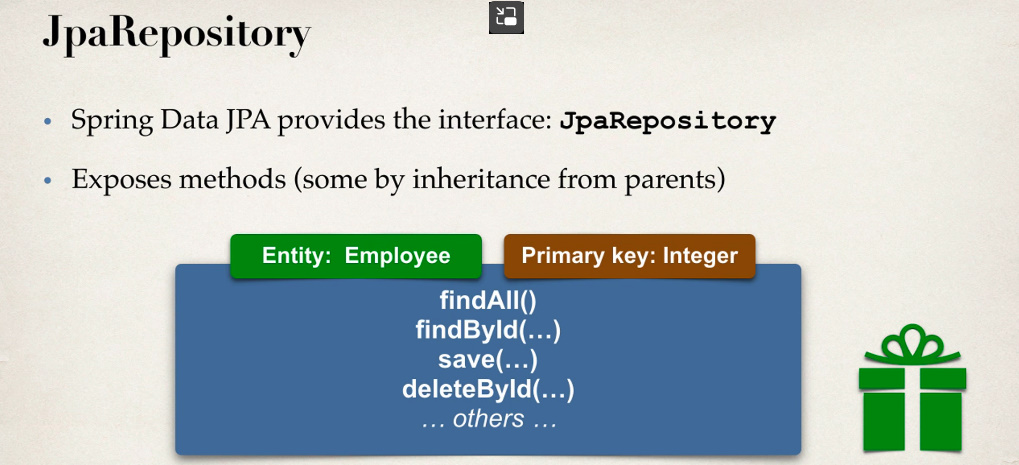




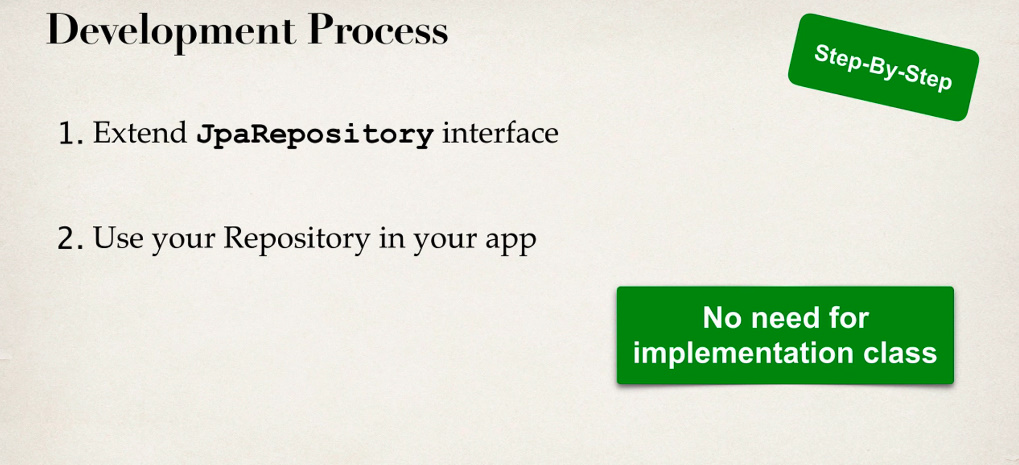
Si vrem inca sa putem folosi diferite Entity pe langa Employee, ca Book, Customer, Client etc.

**Spring Data JPA e solutia**

**Spring Data JPA**

* Spring Data JPA poate fi folosit si cu Spring si Spring Boot
* 
* Deci, Spring Data JPA ofera o interfata numita JpaRepository, care are si ea cateva metode, des utilizate.

**Pasi de Creare**

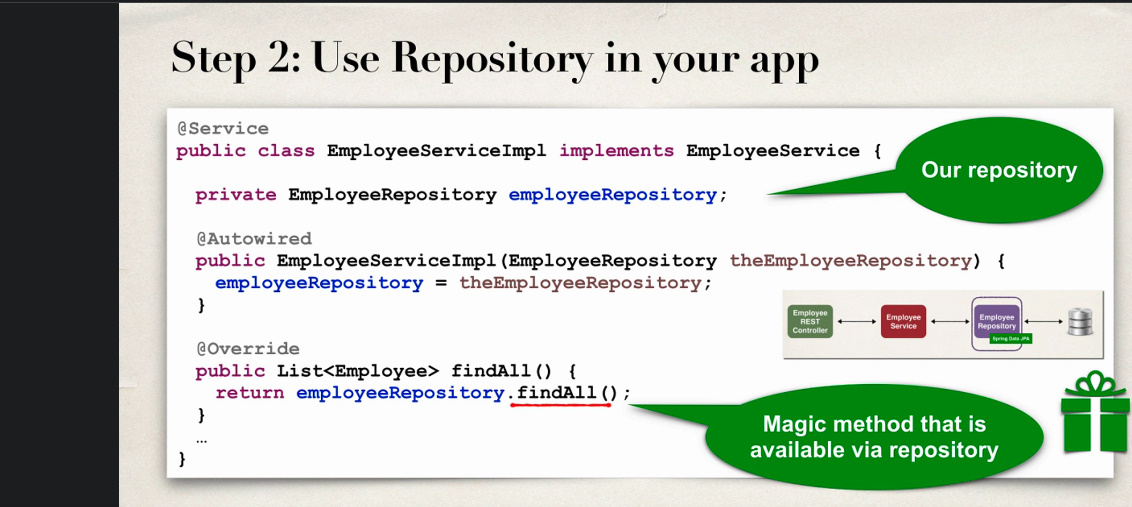
* 
* 

Deci, cream o interfata si o extindem pe JpaRepository, specificand intre <> tipul obiectului pentru care clasa data va fi DAO sau Repository, apoi tipul la primary key.

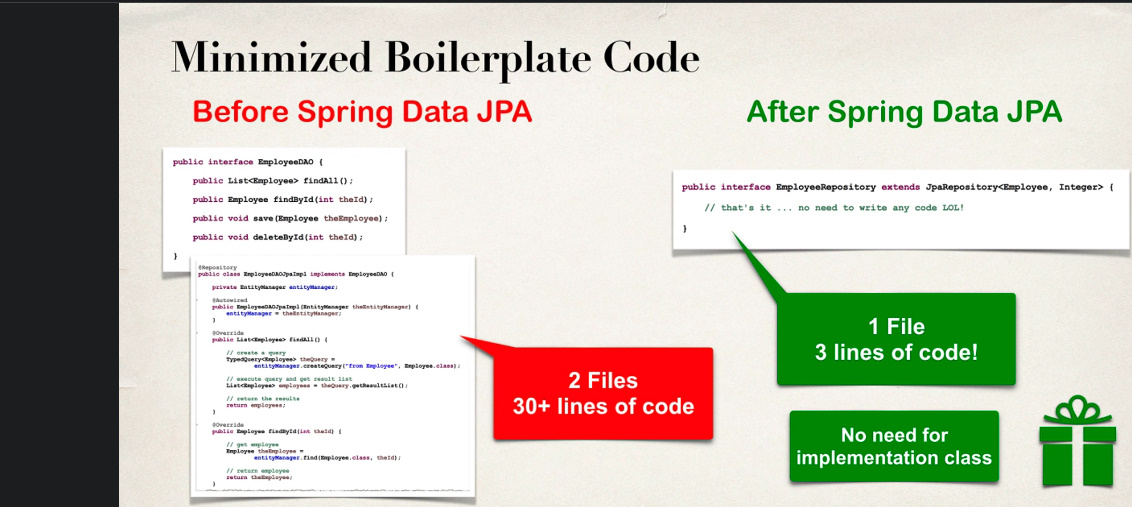
Metodele findAll(), findById() si altele sunt deja oferite de interfata JpaRepository.

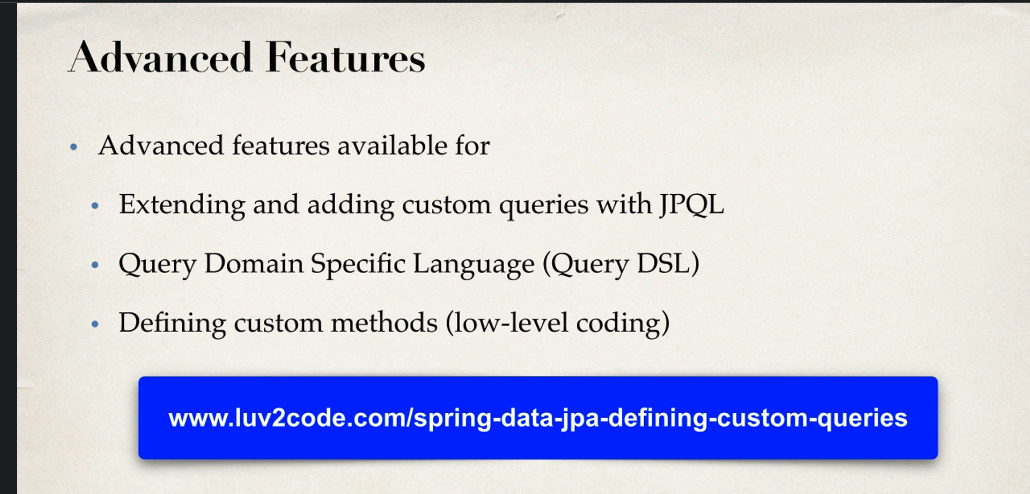
* Lista completa de metode:

[JpaRepository (Spring Data JPA Parent 3.0.1 API)](https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html)

* Daca vrem, putem adauga propriile metode default aici, dar e idee rea.
* Nu putem crea clasa, ci doar interfata, asa cum acele metode din JpaRepository sunt abstracte!!
* 

Dupa ce facem tot ce avem de facut cu Repository, in Service o folosim.





**@EnableJpaRepositories(“packageDeScanat”)**

* Anotatia data face ca sa se scaneze si inregistreze automat JpaRepository, adica inerfatele ce o extind, dar Spring o face automat daca detecteaza ca folosim Spring Data

**Code**

1. Nu avem de nevoie de nicio clasa pentru DAO, adica Repository, doar de o interfata care sa extinda interfata JpaRepository<>

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {  
  
}

Nu trebuie sa o setam ca @Repository, asa cum JpaRepository deja are aceasta anotatie prin configuratie

JpaRepositry are prin interfetele parinti metode exact asa cum am facut noi mai inainte, adica cu save, deleteById etc, deci noi nici nu trebuie macar sa le suprascriem.Ele sunt abstracte, insa la rularea programului, Spring va vedea ca unde dam @Autowired, e un obiect instanta al JpaRepository, deci va crea automat o clasa care implementeaza interfata noastra si suprascrie metodele la JpaRepository.

1. Adaugam aceasta interfata creata in Service

@Service  
public class EmployeeServiceImplem implements EmployeeService{  
  
 @Autowired  
 private EmployeeRepository employeeRepository;  
  
 @Override  
 @Transactional  
 public List<Employee> findAll() {  
 return employeeRepository.findAll();  
 }  
  
 @Override  
 @Transactional  
public Employee findEmployeeId(int id) {  
 Optional<Employee> optional = employeeRepository.findById(id);  
 Employee employee = null;  
 if(optional.isPresent())  
 employee = optional.get();  
 else   
 throw new RuntimeException("Can't find an employee with id "+id);  
   
 return employee;  
}



@Override  
 @Transactional  
 public Employee saveEmployee(Employee employee) {  
 employeeRepository.save(employee);  
  
 return employee;  
 }  
  
 @Override  
 @Transactional  
 public void deleteEmployee(int id) {  
 employeeRepository.deleteById(id);  
 }  
}

Spring se va ocupa singur de injectarea unui obiect potrivit pentru ea.

1. Stergem toate @Transactional, asa cum JpaRepository deja le are.

**Avem grija ca numele metodelor sa fie cele din JpaRepository**

**Optional<>**

Optional a fost adaugat in Java8 ca un nou pattern.

El pur si simplu ne ajuta sa verificam daca obiectul returnat e null sau nu.

* Cam multe metoda din JpaRespository returneaza Optional object, dar nu toate, de ex findAll

**Custom Code**

Daca aceste metode default nu sunt de ajuns si mai vrem sa adaugam, trebuie sa cream o noua interfata, sa o extindem intr-o clasa si sa dam @Autowired la interfata ce extinde JpaRepository si acolo in clasa facem ce vrem.

You need to create a separate interface for your custom methods:

public interface AccountRepository

extends JpaRepository<Account, Long>{ ... }

public interface AccountRepositoryCustom {

public void customMethod();

}

and provide an implementation class for that interface:

public class AccountRepositoryImpl implements AccountRepositoryCustom {

@Autowired

@Lazy

AccountRepository accountRepository; /\* Optional - if you need it \*/

public void customMethod() { ... }

}

**Custom Queries**

* Spring Data ne ofera posibilitatea de a crea custom queries.