# Spring Data Repositories

1. Understanding Spring Data repositories
2. Using a Spring Data repository

# 1. Understanding Spring Data Repositories

- Overview of Spring Data repositories
- Spring Data repository capabilities
- Paging and sorting
- Technology-specific repositories
- Domain-specific repositories

# Overview of Spring Data Repositories

- Spring Data is a data-access abstraction mechanism
  - Makes it very easy to access a wide range of data stores
  - Using a familiar "repository" pattern
  - Create / Read / Update / Delete (CRUD)

- It provides template repositories for…
  - JPA
  - MongoDB, Cassandra, CouchBase
  - Etc.

# Spring Data Repository Capabilities

- Spring Data defines a general-purpose repository interface:

```
public interface CrudRepository<T,ID> {

    long count();

    void delete(T entity);

    void deleteAll();

    void deleteAll(Iterable<T> entities);

    void deleteById(ID id);

    boolean existsById(ID id);

    Iterable<T> findAll();

    Iterable<T> findAllById(Iterable<ID> ids);

    Optional<T> findById(ID id);

    T save(T entity);

    Iterable<T> saveAll(Iterable<T> entities);
}
```

# Paging and Sorting

- Support for paging and sorting is provided via this interface:

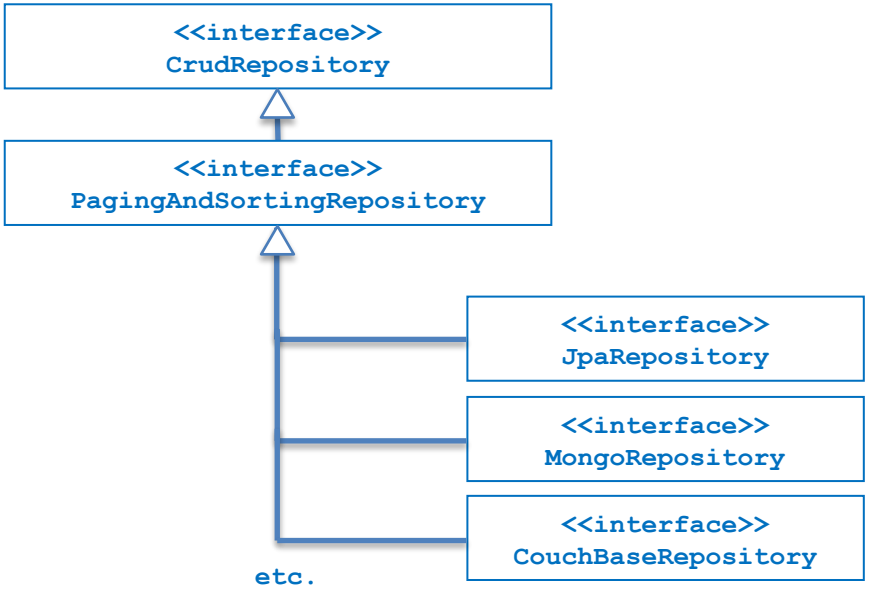
```
public interface PagingAndSortingRepository<T,ID>
       extends CrudRepository<T,ID> {

    Page<T> findAll(Pageable pageable);

    Iterable<T> findAll(Sort sort);
}
```

# Technology-Specific Repositories

- Spring Data also provides technology-specific repositories
  - Interfaces that extend `PagingAndSortingRepository`
  - Provide technology-specific extensions



# Domain-Specific Repositories

- You can define your own domain-specific interfaces
  - Extend `CrudRepository` (or sub-interface)
  - Specify the entity type and the PK type

- You can define specific query methods for your entities
  - Spring Data reflects on method names to create queries
  - You can provide an explicit query string for complex queries

- See next section for an example…

# 2. Using a Spring Data Repository

- Overview
- Defining a repository
- Locating Spring Data repositories
- Using a Spring Data repository



Demo package: demo.springdatarepositories

# Overview

- In this section we'll see how to access a relational database by using a Spring Data repository

- Note the following key points in the demo first:
  - `pom.xml`
  - `application.properties`
  - `Employee.java`
  - `SeedDb.java`

# Defining a Repository

- Here's an example of a domain-specific repository:

```
public interface EmployeeRepository extends CrudRepository<Employee, Long> {

    List<Employee> findByRegion(String region);

    @Query("select e from Employee e where e.dosh >= ?1 and e.dosh <= ?2")
    List<Employee> findInSalaryRange(double from, double to);

    Page<Employee> findByDoshGreaterThan(double salary, Pageable pageable);

}
```

**EmployeeRepository.java**

- Note:
  - Entity type is `Employee`, PK type is `Long`
  - Also, we've defined some custom queries

# Locating Spring Data Repositories

- A Spring Boot application scans for Spring Data JPA repository interfaces when it starts
  - It looks in the main application class package, plus sub-packages

- You can tell it to look elsewhere, if you like
  - Via `@EnableJpaRepositories`

```
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
…

@SpringBootApplication
@EnableJpaRepositories({"repopackage1", "repopackage2"})
public class Application {
    …
}
```

# Using a Spring Data Repository (1 of 2)

- Let's see how to use some standard repository methods:

```
@Component
public class EmployeeService {

    @Autowired
    private EmployeeRepository repository;

    public void useStandardRepoMethods() {

        // Insert an employee.
        Employee newEmp = new Employee(-1, "Simon Peter", 10000, "Israel");
        newEmp = repository.save(newEmp);
        System.out.printf("Inserted employee, id %d\n", newEmp.getEmployeeId());

        // Get count of all employees.
        System.out.printf("There are now %d employees\n",repository.count());

        // Get all employees.
        displayEmployees("All employees: ",repository.findAll());
    }
    …
}
```

EmployeeService.java

# Using a Spring Data Repository (2 of 2)

- Let's see how to use our custom queries in the repository:

```
@Component
public class EmployeeService {

    @Autowired
    private EmployeeRepository repository;

    public void useCustomQueryMethods() {

        // Get all employees by region.
        displayEmployees("All employees in London: ",repository.findByRegion("London"));

        // Get employees by salary range.
        List<Employee> emps = repository.findInSalaryRange(20000,  50000);
        displayEmployees("Employees earning 20k to 50k: ", emps);

        // Get a page of employees.
        Pageable pageable = PageRequest.of(1, 3, Direction.DESC, "dosh");
        Page<Employee> page = repository.findByDoshGreaterThan(50000, pageable);
        displayEmployees("Page 1 of employees more than 50k: ",page.getContent());
    }
}
```

EmployeeService.java

# Summary

- Understanding Spring Data repositories
- Using a Spring Data repository

# Exercise

- We've seen how to define a custom "select" method
  - Annotate a method with `@Query`
  - Specify a "select" JPQL string
- It's also possible to define a custom "modifying" method
  - Annotate with `@Query`, `@Modifying`, `@Transactional`
  - Specify an "insert", "update", or "delete" JPQL string

```
public interface EmployeeRepository extends CrudRepository<Employee, Long> {

    @Modifying(clearAutomatically=true)
    @Transactional
    @Query("delete from Employee e where e.dosh >= ?1 and e.dosh <= ?2")
    int deleteInSalaryRange(double from, double to);
    …
}
```