Aspect-Oriented Programming (AOP)

Aspect-Oriented Programming (AOP) complements Object-Oriented Programming (OOP) by providing another way of thinking about program structure. The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the *aspect*. Aspects enable the modularization of concerns such as transaction management that cut across multiple types and objects. (Such concerns are often termed *crosscutting* concerns in AOP literature.)

One of the key components of Spring is the *AOP framework*. While the Spring IoC container does not depend on AOP, meaning you do not need to use AOP if you don’t want to, AOP complements Spring IoC to provide a very capable middleware solution.

- ***Aspect***: a modularization of a concern that cuts across multiple classes. Transaction management is a good example of a crosscutting concern in enterprise Java applications. In Spring AOP, aspects are implemented using regular classes (the schema-based approach)
- ***Join point***: a point during the execution of a program, such as the execution of a method or the handling of an exception. In Spring AOP, a join point *always* represents a method execution.
- ***Advice***: action taken by an aspect at a particular join point. Different types of advice include "around," "before" and "after" advice. (Advice types are discussed below.) Many AOP frameworks, including Spring, model an advice as an *interceptor*, maintaining a chain of interceptors *around* the join point.
- ***Pointcut***: a predicate that matches join points. Advice is associated with a pointcut expression and runs at any join point matched by the pointcut (for example, the execution of a method with a certain name). The concept of join points as matched by pointcut expressions is central to AOP, and Spring uses the AspectJ pointcut expression language by default.

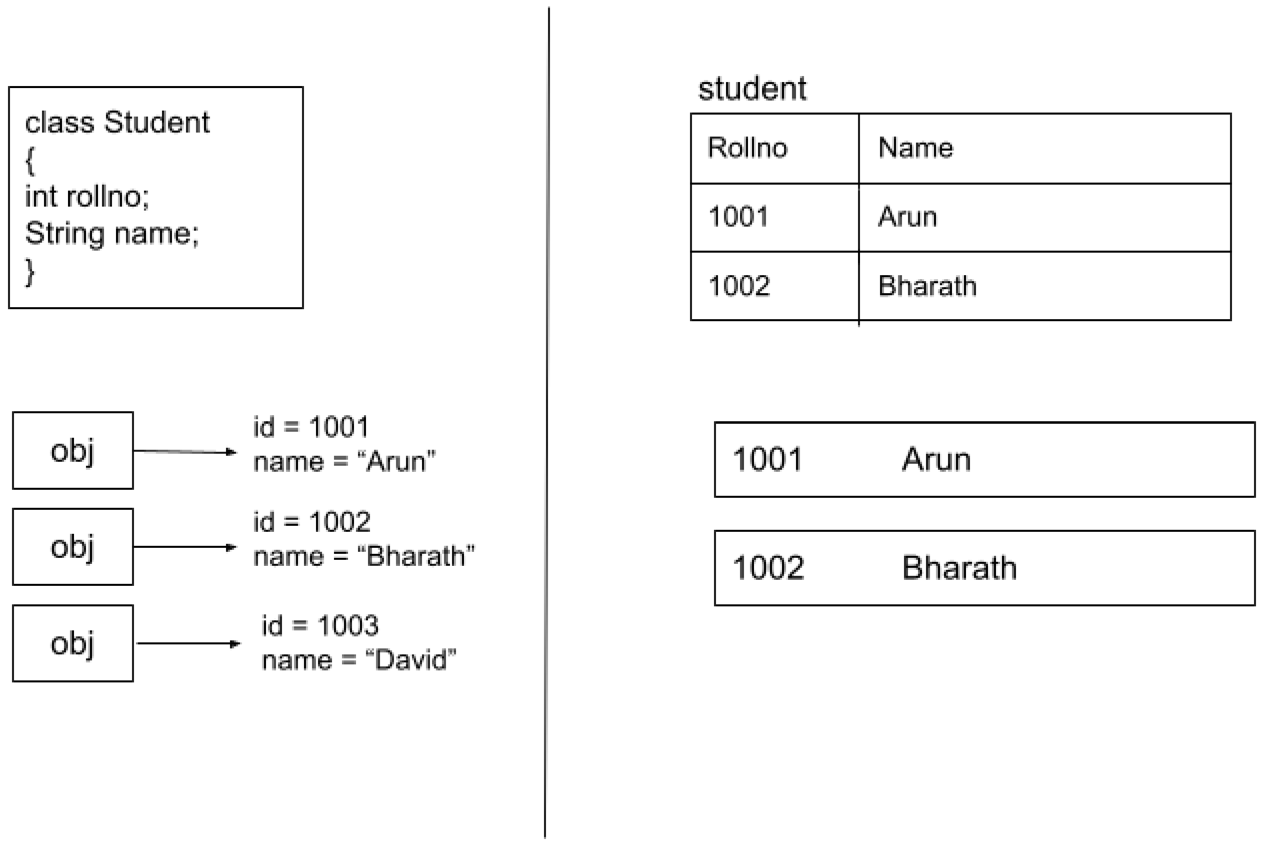
Types of advice:

- *Before advice*: Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception).
- *After returning advice*: Advice to be executed after a join point completes normally: for example, if a method returns without throwing an exception.
- *After throwing advice*: Advice to be executed if a method exits by throwing an exception.
- *After (finally) advice*: Advice to be executed regardless of the means by which a join point exits (normal or exceptional return).

ORM

Spring-ORM is a technique or a Design Pattern used to access a relational database from an object-oriented language. ORM (Object Relation Mapping) covers many persistence technologies. They are as follows:

- **JPA(Java Persistence API)**: It is mainly used to persist data between Java objects and relational databases. It acts as a bridge between object-oriented domain models and relational database systems.
- **JDO(Java Data Objects)**: It is one of the standard ways to access persistent data in databases, by using plain old Java objects (POJO) to represent persistent data.
- **Hibernate** – It is a Java framework that simplifies the development of Java applications to interact with the database.


```
class Student
{
int rollno;
String name;
}
```

student

Rollno	Name
1001	Arun
1002	Bharath

Advantages

- **Less coding is required**: By the help of Spring framework, you don't need to write extra codes before and after the actual database logic such as getting the connection, starting transaction, committing transaction, closing connection etc.
- **Easy to test**: Spring's IoC approach makes it easy to test the application.
- **Better exception handling**: Spring framework provides its own API for exception handling with ORM framework.

Spring Data JPA

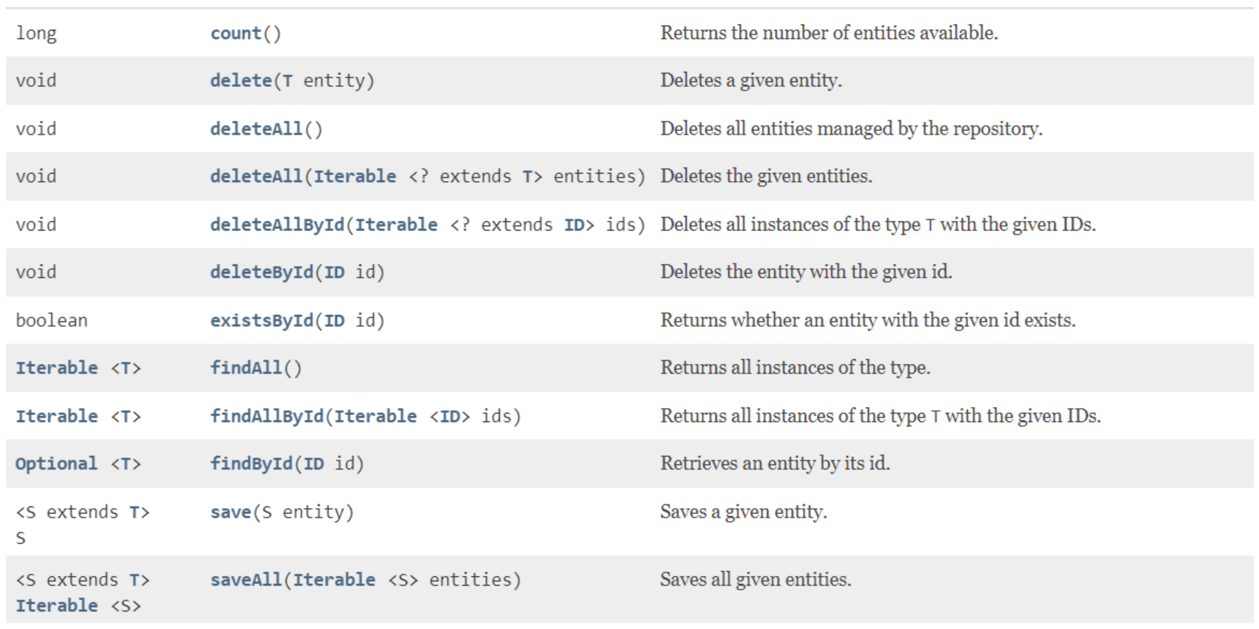
Spring Data JPA, part of the larger Spring Data family, makes it easy to easily implement JPA based repositories. This module deals with enhanced support for JPA based data access layers. It makes it easier to build Spring-powered applications that use data access technologies.

Implementing a data access layer of an application has been cumbersome for quite a while. Too much boilerplate code has to be written to execute simple queries as well as perform pagination, and auditing. Spring Data JPA aims to significantly improve the implementation of data access layers by reducing the effort to the amount that's actually needed. As a developer you write your repository interfaces, including custom finder methods, and Spring will provide the implementation automatically.

CrudRepository

There is an interface available in Spring Boot named as CrudRepository that contains methods for CRUD operations. It provides generic Crud operation on a repository. It is defined in the package ***org.springframework.data.repository*** and It extends the Spring Data **Repository** interface. If someone wants to use CrudRepository in the spring boot application he/she has to create an interface and extend the CrudRepository interface.

Methods

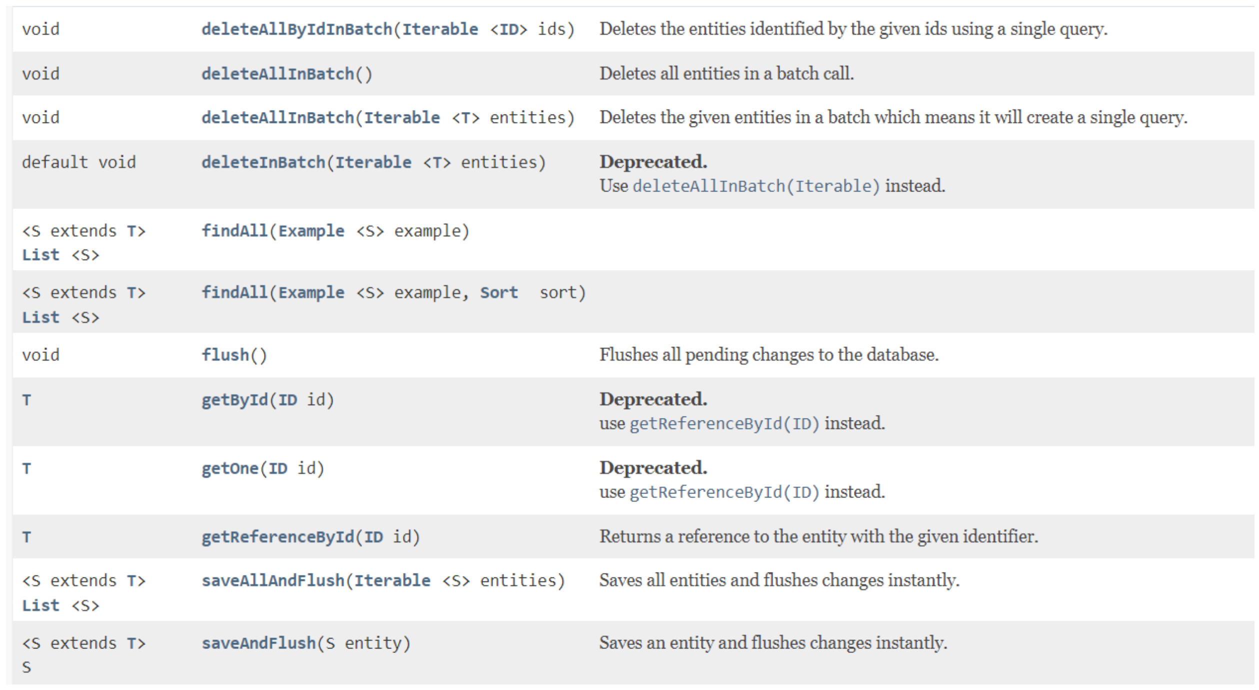


`long`	`count()`	Returns the number of entities available.
`void`	`delete(T entity)`	Deletes a given entity.
`void`	`deleteAll()`	Deletes all entities managed by the repository.
`void`	`deleteAll(Iterable <? extends T> entities)`	Deletes the given entities.
`void`	`deleteAllById(Iterable <? extends ID> ids)`	Deletes all instances of the type `T` with the given IDs.
`void`	`deleteById(ID id)`	Deletes the entity with the given id.
`boolean`	`existsById(ID id)`	Returns whether an entity with the given id exists.
`Iterable <T>`	`findAll()`	Returns all instances of the type.
`Iterable <T>`	`findAllById(Iterable <ID> ids)`	Returns all instances of the type `T` with the given IDs.
`Optional <T>`	`findById(ID id)`	Retrieves an entity by its id.
`<S extends T> S`	`save(S entity)`	Saves a given entity.
`<S extends T> Iterable <S>`	`saveAll(Iterable <S> entities)`	Saves all given entities.

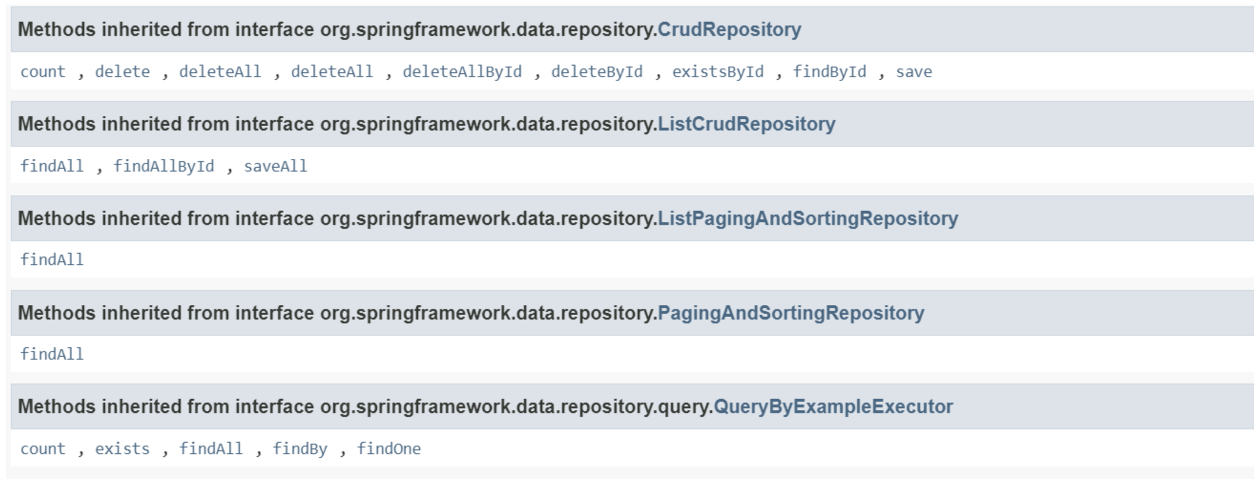
JpaRepository

JpaRepository is a **JPA (Java Persistence API)** specific extension of Repository. It contains the full API of **CrudRepository and PagingAndSortingRepository**. So it contains API for basic CRUD operations and also API for pagination and sorting.

Methods



void	`deleteAllByIdInBatch(Iterable <ID> ids)`	Deletes the entities identified by the given ids using a single query.
void	`deleteAllInBatch()`	Deletes all entities in a batch call.
void	`deleteAllInBatch(Iterable <T> entities)`	Deletes the given entities in a batch which means it will create a single query.
default void	`deleteInBatch(Iterable <T> entities)`	**Deprecated.** Use `deleteAllInBatch(Iterable)` instead.
<S extends T> List <S>	`findAll(Example <S> example)`	
<S extends T> List <S>	`findAll(Example <S> example, Sort sort)`	
void	`flush()`	Flushes all pending changes to the database.
T	`getById(ID id)`	**Deprecated.** use `getReferenceById(ID)` instead.
T	`getOne(ID id)`	**Deprecated.** use `getReferenceById(ID)` instead.
T	`getReferenceById(ID id)`	Returns a reference to the entity with the given identifier.
<S extends T> List <S>	`saveAllAndFlush(Iterable <S> entities)`	Saves all entities and flushes changes instantly.
<S extends T> S	`saveAndFlush(S entity)`	Saves an entity and flushes changes instantly.



Methods inherited from interface org.springframework.data.repository.CrudRepository

`count , delete , deleteAll , deleteAll , deleteAllById , deleteById , existsById , findById , save`

Methods inherited from interface org.springframework.data.repository.ListCrudRepository

`findAll , findAllById , saveAll`

Methods inherited from interface org.springframework.data.repository.ListPagingAndSortingRepository

`findAll`

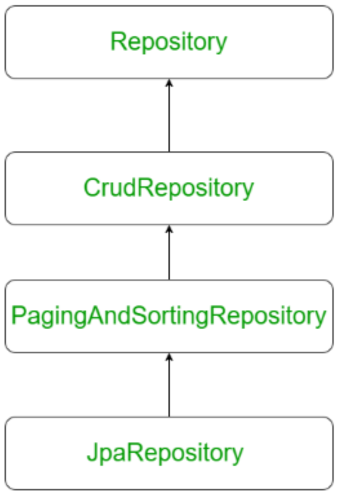
Methods inherited from interface org.springframework.data.repository.PagingAndSortingRepository

`findAll`

Methods inherited from interface org.springframework.data.repository.query.QueryByExampleExecutor

`count , exists , findAll , findBy , findOne`

Spring Data Repository Interface



CrudRepository	JpaRepository
It is a base interface and extends Repository Interface.	It extends PagingAndSortingRepository that extends CrudRepository.
It contains methods for CRUD operations. For example save(), saveAll(), findById(), findAll(), etc.	It contains the full API of CrudRepository and PagingAndSortingRepository. For example, it contains flush(), saveAndFlush(), saveAllAndFlush(), deleteInBatch(), etc along with the methods that are available in CrudRepository.
It doesn't provide methods for implementing pagination and sorting	It provides all the methods for which are useful for implementing pagination.
It works as a marker interface.	It extends both CrudRepository and PagingAndSortingRepository.
To perform CRUD operations, define repository extending CrudRepository.	To perform CRUD as well as batch operations, define repository extends JpaRepository.
Syntax: public interface CrudRepository<T, ID> extends Repository<T, ID>	**Syntax:** public interface JpaRepository<T,ID> extends PagingAndSortingRepository<T,ID>, QueryByExampleExecutor<T>