The internal code of save(-) method

```
=====================================
@Transactional
    @Override
    public <S extends T> S save(S entity) {

        Assert.notNull(entity, "Entity must not be null");  //log message

        if (entityInformation.isNew(entity)) {
            em.persist(entity);         ------> save object operation (inserting record)
            return entity;
        } else {
            return em.merge(entity);    ------> update object operation (updating record)
        }
```

EntityManager obj (represents underlying ORM(hibernate))

=>@Transactional performs commit operation on DB s/w if no exception is raised in the method execution otherwise it will roll back data changes of DB s/w if the exception is raised

note: em.persist(-) internally uses hibernate api to perform save object operation (record insertion)
em.merge(-) internally uses hibernate api to perform update object operation (record updation)

note: The internal hibernate api uses JDBC Code + SQL Queries to complete the persistence operations on DB table records

In Layered Apps

Entity class ----------> on 1 per db table
Repository Interface ------> on 1 per table
Service class ----------> on 1 per module
controller class --------> on 1 per project
Runner classes ------> as needed

=> In CrudRepository there is no separate method for update object operation hence the save(-) method itself can be used for both save object operation or update object operation.

=>EntityManager object is JPA object, that encapsulates entire underlying ORM s/w.. (like hibernate) and also provides api to perform object based persistence operations. (CURD Operations)

Flow of execution

```
//IDoctorRepo.java
package com.nt.repository;
import org.springframework.data.repository.CrudRepository;
import com.nt.entity.Doctor;
                                                      entity class   id property type
public interface IDoctorRepo extends CrudRepository<Doctor, Integer> {

}
```

InMemory Proxy class as the Impl class of our Custom Repository Interface (IDoctorRepo)
given by JDK/CGLIB Libraries (g --- ii) (Dynamically generated InMemory Proxy class)

```
@Repository
public class Proxy2 implements IDoctorRepo ,.....{
    @Autowired
    private EntityManager em;
                (AutoConfiguration)

    @Transactional
        @Override                        (r)
        public <S extends T> S save(S entity) {

            Assert.notNull(entity, "Entity must not be null");

            if (entityInformation.isNew(entity)) {          ses.persist( )
                em.persist(entity);  (s) ------> internally uses ses.save(-) of
                return entity;  (u)                hibernate api to save the object
            } else {
                return em.merge(entity);  --->internally uses ses.merge(-) of
            }                                    hibernate api to update the object
        }

    ...
    ...
    ... other 11 methods are implemented
    ... using EntityManager object and hibernate api support
    ...
}
```

=>The Jpa's EntityManager object will comes as spring bean through AutoConfiguration based on the spring data jpa starter we add.
=>This EntityManager object also holds DataSource obj internally which also came through AutoConfiguration

from hibernate api (hibernate jar files)

```
public class SessionImpl implements Session,EntityManager{

    ...
    ...
    @Override
    public void persist(Object object) throws HibernateException {
 (t)    checkOpen();
        firePersist( new PersistEvent( null, object, this ) );
    }

}
```

(t->(t)) internally generates JDBC code + INSERT/UPDATE SQL Query to insert the record in db table

=> DataSource obj is injected to EntityManager obj
=> EntityManager object is injected Dynamically Generated InMemory Proxy class obj

JPA api internally uses hibernate api

note: EntityManager obj is the object of a class given by underlying ORM s/w like hibernate implementing jakarta.jpa.EntityManager(I) directly or indirectly

=>To open the source code of class whose name is not typed in the Editors of Eclipse IDE use ctrl+Shift+T key

```
//IDoctorService.java
package com.nt.service;
import com.nt.entity.Doctor;

public interface IDoctorService {
    public String registerDoctor(Doctor doctor);
}
```

```
//DoctorMgmtServiceImpl.java     (service impl class)
package com.nt.service;

@Service("doctorService")
public class DoctorMgmtServiceImpl implements IDoctorService {
        @Autowired
    private IDoctorRepo doctorRepo;

     @Override             (p)
     public String registerDoctor(Doctor doctor) {
     (v)    Doctor doc=doctorRepo.save(doctor);  (q)
         return "Doctor obj is saved with id value ::"+doc.getDocId(); (w)
     }
```

application.properties

```
#jdbc properties (for oracle)
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=tiger
spring.datasource.hikari.maximum-pool-size=100
spring.datasource.hikari.minimum-idle=10
spring.datasource.hikari.keepalive-time=100000
#jpa -hibernate entries
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

(e) reads the application.properties file content to Environment object

(f) @EnableAutoConfiguration of @SBA makes the following classes as spring beans based on jar files that are added through AutoConfiguration process. (this also uses the inputs from application.properties)
(i) HikariDataSource pointing to oracle DB (like con pool)
(ii) EntityManager encapsulating the activated HB's SessionFactory, Session objects
iii) multiple other classes become spring beans

uses application.properties file data through Environment object

Client App

```
                  (d) recognizes @SBA                      (a) run the Application
@SpringBootApplication
public class BootDataJpaProj1CrudRepositoryApplication {

                 (b)
    public static void main(String[] args) {
        //get IOC container                 (c) -> boot strapping of spring boot App
  (k) gives IOC container
        ApplicationContext ctx=SpringApplication.run(BootDataJpaProj1CrudRepositoryApplication.class, args);
        //get Service class obj      (l)                    (d)
        IDoctorService service=ctx.getBean("doctorService",IDoctorService.class);
                     (m)
        try {
            // create Doctor class object
            Doctor doctor=new Doctor();
            doctor.setDocName("rajesh"); doctor.setSpecialization("MD-Cardio"); doctor.setIncome(9000000.0);
            //invoke the b.method
      (n)   String resultMsg= service.registerDoctor(doctor);  (o)
            System.out.println(resultMsg);  (y)
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }//main    (z)  app's execution is completed
}//class
```

(c)-> (i) :: IOC container creation (inside SpringApplication.run(-) method)
(d)-> (f) : takes the given main class as configuration class becoz of @Configuration (inside of @SBA)
(e)-> (f) -> Loads the main class and creates the object makes it as the spring bean

(g) @ComponentScan of @SBA detects /scans the following things in the current app's pkg and sub pkgs of "com.nt" (current pkg)
(i) DoctorMgmtServiceImpl as spring bean becoz of @Service
(ii) IDoctorRepo as the custom Repository interface extending from Repository(I) (marker interface) directly or indirectly.. In this process the spring data jpa generates InMemory proxy class as shown above with @Repository annotation having jpa persistence logics for all the CrudRepository methods (12 methods)

(h) IOC container performs the pre-instantiation of singleton scope spring beans in that process the following spring bean class objs will be created
=>HikariDataSource obj | Becoz of @AutoConfiguration
=>EntityManager obj |
=>DoctorServiceImpl obj | Becoz of Stereo type annotations
=>Proxy2 class (InMemory proxy class) |
and etc..

(i) @Autowired perform the necessary injections
=> EntityManager object with HikariDataSource object
=> Proxy2 class obj with EntityManager object (InMemory Proxy class obj with EntityManager obj)
=> DoctorMgmtServiceImpl class obj with Proxy2 class obj(Repository obj) (service impl class obj)
....
..
..

(j) all singleton scope spring beans will be kept in the internal cache of IOC container

Programmer
⇩
spring data jpa api (repo.save(-))
⇩
jpa api (em.persist(-))
⇩
hibernate api (ses.save(-))
⇩
jdbc api (ps.executeUpdate(-))
⇩
generates SQL Query (INSERT SQL Query)
⇩
DB Operation (Record insert operation)

repo: Repository obj
ses : HB's Session obj
em : Entity Manager obj
ps : PreparedStatement obj

(internal cache of the IOC container)

doctorService (tm)	DoctorServiceImpl class obj ref
proxy2	Proxy2 class obj ref
HikariDataSource	HikariDataSource obj ref
...	....
keys (bean ids)	values (spring bean class obj refs)

To see the source code of any class/interface/enum/annotation randomly whose jar file is added eclipse projects
ctrl+shift+T -> type class name ---> open source code

=> To see the source code of certain class/interface/enum/annotation that is used in the eclipse editors
->use F3 key

The internal code of save(-) method

@Transactional

@Override

public <S extends T> S save(S entity) {

=>@Transactional performs commit operation

on DB s/w if no exception is raised in b.method execution otherwise it will rollback data changes of DB s/w if the exeption is raised

Assert.notNull(entity, "Entity must not be null"); //og message

if (entityInformation.isNew(entity)) { em.persist(entity);

return entity;

} else {

EntityManager obj return em.merge(entity);

(represents

underlying ORM(hibernate))

}

save object operation (inserting record)

update object opeation (updating record)

=> In CrudRepository there is no seperate method for update object operation becoz the save(-) method itself can be used for both save object operation or update object operation.

is

=>EntityManager object JPA object that encapulates entire

Based

underlying

CrudRepository<T, ID>

•Asave(S) <S extends T> : S

• AsaveAll(Iterable<S>) <S extends T> : Iterable<S>

⚫ findByld(ID): Optional<T>

⚫ existsByld(ID): boolean

A findAll(): Iterable<T>

A findAllByld(Iterable<ID>): Iterable<T>

• A count() : long

• AdeleteByld(ID): void

A delete(T): void

• A deleteAllByld(Iterable<? extends ID>): void

A deleteAll(Iterable<? extends T>): void AdeleteAll(): void

note:: em.persist(-) intenrally uses em.merge(-) internally uses

ORM env.. (like hibernate) and also provides api to perform object persistence operations. (CURD

Operations)

In Layered Apps

hibernate api to perform save object operation (record insertion) hibernate api to perform update object operation (record updation)

Entity class ----- on 1 per db table Repository Interface ------ on 1 per table Service class controller class ------> on 1 per project --->on 1 per module. Runner classes ------> as needed

↓

note:: The internal hibernate api uses JDBC Code + SQL Queries to complete the persistence operations on DB table records

//IDoctorRepo.java

Flow of execution

================

package com.nt.repository;

import org.springframework.data.repository.CrudRepository;

import com.nt.entity.Doctor;

Entity class Id property type name

public interface IDoctorRepo extends CrudRepository<Doctor, Integer> {

}

InMemory Proxy class as the impl class of our Custom Repository Interface (IDoctorRepo) given JDK/CGLIB Libraries (g ii)

@Repository

public class Proxy2 implements

@Autowired

private EntityManaager em;

(Dynamically generated InMemory Proxy class)

IDoctorRepo,.....{

@Transactional

@Override

(AutoConfiguration)

(r)

=>The Jpa's EntityManager object will come as spring bean through AutoConfiguration based on the spring data jpa starter we add.

=>This EntityManager object also holds DataSource obj internally

which also came through Autoconfigration

from hibenrate api (hibnerate jar files) public class SessionImpl implements Session, EntityManager{

public <S extends T> S save(S entity) {

Assert.notNull(entity, "Entity must not be null");

if (entityInformation.isNew(entity)) {

public void persist(Object object) throws HibernateException { checkOpen();

firePersist(_new PersistEvent(null, object, this));

ses.persist(-)

@Override

em.persist(entity);

internally uses ses.save(-) of

return entity; (u)

hibenrate api to save the object

} else {

}

return em.merge(entity); --->internally uses ses.merge(-) of hibernate api to update the object

}

}

}

(t->(i))

=> DataSource obj is injected to EntityManager obj

internally generates JDBC code +

INSERT UPDATESQL Query to insert the record in db table

InMemory Proxy class obj

... other 11 methods are implemented

=> EntityManager object is injected Dynamically Generated

using EntityManager object and hibernate api support

}

JPA api internally uses hibernate api

//IDoctorService.java

package com.nt.service;

import com.nt.entity.Doctor;

note:: EntityManager obj is the object of a class given by underlying ORM s/w like hibernate implementing jakatra.jpa.EntityManager(1) directly or indirectly

=>To open the source code of class whose name is not typed in the Editors of Eclipse IDE use ctrl+Shift+T key

public interface IDoctorService {

public String registerDoctor(Doctor doctor);

}

//DoctorMgmtServiceImpl.java (service impl class) package com.nt.service;

@Service("doctorService")

public class Doctor MgmtServiceImpl implements IDoctorService { @Autowired

private IDoctorRepo doctorRepo;

@Override

(d)

public String registerDoctor(Doctor doctor) {

(v) Doctor doc=doctorRepo.Save(doctor);

return "Doctor obj is saved with id value :"+doc.getDocId();

}

(w)

application.properites

#jdbc properties (for oracle)

spring.datasource.driver-class-name-oracle.jdbc.driver.Oracle Driver

spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe

spring.datasource.username=system

spring.datasource.password=tiger

spring.datasource.hikari.maximum-pool-size=100

spring.datasource.hikari.minimum-idle=10

spring.datasource.hikari.keepalive-time=100000 #jpa -hibernate entries

spring.jpa.datasource-platform=org.hibernate.dialect.Oracle 10gDialect

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto-update

Client App

=========

@SpringBootApplication

(d) ->recoginizes @SBA

(b) *

public class BootDataJpaProj1Crud RepositoryApplication {

public static void main(String[] args) { //get IOC container

(k) gives IOC container

(a) run the Application

(c)- --> boot strapping of spring boot App

(d)

(e) reads the application.properties file content to Environment object

(f) @EnableAutoConfiguration of @SBA makes the following classes as spring beans based on jar files that are added through 'AutoConfiguation process. (This also uses the inputs from application.properties) (i) HikariDataSource pointing to oracle DB jdbc con pool (ii) EntityManager encapsulating the activated HB's SEsssionFactory, Session objects

iii) multiple other classes become spring beans

ApplicationContext ctx=SpringApplication.run(BootDataJpaProj1CrudRepositoryApplication.class, args); //get Service class obj (1)

IDoctorService service=ctx.getBean("doctorService", IDoctorService.class);

(n)

uses application.properties file

data through Environment object

(c)->(i) :: IOC container creatiion (Inside SpringApplication.run(-) method) (d)->(i) :: takes the given main class

of

try {

// create Doctor class object

}

Doctor doctor=new Doctor();

as configuration class becoz @Configuration (inside of @SBA) (d)-->(ii) --> Loads the main class and creates the object

doctor.setDocName("rajesh"); doctor.setSpecialization("MD-Cardio"); doctor.setIncome(890000.0); //invoke the b.method

(x) String resultMsg=service.registerDoctor(doctor);

System.out.println(resultMsg); (y)

catch (Exception e) {

}

e.printStackTrace();

}//main (z) app's execution is completed }//class

makes

it as the spring bean

Programmer ↓

spring data jpa api (repo.save(-))

↓

jpa api

♫

(em.persist(-))

repo:: Repository obj

ses :: HB's Session obj

em :: Entity Manager obj

ps :: PreparedStatement obj

(g) @ComponentScan of @SBA detects /scans the following things in the current and "sub pkgs of "com.nt" (current pkg)

.process

(i) DoctorMgmtServiceImpl as spring bean becoz of @Service (ii) IDoctorRepo as the custo Repsotiry Interface extending from Reposority(1) (marker interface) directly or indirectly.. In this the spring data jpa generates InMemory proxy class as shown above with @Repository annotation having jpa persistnece logics

for all the Crud Repository methods (12 methods)

(h) IOC container performs the pre-instantiation of singleton scope spring beans

in that proess the following spring bean class objs will be created

=>EntityManager obj

=>HikariDataSource obj

Becoz of @AutoConfiguration

=>DoctorServiceImpl obj

Becoz of Streo type annotations

=>Proxy2 class (InMemory proxy class) and etc..

(i) @Autowired perform the neccessary injections

=> EntityManager object with HikariDataSource object

=> Proxy2 class obj with EntityManager object (InMemory Proxy class obj with EntityManager obj)

=> DoctorMgmtServiceImpl class obj with Proxy2 class obj(Repository obj) (service impl class obj)

hibernate api (ses.save(-))

↓

jdbc api

(ps.executeUpdate(-))

generates SQL Query (INSERT SQL Query)

↓

DB Operation (Record insert operation)

be kept

(j) All singleton scope spring beans will in the internal cache of IOC container

(internal cache of the IOC container)

doctorService (?m) DoctorServiceImpl class obj ef

proxy2

hikariDataSource

Proxy2 class obj ref

HikariDataSource obj ref

...

-----

keys (bean ids)

values

of

To see the source code any class /interface/enum /annotation randomly whose jar file is added eclipse projects ctrl+shift+T -> type class name ---> open source code

=>To see the source code of certain class/interface/enum/annotation that is used in the eclipse editors =>use F3 key

=> Once the source code is opended to get list of methods

ctrl+o

=>To get hierarchy of the classes code

F4

finder methods in CrudRepository to perform SELECT Operations

CrudRepository<T, ID>

save(S) <S extends T> : S

AsaveAll(Iterable<S>) <S extends T> : Iterable<S>

● findById(ID): Optional<T>

•

A existsById(ID): boolean

findAll(): Iterable<T>

• A findAllById(Iterable<ID>): Iterable<T>

Acount() long

AdeleteById(ID): void

A delete(T): void

• A deleteAllById(Iterable<? extends ID>): void

A deleteAll(Iterable<? extends T>): void

A deleteAll(): void

(spring bean class obj refs)

boolean existsById(ID id)

========

Returns whether an entity/record with the given id exists or not

Parameters:

id - must not be null.

Returns:

true if an entity with the given id exists, false otherwise. Throws:

IllegalArgumentException - if id is null.

Q) How to make multi line comment given through short ctrl+shif+/ not distrubing the format of the code?

Ans) window menu ---> preferences ---> search formatting --> go to java-->formatter ---> create new profile(new) with any name (p1) ---> comments ---> deselect the following check box

✓ Enable block comment formatting

-->apply --->ok

=>The methods that are defined in the dynamically generated InMemory Proxy class implementing our Custom repository interface will throw all their exceptions as the Unchecked expceptions.. these exceptions will propagate to service class to Client App/Runnner, So we need to catch and handle the exceptions in client App/runner class to display exception related messages*endusers as the non-technical guiding messages.

code In Service Interface

```
public boolean isCustomerAvailable(Integer id);
```

Code in Service Impl class

```
@Override
public boolean isCustomerAvailable(Integer id) {
//use repo
boolean flag=custRepo.existsById(id);
return flag;
}
```

Code in Runner class

```
try {
boolean flag=custService.isCustomerAvailable(1); if(flag)
System.out.println("customer available");
else
System.out.println("customer not avaiable");
}
catch(Exception e) {
e.printStackTrace();
}
```

pre-defined Repository interfaces hierarchy in spring boot 2.x

Repository(1)

extends

CrudRepository(1)

extends

PagingAndSortingRepository(1)]

extends

JpaRepository(1)

НЫ

Pre-defined Repository Interfacs hierarchy in spring boot 3.x

CrudRepository(1)

extends

H

ListCrudRepository(1)

Repository(1)

extends

extends

JpaRepository(1)

Procedure to configure Lombok API with the new versions of the Eclipse IDE

=> Install Lombok API related p2 plugin as show below

help menu ---> install new softwares ----> select the following url

Work with: lombok api - https://projectlombok.org/p2

type filter text

Name

>

☐☐☐ Lombok

-->next --->next ---> restart IDE ---> ....

PagingSortingRepository(1)

extends

ListPagingAndSortingRepository(1)