Spring framework

→ To tell the ~~bean~~ program that it is a bean we annotation - @component & control 1 to import

→ To tell program that there are the donpendicies (bean) add annotation before class — @Autowired

→ where to search beans ? - @SpringBootApplication

→ To maintain all the beans use Application context

→ To know what's happening in background take the resource → applicatⁿ properties (spring)

→ logging.level.org.springframework=debug

→ along with @component if you add @primary this means you want to give it ~~the~~ more importance

→ If it's compulsory injection go for constructor
→ If it's optional go for setter

→ @Qualifier

· Bean scope –

Default – singleton

- singleton – one instance per spring context
- prototype – New bean whenever requested
- web { request – one bean per HTTP request
- session – one bean per HTTP session

@Scope(ConfigurableBeanFactory.SCOPE_SINGLETON)

Proxy –

Component Scan –

logger.info({}, —)

@PostConstruct – as soon as the dependency's are populated the postConstruct method is called.

→ @PostConstruct
```
public void postContruct() {
        logger.info("prepessing");
}
```

similar syntax for @PreDestroy
(just before the bean is demoted out of
context @preDestroy method is called)

CDI

- Java EE Depency Injection std (JSR-330)
- spring spports most annotations

@Inject (@Autowired)
@Named (@Component & @Qualifier)
@Singleton (Defines a scope of singleton)

Basic management of beans, is defined in
spring-core

COMPONENT ANNOTATIONS

@Component ↓ Generic Component

@Repository ↓ encapsulating storage retrieval & search behaviours typically from a relational database

@Service ↓ Business Service facade

@Controller ↓ Controller in MVC pattern

Spring Boot (starter projects) (Embedded)

There is no code generation in spring Boot &
spring Boot is neither an applicatⁿ server
nor a web server

Spring is famous for making microservices
- productⁿ ready features.

autoConfiguration

pointcut :- ("execution(* com.in28min.spring.aop.
springaop.dat .p. *.*(..))")
defines what of business method we want to intercept

@Aspect - is combinatⁿ of your joint point plus
you advise

Joinpoint - specific execution instance

@ AfterReturning - will get executed only when
the execution gets completed

@ AfterThrowing - This would intercept any
~~exp~~ ~~exe~~ exceptions that are thrown

@Around -> it better than @Afte---- (time)-

Interacting with Databases (spring)

application.properties
↳ spring.h2.console.enable = true

JPA — Java persistence API

JPA is the standard of doing object relational mapping (ORM)

@Entity

@Id

@GeneratedValue

while dealing with embedded sys. JPA automatically creates parameters only we have to insert the data. (only)

- Inserting, updating, implementing findById, deleteById ~~findAll~~ using JPA Repository method

↳ ~~imp~~

```
@Repository
@Transactional
public class PersonJpaRepository {

    // connect to database
    @PersistenceContext
    EntityManager entityManager;

    Public Person findById (int id) {
        return entityManager.find(Person.class, id);
    }
```

```
Public Person update (Person person) {
    return entityManager.merge (person);   //JPA
}

Public Person insert
            ''
```

In spring data JPA

• save

```
Public void deleteById (int id) {
    Person person = findById (id);
    entityManager.remove (person);
```

→ findAll does not work in JPA. it works in JPQL (java persistence query language) does not uses database. it uses entities

Maven Life cycle

$$\begin{tikzcd}
\text{Maven Life cycle} \arrow[d] & & & \\
\text{Validate} \arrow[d] & \text{Deploy.} & \text{Install} \arrow[l] & \text{verify} \arrow[l] \\
\text{compile} \arrow[r] & \text{Test} \arrow[r] & \text{Package} \arrow[r] & \text{Integration Test} \arrow[u]
\end{tikzcd}$$

Hibernate & JPA

application.properties -

# Enabling H2 console

spring.h2.console.enabled = true

# Turn statistics on

spring.jpa.properties.hibernate.generate_statistics=true

logging.level.org.hibernate.stat= debug

# To know parameters

logging.level.org.hibernate.type =debug

# Show all queries

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format_sql = ~~true~~ trace

note can't be used in production!

- unit test is run b/w context launch & Destroy!

deleteById ( )

assertNull (—)

find

@test

@DirtiesContext

Both are same like previous Queue

→ save method to update & insert entity

```
Say public Course save(Couree course) {
    if (course.getId() == null) {
        em.persist(course);      // insert
    }
    else
    { em.merge(couree);  // update
    }
    return course;
}
```

em
→ EntityManager – Keeps track of all entities & persist update all.

em.flush()

methods ← clear   em.clear()
detach   su em.detach(course2);
refresh   em.refresh(course1);

JPQL	SQL
select c from Courses c	select * from Courses

@Table
@column
@NamedQuery
@UpdatedTimestamp

Native Queries –

One to one Relationship

Eg- student & passport

@OneToOne

The defaut fetch ~~for~~ onetoone is Eager
(fetch = fetchType.EAGER)

Criteria Query (cq)

cq.from(course.class);

→ Add predicates etc to the Criteria Query

cq.where(like 100steps);

→ Building typed Query using entity manager & cq

TypedQuery<course> query = em.createQuery(cq.select(course

→ Criteria Builder

~~cq.joins~~ .joins

→ CourseRoot.joins("students", JoinType.LEFT);

Transaction Managment –

annotations make transaction managnt easy.

ACID properties –

A – Atomicity ← transactn is completely scueful or none is su

C – consistency – to ensure it is consitent irrespective of its sucess & it failur of transacth

I – Isolation – changes within transactn are visible to other transactn

D – Durability – any change that is done by transactn should persist, should be durable

Sorting ~~data~~ using spring data JPA repo

Sort. Direction. ~~ASC~~, "name"
DESC

Pagination — " —

PageRequest page = PageRequest. of(0,3)

Page<course> firstpag = repository .findAll(page).

logger. info ("firstpage -> {}", firstpage. getContent());

Custom queries ~~ust~~ — " —

List < course > findByNameAndId (—)
— " — findByName (String name);
— " — CountByName (— " —);
— " — findByNameOrderByIdDesc (— ");
— " — deleteByName (— " —);

Spring Data Rest —

@RepositoryRestResource (path = "courses")

4 Isolation levels of Transaction management

	Dirty Read	Non-repeatable ~~Read~~	phanthon Read
Read Uncommited	Possible	possible	Possible
Read commited	Solved	possible	Possible
Repeatable Read	solved	solved	possible
Serializable	solved	solved	Solved

Hibernate & JPA Tips

@ cacheable

@ SQLDelete (sql="update course set is_deleted = true where id=?)

@ Where (clause ="is_deleted = false)

@ Emmedable

@ Enemurated