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 dependicies (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 → applicat^n 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

request – one bean per HTTP request

session – one bean per HTTP session

@Scope(ConfigurableBeanFactory.SCOPE_SINGLETON)

Proxy –

componentScan –

logger.info(" ");

@PostConstruct – as soon as the dependencies are populated the postConstruct method is called.

```
@PostConstruct
public void postConstruct() {
        logger.info("preDestroy");
}
```

Similar syntax for @PreDestroy
(just before the bean is denoted 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 applicat^n server
nor a web server
spring is famous for making microservices
- product^n ready features.

autoConfiguration

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

@Aspect - is combinat^n of your joinpoint plus
you advise

Joinpoint - specific execution instance

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

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

@Around -> it better than @After--- (time).

Interacting with Databases (Spring)

application.property
↳ 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 sup. JPA automatically creates parameters only we have to insert the data. Only)

- Inserting, updating, implementing findById, deleteBy~~Id~~, ~~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

Validate → compile → Test → Package → Integration Test → verify → Install → Deploy.