e.g. public class A {
private B b;
@Autowired
public A (@Lazy B b) {
this.B b = b;
}

(3) Use setter / field injection.
→ Using setter injection also beans will get created but ~~injection~~ dependencies are injected when they are actually needed.

e.g.

```
class A {
private B b;

@Autowired
public void setB (B b) {
  this.B = B;
}

public B getB() {
  return b;
}
```

```
class B {
  private A a;

  @Autowired
  public void setA (A a) {
    this.a = a;
  }

  Public A getA () {
    return a;
  }
```

(4) Use @PostConstruct
→ injecting a dependency using @Autowired on one of the beans, and then use a method annotated with @postconstruct to set the dependency.

e.g.

```
class A {
@Autowired
private B b;

@postConstruct
public void init() {
  b.setA(this);
}
```

```
public B getB() {
  return B;
}
```

class B {
private A a;

classB

public void setA (A a) {
this.a = a;
}.

⑤ Implement ApplicationContextAware and InitizingBean
→ If one of the beans implement ApplicationContextAware the bean has a access to spring context & can extract the other bean from there.
→ Implementing InitializingBean indicates that this bean has to some actions after all its properties have been set.

e.g.
class A implements ApplicationContextAware, InitializingBean{
private B b;
private ApplicationContext context;

public B getB() {
return b;
}.

@Override
Public void afterPropertiesSet() throws Exception {
b = context.getBean(B.class);
}

@Override
public void setApplicationContext(final ApplicationContext ctx){
context = ctx;
}

* @Immutable in Hibernate.

→ This annotation tells Hibernate that any updates to an ~~and~~ immutable entity should not be passed on to the database without giving any error.
- @Immutable can be placed on a collection too

* Hibernate N+1 problem.

→ - N+1 problem is performance issue in ORM that fires multiple select queries in a database for a single select query at application layer.
- i.e. N = number of records in table and for each record one selection query for mapped table.

eg.

```
* class A {
    private Long id;
    private string name;

    @ManyToMany(fetch = FetchType.LAZY)
    private List<B> ListB;

* Class B {
    private Long id;
    private String name;
}.
```

- From Above, A class is mapped to class B
- To fetch all data from A table, we can use below query:-

```
List<A> findAllBy();
```

- Equivalent Queries that ORM will execute are;
  (i) Select * from A;

ii) select * from B where id = <A_id>;

- So we need one select for "A" and N additional selects for fetching B for each A_id.
- This is N+1 problem in Hibernate and can be identify by enabling SQL logging.

* How to resolve N+1 problem in Hibernate.

→ - N+1 problem can be ~~achie~~ resolve by creating join tables and get the combined result in single query.

① Spring Data JPA approach.

→ using @EntityGraph or using select query with fetch join.

i.e.

(i) @Query("select a from A a left join fetch a.b")
List<A> findAll();

(ii) @EntityGraph(attributePaths = {"b"})
List<A> findAll();

② Hibernate Approach.

→ (i) "select *from A a fetch a.b b"; ← HQL.

(ii) Criteria criteria = session.createCriteria(A.class);
criteria.setFetchMode("b", fetchMode.EAGER);

* How to exclude any package without using the basepackage filter?

→ @SpringBootApplication(exclude = {A.class}).

* How to disable a specific auto-configuration class?

→ @EnableAutoConfiguration((exclude = {DataSourceAuto-configuration.class})

* Spring ~~Init~~ Initializer.

→ The Spring initializer is a web application that generates a spring boot project with everything we need to start it quickly.

* What is filter in spring boot?

→ - A filter is an object used to intercept the HTTP requests and responses of application.
- Purpose of filter is performing filtering on either the request to resource or on the response from a resource or both.

* How to add filter to an application?

→ ① By implementating filter interface.
② Using FilterRegistrationBean.
③ Using MVC controller. (@WebFilter)

* @PathVariable.

→ Used to extract the value from URL.
e.g. http://localhost:8080/url/pankaj
this name can be extracted using @PathVariable.

```
@GetMapping(path="/url/{name}")
public String getName(@PathVariable String name){
    return name;
}
```
→ Pankaj

* What is Swagger?

→ - Swagger is used to describe structure of the APIs.
- It is an open-source service provided in spring boot which make easier for the machines to find out API structure for web services.

S.D.S.
Page:
Date: / /

* What is Hot-swapping in spring-boot?
→ - It is a way to reload the changes without restarting Tomcat or Jetty server. IDE supports byte code hot swapping.
- If we make any changes that don't affect the method signature, it should load without any side effect.

* What is AOP?
→ - Aspect-oriented programming.
- AOP supplements object-oriented programming that aims to increase modularity.
- AOP breaks program logic into various parts, which are called concerns.

* Shutdown in actuator?
→ It is a management endpoint that allows for a smooth and proper shutdown of application.
- It is not authorized by default, but it can be enabled by using management.endpoint.shutdown.enabled = true.

* Can spring boot also be used to create non-web applications?
→ - Yes, spring boot supports both web & non-web application development.
- We need to remove web dependencies from classpath & the application context to create a non-web application.

* What is starter dependency in spring boot module?
→ ① Data JPA starter ② Test starter ③ Security starter
④ Web starter ⑤ Mail starter ⑥ Thymeleaf starter

* @Bean annotation:

→ - Bean annotation is applied to a method to specify that it return a bean to be managed by spring context/spring container.

- They usually used in configuration classes methods so that bean methods may reference other bean method by calling them directly.

* Spring Validation using spring boot annotation:

→ - Annotations that can be used for spring validation defined by Java Bean validation API.

① @NotEmpty :→ not null or not empty

② @NotBlank :→ not null and must contain atleast one non-whitespace character.

③ @NotNull → not null only.

④ @Email → valid email-Id.

⑤ @Min → field must be a number whose value must be higher than or equal to specified value.
e.g: @Min(value = 10) → value >= 10.

⑥ @Max → field must be a number whose value must be less than or equal to specified value
e.g @Max(value = 10) → value <= 10.

⑦ Size → field size falls within the specified range. Can be applied to strings, collections and Arrays.

* @Primary annotation.

→ - @Primary annotation is used to give higher preference to a bean, when there are multiple beans of same type.

- @Primary can be used on any class directly or on methods with @Bean or with @Component

Repository in JPA is central repository marker interface.
→ Repository<T, ID>.

* @Conditional annotation
→ - It is used to indicate whether a given component is eligible for registration based on a defined component
- @Conditional annotation can be used for following:
(i) whether property is available or not using enviromental variables irrespective of its value.
(ii) Like Profiles, condition whether a property value is available or not.
(iii) Conditions based on a Bean definition / Bean objects are present in spring Application Context.
(iv) Conditions based on some Resources are present in spring Application Context or not.

* CRUDRepository
→ - It is a base interface & extends Repository interface
- Provides CRUD operation.
- Return Type of saveAll() method is Iterable.

* JPARepository
→ - extends PagingAndSortingRepository that extends CrudRepository.
- provide CRUD and paging operations, and also additional methods like flush(), saveAndFlush() and deleteBatch().
- Return Type of saveAll() method is List.

* PagingAndSortingRepository
→ - Extension of crudRepository
- Provides addition methods to retrieve entities using paging and sorting abstractions.
→ methods: ① Page<T> p = findAll(Pagable page);
② Iterable<T> i = findAll(Sort sort);