* final, finally and finalize in Java :

ⓐ final keyword :

final is a reserved keyword in java. We can use this keyword in java has diffrent meaning depending upon it is applied to variable, class or method.

1. final with variables:

- The value of variable cannot be changed once initialized.

- If we declare any variable as final, we can't modify it's contents since it is final, and if we modify it then we get Complile Time Error.

2. final with class :

- The class cannot be subclassed.

- Whenever we declare any class as final, it means that we can't extend that class or that class can't be extended.

Syntax :

```
final int b = 6;

final class student {...}
```

3. final with Method :

- The method cannot be overridden by a subclass.

- Whenever we declare any method as final, then it means that we can't override that method.

Note : If a class is declared as final then by default all the methods present in that class are automatically final but variables are not.

ⓑ finally keyword :

- finally also as a reserved keyword. The finally keyword is used ~~as~~ in association with a try/catch block and guarantees that a section of code will be executed, even if an exception is thrown.

The finally block will be executed after the try catch blocks, but before control trasfer back to it's origin.

1. Case 1 : Exceptions do not occure in the program.

2. Case 2 : Exception occures and corresponding catch block matches.

3. Case 3 : Exception occures and corresponding catch block not found / match.

* Application of finally block:

- Basically, the use of finally block is "Resource Deallocation"
- Means all the resources such as Network Connections, Database connections, which we opened in try block are needed to be closed so that we won't lose our resources as opened.
- So those resources are needed to be closed in finally block.

Note: The finally block is a key tool for preventing resource leaks.

(c) Finalize Method:

It is a method that the Garbage Collector always calls just before the deletion / destroying the object which is eligible for Garbage Collection, so as to perform clean-up activity.

It is not a reserved keyword.

Once the finalize method completes immediately, Garbage Collector destroy that object. finalize method is present in Object class. syntax is

protected void finalize throws Throwable { }

"There is no guarentee about the time when finalize is called."

1. Case 1: The object which is eligible for Garbage Collection, that object's corresponding class finalize method is going to be executed.

2. Case 2: We can call finalize method Explicitly then it will be executed just like normal method call but object won't be deleted / destroyed.

Note: As finalize method is a method and not a reserved keyword, so we can call finalize method Explicitely, then it will be executed just like normal method call but won't be deleted / destroyed.

3 Case 3:

Part a: If programmer calls finalize method, while executing finalize method some unchecked exception rises.

Keypoint: If programmer calls finalize method, while executing finalize method some unchecked exception rises, then JVM terminates the program abnormally by rising exception. So in this case the program termination is **Abnormal**.

Part b: If a Garbage Collector calls finalize method, while executing finalize method some un-checked exceptions rises.

Keypoint: JVM ignores that exception and rest of program will be continue normally. So in this case the program termination is Normal and not abnormal.

* Stream In Java :

Introduced in Java 8, the Stream API is used to process collection of objects.

"A stream is a sequence of objects that support various methods which can be pipelined to produce the desired result."

Features of Java Stream are:

- A stream is not a data structure instead it takes input from the collections, Arrays or I/O channels.
- Streams don't change the original data structures, they only provide the result as per the pipelined methods.
- Each intermediate operation is lazily executed and returns a stream as ~~per~~ a result, hence various intermediate operations can be pipelined. # Terminal operations mark the end of the stream and return the result.

③ forEach() :

To iterate through every element of the stream.

Ⓐ Intermediate Operations :

① Map ():

The map method is used to returns a ~~string~~ stream consisting of the results of applying the given function to the elements of this stream.

② filter() :

The filter method is used to select elements as per the predicate passed as argument.

③ Sorted() :

The sorted method is used to sort the stream.

Ⓑ Terminal Operations :

① Collect() :

Return the result of the intermediate operations perform on the stream.

② ~~forEach()~~ : reduce() :

The forEach method is used to reduce the elements of a stream to a single value.

* Lambda Expressions in Java 8 :

Lambda expressions basically express instances of functional interfaces.

Lambda expressions implement the only abstract function and therefore implement functional interfaces.

* Functionalities :

- Enable to treat functionality as a method argument, or code as data.
- A function that can be created without belonging to any class.
- A lambda expression can be passed around as if it was an object and executed on demand.

Syntax :

Lambda operator ⟶ body

where lambda operator can be :

- Zero Parameter :

```
() -> Sysout("zero parameter lambda");
```

- One Parameter :

```
(p) --> Sysout("one parameter: " + p);
```

- Multiple parameters :

```
(p1, p2) --> Sysout("Multiple parameters" + p1 + ", " + p2);
```

Note : Lambda expressions are just like functions and they accept parameters just like functions.

* Important Points :

- The body of a lambda expression can contain zero, one or more statements.
- When there is a single statement, curly brackets are not mandatory and the return type of the anonymous function is the same as that of the body expressions.
- When there are more than one statement, then these must be enclosed in curly brackets (a code block) and the return type of the anonymous function is the same as the type of the value returned within the code block, or void if nothing is returned.

* Reflection In Java :

It is a mechanism which enables a program to examine the meta-data (type info) of it's objects at runtime.

The required classes for reflection are provided under java.lang.reflect package.

- Through reflection, we can invoke methods at runtime irrespective of the access specifier used with them.

getClass(), getConstructors(), getMethods()

We can invoke a method through reflection if we know it's name and parameter types.

ⓐ getDeclaredMethod() :

It creates an object of the method to be invoked.

Class.getDeclaredMethod(name, parametertype)

Parameters :

- Name of a method whose object is to be created
- An array of class object.

ⓑ invoke() :

It invokes a method of the class at runtime.

Method.invoke(object, parameter)

Note : Through reflection, we can access the private variables and methods of a class with the help of it's class object and invoke the method by using the object.

ⓒ Class.getDeclaredField(fieldName) :

Used to get the private field. Returns the object of type field for the specified field name.

ⓓ field.setAccessible(true) :

Allows to access the field irrespective of the access modifier used with the field.

* Important Observations :

1. Extensibility feature
2. Debugging and Testing Tools
3. Performance Overhead
4. Exposure of Internals.

* Java Database Connectivity (JDBC):

"JDBC is an API used in java programming to interact with databases."

The classes and interfaces of JDBC allow the application to send requests made by users to the specified database.

Purpose of JDBC:

Enterprise applications created using the JAVA EE technology need to interact with databases to store application-specific information.
So, interacting with database requires efficient database connectivity, which can be achieved by using the ODBC (open db connectivity) driver.

* Components of JDBC:

① JDBC API:

- It provides various methods and interfaces for easy communication with the database.
- It provides two packages as follows, which contains the Java SE and Java EE platform to exhibit write once run everywhere capabilities.
- package : java.sql.*;
- It also provides a standard connect to a client application.

② JDBC Driver Manager:

- It loads a database specific driver in an application to establish a connection with a database.
- It is used to make a database-specific call to the database to process the user request.

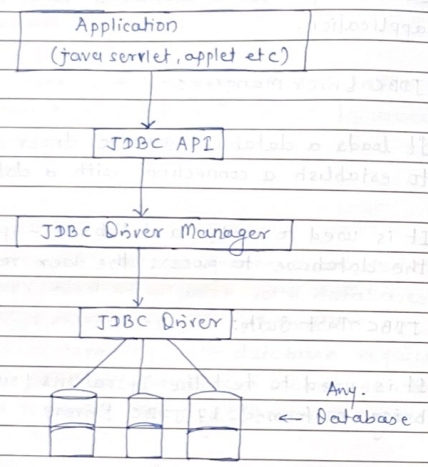
③ JDBC Test Suite:

- It is used to test the operations (such as insertion) being performed by JDBC Drivers.

④ JDBC-ODBC Bridge Drivers:

- It connects database drivers to the database.
- This bridge translates the JDBC method call to the ODBC function call.
- It makes use of the sun.jdbc.odbc package which includes a Native Library to access ODBC characterstics.

* Architecture of JDBC :



① **Application :** It is a java applet or a servlet that communicates with data source.

② **The JDBC API :** The JDBC API allows Java programs to execute SQL statements and retrieve results.

③ **Driver Manager :** It plays an important role in the JDBC architecture. It uses some database-specific drivers to effectively connect enterprise applications to databases.

④ **JDBC Drivers :** To communicate with data sourc through JDBC, you need a JDBC driver that intelligently communicates with the respective data source.

* JDBC Drivers :

JDBC drivers are client-side adapters (installed on the client m/c) that converts requests from Java program to a protocol that the DBMS can understand.

There are 4 types of JDBC Drivers.

1. Type-1 or JDBC-ODBC Bridge Driver
2. Type-2 or Native-API Driver
3. Type-3 or Network Protocol Driver
4. Type-4 or Thin Driver

* Types of JDBC Architecture :

① Two-tier Model :

A java application communicates directly to the data source. The JDBC driver enables the communications between the application and data source.

When a user sends a query to the database, the answers for those queries are sent back to the user in the form of results.

The data source can be located on diffrent m/c on a network to which a user is connected. This is known as a **client/server configuration**, where the user's m/c acts as a client, and the m/c has the data source running acts as the server.

② Three-tier Model :

In this, the user queries are sent to middle-tier services, from which the commands are again sent to the data source.

The results are sent back to the middle-tier, and from there to the user.

* Interfaces of JDBC API :

Driver Interface	Callable Statement I.
Connection Interface	ResultSet Interface
Statement Interface	Rowset Interface
Prepared Statement I.	DatabaseMetaData I.

* Classes of JDBC API :

DriverManager class	Clob class
Types class	Types. class

* JDBC with 5 steps :

① Register the driver class.

② Get Connection.

③ Create Statement.

④ Execute Query.

⑤ Close Connection.

* Object Relational Mapping (ORM) :

"ORM is a system that implements the responsibility of mapping the object to Relational Model."

That means it is responsible to store **Object Model** data into **Relational Model** and further read the data from Relational Model into Object Model.

Student class is object Model

```
@Entity (name = "STUDENT")
public class Student {
    @Id
    @Column (name = "ID")
    private Long studentID;
}
```

std tab in Rel. Model

ORM

student

ID int PRIMARY KEY

* Why ORM :

When work with OOP to persist data in RDBMS, there is mismatches between object model and Relational model if we work with traditional techinque like JDBC.

ORM fills gap in the following mismatches between object Model and Relational Model.

* Impedance Mismatch between Object Model and Relational Model:

The object oriented Model use classes whereas the relational database use tables.

This creates a gap (The Impedance Mismatch)

Loading and storing objects using a tabular relational database exposes us to 5 mismatch prob.

① Granularity:

- Granularity is the extent which a system could be broken down into small parts.
- Sometimes, you will have an object model which has more classes, than the number of corresponding tables in the database
- We says the object model is more granular than the relational model.

This is the mismatch between Object Model and Relational Model.

② Inheritance or Sub-types:

- Inheritance is a natural concept in oop languages.
- However, RDBMSs do not define anything simillar on the whole (yes some databases do have subtype support but it is completely non-standardize.

This is Inheritance Mismatched between Object Model and Relational Model.

③ Identity:

- A RDBMS defines exactly one notion of "sameness": the primary key.
- Java, However defines both object identity a = b and object equality a.equals(b)

This is Identity Mismatch between Object Model and Relational Model.

④ Associations:

- Associations are represented as unidirectional references in oop languages where as RDBMS's associations are bidirectional by using foreign keys.
- If you need bidirectional relationships in Java, you must define association twice.
- Likewise, you can not determine the multiplicity of a relationship by looking at the object domain model.

⑤ Data Navigation:

- The way you access data in Java is fundamentally different than the way you do it in relational database.
- In Java, you navigate from one association to an other walking the object Network.
- This is not an efficient way of retrieving data from a relational database.
- You typically want to minimize the number of SQL queries and thus load serval entities via JOINs and select the targeted entities before you start walking the object network.

* The solution for Granularity, Inheritance, Identity, Associations, Data Navigation problems to use ORM tool:

- An object Relational Mapping Tool provides a simple API for storing and Retrieving Java objects directly to and from the relational Database.

- ORM is a technique that allows an application written in an object oriented language to deal with information as objects, rather than using db specific concepts such as Rows, Columns and Tables which is facilitated by object/Relational Mapper.

* Advantages of ORM:

① Better System Architecture:
- ORM can help support a clean separation of concerns and ~~iden~~ independent development that allows parellel, simultaneous developement of application layers.
- Code Reusability, Cleaner seperation of data access layer.

② Reduce Coding Time:
- If you use JDBC it does not allow you to store objects directly to the database, you must convert the objects to a relational format.
- ORM too helps here, by genrating them on fly and there by saves a lot of time.

③ Caching and Transactions:
- Caching enables you to improve performance of data access operations.
- Most of the ORM tools like Hibernate provides the support for multiple transactions, they has locking mechanism and versioning.
- Which enables an application to have multiple trasactions and it does not affect the application performance.

* Popular ORM tools / frameworks in Java:

① Hibernate

② Top Link

③ Eclipse Link

④ Open JPA

⑤ MyBatis

* Hibernate Framework :

* Need :

① JDBC code is dependent upon the Database software being used. i.e our persistence logic is dependent, because of using JDBC.

② If working with JDBC, changing of Database in middle of the project is very costly.

③ JDBC code is not portable across the multiple db software

④ In JDBC, Exception handling is mandatory.

⑤ While working with JDBC, There is no support for object-level relationship.

⑥ In JDBC, there occures a Boilerplate problem. i.e. for each and every project we have to write the code. That increases the code length and reduces the readability.

To overcome the above problem we use ORM tool, i.e. nothing but Hibernate framework.

* About Hibernate Framework :

- Hibernate is a framework which provides some abstraction layer, meaning that the programmer does not have to worry about the implementations.

- Hibernate does the implementations for you internally like Establishing a connection with the database, writing query to perform CRUD operations etc.

- It is a Java framework which is used to develop persistance logic. persistance logic means to store and process the data for long use.

- More precisely Hibernate is an open-source, non-invasive, light-weight java ORM framework to develop objects which are ~~iden~~ independent of the database software and make independent persistence logic in all JAVA, JEE.

* Framework :

Framework means it is special install-able software that provides an abstraction layer on one or more technologies like JDBC, Servlet, etc to simplify or reduce the complexity for the development process.

* Open Soure Means:

Hibernate framework is available for everyone without any cost.

The source code of Hibernate is also available on the Internet and we can also modify the code.

* Light-Weight Means:

- Hibernate is less in size means the installation packages is not big size.
- Hibernate does not require any heavy container for execution.
- It does not require POJO and POJI model programming.
- Hibernate can be used alone or we can us Hibernate with other Java technology and framework.

* Non-invasive Means:

The classes of Hibernate application development are loosely coupled classes with respect to the Hibernate API.

i.e. Hibernate class need not implement Hibernate API interfaces and need not extend from Hibernate API classes.

* Functionalities supported by Hibernate framework:

- Hibernate framework supports **Auto DDL** operations. In JDBC manually we have to create table and declare the data-type for each and every column. But in Hibernate can do DDL operation for you internally like creation of table, alter a table etc.
- Hibernate supports **Auto Primary Key Generation**. It means in JDBC we have to manually set primary key for a table.
- Hibernate framework is independent of Database because it supports **HQL (Hibernate Query Language)** which is not specific to any database. Whereas JDBC is database dependent.
- In Hibernate **Exception Handling is not mandatory**, whereas In JDBC exception handling is mandatory.
- Hibernate supports **Cache Memory** whereas JDBC does not support cache memory.
- Hibernate is a ORM tool means it supports object relational mapping. Whereas JDBC is not object oriented moreover we are dealing with values means primitive data.

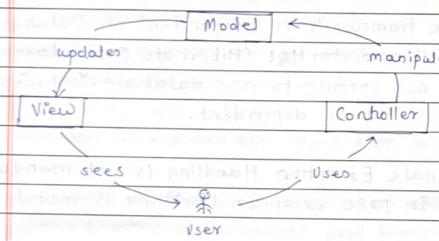
  In hibernate each record is represented as object but in JDBC each record nothing but a data which is nothing but primitive values.

* MVC Design Pattern :

The Model View Controller design pattern specifies that an application consist of a data model, presentation information and control information.

The pattern requires that each of these be separated into different objects.

MVC is more of an architectural pattern, but not for the complete application.



- The Model contains only the pure application data, it contains no logic describing how to present data to the user.
- The view presents the model's data to the user.
- The controller exist between the view and the model. It listens to events triggered by view and executes the appopriate reaction to these events.

* Servlet : (Controller)

Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any request.

Servlet is a web-component that is deployed on the server to create a dynamic web page.

Advantages :

① Better Performance :
Because it creates a thread for each request, not process.

② Robust :
JVM manages servlets, so we dont need to worry about the memory leak, garbage collection etc.

③ Portability : it uses Java language.

④ Secure : it uses Java languge.

* JAVA Server Pages (JSP) (View)

- It is a server side technology. which is used for creating web application with dynamic web content.
- JSP is an advanced version of Servlet Technology. In this JSP tags are used to insert JAVA code into HTML pages. Can be inserted HTML / XML or both.
- JSP is first converted into servelet by JSP container before processing the client's request.

* Features of JSP :

1. Coding in JSP is easy..
2. Reduction in the length of code.
3. Connection to Database is easier.
4. Make interactive websites
5. Portable, Powerful, Flexible and easy to maintain
6. No Redeployment and No Re-Compilation.
7. Extension to Servlet.

* Plain Old JAVA Objects (POJO) : (Model)

It is an ordinary Java object, not bound by any special restriction other than those forced by the Java Language Specification and not requiring any classpath.

POJOs are used for increasing the readability and re-usability of a program.

* Java Beans :

Beans are special type of POJO's. There are some restrictions on POJO to be a bean.

- Fields should be private
- Fields should have getters or setters or both
- A no-arg constructor should be there in a bean.
- Fields are accessed only by constructor or getter setters.

* Spring Framework :

A Spring Framework provides a comprehensive programming and configuration model for modern Java based enterprise applications - on any kind of deployment platform.

A key element of spring is infrastructural support at the application level.

Spring focuses on the "plumbing" of enterprise application so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.

* Features :

1. Core Technologies:
   Dependency Injection, events, resources, validation, data binding, type conversion, AOP, i18n, SPEL.
2. Testing:
   Mock objects, Test Context framework, Spring MVC Test, web Test Client
3. Data Access:
   transactions, DAO support, JDBC, ORM, Marshalling XML
4. Spring MVC and Spring WebFlux web frameworks.

⑤ Integration :
remoting, JMS, JCA, JMX, email, tasks, cache, scheduling etc.

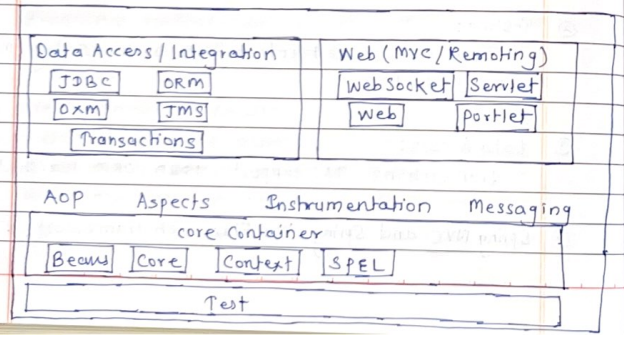
⑥ Languages :
Kotlin, Groovy, dynamic languages.

⑦ Spring framework is a lightweight when comes to it's size and transparency.

* Spring framework Architecture :

Spring could potentially be a one-stop shop for all your enterprise application.
However, Spring is modular, allowing you to pick and choose which modules are applicable to you, without having to bring in the rest.

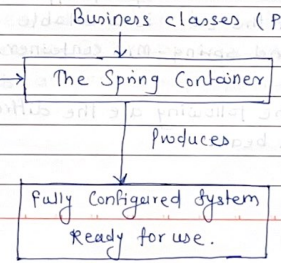
The spring framework provides about 20 modules which can be used based on an application reqt.



* IOC container :

- Spring IoC container is the core of spring framework.
- It creates the objects, configures and assembles their dependencies, manages their entire life cycle.
- The container uses Dependency Injection (DI) to manage the components that make up the application.
- It gets the information about the objects from a configuration file (xml) or Java code or Java Annotations and Java pojo class.
- Since the controlling of Java objects and their lifecycle is not done by the developers, hence the name is Inversion of Control.

following diagram depicts how the container makes use of Configuration metadata and Java pojo classes to manage beans.



* Spring Bean:

In spring, the object that form the backbone of your application and that are managed by the spring IOC container are called beans.

A bean is an object that is instantiated, assembled and otherwise managed by a spring IOC container.

* Scope of bean:

Bean scope refers to the lifecycle of Bean that means when the object of Bean will be created or instantiated, how long does that object live and how many objects will be created for that bean throughout.

- The spring framework provides five scopes for a bean.
- We can use three of them only in the context of web-aware Spring Application Context and the rest of the two is available for both IoC Container and spring-MVC container.

The following are the diffrent scopes provided for a bean:

① Singleton:

Only one instance will be created for a single bean defination per spring IoC container and same object will be shared for each request made for that bean.

② Prototype:

A new instance will be created for a single bean defination every time a request is made for that bean.

③ Request:

A new instance will be created for a single bean defination every time an HTTP request is made for that bean. But only valid in the context of web-~~server~~ aware Spring Application Context.

④ Session:

Scopes a single bean defination to the lifecycle of an HTTP Session. But only valid in the context of a web-aware Spring Application Context.

⑤ Global-Session:

Scopes a single bean defination to the lifecycle of a global HTTP session. It is also only valid in the context of web-aware Spring Application-Context.

* Lazy Loading :

Lazy loading is a software design pattern where the initialization of an object occures when it is actually needed and not before to preserve simplicity of usage and improve performance

- Lazy Loading is essential when the cost of object creation is very high and the use of the object is very rare.

- So, this is the scenario wher it's worth implementing lazy loading.

* Dependency Injection :

Dependency Injection is a fundamental aspect of the spring framework, through which the Spring container "injects" objects into other objects or "dependencies".

Simply, this allows for loose coupling of components and moves the responsibility of managing components onto the container.

Types of Dependency Injection:

① Setter Dependency Injection (SDI):

② Constructor Dependency Injection (CDI):

① Setter Dependency Injection :

This is the simpler of the two DI methods. In this, the DI will be injected with the help of setter and/or getter methods.

Now to set the DI as SDI in the bean, it is done through the bean-configuration file. For this, the property to be set with the SDI is declared under <property> tag in the bean-configu file.

② Constructor Dependency Injection :

In this, the DI will be injected with the help of constructors. Now to set the DI as CDI in bean, it is done through the bean-configuration file.

For this, the property to be set with the CDI is declared under the <constructor-arg> tag in the bean-config file.

* Need of Dependency Injection :

Suppose class one needs the object of class Two to instantiate or operate a method, then class One is said to be dependent on class Two.

It might appear ok but in real world this could lead to lot of problem, including system failure.

Spring IoC resolves such dependencies with DI, which makes the code easier to test and reuse.

* Autowiring :

The spring framework can inject dependencies automatically. The spring container detects those dependencies specified in the configuration file and @ the relationship between the beans.
This is referred as Autowiring in spring.

An autowired application requires fewer lines of code comparatively but at the same time, It provides very little flexibility to the programmer.

Modes of Autowiring :

① No :
This mode tells the framework that autowiring is not supposed to be done.
It is default mode used by spring.

② byName :
It uses the name of the bean for injecting dependencies. However, it requires that the name of the property and bean must be same.
It invokes the setter method internally for autowiring.

③ byType :
It injects the dependency according to the type of the bean. It looks up in the configuration file for the class type of the property. If it finds a bean that matches, it injects the property. If not, the program throws an error.
The names of the property and bean can be diffrent in this case. It invokes the setter method internally for autowiring.

④ constructor :
It injects the required dependencies by invoking the constructor. It works similar to the "byType" mode but it looks for the class type of the constructor argument.
If none or more than one bean are detected, then it throws an error, otherwise, it autowires the "byType" on all constructor arguments.

⑤ autodetect :
The autodetect mode uses two other modes for autowiring - constructor and byType. It first tries to autowire via the constructor mode and if it fails, it uses the byType mode for autowiring.
It works in spring 2.0 and 2.5 but it is deprecated from spring 3.0 onwards.

* Collection in Spring :

① List (<list>) :
Allow duplicates.

② Set (<set>) :
Duplicates not allowed.

③ Map (<map>) :
name - value pair

④ Props (<props>) :
name - value pair where name & values are only type of string.

* Spring Framework Annotations :

Spring Annotations are a form of metadata that provides data about a program.
Annotations are used to provide supplemental information about the program.

① Spring Core Annotations :

Present in org.springframework.beans.factory.annotations and org.springframework.context.annotations packages are commonaly known as Spring Core annotations.

DI - Related Annotations :

@ Autowired
@ Qualifier
@ Primary
@ Bean
@ Lazy
@ Required
@ Value
@ Scope
@ Lookup, etc

Context Configuration Annotations :

@ profile
@ Import
@ ImportResource
@ PropertySource, etc

② Spring Web Annotations :

Present in org.springframework.web.bind.annotations packages are commonly known as web annotations.

@ RequestMapping
@ RequestBody
@ PathVariable
@ RequestParam
@ Controller
@ RestController
@ ModelAttribute
@ CrossOrigin

Response Handling Annotations :

@ ResponseBody
@ ExceptionHandler
@ ResponseStatus.

③ Spring Boot Application:

Annotations present in org.springframework.boot.autoconfigure and org.springframework.boot.autoconfigure.condition packages are commonly known as spring boot annotations.

@SpringBootApplication
@EnableAutoConfiguration

Auto-Configuration Conditions:

@ConditionalOnClass and @ConditionOnMissingClass
@ConditionalOnBean and @ConditionalOnMissingBean
@ConditionalOnProperty
@ConditionalOnResource
@ConditionalOnWebApplication ~~and~~
@ConditionalOnNotWebApplication
@ConditionalExpression
@Conditional

④ Spring Schedulling Applications:

org.springframework.schedulling.annotation

@EnableAsync
@EnableScheduling
@Async
@Scheduled
@Schedules.

⑤ Spring Data Annotations:

Common Spring Data Annotations:

@Transactional
@NoRepositoryBean
@Param
@Id
@Transient
@CreatedBy
@LastModifiedBy
@CreatedDate
@LastModifiedDate

Spring Data JPA Annotations:

@Query
@Procedure
@Lock
@Modifying
@EnableJpaRepositories

Spring Data Mongo Annotations:

@Document
@Field
@Query
@EnableMongoRepositories

⑥ Spring Bean Annotations:

org.springframework.stereotype

@ComponentScan
@Configuration

Stereo-type Annotations:

@Component
@Repository - CRUD Operations
@Controller - req, res
@Service - Business logic

* Aspect Oriented Programming (AOP) :

AOP as the name suggests uses aspects in programming. Also defined as the breaking of code into different modules, also known as modularisation, where the aspect is the key unit of modularity.

Aspects enable the implementations of crosscutting concerns such as - transaction, logging not central to business logic without cluttering the code core to its functionality. It does so by adding additional behaviour that is the advice to the existing code.

Dominant frameworks in AOP:

AOP includes programming methods and framework on which modularisation of code is supported and implemented.

① AspectJ : It is an extension for Java programming created at PARC research centre. It uses Java like syntax and included IDE integrations for displaying crosscutting structure. It has its own compiler and weaver, on using it enables the use of full AspectJ language.

② JBOSS : It is an open source Java application server developed by JBoss, used for Java development.

③ Spring : It uses XML based configuration for implementing AOP, also it uses annotations which are interpreted by using a library supplied by AspectJ for parsing and matching.

* How Aspect-Oriented Programming works with Spring:

- One may think that invoking a method will automatically implement cross-cutting concerns but that is not the case.

- Just invocation of the method does not invoke the advice (the job which is meant to be done).

- Spring uses Proxy Mechanism. i.e. it creates a proxy object which will wrap around the original object and will take up the advice which is relevent to the method call.

- Proxy objects can be created either manually through proxy factory bean or through auto proxy configuration in the XML file and get destroyed when the execution completes.

- Proxy object are used to enrich the original behavior of the real object.

* Common Terminologies in AOP:

① Aspect:

"The class which implements the JEE application cross-cutting concerns (transaction, logger etc) is known as the aspect."

It can be normal class configured through XML configuration or through regular classes annotated with @Aspect.

② Weaving:

"The process of linking Aspects with an Advised object." It can be done at load time, compile time or at run-time.

Spring AOP does weaving at runtime.

③ Advice:

"The job which is meant to be done by an Aspect or it can be defined as the action taken by the Aspect at a particular point."

ⓐ Before: (@Before)

Runs before advised method is invoked

ⓑ After: (@After)

Runs after advised whether it is successful or not

ⓒ AfterReturning (@AfterReturning):

Runs after advised method successfully completes i.e. without any runtime exception.

ⓓ Around: (@Around)

Strongest amongst all. it wraps around and runs before and after the advised method.

This type of advise is used where we need frequent access to a method or database like caching.

④ Joinpoints:

"A Joinpoint is a specific point in the application such as method execution, exception handling, changing object variable values, etc."

An application has thousands of opportunities or points to apply advise. These points are known as join points.

for example: A advice can be applied at every invocation of a method or exception be thrown or at various other points.

But spring AOP currently supports only method execution join points (advising the execution of methods on spring beans).

⑤ Pointcut:

"Since it is not feasible to apply advice at every point of the code, therefore, the selected join points where advice is finally applied are known as the pointcut."

* Boilerplate Code:

Section of code that are repeated in multiple places with little to no variation.

* Proxy class:

"Proxies in spring are an implementation of the proxy design pattern and help facilitate the paradigm of AOP."

Proxies provide a surrogate to control access to an object.

Spring AOP uses either JDK dynamic proxies or 'CGLIB' to create proxy.

* Spring MVC Framework:

Spring MVC framework follows the model-View-Controller design pattern. It is used to develop web applications. It works around DispatcherServlet.

DispatcherServlet handles all the HTTP req and responses. It dispatches the request to handlers. It uses @Controller and @RequestMapping as default request handlers.

The @Controller annotation defines that a particular class is a controller. @RequestMapping annotation maps web requests to spring controller methods.

Model:

The model encapsulates the application data.

View:

View renders the model data and also generates HTML output that the client's browser can interpret.

Controller:

The controller processes the user requests and passes them to the view for rendering.

* Spring MVC framework works as follows:

1. All the incoming request are intercepted by the DispatcherServlet that works as the front controller
2. The DispatcherServlet then gets an entry of handler mapping from the XML file and forwards the request to the controller.
3. The object of modelAndView is returned by the controller
4. The dispatcherServlet checks the entry of view resolver in the XML file that invokes the appropiate view component.

Add dependencies into pom.xml file.

* Dispatcher Servlet :

"Dispatcher servlet acts as the front Controller for spring-based web application".

What is front Controller ?

Any request is going to come into our website the front controller is going to stand in front and is going to accept all the requests and once the front controller accepts that request then this is the job of front controller that it will make a decision ~~for~~ that who is the right controller to handle that request.

Dispatcher servlet handles an incoming HttpRequest, delegates the request, and processes that request according to be configared HandlerAdapter interfaces that have been implemented within the spring application along with accompanying annotations specifying handlers, controller endpoints, and response objects.

* Life Cycle of Servlet :

stages of the Servlet Life Cycle :

1. Loading a servlet
2. Initializing the servlet
3. Request Handling
4. Destroying the Servlet

Servlet Life Cycle Methods :

1. Init() :
   The Servlet.init() method is called by the Servlet container to indicate that this Servlet instance is instantiated successfully. and is about to put into service.

2. service() :
   The service() method of the Servlet is invoked to inform the servlet about the client requests
   - This method uses ServletRequest object to collect the data requested by the client.
   - This method uses Servlet Response object to generate the output content.

3. destroy() :
   The destroy() method is activated, the servlet container realease the servlet instance.