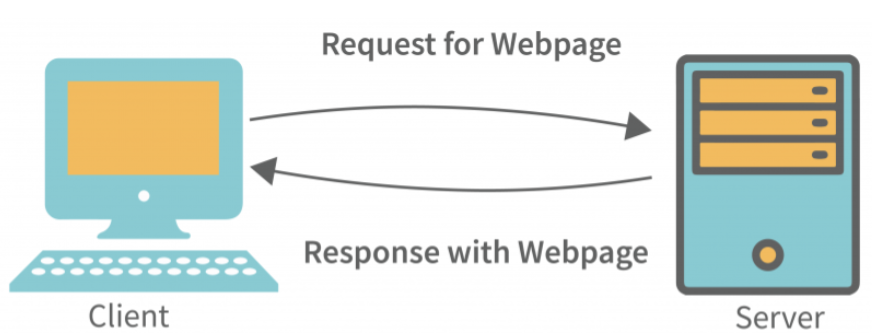
**Application** – Application is a set or collection of programs combined together to perform a specific task for the user.

3 types of Application

1. Standalone application – standalone application does not depend upon any other software and network resources to run they run independently and all the task of that application will be happening in one system(like data-processing, stroage) and hence it does not require any internet connection. For ex – Calculator, Notepad, Word etc
2. Web-Application – Web application depend upon other software(like -web browser) and network resource to run and all the task(like data-storage and processing) will not happen in one device hence it require internet connection.For ex – Amazon, Myntra.
3. Mobile-Application – application which are specifically designed for mobiles are called mobile application. For ex – whatsapp, instagram

**CLIENT-SERVER ARCHITECTURE**

Client will make request to server, server will process that request and give back response to the client.

****

**Client -** Client is anything who is making request to server.

* Client can be a browser, application or system.

Request example are search request, download request, Streaming request etc.

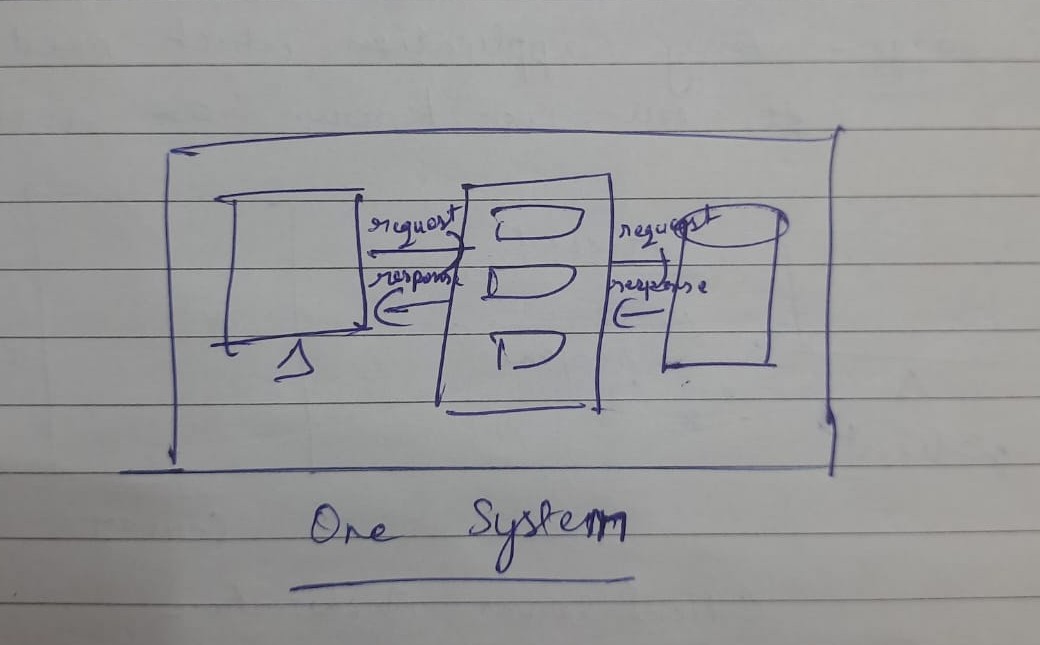
**Server** - Server can be a software or computer system which process the client request and give back response to client.

It consist of 3 main Tier

1. I tier Architecture
2. II tier Architecture
3. III tier Architecture

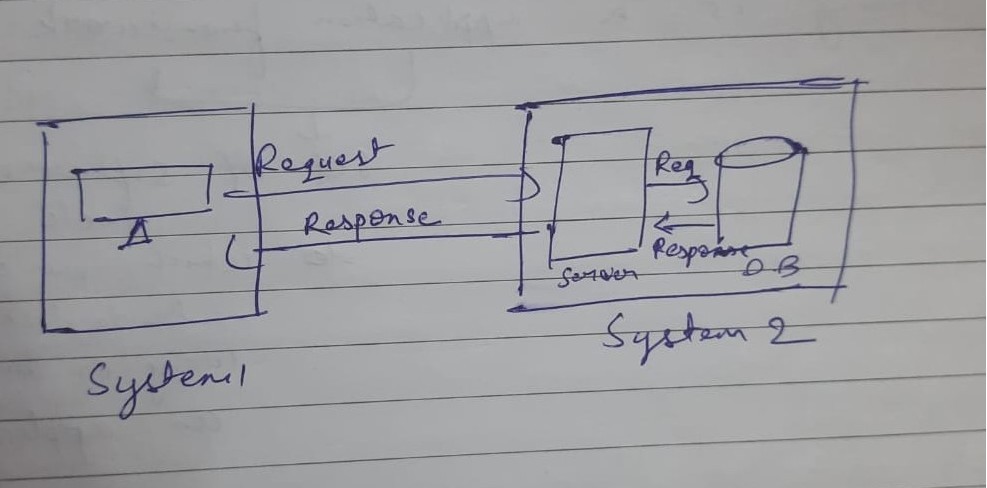
1st tier Architecture

* All the three client, server, database will be in one machine.
* Here client will make request to server and server will forward request to database and database give back response to server and server will send this response to client.
* All this will be happening in one machine that why it is called 1 tier architecture.
* Standalone application like notepad, calculator follow this Architecture.



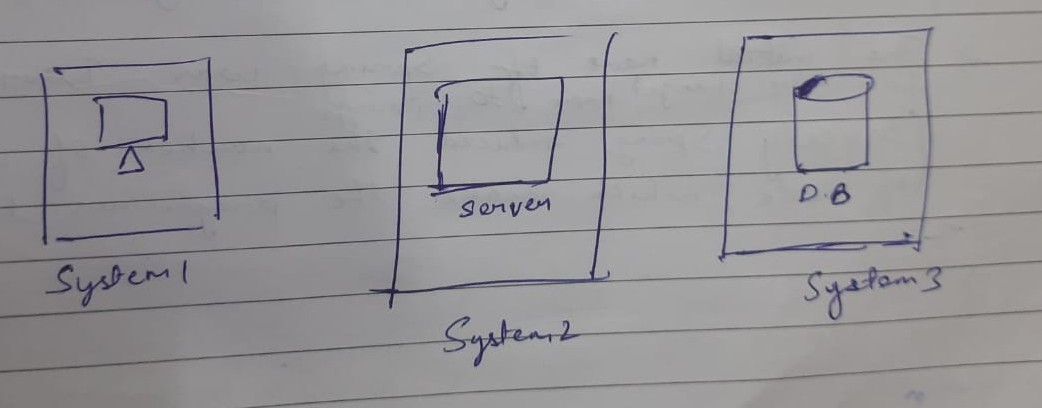
2nd tier Architecture

* Client will be in some other machine and server and database will be in single machine.
* Client will make request to server over the internet to server as both are in different machine, server will look in database, database will give response to server and server again will back response to client over internet.
* Here 2 machines are involved so it is called 2 tier architecture.
* Small banking application will follow 2nd tier architecture where they have client in one machine and database and server in other machine.



3rd tier architecture

* Client, server, database all will be in different machine.
* Client will make request to server over internet and server will make request to database over internet database will give response to server, server will give back response to client.
* All the things will be happening in three different machine that why it is called three-tier architecture.



**Features Of Spring**

1. Spring is light-weight because of POJO implementation.
2. Spring requires less-configuration
3. Spring helps in avoiding boiler-plate code or repetative code
4. Spring helps in developing loosely-coupled application because of separation of layers MVC(Model,view,controller).
5. Spring has IOC(inversion of control), which makes the object implicitly, programmer does not have to make object.

**IOC**

* stands for inversion of control.( control of object creation is given to program from programmer )
* In Spring Programmer does not create the object, IOC container will create the object, programmer just have to give information of object creation in configuration file, IOC will read that file and create the object.

IOC will do four things

Read the configuration file

Create the object

Manage the lifecycle of object

Inject dynamic values to object

Note – Objects inside IOC container are called beans or objects whose lifecycle is managed by IOC are called beans.

Three types Of IOC container

* Core container
* J2EE container
* Web Container

**POJO Class(Plain Old Java Object)**

POJO is a class on which there is no special restrictions other than imposed by the java is called POJO class like(Rules for keyword,identifier etc)

Rules are

1. Class must be public
2. It should not extend or implement any class
3. Private data members
4. Public getter and setter
5. public no argument constructor

Advantage of POJO class – you can build light-weight application using POJO class.

**Dependency Injection** means when IOC inject object into a reference variable is called D.I

Three ways

1. Constructor Injection – Dependencies are injected through a constructor.
2. Setter Injection – Dependencies are injected via setter methods.
3. Field Injection (using @Autowired) – Dependencies are injected directly into fields.

**Components required to make Spring Application**

1. POJO Class
2. Configuration file(.xml) / Configuration class(.java)
3. Driver class

Note- Since Spring components are not present in JDK so we need to add the Spring dependency i.e Spring-context 6.1.6 externally into the pom.xml file

**Spring Configuration file(.xml) root tag is**

* Spring configuration file root tag is <beans>.
* It’ s child tag is <bean>.
* bean tag has two attribute called id & name

In id we mention one id which is used retrieve that object from IOC container.

In name we mention FullyQualifiedClassName of the class whose object we want to create.

Syntax

<beans>

<bean id=” “ class=”fullyQualifiedClassName”>

</bean>

<beans>

**Rules for passing ClassName in Driver Class**

(These rules will be used when we are fetching object from IOC container using classname in getBean() method)

Animal 🡪 animal

RCB 🡪 RCB

BangaloreBulls 🡪 bangaloreBulls

For ex- con.getBean(“animal” , Animal.class)

1. Using Configuration file(.xml) through only tag(without annotation)
2. Using Configuration file(.xml) through annotation
3. Using Configuration class(.java) through annotation

**Creation of object And dependency injection using xml file through tags(without annotation)**

* Here bean tag will be used in xml file to create object.
* There are 2 ways by which you can inject values in non-static variables

Using property tag 🡪 which uses setter function to inject values

Using constructor-arg tag🡪 which uses constructor to inject values

* non-static variable can be three types

Primitive variable

Reference variable

CollectionType variable(List, Map)

(for practice)

Make Program and Inject values in primitive variable,reference variable and CollectionType variable using

1) setter function using xml file without annotation

2) constructor using xml file without annotation

**Creation of object and dependency injection using configuration class**

**Note-** When configuration class is used then only annotation are used for object creation and dependency injection

* @Component is used to create object.
* There are 3 ways by which you can inject values in non-static variable

Using annotation directly in non-static variable

Using annotation with setter function

Using annotation with constructor

* non-static variable can be of two type

primitive variable

reference variable

(for practice)

Make program injecting values in primitive variable, reference variable using configuration class

1. directly inject values in variable.
2. With annotation using setter function.
3. With annotation using constructor.

**Creation of object and dependency injection using configuration file(.xml) through annotation**

In this situation we have to write one line in configuration file(.xml)

<context:component-scan base-package=”PackageName” />

So the IOC will know which package it has to scan for making object and dependency injection.

Rest all will be done through annotation.

(for practice)

Make program injecting values in primitive variable, reference variable using configuration file(.xml) through annotation.

1. directly inject values in variable.
2. With annotation using setter function.
3. With annotation using constructor.

**Annotation**

@Component

* It is a class level annotation
* It is used to tell IOC that this class object it has to make.
* If you are using configuration file(.xml), you have to use <context:component-scan base-package=”PackageName”> so by reading base-package attribute IOC will know in which package it has to go, IOC will go there, and look for the @Component annotation and make the object.
* If you are using configuration class(.java), you have to use @Configuration and @ComponentScan(basePackages=”PackageName”) so by basePackages attribute IOC will know in which package it has to go, IOC will go there and look for @Component annotation and make the object.

@Value

* It is a dependency injection mechanism rather than variable-level annotation as it scope goes beyond variable as it can be used with variable, setter method and constructor.
* Using this attribute we can inject value in the non-static variable

Syntax - @Value(“1”)

Or

@Value(value=”1”)

@Configuration

* It is a class level annotation.
* It is used to tell IOC that this class is the configuration class, it should be followed with @ComponentScan(basePackage=”PackageName”) so that IOC knows which package it has to scan to make object and for dependency injection.
* Both these annotation are used if we want to remove configuration file(.xml) from our project.

@Autowired

* It is variable level annotation
* It will automatically wire/inject the required POJO class object into reference variable.

But if it is a interface type reference variable, it will automatically wire the respective implementation class object of interface into the reference variable. And if there is one interface implemented by two classes then IOC will be confused which implementation class object it has to wire in reference variable then it throws one exception called NoUniqueBeanDefinitionException.

In order to overcome this ambiguity we use @Primary or @Qualifier

@Primary

* It is a class level annotation.
* It is used to overcome ambiguity issue, ambiguity means when multiple bean of the same type are available the IOC will get confused which bean object it has to create, so the class annotated with @Primary, IOC will create the object of that class and inject in reference variable.

@Qualifier

* It is a variable level annotation
* It is also used when multiple beans are available for same type. To avoid confusion which bean object IOC has to create,s we use @Qualifier just above the reference variable and it has one attribute value in which we mention that class name(in lowercase) whose object we want to inject in reference variable.
* @Qualifier has higher priority than @primary.

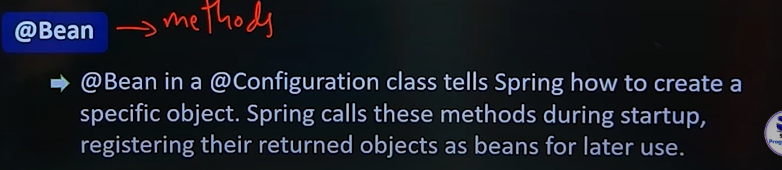
(Make program)

Injecting object in interface type reference variable using annotation direct in the reference variable

Case 1 – one interface has one implementing class

Case 2 – one interface has more than one implementing class (use @Primary or @Qualifier)

@scope



@ResponseBody

* @ResponseBody annotation is used to tell that the return value of controller (means the response) should directly written to Http response body instead of rendering to view page.