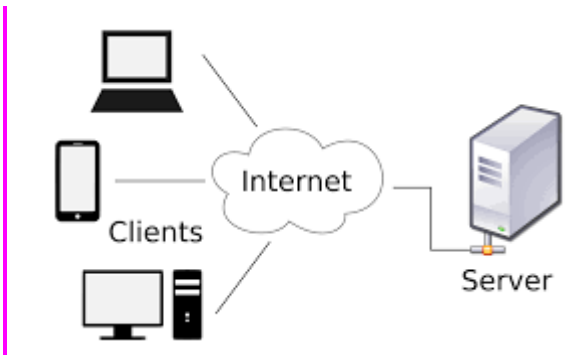
Java 1 SE: CORE JAVA
JAVA ENTERPRISE EDITION:➔EnTERPIRSE ➔ ORGAMOZATIONS➔WEBAPP/API
EJB➔Server-side component architecture

Enterprise JavaBeans Technology

Enterprise JavaBeans (EJB) technology is the server-side component architecture for Java Platform, Enterprise Edition (Java EE). EJB technology enables rapid and simplified development of distributed, transactional, secure and portable applications based on Java technology.

Server ---➔ client server technolyge



What is port ?
Protocol➔httm,udb , ects...
Only java mattum thannaa----------?
Ejb is old versions why not learn ➔provide by java teams
What is community driven?
Framework➔ hibernate, struts, springs

What is spring framework?
The Spring Framework provides a comprehensive programming and configuration model for modern Java-based enterprise applications - on any kind of deployment platform.

comprehensive programming ==all merged

what is spring BOOT?
Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".
Project anna statap pannilam=
production-grade== house

Weekly Test

We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need minimal Spring configuration.

What is Configuration:

- ⇨ Spring boot is opinionated
- ⇨ Spring boot is standalone
- ⇨ Spring boot is production Grade

Configuration::
JAR FILES:➔ JAVA Archive files➔ zip files constrain java class files==core java

Oracle—jer
H2 database
Mysql==jer
Read files === excel,video,image etc…

Web Applications➔ WAR files

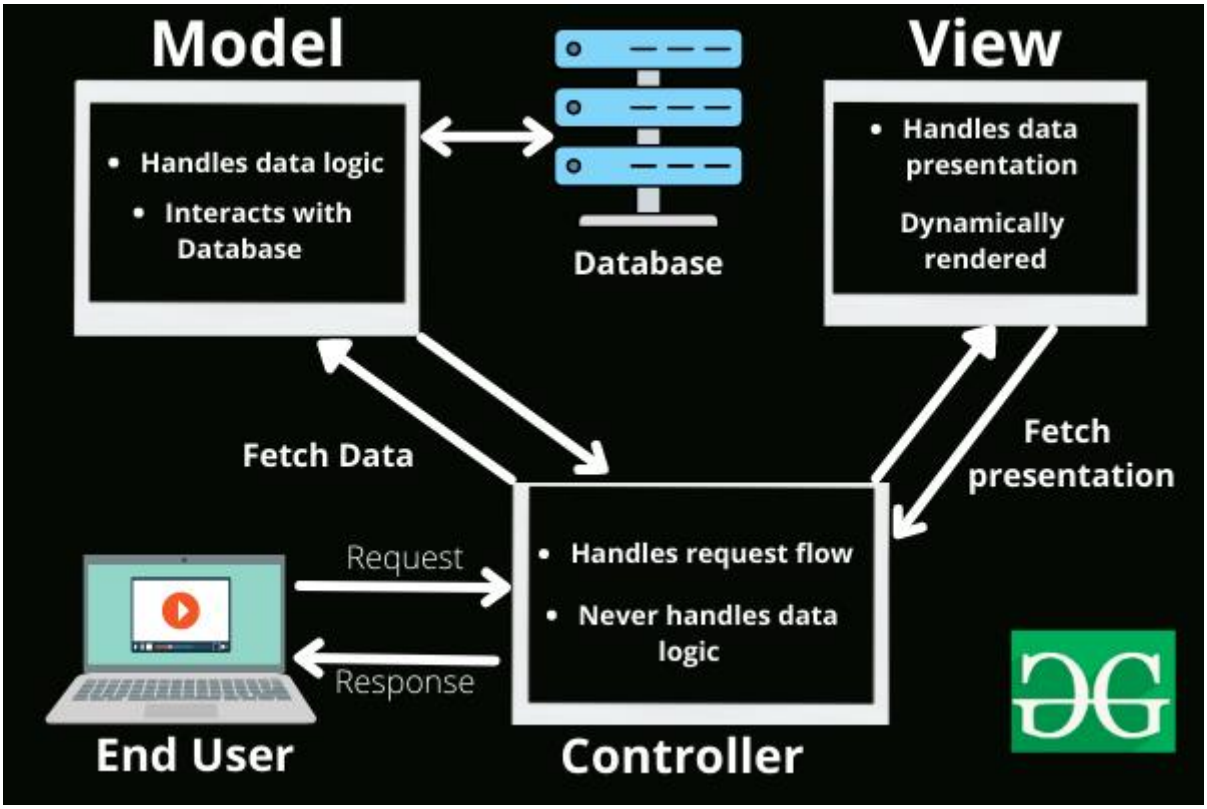
Technical name: bill of materials

Spring strath➔ spring integrates all other frameworks like hibernate and struts.

Spring features:

1. Pojo - Plain Old Java Object===class p== variables➔ getter setter methods
2. Dependency injection
3. Rest API➔ REpresentational State Transfer API
4. ,
5. Security

What is MVC ?
Design patters:



What is MVT?

What is REST API?
httpREquest-user name, password

response =➔xml/json format

A REST API (REpresentational State Transfer API) is a type of API that allows communication between different systems over the internet, using HTTP requests and responses to access and manage resources, often in JSON format.

Database➔ CREATE READ UDDATE DELETE
C=POST,R=GET,U=PUT, DE=DELETE➔ IS HTTP METHODS

6. WHAT Dependency injection?
   ➔DESIGN PATTERN=IDEA
   Namma pannathu illam console applications

What is object graph in java?

=== Architects senior staffs

1) Avoids tight coupling
2) @componet, A=@Autowired , @Qualifier
3) Unit Testing will be easier one

Tools:(IDES)

1. Spring Tool Suite
2. Intellij Idea
3. **Initializr** generates **spring** boot project with just what you need to start quickly!

Web servers in java EX: google ect…
Tomcat server setup,
Aphache http server

Etha the auto-configuration ➔ war ,jar no install

Separate web server is not needed for s[romgboot
No war files Configuration and management

WAR files/ jAR files== bill of materials

MVN==

Maven Repository: Search/Browse/Explore

COMPANY==PROJECT ➔ MAVEN
ORACLE
MYSQL
HOW TO CONNECT==JAR FILES
GERMAN WORD= KNOWLEDGE ACCUMULATER
WHAT is maven?

Maven builds a project using its project object model (POM) and a set of plugins. That is xml file

https://repo1.maven.org/maven2
https://mvnrepository.com/

1) Group Id➔ package name
2) Artifact Id➔ project
3) Version ID

---

1. gives default project structure
2. dependence auto download
3. Auto complie
4. Server starting, stopping

```
package com.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication
public class WebProjectApplication {

 public static void main(String[] args) {
 ConfigurableApplicationContext context=
 SpringApplication.run(WebProjectApplication.class, args);
        //System.out.println("weclome to java");
        //Home ho= new Home();
        //ho.connect();
 Home h=context.getBean(Home.class);
 h.connect();
 Home h1=context.getBean(Home.class);
 h1.connect();
 }

}
```

```
package com.Crudproject;


import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ConfigurableApplicationContext;


@SpringBootApplication

public class WebApplicationCrudApplication {


       public static void main(String[] args) {

              ConfigurableApplicationContext context=
SpringApplication.run(WebApplicationCrudApplication.class, args);

              Home ho1= new Home();

              ho1.connect();

              Home ho2=new Home();

              ho2.connect();

       Home h1=context.getBean(Home.class);

       h1.connect();

       Home h2=context.getBean(Home.class);

       h2.connect();

       System.out.println(h1 == h2); // true

    //System.out.println(h2 == h3); // true

       System.out.println(h1.hashCode());

       System.out.println(h2.hashCode());

       System.out.println(ho1.hashCode());
```

```
        System.out.println(ho2.hashCode());



        }



}package com.Crudproject;

import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component
//@Scope(value="prototype")
public class Home {
        public Home()
        {
                System.out.println("home to home");
        }

        private int id;
        private String name;
        private String home_name;
        public int getId() {
                return id;
        }
        public void setId(int id) {
                this.id = id;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        public String getHome_name() {
                return home_name;
        }
        public void setHome_name(String home_name) {
                this.home_name = home_name;
        }
        public void connect()
        {
                System.out.println("internet connected!");
        }

}
```

Why to give SpringApplication.*run()*

1. Sets up default configuration
2. Starts spring application context
3. Starts tomcat server, stops

4. Class path scan

What is Singleton Scope in Spring?

In Spring, **singleton scope** means that only **one instance** of a bean is created and **shared across the entire application context**.

By default, all Spring beans are **singleton-scoped**, meaning:

- The **first time** you request the bean, Spring **creates** an object.
- Any **subsequent requests** return the **same** object instead of creating a new one.

Example: Singleton Scope in Spring

Let's see this with an example.

1 Singleton Bean Example

```
java
CopyEdit
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.stereotype.Component;

@Component // Mark this class as a Spring bean
class Home {
    public Home() {
        System.out.println("Home object created!");
    }
}

@SpringBootApplication
public class Main {
    public static void main(String[] args) {
        ApplicationContext context = SpringApplication.run(Main.class, args);

        // Getting the bean multiple times
        Home h1 = context.getBean(Home.class);
        Home h2 = context.getBean(Home.class);
        Home h3 = context.getBean(Home.class);

        // Check if all instances are the same
        System.out.println(h1 == h2); // true
        System.out.println(h2 == h3); // true
    }
}
```

Output:

```
csharp
CopyEdit
Home object created!
true
true
```

why to give SpringApplication.run()==interview queusitosn
➔

✅ **Only one instance of `Home` is created, and all requests return the same object.**

---

2 Singleton Scope with `@Scope` (Explicit Declaration)

Although singleton is the **default scope**, you can explicitly define it using `@Scope("singleton")`:

```
java
CopyEdit
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component
@Scope("singleton") // Explicitly specifying singleton scope
class Home {
    public Home() {
        System.out.println("Home object created!");
    }
}
```

The behavior remains **the same**—only one object is created for the entire application.

---

3 Singleton Scope vs Prototype Scope

If you **want a new object every time**, use `@Scope("prototype")` instead of `singleton`.

Prototype Example:

```
java
CopyEdit
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Component;

@Component
@Scope("prototype") // Creates a new object every time
class Home {
    public Home() {
        System.out.println("Home object created!");
    }
}
```

Now, when you get the bean multiple times, it **creates a new instance** each time.

Output:

```
csharp
CopyEdit
Home object created!
Home object created!
Home object created!
```

✅ **Unlike singleton, `prototype` creates a new object each time `context.getBean()` is called.**

---

4 When to Use Singleton Scope?

Scenario	Singleton Scope?
Stateless objects (e.g., services, controllers)	✅ Yes
Heavy objects that should not be created multiple times	✅ Yes
Database connections or configuration classes	✅ Yes
Stateful objects (storing session data, user-specific data)	❌ No (Use prototype)

Conclusion

- `@Scope("singleton")` ensures **only one object is created**, which is **shared across the entire application**.
- **By default, Spring beans are singleton** unless explicitly changed.
- Use `@Scope("prototype")` if you need **a new object every time**.

What is Autowiring in spring boot

```
<dependency>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-devtools</artifactId>
                    <scope>runtime</scope>
            </dependency>
```

class Home{
jio fibre= new jio();
jio.connect();
}
	Class Airtel
	{
	connect();
	}
		Class Jio
		{
		connect();
		}
		Hard coding

Fa

Class Home
{
 	
}
	Class Airtel implements NetConnection
	{
	}
		Class jio implements Netcoonecion
		{
		}

Ftw

Class Netconnecton
{
connection();
}
	Dynamic binding

Spring boot

Class home{
Netconnection fibre;
Home h =new home();
}
	Interface Netconnection
	{
	
	}
		Class jio implements Netconnections
		{
		}

What is a Java Annotation?

Java Annotations are special **metadata tags** in Java that begin with `@`, used to provide **instructions** to the compiler or a framework.

Example:

```
java
CopyEdit
@Override
public String toString() {
    return "Hello";
}
```

Here, `@Override` tells the compiler that this method **overrides** a method from its superclass.

Why Use Annotations?

- ✅ Give **extra meaning** to your code without changing its logic.
- ✅ Help frameworks like **Spring**, **Hibernate**, **JUnit**, etc., understand how to handle your classes and methods.
- ✅ Reduce boilerplate code (e.g., no need to write XML configs).

Common Built-in Java Annotations

Annotation	Description
`@Override`	Ensures you're overriding a method from a superclass.
`@Deprecated`	Marks a method/class as **old** or no longer recommended.
`@SuppressWarnings`	Tells compiler to **ignore specific warnings**.
`@FunctionalInterface`	Ensures an interface has only **one abstract method** (used with lambdas).

Spring Framework Annotations (Very Important in Real Projects)

Annotation	Purpose
`@Component`	Marks a class as a Spring-managed bean
`@Service`	Specialized version of `@Component` for business logic
`@Repository`	For DAO (data access) layer

Annotation	Purpose
`@Controller`	Marks a class as a Spring MVC controller
`@Autowired`	Tells Spring to **inject** dependencies automatically
`@RequestMapping`	Maps HTTP requests to handler methods

How to Create a Custom Annotation

```
java
CopyEdit
import java.lang.annotation.*;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface MyAnnotation {
    String value();
}
```

- `@Retention(RUNTIME)` — annotation is available at **runtime**
- `@Target(METHOD)` — can only be used on **methods**
- `@interface` — defines your own annotation

Usage:

```
java
CopyEdit
@MyAnnotation("custom info")
public void doSomething() {
    // ...
}
```

Summary

Feature	Description
What	Metadata attached to code
Syntax	Starts with `@`
Uses	Framework config, validation, lifecycle management
Benefits	Clean code, easy configuration, reduced XML
Examples	`@Override`, `@Autowired`, `@Controller`

🔍 Code:

```
java
CopyEdit
@GetMapping("/child-form")
public String showForm(Model model) {
    model.addAttribute("child", new Child());
    return "childs_form";
}
```

✅ Line-by-line Explanation:

`@GetMapping("/child-form")`

- This maps the URL `http://localhost:8080/child-form` to this method.
- It means: "When a GET request comes to `/child-form`, call this method."

`public String showForm(Model model)`

- This is the controller method that returns the **view name**.
- `Model model`: This is used to pass data from the controller to the HTML (Thymeleaf) page.

`model.addAttribute("child", new Child());`

- This creates an **empty `Child` object** and gives it the name `"child"` for the form.
- Now in your HTML form, you can use:

  ```
  html
  CopyEdit
  th:object="${child}"
  th:field="*{name}"
  ```

  This allows Spring to **bind the form fields** to the `Child` class automatically.

`return "childs_form";`

- This tells Spring to look for a Thymeleaf HTML template:

```
css
CopyEdit
src/main/resources/templates/childs_form.html
```

- No need to write `.html` — Spring automatically adds that.

💡 Example flow:

1. User visits → `http://localhost:8080/child-form`
2. Spring calls this method
3. The method sends a blank `Child` object to the HTML form
4. Thymeleaf renders `childs_form.html` using that object
5. User fills the form and submits data

Great! Let's break down this **Spring Boot controller code** step by step, like a story — so you fully understand what happens behind the scenes when a user visits your form and submits it. 👇

🚀 Code Breakdown

✅ 1. `showForm(...)` – Show the HTML form

```
java
CopyEdit
public String showForm(Model model) {
    model.addAttribute("child", new Child());
    return "childs_form";
}
```

What happens here:

- 📎 This method is mapped (via `@GetMapping`) to a URL like `/child-form` (assuming you set it).
- 🧱 `new Child()` = You're creating an **empty object** of your `Child` class.
- 📦 `model.addAttribute("child", new Child())` → Adds the object to the **model**, so Thymeleaf can use it in the form.
- 🎯 `return "childs_form"` → Spring will load the file:

```
css
CopyEdit
src/main/resources/templates/childs_form.html
```

Why?

You need to give Thymeleaf an empty `Child` object to bind input fields in the form. It connects like this:

```
html
CopyEdit
<form th:object="${child}" ...>
    <input th:field="*{name}" />
</form>
```

✅ 2. `submitChild(...)` – Handle the form submission

```
java
CopyEdit
@PostMapping("/submit-child")
public String submitChild(@ModelAttribute Child child, Model model) {
    model.addAttribute("child", child);
    return "child_success";
}
```

What happens here:

- 📎 This method is triggered when the form is submitted using:

  ```
  html
  CopyEdit
  <form th:action="@{/submit-child}" method="post">
  ```

- 📥 `@ModelAttribute Child child` → Spring **automatically fills** the `child` object with values from the form. Example:
  - `<input th:field="*{name}">` → goes into `child.setName(...)`
  - `<input th:field="*{books}">` → goes into `child.setBooks(...)`
- 📦 `model.addAttribute("child", child)` → Passes the submitted object to the next page (`child_success.html`), so you can **display the data back** to the user.
- 🎯 `return "child_success"` → Loads the Thymeleaf template:

  ```
  css
  CopyEdit
  src/main/resources/templates/child_success.html
  ```

🧠 Real-world example:

1. User opens this URL:

```
bash
CopyEdit
http://localhost:8080/child-form
```

2. HTML Form (childs_form.html) appears:

```
html
```

```
CopyEdit
<form th:action="@{/submit-child}" th:object="${child}" method="post">
    Name: <input th:field="*{name}" />
    Books: <input th:field="*{books}" />
    College: <input th:field="*{collegename}" />
    <button type="submit">Submit</button>
</form>
```

3. User submits form → hits `/submit-child`

4. `submitChild()` receives the form, and data is displayed in `child_success.html`:

```
html
CopyEdit
<p>Name: <span th:text="${child.name}"></span></p>
<p>Books: <span th:text="${child.books}"></span></p>
<p>College: <span th:text="${child.collegename}"></span></p>
```

✅ Summary of Flow:

Step	Action
1	Controller gives a blank `Child` object to the form
2	Thymeleaf binds form fields to that object
3	User submits the form
4	Spring auto-fills `Child` with form values using `@ModelAttribute`
5	Controller sends that data to the success page

🧠 In Thymeleaf, these symbols have special meanings:

Symbol	Used for	Meaning
`${...}`	Variable Expressions	Access **Java data/objects** from the model
`@{...}`	URL Expressions	Build **dynamic URLs/links**
`:`	Inside `@{}`	**Insert variables** into URLs
`*{...}`	Selection Expressions	Bind **form fields** to a specific object

---

Employeemanagement project

Pojo class➔

```
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
private Long Id;
      private String name;
private String Department;
private Double salary;
```

1.core java or console applications
2.jdbc
3.framework and library
  books books
spring and spring boot, hipernet ,sturts
angular,react js,node js
python=== flask, django,botelpy, ect

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

Spring Boot makes it easy to create stand-alone, production-grade Spring based
Applications that you can "just run".
opinionated view