Object Oriented Programming with Java

AKTU

Unit :- 5

BCS-403

Spring Framework and Spring Boot

Spring Framework: Spring Core Basics-Spring Dependency Injection concepts, Spring Inversion of Control, AOP, Bean Scopes- Singleton, Prototype, Request, Session, Application, Web Socket, Auto wiring, Annotations, Life Cycle Call backs, Bean Configuration styles

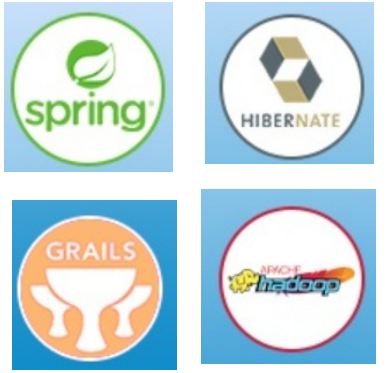
Spring Boot: Spring Boot Build Systems, Spring Boot Code Structure, Spring Boot Runners, Logger, BUILDING RESTFUL WEB SERVICES, Rest Controller, Request Mapping, Request Body, Path Variable, Request Parameter, GET, POST, PUT, DELETE APIs, Build Web Applications

What is a Java Framework?

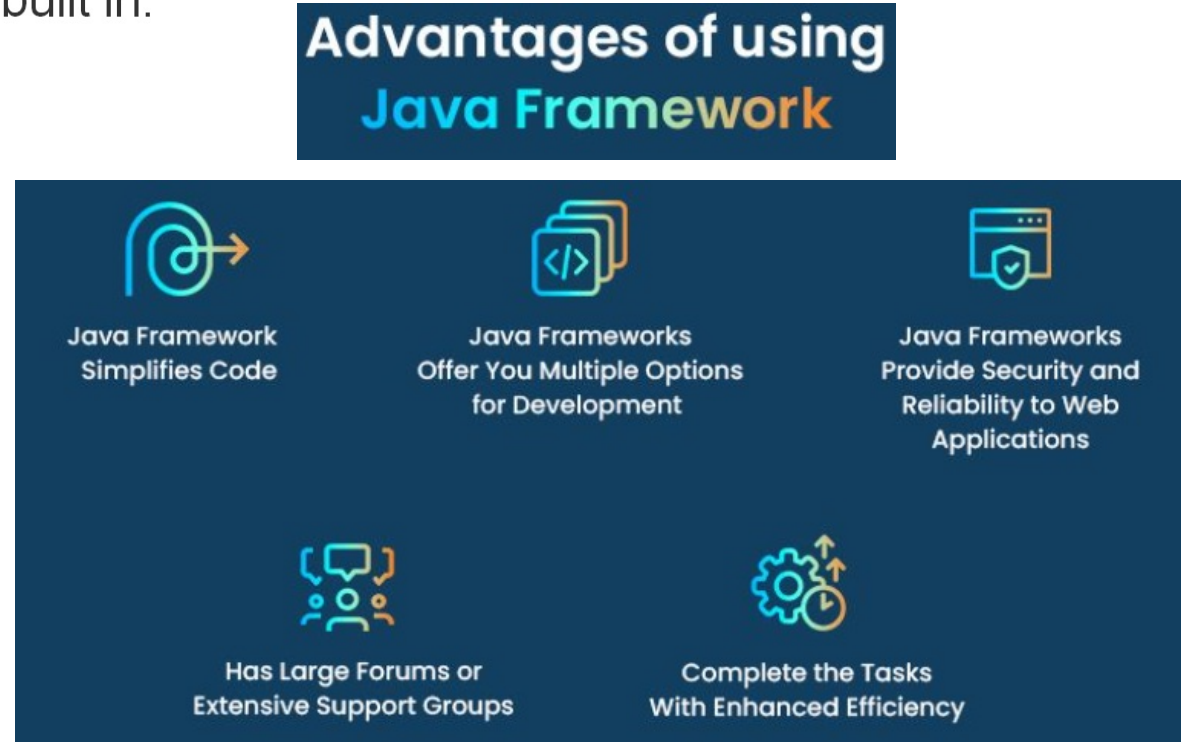
A Java framework is **the body of predefined codes used by programmers to develop applications on the web**. These frameworks in Java development are classes and functions that control hardware, process input, and communicate with system applications.

Why Use Java Frameworks?

It's time-consuming to build big Java applications from the ground up, but integrating a framework into your stack can speed up the process. Data retrieval, security, and privacy are just a few of the numerous app-wide features that are essential. To avoid having to constantly start from scratch with new projects, the finest Java frameworks already have many of these functionalities built in.



Top Java Frameworks



SPRING FRAMEWORK

Spring framework makes the easy development of JavaEE application.

It is helpful for beginners and experienced persons.

It was developed by Rod Johnson in 2003

Spring is a Dependency Injection framework to make java application loosely coupled.

Spring is a *lightweight* framework. It can be thought of as a *framework of frameworks* because it provides support to various frameworks such

as Struts, Hibernate, Tapestry, EJB, JSF, etc.

The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems. The Spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC etc.

Advantages of Spring Framework

There are many advantages of Spring Framework. They are as follows:

1) Predefined Templates

Spring framework provides templates for JDBC, Hibernate, JPA etc. technologies. So there is no need to write too much code. It hides the basic steps of these technologies.

2) Loose Coupling

The Spring applications are loosely coupled because of dependency injection.

3) Easy to test

The Dependency Injection makes easier to test the application. The EJB or Struts application require server to run the application but Spring framework doesn't require server.

4) Lightweight

Spring framework is lightweight because of its POJO implementation. The Spring Framework doesn't force the programmer to inherit any class or implement any interface. That is why it is said non-invasive.

5) Fast Development

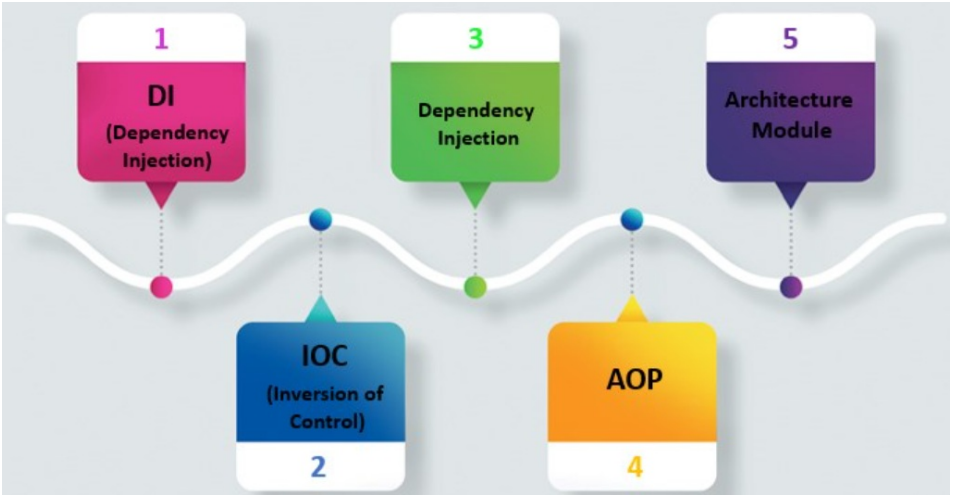
The Dependency Injection feature of Spring Framework and it support to various frameworks makes the easy development of JavaEE application.

6) Powerful abstraction

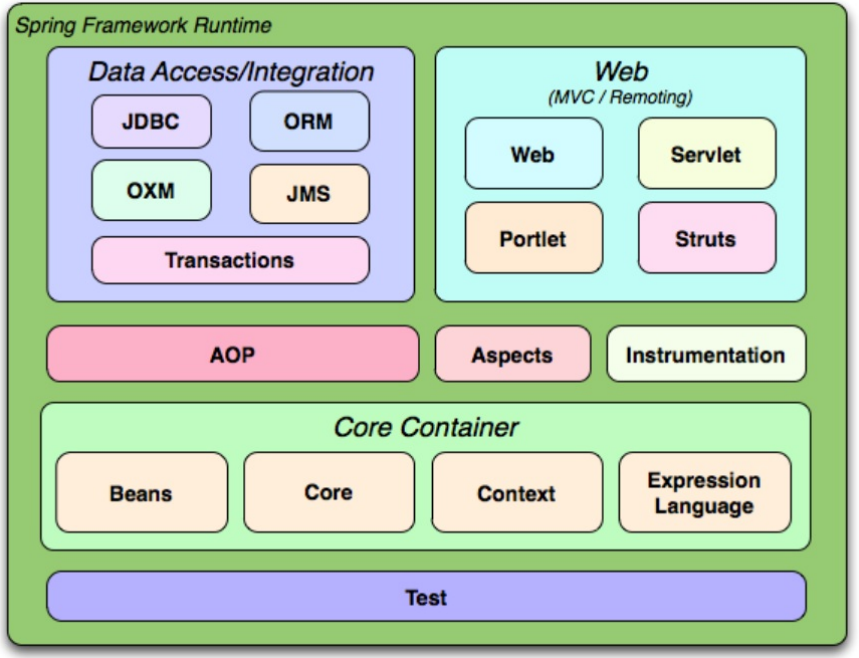
It provides powerful abstraction to JavaEE specifications such as JMS, JDBC, JPA and JTA.

7) Declarative support

It provides declarative support for caching, validation, transactions and formatting.



Spring Architecture



Dependency Injection

It is design pattern

Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application.

Dependency Injection makes our programming code loosely coupled.

```
class Ramu
{

    Geeta ob;

    public void doWork()
    {

    }

}
```

```
class Geeta
{

    public void doWork()
    {

    }

}
```

Every Java-based application has a few objects that work together to present what the end-user sees as a working application. When writing a complex Java application, application classes should be as independent as possible of other Java classes to increase the possibility to reuse these classes and to test them independently of other classes while unit testing. Dependency Injection (or sometime called wiring) helps in gluing these classes together and at the same time keeping them independent.

Dependency Injection is the main functionality provided by Spring IOC(Inversion of Control). The Spring-Core module is responsible for injecting dependencies through either Constructor or Setter methods. The design principle of Inversion of Control emphasizes keeping the Java classes independent of each other and the container frees them from object creation and maintenance. These classes, managed by Spring, must adhere to the standard definition of Java-Bean. Dependency Injection in Spring also ensures loose coupling between the classes.

There are two types of Spring Dependency Injection.

1. Constructor-based dependency injection

Constructor-based DI is accomplished when the container invokes a class constructor with a number of arguments, each representing a dependency on the other class.

2. Setter-based dependency injection

Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

You can mix both, Constructor-based and Setter-based DI but it is a good rule of thumb to use constructor arguments for mandatory dependencies and setters for optional dependencies.

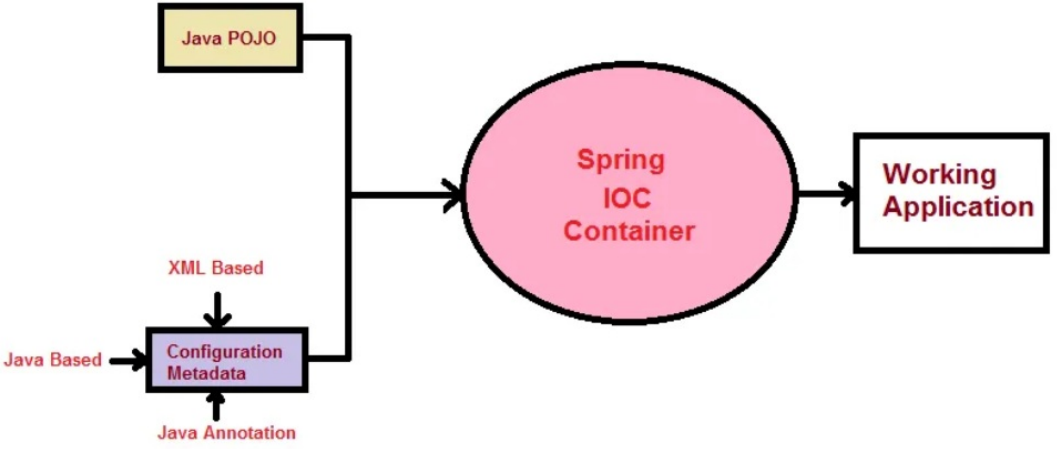
The code is cleaner with the DI principle and decoupling is more effective when objects are provided with their dependencies. The object does not look up its dependencies and does not know the location or class of the dependencies, rather everything is taken care by the Spring Framework.

Spring IoC (Inversion of Control)

Inversion of Control is a principle in software engineering which transfers the control of objects or portions of a program to a container or framework. We most often use it in the context of object-oriented programming.

Spring IoC (Inversion of Control) 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. These objects are called Beans. Since the Controlling of Java objects and their lifecycle is not done by the developers, hence the name Inversion Of Control. The followings are some of the main features of Spring IoC,

- Creating Object for us,
- Managing our objects,
- Helping our application to be configurable,
- Managing dependencies



AOP

Aspect oriented programming(AOP) as the name suggests uses aspects in programming.

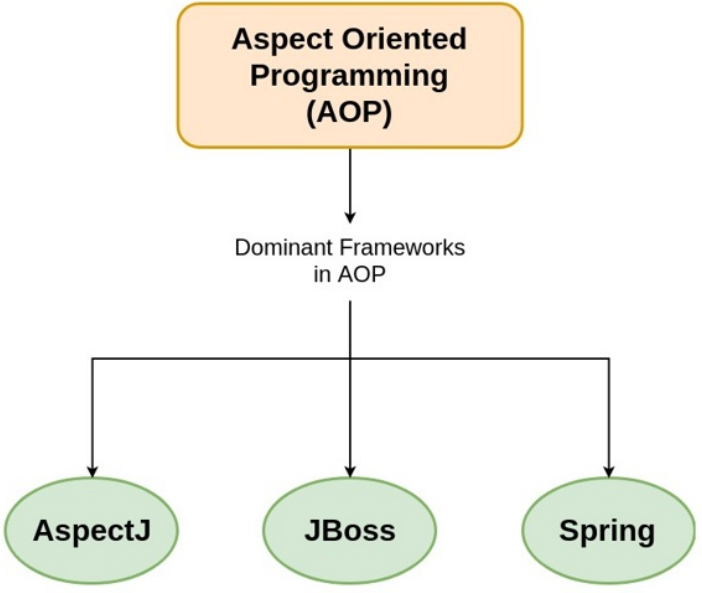
It can be defined as the breaking of code into different modules, also known as modularisation, where the aspect is the key unit of modularity. Aspects enable the implementation of crosscutting concerns such as- transaction, logging not central to business logic without cluttering the code core to its functionality. It does so by adding additional behaviour that is the advice to the existing code.

For example- Security is a crosscutting concern, in many methods in an application security rules can be applied, therefore repeating the code at every method, define the functionality in a common class and control were to apply that functionality in the whole application.

Dominant Frameworks in AOP:

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

Let's have a look the three **dominant frameworks in AOP**



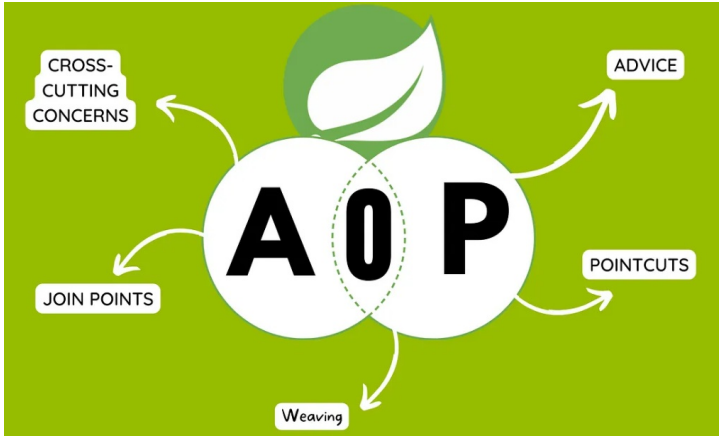
- **AspectJ:** It is an extension for Java programming created at **PARC research centre**. It uses Java like syntax and included IDE integrations for displaying crosscutting structure. It has its own compiler and weaver, on using it enables the use of full AspectJ language.
- **JBoss:** It is an open source Java application server developed by JBoss, used for Java development.
- 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.

Currently, **AspectJ libraries with Spring framework** are dominant in the market, therefore let's have an understanding of how Aspect-oriented programming works with Spring.

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 based mechanism i.e. it creates a proxy Object which will wrap around the original object and will take up the advice which is relevant 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 objects are used to enrich the Original behaviour of the real object.

A **cross-cutting concern** is a concern that can affect the whole application and should be centralized in one location in code as possible, such as transaction management, authentication, logging, security etc.



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 runtime time. Spring AOP does weaving at runtime.

BEAN SCOPE

***Bean Definition**: In Spring, the objects 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*

Bean Scopes refers to the lifecycle of Bean that means when the object of Bean will be instantiated, how long does that object live, and how many objects will be created for that bean throughout. Basically, it controls the instance creation of the bean and it is managed by the spring container.

The following are the different scopes provided for a bean:

1. **Singleton:** Only one instance will be created for a single bean definition per Spring IoC container and the same object will be shared for each request made for that bean.
2. **Prototype:** A new instance will be created for a single bean definition every time a request is made for that bean.
3. **Request:** A new instance will be created for a single bean definition every time an HTTP request is made for that bean. But Only valid in the context of a web-aware Spring ApplicationContext.
4. **Session:** Scopes a single bean definition to the lifecycle of an HTTP Session. But Only valid in the context of a web-aware Spring ApplicationContext.
5. **Global-Session:** Scopes a single bean definition to the lifecycle of a global HTTP Session. It is also only valid in the context of a web-aware Spring ApplicationContext.

Singleton Scope:

If the scope is a singleton, then only one instance of that bean will be instantiated per Spring IoC container and the same instance will be shared for each request. That is when the scope of a bean is declared singleton, then whenever a new request is made for that bean, spring IOC container first checks whether an instance of that bean is already created or not. If it is already created, then the IOC container returns the same instance otherwise it creates a new instance of that bean only at the first request. By default, the scope of a bean is a singleton.

Prototype Scope:

If the scope is declared **prototype**, then spring IOC container will create a new instance of that bean every time a request is made for that specific bean. A request can be made to the bean instance either programmatically using **getBean()** method or by XML for Dependency Injection of secondary type. Generally, we use the prototype scope for all beans that are stateful, while the singleton scope is used for the stateless beans.

Request Scope

In `request` scope, container creates a new instance for each and every HTTP request. So, if the server is currently handling 50 requests, then the container can have at most 50 individual instances of the bean class. Any state change to one instance, will not be visible to other instances. **A bean instance is destructed as soon as the request is completed**.

Session Scope

In `session` scope, the application context creates a new instance for each and every HTTP session. So, if the server has 20 active sessions, then the container can have at most 20 individual instances of the bean class. All HTTP requests within a single session lifetime will have access to the same single bean instance in that session scope.
Any state change to one instance will not be visible to other instances. **An instance is destructed as soon as the session ends.**

Scope	Description
Singleton	(Default)Only one single instance will be created
Prototype	Creates any number of instances from a single bean configuration
Request	Scope of the bean instance will be limited to the Request life cycle
Session	Limited to session
Global session	Limited to global session-Portlet context.

```
<bean name ="student" class ="Student" scope ="prototype"/>
```

Bean Scopes

Application Scope

In `application` scope, the container creates one instance per web application runtime. It is almost similar to `singleton` scope with only two differences i.e.

1. The `application` scoped bean is singleton per `ServletContext`, whereas `singleton` scoped bean is singleton per `ApplicationContext`. Please note that **there can be multiple application contexts within a single application**.
2. The `application` scoped bean is visible as a `ServletContext` attribute.

WebSocket Scope

The WebSocket Protocol enables two-way communication between a client and a remote host that has opted-in to communicate with the client. WebSocket Protocol provides a single TCP connection for traffic in both directions. This is especially useful for multi-user applications with simultaneous editing and multi-user games.

In this type of web application, HTTP is used only for the initial handshake. The server can respond with HTTP status 101 (switching protocols) if it agrees – to the handshake request. If the handshake succeeds, the TCP socket remains open, and both the client and server can use it to send messages to each other.

When first accessed, *WebSocket* scoped beans are stored in the *WebSocket* session attributes. The same bean instance is then returned during the entire *WebSocket* session.

Please note that `websocket` scoped beans are typically singleton and live longer than any individual WebSocket session.

Must SUBSCRIBE

I Tech World (AKTU)

Ritik ThAkur

B.Tech (IT)

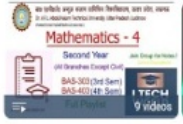
Instagram

@ritik_thakur00

HOME | VIDEOS | SHORTS | PLAYLISTS | COMMUNITY | MEMBERSHIP | CHANNELS | ABOUT

Created playlists

Sort by



Mathematics 4 (AKTU) Acc. to New Syllabus 2nd Year Al...

View full playlist

Compiler Design - CD (AKTU) 3rd Year B.Tech CS/IT (KCS...

Updated today

View full playlist

Discrete Structures & Theory of Logic - DSTL (AKTU) 2nd...

Updated today

View full playlist

Cyber Security (AKTU) - 2nd Year B.Tech All Branches 3r...

Updated today

View full playlist

Operating System One Shot Videos (AKTU) - 2nd Year

View full playlist

Microprocessor One Shot

Important Topics 4th Year

AKTU Updates !!

Environment and Ecology

TECH !!

SUBSCRIBE

Autowiring in Spring

Autowiring in 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 to as **Autowiring in Spring**. To enable Autowiring in the Spring application we should use @Autowired annotation. Autowiring in Spring internally uses constructor injection. 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

Modes	Description
No	This mode tells the framework that auto wiring is not supposed to be done. It is the default mode used by Spring.
byName	It uses the name of the bean for injecting dependencies.
byType	It injects the dependency according to the type of bean.
Constructor	It injects the required dependencies by invoking the constructor.
Autodetect	The autodetect mode uses two other

Advantage of Autowiring

It requires the **less code** because we don't need to write the code to inject the dependency explicitly.

Disadvantage of Autowiring

No control of programmer.

Annotations

Annotations are a form of metadata that provides data about a program. Annotations are used to provide supplemental information about a program. It does not have a direct effect on the operation of the code they annotate. It does not change the action of the compiled program. So, we are going to discuss what are the main types of annotation that are available in the spring framework .

Use of java annotations

Java annotations are mainly used for the following:

- Compiler instructions
- Build-time instructions
- Runtime instructions

Compiler instructions: Java provides the 3 in built annotations which are used to give certain instructions to the compiler. Java in built annotation are @Deprecated, @Override & @SuppressWarnings.

Build-time instructions: Java annotations can be used for build time or compile time instructions. These instructions can be used by the build tools for generating source code, compiling the source, generating XML files, packaging the compiled code and files into a JAR file etc.

Runtime instructions: Normally, Java annotations are not present in your Java code after compilation. However, we can define our own annotations that can be available at runtime. These annotations can be accessed using Java Reflection.

Java annotations basics:

A java annotation always starts with the symbol @ and followed by the annotation name. The @symbol signals the compiler that this is an annotation.

Syntax:

```
@AnnotationName
```

Example:

```
@Entity
```

Here @ symbol signals the compiler that this is an annotation and the Entity is the name of this annotation.An annotation can contain zero, one or multiple elements. We have to set values for these elements. **Example:**

```
@Entity(tableName = "USERS")
```

Where we can use annotations?

We can use java annotations above classes, interfaces, methods, fields and local variables. Here is an example annotation added above a class definition:

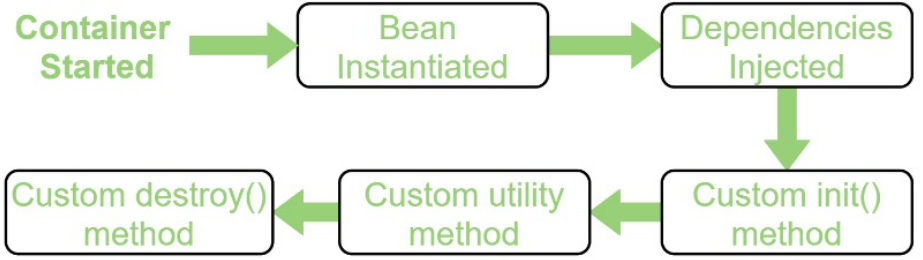
```
@Entity
 public class Users {
}
```

Spring Bean Life Cycle and Callbacks

The lifecycle of any object means when & how it is born, how it behaves throughout its life, and when & how it dies. Similarly, the bean life cycle refers to when & how the bean is instantiated, what action it performs until it lives, and when & how it is destroyed. Now, we will discuss the life cycle of the bean.

Bean life cycle is managed by the spring container. When we run the program then, first of all, the spring container gets started. After that, the container creates the instance of a bean as per the request, and then dependencies are injected. And finally, the bean is destroyed when the spring container is closed. Therefore, if we want to execute some code on the bean instantiation and just after closing the spring container, then we can write that code inside the custom **init()** method and the **destroy()** method.

The following image shows the process flow of the bean life cycle.



Bean Life Cycle Process Flow

Note: We can choose a custom method name instead of **init()** and **destroy()**. Here, we will use init() method to execute all its code as the spring container starts up and the bean is instantiated, and destroy() method to execute all its code on closing the container.

1.Spring bean life cycle involves initialization and destruction callbacks and Spring bean aware classes.
2. Initialization callback methods execute after dependency injection is completed. Their purposes are to check the values that have been set in bean properties, perform any custom initialization or provide a wrapper on original bean etc. Once the initialization callbacks are completed, bean is ready to be used.
3. When IoC container is about to remove bean, destruction callback methods execute. Their purposes are to release the resources held by bean or to perform any other finalization tasks.
4. When more than one initialization and destructions callback methods have been implemented by bean, then those methods execute in certain order.

BEAN CONFIGURATION STYLE

Spring Framework provides three ways to configure beans to be used in the application.

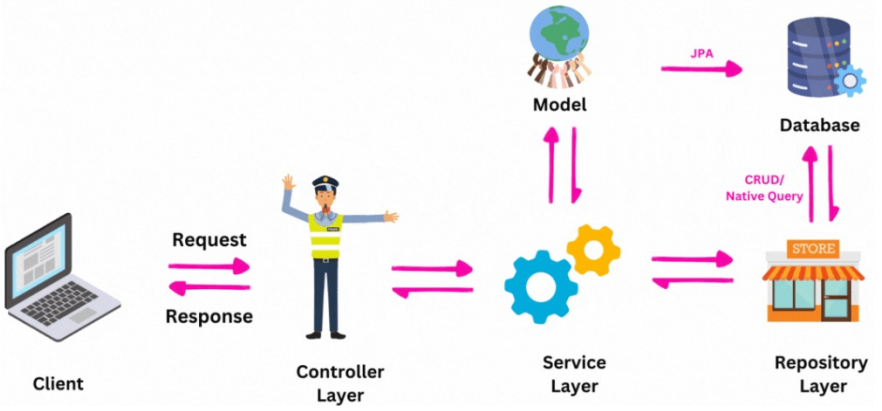
1. **Annotation Based Configuration** - By using @Service or @Component annotations. Scope details can be provided with @Scope annotation.

2. **XML Based Configuration** - By creating Spring Configuration XML file to configure the beans. If you are using Spring MVC framework, the xml based configuration can be loaded automatically by writing some boiler plate code in web.xml file.

3. **Java Based Configuration** - Starting from Spring 3.0, we can configure Spring beans using java programs. Some important annotations used for java based configuration are @Configuration, @ComponentScan and @Bean.

Spring Boot

Spring Boot is most commonly used and contains all Spring Framework features. Nowadays Spring Boot is becoming the best choice for Java developers to build rapid Spring applications. When Java Developers use Spring Boot for developing applications then they will focus on the logic instead of struggling with the configuration setup environment of the application.



In Spring Boot, choosing a build system is an important task. We recommend Maven or Gradle as they provide a good support for dependency management. Spring does not support well other build systems.

Dependency Management

Spring Boot team provides a list of dependencies to support the Spring Boot version for its every release. You do not need to provide a version for dependencies in the build configuration file. Spring Boot automatically configures the dependencies version based on the release. Remember that when you upgrade the Spring Boot version, dependencies also will upgrade automatically.

Note – If you want to specify the version for dependency, you can specify it in your configuration file. However, the Spring Boot team highly recommends that it is not needed to specify the version for dependency.

Maven Dependency

For Maven configuration, we should inherit the Spring Boot Starter parent project to manage the Spring Boot Starters dependencies.
We should specify the version number for Spring Boot Parent Starter dependency. Then for other starter dependencies, we do not need to specify the Spring Boot version number.

Gradle Dependency

We can import the Spring Boot Starters dependencies directly into **build.gradle** file. We do not need Spring Boot start Parent dependency like Maven for Gradle.

SPRING BOOT CODE STRUCTURE

There is no specific layout or code structure for Spring Boot Projects. However, there are some best practices followed by developers that will help us too. You can divide your project into layers like service layer, entity layer, repository layer,, etc. You can also divide the project into modules. For example, the parent project has two child modules. The first module is for the data layer and the second module is for the web layer. You can also divide the project into features.

Note: *Avoid Default Package*

It is. because a class is said to be in a **default package** when it does not include a **package** declaration. It is not a best practice to include a class in the default package. It is because Spring scans the classes in packages and sub-packages mentioned in the annotations like **@ComponentScan**, **@EntityScan**, **@SpringBootApplication** etc.

Note: *It is recommended to use Java's package naming conventions with a reverse domain name.*

It is recommended to place the Main Application class in the root package with annotations.
like **@SpringBootApplication** or **@ComponentScan** or **@EnableAuto Configuration**. It allows the Spring to scan all classes in the root package and sub-packages.

Let us discuss two approaches that are typically used by most developers to structure their spring boot projects.

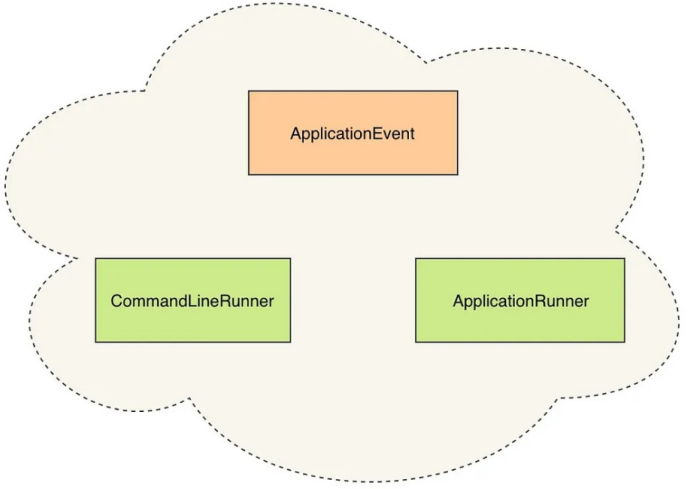
1. Structure by Feature
2. Structure by Layer

Now let us wrap up by acquiring the location of MainApplication.java**.** as in both the structures proposed above we have seen that the **MainApplication.java** is placed in the root package with **@SpringBootApplication** an notation. It s as shown below in as a sample example which is as follows:

```
@SpringBootApplication
public class MyApplication {

    public static void main(String[] args) {
        SpringApplication.run(MyApplication.class, args);
    }

}
```

Spring Boot - Runners

Application Runner and Command Line Runner interfaces lets you to execute the code after the Spring Boot application is started. You can use these interfaces to perform any actions immediately after the application has started.



Application Runner

Application Runner is an interface used to execute the code after the Spring Boot application started.

Command Line Runner

Command Line Runner is an interface. It is used to execute the code after the Spring Boot application started.

LOGGER

Logging in Spring Boot plays a vital role in Spring Boot applications for recording information, actions, and events within the app. It is also used for monitoring the performance of an application, understanding the behavior of the application, and recognizing the issues within the application. Spring Boot offers flexible logging capabilities by providing various logging frameworks and also provides ways to manage and configure the logs.

Why to use Spring Boot – Logging?

A good logging infrastructure is necessary for any software project as it not only helps in understanding what's going on with the application but also to trace any unusual incident or error present in the project. This article covers several ways in which logging can be enabled in a spring boot project through easy and simple configurations. Let's first do the initial setup to explore each option in more depth.

Elements of Logging Framework

- Logger: It captures the messages.
- Formatter: It formats the messages which are captured by loggers.
- Handler: It prints messages on the console, stores them in a file or sends an email, etc.

Java provides several logging frameworks, some of which are:

1. *Logback Configuration logging*
2. *Log4j2 Configuration logging*

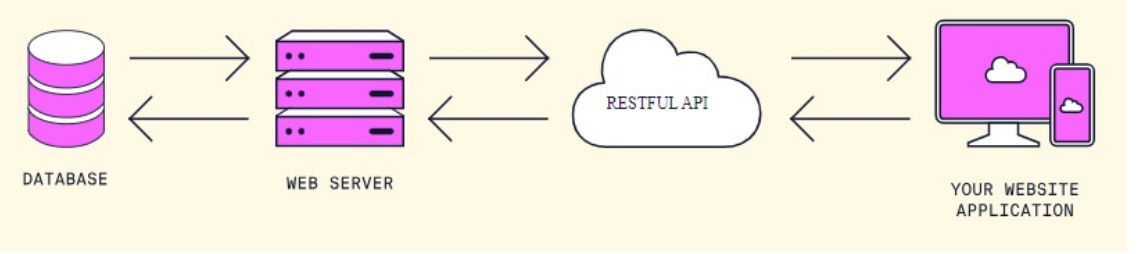
Introduction to RESTful Web Services

REST stands for **REpresentational State Transfer**. It is developed by **Roy Thomas Fielding**, who also developed HTTP. The main goal of RESTful web services is to make web services **more effective**. RESTful web services try to define services using the different concepts that are already present in HTTP. REST is an **architectural approach**, not a protocol.

It does not define the standard message exchange format. We can build REST services with both XML and JSON. JSON is more popular format with REST. The **key abstraction** is a resource in REST. A resource can be anything. It can be accessed through a **Uniform Resource Identifier (URI)**. For example:

The resource has representations like XML, HTML, and JSON. The current state capture by representational resource. When we request a resource, we provide the representation of the resource. The important methods of HTTP are

- **GET:** It reads a resource.
- **PUT:** It updates an existing resource.
- **POST:** It creates a new resource.
- **DELETE:** It deletes the resource.



For example, if we want to perform the following actions in the social media application, we get the corresponding results.

POST /users: It creates a user.

GET /users/{id}: It retrieves the detail of a user.

GET /users: It retrieves the detail of all users.

DELETE /users: It deletes all users.

DELETE /users/{id}: It deletes a user.

GET /users/{id}/posts/post_id: It retrieve the detail of a specific post.

POST / users/{id}/ posts: It creates a post of the user.

HTTP also defines the following standard status code:

- **404:** RESOURCE NOT FOUND
- **200:** SUCCESS
- **201:** CREATED
- **401:** UNAUTHORIZED
- **500:** SERVER ERROR

RESTful Service Constraints

There must be a service producer and service consumer.
The service is stateless.
The service result must be cacheable.
The interface is uniform and exposing resources.
The service should assume a layered architecture.

Advantages of RESTful web services

RESTful web services are **platform-independent**.
It can be written in any programming language and can be executed on any platform.
It provides different data format like **JSON, text, HTML,** and **XML**.
It is fast in comparison to SOAP because there is no strict specification like SOAP.
These are **reusable**.
They are **language neutral**.

Rest Controller

Spring RestController annotation is used to create RESTful web services using Spring MVC. Spring RestController takes care of mapping request data to the defined request handler method. Once response body is generated from the handler method, it converts it to JSON or XML response.

RestController: *RestController is used for making restful web services with the help of the @RestController annotation. This annotation is used at the class level and allows the class to handle the requests made by the client. Let's understand @RestController annotation using an example. The RestController allows to handle all REST APIs such as GET, POST, Delete, PUT requests.*

RestController is very useful when we are working on a real-time REST API Spring Application. This Rest Controller class gives us JSON(JavaScript Object Notation) response as the output. The normal class is annotated with "@RestController" then that class is treated as Rest Controller class in the class

Difference between @Controller and @RestController:

```
@Controller
@ResponseBody
class TestController{
----------------
----------------
}
```

```
@RestController
class TestController{
----------------
----------------
}
```

A normal class annotated with "@Controller" then that class is treated as a Controller class in Spring MVC.
Spring 2.5 introduced the @Controller annotation used for web application return as a view in Spring MVC.
This is the specialization of the "@Component" annotation.
In @Controller we need to use @ReponseBody in every handler method for response output without a view.
Spring 4.0 introduced the "@RestController" for the creation of the Restful web services in the Spring Framework in a simple manner. It is a specialization of the "@Controller" annotation.
In the "@RestController" annotation, we don't need to use the "@ResponseBody" annotation in the handler method. In this "@RestController" we cannot return a view as output.
we can use "@RestController" and "@GetMapping" annotations in the POJO class.

@RestController: This combination of these annotations "@Controller" and "@ResponseBody".This will be used when we are making Rest Apis.
@GetMapping: It will handle requests from HTTP Get Method (Client).

Development Process:

1. Keep eclipse IDE ready
2. Create the Spring Boot Starter Project for this example of the RestController in the Spring Boot(Select Spring Web dependency)
3. Create RestController class
4. Run the Project

When building robust APIs, understanding and appropriately utilizing the HTTP methods GET, POST, PUT, and DELETE is essential. Each method serves a specific purpose and has its own limitations. We will explore these HTTP methods, their characteristics, and discuss their limitations in the context of building robust APIs. Additionally, we will provide Java code examples to demonstrate their usage.

What Are HTTP Methods and Why Are They Important?

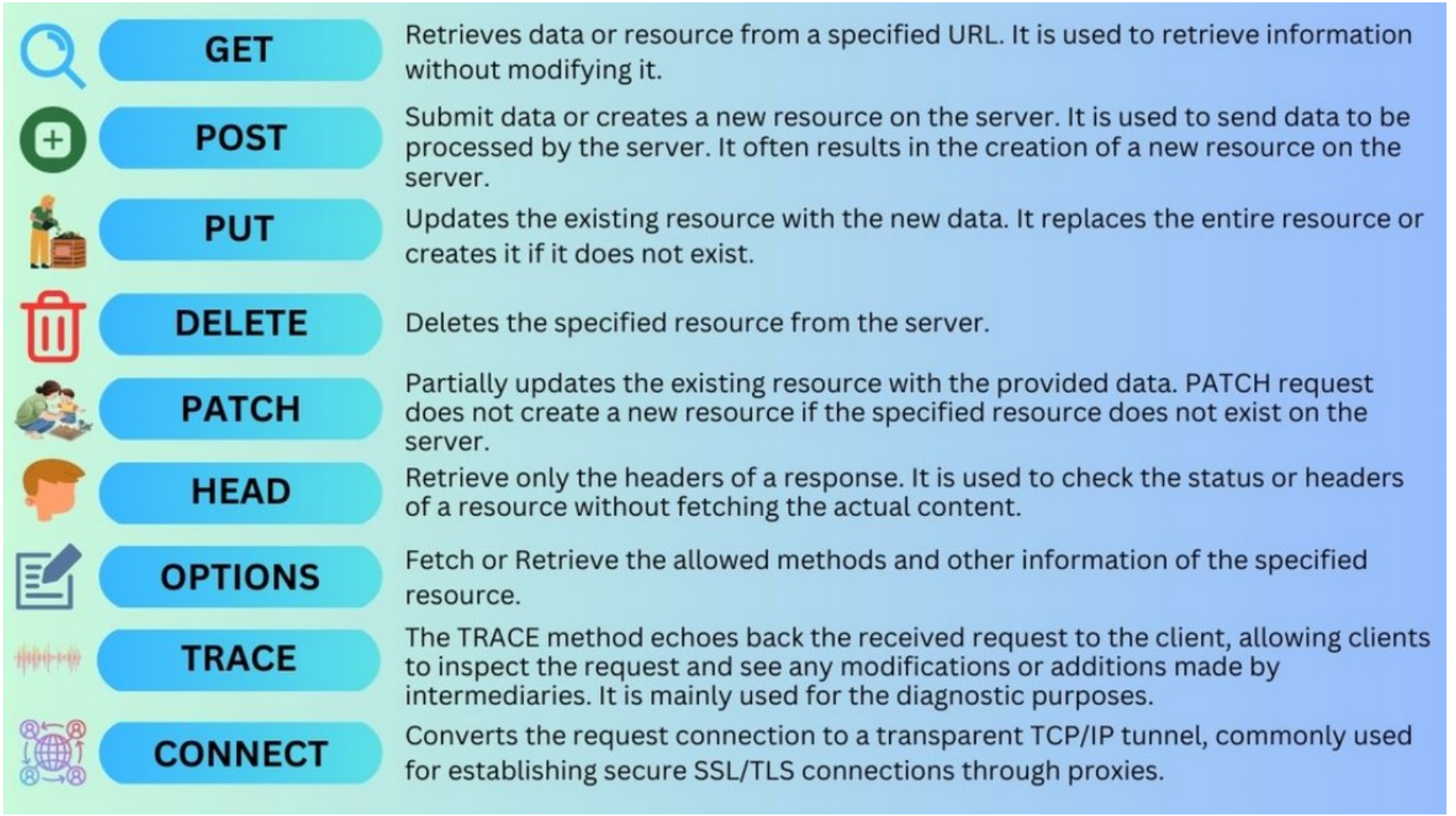
HTTP methods, also known as HTTP verbs, are a set of standardized actions that can be performed on a resource using the Hypertext Transfer Protocol (HTTP). These methods define the intended operation to be performed on the resource and provide a uniform way of interacting with web servers. The most commonly used HTTP methods are GET, POST, PUT, and DELETE, but there are other methods as well, such as PATCH, HEAD, and OPTIONS.

HTTP methods are important for several reasons:

1. **Resource Manipulation**:
2. **Uniform Interface**:
3. **Intent and Semantics**:
4. **Idempotence and Safety**:
5. **RESTful Architecture**:

By understanding and utilizing the appropriate HTTP methods, developers can build robust and well-designed APIs that adhere to the principles of HTTP and REST. This ensures consistency, interoperability, and efficiency in the communication between clients and servers.

Below we will elaborate more on the most commonly used HTTP methods.



GET Method:

The GET method is one of the fundamental HTTP methods used for retrieving data from a server. It is designed to be safe, meaning it should not modify any data on the server. When using the GET method, the client requests a representation of a resource from the server.

Here are some key points to consider when working with the GET method in API development:

1. **Retrieving Data**:
2. **Request Parameters**:
3. **Idempotence:**
4. **Response Codes:**
5. **Limitations**:
   - **Data Length**:
   - **Security Considerations**:

In the given Java code example, the GET request is sent to `https://api.example.com/resource`. The response from the server is then read and stored in a StringBuilder for further processing. If the response code is 200 (HTTP_OK), the response is printed to the console.

while the GET method is efficient for retrieving data, it should not be used for operations that modify the server state. For such operations, other HTTP methods like POST, PUT, or DELETE should be employed.

GET Method:

```
import java.net.HttpURLConnection;
import java.net.URL;
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class GetExample {
    public static void main(String[] args) {
        try {
            URL url = new URL("https://api.example.com/resource");
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            connection.setRequestMethod("GET");

            int responseCode = connection.getResponseCode();
            if (responseCode == HttpURLConnection.HTTP_OK) {
                BufferedReader reader =
                        new BufferedReader(new
InputStreamReader(connection.getInputStream()));
                String line;
                StringBuilder response = new StringBuilder();
                while ((line = reader.readLine()) != null) {
                    response.append(line);
                }
                reader.close();
                System.out.println("Response: " + response.toString());
            } else {
                System.out.println("Error: " + responseCode);
            }
            connection.disconnect();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

POST Method:

The POST method is one of the HTTP methods used for submitting data to be processed by the server. Unlike the GET method, which is used for retrieving data, the POST method is intended for data submission and can cause modifications on the server. Here are some important points to consider when working with the POST method:

1. Submitting Data:

2. Non- Idempotent:

3. Request Headers:

4. Response Codes:

5. Limitations:

- **Lack of Idempotence**:
- **Lack of Caching:**

In the provided Java code example, a POST request is sent to `https://api.example.com/resource` with a request body containing the data to be submitted. The data is written to the request output stream and the response code is checked to handle the response accordingly.
When using the POST method, it is crucial to ensure proper authentication, authorization, and input validation to prevent security vulnerabilities and protect the integrity of the server and data.
Remember to use the appropriate HTTP method based on the intended operation. While the GET method is used for retrieving data, the POST method is suitable for submitting data for processing or creating new resources on the server.

POST Method:

```
import java.net.HttpURLConnection;
import java.net.URL;
import java.io.DataOutputStream;

public class PostExample {
    public static void main(String[] args) {
        try {
      URL url = new URL("https://api.example.com/resource");
            HttpURLConnection connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("POST");
            connection.setDoOutput(true);
            String postData = "data=example";
            DataOutputStream outputStream = new DataOutputStream(connection.getOutputStream());
            outputStream.writeBytes(postData);
            outputStream.flush();
            outputStream.close();

            int responseCode = connection.getResponseCode();
            if (responseCode == HttpURLConnection.HTTP_OK) {
                // Process response
            } else {
                // Handle error
            }
            connection.disconnect();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

PUT Method:

The PUT method is an HTTP method used for updating or replacing a resource on the server. It is idempotent, meaning that multiple identical PUT requests should have the same outcome. Here are some important points to consider when working with the PUT method:

1. Updating Resources:

2. Idempotence

3. Resource Identification:

4. Partial Updates:

5. Response Codes:

6. Limitations:

- **Lack of Partial Updates**:
- **Security Considerations**:

In the provided Java code example, a PUT request is sent to `https://api.example.com/resource` with a request body containing the updated data. The data is written to the request output stream, and the response code is checked to handle the response accordingly.

When using the PUT method, it is important to handle concurrency and data consistency issues appropriately. For example, you may use optimistic locking mechanisms or versioning to ensure that updates do not conflict with other concurrent modifications to the resource.

Remember to use the appropriate HTTP method based on the intended operation. While the GET method is used for retrieving data and the POST method is used for submitting data, the PUT method is suitable for updating existing resources on the server.

PUT Method:

```
import java.net.HttpURLConnection;
import java.net.URL;
import java.io.DataOutputStream;

public class PutExample {
    public static void main(String[] args) {
        try {
            URL url = new URL("https://api.example.com/resource");
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            connection.setRequestMethod("PUT");
            connection.setDoOutput(true);

            String putData = "data=updated";
            DataOutputStream outputStream = new
DataOutputStream(connection.getOutputStream());
            outputStream.writeBytes(putData);
            outputStream.flush();
            outputStream.close();

            int responseCode = connection.getResponseCode();
            if (responseCode == HttpURLConnection.HTTP_OK) {
                // Process response
            } else {
                // Handle error
            }
            connection.disconnect();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

DELETE Method:

The DELETE method is an HTTP method used for deleting a specified resource on the server. It is used to remove a resource permanently from the server. Here are some important points to consider when working with the DELETE method:

1. Deleting Resources:

2. Idempotence:

3. Resource Identification:

4. Response Codes:

5. Limitations:

- **Lack of Safety**:
- **Cascading Deletions**:

In the provided Java code example, a DELETE request is sent to `https://api.example.com/resource/123`, where "123" represents the identifier of the resource to be deleted. The response code is checked to handle the success or failure of the deletion operation.

When using the DELETE method, it is important to implement proper authorization and authentication mechanisms to prevent unauthorized deletions. Additionally, consider providing proper error handling and feedback to the client in case of failures or errors during the deletion process.

Remember to use the appropriate HTTP method based on the intended operation. While the GET method is used for retrieving data, the POST method is used for submitting data, the PUT method is used for updating data, the DELETE method is specifically designed for resource deletion.

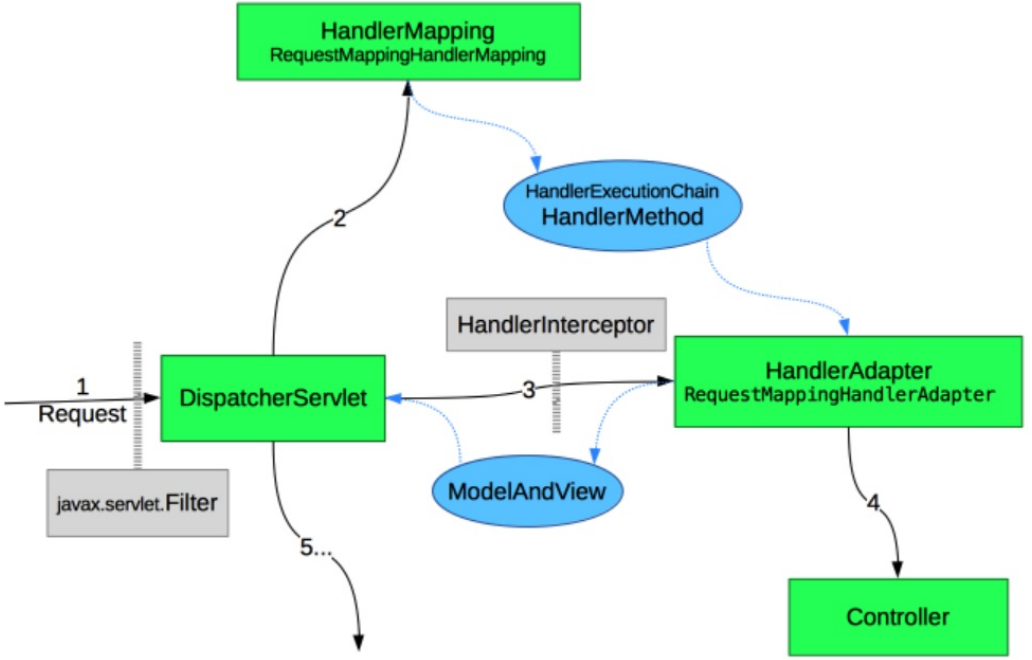
DELETE Method:

```
import java.net.HttpURLConnection;
import java.net.URL;

public class DeleteExample {
    public static void main(String[] args) {
        try {
            URL url = new URL("https://api.example.com/resource/123");
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            connection.setRequestMethod("DELETE");

            int responseCode = connection.getResponseCode();
            if (responseCode == HttpURLConnection.HTTP_NO_CONTENT) {
                // Resource deleted successfully
            } else if (responseCode == HttpURLConnection.HTTP_NOT_FOUND)
{
                // Resource not found
            } else {
                // Handle other errors
            }
            connection.disconnect();

           } catch (Exception e) {
               e.printStackTrace();
           }
       }
   }
```

RequestMapping

One of the most important annotations in spring is the **@RequestMapping Annotation** which is used to map HTTP requests to handler methods of MVC and REST controllers. In Spring MVC applications, the DispatcherServlet (Front Controller) is responsible for routing incoming HTTP requests to handler methods of controllers. When configuring Spring MVC, you need to specify the mappings between the requests and handler methods.

To configure the mapping of web requests, we use the **@RequestMapping** annotation. The @RequestMapping annotation can be applied to class-level and/or method-level in a controller. The class-level annotation maps a specific request path or pattern onto a controller. You can then apply additional method-level annotations to make mappings more specific to handler methods. So let's understand @RequestMapping Annotation at Method-level and Class level by examples.



Requirements:

- *Eclipse (EE version)/STS IDE*
- *Spring JAR Files*
- *Tomcat Apache latest version*

@RequestMapping Annotation at Method-Level

Spring Boot is the most popular framework of Java for building enterprise-level web applications and back-ends. Spring Boot has a handful of features that support quicker and more efficient web app development. Some of them are Auto-configuration, Embedded Server, opinionated defaults, and Annotation Support. In this article, we'll be exploring the core annotation of **Spring Boot – @RequestMapping** which is part of the set of annotations that Spring Boot employs for defining URL endpoints and REST APIs.

@RequestMapping

This annotation is a versatile and flexible annotation that can be used with a controller (class) as well as the methods to map specific web requests with the handler methods and controllers. This annotation is part of a larger set of annotations provided by Spring Framework to define URL endpoints and simplify the development of Spring Boot applications.

It has the following features:

- Define several different endpoints to access a specific resource.
- Build REST APIs to serve web requests.
- Simplify the web development process by simply defining an annotation that offers a set of functionalities for handling requests.
- Define multiple endpoints in a single @RequestMapping annotation.

What is Request Body?

Data sent over the request body can be of any format like json, XML, PDF, Http Forms, and many more. The **Content-Type** header indicates the server's understanding type of request.

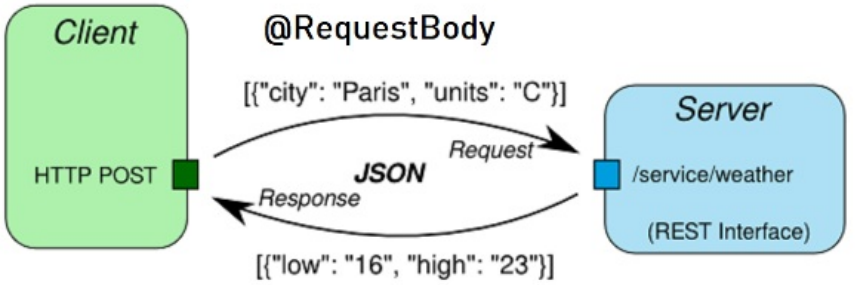
- Spring provides **@RequestBody** annotation to deserialize the incoming payload into java object or map.
- The request body goes along with **content-type** header for handler method to understand and deserialize the payload.

How to Get the Body of Request in Spring Boot?

Java language is one of the most popular languages among all programming languages. There are several advantages of using the Java programming language, whether for security purposes or building large distribution projects. One of the advantages of using Java is that Java tries to connect every concept in the language to the real world with the help of the concepts of classes, inheritance, polymorphism, etc.
There are several other concepts present in Java that increase the user-friendly interaction between the Java code and the programmer such as generic, Access specifiers, Annotations, etc. These features add an extra property to the class as well as the method of the Java program. In this article, we will discuss how to get the body of the incoming request in the spring boot.

@RequestBody: Annotation is used to get the request body in the incoming request.

Spring Initializr is a web-based tool using which we can easily generate the structure of the Spring Boot project. It also provides various features for the projects expressed in a metadata model. This model allows us to configure the list of dependencies that are supported by JVM. Here, we will create the structure of an application using a spring initializer and then use an IDE to create a sample GET route. Therefore, to do this, the following steps are followed sequentially as follows:



@RequestBody Annotation in Spring and Spring Boot

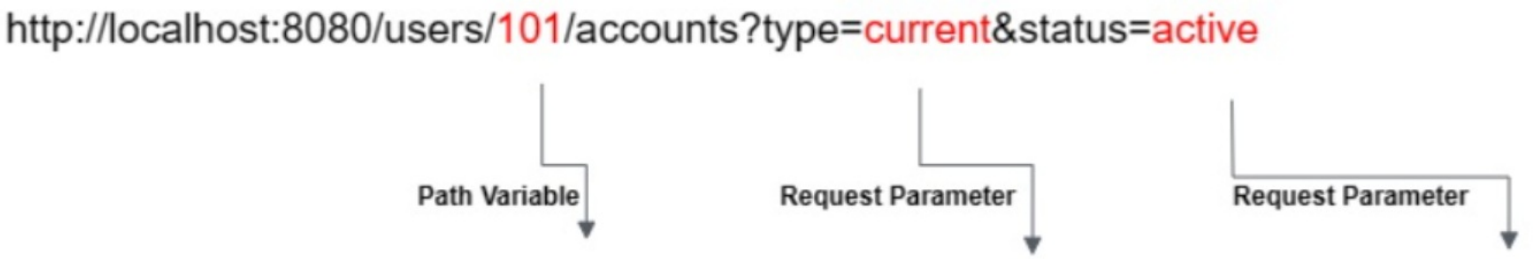
The `@RequestBody` annotation is responsible for binding the HTTPRequest body to the body of the web request. Depending on the content type of the request, the body of the request is given through a `HttpMessageConverter`, which resolves the method argument.

We can also use the `@Valid` annotation to automatically validate the input.
In brief, the `@RequestBody` annotation is responsible for retrieving the request body and automatically converting it to the Java object.

Path variable in spring boot

The @PathVariable annotation is used to retrieve data from the URL path. By defining placeholders in the request mapping URL, you can bind those placeholders to method parameters annotated with @PathVariable. This allows you to access dynamic values from the URL and use them in your code.

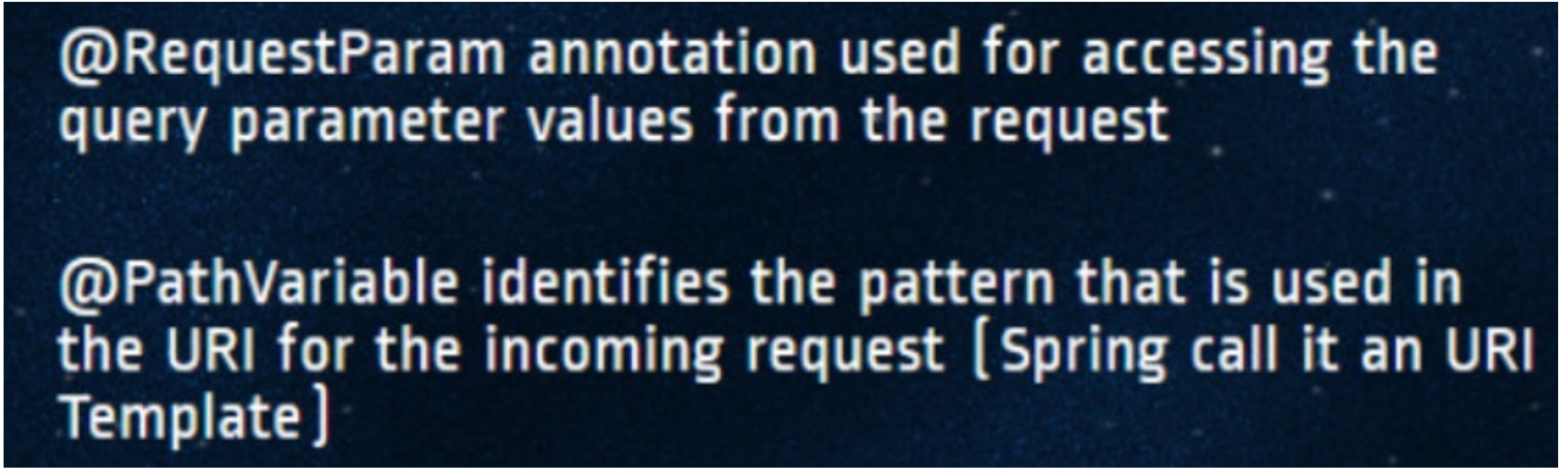
The @PathVariable annotation is used to retrieve data from the URL path. By defining placeholders in the request mapping URL, you can bind those placeholders to method parameters annotated with @PathVariable. This allows you to access dynamic values from the URL and use them in your code.




```
@GetMapping("/user/{id}/accounts")
public List<Account> getAccounts(@PathVariable String id, @RequestParam String type, @RequestParam String status) {

    //...
}
```

Using @PathVariable

The @PathVariable annotation is used to extract data from the URL path. It allows you to define placeholders in your request mapping URL and bind those placeholders to method parameters.



Feature	@RequestParam	@PathVariable
Usage	Query parameters	Path variables
Mapping	Maps to request parameters with matching names	Maps to URI template variables
Position	Can be placed anywhere in the method parameter list	Must be placed directly on the method parameter
Required	Can be optional or required (default is required)	Always required
Binding	Optional binding to a default value if not present	Binds directly to the URI template variable
URL Encoding	Required for special characters in parameter values	Automatically decoded
Example	`@RequestParam("paramName") String paramValue`	`@PathVariable("varName") String varValue`

Build Web Applications

Java is one of the most used programming languages for developing dynamic web applications. A web application is computer software that utilizes the web browser and technologies to perform tasks over the internet. A web application is deployed on a web server.

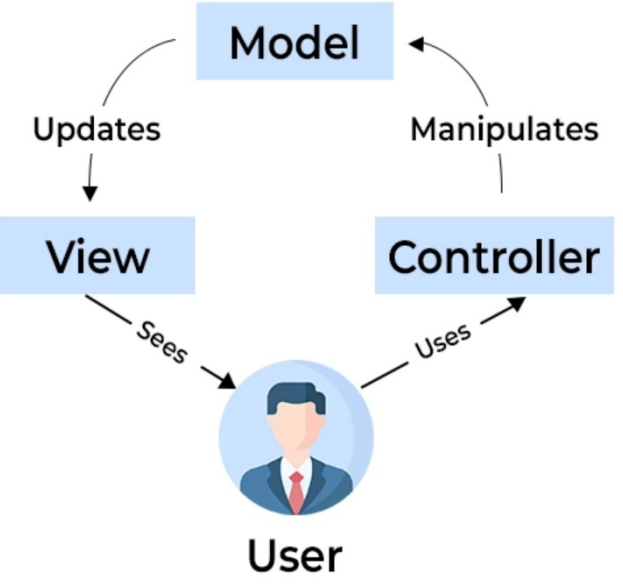
Java provides some technologies like Servlet and JSP that allow us to develop and deploy a web application on a server easily. It also provides some frameworks such as Spring, Spring Boot that simplify the work and provide an efficient way to develop a web application. They reduce the effort of the developer.

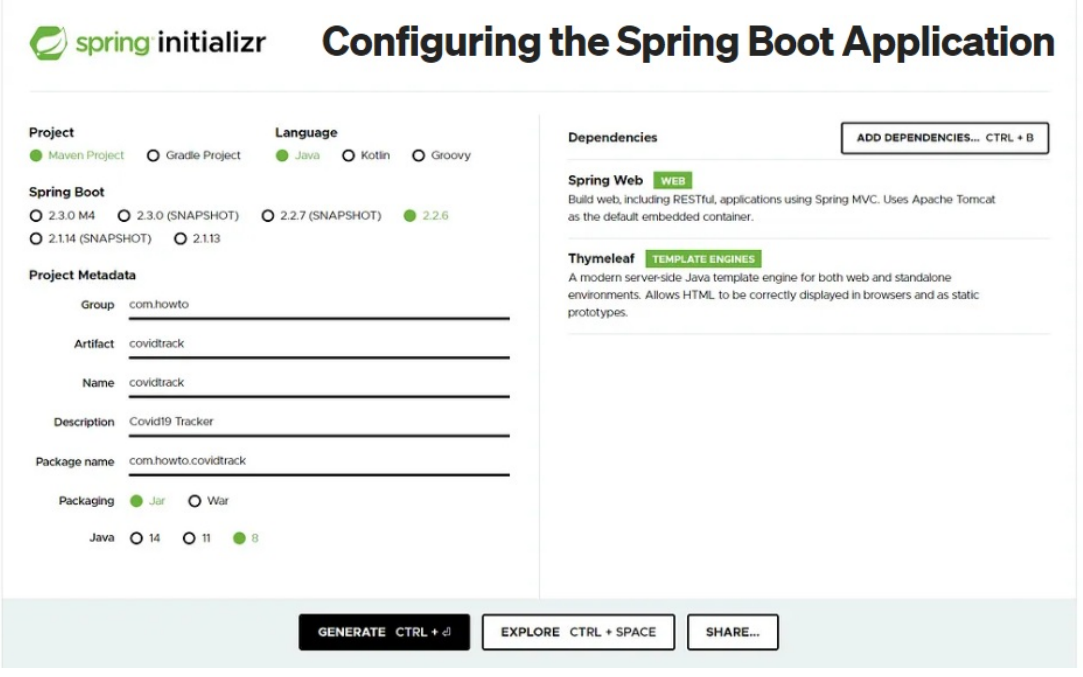
We can create a website using static HTML pages and style them using CSS, but we need server-side technology when we want to create a dynamic website

A web application is computer software that can be accessed using any web browser. Usually, the frontend of a web application is created using the scripting languages such as HTML, CSS, and JavaScript, supported by almost all web browsers. In contrast, the backend is created by any of the programming languages such as Java, Python, Php, etc., and databases. Unlike the mobile application, there is no specific tool for developing web applications; we can use any of the supported IDE for developing the web application.

Building web applications with Spring Boot

We use the concept of MVC, i.e., Model-View-Controller.
MVC is an architecture that separates various components of an application like the Input Logic, Business Logic, and UI Logic.
The views and the models don't interact with each other.
The controller receives the request from the view and





AT THE END YOUR TASK :-

Build a Web Application using Java , Spring etc.

For E.g:- To do List ,
Calculator etc.

THANK YOU!!

I-TECH WORLD (AKTU)

JOIN TELEGRAM GROUP FOR HAND WRITTEN NOTES , QUANTUMS AND OTHER STUDY MATERIAL.

LINK GIVEN IN DESCRIPTION BOX