Cmd 🡪 location of pom.xml file

Cmd 🡪 mcn clean install.

* Invokes the parent constructor through the child constructor.
* super( ) should be the first line of code in the child constructor .
* super( ) is optional , even you don’t define . It is generated by default.
* If we create a parenthesized constructor, it is mandatory to create the non-parenthesized constructor.

//CHILD

**public** **class** Child **extends** Parent {

**public** Child() {

**super**(5); //Invoking the parent constructor. super() should be the first line of code

System.***out***.println("In Child");

}

}

//PARENT

**public** **class** Parent {

**int** variable;

**public** Parent() {

System.***out***.println("In Parent");

}

**public** Parent(**int** variable) {

**super**();

**this**.variable = variable;

System.***out***.println(variable);

}

}

* Major advantage of inheritance is: we can access the grand parent’s methods, parent methods, child’s methods through the child instance.
* Multi-Level Inheritance :

A 🡪 B (A extends B)

B 🡪 C (B extends C)

C 🡪 Object (C extends object)

* No multiple inheritance :

A 🡪 B and A🡪 C not possible.

* **INTERFACE:** It’s a collection of variables but each variable is **public static final variable**.
  + Interface is a collection of one or more abstract method / pure abstract method.
  + Cannot have constructor.
  + Interface cannot be instantiated
  + If two interfaces have same abstract class signature , only one needs to be overridden
* **Why interface?**
  + Interface is introduced in the java to achieve multiple inheritance as the java does not support it
  + When we want common behaviour.
  + For Abstraction – Hiding the implementation details of the class.
* **How to use?**
  + ClassName extends ClassName2 { }
  + ClassName implements Interface1 { }
  + ClassName implements Interface1 , Interface2 { }
  + ClassName extends ClassName2 implements Interface1 ,Interface2 { }
  + After creating an instance , create the class name which implements that interface with the suffix “Impl” //best coding practice
    - if Interface is “Calculator” , class name must be “CalculatorImpl”
* **Syntax of defining an Interface?**
  + Interface should start with capital letter

**package** com.model;

**public** **interface** Calculator {

}

* + Interface <InterfaceName> {

Public static final int VARIABLE = 456; // by default all the variables are public static final constants

Public static final String VARIABLE = “Hello”;

Public static final Boolean VARIABLE = true;

Public abstract Void Behaviour(); // by default the methods are public abstract

Public abstract Void Behaviour1();

Public abstract Void Behaviour2 (); // method declaration, no definition, no body

}

* **STATIC?**
  + Keyword : Reserved word
  + Only one copy is created
  + Memory is pre created.
  + Static variables and methods are pre created by the JVM even before the object variables gets instantiated.
  + Static can be at the variable level and
    - Ex: static int var;
  + static can be at the method level or
    - Ex: static int add(){return 10;)
  + it can be at the block level
    - Ex :static { }

**public** **class** StaticDemoMain {

**public** **static** **void** main(String[] args) {

StaticDemo staticDemo = **new** StaticDemo();//create object

staticDemo.intVariable = 5;

staticDemo.display();

StaticDemo staticDemo2 = **new** StaticDemo();

staticDemo2.intVariable = 13456;

System.***out***.println(StaticDemo.*STATICVAR*); // class name variable

StaticDemo.*display2*(); // class name variable

}

**static** {

System.***out***.println("static block");

}

}

**Output:** static block //executed even before main

non static function

66

static function

* Where to use static block?
  + When there is a common data for all instances ( new)
  + Where developer does not want to create an object.
  + static can be at the class level variable
    - EX : <ClassName>.staticVariable;

Instance variable – first instantiate then call the variable

Employee emp = new Employee ();

Emp.intVar = 20;

* **@override:** this is a meta information to the compiler. It is an annotation. It is an interface.
* **INTERFACE?**
  + Interface is a key word. It cannot be instantiated.
* **POLYMORPHISM?** 
  + Static – method overload
  + Runtime – method override (extends)
* **Method overload:** if a class has two or more functions have same name with different arguments/parameters and return type does not matter in overload , then its k/as Method overloading.
* **Method Override:** Override is possible only in inheritance/Interface i.e., (extends/ implements)
  + The parent class has the method signature and the subclass also has the same method signature of the parent.
  + The return type does matters in method override.
  + Parent can have lower or equal access modifier.
  + But child should have equal or greater access modifier.
* **Protected:** It is an access modifier.
  + Represented by # in the class diagram
  + Protected would act like private to other classes but public to inherited class.
* **MARKER INTERFACE:** An interface which does not have any abstract method. It’s an empty interface (no variables, no methods)
* **Extending Interface :**  An interface can extend another interface.
  + Public interface A {

//abstract method of A

}

Public interface C {

//abstract methods of C

}

Public interface B extends A, C {

//includes Abstract methods of A and C,

//also methods of B

}

* ***STRING*** : String is a Class. (predefined class)
  + ***Default package : java.lang.\* ;***
  + ***Specific : java.lang.String;***
  + ***Import java.lang.\* ;*** //prefer specific locationrather than default
  + java.lang.String; //preferable
* Two types :
  + String object
    - String str = new String (“Hello”);
    - Str = null;
  + String literal :
    - Ex : String str = “Hello”; //created in the string pool
* Strings are immutable (fixed == Constant)
* Cmd -> javap java.lang.String
* Update/Alter/Modify the string data
* **StringBuffer:**

**24 March 2022**

**STATIC DATA AND DYNAMIC DATA:**

* Static data = data is hard coded. Int a = 10;// not advised
* Dynamic data = data is taken from the user.

Consuming data from the Console (Command line argument):

Cmd > Javac FileName.java // javac – java compiler, FileName.java = source File/src

This command generates Byte Code i.e., Filename. Class //not human or machine readable.

Cmd> java FileName // JVM will read the .class file and convert it into low level language/machine level language

Cmd < java FileName arg1 arg2 //arguments passed on run time

Class FileName {

Public static void main (String [ ] args){ //read data from console as array of strings [] from args

SOP{args[0]); // arg1

SOP{args[1]); //arg2

}

In Eclipse : use run configuration and set the arguments to be passed on to the main method args

**WRAPPER CLASS :**

* Byte - byte
* Character -char
* Short -short
* Long -long
* Integer - int
* Float -float
* Boolean -boolean
* Double -double
* A Wrapper class is a class whose object wraps or contains primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

**Need of Wrapper Classes**

* They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
* The classes in java.util package handles only objects and hence wrapper classes help in this case also.
* Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.
* An object is needed to support synchronization in multithreading.

**SCANNER:**

* Scanner (ctrl+space)
* Scanner scan = new Scanner(System.in);

Alt+Shift +S +R 🡪 generates setters and getters in eclipse

**EXCEPTION:**

* Under “Object class “, there is a class called “Throwable “ (subclass)
* Under throwable – 2 subclasses – Error, and Exception
* Object 🡪 Throwable 🡪 Error, Exception
* Errors - cannot be handled.
* Anything outside the JVM we cannot handle it.
* Out of memory
* Exceptions – can be handled by the developers in programs.
* Checked Exception and unchecked Exception
* Checked Exception – we have to check during development **(java.util ) (java.io)**
* If we misses out then the JVM will handle it as an unchecked exception. **Java.lang ,**

**Java.lang.\* Exception .**will have all the possible exceptions which might get missed. We need to handle it.

**try {**

**22/0;**

**}**

**catch(java.lang.ArithmeticException a ){**

**//handle error , provide solution**

**SOP(a);**

**}**

**catch(java.lang.ArrayIndexOutOfBoundException a ){**

**//handle error , provide solution**

**SOP(a);**

**} catch(classCastException a ){**

**//handle error , provide solution**

**SOP(a);**

**} catch(NullPointerException a ){**

**//handle error , provide solution**

**SOP(a);**

**}**

**catch(Exception a ){ //Generic Exception**

**//handle error , provide solution**

**SOP(a);**

**}**

* Try must be mandatorily followed by catch.
* **try {**

**22/0;**

**}**

**catch(Exception a ){ //Generic Exception**

**//handle error , provide solution**

**SOP(a);**

**}**

* **FINALLY BLOCK :**
* **try {**

**22/0;**

**}**

**catch(Exception a ){ //Generic Exception**

**//handle error, provide solution**

**SOP(a);**

**}**

**Finally(**

**//even if there is an exception or not, this block will be executed.**

**//works to deal with the abrupt ending of the program due to exception.**

**//Why finally: ”*to Close the resources. i.e., can use dereference and close.”***

**}**

* **try{**

**//mandatory**

**}**

**Catch() or finally{} – one is mandatory**

**Try{}**

**Catch{}**

**Try{}**

**Finally{}**

**try**

{

Sysout(arg

s[999]); //possible of raising an exception

}

**catch**(java.lang.ArrayIndexOutOfBoundException var)

{

Sysout(“solution”);

}

**USER DEFINED EXCEPTION/CUSTOM EXCEPTION: THROW:**

Throw new Exception ;

We can throw Custom Exception;

Class MyClass extends Exception {

}

* create an new package for all the exceptions , com.exception
  + create a new class and inherit from java.util.Exception
  + Add a String variable and the constructor method
  + We can use the override method in the exception block
* Catch (PositiveNumberException | ArithmeticException e){

//we can use pipeline to add multiple exceptions in the catch block

}

**COLLECTION FRAMEWORK:**

**Collection – collection of objects**

Cmd > javap java.util.collection

Collection is an interface and it inturn extends interable. //12 methods

Java.util.interable(); //3 methods

Javap java.util.set

Set extends collection

Javap java.util.list : list extends collection

Javap java.util.map : does not extend anything

Javap java.util.HashSet

//hashing technique , it’s a class and it extends AbstractSet and it implements Set

So the other 15 methods are overridden by HashSet ., this is the concrete class

//interface var = new Impl();

Set set = **new** HashSet();

**import** java.util.HashSet; //implementation of the set

**import** java.util.Set; //interface

March 28th 2022

JAVA

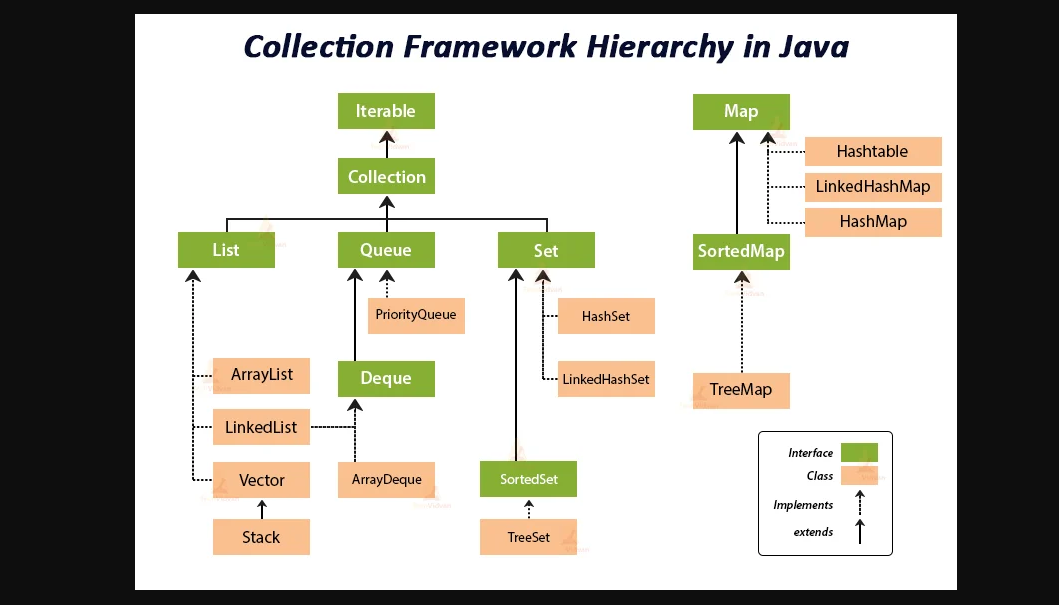
API 🡪 predefined 🡪 .class , .package , interface 🡪 jar files

All predefined .jar files 🡪 rt.jar file : location , c-> program files -> java -> java version-> lib

Predefined Packages 🡪 java.lang.\* ; (all .class ) – default package .

Java.util.\* ; 🡪 utility 🡪 collection 🡪 it is a collection of APIs ( classes, packages , interfaces , Algorithms ), because java does not support pointers . 🡪 Collection also has something called Data Structures , (how data is stored and retrieved internally)

Java Collection :



Interface can extend another interface, all the methods in the parent interface has to be inherited by the child interface

SET <interface>:

* Set is a collection of objects and no primitive data type.
* Set is **Dynamic** in size (add /remove ‘n’ number of objects dynamically).
* Set has **NO** **Duplicates** (not the data, but object from the same memory address).
* While iterating / looping, no order is maintained.
* Set is an **interface** which extends another interface called Collection. Collection extends iterator. Instead of overriding all these methods, we use **Concrete** Class/ **Complete** Class/ **Implementation** Class. They are HashSet, LinkedHashSet and TreeSet.
* **HashSet** – it follows unsorted Order
* **TreeSet** – it follows sorted order
* **LinkedHashSet** – Double Linked
* **Maven Local Repository** (local repo) will be stored in the c-> users->nithinkumarp-> .m2-> repository.

**In for loop : While using a iterator for loop , if the set has all the same type of Objects then we can specify the type of Object in the place of object .If the collection contains different type of objects its preferable to use the generic Object as a type in the statement.**

**for** (Iterator iterator = set.iterator(); iterator.hasNext();) {

Object object = (Object) iterator.next();

}

**Type Casting in java:**

Int a = ( int ) 10.45f;

Object a = (employee) Emp1;

**Instance of in JAVA:**

**If( object instanceof Employee ){ // object instanceof <type**

**Employee new\_name = (Employee ) object; // <type> new\_name = (<type>) object;**

**}**

***“ALWAYS FOLLOW “***

**<Interface> var = new <ClassImpl>();**

***GENERIC COLLECTION:***

Set<Object\_name> set = new HashSet<Object\_name> ( );

Set <Employee> set = new HashSet<set> ( ); // This set can only keep Employee Object

* **Final** : constant , avoids overloading, over ridding
* **Finally** : a block which can come along try block
* **Finalize** : Garbage the collection . Helps in protecting the files before the GC acts on a particular data.
* Every java file can have multiple Classes with default access modifiers. IF there is a class with public access modifier then it should be the file name. There should be only 1 class as public and it should be same as that of the File name

***COMPARABLE & COMPARATOR:***

* Java provides 2 **interfaces** to **sort objects** 🡪 Comparable and Comparator
* Comparable comes under java.lang. Package.
* Javap java.lang.Comparable
* Comparator comes under java.util. Package.
* Javap java.util.Comparator 🡪have to override the public abstract int compareTo( T ) method available in the interface. (T represents any object type)
  + Int return type – it returns 3 values , 1 , 0 , -1
  + If (this.age == arg.age) its equal it returns 0
  + If (this.age > arg.age) its > returns 1
  + If (this.age < arg.age) its < returns -1

Set: no insertion order-no duplicates

Sorted Set – order/sequence -- HashSet

Unsorted Set – no order/ no sequence -- TreeSet

Tree – an algorithm, a binary tree

List: insertion order is maintained – allows duplicate –

* ***Javap java.util.Collections*** : it’s a class has various methods which can be used.
* Comparable provides single sorting sequence , comparator can have multiple sequence
* ***Comparator :*** class Employee implements java.util.Comparator

Client server architecture

Database 🡪 MySql 🡪 open 🡪 Java🡪 Oracle

Persistence 🡪 Storage 🡪 3rd party

Using an App we will store data into the database/data source.

Even if the data is not used, data is available in the DB.

//we can use java to write and then .net to read :

Java 🡪 store 🡪 DB 🡪 Read 🡪 .Net

**LOGGING:**

SOP 🡪 replace 🡪 Logger

Various Logger

🡪 log4j is one of the vendor , his product is part of Apache. But vulnerable

**Logging in JAVA**

🡪log4j has certain vulnerabilities, so we can use, **java.util.log**

API 🡪 jar files ( \*.jar )

How to get log4j files ??

Build tool 🡪 Maven /Gradel

Step 1: Create a Maven project

Step 2 : Maven repository :

Add the log4j maven repository available on the net to the pom file.

<project xmlns=*"http://maven.apache.org/POM/4.0.0"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>com</groupId>

<artifactId>log4jdemo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

<log4j.version>1.2.16</log4j.version>

</properties>

<dependencies>

<!-- https://mvnrepository.com/artifact/log4j/log4j -->

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>${log4j.version}</version>

</dependency>

</dependencies>

</project>

Step 2 : Create a property log4j.properties

It uses LHS = RHS concept

App uses LHS and while running it uses RHS data

Add this file to the 🡪 **src/main/resources**

The resources required by the app are added in this resources

Exception : while creating give the super class as Exception, create a variable of message and a constructor , to pass the message use the override method

SQL : Structured Query Language.

SQL commands are mainly categorized into four categories.

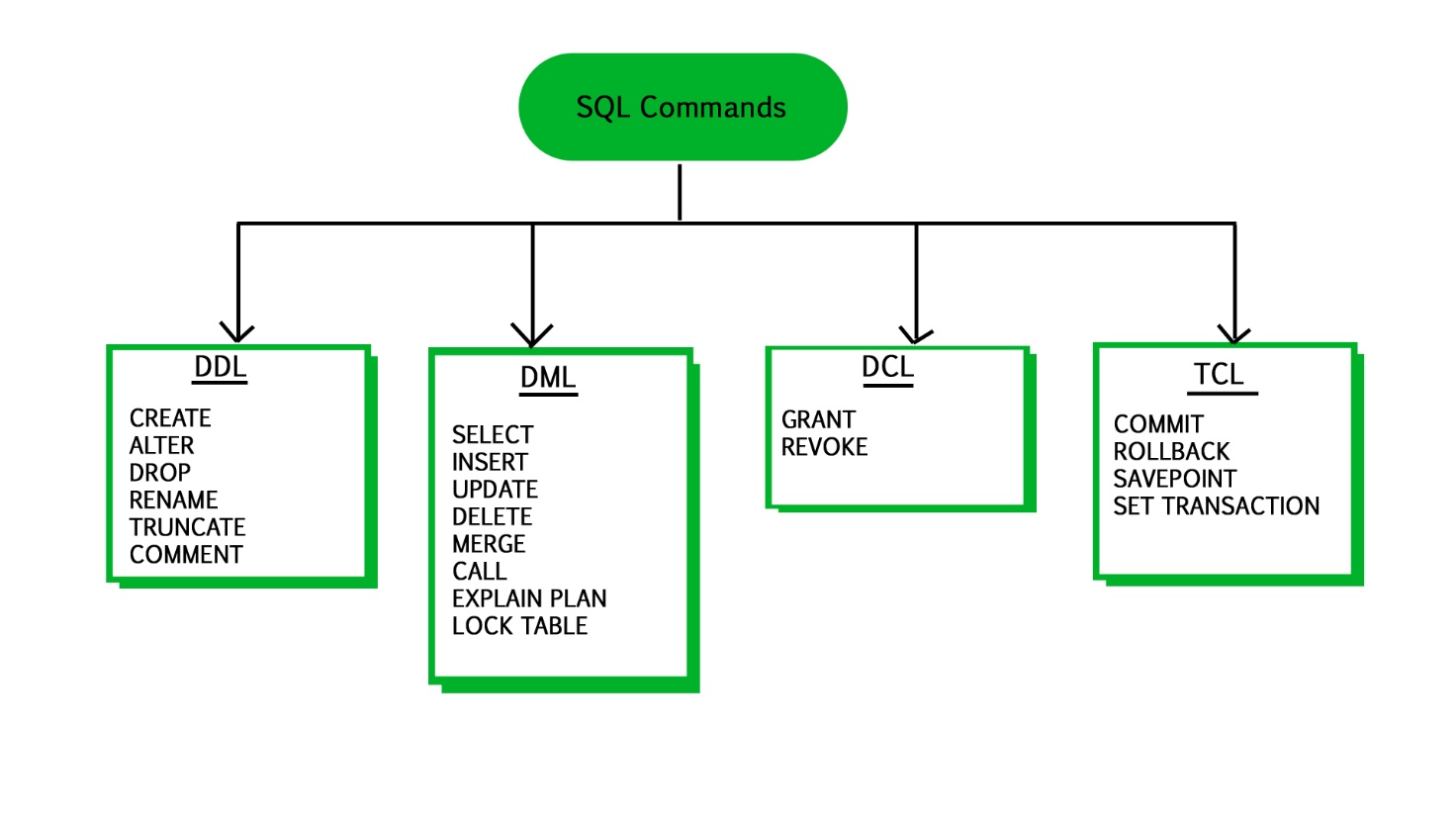
Data Definition language: CREATE, DROP, ALTER, TRUNCATE, COMMENT, RENAME.

Data Query language: SELECT

Data manipulation language: INSERT, UPDATE, DELETE, LOCK

Data control language: GRANT (PRIVILEGES TO THE DATABASE), REVOKE(WITHDRAWS THE USERS ACCESS)

Transaction control language: COMMIT,ROLLBACK,SAVEPOINT



Data od MySQL (open source / freeware) 🡪 oracle

Latest : 8.X

Ways to install :

CLI : command line interface

GUI : Graphic user interface

Default user name: root

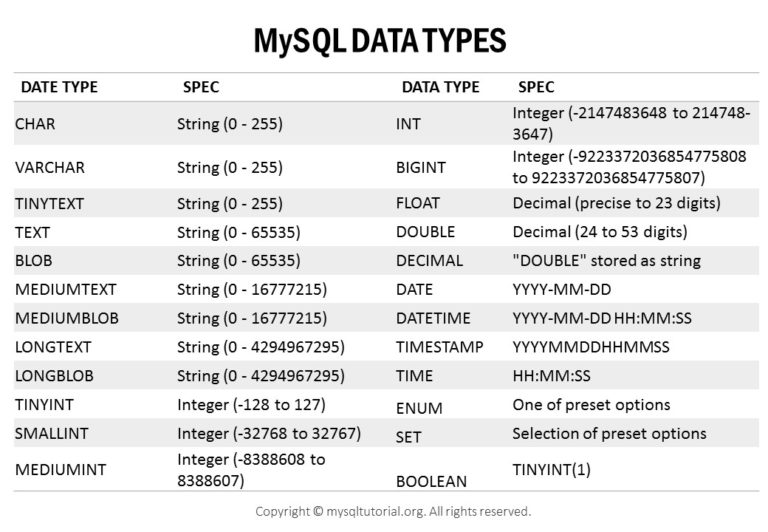
Default password: Reset123/root

**;** 🡺 MANDATORY TO TERMINATE THE STATEMENT IN MYSQL

ALL THE KEY WORDS /RESERVED WORDS ARE IN CAPITAL LETTER – CODING STANDARDS; BUT OTHERWISE NOT CASE SENSITIVE

BLOB – Binary Large Object

There are about 24 different data types in MYSQL



mysql > select version(); //version of mysql

mysql > select current\_date; //current date

mysql > select now(); //date and time

SCHEMA (databases)

mysql > show databases; //gives the databases(i.e., SCHEMA ) available

mysql> select database // current working database

mysql > CREATE DATABASE <demodb1>; //creates a database

mysql > use <demodb1>; //moves the working directory to the database we created

mysql >show tables ; // shows the available tables

mysql >CREATE TABLE IF NOT EXISTS <usertable> (

<id> int(5) NOT NULL auto\_increment ,

<username> varchar(25) NOT NULL DEFAULT ‘’ ,

<password> varchar(25) NOT NULL DEFAULT ‘’ ,

PRIMARY KEY ( id )

);

**CREATE TABLE IF NOT EXISTS library\_table(**

**accession\_Number INT ,**

**book\_Title varchar(25) NOT NULL ,**

**author\_Name varchar(25) NOT NULL,**

**book\_Price FLOAT NOT NULL,**

**issued\_Status BOOLEAN NOT NULL ,**

**PRIMARY KEY (accession\_Number)**

**);**

**INSERT INTO library\_table (accession\_Number, book\_Title, author\_Name, book\_Price, issued\_Status)**

**VALUES(100,’java’ , ‘nithin’ ,500.0,0);**

mysql >DELETE FROM <usertable> where id = 123; //delete a record

mysql >COMMIT //to save

mysql >ALTER TABLE <usertable> RENAME COLUMN <id> TO <userId> ; //rename the column in a table

mysql >ALTER TABLE <usertable> ADD <userlocation> varchar(40) AFTER <password>;

mysql > ALTER TABLE <usertable> DROP COLUMN <userlocation>; //delete column

mysql> ALTER TABLE table\_name RENAME TO new\_table\_name; //rename table

1. mysql > DELETE FROM table\_name WHERE condition; //delete a record

mysql > SELECT

officeCode,

city,

phone,

country

FROM

offices

WHERE

country

IN ('USA' , 'France'); // search for a column in the given set of values

mysql > SELECT

DISTINCT lastname

FROM

employees

ORDER BY

lastname;

Mysql > **SELECT**

customerName,

country,

salesrepemployeenumber

**FROM**

customers

**WHERE**

Salesrepemployeenumber

**IS NULL**

**ORDER BY**

customerName; //to check if the value is null or not(**IS NOT NULL**)

SELECT

select\_list

FROM t1

INNER JOIN t2 ON join\_condition1

INNER JOIN t3 ON join\_condition2

...;

***JDBC:***

Database 🡪 RDBMS 🡪 Multiple table (3rd part)

JAVA application uses API (\*.java) 🡪 Database 🡪 RDBMS 🡪 Multiple table (3rd part)

Here java uses JDBC API to connect to database.

JDBC 🡪 JAVA DATABASE CONNECT

4 types:

1. Type 1 Driver 🡪 java used to connect to Microsoft through ODBC (Open DB connectivity). ODBC is connected to the data source /database. (mySql)
2. Type 2
3. Type 3
4. Type 4 🡪 THIN Driver 🡪 java can directly connect to Database and it requires only (<database>.jar) files. [ex: if oracle, oracle.jar]
   1. Java 🡪 java.sql.\*

🡪 java.sql.\* is an Interface

🡪DriverManager(class)

🡪Mysql has to implement the rules of java.sql to use this JDBC

JDB DEMO:

Step 1. Create a Maven Project (pom.xml)

1. JDK 1.8
2. Mysqlconnector.jar

Step 2. Create packages [com.main , com.service , com.model , com.dao (also k/as Data Access Object /com.repository) , com.exception ].

* 1. Only com.dao has the java.sql statements
  2. Com.dao has to return the POJO files

Step 3. In the Com.dao( Appl🡪 database)

1. To establish the connection, we need the location of the database server .

(URL: location)

It requires the credentials to access the DB

DriverManger(sql) Connection <interface>

Serialization 🡪 to maintain the state of the object , POJO files need to implement Serializable

select \* from library\_table WHERE accession\_number = 101;

# JAVA 8 FEATURES

* JSE java Standard Edition – core java (JDK 17), till Java 8 is open source, most of the companies prefer.
* JEE Java Enterprise Edition – (Advances Java)
* Change in INTERFACE ,

Java 8 provides following features for Java Programming:

* Lambda expressions,
* Method references,
* **Functional interfaces**,
* Stream API,
* **Default methods**,
* Base64 Encode Decode,
* **Static methods in interface**,
* Optional class,
* Collectors class,
* ForEach() method,
* Nashorn JavaScript Engine,
* Parallel Array Sorting,
* Type and Repating Annotations,
* IO Enhancements,
* Concurrency Enhancements,
* JDBC Enhancements etc.
* **Interface** :
  + An interface has static constants final variables and abstract methods An interface is completely “abstract class” (collection of pure abstract method)
  + Interfaces form a contract between the class and the outside world
  + Multiple inheritance
  + To have common behaviour/functions/methods/Business Logic
  + Now in java 8 , interfaces can have concrete methods, static variables , static methods
* **Default Methods :**
  + Default methods are also known as defender methods or virtual extension methods.
  + Backward Compatible (why default).
  + Java provides a facility to create default methods inside the interface. Methods which are defined inside the interface and tagged with default keyword are known as default methods. These methods are non-abstract methods and can have method body.
  + Complete method () == concrete method () == method definition {//body}
  + In case if we have **same default methods** in **different interfaces** and it is implemented by a class then it is **mandatory to override**
  + Default methods enable you to add new functionality to the interfaces of your libraries and ensure binary compatibility with code written for older versions of those interfaces.
  + As name implies, default methods in java 8 are simply default. If you do not override them, they are the methods which will be invoked by caller classes. They are defined in interfaces.
  + Public interface Interface\_name{
    - * Public default int sum()\_{
      * }
      * };
* **STATIC METHOD in an Interface:** 
  + Keyword : Static
  + Only one
  + Pre created/ pre memory allocation
  + To access - ClassName.method( )
  + Also k/as Class level Method / variable
  + Interface can also have static methods
  + These static methods contain the complete definition of the function and since the definition is complete and the method is static, therefore these methods cannot be overridden or changed in the implementation class.
* **Functional Interface : (SINGLE ABSTRACT METHOD – SAM)**
  + A functional interface is an interface that contains only one abstract method.
  + They can have only one functionality to exhibit.
  + From Java 8 onwards, lambda expressions can be used to represent the instance of a functional interface.
  + A functional interface can have any number of **default** methods.
  + A functional interface can have any number of **STATIC** methods.
  + Runnable, ActionListener, Comparable are some of the examples of functional interfaces.
  + Functional interfaces can be annotated. (annotation .i.e., represented by ‘@’ – to give additional information about any particular part of the code)
  + Use : @FunctionalInterface //optional but , it is better to mention the annotations

Because it helps in stopping from having more than 1 abstract method.

* + SAM two methods :
    - User Defined
    - Predefined (provided by JDK 1.8)
* What’s the difference between **abstract class** and java 8 **interface**?
  + Myclass extends OneClass (no multiple inheritance)
  + MyClass implements InterfaceOne , InterfaceTwo (multiple Interface)
  + Abstract Class:
    - Abstract class can have variables + concrete /complete () and define the class abstract. Because the class ins abstract we cannot instantiate( new())
    - In a class if we have one or more abstract method then the class is abstract
    - Abstract class can have constructor but cannot be used.
  + Interface :
    - Public static final VARIABLE = value;
    - Public abstract method(); // method declaration
    - In JAVA 8 ,
      * Default 🡺 complete/concrete/method definition () // with logic
      * Backward Compatibility.
      * Java8 INTERFACE is very equivalent to CLASS
      * Public static Final - Variable - variable
      * Abstract method(); - definition{ }
      * Default method(); { }
    - **INSTANTIATION** is not possible in JAVA **INTERFACE**.
    - **No constructor** can be defined.
* **package** com.service;
* **public** **interface** RulesInterface {
* **public** **static** **final** **int** ***var*** = 20; // public static final = const
* **public** **abstract** **void** display();
* **public** **default** String displayVar() {
* **return** "Welcome to Default";
* };

}

C 🡪 Structural;

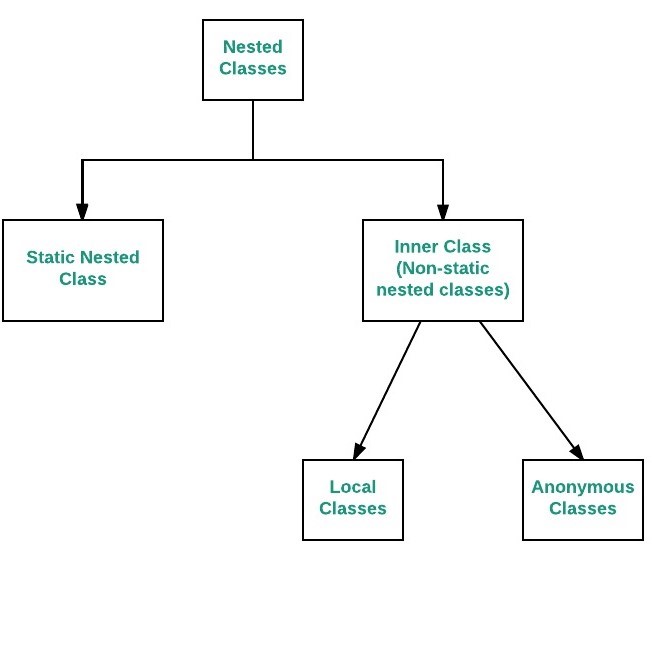
C++, JAVA 🡪 OBJECT ORIENTED;

JAVA 8 🡪 Functional programming

* **Functional Programming:** 
  + Function within a function
  + Int add(int a , int b){ }
  + Int sub (int add (int a, int b ) { } )
  + To implement a functional programming java 8 introduced (lambda express, method reference).
  + LAMBDA – simply a functional pointer (a pointer pointing to the function)
* **LAMBDA Expression :** 
  + - **( ) - > notation**
    - **LHS = RHS**

**Int ans = Obj.add(2,3);**

**Functional Interface = 🡪**

* + Change in interface in java 8, prior to JDK <1.8>.
  + We can use predefined Functional interfaces in java instead of user defined
  + Arrow notation to represent the lambda
  + There can be only abstract methods (pure abstract method) and variables (static final constants) in the Java interface, not the method body. It is used to achieve abstraction and multiple inheritance in Java. Interfaces form a contract between the class and the outside world. **Java Interface** also represents the **IS-A** relationship.
  + In the Java programming language, an ***interface*** is a reference type, similar to a class that can contain *only* constants, method signatures, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods. Interfaces cannot be instantiated—they can only be ***implemented*** by classes or ***extended*** by other interfaces. To implement an interface we use the keyword ‘**implements’**
  + **// A simple interface**
  + **Public interface Player**
  + **{**
  + **Public static final int id = 10;**
  + **Public abstract int move();**
  + **}**
  + **Class InterfaceImpl implements <interface\_name>**
* **Nested Classes in Java**
  + In Java, it is possible to define a class within another class, such classes are known as nested classes.
  + They enable you to logically group classes that are only used in one place, thus this increases the use of encapsulation, and creates more readable and maintainable code.
  + The scope of a nested class is bounded by the scope of its enclosing class/
  + Nested classes are divided into two categories:
  + Static nested class: Nested classes that are declared static are called static nested classes.
  + Inner class: An inner class is a non-static nested class.
* 
* **Anonymous Inner Class in Java**
  + It is an inner class without a name and for which only a single object is created.
  + An anonymous inner class can be useful when making an instance of an object with certain “extras” such as overriding methods of a class or interface, without having to actually subclass a class.
  + An anonymous class has access to the members of its enclosing class.
  + An anonymous class cannot access local variables in its enclosing scope that are not declared as final or effectively final.
  + Like a nested class, a declaration of a type (such as a variable) in anonymous class shadows any other declarations in the enclosing scope that have the same name.
  + An anonymous class has access to the members of its enclosing class.
  + An anonymous class cannot access local variables in its enclosing scope that are not declared as final or effectively final.
  + Like a nested class, a declaration of a type (such as a variable) in anonymous class shadows any other declarations in the enclosing scope that have the same name.
  + An Anonymous inner class will be terminated with a semi colon (;)
* **ANNOTATION** : Meta information
  + @Trainee

**Nithin**

* + @Trainer

JASON

* + Annotations can be at the class, methods, and variable levels.
  + ‘@‘is used to mention Annotations.
  + Annotations are like interfaces
  + JDK has got 3 annotations
    - @override
    - @SuppressWarning
    - @Deprecate

**NOTE**: List<E> --<E> is called Type Safety and it helps avoiding class cast Exception, E represents the element

**NOTE**: date.getClass();//Never use deprecated methods , Eclipse shows a strike through for the methods in the suggestions , In the explanatory part, it shows that the method is deprecated and could not be used in the latest versions

**NOTE**: In method overloading, arguments cannot be same. Return type can be anything

* **forEach() method in Iterable interface**
  + Whenever we need to traverse through a Collection, we need to create an Iterator whose whole purpose is to iterate over, and then we have business logic in a loop for each of the elements in the Collection. We might get ConcurrentModificationException if the iterator is not used properly.
  + Java 8 has introduced forEach method in java.lang.Iterable interface so that while writing code we focus on business logic. The forEach method takes java.util.function.Consumer object as an argument, so it helps in having our business logic at a separate location that we can reuse.
* **Optional :**
  + To avoid null pointer ,
  + “Is present “
* **Java Method References:**
  + In method reference , we use **::** to represent the method reference
  + **FI = ::**
  + Java provides a new feature called method reference in Java 8.
  + Method reference is used to refer method of functional interface.
  + It is compact and easy form of lambda expression.
  + Each time when you are using lambda expression to just referring a method, you can replace your lambda expression with method reference.
  + There are following types of method references in java:
    - **Reference to a static method.**
      * **ContainingClass::staticMethodName**
      * //method reference
      * employees.forEach(System.out::println);
    - **Reference to an instance method.**
      * **containingObject::instanceMethodName**
      * //method reference
      * We are referring non-static methods.
    - **Reference to a constructor.**
* **STREAMS:** 
  + Streams are classified into:
    - Parallel Stream
    - Sequential Stream
  + Streams are applied where the data source has more than one data. Ex : files, collections,
  + Intermediate Operation:
    - Lazy loading , no operation is done until Terminal operation
    - There can be no/ 1 / more intermediate operation
    - **Here is the list of all Stream intermediate operations:**
      * **filter()**
      * **map()**
      * **flatMap()**
      * **distinct()**
      * **sorted()**
      * **peek()**
      * **limit()**
      * **skip()**
  + Terminal Operation:
    - ( ) ,these are functions
    - There can be only one Terminal operation.
    - **Here is the list of all Stream terminal operations:**
      * **toArray()**
      * **collect()**
      * **count()**
      * **reduce()**
      * **forEach()**
      * **forEachOrdered()**
      * **min()**
      * **max()**
      * **anyMatch()**
      * **allMatch()**
      * **noneMatch()**
      * **findAny()**
      * **findFirst()**
* **SUN Technology** 
  + **J2SE/JSE**
    - Standard Edition.
    - Latest version (1.18)
    - Training (1.08)
    - CORE JAVA
    - JDBC java.sql
  + **J2EE/JEE**
    - **Components** - Servlet, JSP (Java Server Page)
    - Used to build Web application
    - **Java Bean**
    - **EJB** – Enterprise Java Bean
      * Most of the service/domain layer is done in EJB
    - **JNDI** – Java Naming and Directory Interface
    - **JAX WS** - Java API for XML Web Services (JAX-WS) is a standardized API for creating and consuming SOAP (Simple Object Access Protocol) web services.

It is also k/as XML /SOAP web services

* + - **JAX RS** – JAVAL XML RESTful service. (REST – Representational state transfer)
    - **JDBC** - javax.sql (x stands for extension of core java)

**NOTE: FRAMEWORK:**

* + - Build on top of JEE [ DOA layer – Data Access Object],
    - customizable ,
    - Pre-defined templates are available.
    - Ex: ORM
  + **ORM** 
    - Object Relational Mapping. (persistency -- > JDBC -- > ORM )
    - **Tools** – **Hibernate**, MyBatis / IBatis , TopLink
  + **Spring** 
    - Open Source / Free ware
    - PCF – Pivotal Cloud Foundary(cloud)

DOA Layer/

Persistent Layer/

Repository

Domain Layer

Web Layer/ Presentation Layer/ UI Layer

Oracle,

MySQL

Tech: JDBC, ORM, hibernate

Frameworks:

Struts, JSP, ADP,

Spring MVC Later

Frameworks: JDBC, ORM, Hibernate, SPRING JDBC, Spring JPA

Frameworks:

EJB, to replace EJB

SPRING (5.3.18)

S{ROINMG

Cloud

Spring MVC Later SPRING (5.3.18) SPRING JDBC, Spring JPA

SPRING

MVC Design Pattern - Model View Controller

* + **J2ME**

**Pre-requisites for Spring:**

* + - **JDK 1.8**
    - **Database**
      * **MySQL**
    - **Build Tool**
      * **Maven / Gradel – pom.xml with dependencies**
    - **IDE – Integrated Development Environment**
      * **ECLIPSE**
      * **STS - Spring tool Suite (has built-in support for spring)**
      * **JDeveloper - ( Oracle )**
      * **Intellij**

# SPRING

Spring is a framework and can only be used by java resources. Version Spring 5.x (cloud).

We can work on Spring using 3 ways.

1. XML - Parser - (negatives – time delay and the involvement of 3rd party), Hence Avoided using

2. @Annotations

3. Java Based.

🡪 Container (life cycle of the bean – Hook)

🡪Bean Factory - Deprecated

🡪ApplicationContext - Superset of BeanFactory

Spring uses:

* **IOC** (**Inversion Of Control**).
  + If we are using Spring avoid using the keyword “new”.
  + Use a configuration file instead.
    - Configuration -- > <filename>.xml -needs Parser <t>data</t>
  + We can also use @ (annotations) - Java base
  + IOC is used to transfer the control to the Spring so that the memory management, POJO files, object creation, deletion are taken care by the Spring.
  + **EAGER LOADING**: Every bean will be instantiated by the container. But it creates only 1 object. So we call it as a Design Pattern. Also k/as SINGLETON.
* DI – Dependency Injection.
* AOP – Aspect Oriented Programming.
* SPEL – Spring Expression Language. It uses $ { } notation.

**STEP 1:**

* Group ID in maven: package name == folder [ex: com.stg.ford…]
* Artifact ID : project name
* Packaging : jar [zip files in windows]
* In Spring, any .java file associated with Spring we call it as “BEAN”.
* Container helps to load the XML config files into the java main file. Container actually takes care of the life cycle of the bean.
* In Spring we have two Container:
  + **BeanFactory :** it produces beans (but deprecated).It’s an interface
  + **ApplicationContext: ApplicationContext extends BeanFactory**).It’s an interface

ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext("com/config/springconfig.xml");

* Employee obj = (Employee) applicationContext.getBean("employee");
  + **EAGER LOADING**: Every bean will be instantiated by the container. But it creates only 1 object. So we call it as a Design Pattern. Also k/as SINGLETON.

ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext("com/config/springconfig.xml");

* Employee obj = (Employee) applicationContext.getBean("employee");
  + Employee obj2 = (Employee) applicationContext.getBean("employee");

🡪 **Bean:**

A bean is **an object that is instantiated, assembled, and otherwise managed by a Spring IoC container**. Otherwise, a bean is simply one of many objects in your application. Beans, and the dependencies among them, are reflected in the configuration metadata used by a container.

Injection (how to pass values to bean)

🡪 Conctructor Injection <constructor-arg>

🡪Setter Injection <property>

Spring Based Project:

1. IDE 🡪 Eclipse🡪 Build tool 🡪 Maven
2. STS (Spring tool suite)

<bean id=*"student"* class=*"com.model.Student"*></bean>

**<property>:**

<bean id=*"student2"* class=*"com.model.Student"* scope=*"prototype"*>

<property name=*"studentId"* value=*"132"*></property>

<property name=*"studentName"* value=*"afaf"*></property>

<property name=*"mark"* value=*"123"*></property>

</bean>

**<Constructor-arg>:**

In SrpingContext.xml/springconfig.xml

<bean id=*"student"* class=*"com.model.Student"* scope=*"prototype"*>

<constructor-arg value=*"30"* name=*"studentId"*></constructor-arg>

<constructor-arg value=*"wednesday"* name=*"studentName"*></constructor-arg>

<constructor-arg value=*"123"* name=*"mark"*></constructor-arg>

</bean>

In main.java

ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext("springconfig.xml");

Student student = (Student) applicationContext.getBean("student");

// type casting

System.***out***.println("Student ID : " + student.getStudentId());

System.***out***.println("Student Name : " + student.getStudentName());

System.***out***.println("Student Mark : " + student.getMark());

* **DEPENDENCY INJECTION :** 
  + HAS A relationship:
    - One to one
    - One to many
* COLLECTION USAGE EXAMPLES:
  + MAPS , SETS , LISTS , PROPS

<bean id="moreComplexObject" class="example.ComplexObject">

**<!-- results in a setAdminEmails(java.util.Properties) call -->**

<property name="adminEmails">

<props>

<prop key="administrator">administrator@example.org</prop>

<prop key="support">support@example.org</prop>

<prop key="development">development@example.org</prop>

</props>

</property>

**<!-- results in a setSomeList(java.util.List) call -->**

<property name="someList">

<list>

<value>a list element followed by a reference</value>

<ref bean="myDataSource" />

</list>

</property>

**<!-- results in a setSomeMap(java.util.Map) call -->**

<property name="someMap">

<map>

<entry key="an entry" value="just some string"/>

<entry key="a ref" value-ref="myDataSource"/>

</map>

</property>

**<!-- results in a setSomeSet(java.util.Set) call -->**

<property name="someSet">

<set>

<value>just some string</value>

<ref bean="myDataSource" />

</set>

</property>

</bean>

//value can be used for system defined data types

// ref can be used for user defined data type in the xml

**Properties FILE :**

<FILENAME>.properties.

Almost like a .txt file

LHS value is taken by the string and substituted by the RHS value

**SPEL** : SPRING EXPRESSION LANGUAGE . IT is represented by ${ }

The Spring Expression Language (“SpEL” for short) is a powerful expression language that supports querying and manipulating an object graph at runtime. SpEL is based on a technology-agnostic API that lets other expression language implementations be integrated, should the need arise.

**INTERNATIONALIZATION : I18N**

What is i18n mean?

1. Internationalization

Internationalization (sometimes shortened to "I18N , meaning "**I - eighteen letters -N**") is the process of planning and implementing products and services so that they can easily be adapted to specific local languages and cultures, a process called localization.

Spring with XML 🡪 performance issue.

Java API for XML processing 🡪 A parser, (3rd party) JAXP used to read from XML.

To overcome this, ANNOTATION is used

**ANNOTATION**: @Metainfo.

@ can be at variable level, method level, and class level.

Spring IOC 🡪 we configured in XML 🡪loaded using 🡪 Spring container ( BeanFactory , ApplicationContext).We will avoid using the keyword ‘new’. Container takes care of Lifecycle.Bean is simply a .java file associated with Spring.

Now 🡪 Spring 🡪 .java🡪 XML/ @Annotation 🡪 Spring takes the control 🡪IOC

**DI 🡪 Dependency Injection**

OOP Has a relation (one to one , one to many)

In XML , we make use of DI by using the keyword ‘**’ref’**

Ref has the ID of the another object which has to be referred.

**Autowiring in Spring :**

* Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.
* Autowiring can't be used to inject primitive and string values. It works with reference only. It requires the **less code** because we don't need to write the code to inject the dependency explicitly.

|  |  |  |
| --- | --- | --- |
| **No.** | **Mode** | **Description** |
| 1) | no | It is the default autowiring mode. It means no autowiring bydefault. |
| 2) | byName | The byName mode injects the object **dependency according to *name* of the bean**. In such case, property name and bean name must be same. It internally calls setter method. |
| 3) | byType | The byType mode injects the object **dependency according to *type***. So property name and bean name can be different. It internally calls setter method. |
| 4) | constructor | The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters. |

.

**SPRING WITH ANNOTATION: [“@” metainfo ] :**

**XML - XML based**

**XML + annotation - Annotation**

**Annotation - Java Based**

**<bean id = “???“ class = “com.model.Employee”>**

**applicationContext.getBean(“???”);**

**<bean id = “employee” class = “com.model.Employee”>**

**applicationContext.getBean(“employee”);**

**<bean id = “student” class = “com.model.Student”>**

**applicationContext.getBean(“student”);**

***Spring core Application Development steps:***

* **Pre – requisites:**

JDK 1.8

MAVEN 3.8

IDE – Eclipse/ STS

* **Step 1:**

**Create** a **Maven project** .

* **Step 2:**

**Pom.xml**

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

<spring.version>5.3.18</spring.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring ORM -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>${spring.version}</version>

</dependency>

</dependencies>

* **Step 3 :**

For the first time we use **: cmd > mvn clean install**

It helps in downloading the files in .m2 repo;

* **Step 4 :**

**Create packages.**

com.model

com.service

com.main

com.config

com.exception

* **Step 5:**

***Create a POJO file.***

***Generate Getters and setters***

***Address***

**package** com.model;

**import** org.springframework.stereotype.Component;

//@Repository

//@Service

@Component (value = "break") //alias name ==another name to refer

**public** **class** Address {

**private** **int** doorNo;

**private** String city;

**private** String state;

**public** Address() {

**super**();

System.***out***.println("Eager loading");

}

**public** Address(**int** doorNo, String city, String state) {

**super**();

**this**.doorNo = doorNo;

**this**.city = city;

**this**.state = state;

}

**public** **int** getDoorNo() {

**return** doorNo;

}

**public** **void** setDoorNo(**int** doorNo) {

**this**.doorNo = doorNo;

}

**public** String getCity() {

**return** city;

}

**public** **void** setCity(String city) {

**this**.city = city;

}

**public** String getState() {

**return** state;

}

**public** **void** setState(String state) {

**this**.state = state;

}

@Override

**public** String toString() {

**return** "Address [doorNo=" + doorNo + ", city=" + city + ", state=" + state + "]";

}

}

* **Step 6 :**

***Create the configuration file***

***MyAnnotationsConfig***

**package** com.config;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.ComponentScan;

**import** org.springframework.context.annotation.Configuration;

**import** com.model.Address;

@Configuration //replacement of XML

@ComponentScan(basePackages = "com.model")

**public** **class** MyAnnotationsConfig {

}

* **Step 7 :**

**Load the configuration file through the main class**

***AnnotationsMainApp***

**package** com.main;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.annotation.AnnotationConfigApplicationContext;

**import** com.model.Address;

**public** **class** AnnotationsMainApp {

**public** **static** **void** main(String[] args) {

// load XML in general by ApplicationContext

ApplicationContext applicationContext = **new** AnnotationConfigApplicationContext(

com.config.MyAnnotationsConfig.**class**);

Address address = (Address) applicationContext.getBean("break");

// Address address = (Address) applicationContext.getBean("address");

System.***out***.println(address.getDoorNo());

System.***out***.println(address.getCity());

System.***out***.println(address.getState());

System.***out***.println("the end");

System.***out***.println("the end");

}

}

**XML + ANNOTATION:**

Add the below content in the XML file

XML and annotation

-------------------

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd">

<context:component-scan base-package="com" />

</beans>

**<Constructor-arg> - Constructor injection**

**Annotations:**

@Configuation

@Bean

@Service

@Repository

@Value //constructor injection

@Component

@ComponentScan

**Element :** component-scan

Scans the classpath for annotated components that will be auto-registered as Spring beans. By default, the

Spring-provided @Component, @Repository, @Service, @Controller, @RestController, @ControllerAdvice, and

@Configuration stereotypes will be detected. Note: This tag implies the effects of the 'annotation-config' tag,

activating @Required, @Autowired, @PostConstruct, @PreDestroy, @Resource, @PersistenceContext and

@PersistenceUnit annotations in the component classes, which is usually desired for autodetected components

(without external configuration). Turn off the 'annotation-config' attribute to deactivate this default behavior, for

example in order to use custom BeanPostProcessor definitions for handling those annotations. Note: You may

use placeholders in package paths, but only resolved against system properties (analogous to resource paths). A

component scan results in new bean definitions being registered; Spring's

PropertySourcesPlaceholderConfigurer will apply to those bean definitions just like to regular bean definitions,

but it won't apply to the component scan settings themselves. See javadoc for

org.springframework.context.annotation.ComponentScan for information on code-based alternatives to

bootstrapping component-scanning.

**Content Model :** (include-filter\*, exclude-filter\*)

@Lazy

@Lazy(value=true) //eager loading

@Autowired //Dependency Injection [in xml – we use ref,autowired]

//@@Autowired

//private <interface\_name> variableName ;

@Autowired

**private** DepartmentService departmentService;

**JSE** – Stand-alone application -- GUI -- they use AWT/ Swing

JEE – Enterprise --WEB Application -- internet based application

Prerequisite for JEE is **WEB SERVER** [which can handle HTTP request (client) and response (server) ]

Using JAVA 🡪 to **Web Application** 🡪 to create a Web server we can use **Apache Tomcat** (freeware)

🡪**Application Server** 🡪 App Server = Web server + some additional information

🡪Examples of Application server:

* + WebLogic - IBM
  + WebSphere - Oracle
  + JBoss - RedHat
  + Glassfish - Sun

All of these have inbuilt web server

To Start webserver – TOMCAT:

Step 1: PATH is mapped to the location of JDK\bin

Cmd> echo %PATH%

Step 2 : JAVA\_HOME = location of JDK

Cmd> echo %JAVA\_HOME%

[**http://localhost:8080/**](http://localhost:8080/)

**CTRL + C -- to stop the server**

**WEB APPLICATION with the help of JEE**

1) **Servlet,** these 2 are web components

2) **JSP** (JAVA SERVER PAGE) these are reusable

* Servlet == controller
* JSP == View
* POJO == MODEL
* Design Pattern === MVC == Model view controller

***SPRING MVC:***

Deployment Descriptor (web.xml) 🡪 Spring controller = Dispatcher Servlet (given by or.springframework)

C:/Users/maniceto/Documents/workspace/.metadata/.plugins/org.eclipse.wst.server.core/

Could not delete C:/Users/nithinkumarp/eclipse-workspace/.metadata/.plugins/org.eclipse.wst.server.core/tmp0/wtpwebapps/Sprinmvcdemo1/WEB-INF/lib. May be locked by another process.

<!-- Specifying base package of the Components like Controller, Service,DAO -->

<context:component-scan base-package="com" />

<annotation-driven />

<!-- Getting Database properties -->

<context:property-placeholder location="classpath:application.properties" />

<!--Specifying the Resource location to load JS, CSS, Images etc -->

<resources mapping="/resources/\*\*" location="/resources/" />

<!-- View Resolver -->

<beans:bean

class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<beans:property name="prefix" value="/WEB-INF/pages/" />

<beans:property name="suffix" value=".jsp" />

</beans:bean>

***WEB SERVICE:***

JEE -

* JAX WS - JAVA API for XML Web Services
* JAX RS – JAVA API for RESTful services
* What is meant by service?
  1. A Function/Business logic / behaviour which is exposed to WWW.
  2. It is reusable/ avoiding re-creating.
  3. Interoperability

**Interoperability** – if the app is developed by Java it can be read by .net, php ,any other

Java follows certain standard of giving output as XML file. So any programming language which can read XML file can be use interoperability.

Java renders/gives out XML output by using an API called JAXP and JAXB

JAXP – JAVA API for XML Parser. (To read an XML file JAXP)

JAXB- JAVA API for XML Binding (to generate an XML file).

JAXP, JAXB - components of the JEE

How Java App is exposed as a service using XML format. But XML cannot be passed/travel through HTTP. So we put the XML file into a wrapper called SOAP. (it acts like an envelope).

Hence today JAX WS== can also k/as XML Web service. And also SOAP Service (Simple Object Access Protocol). These days SOAP is deprecated and moved to RESTful service

XML - Extensible Mark-up Language

XML format - <tag></tag>

Framework – uses JAX WS/JAX RS

RMI - Remote method invocation

COM

CORBA

WEB SERVICES

Web service can be created in two ways.

* + - 1. Top down Approach wsdl 🡪 java
      2. Bottom Up Approach .java 🡪 XML file(bottom up)

XML file is called – WSDL file

Web Services Description Language.WSDL is an XML notation for describing a web service. A WSDL definition tells a client how to compose a web service request and describes the interface that is provided by the web service provider.

MyApp 🡪 JAVA 🡪 another java

**Demo to create JAX WS :**

**Prerequisites-**

Any version of JDK 1.8

Web Server - Apache Tomcat

Web Service Server – Axis [Axis is also a part of Apache]

IDE – Eclipse

**Step 1:**

create - Dynamic web project

* Dynamic web module version , choose 2.5
* Give project name
* Use Apache

Step 2 :

Create package, right click on src/main , create package, insider package create a class

Business logic is written in class.

Step 3 :

Right click on the java file in the left panel and choose “create web services”

Check the server version , service project name and service run time , click next and choose the methods required . set the approach in the drop down (bottom –up or top to bottom) and the start the server and don’t publish(if not required)

Use the url to cross check in the browser

**http://localhost/8080/<project\_name>/services**

<http://localhost/8080/soaptraildemo/services>

It gives a list services available on the page, which contains WSDL

**Working With WSDL File:**

WSDL stands for Web Service Description Language and it is platform-independent.. It is used to create a relationship between the service requester and the service provider. It is an XML based language to describe the functionality of a SOAP-based web service.

**WSDL Document Elements**

**Definitions**: It is the basic element of the WSDL document that contains the **definition of one** or more services.

**Types**: The Type element is used to give information about the **complicated data types** used within the WSDL document.

**Message**: It contains **abstract data** that is being used **in communication between client and web** server. It also defines the **data elements for each operation**.

**portType**: It contains the **collection of abstract operation** supported by one or more endpoints.

**Port**: It is used to define the **single endpoint as** an address for the binding.

**Services**: It is a **collection of endpoint networks** that specify the port address for the binding.

**Binding**: It specifies how operations are implemented by **concrete protocols** and data format features for operation and messaging

<definitions>

<types>

Definition of types goes here.

</types>

<message>

Definition of a message goes here.

</message>

<portType>

<operation>

Definition of an operation goes here.

</operation>

</portType>

<binding>

Definition of a binding goes here.

</binding>

<service>

Definition of service goes here.

</service>

</definition>

Wsdl : Tmpbs

Types, Message, Port, Binding, Service

Testing of SOAP -> SOAP UI

**RESTful Web Service:**

Representational State Transfer (REST).

RESTful web services are built to work best on the Web

RESTful Web Services are basically REST Architecture based Web Services. REST was first introduced by Roy Fielding in 2000

REST uses various representation to represent a resource like text, JSON, XML. **JSON** is the most popular one (JAVA SCRIPT OBJECT NOTATION).

JSON Representation

{

key1: “value”,

key2: “value1”

}

Array of JSON objects [

{key1: “value”, key2: “value1”},

{key1: “value”, key2: “value1”},

{key1: “value”, key2: “value1”}

]

In REST architecture, a REST Server simply provides access to resources and REST client accesses and modifies the resources. Here each resource is identified by URIs/ global IDs. REST uses various representation to represent a resource like text, JSON, XML. JSON is the most popular one.

**HTTP methods**

|  |  |
| --- | --- |
| **S.N.** | **Method and Description** |
| 1 | **GET**  The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data. |
| 2 | **HEAD**  Same as GET, but transfers the status line and header section only. |
| 3 | **POST**  A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms. |
| 4 | **PUT**  Replaces all current representations of the target resource with the uploaded content. |
| 5 | **DELETE**  Removes all current representations of the target resource given by a URI. |
| 6 | **CONNECT**  Establishes a tunnel to the server identified by a given URI. |
| 7 | **OPTIONS**  Describes the communication options for the target resource. |
| 8 | **TRACE**  Performs a message loop-back test along the path to the target resource. |

The set of common methods for HTTP/1.1 is defined below and this set can be expanded based on requirements. These method names are case sensitive and they must be used in uppercase.

1. Following four HTTP methods are commonly used in REST based architecture.
   * 1. **POST** − Used to create a new resource. 🡪 CREATE
     2. **GET** − Provides a read only access to a resource. 🡪 READ
     3. **PUT** − Used to update a existing resource or create a new resource. 🡪 UPDATE
     4. **DELETE** − Used to remove a resource. 🡪 DELETE
2. URI (uniform resource Identifier)
3. URL (uniform resource Locator)
4. Every method/behaviour is called as RESOURCE.

Every Resource can be accessed by 🡪 URI (uniform resource Identifier) also k/as **END POINT** : it helps in identifying the logical /physical address of the resource

1. Frameworks to create a RESTful service :
2. **Jersey** Framework to create RESTful Web Services.
3. **RESTeasy**
4. **Spring REST**

**Demo to create RESTful Services**

**Prerequisite:**

* 1. JDK 1.8
  2. Build tool – Maven
  3. Web server – Apache Tomcat
  4. API -- > Spring Web jar files - \*.jar
  5. IDE - Eclipse/sti/intellij
  6. REST container – Spring --> ApplicationContext

**STEP 1:**

Set **PATH** variable, **environment** variable.

SET **M2\_HOME** :

**D:\Java\_Software\apache-maven-3.8.4-bin\apache-maven-3.8.4\bin**

**🡪 set PATH** =

**C:\Program Files\Java\jdk1.8.0\_77\bin;**

%PATH%

**D:\Java\_Software\apache-maven-3.8.4-bin\apache-maven-3.8.4\bin**

* **set JAVA\_HOME** =

**C:\Program Files\Java\jdk1.8.0\_77**

Check

* Cmd > java –version
* Cmd > javap –version
* Cmd > mvn to check maven
* Cmd > echo %PATH%  
  cmd > echo %JAVA\_HOME%
* Cmd > echo %M2\_HOME%

**STEP 2**:

1. Create **Maven** project 🡪 **group** id – <packagename> com

Artefact **id** - < projectname> springrest

**Packaging** - <webapp> war

**WAR files:**

web application archive (**WAR**) files. WAR - In software engineering, a WAR file is a file used to distribute a collection of JAR-files, Java Server Pages, Java Servlets, Java classes, XML files, tag libraries, static web pages and other resources that together constitute a web application.

**Step 3:**

Create a **folder** for “**WEB-INF**” under **src/main/webapp** (all in caps)

**Step 4 :**

Create / copy under **web.xml** the WEB-INF folder the DD (**deployment descriptor** .i.e**., web.xml**)

(front controller - DispatcherServlet)

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns=*"http://java.sun.com/xml/ns/javaee"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"*

id=*"WebApp\_ID"* version=*"3.0"*>

<display-name>Archetype Created Web Application</display-name>

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/dispatcher-servlet.xml</param-value>

</context-param>

</web-app>

NOTE :

* **load on start-up in web.xml**

1. The load-on-start-up element of web-app loads the servlet at the time of deployment or server start if value is positive. It is also known as **pre initialization of servlet**.
2. The element load-on-startup indicates that this servlet should be loaded (instantiated and have its init() called) on the startup of the Web application. The element content of this element must be an integer indicating the order in which the servlet should be loaded. If the value is a negative integer, or the element is not present, the container is free to load the servlet whenever it chooses. If the value is a positive integer or 0, the container must load and initialize the servlet as the application is deployed. The container must guarantee that servlets marked with lower integers are loaded before servlets marked with higher integers. The container may choose the order of loading of servlets with the same load-on-startup value.

* You can pass positive and negative value for the servlet
* This is a front controller and it is a design pattern, Every web request would go through this front controller. (**springframework.Dispatcherservlet)**

**Step 5:**

Create/copy/Configure the Spring configuration file and name as per

**< servlet-name >**tag in the web.xml file **<servlet-name >-servlet.xml** under **the WEB-INF** folder

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd"*>

<context:component-scan

base-package=*"com.controller"* />

<!-- configure bean to convert JSON to POJO and vice versa -->

<bean id=*"jsonMessageConverter"*

class=*"org.springframework.http.converter.json.MappingJackson2HttpMessageConverter"*>

</bean>

<!-- Configure bean to convert POJO to XML and vice versa -->

<bean id=*"xmlMessageConverter"*

class=*"org.springframework.http.converter.xml.Jaxb2RootElementHttpMessageConverter"*>

</bean>

<bean

class=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"*>

<property name=*"messageConverters"*>

<list>

<ref bean=*"jsonMessageConverter"* />

<ref bean=*"xmlMessageConverter"* />

</list>

</property>

</bean>

</beans>

**STEP 6:**

* Configure the dependencies in the pom.xml file
* Traines demo folder. -> springrest folder ->pom file

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>com</groupId>

<artifactId>springrest</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<name>SpringRestMaven</name>

<properties>

<spring.version>5.2.8.RELEASE</spring.version>

<jackson.databind-version>2.9.0</jackson.databind-version>

<org.slf4j-version>1.7.5</org.slf4j-version>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

**<!-- Spring dependencies -->**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

**<!-- Jackson -->**

**<!-- helps in converting to JSON file -->**

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>${jackson.databind-version}</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>${org.slf4j-version}</version>

</dependency>

</dependencies>

<build>

<sourceDirectory>src/main/java</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

***jackson.databind***

Data Binding API is used to convert JSON to / from POJO (Plain Old Java Object) using property accessor or using annotations. It is of two type.

* Simple Data Binding - Converts JSON to and from Java Maps, Lists, Strings, Numbers, Booleans and null objects.
* Full Data Binding - Converts JSON to and from any JAVA type.

ObjectMapper reads/writes JSON for both types of data bindings. Data Binding is most convenient way and is analogous to **JAXB parser for XML**

**Step 7 :**

Create packages in springrest/src/main/java , com.controller ,com.model , com.service

**Step 8:**

**Create a POJO file / BEAN** - Src/main/java/com.model

**public** **class** User {

**private** **int** userId;

**private** String userName;

**private** String password;

}

Generate constructors and getters,setters;

**Step 9:**

**Create REST controller**

Src/main/java/com.controller

@RequestMapping(value = "/getDept", produces = MediaType.***APPLICATION\_JSON\_VALUE***)

@RequestMapping(value = "/getDept", produces = MediaType.***APPLICATION\_JSON\_VALUE***, method = RequestMethod.***GET***)

**SOA – Service Oriented Architecture:**

**SOA** is a design pattern. It is used to build distributed systems that deliver services to other applications through the protocol. It is only a concept and not limited to any programming language or platform.

**What is Service?**

A service is a well-defined, self-contained function that represents a unit of functionality. A service can exchange information from another service. It is not dependent on the state of another service. It uses a loosely coupled, message-based communication model to communicate with applications and other services.

**Services** - The services are the logical entities defined by one or more published interfaces.

**Service provider** - It is a software entity that implements a service specification.

**Service consumer** - It can be called as a requestor or client that calls a service provider. A service consumer can be another service or an end-user application.

**Service locator** - It is a service provider that acts as a registry. It is responsible for examining service provider interfaces and service locations.

**Service broker** - It is a service provider that pass service requests to one or more additional service providers.

**Multipurpose Internet Mail Extensions or MIME type:**

**What is MIME type?**

A multipurpose internet mail extension, or MIME type, is an internet standard that describes the contents of internet files based on their natures and formats. This cataloging helps the browser open the file with the appropriate extension or plugin.

# **TESTING:**

Development – two types of testing : TDD – Test Driven Development - preferred these days

BDD- **Behavior-driven development** (Business)

**Test-driven development (TDD)** is a software development process relying on software requirements being converted to test cases before software is fully developed, and tracking all software development by repeatedly testing the software against all test cases.

**Behavior-driven development (BDD**) is an Agile software development methodology in which an application is documented and designed around the behavior a user expects to experience when interacting with it.

**UNIT testing:** Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation.

Junit - class / Function

Test Case for every behaviour ( +ve ,-ve, 0 scenario)

+ve - Happy Path

-ve - Sad Path

Framework - JUNIT

* **Junit 4** and 5 versions.

Test Suite collection of test case

Every class would have a function and every function should be tested. If all the functions in the class is tested then the test Coverage

**JUnit unit testing**

JUnit is a Regression Testing Framework used by developers to implement unit testing in Java, and accelerate programming speed and increase the quality of code. JUnit Framework can be easily integrated with either of the following − Eclipse. Ant. Maven

JUNIT functions does not returns a value. Only functions which returns can only be tested

JUnit test framework provides the following important features −

* Fixtures
* Test suites
* Test runners
* JUnit classes

**DEMO :**

1. **Create a Maven project**
2. **Add dependencies – Add JDK , Junit , Hamcrest**

**package** com.service;

**import** org.junit.After;

**import** org.junit.AfterClass;

**import** org.junit.Before;

**import** org.junit.BeforeClass;

**import** org.junit.Test;

**public** **class** JunitLifeCycle {

@BeforeClass

**public** **static** **void** initClass() {

System.***out***.println("This is class initialisation------ Setup");

}

@Before

**public** **void** init() {

System.***out***.println("This is initialisation------ Setup");

}

@Test

**public** **void** sayHello(){

System.***out***.println("Say Hello function is tested 0");

}

@Test

**public** **void** sayHello1(){

System.***out***.println("Say Hello function is tested 1");

}

@Test

**public** **void** sayHello2(){

System.***out***.println("Say Hello function is tested 2");

}

@After

**public** **void** bye() {

System.***out***.println("Good Bye");

}

@AfterClass

**public** **static** **void** finalbye() {

System.***out***.println("Good Bye tata see you from class");

}

}

# Core Spring Framework Annotations

**@Required**: It applies to the bean setter method. It indicates that the annotated bean must be populated at configuration time with the required property, else it throws an exception BeanInitilizationException.

**@Autowired**: Spring provides annotation-based auto-wiring by providing @Autowired annotation. It is used to autowire spring bean on setter methods, instance variable, and constructor. When we use @Autowired annotation, the spring container auto-wires the bean by matching data-type.

**@Configuration:** It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions.

**@ComponentScan**: It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.

**@Bean:** It is a method-level annotation. It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed by Spring Container.

# Spring Framework Stereotype Annotations

**@Component**: It is a class-level annotation. It is used to mark a Java class as a bean. A Java class annotated with @Component is found during the classpath. The Spring Framework pick it up and configure it in the application context as a Spring Bean.

**@Controller:** The @Controller is a class-level annotation. It is a specialization of @Component. It marks a class as a web request handler. It is often used to serve web pages. By default, it returns a string that indicates which route to redirect. It is mostly used with @RequestMapping annotation.

**@Service:** It is also used at class level. It tells the Spring that class contains the business logic.

**@Repository:** It is a class-level annotation. The repository is a DAOs (Data Access Object) that access the database directly. The repository does all the operations related to the database.

# Spring Boot Annotations

**@EnableAutoConfiguration**: It auto-configures the bean that is present in the classpath and configures it to run the methods. The use of this annotation is reduced in Spring Boot 1.2.0 release because developers provided an alternative of the annotation, i.e. @SpringBootApplication.

**@SpringBootApplication**: It is a combination of three annotations @EnableAutoConfiguration, @ComponentScan, and @Configuration.

**Spring MVC and REST Annotations**

**@RequestMapping:** It is used to map the web requests. It has many optional elements like consumes, header, method, name, params, path, produces, and value. We use it with the class as well as the method.

**@GetMapping:** It maps the HTTP GET requests on the specific handler method. It is used to create a web service endpoint that fetches It is used instead of using: @RequestMapping(method = RequestMethod.GET)

**@PostMapping:** It maps the HTTP POST requests on the specific handler method. It is used to create a web service endpoint that creates It is used instead of using: @RequestMapping(method = RequestMethod.POST)

**@PutMapping:** It maps the HTTP PUT requests on the specific handler method. It is used to create a web service endpoint that creates or updates It is used instead of using: @RequestMapping(method = RequestMethod.PUT)

**@DeleteMapping:** It maps the HTTP DELETE requests on the specific handler method. It is used to create a web service endpoint that deletes a resource. It is used instead of using: @RequestMapping(method = RequestMethod.DELETE)

**@PatchMapping:** It maps the HTTP PATCH requests on the specific handler method. It is used instead of using: @RequestMapping(method = RequestMethod.PATCH)

**@RequestBody:** It is used to bind HTTP request with an object in a method parameter. Internally it uses HTTP MessageConverters to convert the body of the request. When we annotate a method parameter with @RequestBody, the Spring framework binds the incoming HTTP request body to that parameter.

**@ResponseBody:** It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.

**@PathVariable:** It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple @PathVariable in a method.

**@RequestParam:** It is used to extract the query parameters form the URL. It is also known as a query parameter. It is most suitable for web applications. It can specify default values if the query parameter is not present in the URL.

**@RequestHeader:** It is used to get the details about the HTTP request headers. We use this annotation as a method parameter. The optional elements of the annotation are name, required, value, defaultValue. For each detail in the header, we should specify separate annotations. We can use it multiple time in a method

**@RestController:** It can be considered as a combination of @Controller and @ResponseBody annotations. The @RestController annotation is itself annotated with the @ResponseBody annotation. It eliminates the need for annotating each method with @ResponseBody.

**@RequestAttribute:** It binds a method parameter to request attribute. It provides convenient access to the request attributes from a controller method. With the help of @RequestAttribute annotation, we can access objects that are populated on the server-side.

**Spring – @GetMapping and @PostMapping**

**Spring WebMVC:**

Learn to create Spring WebMVC controllers with @Controller annotation and map HTTP requests with annotations like @RequestMapping, @GetMapping, @PostMapping, @PutMapping, @DeleteMapping and @PatchMapping.

**Table Of Contents**

1. Request Mapping Annotations

2. Spring @GetMapping Example

3. Spring @PostMapping Example

4. Shared Class Level Attributes

5. Difference between @PostMapping and @RequestMapping

6. Summary

**1. Request Mapping Annotations**

Before Spring 4.3, Spring had only @RequestMapping annotation for mapping all the incoming HTTP request URLs to the corresponding controller methods.

For example, in the given below code, we are using the @RequestMapping annotation to map 3 different HTTP requests to their respective controller methods. Notice that we have specified the HTTP request type (GET, POST etc.) as the annotation attribute method.

Mapping controller methods with @RequestMapping

@RequestMapping(value = "/users", method = RequestMethod.GET)

public Users getUsers() {

}

@RequestMapping(value = "/users", method = RequestMethod.POST)

public User createUser(User user) {

}

@RequestMapping(value = "/users/{id}", method = RequestMethod.GET)

public User getUser(@PathVariable("id") String id) {

}

Spring 4.3 introduced five new and more specific annotations for each HTTP request type.

**@GetMapping**

**@PostMapping**

**@PutMapping**

**@DeleteMapping**

**@PatchMapping**

Using these new annotations, we can rewrite the @RequestMapping example as given below. If you see carefully, we do not have the method attribute anymore in the new annotations.

Mapping controller methods with more specific annotations

@GetMapping(value = "/users")

public Users getUsers() {

}

@PostMapping(value = "/users")

public User createUser(User user) {

}

@GetMapping(value = "/users/{id}")

public User getUser(@PathVariable("id") String id) {

}

**2. Spring @GetMapping Example**

The @GetMapping annotation is a specialized version of @RequestMapping annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET).

The @GetMapping annotated methods in the @Controller annotated classes handle the HTTP GET requests matched with given URI expression.

Let us understand with example how to write controller methods mapped with @GetMapping annotations.

@GetMapping Example

@RestController

public class UserController {

@Autowired

UserService userService;

@GetMapping("users")

public ResponseEntity<List<User>> getAll() {

return new ResponseEntity<>(userService.getAll(), HttpStatus.OK);

}

@GetMapping("users/{id}")

public ResponseEntity<User> getById(@PathVariable long id) {

Optional<User> user = userService.getById(id);

if (user.isPresent()) {

return new ResponseEntity<>(user.get(), HttpStatus.OK);

} else {

throw new RecordNotFoundException();

}

}

}

**3. Spring @PostMapping Example**

The @PostMapping is specialized version of @RequestMapping annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.POST).

The @PostMapping annotated methods in the @Controller annotated classes handle the HTTP POST requests matched with given URI expression.

Let us understand with example how to write controller methods mapped with @PostMapping annotations.

@PostMapping Example

@PostMapping(path = "users",

consumes = MediaType.APPLICATION\_JSON\_VALUE,

produces = MediaType.APPLICATION\_JSON\_VALUE)

public ResponseEntity<User> create(@RequestBody User newUser) {

User user = userService.save(newUser);

if (user == null) {

throw new ServerException();

} else {

return new ResponseEntity<>(user, HttpStatus.CREATED);

}

}

**4. Shared Class Level Attributes**

All the above-discussed request mapping annotations such as @RequestMapping, @GetMapping, @PostMapping etc., inherit the annotation attributes values from the @RequestMapping annotation applied at the @Controller class.

The method-level annotations may override the default values by providing their own set of values.

For example, in HomeController.java, @RequestMapping annotation at line no. 4 provides the default values of produces attribute. It means that all the request handler methods in this controller class will, by default, will return the JSON response.

But addMember\_V2() method at line no. 12, overrides the produces attribute and it will return the XML response to the clients.

Note that addMember\_V1() method will produce the content in default media type i.e. application/json.

HomeController.java

package com.howtodoinjava.web;

@Controller

@RequestMapping(path = "/", produces = MediaType.APPLICATION\_JSON\_VALUE)

public class HomeController

{

@PostMapping(path = "/members")

public void addMember\_V1(@RequestBody Member member) {

//code

}

@PostMapping(path = "/members", produces = MediaType.APPLICATION\_XML\_VALUE)

public void addMember\_V2(@RequestBody Member member) {

//code

}

}

**5. Difference between @PostMapping and @RequestMapping**

As noted above @PostMapping annotation is one specialized version of @RequestMapping annotation which handles only the HTTP POST requests.

The difference in one line

@PostMapping = @RequestMapping(method = { RequestMethod.POST })

Let’s see the difference between PostMapping and @RequestMapping annotations with a very simple example. Both versions in the given example will work exactly the same. They just have a slightly different syntax.

@RequestMapping(value = "/employees", method = RequestMethod.POST) //1

@PostMapping("/employees") //2

@PostMapping acts as a shortcut for @RequestMapping(method = RequestMethod.POST). We can see the sourcecode of the @PostMapping annotation which internally uses the @RequestMapping annotation.

PostMapping.java

@Target({ java.lang.annotation.ElementType.METHOD })

@Retention(RetentionPolicy.RUNTIME)

@Documented

@RequestMapping(method = { RequestMethod.POST })

public @interface PostMapping

{

//code

}

Spring MVC has made writing request handler controller classes and methods very easy. Just add a few annotations like @GetMapping and @PostMapping and put the class where component scanning can find them and configure them in the web application context.

It is also very easy to create attributes at the class level so that all handler methods inherit them by default, and can override them when needed.

Same way, you can use other request mapping annotations e.g. @PutMapping, @DeleteMapping and @PatchMapping.

@SpringBootApplication

A single @SpringBootApplication annotation can be used to enable those three features, that is:

@EnableAutoConfiguration: enable Spring Boot’s auto-configuration mechanism

@ComponentScan: enable @Component scan on the package where the application is located (see the best practices)

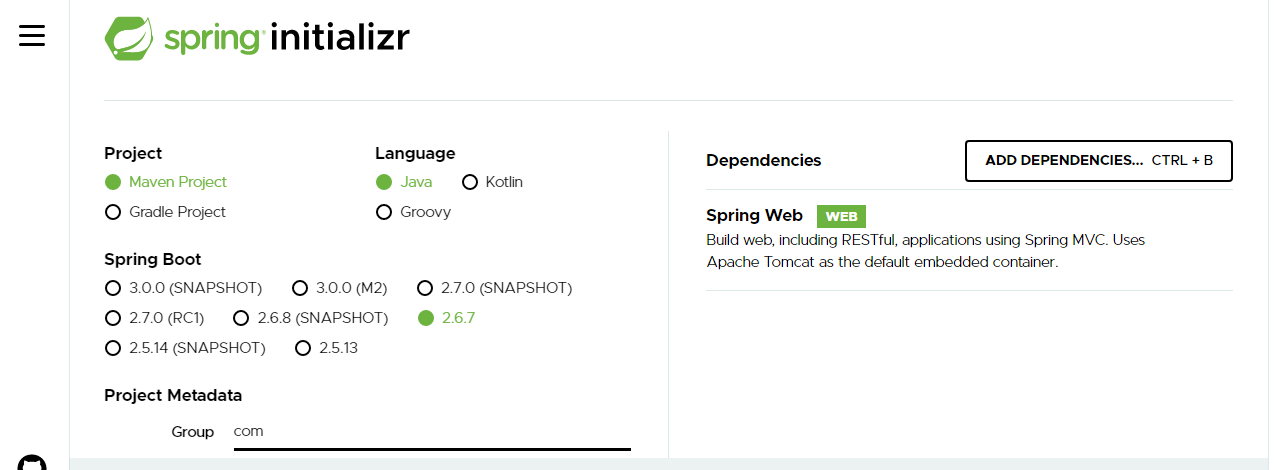
@Configuration: allow to register extra beans in the context or import additional configuration classes

The @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration, and @ComponentScan with their default attributes,

<https://start.spring.io/>

In this site:

version 2.6.7



Download and extract <filename>.jar

IDE 🡪 Eclipse 🡪 menu 🡪 file🡪 import 🡪 maven project 🡪 existing maven project .

Application.properties file

Server.servlet.context-path-/springbootcontext

Spring Boot + Spring Data JPA (CrudRepository) 🡪 custom 🡪 @Query , @Query(nativeQuery = true)

ONE to ONE

**@ONE to MANY**

DEPT 🡪 Employees (has – a / DI – dependency Injection)

If Dept has a reference of an employee 🡪 **Unidirection**

If both has a reference of other 🡪 **Bidirection**

**Department :**

**@ID**

**deptID;**

**deptName;**

**Employee var ; //one dept one employee**

**List<Employee> EmployeeList = new ArrayList<Employee>() // one to many – //collection – (list,set,Map)**

**Employee [] var //one to many in arrays**

**Employee:**

**@ID**

**empId;**

**empName;**

**salary;**

**Department var;**

# EXCEPTION HANDLING :

Try catch - in java.

Try(){

}catch( ){

}

Exception Handlers :

Generally , we develop Spring Boot app with REST (end point/payload)

RESTful service - Representation of HTTP Method

@PostMapping @RequestBody create/insert

@GetMapping @PathVariable read/get /{} URI

@RequestParam ?key=value&key2 =value2

@PutMapping @RequestBody update

@DeleteMapping @PathVariable Delete

{

“key” : ”value”,

“key” : “value”

}

We do have Spring Boot with Micro Services.

Exception Handling

* Controller level

@ExceptionHandler ( value= <which\_runtime\_exception>.class)

* Global level (application level)

@ControllerAdvice

public class EmployeeExceptionController {

@ExceptionHandler(value = EmployeeNotfoundException.class)

public ResponseEntity<Object> exception(EmployeeNotfoundException exception) {

return new ResponseEntity<>("Employee not found", HttpStatus.NOT\_FOUND);

}

}

Spring Boot - Exception Handling

--------------------------------

1) The @ControllerAdvice is an annotation, to handle the exceptions globally.

2) @ExceptionHandler is an annotation used to handle the specific exceptions and sending the custom responses to the client.

Demo

Step 1 :

package com.exception;

public class EmployeeNotfoundException extends RuntimeException {

private static final long serialVersionUID = 1L;

}

Step 2 :

@RequestMapping(value = "/products/{id}")

public ResponseEntity<Object> updateProduct(@PathVariable("id") int val ) {

if(val == 0)throw new EmployeeNotfoundException();

}

Step 3 :

@ControllerAdvice

public class EmployeeExceptionController {

@ExceptionHandler(value = EmployeeNotfoundException.class)

public ResponseEntity<Object> exception(EmployeeNotfoundException exception) {

return new ResponseEntity<>("Employee not found", HttpStatus.NOT\_FOUND);

}

}

To send as response by the ControllerAdvice class.

---------------------------------------------

Step 1 :

public class ExceptionResponse {

private String errorMessage;

private String requestedURI;

Step 2 :

@ControllerAdvice

public class ExceptionHandlerControllerAdvice {

@ExceptionHandler(ResourceNotFoundException.class)

@ResponseStatus(value = HttpStatus.NOT\_FOUND)

public @ResponseBody ExceptionResponse handleResourceNotFound(final ResourceNotFoundException exception,

final HttpServletRequest request) {

ExceptionResponse error = new ExceptionResponse();

error.setErrorMessage(exception.getMessage());

error.callerURL(request.getRequestURI());

return error;

}

@ExceptionHandler(Exception.class)

@ResponseStatus(value = HttpStatus.INTERNAL\_SERVER\_ERROR)

public @ResponseBody ExceptionResponse handleException(final Exception exception,

final HttpServletRequest request) {

ExceptionResponse error = new ExceptionResponse();

error.setErrorMessage(exception.getMessage());

error.callerURL(request.getRequestURI());

return error;

}

}