**CORE JAVA**

Java is a programming language and a platform. Java is a high level, robust, object-oriented and secure programming language.

**JVM**

It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

**JRE**

JRE is an acronym for Java Runtime Environment. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment.

**JDK**

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications.

**List of Java Keywords**

A list of Java keywords or reserved words are given below:

1. **abstract**: Java abstract keyword is used to declare an abstract class. An abstract class can provide the implementation of the interface. It can have abstract and non-abstract methods.

2. **boolean**: Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.

3. **break**: Java break keyword is used to break the loop or switch statement. It breaks the current flow of the program at specified conditions.

4. **byte**: Java byte keyword is used to declare a variable that can hold 8-bit data values.

5. **case:** Java case keyword is used with the switch statements to mark blocks of text.

6. **catch:** Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.

7. **char**: Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters

8. **class**: Java class keyword is used to declare a class.

9. **continue**: Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.

10. **default:** Java default keyword is used to specify the default block of code in a switch statement.

11. **do:** Java do keyword is used in the control statement to declare a loop. It can iterate a part of the program several times.

12. **double**: Java double keyword is used to declare a variable that can hold 64-bit floating-point number.

13. **else:** Java else keyword is used to indicate the alternative branches in an if statement.

14. **enum**: Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.

15. **extends**: Java extends keyword is used to indicate that a class is derived from another class or interface.

16. **final**: Java final keyword is used to indicate that a variable holds a constant value. It is used with a variable. It is used to restrict the user from updating the value of the variable.

17. **finally**: Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether an exception is handled or not.

18. **float**: Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.

19. **for**: Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some condition becomes true. If the number of iteration is fixed, it is recommended to use for loop.

20. **if**: Java if keyword tests the condition. It executes the if block if the condition is true.

21. **implements**: Java implements keyword is used to implement an interface.

22. **import**: Java import keyword makes classes and interfaces available and accessible to the current source code.

23. **int:** Java int keyword is used to declare a variable that can hold a 32-bit signed integer.

24. **interface**: Java interface keyword is used to declare an interface. It can have only abstract methods.

25. l**ong**: Java long keyword is used to declare a variable that can hold a 64-bit integer.

26. **new:** Java new keyword is used to create new objects.

27. **null:** Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.

28. **package:** Java package keyword is used to declare a Java package that includes the classes.

29. **private:** Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.

30. **protected:** Java protected keyword is an access modifier. It can be accessible within the package and outside the package but through inheritance only. It can't be applied with the class.

31. **public:** Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.

32. **return:** Java return keyword is used to return from a method when its execution is complete.

33. **short:** Java short keyword is used to declare a variable that can hold a 16-bit integer.

34. **static:** Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is mainly used for memory management.

35. **super**: Java super keyword is a reference variable that is used to refer to parent class objects. It can be used to invoke the immediate parent class method.

36. **switch:** The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.

37. **synchronized:** Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.

38. t**his**: Java this keyword can be used to refer the current object in a method or constructor.

39. **throw:** The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exceptions. It is followed by an instance.

40. **throws:** The Java throws keyword is used to declare an exception. Checked exceptions can be propagated with throws.

41. **try:** Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.

42. **void:** Java void keyword is used to specify that a method does not have a return value.

43. **while:** Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use the while loop.

**Variable:**

A variable is the name of a reserved area allocated in memory. In other words, it is a name of the memory location.

Ex: int data=50;//Here data is variable

**1) Local Variable**

A variable declared inside the body of the method is called local variable.

**2) Instance Variable**

A variable declared inside the class but outside the body of the method, is called an instance variable.

**3) Static variable**

A variable that is declared as static is called a static variable.

**Data Types in Java**

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

Primitive data types: The primitive data types include boolean, char, byte, short, int, long, float and double.

Non-primitive data types: The non-primitive data types include Classes, Interfaces, and Arrays.

**Methods in Java**

A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.

Syntax- public static int methodName(int a, intb){

//body

}

**Access Modifier**

Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier:

* **Public:** The method is accessible by all classes when we use public specifier in our application.
* **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
* **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
* **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

**Java OOPs Concepts**

The main aim of object-oriented programming is to implement real-world entities, for example, object, classes, abstraction, inheritance, polymorphism, etc.

**Object**

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

**Class**

Collection of objects is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object

**Inheritance**

When one object acquires all the properties and behaviours of a parent object, it is known as inheritance.

## Types of Inheritance

**Single Inheritance:**When a class inherits another class, it is known as a single inheritance.

## Multilevel Inheritance:When there is a chain of inheritance, it is known as multilevel inheritance.

## Hierarchical Inheritance:When two or more classes inherits a single class, it is known as hierarchical inheritance

**Multiple Inheritance:** When one class inherits multiple classes, it is known as multiple inheritance.

### **Hybrid Inheritance:** Hybrid means consist of more than one. Hybrid inheritance is the combination of two or more types of inheritance.

**Polymorphism**

If one task is performed in different ways, it is known as polymorphism.

In Java, we use method overloading and method overriding to achieve polymorphism.

# Method Overloading in Java

If class has multiple methods having same name but different in parameters, it is known as **Method Overloading.**

# Method Overriding in Java

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

**Abstraction**

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

### **Abstract class in Java**

A class which is declared as abstract is known as an **abstract class.** It can have abstract and non-abstract methods.

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is a mechanism to achieve *abstraction.* There can be only abstract methods in the Java interface, not method body.

**Encapsulation**

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

# **Association**-It is a relation between two separate classes which establishes through their Objects. Association can be one-to-one, one-to-many, many-to-one, many-to-many.

# **Aggregation -**If a class have an entity reference, it is known as Aggregation. Aggregation represents HAS-A relationship.

**Constructors in Java**

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created.

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type

## Types of Java constructors:

## Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

### Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

**Collections**

# The Java **collections** framework provides a set of interfaces and classes to implement various data structures and algorithms.

# **Java ArrayList**

Java **ArrayList** class uses a dynamic [array](https://www.javatpoint.com/array-in-java) for storing the elements. It is like an array, but there is no size limit. We can add or remove elements anytime.

The ArrayList in Java can have the duplicate elements also. It implements the List interface so we can use all the methods of the List interface here. The ArrayList maintains the insertion order internally.

# **Java LinkedList class**

Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure.

* Java LinkedList class can contain duplicate elements.
* Java LinkedList class maintains insertion order.
* Java LinkedList class is non synchronized.

# **Java HashSet**

Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

The important points about Java HashSet class are:

* HashSet stores the elements by using a mechanism called **hashing.**
* HashSet contains unique elements only.
* HashSet allows null value.
* HashSet class is non synchronized.

HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.

# **Java LinkedHashSet**

Java LinkedHashSet class is a Hashtable and Linked list implementation of the set interface. It inherits HashSet class and implements Set interface.

The important points about Java LinkedHashSet class are:

* Java LinkedHashSet class contains unique elements only like HashSet.
* Java LinkedHashSet class provides all optional set operation and permits null elements.
* Java LinkedHashSet class is non synchronized.
* Java LinkedHashSet class maintains insertion order.

# **Java TreeSet**

# Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements the NavigableSet interface. The objects of the TreeSet class are stored in ascending order.

The important points about Java TreeSet class are:

* Java TreeSet class contains unique elements only like HashSet.
* Java TreeSet class access and retrieval times are quiet fast.
* Java TreeSet class doesn't allow null element.
* Java TreeSet class is non synchronized.
* Java TreeSet class maintains ascending order.

# **Java Queue Interface**

Java Queue interface orders the element in FIFO(First In First Out) manner. In FIFO, first element is removed first and last element is removed at last.

# **Java Map Interface**

A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.

A Map is useful if you have to search, update or delete elements on the basis of a key.

# **Java HashMap**

Java **HashMap** class implements the Map interface which allows us to store key and value pair, where keys should be unique. If you try to insert the duplicate key, it will replace the element of the corresponding key. It is easy to perform operations using the key index like updation, deletion, etc. HashMap class is found in the java.util package.

HashMap in Java is like the legacy Hashtable class, but it is not synchronized. It allows us to store the null elements as well, but there should be only one null key

# **Java LinkedHashMap class**

Java LinkedHashMap class is Hashtable and Linked list implementation of the Map interface, with predictable iteration order. It inherits HashMap class and implements the Map interface.

### **Points to remember**

* Java LinkedHashMap contains values based on the key.
* Java LinkedHashMap contains unique elements.
* Java LinkedHashMap may have one null key and multiple null values.
* Java LinkedHashMap is non synchronized.
* Java LinkedHashMap maintains insertion order.

# **Java TreeMap class**

Java TreeMap class is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.

The important points about Java TreeMap class are:

* Java TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
* Java TreeMap contains only unique elements.
* Java TreeMap cannot have a null key but can have multiple null values.
* Java TreeMap is non synchronized.
* Java TreeMap maintains ascending order.

# **Java Vector**

**Vector** is like the dynamic array which can grow or shrink its size. Unlike array, we can store n-number of elements in it as there is no size limit. It is a part of Java Collection framework.

# **Java Stack**

The **stack** is a linear data structure that is used to store the collection of objects. It is based on **Last-In-First-Out** (LIFO). [Java collection](https://www.javatpoint.com/collections-in-java) framework provides many interfaces and classes to store the collection of objects. One of them is the **Stack class** that provides different operations such as push, pop, search, etc.

## Java Iterator

An Iterator is an object that can be used to loop through collections, like [ArrayList](https://www.w3schools.com/java/java_arraylist.asp) and [HashSet](https://www.w3schools.com/java/java_hashset.asp). It is called an "iterator" because "iterating" is the technical term for looping.

# **Exception Handling**

The **Exception Handling in Java** is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.

### **Types of Java Exceptions**

### 1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

### 2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

### 3) Error

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

## Java Threads

Threads allows a program to operate more efficiently by doing multiple things at the same time.Threads can be used to perform complicated tasks in the background without interrupting the main program.

There are two ways to create a thread.

It can be created by extending the Thread class and overriding its run() method.

## Running Threads

If the class extends the Thread class, the thread can be run by creating an instance of the class and call its start() method:

If the class implements the Runnable interface, the thread can be run by passing an instance of the class to a Thread object's constructor and then calling the thread's start() method.

**Arrays:array** is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location.

**Java String Methods**

Java String length(): The Java String length() method tells the length of the string. It returns count of total number of characters present in the String.

Java String compareTo(): The Java String compareTo() method compares the given string with current string

Java String concat() **:** The Java String concat() method combines a specific string at the end of another string and ultimately returns a combined string. It is like appending another string

Java String IsEmpty() : This method checks whether the String contains anything or not. If the java String is Empty, it returns true else false.

Java String toLowerCase() : The java string toLowerCase() method converts all the characters of the String to lower case.

Java String toUpper() : The Java String toUpperCase() method converts all the characters of the String to upper case.

Java String replace(): The Java String replace() method returns a string, replacing all the old characters or CharSequence to new characters. There are 2 ways to replace methods in a Java String.

Java String equals() : The Java String equals() method compares the two given strings on the basis of content of the string i.e Java String representation. If all the characters are matched, it returns true else it will return false

Java String equalsIgnoreCase(): This method compares two string on the basis of content but it does not check the case like equals() method. In this method, if the characters match, it returns true else false.

Java String toCharArray(): This method converts the string into a character array i.e first it will calculate the length of the given Java String including spaces and then create an array of char type with the same content.

**JAVA WEB DEVELOPMENT**

**Java EE/J2EE/Adv Java** :

* Java EE stands for Java Enterprise Edition
* Java EE is used to develop web applications and enterprise applications

**Components of Java EE** :

Web Componentsused to develop web applications

* Servlets
* JSP (Java Server Pages)

Enterprise Componentsused to develop enterprise applications (secured, distributed)

* EJB (Enterprise Java Beans)

Java EE applications are used by multiple clients and in order to make the Java EE application to be used by multiple clients we need to deploy(Copy) the applicationonto the server.

**Types of Servers** :

* Web Servers
* Application Servers

**Web Servers :**

Web servers are used to develop web applications only

* Apache Tomcat
* Jetty

**Servlets** :

Servlets are Java programs used to develop server side dynamic webapplications.

Technologies used to develop server side dynamic web applications are

* Servlets
* JSP

**Life cycle of a Servlet :**

Life cycle is a collection of methods which gets executed in an order automatically

by the servlet container (server).

init() :

* called only once when Servlet is loaded
* generally we establish database connections

service() :

* called for every request

destroy() :

* called only once when Servlet is destroyed
* generally we close database connections

**Creating a Servlet :**

By creating a class which extends GenericServletclass **(or)**

By creating a class which extends HttpServlet class

**Using GenericServletclass :**

public class WelcomeServlet extends GenericServlet{

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException {//handle the request }

}

**Using HttpServletclass :**

public class WelcomeServlet extends HttpServlet{

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException{//handle the reques }

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException{//handle the request }

}

**Note :**

interface HttpServletRequest extends ServletRequest {}

interface HttpServletResponse extends ServletResponse {}

**GenericServlet class vs HttpServletclass** :

GenericServlet class is used for different protocols like ftp, smtp, http etc.

where asHttpServlet class is used for only http protocol

ftp => file transfer protocol

smtp => simple mail transfer protocol

http => hypertext transfer protocol

**Sending HTML as response** :

* response.setContentType("text/html") method is used to send HTML as response

**Reading HTML Form data into a Servlet :**

* HTML Form is used to accept the user input
* request.getParameter() is used to read HTML form data into a Servlet
* The return type of request.getParameter() is String

**response.sendRedirect()** method is used to redirect to other Servlets/JSPs/HTML pageslike below example.

* response.sendRedirect("/WebProj/login.html");

**GET vs POST methods :**

* In case of GET method the data goes along with the url (visible in URL) where as in case of POST method the data goes as a separate page (not visible in URL)
* In case of GET method we can send only limited data (255 characters) where as in case of POST method we can send unlimited data
* Performance of GET method is more than POST method

**ServletConfiginterface :**

This interface is used to read initial parameters from web.xml file into a

particular Servlet.

**ServletContextinterface :**

This interface is used to read context parameters (global) from web.xml file

into all Servlets of web application.

**Session Tracking :**

Session means moving from one page to another pages

It is used to track whether the user is coming from previous pages properly or not

Ex:Page1 -> Page2 -> Page3

**4 ways of Session Tracking :**

* Hidden Form Fields
* URL Rewriting
* Cookies
* HttpSession

**Hidden Form Fields :**

In this we track the user session by using hidden form fields created in previous

pages and read the value of hidden form fields in the next pages.

**URL Rewriting :**

In this we track the user session by sending the data along with the url to the next

pages

**Cookies :**

* Cookies are used to remember the user data
* Cookies are used to store the data onto the client browse

**Creating Cookies :**

Syntax :Cookie cookieObj = new Cookie("cookie-name","cookie-value");

Ex:Cookie c1 = new Cookie("username","admin");

response.addCookie(c1);//adds the cookie onto the client browser

**Reading Cookies :**

Cookie[] cookies = request.getCookies();

for(Cookie c : cookies)

out.println(c.getName()+":"+c.getValue());

getName() => returns the cookie-name

getValue() => returns the cookie-value

**HttpSessioninterface** is used to track the user session by using attributes

Page1

HttpSession session = request.getSession();

session.setAttribute("attribute-name","attribute-value");

Page2

HttpSession session = request.getSession();

session.getAttribute("attribute-name");

**Filters :**

Filters are used for the following

* provide the preprocessing logic before the request is handled
* provide the postprocessing logic after the request is handle

**Listeners :**

Listeners are executed automatically based on the events like the following

* creating a ServletContext object
* creating a HttpSession object

**JSP (JAVA SERVER PAGES) :**

* JSP stands for Java Server Pages
* JSP is used to develop server side dynamic web applications like Servlets

**Servlets vs JSP :**

* In case of Servlets, Java code contains HTML code where as in case of JSP, HTML code contains Java code
* In case of Servlets, if the code is modified we need to recompile and redeploy the application where as in case of JSP, if the code is modified we need not recompile and redeploy the application
* JSP code is less than Servlet as in JSP we can use JSP Tags, JSP Implicit Objects and JSTL

**JSP Architecture :**

***Two Phases***

* **Translation Phase**
* When a request is made to a JSP page, the JSP page is translated into a Servlet only once until JSP page changes
* **Request Process Phase**
* the request is handled by the translated servlet

**JSP Life Cycle :**

jspInit():

* called only once when JSP page is loaded
* generally we establish Database connections

\_jspService():

* called for every request
* cannot be overridden

jspDestroy():

* called only once when JSP page is destroyed
* generally we close database connections

**JSP Tags :**

* *Directive Tags :*
* page directive tag =><%@page .... %>
* include directive tag =><%@include .... %>
* taglib directive tag =><%@taglib .... %>
* *ScriptletTags :*
* Declaration tag =><%! ... %>
* Expression tag =><%= ... %>
* Script tag =><% ... %>
* Action Tags
* <jsp:include>
* <jsp:forward>
* <jsp:param>
* <jsp:useBean>
* <jsp:setProperty>
* <jsp:getProperty>

**Custom Tags** used to create user defined tags in JSP

**page directive tag :**

<%@page ... %>

* used to import the Java packages

Ex:<%@page import = "java.util.\*" %>

<%@page import = "java.sql.\*" %>

or

<%@page import = "java.util.\*,java.sql.\*" %>

* used to set the content type

Ex:<%@page contentType="text/html" %>//response.setContentType("text/html");

<%@page contentType="application/vnd.ms-excel" %>

* used to handle the exceptions

**include directive tag :**

<%@include ... %>

* used to include other JSP pages

Ex:<%@include file="header.jsp" %>

...

*<%@include file="footer.jsp" %>*

**Declaration Tag**

<%! ... %>

* used to declare variables and to define methodsthe code will go into the translated servlet class as data members and methods

Ex:<%! int n = 100; %>

<%! void xxx() {... }%>

<%! void jspInit() { ... }%>

**Expression Tag :**

<%= ... %>

* used to access the value of the variable and to call methods
* the code goes into \_jspService() of translated servlet class

Ex:<%= n %> //100 will get displayed on client browser

<%= xxx() %>

<%= new java.util.Date() %>

**Script Tag :**

<% ... %>

* used to include the Java Statements
* the code goes into \_jspService() of translated servlet class

Ex: <%for(int i=1;i<=10;i++)

out.println(i);%>

**Note**

* HttpJspBase extends HttpServlet
* JspWriter extends PrintWriter

**config implicit object :**

config is an implicit object of ServletConfig interface which is used to read

initial parameters from web.xml file into a particular JSP file

**application implicit object :**

application is an implicit object of ServletContext interface which is used to

read context parameters (global) from web.xml file into all JSP pages of web application

**session implicit object:**

Session Tracking is used to track whether the user is coming from previous pages

properly or not

Ex: Page1 -> Page2 -> Page3

Page1 -> Page2 -> Page4

session is an implicit object of HttpSession interface which is used to track the

user session by using attributes

**pageContext implicit object :**

pageContext is an implicit object PageContext class and it is used to set, get and

remove attributes from the following scopes

* page
* request
* session
* application

**Note:** In JSP, page scope is the default scope

**Handling Exceptions in JSP :**

In Javatry{ statements;}

catch(Exception e){ //handle}

In case of JSP, if an exception occurs in a JSP page we can redirect to another JSP pageto handle the exception

Ex:<%@page errorPage="ShowErrors.jsp" %>

exception is an implicit object of Throwable class is used to handle exceptions in JSP

**JSP Implicit Objects :**

* out
* request
* response
* config
* application
* session
* page
* pageContext
* exception

**JSP Action Tags :**

* <jsp:include>
* <jsp:forward>
* <jsp:param>
* <jsp:useBean>
* <jsp:setProperty>
* <jsp:getProperty>

**<jsp:include> and <jsp:forward> :** These tags are used to include other JSP files

In case of <jsp:include> the response is handled by all the JSP pages where as

In case of <jsp:forward> the response is handled by only the last page

**<%@include ... %> directive vs <jsp:include> tag :**

In case of <%@include ...%> directive the content will get placed during translation phase where as in case of <jsp:include> tag the content will get placed during request process phase

<%@include ... %> directive is static where as<jsp:include> tag is dynamic

**<jsp:param> :**

It is used to send the parameters from one JSP page to another JSP page

**Using JavaBeans in JSP :**

A JavaBean is a public class with private properties and public setters and getter methodsfor every property

Ex:public class Book{

private int bno;

public void setBno(int bno) {

this.bno = bno;

}

public int getBno() {

return bno;

}

**JSTL :**

* JSTL stands for JSP Standard Template Library
* JSTL is created by Apache Company using Custom Tags of JSP
* In order to use the JSTL tags we need to copy "jstl1.2.jar" into WEB-INF/lib folder

### **Junit& Mockito**

### **JUnitisatestingframeworkforJava**

Use: Itisasimpleframeworktowriterepeatabletests

* AtestcaseisaprogramwritteninJava
* JUnit is linked as a JAR at compile-time
* Integrate JUnit in your project (with Maven)

**WithMaven**

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

...

</dependencies>

* WithoutMaven:add **junit.jar**onyourclasspath

**Junit:key concepts**

* JUnitisbasedon **Javaannotations**
* Javaannotationsareaformof metadata,providedataaboutaprogramthatisnotpartoftheprogramitself.
* Javaannotationshaveseveraluses:
  + Informationforthecompiler
  + Compile-timeanddeployment-timeprocessing
  + Runtimeprocessing

**Junit most used annotations**

* @org.junit.Test
* @org.junit.BeforeClass
* @org.junit.Before
* @org.junit.AfterClass
* @org.junit.After

Importorg.junit.\*;

publicclasstestClass1{

@BeforeClass

publicstaticvoidsetUpClass()throwsException{

// Code executed before the first test method

}

@Before

publicvoidsetUp()throwsException{

// Code executed before each test

}

@AfterClass

publicstaticvoidtearDownClass()throwsException{

// Code executed after the last test method

}

@After

publicvoidtearDown()throwsException{

// Code executed after each test

}

}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

**Junitassertions**

JUnitprovides**assertionmethods**forallprimitivetypesandObjectsandarrays

* Inthesemethodsthe**expectedvalue**iscomparedwiththe

### actualvalue.

* Theparameterorderis:
  + Optional:astringthatisoutputonfailure
  + expectedvalue
  + actualvalue

**Junit assertions**

### importstaticorg.junit.Assert.\*;

assertEquals("failure-stringsnotequal","text",

### "text");

assertFalse("failure - should be false", false);assertSame("should be same", number, number);assertArrayEquals("failure - byte arrays not same",expected,actual);

**Mockito**

**Introduction to Mockito**

* **What is Mockito?**

**Mockito**isaJavaframeworkallowingthecreationof**mockobjects**in automatedunittests

* A**mockobject**isadummyimplementationforaninterfaceoraclassinwhichyoudefinetheoutputofcertainmethodcalls.

**Why Mocking**

* Some“real”objectsrequiredinUnittestsarereallycomplextoinstantiate and/orconfigure
* Sometimes,onlyinterfacesexist,implementationsarenotevencoded.
* Ifyou useMockitointestsyou typically:
  + Mockawayexternaldependenciesandinsertthemocksintothecode undertest
  + Executethecodeunder test
  + Validatethatthecodeexecutedcorrectly

**How to use Mockito**

* IntegrateMockitoinyourprojectwithMaven
  + WithMaven

<dependencies>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-all</artifactId>

<version>1.10.19</version>

<scope>test</scope>

</dependency>

...

</dependencies>

* + WithoutMaven:addMockitoJARsonyourclasspath

**Moking a class**

**Import static org.mockito.**Mockitoto.\*;**importstaticorg.junit.**Assert.\*;

**@Test**

**publicvoidtest1(){**

//create Mock

**MyClasstest=mock(MyClass.class);**

// define return value for method getUniqueId()

**when(test.getUniqueId()).thenReturn(43);**

//usemockintest....

**assertEquals(test.getUniqueId(),43);**

**}**

**Argument matchers:**Mockito verifiesargumentvaluesby usinganequals()method

* When flexibility is required then you should use **argumentmatchers**

//stubbing using anyInt() argument matcher

**when(mockedList.get(anyInt())).thenReturn("element");**

//verify using an argument matcher

**verify(mockedList).get(anyInt());**

* Otherargumentmatchers:anyString(),anyObject(),anyVararg(),…

**Note:**Ifyou areusingargumentmatchers,allargumentshavetobe provided by matchers

**Mockito:spy**

With Mockito you can **spy** a real class. When you use the spythentherealmethodsarecalled(unlessamethodwasstubbed)**List<String>list=newLinkedList<>();**

**List<String>spy=spy(list);**

//optionally, you can stub out some methods:

**when(spy.size()).thenReturn(100);**

//usingthespycalls\*real\*methods

**spy.add("one");**

**spy.add("two");**

//prints "one" - the first element of a list

**System.out.println(spy.get(0));**

//size()methodwasstubbed-100isprinted

**System.out.println(spy.size());**

//optionally, you can verify

**verify(spy).add("one");**

**verify(spy).add("two");**

**GITHUB**

GitHub is a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere. This tutorial teaches you GitHub essentials like repositories, branches, commits, and pull requests.

Git is a distributed revision control and source code management system with an emphasis on speed. Git was initially designed and developed by Linus Torvalds for Linux kernel development. Git is a free software distributed under the terms of the GNU General Public License version 2. This tutorial explains how to use Git for project version control in a distributed environment while working on web-based and non web-based applications development.

What are GitHub’s Features

1. Easy Project Management

GitHub is a place where project managers and developers come together to coordinate, track, and update their work so that projects are transparent and stay on schedule.

2. Increased Safety With Packages

Packages can be published privately, within the team, or publicly to the open-source community. The packages can be used or reused by downloading them from GitHub.

3. Effective Team Management

GitHub helps all the team members stay on the same page and organized. Moderation tools like Issue and Pull Request Locking help the team to focus on the code.

4. Improved Code Writing

Pull request help the organizations to review, develop, and propose new code. Team members can discuss any implementations and proposals through these before changing the source code.

5. Increased Code Safety

GitHub uses dedicated tools to identify and analyze vulnerabilities to the code that other tools tend to miss. Development teams everywhere work together to secure the software supply chain, from start to finish.

6. Easy Code Hosting

All the code and documentation are in one place. There are millions of repositories on GitHub, and each repository has its own tools to help you host and release code.

List of useful Github Commands

Github is a distributed version control system which helps to manage the repositories.

These are a list of few commands that you can use frequently on github(git bash) 

1.git help

Take help from github help section for different commands and other errors 

2.git config

To set the basic configurations on github like your name and email. 

3.git config –-global user.name “Ashish Madaan”

Sets configuration values for your user name on git. 

4.git config –-global user.email ashishmadaan6@gmail.com

Sets configuration values for your user email on git. 

5.git config –-global color.ui true

To see different colours on command line for different outputs. 

6.mkdir store

Create a directory if not created initially. 

7.cd store

To go inside the directory and work upon its contents. 

8.git init

To create a local git repository for us in our store folder.This will help to manage the git commands for that particular repository. 

9.git status

To see whats changed since last commit.It shows all the files that have been added and modified and ready to be commmitted and files which are untracked 

10.git add Readme.txt

To add a file Readme.txt to the staging area to track its changes. 

11.git commit -m “Created a Readme.txt”

To commit our changes(taking a snapshot) and providing a message to remember for future reference. 

12.git log

To check the history of commits for our reference.

Different ways to use add command: 

13.git add

To add a specific list of files to staging area. 

14.git add --all

To add all files of current directory to staging area. 

15.git add \*.txt

To add all text files of the current directory to staging area. 

16.git add docs/\*.txt

To add all text files of a particular directory(docs) to staging area. 

17.git add docs/

To add all files in a particular directory(docs) to staging area. 

18.git add “\*.txt”

To add text files of entire project to staging area.   
More Commands: 

19.git diff

To figure out what changes you made since last commit. 

20.git reset head license

To undo staging of the file that was added in the staging area. 

21.git checkout –license

To Blow away all changes since the last commit of the file. 

22.git commit -a -m “Readme.md”

To add any of our tracked files to staging area and commit them by providing a message to remember. 

23.git reset –soft HEAD^

To undo last commit and bring file to staging area. 

24.git reset –hard HEAD^

To undo last commit and remove file from the staging area as well(In case we went horribly wrong). 

25.git reset –hard HEAD^^

To undo last 2 commits and all changes. 

26.git remote add origin https://github.com/madaan123/MyAlgorithms.git

This commands make a bookmark which signifies that this particular remote refers to this URL.   
This remote will be used to pull any content from the directory and push our local content to the global server. 

27.git remote add <address>

To add new remotes to our local repository for a particular git address. 

28.git remove rm

To remove a remote from our local repository. 

29.git push -u origin master

To push all the contents of our local repository that belong to master branch to the server(Global repository). 

30.git clone https://github.com/madaan123/MyAlgorithms.git

To clone or make a local copy of the global repository in your system   
(git clone command downloads the repository and creates a remote named as origin which can be checked by command – git remote -v). 

31.git branch Testing

To create a new branch named as Testing. 

32.git branch

To see all the branches present and current branch that we are working on. 

33.git checkout Testing

To switch to branch Testing from master branch. 

34.ls

To see directories and files in the current directory. 

35.git merge Testing

To merge Testing branch with master branch. 

36.git branch -d Testing

To delete Testing branch. 

37.git checkout -b admin

To create a new branch admin and set it as current branch. 

38.git branch -r

To look at all the remote branches. 

39.git branch -D Testing

To forcefully delete a branch without making commmits. 

40.git tag

To see the list of available tags. 

41.git checkout v0.0.1

To set the current tag to v0.0.1. 

42.git tag -a v0.0.3 -m “version 0.0.3”

To create a new tag. 

43.git push –tags

To push the tags to remote repository. 

44.git fetch

To fetch down any changes from global repository to current repository 

45.git rebase

Three tasks are performed by git rebase 

Move all changes to master which are not in origin/master to a temporary area.

Run all origin master commits.

Run all commits in the temporary area on top of our master one at a time, so it avoids merge commits.

1) what is maven?

\*Maven is a project management and comprehension tool that provides developers a complete build lifecycle framework.

\*Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.

\*Maven can set-up the way to work as per standards in a very short time.

\*As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

\* It allows developers to create projects, dependency, and documentation using Project Object Model and plugins.

2) what is usage of Maven?

\*Maven can add all the dependencies required for the project automatically by reading pom file.

\*One can easily build their project to jar,war etc. as per their requirements using Maven.

\*Maven makes easy to start project in different environments and one doesn’t needs to handle the dependencies injection, builds, processing, etc.

\*Adding a new dependency is very easy. One has to just write the dependency code in pom file.

3) How to create a maven project?

\*From the File menu, select New > Project.

\*The New Project screen opens.

\*Expand Maven, select Maven Project, and click Next.

\*The New Maven project wizard opens.

\*Leave the default, Use default Workspace location box selected and click Next.

\*The Select an archetype page opens.

\*click Add Archetype and supply the following values:

\*Archetype Group Id: com.lightbend.lagom

\*Archetype Artifact Id: maven-archetype-lagom-java

\*Version: The Lagom version number. Be sure to use the current stable release.

\*Repository URL: Leave blank

\*Click OK.

\*The next page of the wizard opens, providing fields to identify the project and displaying the hello and stream properties from the archetype.

\*To identify your project, enter the following:

\*Group Id - Usually a reversed domain name, such as com.example.hello.

\*Artifact Id - Maven also uses this value as the name for the top-level project folder. You might want to use a value such as my-first-system.

\*Version - A version number for your project.

\*Package - By default, the same as the groupId.

\*Click Finish and the projects created by the archetype display in the Package Explorer.

\*Run the project:

\*Right-click the parent project folder.

\*Eclipse puts all of the Maven project folders at the same level, so be sure to select the correct one. For example, if you used my-first-system as the Maven artifact ID, right-click my-first-system.

\*Select Run as … > Maven Build.

\*In the Goals field, enter lagom:runAll.

\*Select the JRE tab and make sure it is pointing at a JRE associated with a JDK.

\*Click Run.

4) what commands in maven? and what is the purpose of maven?

\*Maven Commands and purpose:

1.mvn --version =Prints out the version of Maven you are running.

2.mvn clean = Clears the target directory into which Maven normally builds your project.

3.mvn package = Builds the project and packages the resulting JAR file into the target directory.

4.mvn package -Dmaven.test.skip=true -> Builds the project and packages the resulting JAR file into the target directory - without running the unit tests during the build.

5.mvn clean package = Clears the target directory and Builds the project and packages the resulting JAR file into the target directory.

6.mvn clean package -Dmaven.test.skip=true -> Clears the target directory and builds the project and packages the resulting JAR file into the target directory - without running the unit tests during the build.

7.mvn verify = Runs all integration tests found in the project.

8.mvn clean verify = Cleans the target directory, and runs all integration tests found in the project.

9.mvn install = Builds the project described by your Maven POM file and installs the resulting artifact (JAR) into your local Maven repository

10.mvn clean install -Dmaven.test.skip=true =Clears the target directory and builds the project described by your Maven POM file without running unit tests, and installs the resulting artifact (JAR) into your local Maven repository.

11.mvn dependency:copy-dependencies =Copies dependencies from remote Maven repositories to your local Maven repository.

12.mvn clean dependency:copy-dependencies = Cleans project and copies dependencies from remote Maven repositories to your local Maven repository.

13.mvn clean dependency:copy-dependencies package =Cleans project, copies dependencies from remote Maven repositories to your local Maven repository and packages your project.

14.mvn dependency:tree =Prints out the dependency tree for your project - based on the dependencies configured in the pom.xml file.

15.mvn dependency:tree -Dverbose =Prints out the dependency tree for your project - based on the dependencies configured in the pom.xml file. Includes repeated, transitive dependencies.

16.mvn dependency:tree -Dincludes=com.fasterxml.jackson.core -> Prints out the dependencies from your project which depend on the com.fasterxml.jackson.core artifact.

17.mvn dependency:tree -Dverbose -Dincludes=com.fasterxml.jackson.core -> Prints out the dependencies from your project which depend on the com.fasterxml.jackson.core artifact. Includes repeated, transitive dependencies.

18.mvn dependency:build-classpath = Prints out the classpath needed to run your project (application) based on the dependencies configured in the pom.xml file.

**Spring**

Spring is a lightweight framework.

Application Context:Spring ApplicationContext is where Spring holds instances of objects that it has identified to be managed

and distributed automatically. These are called beans.Using the Inversion of Control principle, Spring collects bean instances from our application and uses them at the appropriate time.

Inversion Of Control (IOC):

The main tasks performed by IoC container are:

to instantiate the application class

to configure the object

to assemble the dependencies between the objects

There are two types of IoC containers. They are:

BeanFactory-->org.springframework.beans.factory.BeanFactory(interface),The XmlBeanFactory is the implementation class for the BeanFactory interface.

Resource resource=new ClassPathResource("applicationContext.xml");

BeanFactory factory=new XmlBeanFactory(resource);

ApplicationContext-->org.springframework.context.ApplicationContext(interface),It adds some extra functionality than BeanFactory.

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

In Spring framework, IOC container is responsible to inject the dependency.

We provide metadata to the IOC container either by XML file or annotation.

Dependency Injection:

Dependency Injection (DI) is a design pattern that removes the dependency from the programming code

In such case we provide the information from the external source such as XML file.

It makes our code loosely coupled and easier for testing.

Spring framework provides two ways to inject dependency:

By Constructor-->We can inject the dependency by constructor.

The <constructor-arg> subelement of <bean> is used for constructor injection where we can pass value and type of parameter.

By Setter method-->The <property> subelement of <bean> is used for setter injection.setter injection is flexible than constructor injection.

Autowiring in Spring:

Autowiring feature of spring framework enables you to inject the object dependency implicitly.

It internally uses setter or constructor injection.

Spring (AOP)Aspect Oriented Programming:

AOP is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns.

1Aspect:This is a module which has a set of APIs providing cross-cutting requirements. For example, a logging module would be called AOP aspect for logging.

An application can have any number of aspects depending on the requirement.

2.Join point:This represents a point in your application where you can plug-in the AOP aspect. You can also say, it is the actual place in the application where an action will be taken using Spring

AOP framework.

3.Advice: This is the actual action to be taken either before or after the method execution. This is an actual piece of code that is invoked during the program execution by Spring AOP framework.

4.Pointcut: This is a set of one or more join points where an advice should be executed. You can specify pointcuts using expressions or patterns as we will see in our AOP examples.

5.Introduction: An introduction allows you to add new methods or attributes to the existing classes.

6. Target object : The object being advised by one or more aspects. This object will always be a proxied object, also referred to as the advised object.

7. Weaving : Weaving is the process of linking aspects with other application types or objects to create an advised object. This can be done at compile time, load time, or at runtime.

Types of Advice:

1. before : Run advice before the a method execution.

2. after: Run advice after the method execution, regardless of its outcome.

3. after-returning : Run advice after the a method execution only if method completes successfully.

4. after-throwing: Run advice after the a method execution only if method exits by throwing an exception.

5.around : Run advice before and after the advised method is invoked.

JdbcTemplate Class:

The JDBC Template class executes SQL queries, updates statements, stores procedure calls, performs iteration over ResultSets, and extracts returned parameter values. It also catches JDBC exceptions and translates them to the generic, more informative, exception hierarchy defined in the org.springframework.dao package.

Transaction Management:

Atomicity − A transaction should be treated as a single unit of operation, which means either the entire sequence of operations is successful or unsuccessful.

Consistency − This represents the consistency of the referential integrity of the database, unique primary keys in tables, etc.

Isolation − There may be many transaction processing with the same data set at the same time. Each transaction should be isolated from others to prevent data corruption.

Durability − Once a transaction has completed, the results of this transaction have to be made permanent and cannot be erased from the database due to system failure.

Following are the possible values for isolation level

1.TransactionDefinition.ISOLATION\_DEFAULT : This is the default isolation level.

2. TransactionDefinition.ISOLATION\_READ\_COMMITTED : Indicates that dirty reads are prevented; non-repeatable reads and phantom reads can occur.

3. TransactionDefinition.ISOLATION\_READ\_UNCOMMITTED : Indicates that dirty reads, non-repeatable reads, and phantom reads can occur.

4. TransactionDefinition.ISOLATION\_REPEATABLE\_READ : Indicates that dirty reads and non-repeatable reads are prevented; phantom reads can occur.

5. TransactionDefinition.ISOLATION\_SERIALIZABLE : Indicates that dirty reads, non-repeatable reads, and phantom reads are prevented.

Spring mvc:

The Model encapsulates the application data and in general they will consist of POJO.

The View is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.

The Controller is responsible for processing user requests and building an appropriate model and passes it to the view for rendering.

The DispatcherServlet delegates the request to the controllers to execute the functionality specific to it.

The @Controller annotation indicates that a particular class serves the role of a controller.

The @RequestMapping annotation is used to map a URL to either an entire class or a particular handler method.

Annotation Based configuration:

1.@Required : The @Required annotation applies to bean property setter methods.

2. @Autowired : The @Autowired annotation can apply to bean property setter methods, non-setter methods, constructor and properties.

3. @Qualifier : The @Qualifier annotation along with @Autowired can be used to remove the confusion by specifiying which exact bean will be wired.

Bean Scope:

1. singleton : This scopes the bean definition to a single instance per Spring IoC container (default).

2. prototype : This scopes a single bean definition to have any number of object instances.

3. request : This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.

4. session : This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

5. global-session : This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

Bean Properties and Description:

1. class: This attribute is mandatory and specifies the bean class to be used to create the bean.

2. name : This attribute specifies the bean identifier uniquely. In XMLbased configuration metadata, you use the id and/or name attributes to specify the bean identifier(s).

3. scope : This attribute specifies the scope of the objects created from a particular bean definition.

4. constructor-arg: This is used to inject the dependencies.

5. properties : This is used to inject the dependencies.

6. autowiring mode : This is used to inject the dependencies.

7. lazy-initialization mode : A lazy-initialized bean tells the IoC container to create a bean instance when it is first requested, rather than at the startup.

8. initialization method : A callback to be called just after all necessary properties on the bean have been set by the container.

9. destruction method : A callback to be used when the container containing the bean is destroyed.

**HIBERNATE**

Overall Overview:

1. Spring jdbc to jpa.

1. World before jpa.

2. World after jpa introduced.

2. Simple jdbc application and mitigate it to jpa.

3. Basics of Spring, Spring Boot, Junit.

Almost 65% Java Projects use Spring and Spring boot

1. A big chunk of rest API, web services and microservices are being built with spring on spring boot.

2. Maven to manage dependencies and Git as a version Control

Tools Necessary

i) Eclipse -IDE

ii) Maven - dependency management

Earlier Approaches: JDBC on Spring JDBC

a)Lot of queries

1)create a query

2)Populate the right values into query.

3)fire it into database.

Problem arised: Queries become complex as application gets more complex and it is difficult to maintain.

Solution: JPA (JAVA PERSISTENCE API) - JPA creates it queries.

When the need arises to define your queries in jpa : you have three options

1. Jpql

2. Criteria query

3. Native Query

This API requires

1. What are the java classes or objects that you are creating - define your entities

2. How do they map to your tables - map fields to the columns in database.

3. Relationship among objects

4. JPA creates a query for you.

Entities - Object that we store in database

Spring Boot in 10 steps:

1)Enabling building production ready applications faster

2)Provides common non-functional features

-embedded servers

-metrics

-health checks

-externalized configuration

Spring Boot

1)does not generate any code

2)It i neither an application server nor a web server

Features of spring boot

1)Quick Starter projects with Auto-configuration

spring-boot-starter-web (consist spring-mvc, spring-core, validation)

spring-boot-starter-jpa (default implementation of jpa, hibernate and auto-configuration)

Using spring Initializer to create a Spring Boot Application

To signify something as a control use

Creating a simple Rest Controller

@RestController

\* Serves request

@GetMapping(“/url”)

\* To retrieve a list of values

Spring Boot AutoConfiguration

\* Spring context

\* Auto configuration

\* Component scan

Spring JDBC to JPA

Required dependencies - h2, jdbc, web, jpa

In application.properties set the below configuration:

spring.database.url = jdbc:h2:mem:testdb

spring.data.jpa.repositories.bootstrap-mode=default

spring.h2.console.enabled = true // to enable h2 console

spring.jpa.show-sql = true // to see the queries being executed

@Entity - to signify it is an entity

@Table(name=”tablename”) - table name to be referred in database

@Column(name=”columnname”) - column name to be referred in database

@id - Primary key

@GeneratedValue - to be automatically incremented

@PersistenceContext

All the operations that are performed on a specific or all entities, are stored in PersistenceContext

@EntityManager

EntityManager is the interface to PersistenceContext

Entity manager has many methods provided such as find, merge, persist, remove.

JPA AND HIBERNATE

1)Create entities

Make sure to have a default constructor

2) create a repository ( a method or action to be performed. Ex - findById)

3)create a sql file

4)When application would be launched( it implements a CommandLineRunner and run method would be launched up and tries to find the specified requirement)

5)Create a junit test for that method

@RunWith(SpringRunner.class) // launches a spring context in my Unit test

The spring context to launch is @SpringBootTest

@DirtiesContext - to leave the state of application in its original form.

Hibernate - sequence : it is used to insert the values by default

EntityManager methods

1. Flush - changes done until then will be sent out to database.

2. detach(obj) - changes of that particular object are no longer tracked by Entity manager

3. Clear - used to clear all objects that are managed by entity manager

4. Refresh - refresh of entity manager

5. Persist - insert

6. Merge - update

7. Remove - delete

8. Find - to find

9. createQuery - to create a query

JPQL - Java Persistence Query Language

We query from entities and we write our queries in the form jpql which is converted into sql queries by JPA implementation( i.e., Hibernate)

Constraints on Columns

nullable,unique,insertable,updatable,length,precision,scale

Annotations

@UpdateTimestamp - updates time

@CreationTimestamp - creates time

@NamedQuery(name=”query\_name”,query = “select var from Entity var)

@NamedQueries(value={

@NamedQuery(),

@NamedQuery()

})

Use createNamedQuery(“query\_name”) - to call the specified query

Native queries - to use native sql queries in jpa

createNativeQuery(“select \* from course where id=?”,Course.class)

\* You can have positional or named parameters

Establishing Relationship with jpa and hibernate

a)one to one - @OnetoOne

1)One to one relationship is always Eager Fetch

Eager fetch - retrieves the details of both entities

To change the fetch use - @OnetoOne(fetch=FetchType.LAZY) -

It retrieves data only from the specified one.

Ex: each student have a unique passport

2)Make sure to declare a owning side of relationship :

Add this on non-owning side of entity @OnetoOne(mappedby=”owningentityname”)

@Transactional - to make a transaction a complete success or nothing succed.

If any operation fails, all operations done must be rolled back

As soon as you create a transaction, you would create a persistence context( a place where it holds all the entities that are being operated on)

A way to interact with persistence context is by using entity manager.

2)OnetoMany -

Default fetching - lazy

ex: course can have many reviews or no reviews

3)ManytoOne -

Default fetching is eager

4)ManytoMany

Default fetching is lazy

Ex: course can have multiple students enrolled and a student can enroll for multiple courses

@JoinTable(name=”jointablename”,

joinColumns=@JoinColumn(name=”owning\_entityname”),

inverseJoinColumns=@JoinColum(name=”nonowning\_entityname”))

Queries using JAVA API - Criteria Queries

1.Use Criteria Builder to create a criteria Query returning the expected result object

2.Define roots for tables which are invloved in the query

3.Define Predicates etc using Criteria Builder

4.Add predicates etc to the Criteria Query

5.Build the TypedOuery using the entity manager and criteria query

TRANSACTION MANAGEMENT

It is an important concept while making changes or updates to database.

Transaction management make sure that when a failure occurs, to roll back to its previous state.

Spring uses annotation - @Transctional

ACID PROPERTIES

1)Atomicity - Transaction must be completely successful or all the changes that are done by transaction must be reverted back when it fails.

2)Consistency - Transaction must leave the system in consistent state

3)Isolation - How changes within a transaction must be visible to other transactions

4)Durability - Once a transaction is successful, even when the system fails or crashes, the end state of transaction must be persistence.

IMPORTANT CONCEPTS RELATED TO ISOLATION LEVEL

1)Dirty Read - Two transactions are running in parallel, another transaction reading the modified value before the given transaction is committed.

2)Non-repeatable Read - When i am reading the same thing twice, i'm getting two different values.

3)Phanthom Read - at different times, i'm getting different number of rows for the same transaction.

Four Isolation Levels

1)Read uncommitted - no restriction on data (no locks at all)

problems faced - dirty read, non-repeatable read, phanthom read

2)Readcommitted - read only the transactions of data which is committed by another transaction.(lock is set until the transaction is committed)

problems solved - dirty read

3)Repeatable read - locks both modified as well as read values during the transaction.

problems solved - dirty read, non-repeatable read

4)Serializable - table lock on constraint : cant update, delete or insert until that transaction has been completed.

problems solved - dirty read, non-repeatable read, phanthom read

THREE THINGS TO DECIDE WHILE DEALING WITH TRANSACTION MANAGEMENT

1)Spring or jpa transaction management

2)decide the isolation level

3)default isolation level by configuring the settings in spring.jpa.hibernate.connection.isolation = 2;

values

1 - Read uncommitted

2 - Readcommitted

4 - Repeatable read

8 - Serializable

Spring Data JPA and Spring Data REST

earlier problems - lot of duplication amon repositories, proliferation of data source

so spring provides simple abstraction being able to access any kind of data.

Spring data JPA is the JPA specific implementation of spring data

Spring data Jpa repository returns an Optional tag

Optional tag provides to check if entity object exist.

CRUD Repository is one of the fundamental Spring Data Repositories

Basic Crud operations

methods supported

1)findById()

2)existsById()

3)findAll()

4)findByAllId(Iterable)

5)count()

6)deleteById(ID)

7)deleteAll()

8)save(s) -- > either to create or update

Paging and sorting repository provides facilities methods to sort based on specific field, add

pagination to the results that are delivered by the crud repository

JPA Repository has specific methods

1)flush()

2)saveAndFlush()

3)deleteInBatch()

**MICROSERVICE ARCHITECTURE**

**What is Micro service architecture?**

Micro services is an architectural design for building a distributed application using containers. They get their name because each function of the application operates as an independent service. This architecture allows for each service to scale or update without disrupting other services in the application.

**Why do we need Microservices?**

Microservices are increasingly used in the development world as developers work to create larger, more complex applications that are better developed and managed as a combination of smaller services that work cohesively together for larger, application-wide functionality. Microservices enable IT organizations to be more agile. And build more scalable applications. Microservices translate into faster development and change cycles. As Figure 2 suggests, breaking down a large application into smaller services makes the development workflow faster

**What is monolithic architecture?**

Microservices architecture. While a monolithic application is a single unified unit, a microservices architecture breaks it down into a collection of smaller independent units. These units carry out every application process as a separate service.

**What is SOA (Service-Oriented Architecture)?**

Service-Oriented Architecture (SOA) is a stage in the evolution of application development and/or integration. It defines a way to make software components reusable using the interfaces.

Formally, SOA is an architectural approach in which applications make use of services available in the network. In this architecture, services are provided to form applications, through a network call over the internet. It uses common communication standards to speed up and streamline the service integrations in applications. Each service in SOA is a complete business function in itself. The services are published in such a way that it makes it easy for the developers to assemble their apps using those services. Note that SOA is different from microservice architecture.

* SOA allows users to combine a large number of facilities from existing services to form applications.
* SOA encompasses a set of design principles that structure system development and provide means for integrating components into a coherent and decentralized system.
* SOA-based computing packages functionalities into a set of interoperable services, which can be integrated into different software systems belonging to separate business domains.

**What are the problems with monolith and SOA?**

1. Single Technology
2. Inflexible Deployment
3. Inefficient compute resources
4. Large and complex
5. Complicated and Expensive ESB
6. Lack of Tooling

**Characteristics of a Microservice Architecture**

1. Componentization via Services. ...
2. Organized around Business Capabilities. ...
3. Products not Projects. ...
4. Smart endpoints and dumb pipes. ...
5. Decentralized Governance. ...
6. Decentralized Data Management. ...
7. Infrastructure Automation. ...
8. Design for failure

**Defining Communication Patterns?**

1. 1-to-1 Synchronous
2. 1-to-1 Asynchronous
3. Pub-Sub/Event Driven

**What is synchronous in microservices?**

A synchronous microservice is one in which data moves to and from an endpoint of a service in a blocking interaction. A typical example of a synchronous data exchange is an HTTP request/response interaction, seen in Figure 1 below. When a request is made to an endpoint under HTTP, the caller is locked in the interaction until a response is received. The caller might receive the response in a mere millisecond or in a few seconds. Regardless of the application latency, the caller cannot move forward to the next task until the response is received. REST is a good example of a synchronous microservice.

**What is Asynchronous in microservices?**

An asynchronous microservice is one in which a request to a service and the subsequent response occur independently from each other. The general practice for implementing an asynchronous microservice is to use a message broker technology, such as Kafka or RabbitMQ, to act as a go-between for services, as seen in Figure 2 below. One service will publish a message to another service using the message broker.The intended service receives the message in its own time. The sending service is not locked to the broker. It simply fires and forgets.

**Testing Microservice?**

1. Unit Tests
2. Integrating Tests
3. End to End Tests

**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. This testing methodology is done during the development process by the software developers and sometimes QA staff.

**Integrating Testing**

Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group. Integration testing is conducted to evaluate the compliance of a system or component with specified functional requirements.

**End to End Testing**

End-to-end testing is a methodology used in the software development lifecycle (SDLC) to test the functionality and performance of an application under product-like circumstances and data to replicate live settings. The goal is to simulate what a real user scenario looks like from start to finish.

**What Is Service Mesh Architecture?**

A service mesh provides a consistent, decentralized mechanism for managing communication between multiple services within a system. It can be used to implement features such as encryption, logging, tracing and load balancing, thereby improving security, reliability and observability.

Service mesh examples:-

Incoming traffic (called ingress), outgoing traffic (called egress), and traffic between services (mesh traffic). Each microservice instance (container or VM) has a dedicated Envoy proxy. The control plane is the management layer for the Envoy proxies.

**What is Logging and Monitoring?**

We will discuss one of the most important aspects of the Microservices system, which is the logging and monitoring. Without proper logging and monitoring, we are going to have a lot of problems with our system. It might lead to its failure too.

As I said earlier, this is extremely important in Microservices projects even more than monolith. The reason for that is that in Microservices, the flow goes through multiple processes. In this case it’s difficult to get holistic view of the system or entire flow. Most of the time in traditional monolith, we can examine logs and see what went wrong or what is happening. In this case with Microservices, it is difficult to stitch things together as it involve many processes.

We can look at a specific service and understand if there are any problems with it. It’s very difficult to figure out what is going on with all the services working together. All these problems are handled by well-designed logging and monitoring.

**Logging**

1. Recording the system’s activity. We document what the system did and what the users did and how did the system behave etc.. This is useful for analysing the system’s behaviour and making sure everything is good.
2. Auditing -We can see user’sbehaviour and see what they did.
3. Documenting errors -Logging is the best way to write everything that has to do with errors happening in the system.

**Monitoring**

1. Based on system’s metrics -Monitoring tools look at the various metrics of the system, from infrastructure related metrics such as CPU, RAM, and Disk etc...To application related metrics such as requests per minute, orders per day etc... These metrics then available for users via sophisticated dashboards.
2. Alerting -We can define alerts that will be raised when a specific metric goes outside the normal range. For example, when CUP goes high, and Memory usage is high, we can configure monitoring tools to send alerts to a predefined group of people, so they can handle.

As you can see both should implemented in order to make sure our Microservices system is reliable and stable.

**Implementing Logging**

As I said, logging should provide holistic view on the system. It should allow tracing end to end flow. End to end means from the service where the flow originated to the service that ended and with all the services in between included. Logging should contain as much information as possible. So make sure log whatever information you can think of, even things that are first look might seem useless.

For example we definitely want to look at least the timestamp, severity model, message, machine id and may be ip address etc. Then you should be able to filter based on these values.

Logging in Microservices architecture should work quite differently from the traditional logging.

In a traditional logging, if we have, let’s say two services, then each one of them has its own logging infra. The service is using its own logging library and it logs to service’s own repository, may be in to file or database. This is easiest way and when our application is composed of a single process, then there is no problem with such implementation.

However, with Microservices, we have some problems implementing the logging like that. As we want to look at the logs and trace an end to end flow, that would be a problem. Also using different logging libraries means we might utilize different logging formats. For example, we may have JSON in our file and plain text in another file. In addition, we cannot aggregate easily to see how many error per day, how many DB access per day etc.

**Implementing Monitoring**

Monitoring looks at metrics and detects anomalies. It also provides simplified view of the system status usually using dashboards that presents the important metrics in an easy to grasp format.

In addition, we can set alerts to trigger when there is a problem and relevant people can notify. There are two types of monitoring,

Here we monitors the underlines servers. This means, we usually monitor things such as CPU, RAM, Disk, Network and any other metrics that are infrastructure related. Then we can get alerts when a problem is detected in the infrastructure such as high CPU or high disk IO etc.

**When should you not use a microservice?**

1. Loose coupling.
2. Surgically scalable.
3. Enhanced development productivity.
4. Alignment with domain-driven design (DDD)
5. Extensibility.
6. More agile enterprise.

**JENKINS**

**Jenkins** – an open source automation server which enables developers around the world to reliably build, test, and deploy their software.

Why Jenkins?

Jenkins is a software that allows **continuous integration**. Jenkins will be installed on a server where the central build will take place. The following flowchart demonstrates a very simple workflow of how Jenkins works.

Along with Jenkins, sometimes, one might also see the association of **Hudson**. Hudson is a very popular open-source Java-based continuous integration tool developed by Sun Microsystems which was later acquired by Oracle. After the acquisition of Sun by Oracle, a fork was created from the Hudson source code, which brought about the introduction of Jenkins.

What is Continuous Integration?

Continuous Integration is a development practice that requires developers to integrate code into a shared repository at regular intervals. This concept was meant to remove the problem of finding later occurrence of issues in the build lifecycle. Continuous integration requires the developers to have frequent builds. The common practice is that whenever a code commit occurs, a build should be triggered.

Starting Jenkins

Open the command prompt. From the command prompt, browse to the directory where the jenkins.war file is present. Run the following command

D:\>Java –jar Jenkins.war

After the command is run, various tasks will run, one of which is the extraction of the war file which is done by an embedded webserver called winstone.

D:\>Java –jar Jenkins.war

Running from: D:\jenkins.war

Webroot: $user.home/ .jenkins

Sep 29, 2015 4:10:46 PM winstone.LoggerlogInternal

INFO: Beginning extraction from war file

Once the processing is complete without major errors, the following line will come in the output of the command prompt.

INFO: Jenkins is fully up and running

Accessing Jenkins

Once Jenkins is up and running, one can access Jenkins from the link − **http://localhost:8080**

This link will bring up the Jenkins dashboard.

For this exercise, we will create a job in Jenkins which picks up a simple HelloWorld application, builds and runs the java program.

**Step 1** − Go to the Jenkins dashboard and Click on New Item

**Step 2** − In the next screen, enter the Item name, in this case we have named it Helloworld. Choose the ‘Freestyle project option’

**Step 3** − The following screen will come up in which you can specify the details of the job.

**Step 4** − We need to specify the location of files which need to be built. In this example, we will assume that a local git repository(E:\Program) has been setup which contains a ‘HelloWorld.java’ file. Hence scroll down and click on the Git option and enter the URL of the local git repository.

**Note** − If you repository if hosted on Github, you can also enter the url of that repository here. In addition to this, you would need to click on the Add button for the credentials to add a user name and password to the github repository so that the code can be picked up from the remote repository.

**Step 5** − Now go to the Build section and click on Add build step → Execute Windows batch command

**Step 6** − In the command window, enter the following commands and then click on the Save button.

Javac HelloWorld.java

Java HelloWorld

**Step 7** − Once saved, you can click on the Build Now option to see if you have successfully defined the job.

**Step 8** − Once the build is scheduled, it will run. The following Build history section shows that a build is in progress.

**Step 9** − Once the build is completed, a status of the build will show if the build was successful or not. In our case, the following build has been executed successfully. Click on the #1 in the Build history to bring up the details of the build.

**Step 10** − Click on the Console Output link to see the details of the build

Apart from the steps shown above there are just so many ways to create a build job, the options available are many, which what makes Jenkins such a fantastic continuous deployment tool.

Jenkins provides an out of box functionality for Junit, and provides a host of plugins for unit testing for other technologies, an example being MSTest for .Net Unit tests. If you go to the link <https://wiki.jenkins-ci.org/display/JENKINS/xUnit+Plugin> it will give the list of Unit Testing plugins available.

The following are some of the basic activities you will carry out, some of which are best practices for Jenkins server maintenance

URL Options

The following commands when appended to the Jenkins instance URL will carry out the relevant actions on the Jenkins instance.

**http://localhost:8080/jenkins/exit** − shutdown jenkins

**http://localhost:8080/jenkins/restart** − restart jenkins

**http://localhost:8080/jenkins/reload** − to reload the configuration

Backup Jenkins Home

The Jenkins Home directory is nothing but the location on your drive where Jenkins stores all information for the jobs, builds etc. The location of your home directory can be seen when you click on Manage Jenkins → Configure system.

Set up Jenkins on the partition that has the most free disk-space – Since Jenkins would be taking source code for the various jobs defined and doing continuous builds, always ensure that Jenkins is setup on a drive that has enough hard disk space. If you hard disk runs out of space, then all builds on the Jenkins instance will start failing.

Another best practice is to write cron jobs or maintenance tasks that can carry out clean-up operations to avoid the disk where Jenkins is setup from becoming full.

Jenkins provides good support for providing continuous deployment and delivery. If you look at the flow of any software development through deployment.

The main part of Continuous deployment is to ensure that the entire process which is shown above is automated. Jenkins achieves all of this via various plugins, one of them being the “Deploy to container Plugin” which was seen in the earlier lessons.

# **Jenkins - Management**

To manage Jenkins, click on the ‘Manage Jenkins’ option from the left hand menu side.

So one can get the various configuration options for Jenkins by clicking the ‘Manage Jenkins’ option from the left hand menu side.

Some of the management options are as follows −

## Configure System

This is where one can manage paths to the various tools to use in builds, such as the JDKs, the versions of Ant and Maven, as well as security options, email servers, and other system-wide configuration details. When plugins are installed. Jenkins will add the required configuration fields dynamically after the plugins are installed.

## Reload Configuration from Disk

Jenkins stores all its system and build job configuration details as XML files which is stored in the Jenkins home directory. Here also all of the build history is stored. If you are migrating build jobs from one Jenkins instance to another, or archiving old build jobs, you will need to add or remove the corresponding build job directories to Jenkins’s builds directory. You don’t need to take Jenkins offline to do this—you can simply use the “Reload Configuration from Disk” option to reload the Jenkins system and build job configurations directly.

## Manage Plugin

Here one can install a wide variety of third-party plugins right from different Source code management tools such as Git, Mercurial or ClearCase, to code quality and code coverage metrics reporting. Plugins can be installed, updated and removed through the Manage Plugins screen.

## System Information

This screen displays a list of all the current Java system properties and system environment variables. Here one can check exactly what version of Java Jenkins is running in, what user it is running under, and so forth.

The following screenshot shows some of the name-value information available in this section.

### **System Log**

The System Log screen is a convenient way to view the Jenkins log files in real time. Again, the main use of this screen is for troubleshooting.

### **Load Statistics**

This pages displays graphical data on how busy the Jenkins instance is in terms of the number of concurrent builds and the length of the build queue which gives an idea of how long your builds need to wait before being executed. These statistics can give a good idea of whether extra capacity or extra build nodes is required from an infrastructure perspective.

### **Script Console**

This screen lets you run Groovy scripts on the server. It is useful for advanced troubleshooting since it requires a strong knowledge of the internal Jenkins architecture.

### **Manage nodes**

Jenkins is capable of handling parallel and distributed builds. In this screen, you can configure how many builds you want. Jenkins runs simultaneously, and, if you are using distributed builds, set up build nodes. A build node is another machine that Jenkins can use to execute its builds.

### **Prepare for Shutdown**

If there is a need to shut down Jenkins, or the server Jenkins is running on, it is best not to do so when a build is being executed. To shut down Jenkins cleanly, you can use the Prepare for Shutdown link, which prevents any new builds from being started. Eventually, when all of the current builds have finished, one will be able to shut down Jenkins cleanly.

# **Jenkins - Metrics & Trends**

There are various plugins which are available in Jenkins to showcase metrics for builds which are carried out over a period of time. These metrics are useful to understand your builds and how frequently they fail/pass over time. As an example, let’s look at the ‘Build History Metrics plugin’.

This plugin calculates the following metrics for all of the builds once installed

* Mean Time To Failure (MTTF)
* Mean Time To Recovery (MTTR)
* Standard Deviation of Build Times