**JAVA DEFINATIONS**

**SDLC:** Software development life cycle

It is a process for project management which includes planning, design, development, testing, and deploying, maintenance an information system. We have used agile and waterfall methodologies.

>>**AGILE:**

Agile software development focuses on keeping code simple, testing often, and delivering functional bits of the application as soon as they're ready.

The goal of agile software development (ASD) is to build upon small client-approved parts as the project progresses, as opposed to delivering one large application at the end of the project.

We used jira --🡪 to-do, in-progress, code review, testing, fixed and closed.

**WATERFALL:**

The waterfall model is a sequential (non-iterative) design process, used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of conception, initiation, analysis, design, construction, testing, production/implementation and maintenance.

>>**TTD**: Test-driven development

TDD is an evolutionary approach to development where you write a test before you write just enough production code to fulfill that test and refactoring.

One view is the goal of TDD is it’s one way to think through your requirements or design before your write your functional code. Important for agile design.

=====================================================================================

**MVC**

**Model:** It will use the Components to get the job done. (We use Plain, Old java objects (pojos) for the model.)

**View:** It will create a response to send back to the client.

**Controller**: Controller will get the request and decide what needs to be accomplished.

All of the processed information is put together with HTML, CSS and Java script to create a response.

What is difference between **JDK, JRE and JVM**?

**JVM**

JVM is an acronym for Java Virtual Machine, it is an abstract machine which provides the runtime environment in which java bytecode can be executed. It is a specification.

JVMs are available for many hardware and software platforms (so JVM is platform dependent).

**JRE**

JRE stands for Java Runtime Environment. It is the implementation of JVM.

**JDK**

JDK is an acronym for Java Development Kit. It physically exists. It contains JRE + development tools.

 What is **Just-In-Time (JIT) compiler**?

It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here the term “compiler” refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

**OOPS**

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects.

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

**CLASS:** Collection of objects is called class. It is a logical entity.

A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.

**Methods** − A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.

#### **Inheritance:** When one object acquires all the properties and behaviors of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

**Polymorphism**

When one task is performed by different ways i.e. known as polymorphism. In java, we use method overloading and method overriding to achieve polymorphism. Example can be to speak something e.g. cat speaks meow, dog barks woof etc.

**Abstraction**

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing. In java, we use abstract class and interface to achieve abstraction.

**Encapsulation**

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

What is **constructor?**

* Constructor is just like a method that is used to initialize the state of an object. It is invoked at the time of object creation.
* constructor is not inherited, constructor can't be final.

What is **this** in java?

It is a keyword that that refers to the current object.

What is **Inheritance?**

Inheritance is a mechanism in which one object acquires all the properties and behavior of another object of another class. It represents IS-A relationship. It is used for Code Reusability and Method Overriding.

What is **method overloading?**

Having the same method names with different arguments.

What is **method overriding?**

If a subclass provides a specific implementation of a method that is already provided by its parent class, it is known as Method Overriding. It is used for runtime polymorphism and to provide the specific implementation of the method.

What is **Runtime Polymorphism?** is a process in which a call to an overridden method is resolved at runtime rather than at compile-time. In this process, an overridden method is called through the reference variable of a super class. The determination of the method to be called is based on the object being referred to by the reference variable.

What is **interface?**

Interface is a blueprint of a class that have static constants and abstract methods. It can be used to achieve fully abstraction and multiple inheritance.

What is **package?**

A package is a group of similar type of classes interfaces and sub-packages. It provides access protection and removes naming collision.

**EXCEPTION HANDLING/ERRORS**

What is **Exception Handling?**

Exception Handling is a mechanism to handle runtime errors. It is mainly used to handle checked exceptions.

**Checked Exception**

The classes that extend Throwable class except Runtime Exception and Error are known as checked exceptions e.g. IO Exception, SQL Exception etc. Checked exceptions are checked at compile-time.

**Unchecked Exception**

The classes that extend Runtime Exception are known as unchecked exceptions e.g. Arithmetic Exception, Null Pointer Exception etc. Unchecked exceptions are not checked at compile-time.

**TRY CATCH BLOCK:**

The try block contains a block of program statements within which an exception might occur. A try block is always followed by a catch block, which handles the exception that occurs in associated try block. A try block must followed by a Catch block or Finally block or both.

What is **finally block?**

finally block is a block that is always executed.

**Throw vs Throws** in java

1. **Throws clause** in used to declare an exception and **thow** keyword is used to throw an exception explicitly.

2. If we see syntax wise than **throw** is followed by an instance variable and **throws** is followed by exception class names.

3. The keyword **throw** is used inside method body to invoke an exception and **throws clause** is used in method declaration (signature).

What is difference between **final, finally and finalize?**

|  |
| --- |
| **final:** final is a keyword, final can be variable, method or class. You, can't change the value of final variable, can't override final method, can't inherit final class. |
| **finally:** finally block is used in exception handling. finally block is always executed. |
| **finalize():** finalize() method is used in garbage collection. finalize() method is invoked just before the object is garbage collected. The finalize() method can be used to perform any cleanup processing.  ===================================================================================  What is **serialization?**  Serialization is a process of writing the state of an object into a byte stream. It is mainly used to travel object's state on the network.  What is **Deserialization?** Deserialization is the process of reconstructing the object from the serialized state. It is the reverse operation of serializatioangn.  What are **wrapper classes?**  Wrapper classes are classes that allow primitive types to be accessed as objects.  ============================================================================  **THREAD**  A thread is an independent path of execution within a program. Many threads can run concurrently within a program. Every thread in Java is created and controlled by the java.lang.Thread class  What is **multithreading**?  Multithreading is a process of executing multiple threads simultaneously. Its main advantage is:   * Threads share the same address space, Thread is lightweight. * Cost of communication between process is low.   How can we create a **Thread** in Java?  There are two ways to create Thread in Java – first by implementing Runnable interface and then creating a Thread object from it and second is to extend the Thread Class. Read this post to learn more about [creating threads in java](http://www.journaldev.com/1016/java-thread-example).  What does **join()** method?  The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.  What is difference between **wait() and sleep()** method?   |  |  | | --- | --- | | **wait()** | **sleep()** | | 1) The wait() method is defined in Object class. | The sleep() method is defined in Thread class. | | 2) wait() method releases the lock. | The sleep() method doesn't releases the lock. |   Can we call the **run() method** instead of **start()?**  yes, but it will not work as a thread rather it will work as a normal object so there will not be context-switching between the threads.  What is **Garbage Collection**?  Garbage collection is a process of reclaiming the runtime unused objects. It is performed for memory management.  What is **gc()?**  gc() is a daemon thread.gc() method is defined in System class that is used to send request to JVM to perform garbage collection.  What is difference between **user Thread and daemon Thread**?  When we create a Thread in java program, it’s known as user thread. A daemon thread runs in background and doesn’t prevent JVM from terminating. When there are no user threads running, JVM shutdown the program and quits. A child thread created from daemon thread is also a daemon thread.  What are different states **in lifecycle of Thread**?  When we create a Thread in java program, its state is New. Then we start the thread that change it’s state to Runnable. Thread Scheduler is responsible to allocate CPU to threads in Runnable thread pool and change their state to Running. Other Thread states are Waiting, Blocked and Dead. Read this post to learn more about [life cycle of thread](http://www.journaldev.com/1044/thread-life-cycle-in-java-thread-states-in-java).  What is the difference between **notify()** and **notify All()?**  The notify() is used to unblock one waiting thread whereas notify All() method is used to unblock all the threads in waiting state.  What is **deadlock?**  Deadlock is a situation when two threads are waiting on each other to release a resource. Each thread waiting for a resource which is held by the other waiting thread. |
|  |

**COLLECTION**

Collections in java is a framework that provides an architecture to store and manipulate the group of objects.

**ARRAY LIST:**

It inherits Abstract List class and implements List interface. Java Array List class can contain duplicate elements, maintains insertion order, class is non-synchronized. Array List supports dynamic arrays that can grow as needed. Standard Java arrays are of a fixed length.

**LINKED LIST:**

Java Linked List class uses doubly linked list to store the elements. It provides a linked-list data structure. It inherits the Abstract List class and implements List and Deque interfaces.

* Java Linked List class can contain duplicate elements, maintains insertion order.
* Java Linked List class is non-synchronized.
* In Java Linked List class, manipulation is fast because no shifting needs to be occurred.

****

Difference btw **Array List Linked List**

|  |  |
| --- | --- |
| **Array List** | **Linked List** |
| 1) Array List internally uses **dynamic array** to store the elements. | Linked List internally uses **doubly linked list** to store the elements. |
| 2) Manipulation with Array List is **slow** because it internally uses array. | Manipulation with Linked List is **faster** than Array List because it uses doubly linked list |
| 4) Array List is **better for storing and accessing** data. | Linked List is **better for manipulating** data. |

**HASH SET**

Hash Set extends Abstract Set and implements the Set interface. It creates a collection that uses a hash table for storage.

A hash table stores information by using a mechanism called **hashing**. In hashing, the informational content of a key is used to determine a unique value, called its hash code.

The hash code is then used as the index at which the data associated with the key is stored. The transformation of the key into its hash code is performed automatically.

**TREE SET** Java Tree Set class implements the Set interface that uses a tree for storage. It inherits Abstract Set class and implements Navigable Set interface. The objects of Tree Set class are stored in ascending order. Contains unique elements only like Hash Set, Maintains ascending order.Access and retrieval times are quite fast.

**QUEUE:**

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.

**Priority Queue class**

The Priority Queue class provides the facility of using queue. But it does not order the elements in FIFO manner. It inherits Abstract Queue class.

**HASHMAP**

Java Hash Map class implements the map interface by using a hash table. It inherits Abstract Map class and implements Map interface.

A Hash Map contains values based on the key. It may have one null key and multiple null values. It contains only unique elements. It maintains no order.

**Addition of element in HashMap**

In order to add any element you need to provide 2 thing, key and value.

**Key :** key with which specified value will be associated. **null** is allowed.

**Value :** value to be associated with specified key.

First HashMap will generate hashcode for given key and then check if there is any value already associated with given key or not. If yes then it will return already associated value. Else it will add value in HashMap in with provided key.

**What is ReHashing**

Moment you add 13th element in given HashMap, Threshold limit is crossed for given HashMap and system will create a new backing keyset array(Size of this array will be double of previous array). System will have to again calculate exact bucket where elements from previous bucket should be placed and all elements from old HashMap will be copied to new HashMap. This whole process is called ReHashing because Hashcode is calculated for each element again.

Because overtime HashMap might be reHashed and order could get change.

**HASHTABLE**

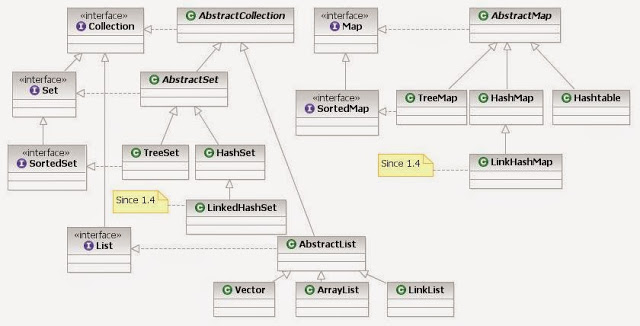
Java Hash table class implements a hash table, which maps keys to values. It inherits Dictionary class and implements the Map interface.

DIFFERENCE BETWEEN **Hash Map -Hash table**

|  |  |
| --- | --- |
| **Hash Map** | **Hash table** |
| 1) Hash Map is non-synchronized. It is not-thread safe | Hash table is synchronized. It is thread-safe and can be shared with many threads. |
| 2) Hash Map allows one null key and multiple null values. | Hash table doesn't allow any null key or value. |
| 4) Hash Map is fast. | Hash table is slow. |

DIFFERENCE BETWEEN **Array List and Vector**

|  |  |
| --- | --- |
| **Array List** | **Vector** |
| 1) Array List is **not synchronized**. | Vector is **synchronized**. |
| 2) Array List **increments 50%** of current array size if number of element exceeds from its capacity. | Vector **increments 100%** means doubles the array size if total number of element exceeds than its capacity. |
| 4) Array List is **fast** because it is non-synchronized. | Vector is **slow** because it is synchronized |



**What is the difference between List and Set ?**  
  
Set contain only unique elements while List can contain duplicate elements.  
Set is unordered while List is ordered . List maintains the order in which the objects are added .  
  
**Q7 What is the difference between Map and Set ?**  
  
Map object has unique keys each containing some value, while Set contain only unique values.  
  
**Q8 What are the classes implementing List and Set interface ?**  
  
***Class implementing List interface :***  ArrayList , Vector , LinkedList ,  
  
***Class implementing Set interface :***HashSet , TreeSet

**Q9 What is an iterator ?**  
  
Iterator is an interface . It is found in java.util package. It provides methods to iterate over any Collection.  
  
  
**Q10 What is the difference between Iterator and Enumeration ?**  
  
The main difference between Iterator and Enumeration is that Iterator has remove() method while Enumeration doesn't.  
Hence , using Iterator we can manipulate objects by adding and removing the objects from the collections. Enumeration behaves like a read only interface as it can only traverse the objects and fetch it .  
  
**Q11 Which design pattern followed by Iterator ?**  
  
It follows iterator design pattern. Iterator design pattern provides us to navigate through the collection of objects by using a common interface without letting us know about the underlying implementation.  
  
Enumeration is an example of Iterator design pattern.

**What is the difference between Queue and Stack ?**  
  
Queue is a data structure which is based on FIFO ( first in first out ) property . An example of Queue in real world is buying movie tickets in the multiplex or cinema theaters.  
  
Stack is a data structure which is based on LIFO (last in first out) property . An example of Stack in real world is  insertion or removal of CD  from the CD case.

**What is the difference between peek(),poll() and remove() method of the Queue interface ?**  
  
Both poll() and remove() method is used to remove head object of the Queue. The main difference lies when the Queue is empty().  
If Queue is empty then poll() method will return null . While in similar case , remove() method will throw NoSuchElementException .  
peek() method retrieves but does not remove the head of the Queue. If queue is empty then peek() method also returns null.

The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java EE platform

### What is Executors Framework?

In Java 5, Executor framework was introduced with the java.util.concurrent.Executor interface.

The Executor framework is a framework for standardizing invocation, scheduling, execution, and control of asynchronous tasks according to a set of execution policies.

Creating a lot many threads with no bounds to the maximum threshold can cause application to run out of heap memory. So, creating a ThreadPool is a better solution as a finite number of threads can be pooled and reused. Executors framework facilitate process of creating Thread pools in java. Check out this post to learn with example code to [create thread pool using Executors framework](http://www.journaldev.com/1069/threadpoolexecutor-java-thread-pool-example-executorservice).

### What is Executors Class?

Executors class provide utility methods for Executor, ExecutorService, ScheduledExecutorService, ThreadFactory, and Callable classes.

Executors class can be used to easily create Thread Pool in java, also this is the only class supporting execution of Callable implementations.

**Deadlock** describes a situation where two or more threads are blocked forever, waiting for each other. ... A **Java** multithreaded program may suffer from the**deadlock** condition because the synchronized keyword causes the executing thread to block while waiting for the lock, or monitor, associated with the specified object.

The problem discussed before can be avoided by applying a suitable deadlock prevention technique. Following are some techniques available that help mitigate the deadlock problem:

* Non Overlapping Locks
* Lock Ordering
* Lock Timeout
* Single Thread

**JDBC**

**JDBC:** Java Database Connectivity

JDBC is a Java API that is used to connect and execute query to the database. JDBC API uses jdbc drivers to connects to the database.

**JDBC Driver**

JDBC Driver is a software component that enables java application to interact with the database.

JDBC-ODBC bridge driver

How to you **load the drivers** in JDBC?

Class.forName() method is used in JDBC to load the JDBC drivers dynamically.

What are the steps to **connect** to the database in java?

* Registering the driver class
* Creating connection
* Creating statement
* Executing queries
* Closing connection

Explain how you **can establish a connection** ?

Loading Drivers  
Class.forName(“Driver”);  
Getting connection  
Connection con = DriverManager.getConnection(url,”myLogin”,

“myPassword”);

What is the difference between **Statement and Prepared Statement** interface?

In case of Statement, query is complied each time whereas in case of Prepared Statement, query is complied only once. So performance of Prepared Statement is better than Statement.

How can we execute stored procedures and functions?

By using **Callable statement** interface, we can execute procedures and functions.

**J2EE DEFINITIONS**

 What is **J2EE**?

J2EE means Java 2 Enterprise Edition. The functionality of J2EE is developing multitier web-based applications. The J2EE platform is consists of a set of services, application programming interfaces (APIs), and protocols.

=====================================================================================

**Web server:**

It will handle the traffic coming in and determine what to do. It will create the request and response objects. I have used Apache **Tomcat** web server.

Apache **Tomcat** is used to deploy your Java Servlets and JSPs. So, in your Java project you can build your WAR (short for Web Archive) file, and just drop it in the deploy directory in **Tomcat**. So basically, Apache is an HTTP **Server**, serving HTTP. **Tomcat** is a Servlet and JSP **Server** serving Java technologies.

=====================================================================================

**Servlets:**

Servlets is a server side components that provide a powerful mechanism for developing server side programs. Servlets is a server as well as platform-independent and Servlets are designed for a various protocol. Most commonly used HTTP protocols. All servlets must implement the Servlet interface, which defines life-cycle methods.

==>A Servlet is a Java-based, server-side web technology.

==>It is a special class in Java EE that may respond to requests.

==>They are generally used as the Controller component in the MVC application.

**Life-cycle of a Servlet:**

Servlet is loaded

servlet is instantiated

servlet is initialized

service the request servlet is destroyed

**JSP**: Java Server Pages.

JSP is used to create web application just like Servlet technology.

A JSP page consists of HTML tags and JSP tags.

Easy to maintain than Servlet.

Less code than Servlet.

It provides some additional features such as Expression Language, Custom Tag etc.

What are the **JSP tag?**

In JSP tags can be divided into 4 different types.

* Directives
* Declarations
* Scriplets
* Expressions

What are **JSP Directives**?

* 1.page Directives <%@page language=”java” %>
* 2. include Directives: <%@ include file=”/header.jsp” %>
* 3. Tag lib Directives <%@ taglib uri=”tlds/taglib.tld” prefix=”html” %>

What is **Struts**?

Struts framework is a Model-View-Controller(MVC) architecture for designing large scale applications. Which is combines of Java Servlets, JSP, Custom tags, and message. Struts helps you to create an extensible development environment for your application, based on published standards and proven design patterns. Model in many applications represent the internal state of the system as a set of one or more JavaBeans. The *View* is most often constructed using Java Server Pages (JSP) technology. The Controller is focused on receiving requests from the client and producing the next phase of the user interface to an appropriate View component. The primary component of the Controller in the framework is a servlet of class Action Servlet. This servlet is configured by defining a set of Action Mappings.

**Cookies:** Cookies are text files stored on the client computer. They are used for information tracking purpose. It will not work if cookie is disabled from the browser.**Diff b/w Cookies and Http Session:**Cookie works at client side whereas Http Session works at server side.

>>>**AJAX**

Asynchronous JavaScript and XML

It can send and receive information in various formats, including JSON, XML, HTML, and text files.

=====================================================================================

Spring

Spring is the most popular application development framework for enterprise Java



## **Core Container**

* The **Core** module provides the fundamental parts of the framework, including the IoC and Dependency Injection features.
* The **Bean** module provides BeanFactory, which is a sophisticated implementation of the factory pattern.
* The **Context** module builds on the solid base provided by the Core and Beans modules and it is a medium to access any objects defined and configured. The ApplicationContext interface is the focal point of the Context module.
* The **SpEL** module provides a powerful expression language for querying and manipulating an object graph at runtime.

## **Data Access/Integration**

* The **JDBC** module provides a JDBC-abstraction layer that removes the need for tedious JDBC related coding.
* The **ORM** module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.
* The **OXM** module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.
* The Java Messaging Service **JMS** module contains features for producing and consuming messages.
* The **Transaction** module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.

## **Web**

* The **Web** module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.
* The **Web-MVC** module contains Spring's Model-View-Controller (MVC) implementation for web applications.
* The **Web-Socket** module provides support for WebSocket-based, two-way communication between the client and the server in web applications.
* The **Web-Portlet** module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

IoC Container

The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets informations from the XML file and works accordingly. 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:

1. **BeanFactory container—**XmlBeanFactory class.

**T**he BeanFactory is usually preferred where the resources are limited like mobile devices or applet-based applications.

1. **ApplicationContextcontainer-** FileSystemXmlApplicationContext /ClassPathXmlApplicationContext/WebXmlApplicationContext

This container adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners.

The *ApplicationContext* container includes all functionality of the *BeanFactory*container, so it is generally recommended over *BeanFactory*. BeanFactory can still be used for lightweight applications like mobile devices or applet-based applications where data volume and speed is significant.

# **Bean**

The objects that form the backbone of your application and that are managed by the Spring IoC container are called **beans**.

**BEAN LIFE CYCLE**

To define setup and teardown for a bean, we simply declare the <bean> with **initmethod** and/or **destroy-method** parameters.

The **init-method** attribute specifies a method that is to be called on the bean immediately upon instantiation.

Similarly, **destroymethod** specifies a method that is called just before a bean is removed from the container.

# **Bean Scopes**

When defining a <bean> you have the option of declaring a scope for that bean.

## **The singleton scope**

If a scope is set to singleton, the Spring IoC container creates exactly one instance of the object defined by that bean definition.

<!-- A bean definition with singleton scope -->

<bean id = "..." class = "..." scope = "singleton">

<!-- collaborators and configuration for this bean go here -->

</bean>

## **The prototype scope**

If the scope is set to prototype, the Spring IoC container creates a new bean instance of the object every time a request for that specific bean is made.

As a rule, use the prototype scope for all state-full beans and the singleton scope for stateless beans

<!-- A bean definition with singleton scope -->

<bean id = "..." class = "..." scope = "prototype">

<!-- collaborators and configuration for this bean go here -->

</bean>

# Bean Post Processors

The **BeanPostProcessor** interface defines callback methods that you can implement to provide your own instantiation logic, dependency-resolution logic, etc.

You can also implement some custom logic after the Spring container finishes instantiating, configuring, and initializing a bean by plugging in one or more BeanPostProcessor implementations.

You can configure multiple BeanPostProcessor interfaces and you can control the order in which these BeanPostProcessor interfaces execute by setting the **order** property provided the BeanPostProcessor implements the **Ordered**interface.

# Injecting Inner Beans

Thus, a <bean/> element inside the <property/> or <constructor-arg/> elements is called inner bean and it is shown below.

<bean id = "outerBean" class = "...">

<property name = "target">

<bean id = "innerBean" class = "..."/>

</property>

</bean>

# Dependency Injection in Spring

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

When writing a complex Java application, application classes should be as independent as possible of other Java classes to increase the possibility to reuse these classes and to test them independently of other classes while unit testing. Dependency Injection (or sometime called wiring) helps in gluing these classes together and at the same time keeping them independent.

### **Two ways to perform Dependency Injection in Spring framework**

Spring framework provides two ways to inject dependency

* **By Constructor**
* **By Setter method**

# Dependency Injection by Constructor Example

We can inject the dependency by constructor. The **<constructor-arg>**subelement of **<bean>** is used for constructor injection

<bean id = "textEditor" class = "com.tutorialspoint.TextEditor">

<constructor-arg ref = "spellChecker"/>

</bean>

# Dependency Injection by setter method

We can inject the dependency by setter method also. The **<property>**subelement of **<bean>** is used for setter injection

<bean id = "textEditor" class = "com.tutorialspoint.TextEditor">

<property name = "spellChecker" ref = "spellChecker"/>

</bean>

You can mix both, Constructor-based and Setter-based DI

But it is a good rule of thumb to **use constructor arguments for mandatory dependencies and setters for optional dependencies**.

# 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.

# **Beans Auto-Wiring**

The Spring container can **autowire**relationships between collaborating beans without using <constructor-arg> and <property> elements, which helps cut down on the amount of XML configuration you write for a big Spring-based application.

## **Autowiring Modes**

Following are the autowiring modes, which can be used to instruct the Spring container to use autowiring for dependency injection. You use the autowire attribute of the <bean/> element to specify **autowire** mode for a bean definition.

|  |  |
| --- | --- |
| **Sr.No** | **Mode & Description** |
| 1 | **No**  This is default setting which means no autowiring and you should use explicit bean reference for wiring. You have nothing to do special for this wiring. This is what you already have seen in Dependency Injection chapter. |
| 2 | [**byName**](https://www.tutorialspoint.com/spring/spring_autowiring_byname.htm)  Autowiring by property name. Spring container looks at the properties of the beans on which *autowire* attribute is set to *byName* in the XML configuration file. It then tries to match and wire its properties with the beans defined by the same names in the configuration file. |
| 3 | [**byType**](https://www.tutorialspoint.com/spring/spring_autowiring_bytype.htm)  Autowiring by property datatype. Spring container looks at the properties of the beans on which *autowire* attribute is set to *byType*in the XML configuration file. It then tries to match and wire a property if its **type** matches with exactly one of the beans name in configuration file. If more than one such beans exists, a fatal exception is thrown. |
| 4 | [**constructor**](https://www.tutorialspoint.com/spring/spring_autowiring_byconstructor.htm)  Similar to byType, but type applies to constructor arguments. If there is not exactly one bean of the constructor argument type in the container, a fatal error is raised. |
| 5 | **autodetect**  Spring first tries to wire using autowire by *constructor*, if it does not work, Spring tries to autowire by *byType*. |

You can use **byType** or **constructor** autowiring mode to wire arrays and other typed-collections.

Starting from Spring 2.5 it became possible to configure the dependency injection using **annotations**. So instead of using XML to describe a bean wiring, you can move the bean configuration into the component class itself by using annotations on the relevant class, method, or field declaration.

Annotation injection is performed before XML injection. Thus, the latter configuration will override the former for properties wired through both approaches.

Annotation wiring is not turned on in the Spring container by default. So, before we can use annotation-based wiring, we will need to enable it in our Spring configuration file. So consider the following configuration file in case you want to use any annotation in your Spring application.

<?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-3.0.xsd

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

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:annotation-config/>

<!-- bean definitions go here -->

</beans>

|  |  |
| --- | --- |
| **Sr.No.** | **Annotation & Description** |
| 1 | [**@Required**](https://www.tutorialspoint.com/spring/spring_required_annotation.htm)  The @Required annotation applies to bean property setter methods. |
| 2 | [**@Autowired**](https://www.tutorialspoint.com/spring/spring_autowired_annotation.htm)  The @Autowired annotation can apply to bean property setter methods, non-setter methods, constructor and properties. |
| 3 | [**@Qualifier**](https://www.tutorialspoint.com/spring/spring_qualifier_annotation.htm)  The @Qualifier annotation along with @Autowired can be used to remove the confusion by specifiying which exact bean will be wired. |
| 4 | [**JSR-250 Annotations**](https://www.tutorialspoint.com/spring/spring_jsr250_annotations.htm)  Spring supports JSR-250 based annotations which include @Resource, @PostConstruct and @PreDestroy annotations. |

# Spring @Required Annotation

# The @Required annotation applies to bean property setter methods

Here is the content of **Student.java** file −

package com.tutorialspoint;

import org.springframework.beans.factory.annotation.Required;

public class Student {

private Integer age;

private String name;

@Required

public void setAge(Integer age) {

this.age = age;

}

public Integer getAge() {

return age;

}

@Required

public void setName(String name) {

this.name = name;

}

public String getName() {

return name;

}

}

<?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-3.0.xsd

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

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:annotation-config/>

<!-- Definition for student bean -->

<bean id = "student" class = "com.tutorialspoint.Student">

<property name = "name" value = "Zara" />

<property name = "age" value = "11"/>

</bean>

</beans>

# **@Autowired Annotation**

# The @Autowired annotation can be used to autowire bean on the setter method just like @Required annotation, constructor, a property or methods with arbitrary names and/or multiple arguments.

Here is the content of **TextEditor.java** file

package com.tutorialspoint;

import org.springframework.beans.factory.annotation.Autowired;

public class TextEditor {

private SpellChecker spellChecker;

@Autowired

public void setSpellChecker( SpellChecker spellChecker ){

this.spellChecker = spellChecker;

}

public SpellChecker getSpellChecker( ) {

return spellChecker;

}

public void spellCheck() {

spellChecker.checkSpelling();

}

}

Following is the configuration file **Beans.xml**

<?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-3.0.xsd

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

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:annotation-config/>

<!-- Definition for textEditor bean without constructor-arg -->

<bean id = "textEditor" class = "com.tutorialspoint.TextEditor"></bean>

<!-- Definition for spellChecker bean -->

<bean id = "spellChecker" class = "com.tutorialspoint.SpellChecker"></bean>

</beans>

## **@Autowired with (required=false) option**

By default, the @Autowired annotation implies the dependency is required similar to @Required annotation, however, you can turn off the default behavior by using the **(required=false)** option with @Autowired.

# **@Qualifier Annotation**

# There may be a situation when you create more than one bean of the same type and want to wire only one of them with a property. In such cases, you can use the @Qualifier annotation along with @Autowired to remove the confusion by specifying which exact bean will be wired. Following is an example to show the use of @Qualifier annotation.

Here is the content of **Profile.java** file

package com.tutorialspoint;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Qualifier;

public class Profile {

@Autowired

@Qualifier("student1")

private Student student;

public Profile(){

System.out.println("Inside Profile constructor." );

}

public void printAge() {

System.out.println("Age : " + student.getAge() );

}

public void printName() {

System.out.println("Name : " + student.getName() );

}

}

Consider the example of following configuration file **Beans.xml**

<?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-3.0.xsd

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

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:annotation-config/>

<!-- Definition for profile bean -->

<bean id = "profile" class = "com.tutorialspoint.Profile"></bean>

<!-- Definition for student1 bean -->

<bean id = "student1" class = "com.tutorialspoint.Student">

<property name = "name" value = "Zara" />

<property name = "age" value = "11"/>

</bean>

<!-- Definition for student2 bean -->

<bean id = "student2" class = "com.tutorialspoint.Student">

<property name = "name" value = "Nuha" />

<property name = "age" value = "2"/>

</bean>

</beans>

Inside Profile constructor.

Age : 11

Name : Zara

# Spring AOP Tutorial

**Aspect Oriented Programming** (AOP) compliments OOPs in the sense that it also provides modularity. But the key unit of modularity is aspect than class.

AOP breaks the program logic into distinct parts (called concerns). It is used to increase modularity by **cross-cutting concerns**.

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

## **AOP Terminologies**

Before we start working with AOP, let us become familiar with the AOP concepts and terminology. These terms are not specific to Spring, rather they are related to AOP.

|  |  |
| --- | --- |
| **Sr.No** | **Terms & Description** |
| 1 | **Aspect**  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**

Spring aspects can work with five kinds of advice mentioned as follows −

|  |  |
| --- | --- |
| **Sr.No** | **Advice & Description** |
| 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. |

# **Transaction Management**

Transaction management is an important part of RDBMS-oriented enterprise application to ensure data integrity and consistency. The concept of transactions can be described with the following four key properties described as **ACID** −

* **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.

A real RDBMS database system will guarantee all four properties for each transaction. The simplistic view of a transaction issued to the database using SQL is as follows −

* Begin the transaction using *begin transaction* command.
* Perform various deleted, update or insert operations using SQL queries.
* If all the operation are successful then perform *commit* otherwise *rollback* all the operations.

# **Spring - MVC Framework**

* 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.

**Hibernate**:

SDLC, Java, Framework, Design patterns, Tools, Databases, Team role.

==>Hibernate is an Object Relationship Mapping(ORM) Framework.

**Hibernate is database independent.**

**ORM**: Is a framework for mapping an Object-oriented domain model to a traditional relational database.

==>Hibernate generates the SQL automatically.

Hibernate is a framework to map Java classes into Database tables.

Hibernate is used to change from one database to another database.

==>Hibernate basically requires **two files**:

\*Hibernate Configuration file: It has the database connection parameter.

\*Hibernate Mapping file (Hbm) : It will map the bean class to database.

We can also use annotations rather than mapping files(XML).

**Mapping in Hibernate:**

One-to-One one-to-Many Many-to-Many many-one

**Why is Hibernate better than JDBC**

**Advantages** of hibernates:

* Hibernate supports relationships like One-To-Many, One-To-One, Many-To-Many-to-Many, Many-To-One
* This will also supports collections like List, Set, Map (Only new collections)
* In jdbc all exceptions are checked exceptions, so we must write code in try, catch and throws, but in hibernate we only have Un-checked exceptions, so no need to write try, catch, or no need to write throws.  Actually in hibernate we have the translator which converts checked to Un-checked ;)
* Hibernate has its own query language, i.e hibernate query language which is database independent
* So if we change the database, then also our application will works as HQL is database independent
* HQL contains database independent commands
* Hibernate supports annotations, apart from XML

**Disadvantages** of hibernates:

* You know some thing.., Its saying hibernate is little slower than pure JDBC, actually the reason being hibernate used to generate many SQL statements in run time, but i guess this is not the disadvantage :-)
* But there is one major disadvantage, which was boilerplate code issue, actually we need to write same code in several files in the same application, but spring eliminated this.

==========================================================================================================================================================================

Define **connection pooling**?

Connection pooling is a mechanism reuse the connection. which contains the number of already created object connection. So whenever there is a necessary for object, this mechanism is used to directly get objects without creating it.

=====================================================================================

==> **Caching?**

Storing the data temporarily in the RAM instead of querying again and again with database.

It will improve the performance of the system.

**SOAP:** Simple Object Access Protocol

It is a XML-based protocol for accessing web services.

The Simple Object Access Protocol (SOAP) uses XML to define a protocol for the exchange of information in distributed computing environments

**Advantages** of SOAP web services?

* WS Security (**WS Security**: SOAP defines its own security known as WS Security.)
* Language Independent Platform Independent.

**Disadvantages** of SOAP web services?

* Slow
* WSDL Dependent

What is **WSDL**?

WSDL stands for Web Services Description Language. It is a xml document containing information about web services such as method name, method parameter.

**UDDI** is an XML-based standard for describing, publishing, and finding web services.

* UDDI stands for **Universal Description, Discovery, and Integration.**
* UDDI is a specification for a distributed registry of web services.
* UDDI is a platform-independent, open framework.
* UDDI uses Web Service Definition Language(WSDL) to describe interfaces to web services.

**RESTful:**  Representational State Transfer

REST stands for **RE**presentational **S**tate **T**ransfer. REST is a web standards based architecture and uses HTTP Protocol for data communication. It revolves around resources where every component is a resource and a resource is accessed by a common interface using HTTP standard methods. REST was first introduced by Roy Fielding in year 2000.

In REST architecture, a REST Server simply provides access to resources and the REST client accesses and presents the resources. Here each resource is identified by URIs/ Global IDs. REST uses various representations to represent a resource like Text, JSON and XML. JSON is now the most popular format being used in Web Services.

It is an architectural style. It is not a protocol like SOAP.

**advantages** of RESTful web services?

* Fast
* Language Independent
* Platform Independent
* Can use SOAP.
* Allows different data format.
* What is the difference between SOAP and REST web services?

|  |  |  |
| --- | --- | --- |
| **No.** | **SOAP** | **REST** |
| 1) | SOAP is a **protocol**. | REST is an **architectural style**. |
| 3) | SOAP **can't use REST** because it is a protocol. | REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP. |
| 4) | SOAP **uses services interfaces to expose the business logic**. | REST **uses URI to expose business logic**. |
| 6) | SOAP **permits XML** data format only. | REST **permits different** data format such as Plain text, HTML, XML, JSON etc. |

====================================================================================

**SOA:** Service Oriented Architecture

It is a design pattern to provide services to other application through protocol.

**BUILDING TOOLS: Maven**

Maven is a project management tool. It is based on POM (Project Object Model).

**Advantages** of Maven?

* No need to add jar file in each project
* Creates right directory structure
* Builds and deploys the project

**POM:** Project Object Model.

The pom.xml file contains information of project and project configuration.

**ANT:** Another Needed Tool

Ant is a build tool that is java based. A build tool performs the following tasks:  
  
Compiling java code into byte code  
Placing this byte code in a package  
Deployment to production systems  
Document creation and release notes preparation.

What is the difference between Ant and Maven?

|  |  |
| --- | --- |
| **Ant** | **Maven** |
| It is **a tool** box. | It is **a framework**. |
| It is **mainly a build tool**. | It is **mainly a project management tool**. |
| There is **no life cycle**. | There is **life cycle**. |
| Ant **doesn't have formal conventions**. | Maven **has a convention** to place source code, compiled code etc. |
| Ant is **procedural**. | Maven is **declarative**. |
| The ant scripts are **not reusable**. | The maven plugins are **reusable**. |

**JAVA TESTING**

**=====================================================================================**

What is **Testing**?

Testing is the process of checking the functionality of the application whether it is working as per requirements.

What is **Unit Testing**?

Unit testing is the testing of single entity (class or method). Unit testing is very essential to every software company to give a quality product to their customers.

What is **JUnit**? 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.

What is a **Unit Test Case**?

A Unit Test Case is a part of code which ensures that the another part of code (method) works as expected. To achieve those desired results quickly, test framework is required. JUnit is perfect unit test framework for java programming language.

What is **log4j**?

log4j is a reliable, fast and flexible logging framework (APIs) written in Java, which is distributed under the Apache Software License. log4j is highly configurable through external configuration files at runtime. It views the logging process in terms of levels of priorities and offers mechanisms to direct logging information to a great variety of destinations, such as a database, file, console, UNIX Syslog, etc.

Difference between **client and server and web based application**?

Client server applications Majorly used in Intranet areas, but web server applications used in Internet areas. ... Web apps runs on a browser, client-server apps runs as .exe. Web applications run on server, Client-Server apps run on client side. Web application is thin client, client-server app is thick client.

**SQL DEFINITIONS**

What is **DBMS?**

A Database Management System (DBMS) is a program that controls creation, maintenance and use of a database. DBMS can be termed as File Manager that manages data in a database rather than saving it in file systems.

 What is **RDBMS?**

RDBMS stands for Relational Database Management System. RDBMS store the data into the collection of tables, which is related by common fields between the columns of the table. It also provides relational operators to manipulate the data stored into the tables

what is **transaction** in sql?

A transaction is a single unit of work. If a transaction is successful, all of the data modifications made during the transaction are committed and become a permanent part of the database. If a transaction encounters errors and must be canceled or rolled back, then all of the data modifications are erased.

What is **SQL?**

SQL stands for structured query language. It is a database language used for database creation, deletion, fetching rows and modifying rows etc. sometimes it is pronounced as se-qwell.

What is **data definition language(DDL)?**

Data definition language(DDL) allows you to **CREATE, ALTER and DELETE** database objects such as schema, tables, view, sequence etc.

What is **data manipulation language(DML)?**

Data manipulation language makes user able to access and manipulate data. It is used to perform following operations.

* Insert data into database
* Retrieve data from the database
* Update data in the database
* Delete data from the database

What is **data control language(DCL)?**

Data control language allows you to control access to the database. It includes two commands GRANT and REVOKE.

**GRANT:** to grant specific user to perform specific task.

**REVOKE:** to cancel previously denied or granted permissions.

What is a **primary key?**

A primary key is a combination of fields which uniquely specify a row. This is a special kind of unique key. Primary key values cannot be NULL.

What is a **foreign key?**

A foreign key is specified as a key which is related to the primary key of another table. Relationship needs to be created between two tables by referencing foreign key with the primary key of another table.

What is a **unique key?**

A Unique key constraint uniquely identifies each record in the database. This provides uniqueness for the column or set of columns.

What is a **join**?

This is a keyword used to query data from more tables based on the relationship between the fields of the tables. Keys play a major role when JOINs are used.

What are the **types of join** and explain each?

There are various types of join which can be used to retrieve data and it depends on the relationship between tables.

**Inner join.**

Inner join return rows when there is at least one match of rows between the tables.

**Right Join.**

Right join return rows which are common between the tables and all rows of Right hand side table. Simply, it returns all the rows from the right hand side table even though there are no matches in the left hand side table.

**Left Join.**

Left join return rows which are common between the tables and all rows of Left hand side table. Simply, it returns all the rows from Left hand side table even though there are no matches in the Right hand side table.

**Full Join.**

Full join return rows when there are matching rows in any one of the tables. This means, it returns all the rows from the left hand side table and all the rows from the right hand side table.

What is **Stored procedure?**

A stored procedure is a named group of sql statements that has been previously created and stored in the server database.

**USES:** Stored procedures are often used for data validation and as access control mechanism. Logic applied in applications can be centralized and stored in applications.

What is a **function** in sql?

A user-defined **function** is a Transact-**SQL** or common language runtime (CLR) routine that accepts parameters, performs an action, such as a complex calculation, and returns the result of that action as a value. The return value can either be a scalar (single) value or a table

What is a **SQL View**?

**SQL** CREATE **VIEW** Statement. In **SQL**, a **view** is a virtual table based on the result-set of an **SQL** statement. A **view** contains rows and columns, just like a real table. The fields in a **view** are fields from one or more real tables in the database.

What is a **query**?

A DB query is a code written in order to get the information back from the database. Query can be designed in such a way that it matched with our expectation of the result set. Simply, a question to the Database.

What is **subquery**?

A subquery is a query within another query. The outer query is called as main query, and inner query is called subquery. Sub Query is always executed first, and the result of subquery is passed on to the main query.

What is a **trigger**?

A DB trigger is a code or programs that automatically execute with response to some event on a table or view in a database. Mainly, trigger helps to maintain the integrity of the database.

Example: When a new student is added to the student database, new records should be created in the related tables like Exam, Score and Attendance tables.

What is the difference between **DELETE, TRUNCATE, DROP** commands?

**DELETE** command is used to remove rows from the table, and WHERE clause can be used for conditional set of parameters. Commit and Rollback can be performed after delete statement.

**TRUNCATE** removes all rows from the table. Truncate operation cannot be rolled back.

**DROP** command removes a table from the database and operation cannot be rolled back.

**UI DEFINITIONS**

**static web-page:** A static website contains Web pages with fixed content. Static sites are the most basic type of website and are the easiest to create.

**Dynamic Web-page:**

A dynamic web page is a web page that displays different content each time it's viewed.

A dynamic website has a backend written in a language like JSP, Node.js, etc.

**JSON**

What is **JSON?**

JSON is a simple data exchange format.  JSON means JavaScript Object Notation; it is language and platform independent.

Explain what is **JSON objects?**

An object can be defined as an unordered set of name/value pairs.  An object in JSON starts with {left brace} and finish or ends with {right brace}.  Every name is followed by: (colon) and the name/value pairs are parted by, (comma).

Explain how to **transform JSON text to a JavaScript object**?

One of the common use of JSON is to collect JSON data from a web server as a file or HTTP request, and convert the JSON data to a JavaScript, ant then it avails the data in a web page.

**HTML/CSS/XML**

What is **HTML?**

HTML stands for Hyper Text Markup Language. It is a language of World Wide Web. It is a standard text formatting language which is used to create and display pages on the Web.

**Latest version- html5**

Html5 is more power full and easier than Html4. It has lot of new tags like <header>, <footer>, <nav>, <audio>, <video>, <main> etc. It also support graphics.

What is a **cascading style sheet**?

CSS stands for Cascading Style Sheets. CSS describes how HTML elements are to be displayed on screen, paper, or in other media.  It can control the layout of multiple web pages all at once. External stylesheets are stored in CSS files.

What is **CSS 3?**

**CSS3** is the latest evolution of the Cascading Style Sheets language and aims at extending CSS2.1. It brings a lot of long-awaited novelties, like rounded corners, shadows, gradients, transitions or animations, as well as new layouts like multi-columns, flexible box or grid layouts.

Difference between **XML and HTML**?

**XML**  **HTML**

User definable tags Defined set of tags designed for web display

Content driven Format driven

End tags required for well formed documents End tags not required

Quotes required around attributes values Quotes not required

Slash required in empty tags Slash not required

>>>**XML**

-->Extensible markup language.XML was designed to store and transport data.

What is a **DTD(**Document Type Declaration**)** and a **Schema**?

The XML DTD contains markup declarations that provide a grammar for a class of documents.

This grammar is called as DTD.

**A Schema is:**

XML Schemas express shared vocabularies and allow machines to carry out rules made by people. They provide a means for defining the structure, content and semantics of XML documents. Schemas are a richer and more powerful of describing information than what is possible with DTDs.

What is **XSL**?

XSL stands for Extensible Stylesheet Language. It is a language for expressing stylesheets. These stylesheets are like CSS which describes how to display an XML document of a given type.

What is **XPath in XML**?

It is used *to retrieve elements from XML documents*. XPath expressions can be used to locate and retrieve elements, attributes and values from XML files because XML documents are structured.

What is **XSLT**?

**XSLT** is a popular XML technology which is used *to transform one XML file to other format like HTML* etc. XSLT is like a language which has its own syntax, functions and operator to transform XML documents. XSLT is also used to display data present in XML files as HTML pages.

**DTD vs XSD**

There are many differences between DTD (Document Type Definition) and XSD (XML Schema Definition). In short, DTD provides less control on XML structure whereas XSD (XML schema) provides more control.

The important differences are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **DTD** | **XSD** |
| 1) | DTD stands for **Document Type Definition**. | XSD stands for XML Schema Definition. |
| 2) | ~~DTDs are derived from~~**~~SGML~~**~~syntax.~~ | XSDs are written in XML. |
| 3) | DTD **doesn't support datatypes**. | XSD **supports datatypes** for elements and attributes. |
| 4) | DTD **doesn't support namespace**. | XSD **supports namespace**. |
| 5) | DTD **doesn't define order** for child elements. | XSD **defines order** for child elements. |
| 6) | DTD is **not extensible**. | XSD is **extensible**. |
| 7) | DTD is **not simple to learn**. | XSD is **simple to learn** because you don't need to learn new language. |
| 8) | DTD provides **less control** on XML structure. | XSD provides **more control** on XML structure. |

=====================================================================================

**jQuery:**

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation.

**JavaScript:**

**JavaScript** is a high-level, dynamic and interpreted run-time language.

JavaScript enables you to add powerful interactions to websites.

JavaScript is the foundation of a lot of commonly used libraries (like jQuery) and frameworks like AngularJS and used for validations. JavaScript enables you to add powerful interactions to websites.

**Angular js1**

AngularJS is a **JavaScript framework**. It can be added to an HTML page with a <script> tag.

AngularJS extends HTML attributes with **Directives**, and binds data to HTML with **Expressions**.

AngularJS is distributed as a JavaScript file, and can be added to a web page with a script tag:

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js"></script>

AngularJS extends HTML with **ng-directives**.

The **ng-app** directive defines an AngularJS application.

The **ng-model** directive binds the value of HTML controls (input, select, textarea) to application data.

The **ng-bind** directive binds application data to the HTML view.

<!DOCTYPE html>  
<html>  
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.4/angular.min.js"></script>  
<body>  
  
<div ng-app="">  
  <p>Name: <input type="text" ng-model="name"></p>  
  <p ng-bind="name"></p>  
</div>  
  
</body>  
</html>

AngularJS starts automatically when the web page has loaded.

The **ng-app** directive tells AngularJS that the <div> element is the "owner" of an AngularJS **application**.

The **ng-model** directive binds the value of the input field to the application variable **name**.

The **ng-bind** directive binds the **innerHTML** of the <p> element to the application variable **name**.

AngularJS Directives

As you have already seen, AngularJS directives are HTML attributes with an **ng** prefix.

The **ng-init** directive initializes AngularJS application variables.

## AngularJS Expressions

AngularJS expressions are written inside double braces: **{{ expression }}**.

AngularJS will "output" data exactly where the expression is written:

## AngularJS Applications

AngularJS **modules** define AngularJS applications.

AngularJS **controllers** control AngularJS applications.

The **ng-app** directive defines the application, the **ng-controller** directive defines the controller.

## The ng-app Directive

The ng-app directive defines the **root element** of an AngularJS application.

The ng-app directive will **auto-bootstrap** (automatically initialize) the application when a web page is loaded.

New directives are created by using the .directive function.

# **AngularJS**Scope

The scope is the binding part between the HTML (view) and the JavaScript (controller).

The scope is an object with the available properties and methods.

The scope is available for both the view and the controller.

When you make a controller in AngularJS, you pass the $scope object as an argument:

* View, which is the HTML.
* Model, which is the data available for the current view.
* Controller, which is the JavaScript function that makes/changes/removes/controls the data.

Then the scope is the Model.

The scope is a JavaScript object with properties and methods, which are available for both the view and the controller.

**Controller**

ng-controller directive tells AngularJS what controller to use with this view. AngularJS application mainly relies on controllers to control the flow of data in the application. A controller is a JavaScript object containing attributes/properties and functions. Each controller accepts $scope as a parameter which refers to the application/module that controller is to control

In AngularJS, a Controller is defined by a JavaScript constructor function that is used to augment the AngularJS Scope. When a Controller is attached to the **DOM** via the**ng-controller directive**, AngularJS will instantiate a new Controller object, using the**specified** Controller's constructor function.

**DATA BINDING**

Data binding is the automatic synchronization of data between model and view components. ng-model directive is used in data binding.

**Promises**

How does promise work?

The **Promise** constructor takes a function (an executor) that will be executed immediately and passes in two functions: resolve, which must be called when the**Promise** is resolved (passing a result), and reject, when it is rejected (passing an error).

What is a promise in Angularjs?

An **AngularJS promise** is a mechanism that lets you defer a stated action or series of actions at an earlier point of time until you explicitly declare that **promise** to be fulfilled (or resolved). **Promises** are useful for asynchronous operations. This video introduces the basic way to declare and resolve **promises**.

**Security Features** Provided By AngularJS?

AngularJS provides built-in protection from the following security flaws.

It prevents cross-side scripting attacks: Cross-site scripting is a technique where anyone can send a request from client side and can get the confidential information easily.

It prevents HTML injection attacks.

It prevents XSRF protection for server side communication: It can be handled by “Auth token” mechanism. When the user logins for the first time a user id and password is sent to the server and it will, in turn, return an auth token. Now, this token does the authentication in the future transactions.

What Are **Filters?** Explain Different Filters Provided By AngularJS?

An AngularJS Filter changes or transforms the data before passing it to the view. These Filters work in combination with AngularJS expressions or directives. AngularJS uses pipe character (“|”) to add filters to the expressions or directives. For example:

AngularJS provides following filters to transform data.

currency – It is used to format a number to a currency format.

date – It is required to format a date to a specified format.

filter – It chooses a subset of items from an array.

json – It formats an object to a JSON string.

limitTo – Its purpose is to create an array or string containing a specified number of elements/characters. The elements are selected, either from the beginning or the end of the source array or string. This depends on the value and sign (positive or negative) of the limit.

lowercase – This filter converts a string to lower case.

number – It formats a number as a text.

orderBy – It enables to sort an array. By default, sorting of strings happens alphabetically. And sorting of numbers is done numerically. It also supports a comparator function where we can define what will be counted as a match or not.

uppercase – This filter converts a string to upper case.

### Angular **Prefixes $ And $$?**

To prevent accidental name collisions within the code, AngularJS prefixes the names of public objects with $ and the names of private objects with $$.It is recommended that $ or $$ prefix should not be used in the code otherwise.

### **$RootScope**” In AngularJS?

**Answer.**

Every AngularJS application has a “$rootScope” that is the top most scope created on the DOM element. An app can have only one $rootScope which will be shared among all its components. It contains the ng-app directive. Every other scope is its child scope. It can watch expressions and propagate events. Using root scope we can set the value in one controller and read it from the other controller.

### Difference Between **$Scope And Scope**?

It is mandatory to use <$scope> while defining a controller. However, the “scope” will be used to create a link function for the custom directive. Both of them refer to “scope” object in AngularJS. The difference between them is that <$scope> uses dependency injection whereas “scope” does not.Factory methods like controller, filter, service etc receive its arguments via dependency injection (DI). In DI, the order of passing the arguments does not matter.

### AngularJS **Compiled?**

AngularJS uses <$compiler> service to compile the angular HTML page. Its compilation begins after the HTML page (static DOM) is fully loaded.

It occurs in two phases.

* **Compile –** It looks into the entire DOM and collects all of the directives. The result is a linking function.
* **Link –** It combines the directives with a scope and produces a live view. Any changes in the <scope model> get reflected in the view, and any operations done by the user in the view gets reflected in the <scope model>.

**Angular 2**

What is **Angular 2**?

Angular 2 is a most popular framework for developing mobile apps.  It is also for “desktop” as well “mobile” applications. Angular js2 has most of the architectural component from such as module, directive, service, dependency injection, change detection and modularity.

Advantages of **angular 2 over angular 1**?

The core differences and many more advantages on Angular2 vs. Angular 1 as following,

1.      Angular2 has better performance, more powerful template system.

2.      Angular2 provide simpler APIs, lazy loading and easier to application debugging.

3.      Angular2 much more testable, provides to nested level components

4.      Angular2 execute run more than two programs at the same time.

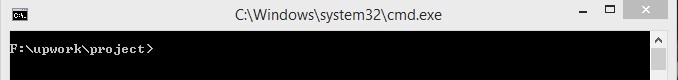
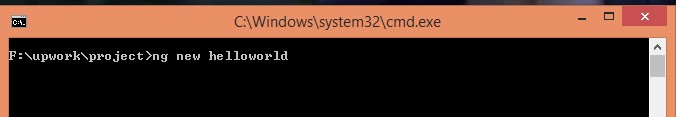
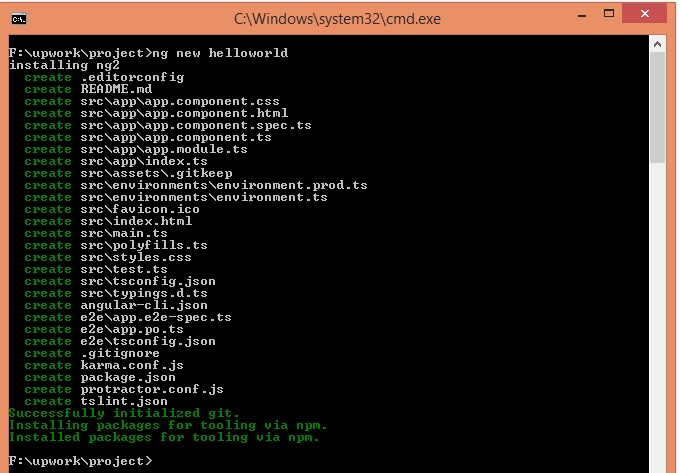
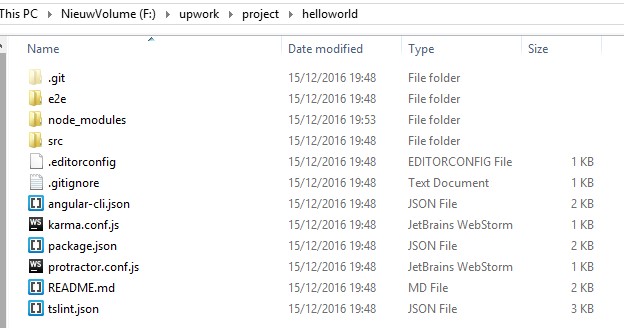
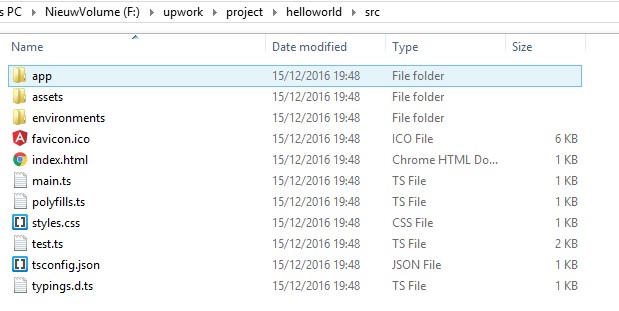
5.      Angular1 is controllers and $scope based but Angular2 is component based.

1. Angular 2 is a platform not only a language:
2. Better Speed and Performance: *No $Scope in Angular 2, AOT*
3. Simpler Dependency Injection
4. Modular, cross platform
5. Benefits of ES6 and Typescript.
6. Flexible Routing with Lazy Loading Features
7. Easier to Learn

**Responsive Web design** is the approach that suggests that design and development should respond to the user's behavior and environment based on screen size, platform and orientation.

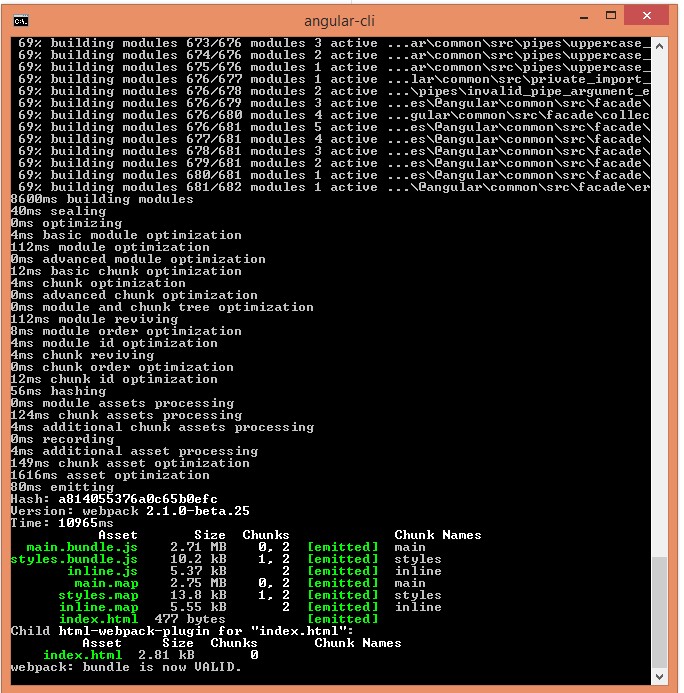
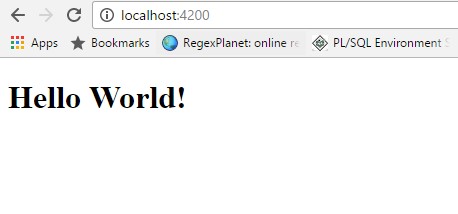
### How to develop your first Angularjs 2 application in steps to render simple text?

We are going to follow a step by step approach to develop our first Angular2 Application.

1. Open the command prompt on the workspace folder called ‘project’.
2. Run command **‘ng new <project name>’**. Lets suppose the project name is ‘helloworld’. This command will create a new project named ‘helloworld’. This may take few minutes.  
   This command will create the main app component and relevant template along with typings, tsconfig, styles and e2e configuration.  
   This step will also install all necessary required dependency library for the new project.Inside ‘helloworld’ project folder, there will be these following files created.  
   The source code for the application are inside ‘src’ folder as following:Index.html is the main html file for the application.Therefore, in app.component.ts **<app-root>** has been described as the main selector.  
   Now as we want to render a simple text. So in our AppComponent, we have taken a variable named ‘title’ and the template url of AppComponent is app.component.html.Next step is to render title in app.component.html as following



|  |  |
| --- | --- |
| 1  2  3 | <h1>     {{title}}  </h1> |

1. The next step is to run the project with the command **‘ng serve’.**a. Go to the ‘helloworld’ folder by running  ‘cd helloworld’ command  
   b. Run ‘ng serve’ in command prompt
2. From browser run ‘http://localhost:4200’and the output would be as following:

**2way data binding:**

There’s one directive in Angular2 that implements two-way data binding: ngModel.

Bi direction **databinding**: data from view to component and component to view.

**Inbuilt directives-ng model:**

* [**ngIf**](http://lishman.io/angular-ngif) adds and removes elements in the DOM based on the results of an expression.
* [**ngSwitch**](http://lishman.io/angular-ngswitch) displays one element (and its children) from a set of possible options, based on some condition.
* [**ngFor**](http://lishman.io/angular-ngfor) is a repeater directive which outputs a list of elements by iterating over an array.
* [**ngClass**](http://lishman.io/angular-ngclass) adds and removes CSS classes on an element.
* [**ngStyle**](http://lishman.io/angular-ngstyle) sets CSS styles on an HTML element conditionally.

**MEAN STACK**

The MEAN stack is a free and open-source JavaScript software stack for building dynamic web sites and web applications. The MEAN stack makes use of MongoDB, EXPRESS.JS, Angular, and Node.js. MEAN applications can be written in one language for both server-side and client-side execution environments.

What is **Node.js**?

Node.js is a web application framework built on Google Chrome's JavaScript Engine (V8 Engine).

Node.js comes with runtime environment on which a Java script based script can be interpreted . This runtime allows to execute a JavaScript code on any machine outside a browser. Because of this runtime of Node.js, JavaScript is now can be executed on server as well.

Node.js also provides a rich library of various java script modules which eases the development of web application using Node.js to great extents.

Node.js = Runtime Environment + JavaScript Library.

**REACT JS**

**React** is an open-source JavaScript library for building user interfaces. It is maintained by Facebook, Instagram and a community of individual developers and corporations.

Difference between **angular and react js**?

The main **differences between AngularJS** (the framework) and **React JS** (the library) are in the following aspects: componentization, data binding, performance, dependency resolution, directives, and templating. But **React JS** still doesn't let us create applications on its own.

**BOOTSTRAP:**

**Bootstrap** is a [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source_software) front-end [web framework](https://en.wikipedia.org/wiki/Web_framework) for designing [websites](https://en.wikipedia.org/wiki/Website) and [web applications](https://en.wikipedia.org/wiki/Web_application). It contains [HTML](https://en.wikipedia.org/wiki/HTML)- and [CSS](https://en.wikipedia.org/wiki/CSS)-based design templates for [typography](https://en.wikipedia.org/wiki/Typography), forms, buttons, navigation and other interface components, as well as optional [JavaScript](https://en.wikipedia.org/wiki/JavaScript) extensions. it concerns itself with [front-end development](https://en.wikipedia.org/wiki/Front-end_web_development) only.

**BOOTSTRAPPING:**

**bootstrapping** usually refers to a self-starting process that is supposed to proceed without external input. In [computer technology](https://en.wikipedia.org/wiki/Computer_technology) the term (usually shortened to **booting**) usually refers to the process of loading the basic software into the memory of a computer after power-on or general reset, especially the [operating system](https://en.wikipedia.org/wiki/Operating_system) which will then take care of loading other software as needed.

What is **component** in Angular 2?

In [Angular 2](http://www.code-sample.com/2015/07/angularjs-2-documentation-with-example.html), the components are the main way to build or specify HTML elements and business logic on the page. The component is the core functionality of Angular 2 app but we need to know to pass the data in to the components to configure them. To build an [Angular 2](http://www.code-sample.com/2015/07/angularjs-2-documentation-with-example.html) application you define a set of [components](http://www.code-sample.com/2016/06/angular-2-components.html), for every HTML elements, views, and route.

**Dependency injection:**

DI is a coding pattern in which a class receives dependencies from external source rather than creating them itself.

Injectable is a dependency (Any time you create a service include @Injectable decorator)

Angular create single instance of the service and inject it to both or many components==> "SINGLETON SERVICE".

**PROMISES:**

You can use a promise to specify what to do when an operation eventually succeeds or fails.

**Event binding:** Event binding allows us to work in reverse from property binding. We can send information **from** the view, **to** the component class. Such information usually involves a click, hover or typing. It is declared as a cur

**$event:** The **event** object will contain a lot of information that can be used in the logic defining the method.

**Services:**

A service is a class with specific purpose.

In angular we use services for 3 purposes

1. To Implement any business logic that is independent of any component. (Data of birth input🡺Age o/p, we don’t need any html view rather services do the work for us)
2. We use services to give components access to shared data.

Angular create single instance of the service and inject it to both or many components==> "SINGLETON SERVICE".

1. To handle external scenarios like connecting to a database.

**CONTROLLER:**

A controller is defined using ng-controller directive. A controller is a JavaScript object containing attributes/properties and functions. Each controller accepts $scope as a parameter which refers to the application/module that controller is to control.

What is **Directive** in Angular 2?

There are 3 types of directives in Angular 2.

1.     **Components** **Directives** - directives with a template

2.     **Structural Directives** - change the DOM layout by adding and removing DOM elements.

3.     **Attribute Directives** - change the appearance or behavior of an element, component, or other directive.

**Http service/call:**

The data which is given to the components through the service is fetched from a server. For that we should make use of the Http service that angular provides.

First inside the service component we invoke the GET method of the Http and in turn the Http service will send the request to the server.

The request is going to hit a web service or Web API which will fetch the data from the database and send it back as a Http response.

**Observables:**

The response we get from the Http service is called an **Observable.**

🡺The service component makes the request to the database using Http GET call. In return we get an Observable of type response.

🡺 An Observable is a sequence of items that arrive async over time. Observable is nothing but a single item that is Http response.

🡺Observable is not a desired format for our application, so we are going to map this observable to a JSON format.

map ==> Map operator is used to convert data to the desired format (JSON data).

catch ==> Works on an observable used to handle exceptions (This gets executed when there is an error or exemption).

* Promises:
  1. returns a single value
  2. not cancellable
* Observables:

1. works with multiple values over time
2. cancellable
3. supports map, filter, reduce and similar operators
4. proposed feature for ES 2016
5. use Reactive Extensions (RxJS)
6. an array whose items arrive asynchronously over time

**Routing** in Angular2?

Angular2 has improved so many features from AngularJs 1.x, Router component is one of them.

Routing is a mechanism which enables user to navigate between views/components. Angular 2 simplifies the routing and provide flexibility to configure and define at module level (Lazy loading).

<router-outlet> </router-outlet>.

What are **Modules** in Angularjs 2.0?

Both Angular 1.x and 2.0 apps are modular. However, in ***Angular2.0 NgModules*** has been introduced as its modularity system. Every Angular2.0 must have one module which acts as the root module and conventionally named as ***AppModule***.

**Lazy loading**

Lazy lading enables us to load only the module user is interacting and keep the rest to be loaded at runtime on demand.Lazy loading speeds up the application initial load time by splitting the code into multiple bundles and loading them on demand.

**LIFE CYCLE HOOKS**

**ngOnChanges**: Responds when angular sets its data-bound property which receives the current and previous object values.

**ngOnInit**: Initializes the component/directive after first ngOnChange triggers. This is most frequently used method to retrieve the data for the template from a back-end service.

**ngDoCheck**: Detect and act upon changes occuring outside Angular context. It is called when every change detection run.

**ngOnDestroy**: Cleanup just before Angular destroys the directive/component. Unsubscribe observables and detach event handlers to avoid memory leaks.

## **Deployment on NGNIX Servers on Windows**

One way Copy the src file and paste in the ngnix-html and open the <http://local> host.

Or giving ng-build and getting the dist folder and giving ng-serve for using web pack.

=====================================================================================

**TESTING**

AngularJS can be tested in a lot of ways, by lots of testing frameworks, including but certainly not limited to Mocha, Jasmine, Aunit, and Sinon.

In AngularJS 2, testing is provided in two ways: unit testing and end-to-end testing.

Writing Unit Tests in Angular 2 – Using Jasmine and Karma,

end-to-end tests using the **Protractor.**

**UNIT TESTING:**

Unit tests are all about splitting your code into small testable modules with each module having its own functionality and level of abstraction. This is where AngularJS shines.

**KARMA:**

**The karma test runner is ideal for writing and running unit tests while developing the application. It can be an integral part of the project's development and continuous integration processes.**

**Using Karma (with Jasmine)**

Karma creates a browser environment for the testing, so that you can test your code besides running your application. Karma is highly sophisticated, AngularJS 2 made it the preferred test runner. We can not only test code in your browser, but also in other devices, like phones, tablets, etc. Karma uses Jasmine, but it can be replaced with other frameworks, like Mocha and QUnit.

**Why Jasmine?** the most important thing about Jasmine is that it is dependency free and doesn’t even require a DOM! The provided spy function helps in telling how many times the function is called or not, and many more features!

**PROTRACTOR: END TO END TESTING**

Use protractor to write and run end-to-end (e2e) tests. End-to-end tests explore the application as users experience it. In e2e testing, one process runs the real application and a second process runs protractor tests that simulate user behavior and assert that the application respond in the browser as expected.

**Testing stub**

Test stubs are programs that simulate the behaviors of software components (or modules) that a module undergoing tests depends on.

**MONGO DB:**

MongoDB stores data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time

The document model maps to the objects in your application code, making data easy to work with

Ad hoc queries, indexing, and real time aggregation provide powerful ways to access and analyze your data

MongoDB is a distributed database at its core, so high availability, horizontal scaling, and geographic distribution are built in and easy to use

**DEBUG**

Firstly we have to enable production mode in main.ts and do the debug in the browser console.

**Life cycle of angular**

The three phases of the life cycle of an AngularJS application happen each time a web page is loaded in the browser. The following sections describe these phases of an AngularJS application.

**The Bootstrap Phase**

The first phase of the AngularJS life cycle is the bootstrap phase, which occurs when the AngularJS JavaScript library is downloaded to the browser. AngularJS initializes its own necessary components and then initializes your module, which the ng-app directive points to. The module is loaded, and any dependencies are injected into your module and made available to code within the module.

**The Compilation Phase**

The second phase of the AngularJS life cycle is the HTML compilation stage. Initially when a web page is loaded, a static form of the DOM is loaded in the browser. During the compilation phase, the static DOM is replaced with a dynamic DOM that represents the AngularJS view.

This phase involves two parts: traversing the static DOM and collecting all the directives and then linking the directives to the appropriate JavaScript functionality in the AngularJS built-in library or custom directive code. The directives are combined with a scope to produce the dynamic or live view.exception

**The Runtime Data Binding Phase**

The final phase of the AngularJS application is the runtime phase, which exists until the user reloads or navigates away from a web page. At that point, any changes in the scope are reflected in the view, and any changes in the view are directly updated in the scope, making the scope the single source of data for the view.

AngularJS behaves differently from traditional methods of binding data. Traditional methods combine a template with data received from the engine and then manipulate the DOM each time the data changes. AngularJS compiles the DOM only once and then links the compiled template as necessary, making it much more efficient than traditional methods.

**When will the angular application closed?**

By closing command terminal.

**EXT.JS**

What is **Ext JS**

Ext JS is a popular JavaScript framework which provide rich UI for building web applications with cross browser functionality.

Ext JS is based on MVC/ MVVM architecture. The latest version of Ext JS 6 is a single platform which can be used for both desktop and mobile application without having different code for different platform.

MVC – Model View Controller architecture (version 4)

MVVM – Model View Viewmodel (version 5)

**ViewModel:** It is basically medicates the changes between view and model. It binds the data from model to view. At the same time it does not have any direct interaction with view it has only knowledge of model.

**index.htm**

<!DOCTYPE html>

<html>

<head>

<link href="https://cdnjs.cloudflare.com/ajax/libs/extjs/6.0.0/classic/theme-classic/resources/theme-classic-all.css" rel="stylesheet" />

<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/extjs/6.0.0/ext-all.js"></script>

<script type="text/javascript">

Ext.onReady(function() {

Ext.create('Ext.Panel', {

renderTo: 'helloWorldPanel',

height: 200,

width: 600,

title: 'Hello world',

html: 'First Ext JS Hello World Program'

});

});

</script>

</head>

<body>

<div id="helloWorldPanel" />

</body>

</html>

**EXPLANATION**

* Ext.onReady() method will be called once the Ext JS is ready to render the Ext JS elements.
* Ext.create() method is used to create object in Ext JS here we are creating an object of simple panel class Ext.Panel.
* Ext.Panel is the predefined class in Ext JS for creating a panel.
* Every Ext JS class has different properties to perform some basic functionalities.

Ext.Panel class has various properties as:

* **renderTo** is the element where this panel has to be render. 'helloWorldPanel' is the div id in Index.html file.
* **Height** and width properties are for giving custom size of the panel.
* **Title** property is to provide the title to the panel.
* **Html** property is the html content to be shown in the panel.

Ext JS is a JavaScript framework which has functionalities of object oriented programming.

Ext.define() is used for defining classes in Ext JS.

**Syntax:**

Ext.define(class name, class members/properties, callback function);

Different ways of creating objects in Ext JS-

Using new keyword:

var studentObject = new student();

studentObject.getStudentName();

Using Ext.create():

Ext.create('Ext.Panel', {

renderTo : 'helloWorldPanel',

height : 100,

width : 100,

title : 'Hello world',

html : 'First Ext JS Hello World Program'

});

**Inheritance in Ext JS**

Inheritance is the principle of using functionality defined in class A into class B.

In Ext JS inheritance can be done using two methods-

Ext.extend:

Ext.define(studentApp.view.StudentDetailsGrid, {

extend : 'Ext.grid.GridPanel',

...

});

Using Mixins:

Mixins is the different way of using class A in class B without extend.

mixins : {

commons : 'DepartmentApp.utils.DepartmentUtils'

},

Mixins we add in controller where we declare all the other classes such as store, view etc. In this way we can call Department Utils class and use its functions in controller or in this application.

## **Containers**

**Container** in Ext JS is the component where we can add other container or child components.

**Ext.container.Container** is the base class for all the containers in Ext JS.

There are various type of containers Ext.panel.Panel, Ext.form.Panel, Ext.tab.Panel and Ext.container.Viewport are frequently used containers in Ext JS.

**Layout** is the way the elements are arranged in a container. That could be horizontal, vertical or any other.

ExtJS UI is made up of one or many widgets called **Components.Ext Js** has various UI components defined which can be customised as per your requirements.

# **Ext.js - Data**

Data package is used for loading and saving all the data in the application.

The data package has numerous number of classes but the most important classes are:

1. Modal
2. Store
3. Proxy

## **Model:**

The base class for modal is Ext.data.Model.It represents an entity in an application. It binds the store data to view. It has mapping of backend data objects to the view dataIndex. The data is fetched with the help of store.

### **Creating a Model:**

For creating a model we need to extend Ext.data.Model class and we need to define fields their name and mapping.

Ext.define('StudentDataModel', {

extend: 'Ext.data.Model',

fields: [

{name: 'name', mapping : 'name'},

{name: 'age', mapping : 'age'},

{name: 'marks', mapping : 'marks'}

]

});

## **Store:**

The base class for store is Ext.data.Store. It contains the data locally cached which is to be rendered on view with the help of model objects. Store fetches the data using proxies which has the path defined for services to fetch the backend data.

Store data can be fetched from two ways static or dynamic.

### **Static store:**

For static store we will have all the data present in the store as below:

Ext.create('Ext.data.Store', {

model: 'StudentDataModel',

data: [

{ name : "Asha", age : "16", marks : "90" },

{ name : "Vinit", age : "18", marks : "95" },

{ name : "Anand", age : "20", marks : "68" },

{ name : "Niharika", age : "21", marks : "86" },

{ name : "Manali", age : "22", marks : "57" }

];

});

### **Dynamic store:**

The dynamic data can be fetched using proxy. we can have proxy can fetched data from Ajax, Rest and Json.

## **Proxy:**

The base class for proxy is Ext.data.proxy.Proxy. Proxy is used by Models and Stores to handle the loading and saving of Model data.

There are two types of proxies:

1. Client Proxy
2. Server Proxy

### **Client Proxy**

Client proxies include Memory and Local Storage using HTML5 local storage.

### **Server Proxy**

Server proxies handles data from remote server using Ajax, Json data and Rest service.

#### DEFINING PROXIES IN THE SERVER:

Ext.create('Ext.data.Store', {

model: 'StudentDataModel',

proxy : {

type : 'rest',

actionMethods : {

read : 'POST' // Get or Post type based on requirement

},

url : 'restUrlPathOrJsonFilePath', // here we have to include the rest URL path which fetches data from database or Json file path where the data is stored

reader: {

type : 'json', // the type of data which is fetched is of JSON type

root : 'data'

},

}

});

**Java script**

**DOM**

The Document Object Model (DOM) It defines the logical structure of documents and the way a document is accessed and manipulated.

is an application programming interface ([API](https://www.w3.org/TR/DOM-Level-2-Core/glossary.html#dt-API)) for valid [HTML](https://www.w3.org/TR/DOM-Level-2-Core/glossary.html#dt-HTML) and well-formed [XML](https://www.w3.org/TR/DOM-Level-2-Core/glossary.html#dt-XML) documents.

Root element🡪 <html>

**What are disadvantages of using JavaScript?**

* Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
* JavaScript doesn't have any multithreading or multiprocess capabilities.

**What are the valid scopes of a variable in JavaScript?**

The scope of a variable is the region of your program in which it is defined. JavaScript variable will have only two scopes.

* **Global Variables −** A global variable has global scope which means it is visible everywhere in your JavaScript code.
* **Local Variables −** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

**Closure:**

A **closure** is an inner function that has access to the outer (enclosing) function's variables.

**Timers:**

Timers are used to execute a piece of code at a set time or also to repeat the code in a given interval of time. This is done by using the functionangulajss setTimeout, setInterval and clearInterval.

The **setTimeout(function, delay)** function is used to start a timer that calls a particular function after the mentioned delay.

The **setInterval(function, delay)** function is used to repeatedly execute the given function in the mentioned delay and only halts when cancelled. The clearInterval(id) function instructs the timer to stop.

Timers are operated within a single thread, and thus events might queue up, waiting to be executed.

**What is the difference between ViewState and SessionState?**

‘ViewState’ is specific to a page in a session.

‘SessionState’ is specific to user specific data that can be accessed across all pages in the web application.

**What is called Variable typing in Javascript?**

Variable typing is used to assign a number to a variable and the same variable can be assigned to a string.

I=10;

I = “string”;

**How can you convert the string of any base to integer in JavaScript?**

The parseInt() function is used to convert numbers between different bases. parseInt() takes the string to be converted as its first parameter, and the second parameter is the base of the given string.

**Explain the difference between “==” and “===”?**

“==” checks only for equality in value whereas “===” is a stricter equality test and returns false if either the value or the type of the two variables are different.

**What is the difference between an alert box and a confirmation box?**

An alert box displays only one button which is the OK button.

But a Confirmation box displays two buttons namely OK and cancel.

**What is the ‘Strict’ mode in JavaScript and how can it be enabled?**

Strict Mode adds certain compulsions to JavaScript. Under the strict mode, JavaScript shows errors for a piece of codes, which did not show an error before, but might be problematic and potentially unsafe. Strict mode also solves some mistakes that hamper the JavaScript engines to work efficiently.

Strict mode can be enabled by adding the string literal “use strict” above the file

**Explain window.onload and onDocumentReady?**

The onload function is not run until all the information on the page is loaded. This leads to a substantial delay before any code is executed.

onDocumentReady loads the code just after the DOM is loaded. This allows early manipulation of the code.