1. How constructor can be used for a servlet?

Answer :- Initialization and Constructor function. We cannot declare constructors for interface in Java. This means we cannot enforce this requirement to any class which implements Servlet interface.  
Also, Servlet requires ServletConfig object for initialization which is created by container.

1. Can servlet class declare constructor with ServletConfig object as an argument?

Answer: NO ServletConfig object is created after the constructor is called and before init() is called. So, servlet init parameters cannot be accessed in the constructor.

1. Which of the following code is used to get an attribute in a HTTP Session object in servlets?

Answer: session.getAttribute(String name)

1. What is servlet?

Answer : Servlets execute within the address space of web server, platform independent and uses the functionality of java class libraries

1. What are different Session tracking techniques?

Answer: URL rewriting , Using session object , Using hidden fields , Using cookies

1. How many objects of a servlet is created?

Only one object at the time of first request by servlet or web container.

1. What is the life-cycle of a servlet?
2. Servlet is loaded
3. servlet is instantiated
4. servlet is initialized
5. service the request
6. servlet is destroyed

# Life Cycle of a Servlet (Servlet Life Cycle)

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.



1. What is difference between Get and Post method?

|  |  |
| --- | --- |
| **Get** | **Post** |
| 1) Limited amount of data can be sent because data is sent in header. | Large amount of data can be sent because data is sent in body. |
| 2) Not Secured because data is exposed in URL bar. | Secured because data is not exposed in URL bar. |
| 3) Can be bookmarked | Cannot be bookmarked |
| 4) Idempotent | Non-Idempotent |
| 5) It is more efficient and used than Post | It is less efficient and used |

### What is difference between PrintWriter and ServletOutputStream?

PrintWriter is a character-stream class where as ServletOutputStream is a byte-stream class. The PrintWriter class can be used to write only character-based information whereas ServletOutputStream class can be used to write primitive values as well as character-based information.

### What is difference between GenericServlet and HttpServlet?

The GenericServlet is protocol independent whereas HttpServlet is HTTP protocol specific. HttpServlet provides additional functionalities such as state management etc.

1. What is servlet collaboration?

When one servlet communicates to another servlet, it is known as servlet collaboration. There are many ways of servlet collaboration:

* RequestDispacher interface
* sendRedirect() method etc.

### What is the purpose of RequestDispatcher Interface?

The RequestDispacher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interceptor can also be used to include the content of antoher resource.

1. Can you call a jsp from the servlet?

Yes, one of the way is RequestDispatcher interface for example:

1. RequestDispatcher rd=request.getRequestDispatcher("/login.jsp");
2. rd.forward(request,response);
3. Difference between forward() method and sendRedirect() method ?

|  |  |
| --- | --- |
| **forward() method** | **sendRedirect() method** |
| 1) forward() sends the same request to another resource. | 1) sendRedirect() method sends new request always because it uses the URL bar of the browser. |
| 2) forward() method works at server side. | 2) sendRedirect() method works at client side. |
| 3) forward() method works within the server only. | 3) sendRedirect() method works within and outside the server. |

### How can we perform any action at the time of deploying the project?

By the help of ServletContextListener interface.

1. What are the annotations used in Servlet 3?

There are mainly 3 annotations used for the servlet.

1. @WebServlet : for servlet class.
2. @WebListener : for listener class.
3. @WebFilter : for filter class.

### What is the use of welcome-file-list?

It is used to specify the welcome file for the project.

### What is the use of attribute in servlets?

Attribute is a map object that can be used to set, get or remove in request, session or application scope. It is mainly used to share information between one servlet to another.

1. What is the difference between an application server and a Web server?

A Web server exclusively handles HTTP requests, whereas an application server serves business logic to application programs through any number of protocols.

## The Web server

A Web server handles the HTTP protocol. When the Web server receives an HTTP request, it responds with an HTTP response, such as sending back an HTML page. To process a request, a Web server may respond with a static HTML page or image, send a redirect, or delegate the dynamic response generation to some other program such as CGI scripts, JSPs (JavaServer Pages), servlets, ASPs (Active Server Pages), server-side JavaScripts, or some other server-side technology. Whatever their purpose, such server-side programs generate a response, most often in HTML, for viewing in a Web browser.

## The application server

As for the application server, according to our definition, an application server exposes business logic to client applications through various protocols, possibly including HTTP. While a Web server mainly deals with sending HTML for display in a Web browser, an application server provides access to business logic for use by client application programs. The application program can use this logic just as it would call a method on an object (or a function in the procedural world).

Following are the important differences between Web Server and Application Server.

| **Sr. No.** | **Key** | **Web Server** | **Application Server** |
| --- | --- | --- | --- |
| 1 | Purpose | Web Server contains Web container only. | Application Server contains Web Container plus EJB Container. |
| 2 | Useful | A web server is good in case of static contents like static html pages. | Applcation server is relevant in case of dynamic contents like bank websites. |
| 3 | Resource Consumption | Web server consumes less resources like CPU, Memory etc. as compared to application server. | Application server utilizes more resources. |
| 4 | Target Environment | Web Server provides the runtime environment for web applications. | Application server provides the runtime environment for enterprise applications. |
| 5 | Multithreading support | Multithreading is not supported. | Multithreading is supported. |
| 6 | Protocol(s) supported | Web Server supports HTTP Protocol. | Application Server suppots HTTP as well as RPC/RMI protocols. |
| 7 | Example | Apache Web Server. | Weblogic, JBoss. |

1. Difference between Servlet and Spring

Servlet: a server side java class to produce the html content.  
however  
Spring: A framework to develop Big Enterprise Application which include your servlet as well.

# Difference between Servlet and JSP

|  |  |
| --- | --- |
| Servlet | JSP |
| Servlet is a java code. | JSP is a html based code. |
| Writing code for servlet is harder than JSP as it is html in java. | JSP is easy to code as it is java in html. |
| Servlet plays a controller role in MVC approach. | JSP is the view in MVC approach for showing output. |
| Servlet is faster than JSP. | JSP is slower than Servlet because the first step in JSP lifecycle is the translation of JSP to java code and then compile. |
| Servlet can accept all protocol requests. | JSP only accept http requests. |
| In Servlet, we can override the service() method. | In JSP, we cannot override its service() method. |
| In Servlet by default session management is not enabled, user have to enable it explicitly. | In JSP session management is automatically enabled. |
| In Servlet we have to implement everything like business logic and presentation logic in just one servlet file. | In JSP business logic is separated from presentation logic by using javaBeans. |
| Modification in Servlet is a time consuming task because it includes reloading, recompiling and restarting the server. | JSP modification is fast, just need to click the refresh button. |

# Spring vs. Spring Boot vs. Spring MVC

## Spring vs. Spring Boot

**Spring:** Spring Framework is the most popular application development framework of Java. The main feature of the Spring Framework is **dependency Injection** or **Inversion of Control** (IoC). With the help of Spring Framework, we can develop a **loosely** coupled application. It is better to use if application type or characteristics are purely defined.

**Spring Boot:** Spring Boot is a module of Spring Framework. It allows us to build a stand-alone application with minimal or zero configurations. It is better to use if we want to develop a simple Spring-based application or RESTful services.

|  |  |
| --- | --- |
| **Spring** | **Spring Boot** |
| **Spring Framework** is a widely used Java EE framework for building applications. | **Spring Boot Framework** is widely used to develop **REST APIs**. |
| It aims to simplify Java EE development that makes developers more productive. | It aims to shorten the code length and provide the easiest way to develop **Web Applications**. |
| The primary feature of the Spring Framework is **dependency injection**. | The primary feature of Spring Boot is **Autoconfiguration**. It automatically configures the classes based on the requirement. |
| It helps to make things simpler by allowing us to develop **loosely coupled** applications. | It helps to create a **stand-alone** application with less configuration. |
| The developer writes a lot of code (**boilerplate code**) to do the minimal task. | It **reduces** boilerplate code. |
| To test the Spring project, we need to set up the sever explicitly. | Spring Boot offers **embedded server** such as **Jetty** and **Tomcat**, etc. |
| It does not provide support for an in-memory database. | It offers several plugins for working with an embedded and **in-memory** database such as **H2**. |
| Developers manually define dependencies for the Spring project in **pom.xml**. | Spring Boot comes with the concept of **starter** in pom.xml file that internally takes care of downloading the dependencies **JARs** based on Spring Boot Requirement. |

## Spring Boot vs. Spring MVC

**Spring Boot:** Spring Boot makes it easy to quickly bootstrap and start developing a Spring-based application. It avoids a lot of boilerplate code. It hides a lot of complexity behind the scene so that the developer can quickly get started and develop Spring-based applications easily.

**Spring MVC:** Spring MVC is a Web MVC Framework for building web applications. It contains a lot of configuration files for various capabilities. It is an HTTP oriented web application development framework.

|  |  |
| --- | --- |
| **Spring Boot** | **Spring MVC** |
| **Spring Boot** is a module of Spring for packaging the Spring-based application with sensible defaults. | **Spring MVC** is a model view controller-based web framework under the Spring framework. |
| It provides default configurations to build **Spring-powered** framework. | It provides **ready to use** features for building a web application. |
| There is no need to build configuration manually. | It requires build configuration manually. |
| There is **no requirement** for a deployment descriptor. | A Deployment descriptor is **required**. |
| It avoids boilerplate code and wraps dependencies together in a single unit. | It specifies each dependency separately. |
| It **reduces** development time and increases productivity. | It takes **more** time to achieve the same. |

1. Jakarta RESTful Web Services

Jakarta RESTful Web Services, is a Jakarta EE API specification that provides support in creating web services according to the Representational State Transfer architectural pattern. JAX-RS uses annotations, introduced in Java SE 5, to simplify the development and deployment of web service clients and endpoints.

1. What is JSON?

JavaScript Object Notation. It is used in Android , Web APi , Rest APi, everywhere. It is a format in which you send data to server or from server to client.

JSON.stringify turns a JavaScript object into JSON text and stores that JSON text in a string.

JSON.parse turns a string of JSON text into a JavaScript object

JSON.parse() is for "parsing" something that was received as JSON.  
JSON.stringify() is to create a JSON string out of an object/array.

1. ApplicationContext

**ApplicationContext** is a corner stone of a **Spring** Boot application. It represents the **Spring** IoC container and is responsible for instantiating, configuring, and assembling the beans. The container gets its instructions on what objects to instantiate, configure, and assemble by reading configuration metadata.

# Difference Between BeanFactory and ApplicationContext in Spring

### The BeanFactory and ApplicationContext interfaces both are used to get Spring beans from a container.

### What is a Spring Bean?

### Ans:- **Usually, Spring beans are Java objects that are managed by the Spring container.**

### Here is a simple Spring bean example:

### package com.example;

### public class HelloWorld {

### private String message;

### public void setMessage(String message){

### this.message = message;

### }

### public void getMessage(){

### System.out.println("My Message : " + message);

### }

### }

### In the XML-based configuration, **beans.xml** supplies the metadata for the Spring container to manage the bean.

## ****What Is the Spring Container?****

### Ans:- The Spring container is responsible for instantiating, configuring, and assembling the Spring beans. Here is an example of how we configure our HelloWorld POJO for the IoC container:

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

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

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

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

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

### <bean id = "helloWorld" class = "com.zoltanraffai.HelloWorld">

### <property name = "message" value = "Hello World!"/>

### </bean>

### </beans>

Now, it managed by the Spring container. The only question is: how we can access it?

## The Difference Between BeanFactory and ApplicationContext

**The BeanFactory Interface**

This is the root *interface* for accessing the Spring container. To access the Spring container, we will be using Spring's dependency injection functionality using this BeanFactory interface and its sub-interfaces.

Features:

* **Bean instantiation/wiring**

Usually, the implementations **use lazy loading**, which means that beans are only instantiating when we directly calling them through the getBean() method.

The most used API that implements the **BeanFactory** is the **XmlBeanFactory**.

Here is an example of how to get a bean through the BeanFactory:

package com.example;

​import org.springframework.core.io.ClassPathResource;

import org.springframework.beans.factory.InitializingBean;

import org.springframework.beans.factory.xml.XmlBeanFactory;

​public class HelloWorldApp{

public static void main(String[] args) {

XmlBeanFactory factory = new XmlBeanFactory (new ClassPathResource("beans.xml"));

HelloWorld obj = (HelloWorld) factory.getBean("helloWorld");

obj.getMessage();

}

}

**The ApplicationContext Interface**

The **ApplicationContext**is the central interface within a Spring application that is used for providing configuration information to the application.

It implements the BeanFactory interface. Hence, the ApplicationContext includes all functionality of the BeanFactory and much more! Its main function is to support the creation of big business applications.

Features:

* **Bean instantiation/wiring**
* **Automatic BeanPostProcessor registration**
* **Automatic BeanFactoryPostProcessor registration**
* **Convenient MessageSource access (for i18n)**
* **ApplicationEvent publication**

It **uses eager loading**, so every bean instantiate after the ApplicationContext is started up.

Here is an example of the **ApplicationContext** usage:

package com.example;

​import org.springframework.core.io.ClassPathResource;

import org.springframework.beans.factory.InitializingBean;

import org.springframework.beans.factory.xml.XmlBeanFactory;

​

public class HelloWorldApp{

public static void main(String[] args) {

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

HelloWorld obj = (HelloWorld) context.getBean("helloWorld");

obj.getMessage();

}

Summary :- The **ApplicationContext**includes all the functionality of the **BeanFactory.** It is generally recommended to use the former. There are some limited situations, such as in mobile applications, where memory consumption might be critical. In those scenarios, it would be justifiable to use the more lightweight **BeanFactory**. However, in most enterprise applications, the **ApplicationContext** is what you will want to use

## Difference Between REST vs RESTful

### REST denotes a server that shares the JSON files with a client over HTTP. It is a complete version which doesn’t completely dependent on JSON, XML or HTTP. A Restful is an advanced form of web exchange server that shares any other documents or JSON to develop any new applications. In simple, the REST represents the Representational State Transfer which uses an architectural pattern for developing new web services and to implement that architectural pattern RESTful services are implemented. In this article we will see difference between REST vs RESTful.

### REST vs RESTful info

### 

## What is the use of DispatcherServlet in Spring MVC? The DispatcherServlet is one of the important components of the [Spring MVC](https://javarevisited.blogspot.com/2018/11/top-20-spring-mvc-interview-questions-answers-for-java-developers.html) web framework and acts as a Front Controller. The DispatcherServlet is a front controller like it provides a single entry point for a client request to Spring MVC web application and forwards request to Spring MVC controllers for processing. How to configure DispatcherServlet in Spring?

The [DispatcherServlet](http://javarevisited.blogspot.sg/2016/01/solving-javalangclassnotfoundexception-org.springframework.web.servlet.DispatcherServlet.html" \t "_blank) is like any other [Servlet](http://javarevisited.blogspot.sg/2017/05/difference-between-servlet-and-jsp.html" \t "_blank) class and it has to be declared inside the deployment descriptor or web.xml file as shown below:

<servlet>

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

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<init-param>

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

<!--Defaults to WEB-INF\dispatcher-servlet.xml -->

<param-value>classpath:mvc-config.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

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

<url-pattern>/\*</url-pattern>

</servlet-mapping>

## How Dispatcher Servlet works Internally in Spring?

### DispatcherServlet wears many hats in Spring. It acts as a front controller and provides a single entry point for the application. It then uses handler mappings and handler adapters to map a request to the [Spring MVC controllers](http://javarevisited.blogspot.sg/2012/05/what-is-bean-scope-in-spring-mvc.html#axzz4jWEJmi6S). It uses @Controller and @RequestMapping annotation for that purpose. Once the request is processed by the Spring MVC controller, it returns a logical view name instead of the view. Though, you can even configure Controler's handler methods to not return any View name by declaring return type as void. You can even use @ResponseBody annotation in the case of REST to directly write the output to the HTTP response body. When [DispatherServlet](http://javarevisited.blogspot.sg/2016/01/solving-javalangclassnotfoundexception-org.springframework.web.servlet.DispatcherServlet.html) receives a view name, it consults the ViewResolver to find the right view. There is a chain of ViewResolver that is maintained at the Spring MVC framework. They try to resolve the logical view name into a Physical resource like a JSP page or a FreeMaker or Velocity template. The ViewResolver is invoked in order, if first in the chain not able to resolve the view then it returns null and next ViewResolver in the chain is consults. Once the right view is found, DispatcherServlet forwards the request along with Model data to the View for rendering like a [JSP page](http://javarevisited.blogspot.sg/2017/01/best-books-to-learn-servlet-and-jsp.html). When [DispatherServlet](http://javarevisited.blogspot.sg/2016/01/solving-javalangclassnotfoundexception-org.springframework.web.servlet.DispatcherServlet.html) receives a view name, it consults the ViewResolver to find the right view. There is a chain of ViewResolver that is maintained at the Spring MVC framework. They try to resolve the logical view name into a Physical resource like a JSP page or a FreeMaker or Velocity template. The ViewResolver is invoked in order, if first in the chain not able to resolve the view then it returns null and next ViewResolver in the chain is consults. Once the right view is found, DispatcherServlet forwards the request along with Model data to the View for rendering like a [JSP page](http://javarevisited.blogspot.sg/2017/01/best-books-to-learn-servlet-and-jsp.html). What is the use of DispatcherServlet in Spring MVC? Interview Question

In short, **DispatcherServlet is used for following things in Spring MVC**:

1. Receives all request as Front Controller  and provides a single entry point to the application
2. Mapping requests to correct [Spring MVC](https://javarevisited.blogspot.sg/2013/07/role-based-access-control-using-spring-security-ldap-authorities-mapping-mvc.html) controller
3. Consulting ViewResolvers to find correct View
4. Forwarding request to chosen View for rendering
5. Returning the response to the client
6. Creates web-context to initialize the web-specific beans like [controllers](http://javarevisited.blogspot.sg/2011/09/spring-interview-questions-answers-j2ee.html), view resolvers and handler mapping

### How Spring MVC works internally?

### It all starts with the client, which sends a request to a specific URL. When that request hits the web container like Tomcat it looks into web.xml and finds the Servlet or Filter which is mapped to that particular URL. It the delegate that Servlet or Filter to process the request. Since Spring MVC is built on top of Servlet, this is also the initial flow of request in any Spring MVC based Java web application. Web container like Tomcat is responsible for creating Servlet and Filter instances and invoking their various life-cycle methods like [init()](http://javarevisited.blogspot.sg/2015/02/constructor-vs-init-method-in-servlet.html), service(), destroy(). In the case of an HTTP request, HttpServlet handles that, and depending upon the HTTP request method various doXXX() method is invoked by container like doGet() to process GET request and doPost() to process POST request. Difference between @RestController and @Controller?

### The job of @Controller is to create a Map of the model object and find a view but @RestController simply returns the object and object data is directly written into HTTP response as JSON or XML.

### Difference between @Autowired and @Injection annotations in Spring? The @Autowired annotation is used for auto-wiring in Spring framework. If you don't know, autowiring is a process on which Spring framework figure out dependencies of a [Spring bean](http://javarevisited.blogspot.sg/2012/05/what-is-bean-scope-in-spring-mvc.html), instead of you, a developer, explicitly specifying them in the application context file. You can annotate fields and constructor using @Autowired to tell Spring framework to find dependencies for you. The @Inject annotation also serves the same purpose, but the main difference between them is that @Inject is a standard annotation for dependency injection and @Autowired is spring specific. What is Spring MVC?

A Spring MVC is a Java Framework which is used to develop dynamic web applications. It implements all the basic features of a core spring framework like Inversion of Control and Dependency Injection. It follows the Model-View-Controller design pattern.

### Spring MVC Tutorial

* **Model** - A model contains the data of the application. Data can be a single object or a collection of objects.
* **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
* **View** - A view represents the provided information in a particular format. So, we can create a view page by using view technologies like JSP+JSTL, Apache Velocity, Thymeleaf, and FreeMarker.

Explain the flow of Spring MVC?



* Once the request has been generated, it intercepted by the DispatcherServlet that works as the front controller.
* The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller.
* The controller returns an object of ModelAndView.
* The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.

What does an additional configuration file contain in Spring MVC application?

### The Spring MVC application contains an additional configuration file that contains the properties information. This file can be created either in the form of **an xml** file or **properties** file. In this file, we generally define the base-package and view resolver where **DispatcherServlet** searches for the controller classes and view components path. However, it can also contain various other configuration properties.

 What is an InternalResourceViewResolver in Spring MVC?

The **InternalResourceViewResolver** is a class which is used to resolve internal view in Spring MVC. Here, you can define the properties like prefix and suffix where prefix contains the location of view page and suffix contains the extension of view page. For example:-

**<bean** id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>**

**<property** name="prefix" value="/WEB-INF/jsp/"**></property>**

**<property** name="suffix" value=".jsp"**></property>**

**</bean>**

How to declare a class as a controller class in Spring MVC?

The @Controller annotation is used to declare a class as a controller class. It is required to specify this annotation on the class name

How to map controller class and its methods with URL?

The **@RequestMapping** annotation is used to map the controller class and its methods. You can specify this annotation on the class name as well as method name with a particular URL that represents the path of the requested page. For example:-

@Controller

@RequestMapping("/ form")

class Demo

{

@RequestMapping("/show")

public String display()

{

}

}

Name the annotations used to handle different types of incoming HTTP request methods?

The following annotations are used to handle different types of incoming HTTP request methods: -

* @GetMapping
* @PostMapping
* @PutMapping
* @PatchMapping
* @DeleteMapping

What is the purpose of @PathVariable annotation in Spring MVC?

The @PathVariable annotation is used to extract the value of the URI template. It is passed within the parameters of the handler method.

For example :-

@RequestMapping("/show/{id}")

public String handler(@PathVariable("id") String s, Model map)

  {

}

 What is the role of @ResponseBody annotation in Spring MVC?

The @ResponseBody annotation is used to serialize the returned object automatically in JSON and bind it with the Http response body. Here, it not required to invoke the model.

For example :-

@RequestMapping("/show")

    @ResponseBody

    public ResponseHandler display(

      @RequestBody ShowForm form) {

        return new ResponseHandler("display form");

     }

}

What is the role of the Model interface in Spring MVC?

The **Model** interface works as a container that contains the data of the application. Here, data can be in any form such as objects, strings, information from the database, etc.

What do you mean by ModelAndView in Spring MVC?

The **ModelAndView** is a class that holds both Model and View where the model represents the data, and view represents the representation of that data. This class returns the model and view in the single return value.

What is ModelMap in Spring MVC?

The **ModelMap** is a class that provides the implementation of Map. It extends the LinkedHashMap class. It facilitates to pass a collection of values as if they were within a Map.

What are the ways of reading data from the form in Spring MVC?

The following ways to read the data from the form are: -

* **HttpServletRequest interface** - The **HttpServletRequest** is a java interface present in javax.servlet.http package. Like Servlets, you can use HttpServletRequest in Spring to read the HTML form data provided by the user.
* **@RequestParam annotation** - The **@RequestParam** annotation reads the form data and binds it automatically to the parameter present in the provided method.
* **@ModelAttribute annotation** - The **@ModelAttribute** annotation binds a method parameter or its return value to a named model attribute.

What is Spring MVC form tag library?

* The Spring MVC form tags can be seen as data binding-aware tags that can automatically set data to Java object/bean and also retrieve from it. These tags are the configurable and reusable building blocks for a web page. It provides view technologies, an easy way to develop, read, and maintain the data.

What is the use of @Valid annotation in Spring MVC?

* The **@Valid** annotation is used to apply validation rules on the provided object.

What will be the initial value of an object reference which is defined as an instance variable?

* All object references are initialized to null in Java.

Does constructor return any value?

**Ans:** yes, The constructor implicitly returns the current instance of the class (You can't use an explicit return type with the constructor)

Is constructor inherited?

No, The constructor is not inherited.

What do you understand by copy constructor in Java?

There is no copy constructor in java. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in java. They are:

* By constructor
* By assigning the values of one object into another
* By clone() method of Object class

What is the static variable?

* The static variable is used to refer to the common property of all objects (that is not unique for each object), e.g., The company name of employees, college name of students, etc. Static variable gets memory only once in the class area at the time of class loading. Using a static variable makes your program more memory efficient (it saves memory). Static variable belongs to the class rather than the object.

1. //Program of static variable
3. **class** Student8{
4. **int** rollno;
5. String name;
6. **static** String college ="ITS";
8. Student8(**int** r,String n){
9. rollno = r;
10. name = n;
11. }
12. **void** display (){System.out.println(rollno+" "+name+" "+college);}
14. **public** **static** **void** main(String args[]){
15. Student8 s1 = **new** Student8(111,"Karan");
16. Student8 s2 = **new** Student8(222,"Aryan");
18. s1.display();
19. s2.display();
20. }
21. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=Student8)

Output:111 Karan ITS

222 Aryan ITS

### Static Variable

 Why is the main method static?

Because the object is not required to call the static method. If we make the main method non-static, JVM will have to create its object first and then call main() method which will lead to the extra memory allocation.

Can we override the static methods?

No, we can't override static methods.

What is the static block?

Static block is used to initialize the static data member. It is executed before the main method, at the time of classloading.

1. **class** A2{
2. **static**{System.out.println("static block is invoked");}
3. **public** **static** **void** main(String args[]){
4. System.out.println("Hello main");
5. }
6. }
7. Output: static block is invoked
8. Hello main

Can we execute a program without main() method?

Ans) No, It was possible before JDK 1.7 using the static block. Since JDK 1.7, it is not possible.

What if the static modifier is removed from the signature of the main method?

Program compiles. However, at runtime, It throws an error "NoSuchMethodError."

What is aggregation?

Aggregation can be defined as the relationship between two classes where the aggregate class contains a reference to the class it owns. Aggregation is best described as a **has-a** relationship. For example, The aggregate class Employee having various fields such as age, name, and salary also contains an object of Address class having various fields such as Address-Line 1, City, State, and pin-code. In other words, we can say that Employee (class) has an object of Address class. Consider the following example.

**Address.java**

1. **public** **class** Address {
2. String city,state,country;
4. **public** Address(String city, String state, String country) {
5. **this**.city = city;
6. **this**.state = state;
7. **this**.country = country;
8. }
10. }

**Employee.java**

1. **public** **class** Emp {
2. **int** id;
3. String name;
4. Address address;
6. **public** Emp(**int** id, String name,Address address) {
7. **this**.id = id;
8. **this**.name = name;
9. **this**.address=address;
10. }
12. **void** display(){
13. System.out.println(id+" "+name);
14. System.out.println(address.city+" "+address.state+" "+address.country);
15. }
17. **public** **static** **void** main(String[] args) {
18. Address address1=**new** Address("gzb","UP","india");
19. Address address2=**new** Address("gno","UP","india");
21. Emp e=**new** Emp(111,"varun",address1);
22. Emp e2=**new** Emp(112,"arun",address2);
24. e.display();
25. e2.display();
27. }
28. }

**Output**

111 varun

gzb UP india

112 arun

gno UP india

What is the difference between aggregation and composition?

Aggregation represents the weak relationship whereas composition represents the strong relationship. For example, the bike has an indicator (aggregation), but the bike has an engine (composition).

Can you use this() and super() both in a constructor?

No, because this() and super() must be the first statement in the class constructor.

What is object cloning?

The object cloning is used to create the exact copy of an object. The clone() method of the Object class is used to clone an object. The **java.lang.Cloneable** interface must be implemented by the class whose object clone we want to create. If we don't implement Cloneable interface, clone() method generates CloneNotSupportedException.

**protected** Object clone() **throws** CloneNotSupportedException

What is method overloading with type promotion?

By Type promotion is method overloading, we mean that one data type can be promoted to another implicitly if no exact matching is found.

### Java Method Overloading with Type Promotion

As displayed in the above diagram, the byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int, long, float or double. The char datatype can be promoted to int, long, float or double and so on. Consider the following example.

1. **class** OverloadingCalculation1{
2. **void** sum(**int** a,**long** b){System.out.println(a+b);}
3. **void** sum(**int** a,**int** b,**int** c){System.out.println(a+b+c);}
5. **public** **static** **void** main(String args[]){
6. OverloadingCalculation1 obj=**new** OverloadingCalculation1();
7. obj.sum(20,20);//now second int literal will be promoted to long
8. obj.sum(20,20,20);
9. }
10. }

**Output**

40

60

What is covariant return type?

Now, since java5, it is possible to override any method by changing the return type if the return type of the subclass overriding method is subclass type. It is known as covariant return type. The covariant return type specifies that the return type may vary in the same direction as the subclass.

1. **class** A{
2. A get(){**return** **this**;}
3. }
5. **class** B1 **extends** A{
6. B1 get(){**return** **this**;}
7. **void** message(){System.out.println("welcome to covariant return type");}
9. **public** **static** **void** main(String args[]){
10. **new** B1().get().message();
11. }
12. }

Output: welcome to covariant return type

What is the final variable?

In Java, the final variable is used to restrict the user from updating it. If we initialize the final variable, we can't change its value. In other words, we can say that the final variable once assigned to a value, can never be changed after that. The final variable which is not assigned to any value can only be assigned through the class constructor.

 What is the final method?

If we change any method to a final method, we can't override it.

What is the final class?

If we make any class final, we can't inherit it into any of the subclasses.

What is the final blank variable?

A final variable, not initialized at the time of declaration, is known as the final blank variable. We can't initialize the final blank variable directly. Instead, we have to initialize it by using the class constructor. It is useful in the case when the user has some data which must not be changed by others, for example, PAN Number.

Can we initialize the final blank variable?

Yes, if it is not static, we can initialize it in the constructor. If it is static blank final variable, it can be initialized only in the static block.

Can we declare a constructor as final?

The constructor can never be declared as final because it is never inherited. Constructors are not ordinary methods; therefore, there is no sense to declare constructors as final. However, if you try to do so, The compiler will throw an error.

What is the difference between the final method and abstract method?

The main difference between the final method and abstract method is that the abstract method cannot be final as we need to override them in the subclass to give its definition.