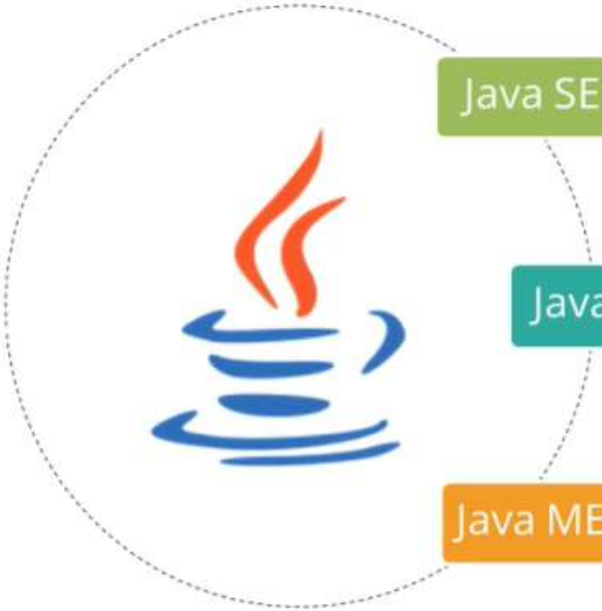
Press Esc to exit full screen

Advanced Java

Introduction to Java EE

00:03 / 20:32

Java Editions



Java SE stands for Java's Standard Edition

Java EE stands for Enterprise Edition

Java ME stands for Micro Edition

©Simplilearn. All rights reserved.

Java EE vs. Java SE

The Java EE (Enterprise Edition) differs from Java SE (Java Standard Edition Platform) in terms of the libraries it provides to deploy:

- fault-tolerant,
- distributed, and
- multi-tier Java software, based largely on modular components running on an application server.

©Simplilearn. All rights reserved.

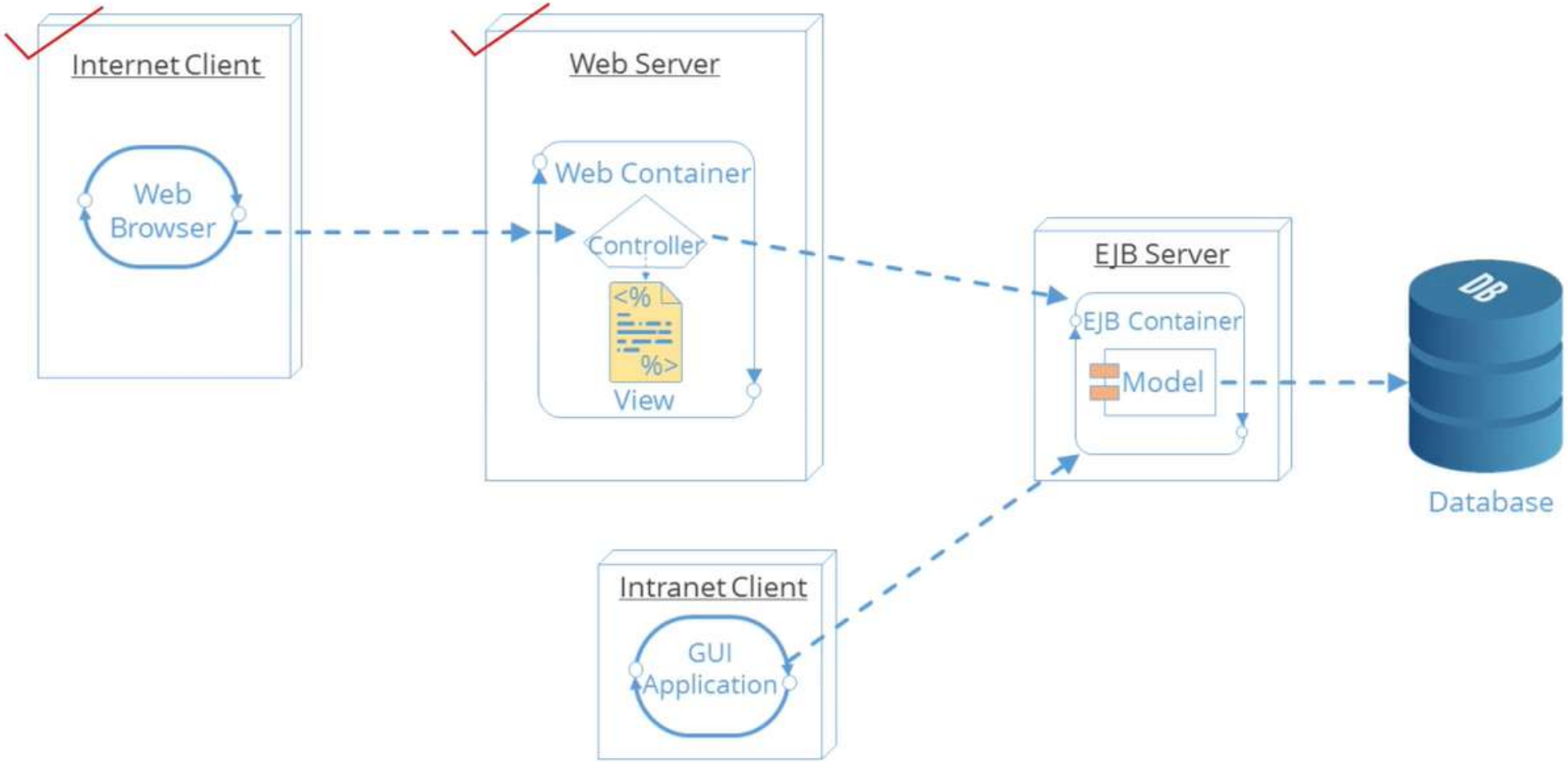
Java EE Libraries and APIs

- Built on top of Java SE, Java EE provides additional libraries for
  - database access [JDBC (Java Database Connectivity), JPA (Java Persistence API)]
  - remote method invocation (RMI)
  - messaging [JMS (Java Message Service)]
  - web services
  - XML (eXtensible Markup Language) processing
- It also defines standard APIs for Enterprise JavaBeans, Servlets, Portlets, and Java Server Pages (JSP)

The installation guide containing the steps to download and configure Java JDK (standard edition) and Apache Tomcat (to access additional libraries) can be downloaded from your LMS.

©Simplilearn. All rights reserved.

Java EE Architecture



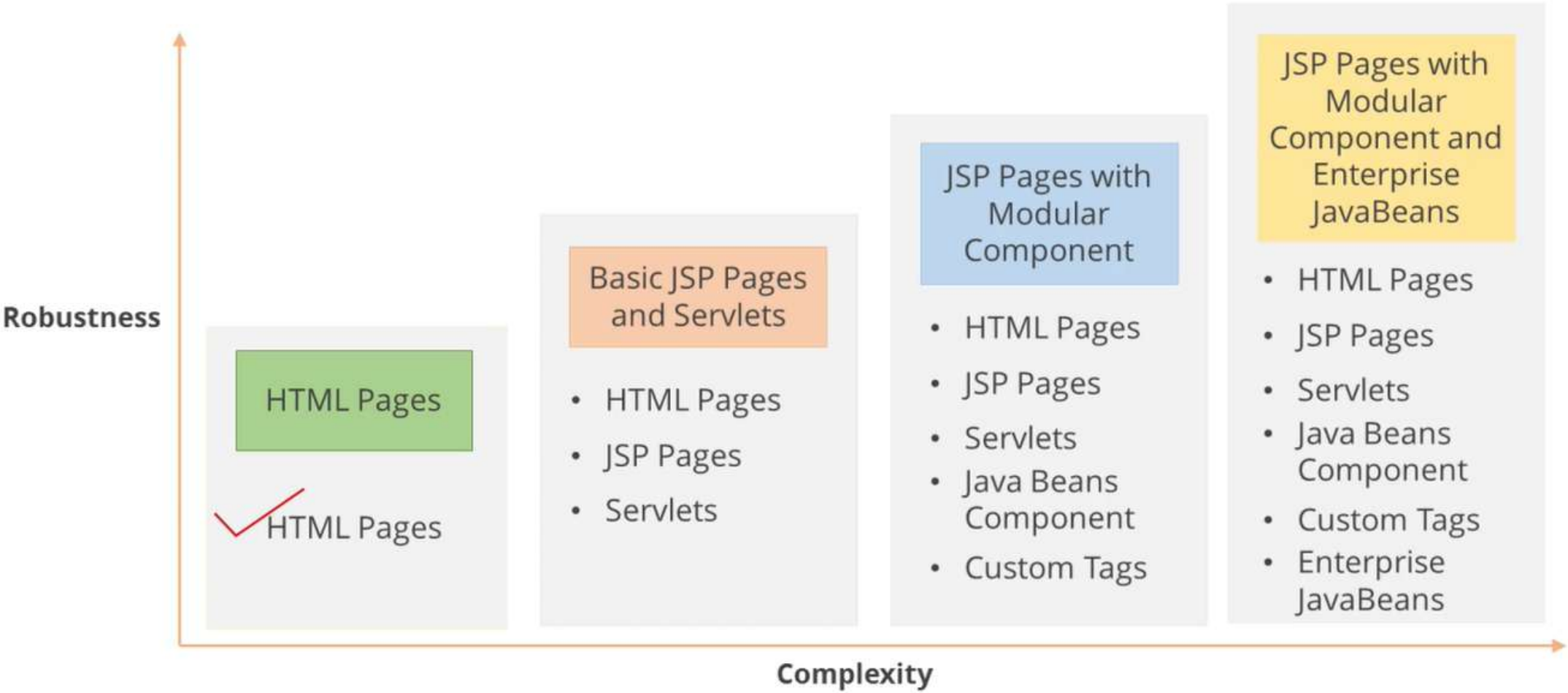
©Simplilearn. All rights reserved.

Java EE Architecture: Features

- Java EE platform facilitates an architecture in which the business components are placed in a separate tier.
- The Java EE architecture enhances features such as scalability, extensibility, and maintainability.
- Its modular design allows easy modification of the business logic. It provides more security because it partitions business services from the web tier. It also permits clean job separation of job roles.
- The enterprise components can leverage their EJB container for service, such as component and resource management, security, persistence, and transactions.

©Simplilearn. All rights reserved.

Relating Java EE Architecture Complexity and Robustness



©Simplilearn. All rights reserved.

simplilearn

Press Esc to exit full screen

Introduction to Java EE

Topic 2—Distributed Applications and Client-Server Model

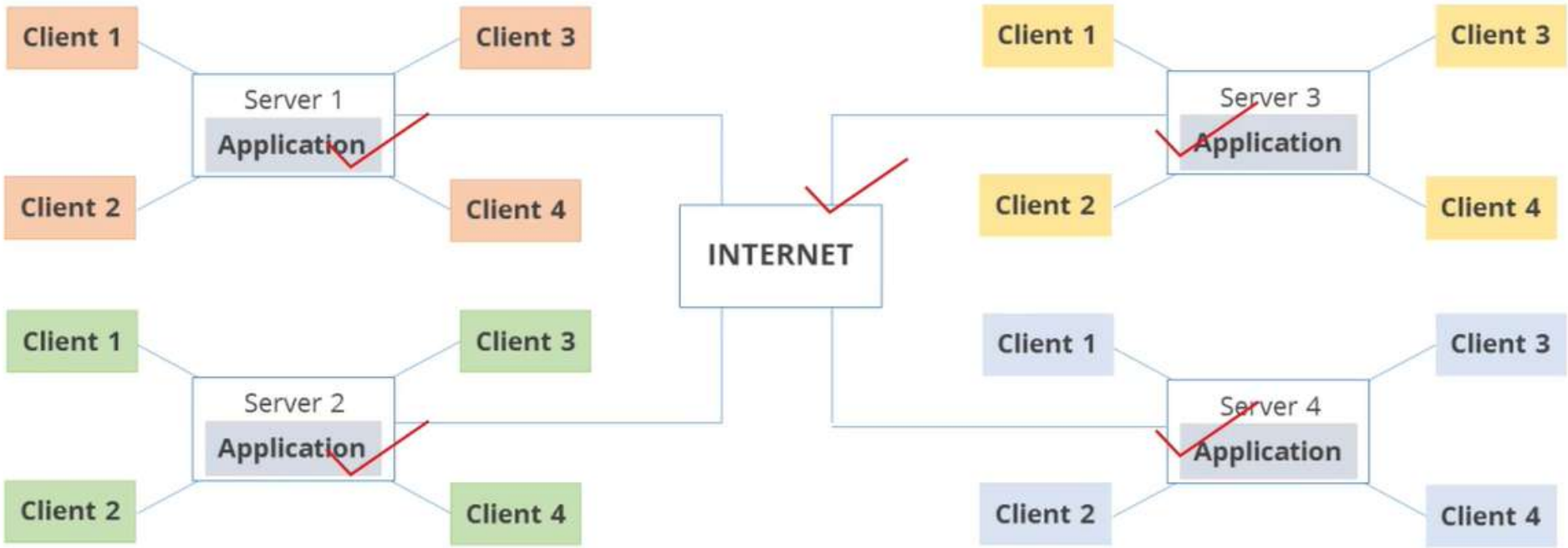
- Distributed Applications
- Client Server Model
- HTTP Protocol
- Types of Server Response
- Client Request Types
- HTML
- Running HTML code with Apache

©Simplilearn. All rights reserved.

simplilearn

Distributed Applications

- In distributed applications, same application can run across different machines at the same time using internet.
- One machine acts as Server where application is deployed and the other machines act as Client.



©Simplilearn. All rights reserved.

Client-Server Model

DEFINITION

The client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service (web servers) and service requesters (web clients).

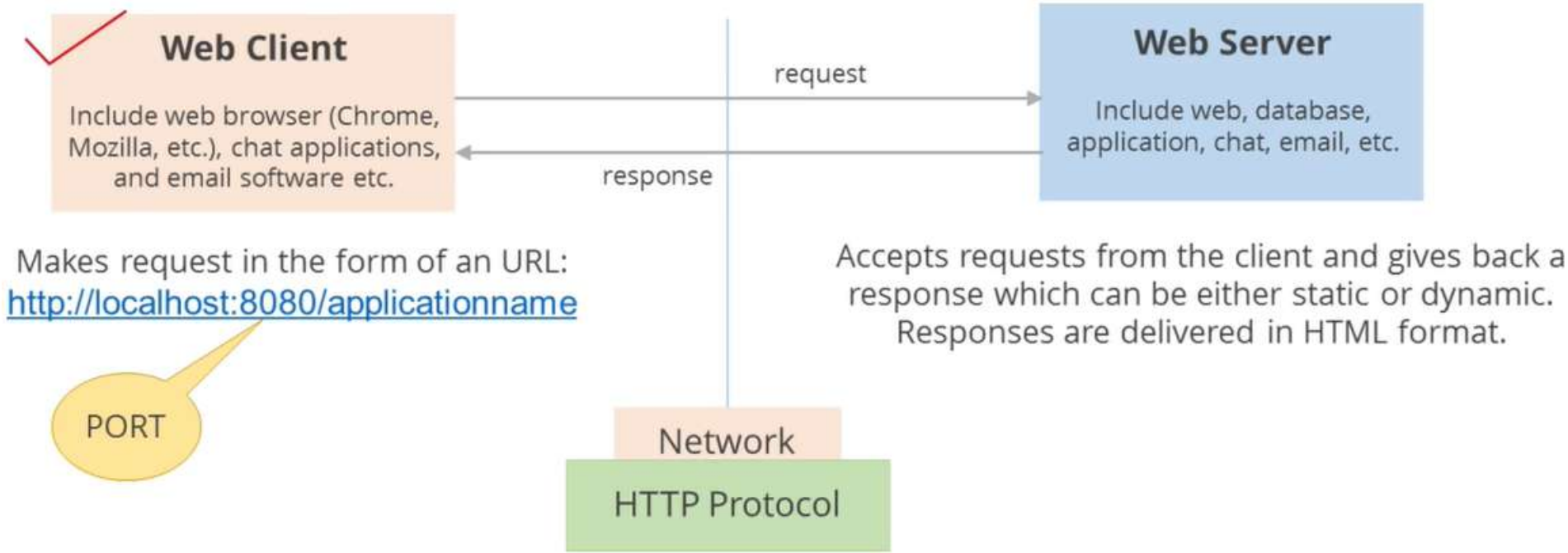
Both web client and web server are programs.

A web client is not a user, but a program. Someone who uses this program to send request to web server is known as "end user."

©Simplilearn. All rights reserved.

Client-Server Model

ARCHITECTURE



HTTP (Hypertext Transfer Protocol) is a TCP/IP based communication protocol, which is used to deliver data over the web.

It provides a standardized way for the client to communicate with the server.

©Simplilearn. All rights reserved.

Client-Server Model

SIGNIFICANCE OF URL AND PORT

Uniform Resource Locator (URL) uniquely identifies a resource on the internet and follows HTTP protocol.

The machine on which the application is hosted/deployed

http://localhost:8080/MyApplication

Server port number on which the server is functional on the machine

Name of the application/context root

Tomcat server runs on port 8080. Remember to check during installation.

©Simplilearn. All rights reserved.

Client-Server Model

FEATURES OF HTTP PROTOCOL

- HTTP (Hypertext Transfer Protocol) is a TCP/IP based communication protocol, which is used to deliver data over the web.
- This protocol is used to change or transfer the hypertext.
- Hypertext is a structure text that uses logical links between containing text.
- It is a stateless protocol—it does not require the HTTP server to retain information about each user.

©Simplilearn. All rights reserved.

Client-Server Model

SERVER RESPONSE

Server generates response for Client request. It sends three things:

1. Status which tells whether client's response has successfully reached the server or not. Example: 404 error for page not found.
2. Information about the type of content that the server is going to send. The server sends HTML pages as response.
3. Actual content, example: when client types **http://www.google.com** in any browser, the server navigates the client to the Google home page.

©Simplilearn. All rights reserved.

Client-Server Model

TYPES OF SERVERS

Commonly used server types are as follows:

- Web Server
- Application Server
- Proxy Server
- Virtual Server
- Blade Server
- File Server
- Policy Server

We will focus on web and application server in this course.

©Simplilearn. All rights reserved.

Web Server Vs. Application Server

Web Server	Application Server
Web Server program runs on the server.	It is a component based product.
It looks for incoming requests and services those requests. Once the web server receives a request, depending on the type of request, it might look for a web page, or execute a program on the server.	It provides middleware services for security and state maintenance, along with data access and persistence.
Web servers have plugins to support scripting languages like Perl, PHP, ASP, JSP etc. through which they can generate dynamic HTTP content.	Application Server can do whatever Web Server is capable of doing as it has Web Server as an integral part. Additionally, it has components and features to support Application level services such as Connection Pooling, Object Pooling, Transaction Support, Messaging services, etc.
It acts as a container which serves static content.	It acts as container upon which business logic is built.
Web Server only supports Servlets and JSP.	Application Server supports distributed transaction and EJB.

©Simplilearn. All rights reserved.

Client-Server Model

CLIENT REQUEST TYPES

- GET
- POST
- PUT
- DELETE
- TRACE
- OPTIONS
- HEAD

©Simplilearn. All rights reserved.

Client-Server Model

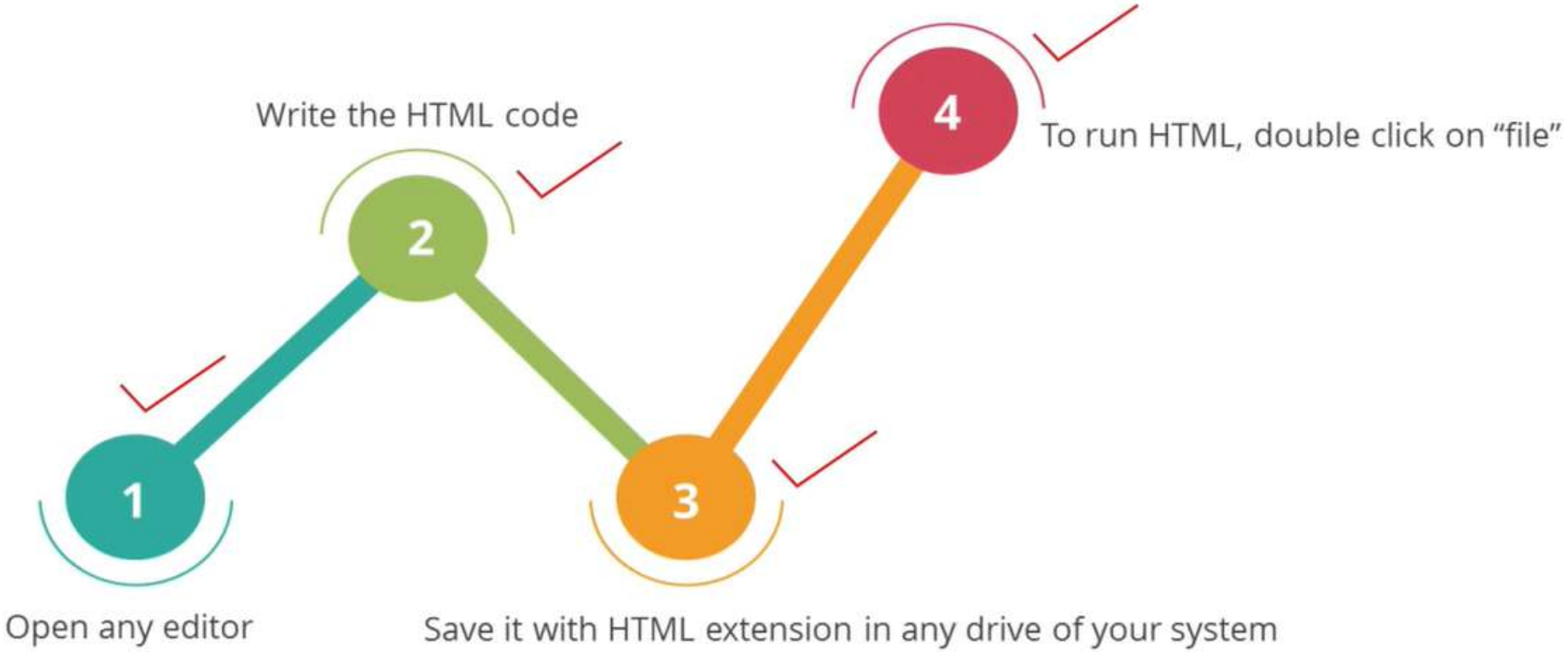
SERVER RESPONSE: HTML

- HTML stands for Hypertext Mark-up Language.
- It refers to the way the web pages (HTML documents) are linked together. The link available on a web page is called Hypertext.
- HTML contains tags and elements which are used to format web pages.

©Simplilearn. All rights reserved.

Client-Server Model

SERVER RESPONSE: RUNNING AN HTML CODE



©Simplilearn. All rights reserved.

Press Esc to exit full screen

Advanced Java

Java Servlet I—Servlets API, Interfaces, and Methods

Why Servlet?

We have learned that web server generates two types of responses: Static and Dynamic.

However, a web server can generate static responses in the form of HTML.

To generate dynamic response, a programming language that performs dynamic operations is needed.

Servlet is a Java program that can perform dynamic operations and send to web server. The web server then sends this response to web client.

©Simplilearn. All rights reserved.

What is a Web Container?

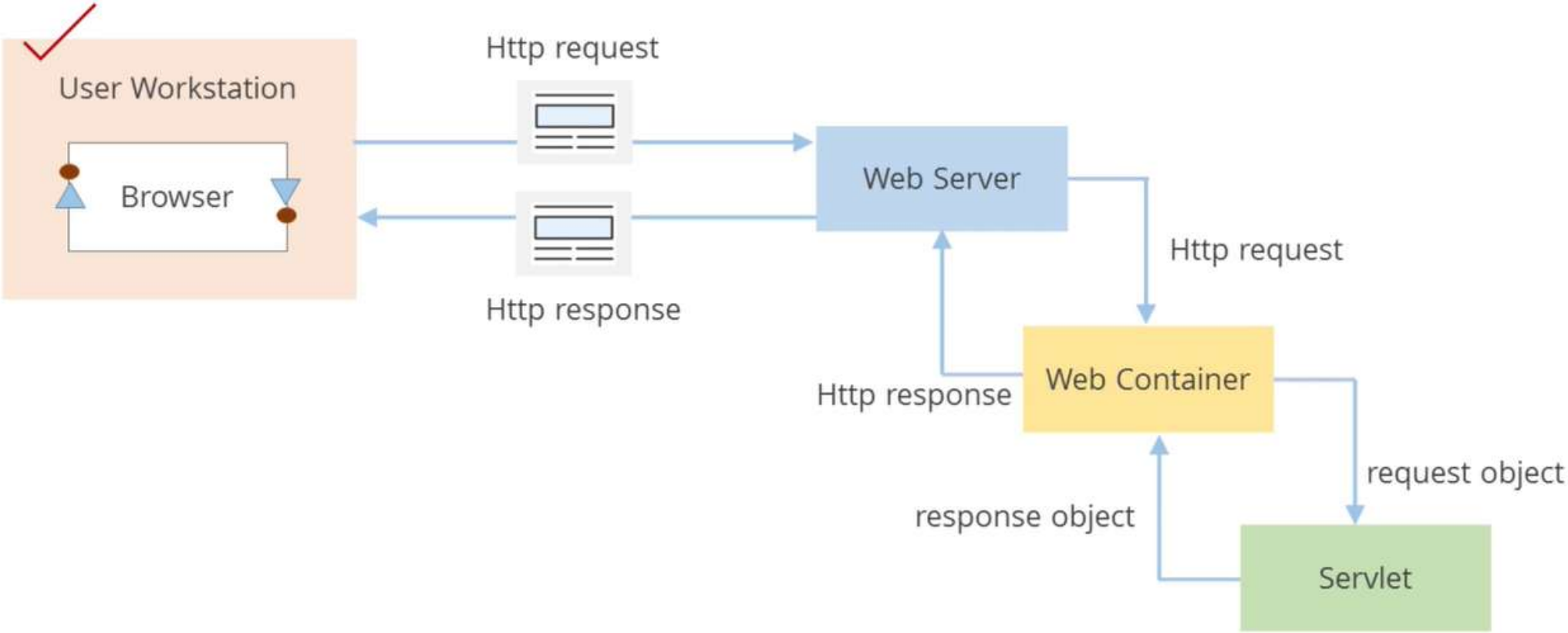
A web container is built on top of the Java EE platform, and it implements the Servlet API and the services required to process HTTP (and other Transmission Control/Internet Protocol [TCP/IP]) requests.

- Java Servlets are components that must exist in a web container. Web container activates the Servlet that matches the requested URL by calling the service method on an instance of Servlet class.
- Activation of the service method for a given HTTP request is handled in a separate thread within the web container protocol.

©Simplilearn. All rights reserved.

Client-Server Architecture with Web Container

The lifecycle of a Servlet is controlled by the web container in which the Servlet has been deployed.



©Simplilearn. All rights reserved.

Servlets API

Servlet API contains a number of classes and interfaces that describe the contracts between a servlet class and the runtime environment provided for an instance by a conforming servlet container.

Servlet API provides the following two packages that contain its classes and interfaces:

javax.servlet.*;	Classes and interfaces defines the contracts between a Servlet class and the runtime environment provided for an instance of such a class by a conforming Servlet container.
javax.servlet.http.*;	Classes and interfaces defines the contracts between a Servlet class running under the HTTP protocol and the runtime environment provided for an instance of such a class by a conforming Servlet container.

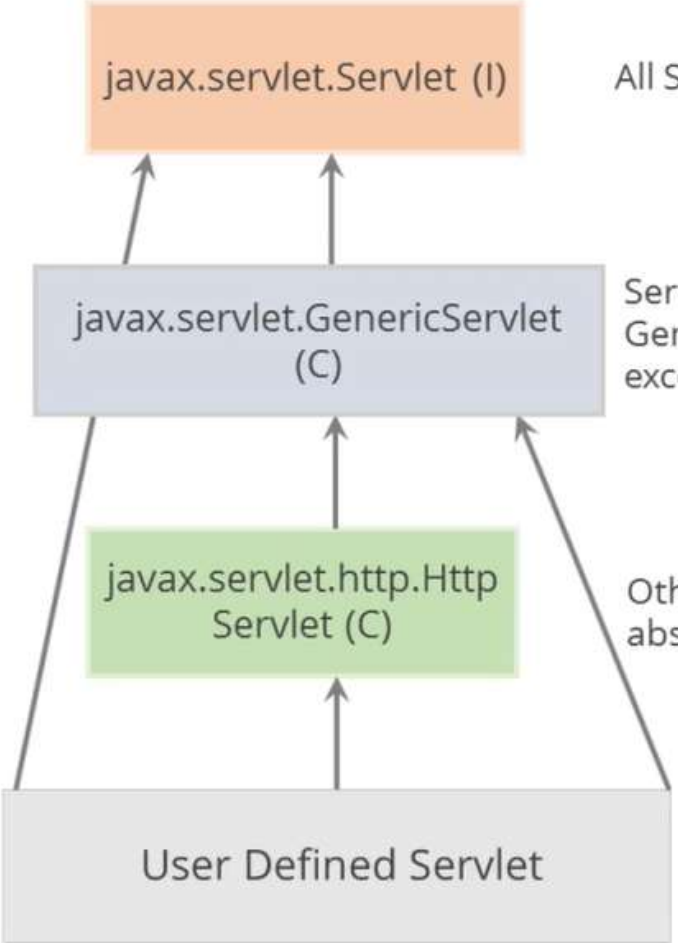
©Simplilearn. All rights reserved.

Servlet API Interface

javax.servlet.FilterConfig	Passes information to a filter during initialization
javax.servlet.RequestDispatcher	Sends request and response object to any resource (such as a Servlet, HTML file, or JSP file) on the server
javax.servlet.Servlet	Defines methods that all Servlets must implement
javax.servlet.ServletConfig	A Servlet configuration object used by a Servlet container to pass information to a Servlet during initialization
javax.servlet.ServletContext	Defines a set of methods that a Servlet uses to communicate with its Servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file
javax.servlet.ServletContextAttributeListener	Implementation of this interface receives notifications of changes to the attribute list on the Servlet context of a web application
javax.servlet.ServletContextListener	Implementation of this interface receive notifications about changes to the Servlet context of the web application they are a part of
javax.servlet.ServletRequest	Defines an object to provide client request information to a Servlet
javax.servlet.ServletRequestAttributeListener	A ServletRequestAttributeListener can be implemented by the developer interested in being notified of request attribute changes
javax.servlet.ServletRequestListener	A ServletRequestListener can be implemented by the developer interested in being notified of requests coming in and out of scope in a web component
javax.servlet.ServletResponse	Defines an object to assist a Servlet in sending a response to the client

©Simplilearn. All rights reserved.

Servlets API Hierarchy to Create Servlet



All Servlets must implement Servlet interface that defines lifecycle methods.

Servlet can also be created by extending GenericServlet abstract class. GenericServlet abstract class implements all methods of Servlet interface except service method.

Other way of creating a Servlet is to extend HttpServlet abstract class. HttpServlet abstract class extends GenericServlet abstract class.

I—Interface
C—Class

©Simplilearn. All rights reserved.

Servlet Methods

Servlet interface has 5 methods:

1. init
2. service
3. destroy
4. getServletInfo
5. getServletConfig

©Simplilearn. All rights reserved.

Generic Servlet Abstract Class

- GenericServlet implements Servlet interface.
- It defines Servlet generic and is independent of protocols.
- It doesn't give implementation for service method of Servlet interface.
- If your Servlet extends Generic Class, implementing service method is necessary.

©Simplilearn. All rights reserved.

Generic Servlet Abstract Class Methods

`public abstract class GenericServlet`	Extends java.lang.Object, implements Servlet, ServletConfig, and java.io.Serializable
`java.lang.String getInitParameter(java.lang.String name)`	Returns a String containing the value of the named initialization parameter, or returns null if the parameter does not exist
`java.util.Enumeration getInitParameterNames()`	Returns the names of the Servlet's initialization parameters as an Enumeration of String objects, or returns an empty Enumeration if the Servlet has no initialization parameters
`ServletContext getServletContext()`	Returns a reference to the ServletContext in which the Servlet is running
`java.lang.String getServletName()`	Returns the name of this Servlet instance
`void init()`	A convenient method which can be overridden so that there's no need to call super.init(config)

©Simplilearn. All rights reserved.

Generic Servlet Abstract Class Methods

`void init(ServletConfig config)`	Called by the Servlet container to indicate to a Servlet that the Servlet is being placed into service
`void log(java.lang.String msg)`	Writes the specified message to a Servlet log file, prepended by the Servlet's name
`void log(java.lang.String message, java.lang.Throwable t)`	Writes an explanatory message and a stack trace for a given Throwable exception to the Servlet log file prepended by the Servlet's name
`abstract void service(ServletRequest req, ServletResponse res)`	Allows the Servlet to respond to a request.

©Simplilearn. All rights reserved.

HttpServlet Abstract Class Methods

It provides an abstract class to be sub-classed to create an HTTP Servlet suitable for a website.

Following is the list of methods:

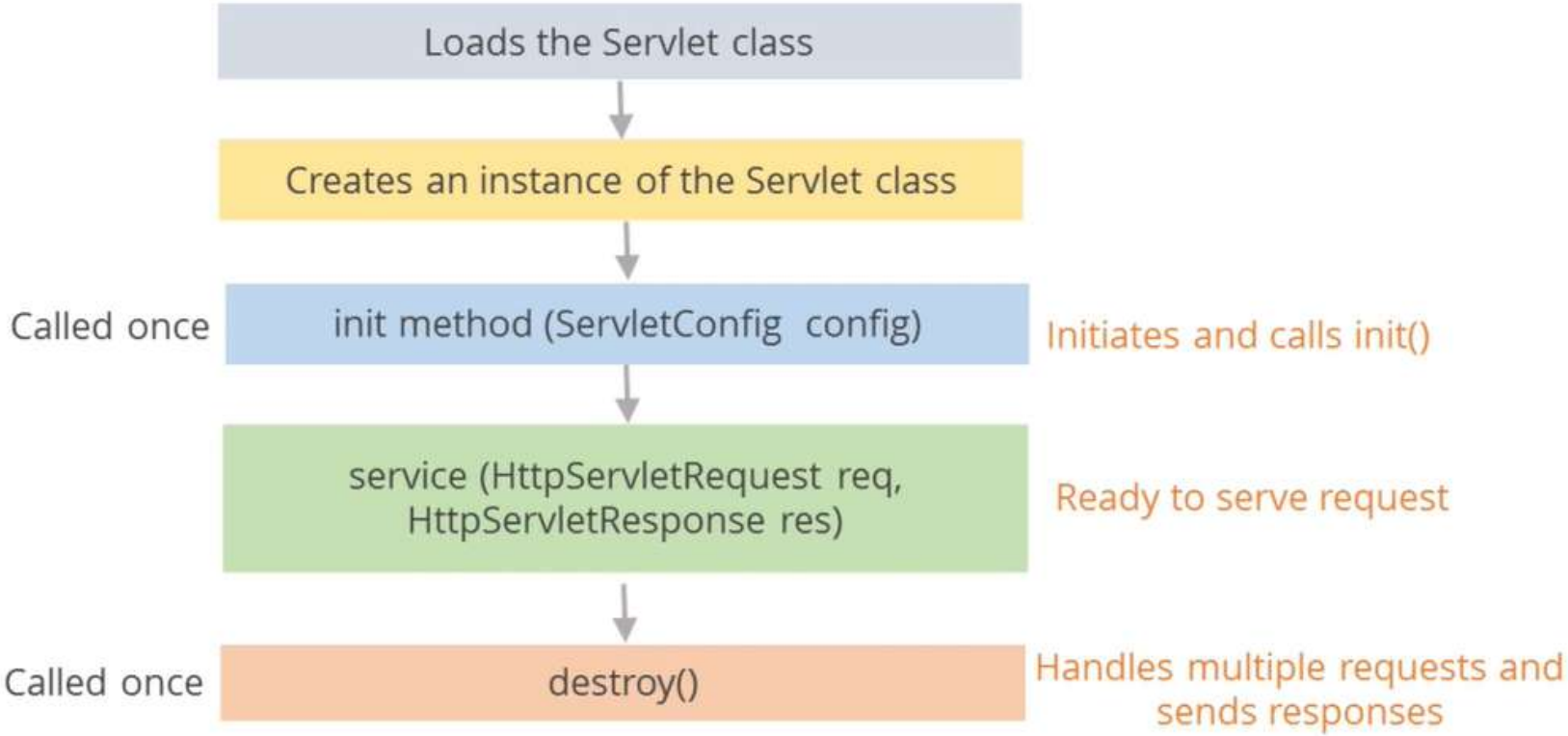
`public abstract class` **`HttpServlet`**	Extends GenericServlet Implements java.io.Serializable
`doGet`	Used if the Servlet supports HTTP GET requests
`doPost`	Used for HTTP POST requests
`doPut`	Used for HTTP PUT requests
`doDelete`	Used for HTTP DELETE requests
`init and destroy`	Used to manage resources that are held for the life of the Servlet
`getServletInfo`	Used by the Servlet to provide information about itself

©Simplilearn. All rights reserved.

Servlet Lifecycle

The lifecycle of a Servlet is controlled by the container in which the Servlet has been deployed.

When a request is mapped to a Servlet, the container performs the following steps if an instance of the Servlet does not exist.



©Simplilearn. All rights reserved.

ServletRequest

public interface **ServletRequest**

- It defines an object to provide client request information to a Servlet.
- The Servlet container creates a ServletRequest object.
- Servlet Container calls service method by passing ServletRequest object.
- This object provides data including parameter name and values, attributes, and an input stream.

©Simplilearn. All rights reserved.

ServletRequest Interface

`java.lang.Object getAttribute(java.lang.String name)`	Returns the value of the named attribute as an object, or returns null if no attribute of the given name exists.
`java.util.Enumeration getAttributeNames()`	Returns an Enumeration containing the names of the attributes available to this request.
`java.lang.String getCharacterEncoding()`	Returns the name of the character encoding used in the body of this request.
`int getContentLength()`	Returns the length of the request body (in bytes) and is made available by the input stream, or -1 if the length is not known.
`java.lang.String getContentType()`	Returns the MIME type of the body of the request, or returns null if the type is not known.
`ServletInputStream getInputStream()`	Retrieves the body of the request as binary data using a ServletInputStream.
`java.util.Locale getLocale()`	Returns the preferred Locale that the client will accept content, based on the Accept-Language header.
`java.util.Enumeration getLocales()`	Returns an Enumeration of Locale objects indicating the locales (in decreasing order starting with the preferred locale) that are acceptable to the client based on the Accept-Language header.

©Simplilearn. All rights reserved.

ServletRequest Interface (Cont.d)

`java.lang.String getLocalName()`	Returns the host name of the Internet Protocol (IP) interface on which the request was received
`int getLocalPort()`	Returns the Internet Protocol (IP) port number of the interface on which the request was received
`java.lang.String getParameter(java.lang.String name)`	Returns the value of a request parameter as a String, or returns null if the parameter does not exist
`java.util.Map getParameterMap()`	Returns a java.util.Map of the parameters of this request
`java.util.Enumeration getParameterNames()`	Returns an Enumeration of String objects containing the names of the parameters contained in this request
`java.lang.String[] getParameterValues(java.lang.String name)`	Returns an array of String objects containing all of the values the given request parameter has, or returns null if the parameter does not exist
`java.lang.String getProtocol()`	Returns the name and version of the protocol the request uses in the form protocol/majorVersion.minorVersion, for example, HTTP/1.1
`java.io.BufferedReader getReader()`	Retrieves the body of the request as character data using a BufferedReader
`void setAttribute(java.lang.String name, java.lang.Object o)`	Stores an attribute in this request

©Simplilearn. All rights reserved.

ServletResponse

public interface **ServletResponse**

- It defines an object to assist a Servlet in sending a response to the client.
- The Servlet Container creates these objects to call Servlet's service method.
- To send binary data in a MIME body response, use the ServletOutputStream returned by getOutputStream().
- To send character data, use the PrintWriter object returned by getWriter(). To mix binary and text data, for example, to create a multipart response, use a ServletOutputStream and manage the character sections manually.

©Simplilearn. All rights reserved.

ServletConfig, ServletContext

ServletConfig:

1. The config object is created by the web container based on the initialization parameters specified in the deployment descriptor
2. One ServletConfig per Servlet

ServletContext:

1. To share application-specific data across independent web components
2. One ServletContext per application
3. The context object is accessible to all Servlets in web application
4. A ServletContext object is the runtime representation of the web application

©Simplilearn. All rights reserved.

ServletConfig Methods

java.lang.String getInitParameter(java.lang.String name)	Returns a String containing the value of the named initialization parameter, or returns null if the parameter does not exist
java.util.Enumeration getInitParameterNames()	Returns the names of the servlet's initialization parameters as an Enumeration of String objects, or returns an empty Enumeration if the servlet has no initialization parameters
ServletContext getServletContext()	Returns a reference to the ServletContext in which the caller is executing.
java.lang.String getServletName()	Returns the name of this servlet instance
public interface ServletContext	Defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file
java.lang.Object getAttribute(java.lang.String name)	Returns the servlet container attribute with the given name, or returns null if there is no attribute by that name

©Simplilearn. All rights reserved.

ServletConfig Methods

java.util.Enumeration getAttributeNames()	Returns an Enumeration containing the attribute names available within this servlet context
ServletContext getContext(java.lang.String uripath)	Returns a ServletContext object that corresponds to a specified URL on the server
java.lang.String getInitParameter(java.lang.String name)	Returns a String containing the value of the named context-wide initialization parameter, or returns null if the parameter does not exist
java.util.Enumeration getInitParameterNames()	Returns the names of the context's initialization parameters as an Enumeration of String objects, or returns an empty Enumeration if the context has no initialization parameters

©Simplilearn. All rights reserved.

Servlet Attributes

Attributes allow data to be stored and used in inter servlet communication.

The methods associated with attribute manipulation are:

Attribute Scope	
Attribute	**Scope of data**
ServletRequest	Duration of the request
HttpSession	While the client is active
ServletContext	The life of the web application

We will learn about Session scope in lesson 3 and management.pageContext is described in JSP (lesson 4)

©Simplilearn. All rights reserved.

Attribute Specific Methods

1. public void setAttribute (String name, Object object): This method is used to set the given object in the application scope.

2. public Object getAttribute(String name): This method is used to return the attribute for the specified name.

3. public Enumeration getInitParameterNames(): This method is used to return the names of the context's initialization parameters as an Enumeration of String objects.

4. public void removeAttribute(String name): This method is used to remove the attribute with the given name from the servlet context.

©Simplilearn. All rights reserved.

What is Servlet Collaboration?

- The Servlet collaboration involves sharing of information among the Servlets.

Collaborating Servlets is passing the common information that is to be shared directly by one Servlet to another Servlet of HTML or JSP through various invocations of methods.

Servlet should know about the other Servlets with which it is collaborated.

©Simplilearn. All rights reserved.

Ways of Servlet Collaboration

- Using RequestDispatchers *include()* and *forward()* method;
- Using HttpServletResponse *sendRedirect()* method;
- Using ServletContext *setAttribute()* and *getAttribute()* methods;

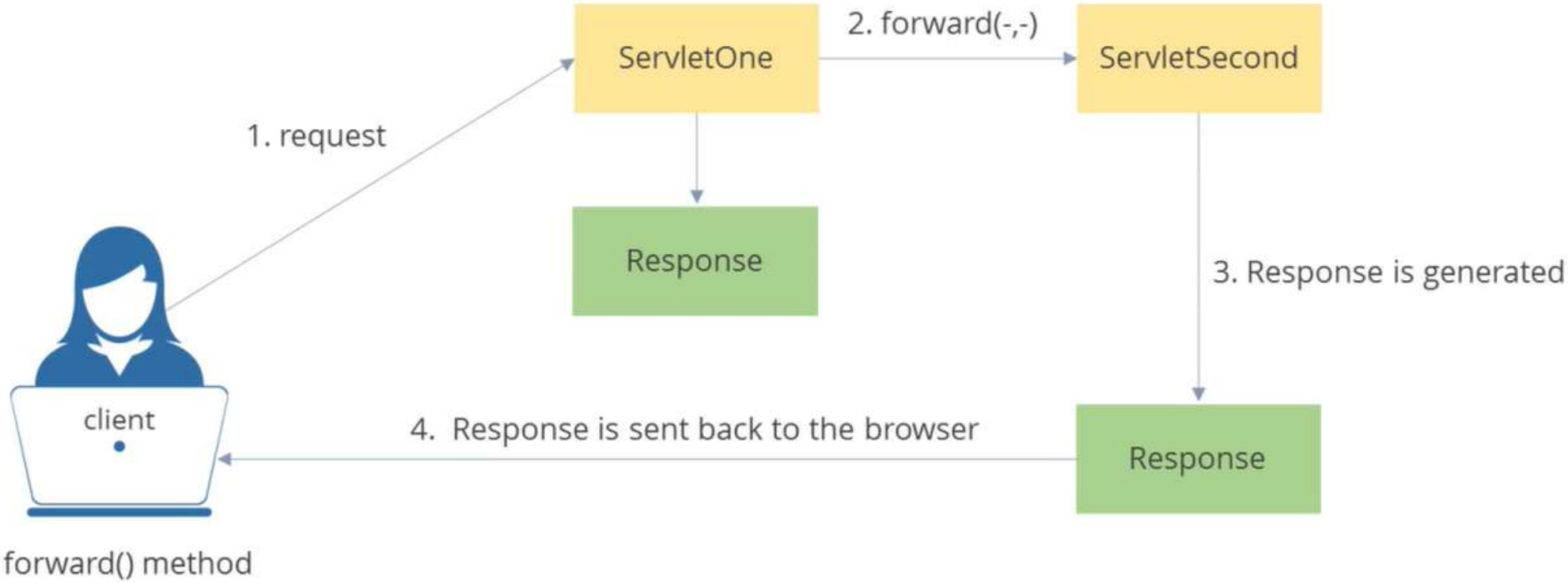
©Simplilearn. All rights reserved.

Forwarding and Redirecting

- RequestDispatcher interface is used to forward request and response objects to another Servlet
- sendRedirect method is used for Redirecting request and response object
- HttpServletResponse has sendRedirect method: response.sendRedirect("url");
- HttpServletRequest has getRequestDispatcher method that returns RequestDispatcher object
- There are two methods, forward and include

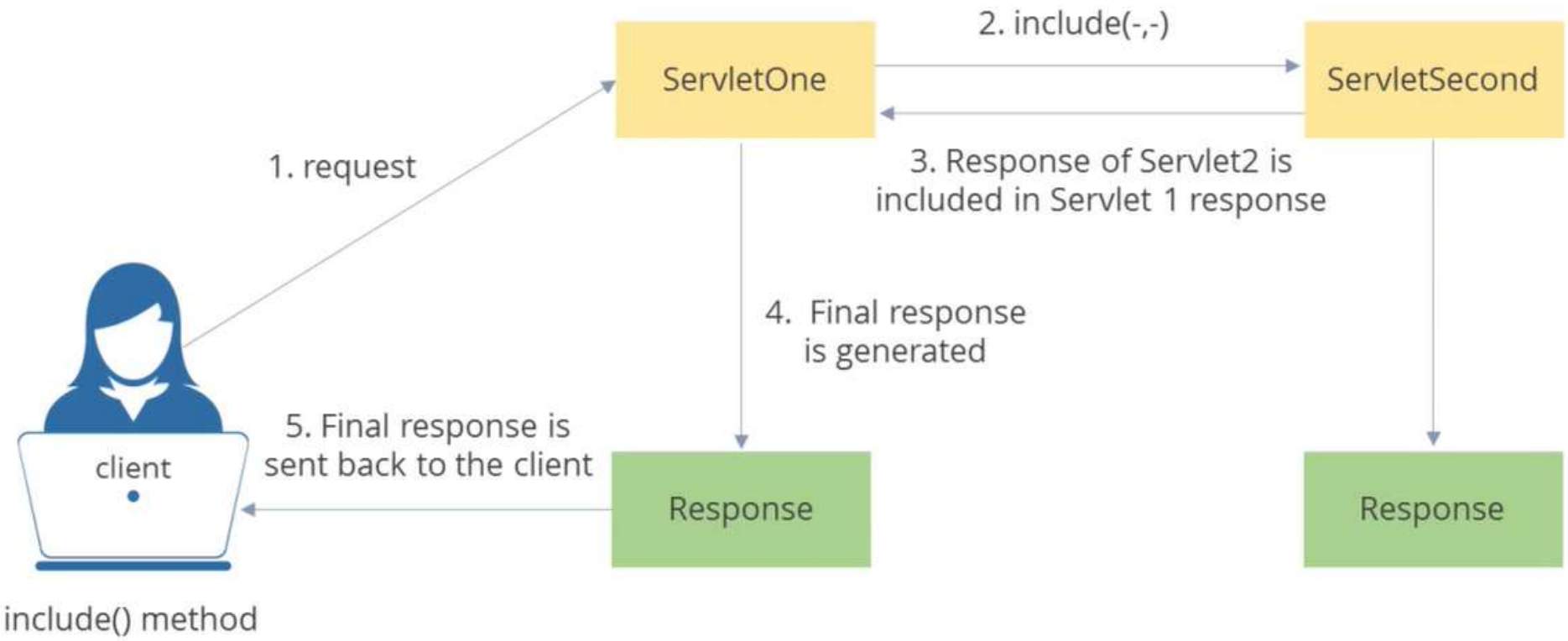
©Simplilearn. All rights reserved.

forward() method



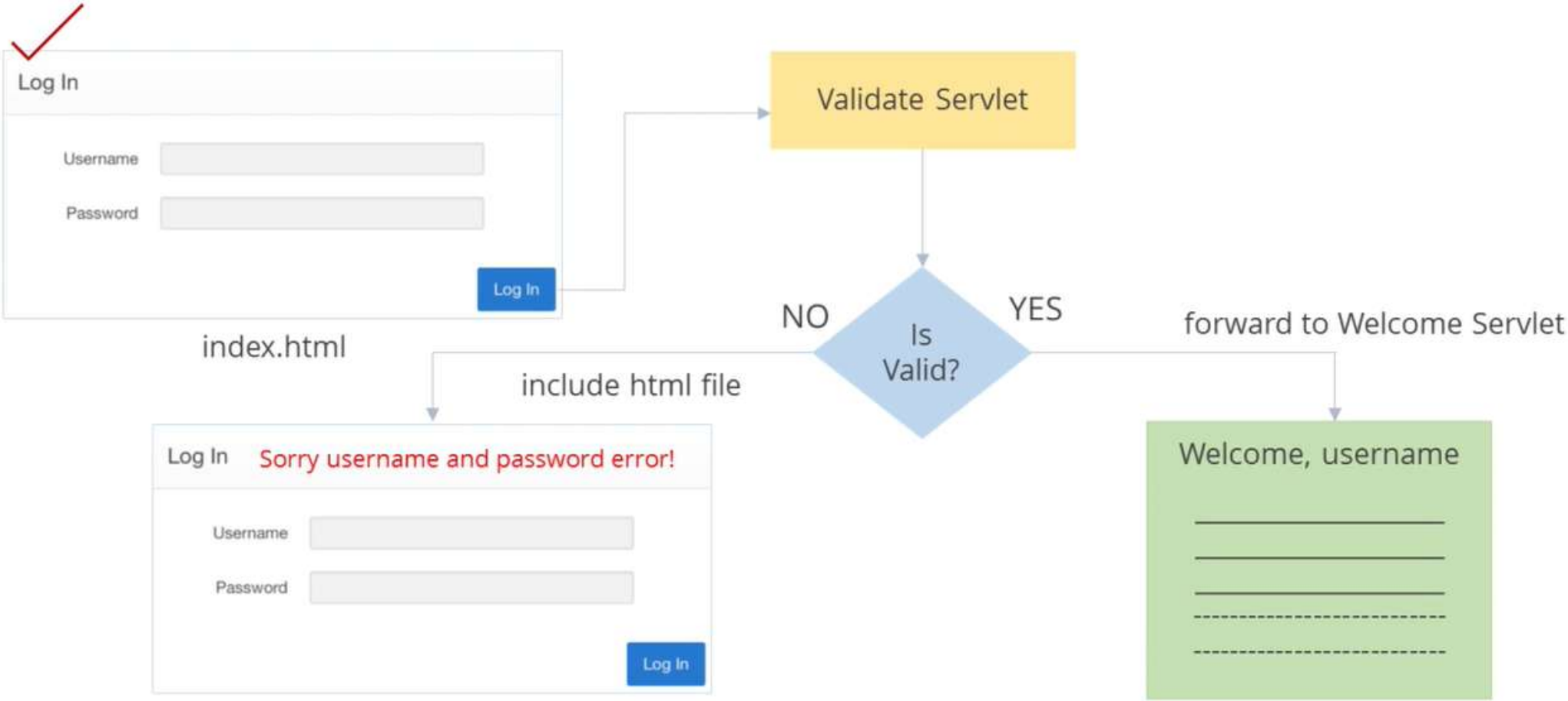
©Simplilearn. All rights reserved.

include() method



©Simplilearn. All rights reserved.

RequestDispatcher Interface



©Simplilearn. All rights reserved.

Advanced Java

Java Servlet II—Session Management

00:02 / 34:20

Stateless Web Application

We have learned that web applications are stateless (by default) if given protocol is HTTP (stateless protocol).

It does not allow or provide data access to previous request during the processing of current request in the Servlet or JSP.

In stateless web app, there is no way to preserve client data across multiple requests:

©Simplilearn. All rights reserved.

Stateful Web Application

- A web application that can remember or use the data of previous request during the processing of current request is a called stateful web application.
- In servlet API, there are different ways of making web application stateful so that it remembers client data across request from same browser.
- Making web application stateful is known as session tracking or session management.

©Simplilearn. All rights reserved.

Introduction to Session Management

Session Management is a mechanism used by the **web container** to store **session** information for a particular user.

Session is a conversional state between client and server, and it can consist of multiple requests and responses between client and server. Sessions are used for maintaining user specific state, including persistent objects.

User Session: It refers to a series of user application interactions that are tracked by the server.

© Simplilearn. All rights reserved.
03:13 / 34:20

Types of Session Management

1. Hidden Form Field
2. URL Writing
3. Cookies
4. HttpSession

©Simplilearn. All rights reserved.

Types of Session Management

HIDDEN FORM FIELD

Hidden Form Field
URL Writing
Cookie
HttpSession

- When Server sends response for one particular request, it sends one hidden field attached to it
- Client gets response with one Hidden Field
- In Hidden Field name and value are set
- If the same page sends request the next time, server can recognize it by getting the parameter of Hidden Field

```
<input type ="hidden field" name="somename" value="somename"/>
```

One of the limitations of a Hidden Form Field is that it requires extra form submission on each page.

©Simplilearn. All rights reserved.

Types of Session Management

URL WRITING

Hidden Form Field
URL Writing
Cookie
HttpSession

- In URL writing, one token is appended to URL itself
- This token contains one name and one value
- With the help of hyperlink, token's name and values are passed to server
- A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&)
- Code: `url?name1=value1&name2=value2&?`

URL writing works with hyperlinks and sends text information.

©Simplilearn. All rights reserved.

Types of Session Management

COOKIE

Hidden Form Field
Url Writing
Cookie
HttpSession

- Servlet API provides javax.servlet.http.Cookie class to maintain session
- This class extends Object class and implements Cloneable interface
- While creating a cookie, a small amount of information is sent by a Servlet to a web browser, saved by the browser, and later sent back to the server
- Cookie has a name, value, and optional attribute
- The browser returns cookies to the servlet by adding fields to HTTP request headers

©Simplilearn. All rights reserved.

Types of Session Management

HttpSesson

Hidden Form Field
Url Writing
Cookie
HttpSession

- javax.servlet.http.HttpSession interface provides a way to identify one user across more than one page
- Servlet Container uses this interface to create a session between Http Client and Http Server
- Servlet can view and manipulate the information about servlet
- javax.servlet.http.HttpSession has method named as getSession
- getSession method gets the current valid session associated with this request if create is false or, if necessary, it creates a new session for the request

©Simplilearn. All rights reserved.

Events in Servlet API

Change in the state of an object is known as an event, which occurs during the lifecycle of an object.

There are important events related to a web application and Servlets which are handled by event listener.

To be notified of events, a custom class, that implements the correct listener interface, needs to be coded and the listener class needs to be deployed via web.xml.

©Simplilearn. All rights reserved.

Event Class Servlet APIs

1. ServletRequestEvent
2. ServletContextEvent
3. ServletRequestAttributeEvent
4. ServletContextRequestArrtibuteEvent
5. HttpSessionEvent
6. HttpSessionEventBindingEvent

©Simplilearn. All rights reserved.

Event Listener Interface

- Listeners are objects that notify whenever any event occurs. They contain event processing logics.

Listener Interfaces in servlet APIs are as follows:

1. ServletRequestListener
2. ServletContextListener
3. ServletRequestAttributeListener
4. ServletContextAttributeListener
5. HttpSessionListener
6. HttpSessionAttributeListener
7. HttpSesionBindingListener
8. HttpSessionActivationListener

©Simplilearn. All rights reserved.

Why Filters?

Filters are used when there is a need:

- For an operation to occur every time a particular request is made
- To perform one operation on other request in the web application
- To allow one operation to be turned off at deployment

The filter API interface can be found in javax.servlet package

©Simplilearn. All rights reserved.

What is a Filter?

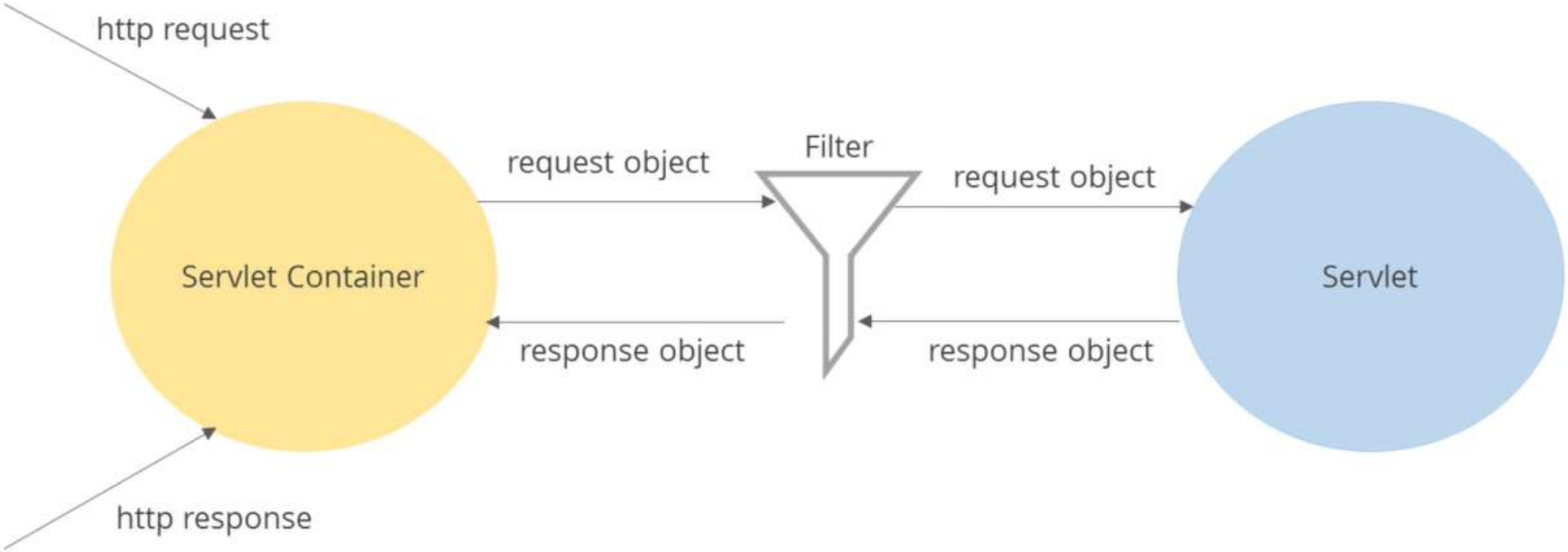
- A filer is an interface used to filter tasks on the request to a resource (a servlet or static content), on the response from a resource, or both
- It has 3 methods (lifecycle)

```
public void init(FilterConfig config);
public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain);
public void destroy();
```

- Filters are configured in deployment descriptor of a web application

©Simplilearn. All rights reserved.

Filter in Servlets



©Simplilearn. All rights reserved.

Filter Types

1. Request Filter: Contains only pre-request processing logic.
2. Response Filter: Contains post-response generation logic.
3. Request-Response Filter: Contains both pre-request processing and post-response generation logic.

©Simplilearn. All rights reserved.

Developing and Mapping a Filter Class

- Declaring a Filter in the web.xml

```
<filter>
  <filter-name> onelogicalname </filter-name>
  <filter-class> filterclassname </filter-class>
  </filter>
```

- Mapping a Filter in the web.xml

```
<filter-mapping>
  <filter-name> onelogicalname </filter-name>
  <furl-pattern> /urlpatter </url-pattern>
  </filter-mapping>
```

©Simplilearn. All rights reserved.

Writing a Filter Program

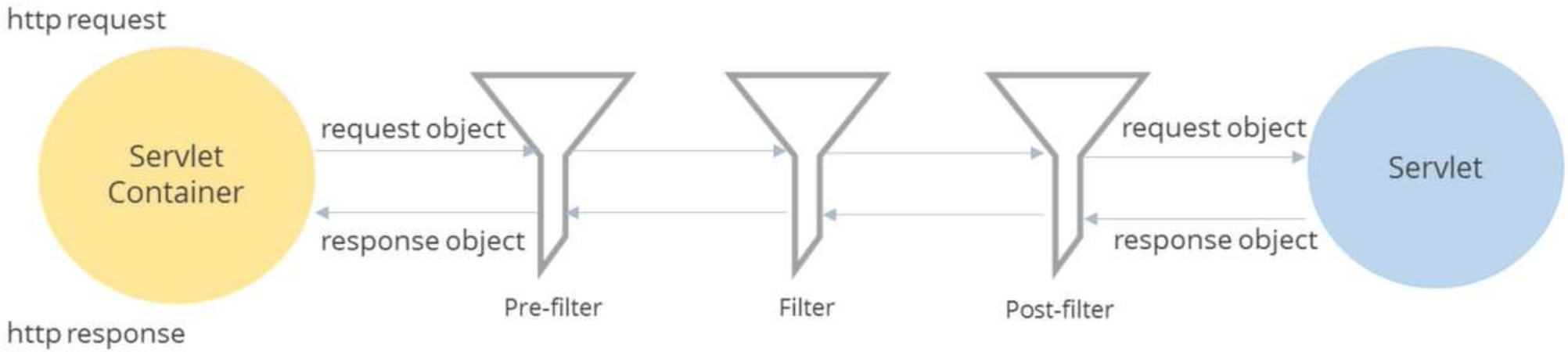
To write a filter program:

- Create a Java class to implement Filter interface that has the following three methods:
  - init() method – It is called once when the container instantiates a filter
  - doFilter() method – It is called for every request intercepted by the filter
  - destroy() method - Before the web container removes a filter instance from service, the destroy method is called
- Create one web.xml file to configure the filter

©Simplilearn. All rights reserved.

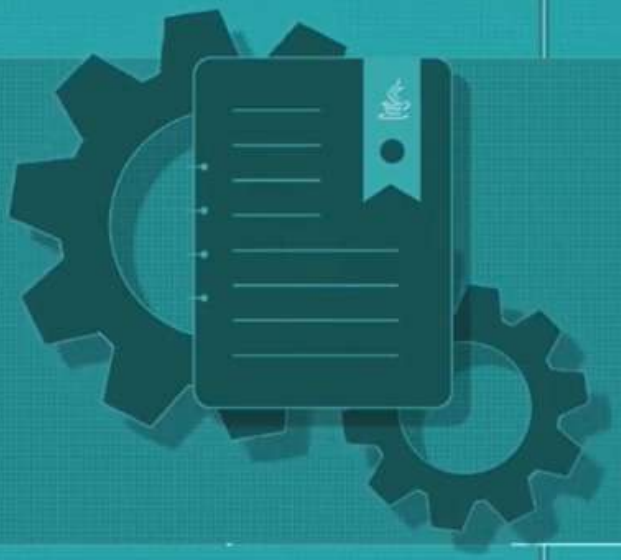
Filter Chain

If multiple filters are configured for servlet, they are executed in the same order in which they are defined in web.xml



©Simplilearn. All rights reserved.

Press Esc to exit full screen



Advanced Java

Java Server Pages—JSP Lifecycle

What Is JSP?

- JSP (Java Server Pages) is a text document consisting of Hyper Text Markup Language(HTML), Extensible Markup language(XML), and JSP elements which can be expressed in standard and XML syntax.

©Simplilearn. All rights reserved.

Advantages of JSP

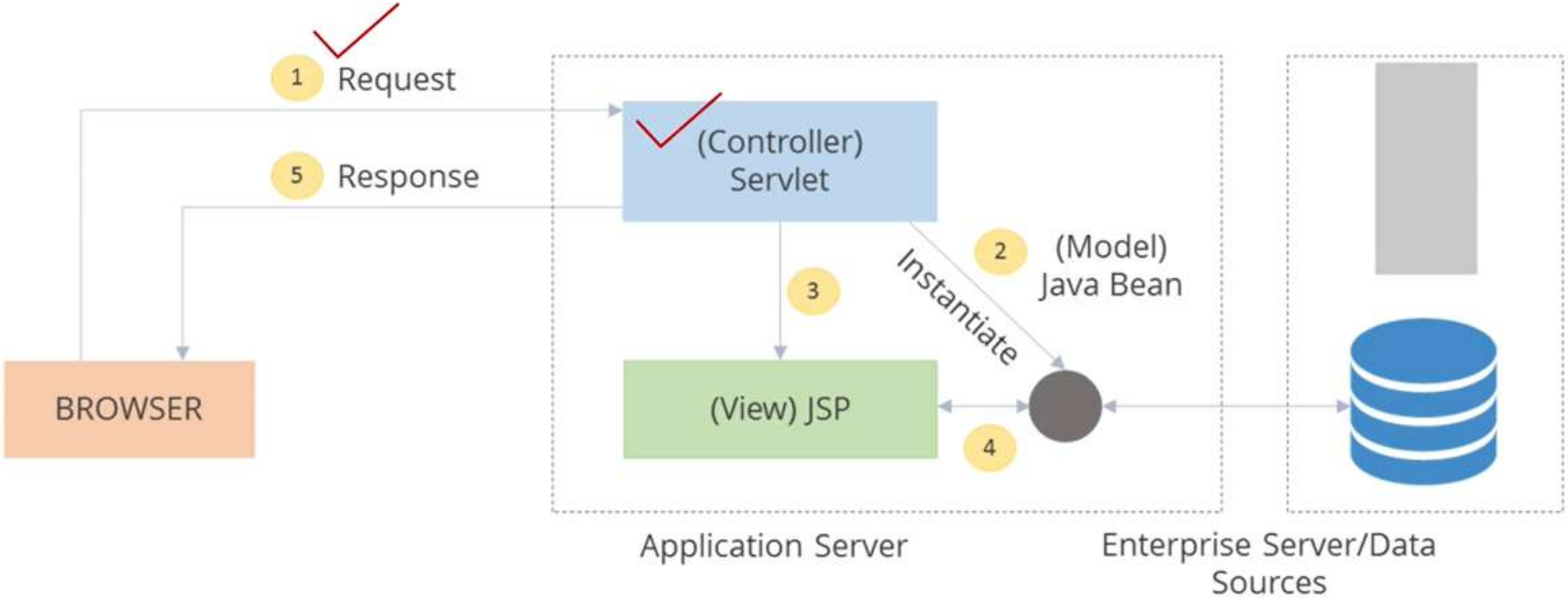
JSP is used to develop web based applications. It supports the separation of presentation and business logic as follows:

- Web designers can design and update pages without learning the Java programming language.
- Java technology programmers can write code without having to be concerned with web page design.
- It provides the optional mechanism to configure web application file (web.xml).
- JSP programming eliminates the need for repeated deployment (saving the JSP and making a request whenever the JSP is changed). The automatic deployment is taken care by container JASPER [JSP Execution Environment].
- JSP programming provides custom tags development.
- JSP programming environment provides automatic page compilation.

To configure JSP, create a dynamic web project in eclipse, configure Apache Tomcat, and create JSP file in web content. The steps are already discussed in the installation guide in your LMS.

©Simplilearn. All rights reserved.

JSP Architecture



©Simplilearn. All rights reserved.

JSP vs. Servlet

JSP	Servlets
JSP is a web page scripting language that can generate dynamic content.	Servlets are Java programs that are Compiled. They create dynamic pages.
It is easier to code in JSP than in Java Servlet.	Involves writing a lot of code
In MVC model, JSP acts as a view.	In MVC model, Servlets act as controllers.
Custom tag can directly call Java beans.	There is no such custom tag.
JSP is generally preferred when there is not much processing of data required. It runs slower compared to Servlet as it takes compilation time to convert into Java Servlet.	Servlets are best for use when there is more processing and manipulation involved. Servlet runs faster than JSP.

©Simplilearn. All rights reserved.

Web Container Responsibilities

What does a web container do in a JSP?

- It provides standard libraries required for coding.
- It provides a suitable environment for execution.
- It calls the call-back methods at appropriate times.
- It maintains the life cycle of the program.

©Simplilearn. All rights reserved.

JSP API

JSP technology is based on JSP API. It consists of the following packages:

- javax.servlet.jsp (Classes and interface for core JSP)
- javax.servlet.jsp.tagext (Java server pages library)
- javax.servlet.jsp.el (Expression language)

©Simplilearn. All rights reserved.

JSP Lifecycle

1. Translation of JSP to Servlet code
2. Compilation of Servlet to bytecode
3. Loading Servlet class
4. Creating Servlet instance
5. Initialization by calling _jspInit() method
6. Request processing by calling _jspService() method
7. Destroying object by calling _jspDestroy() method

©Simplilearn. All rights reserved.

What are Implicit Objects?

Implicit objects are the Java objects that the JSP Container makes available to the developers in each page; the developers can call them directly without explicitly declaring them.

- JSP technology has 9 implicit variables. They represent commonly used objects for servlets that JSP page developers might need to use.
- You can retrieve HTML from parameter data by using the request variable, which represents the HttpServletRequest object.

©Simplilearn. All rights reserved.

Implicit Variables

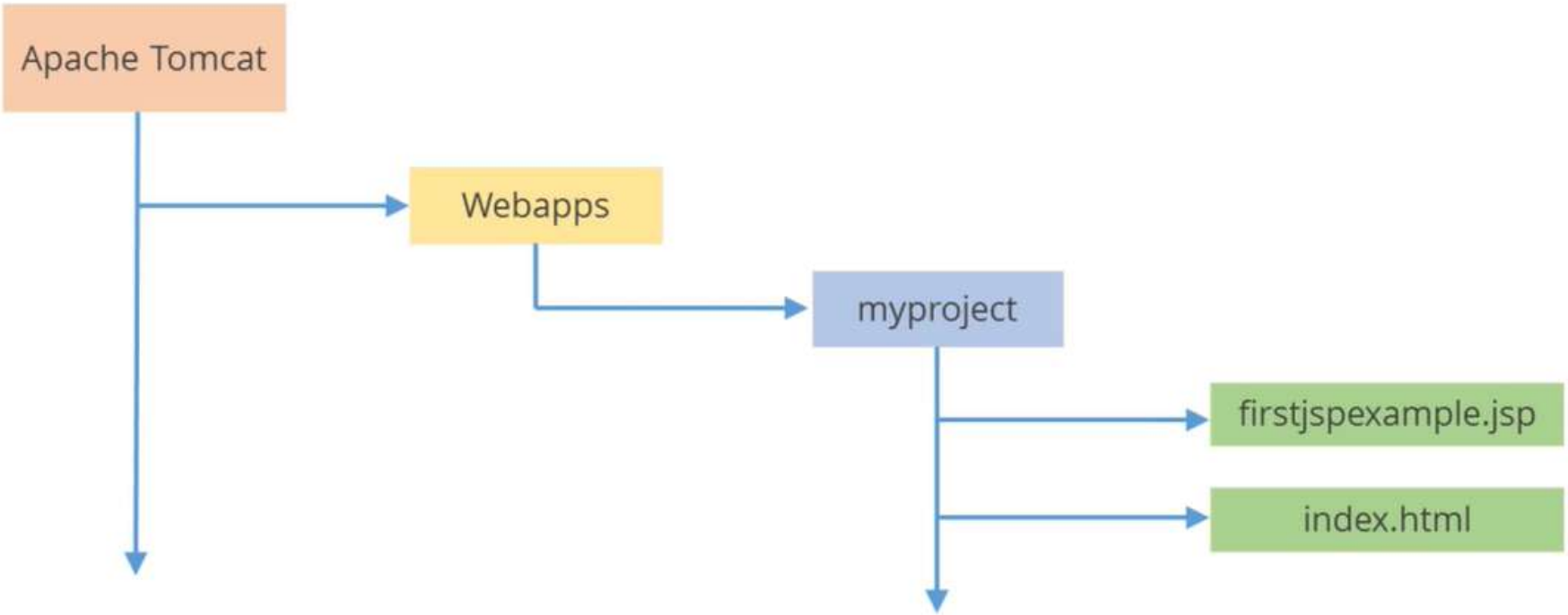
Variable Name	Class Name	Description
request	javax.servlet.http.HttpServletRequest	The HttpServletRequest object associated with the request
response	javax.servlet.http.HttpServletResponse	The HttpResponse object associated with the response that is sent back to the browser
out	javax.servlet.jsp.JspWriter	The JspWriter object associated with the output stream of the response
session	javax.servlet.http.HttpSession	The HttpSession object associated with the session for the given user of the request—only meaningful if the JSP page is participating in an HTTP session
application	javax.servlet.ServletContext	The ServletContext object for the web application
config	javax.servlet.ServletConfig	The ServletConfig object for the web application
pageContext	javax.servlet.jsp.PageContext	The pageContext object that encapsulates the environment of a single request for this JSP page
exception	java.lang.Object.Throwable	The Throwable object that was thrown by some other JSP page—only available in a JSP error page
page	java.lang.Object	Represents current JSP page

©Simplilearn. All rights reserved.

Creating a JSP and Running It In a Web Application

Unlike servlets, deploying JSP pages is as easy as deploying static pages. JSP pages can be placed in the same directory hierarchy as HTML pages.

In the development environment, JSP pages are placed in the web directory. In the deployment environment, JSP pages are placed in the top-level directory of the web application.



©Simplilearn. All rights reserved.

Creating a JSP and Running It In a Web Application

1. Configure Server
2. Create Dynamic Web Project
3. Create JSP file in WebContent Folder
4. Create HTML file in WebContent folder

©Simplilearn. All rights reserved.

JSP Elements

JSP elements in a JSP page can be expressed in two types of syntax:

1. standard
2. XML

Syntax Elements	Standard Syntax	XML Syntax
1. Comments	<% -- -- %>	<!-- -->
2. Declaration	<% ! %>	<jsp: declaration> </jsp: declaration>
3. Directives	<%@ include %> <% @page %> <%@taglib %>	<jsp : directive include../> <jsp:directive page ../> <xmlns :prefix = "tag library url">
4. Expression	<% = %>	<jsp:expression> </jsp:expression>
5. Scriptlets	<% %>	<jsp:scriptlet> </jsp:scriptlet>

©Simplilearn. All rights reserved.

JSP Comment Tag

- JSP comment is used when you are creating a JSP page and want to put in comments about what you are doing.
- JSP comments are only seen in the JSP page.
- These comments are not included in servlet source code during translation phase; they do not appear in the HTTP response.

Comment Tag	Declaration Tag	Directive Tag	Expression Tag	Scriptlets Tag

©Simplilearn. All rights reserved.

Caution

WHEN TO USE DECLARATION TAG OVER SCRIPLET TAG

- If you want to include any method in your JSP file, use declaration tag.
- During translation phase of JSP, methods and variables inside the declaration tag become instance methods and instance variables and are also assigned default values.
- Anything we add in scriplet tag goes inside the _jspService() method. We cannot add any function inside the scriplet tag as it creates a function inside the service method during compilation, which is not allowed in a Java method.

©Simplilearn. All rights reserved.

JSP Standard Action Elements

- Standard action elements are basically the tags that can be embedded into a JSP page.
- During compilation, they are also replaced by the Java code that corresponds to the pre-defined task.
- Action tags can be written only in XML syntax and can be used for communication.

©Simplilearn. All rights reserved.

List of JSP Standard Action Elements

- <jsp : forward >
- <jsp : include >
- <jsp : plugin >
- <jsp : param>
- <jsp : params >
- <jsp : fallback >
- <jsp:usebean>
- <jsp:setProperty>
- <jsp:getProperty>

©Simplilearn. All rights reserved.

UseBean Tag

This tag is used to interact with a JavaBean component using the standard tags in a JSP page.

```
Syntax :

<jsp:useBean id="beanName" scope="page | request | session | application "
 class ="className" / >
```

- id attribute specifies the attribute name of the bean.
- Location of the bean is specified by the cope attribute.
- The class attribute specifies the fully qualified classname.

©Simplilearn. All rights reserved.

setProperty Tag

The setProperty tag is used to store data in the JavaBeans instance.

```
Syntax:

<jsp:setProperty name="beanName" property_expresion / >
```

- Name attribute specifies the name of the JavaBeans instance.
- This must match the id attribute used in useBean tag.
- The property_expression can be represented as follows:

```
property="*" | property ="propertyName" | property="propertyName"
param="parameterName" | property="propertyName" value="propertyValue"
```

Property attribute specifies the property within the bean that will be set.

©Simplilearn. All rights reserved.

The getProperty Tag

The getProperty tag is used to retrieve a property from a JavaBeans instance and display it in the output stream.

```
Syntax:

<jsp:getProperty name="beanName" property_expresion / >
```

- Name attribute specifies the name of the JavaBeans instance.
- This must match the id attribute used in useBean tag, where property attribute specifies the property used for the get method.

©Simplilearn. All rights reserved.

Action Tag

- The action tag is used to insert the output of another JSP page into the current JSP page.
- The syntax for the jsp:include action has two forms.

```
jsp:include element that does not have a parameter name / value pair.
<jsp:include page = "relative URL" flush"true"/>
```

```
This syntax is used to pass information to the included resources.
<jsp:include page ="relative url" flush ="true">
<jsp:param… />
</jsp:include>
```

©Simplilearn. All rights reserved.

Include Directive vs. Include Action

include directive	include action
include directive includes the source code, so it is static	include action includes the output so it is dynamic.
include directive has an attribute for file.	include action has an attribute for page.
It can be written in HTML or XML syntax.	It can be written in XML syntax only.
It can include source code of a JSP and HTML, but not Servlet.	It can include the output of Servlet JSP and HTML.
We can't pass the expression as a value to a file.	We can pass the expression as a value to the page attribute.

©Simplilearn. All rights reserved.

JSP–Expression Language (EL)

JSP EL allows you to create arithmetic expression and logical expression. It uses integers, floating point numbers, strings, the built-in constants (true and false for Boolean values), and null.

It allows you to access application data stored in JavaBeans components easily.

```
Syntax:

It allows to specify an expression for any of attribute values as follows:

${expr}

Where expr specifies the expression

Example: <jsp:setProperty name = "rectangle" property="area" value=
"${length*width}"/>
```

©Simplilearn. All rights reserved.

EL Implicit Object

pageScope	Provides access to variable stored in page scope level
requestScope	Provides access to variable stored in page request scope
sessionScope	Provides access to variable stored in page session scope
applicatrionScope	Provides access to variable stored in page application scope
param	Gives you access to the parameter values available through the request.getParameter
paramValues	Gives you access to the parameter values available through the request.getParameterValue
header	Gives you HTTP request headers as strings
headerValues	Gives you HTTP request headers as collections of strings
initParam	Used as Context-initialization parameter
cookie	Maps the given cookie name to the cookie value
pageContext	Provides access to object request, session, etc.

©Simplilearn. All rights reserved.

JSTL

- JSTL 1.2 is a part of Java EE 5 platform
- JSP standard tag library (JSTL) represents a set of tags used to simplify development of JSP
- We can use the JSTL tags in various pages
- It avoids the use of scriptlet tag

JSTL Jar files can be downloaded from: http://www.oracle.com/technetwork/java/index-137889.html

©Simplilearn. All rights reserved.

Core JSTL Tag: Example

<c:out>

- Core JSTL tag is used to write expression and render data to page.

```
Welcome.html

<form  action =" process.jsp" method = "post">
First name: <input type = "text" name ="fname"/> </br>
Last name <input type = "text" name = "fname"/> </br/>
<input type = "Submit" value = "submit"/>
  </form>
```

```
Process.jsp

<%@taglib uri = "http://java.sun.com/jsp/jstl/core" prefix =
"c"%>
First name: <c: out value = "${ param.fname}"> </c: out> <br/>
Second name: <c:out value = "${param.fname}"> </c:out> <br/>
```

©Simplilearn. All rights reserved.

Custom Tag Syntax Rules

JSP allows you to create your own tags. They are known as custom tags. They uses XML syntax. There are four fundamental XML rules that all custom tags must follow:

1. Standard tag syntax must conform to the following structure:

```
<prefix:name { attribute ={"value" | 'value' } } * >
body
</prefix:name>
```

2. Empty tag syntax must conform to the following structure:

```
<prefix :name { attribute ={ "value" | 'value' } } * />
```

3. Tag name, attributes and prefix are case sensitive

4. Tag must follow nesting rule:

```
<tag1>
<tag2>
</tag2>
</tag1>
```

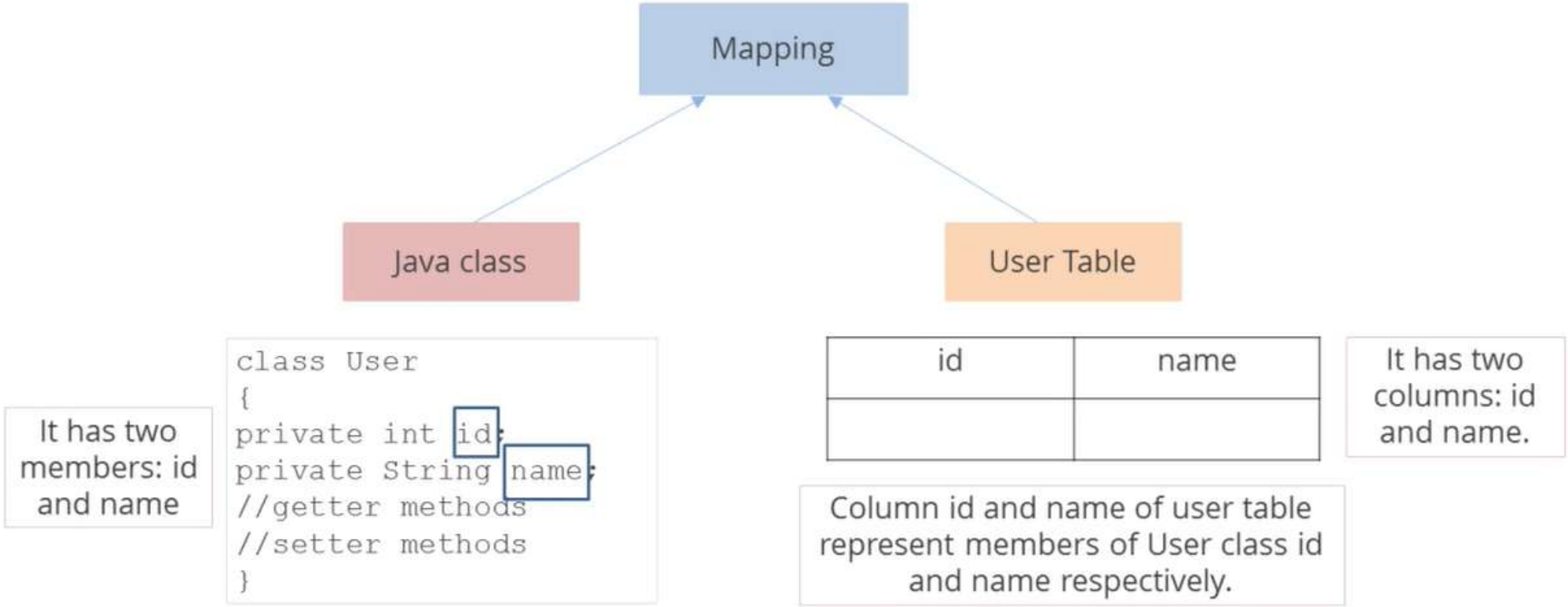
©Simplilearn. All rights reserved.

Advanced Java

Introduction to Hibernate

ORM

ORM refers to the automated (and transparent) persistence of objects in a Java application to the tables in a relational database, using metadata that describes the mapping between the objects and the database.



©Simplilearn. All rights reserved.

Advantages of using ORM

- It can significantly reduce the development time
- It involves writing less code
- It increases system performance
- You can support different database management systems by adopting ORM. It provides portability

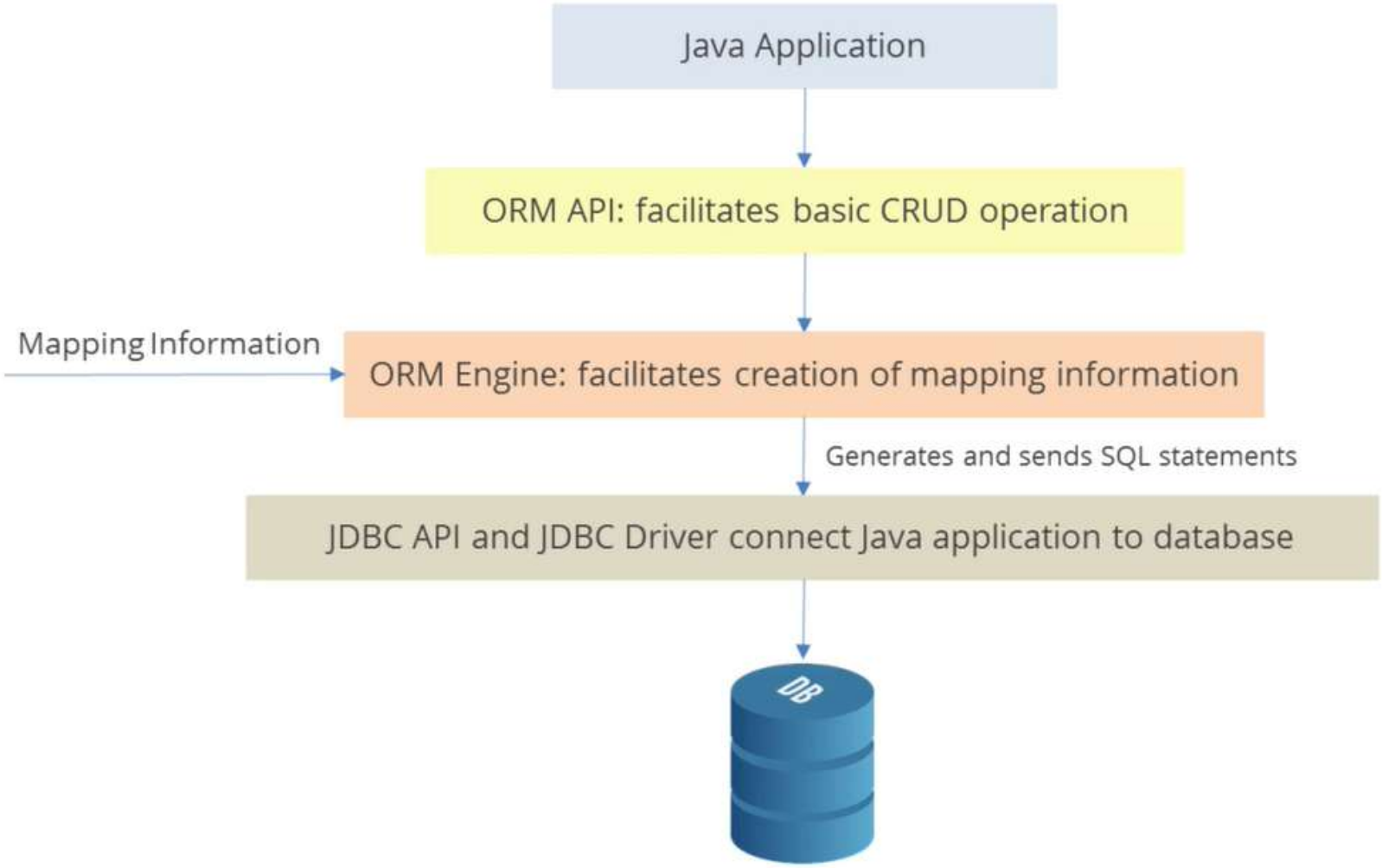
©Simplilearn. All rights reserved.

ORM: Features

- By mapping between logical business model and physical storage model, ORM implements domain model pattern
- It navigates object relational transparency
- ORM provides one way to translate query to appropriate database syntax
- It provides concurrency support
- It provides a host of services that enable developers to focus on business logic
- It provides cache management to reduce load on database
- Surrogate key, Identifier, and other key features can be automated in ORM

©Simplilearn. All rights reserved.

ORM Architecture



©Simplilearn. All rights reserved.

Hibernate as an ORM Framework

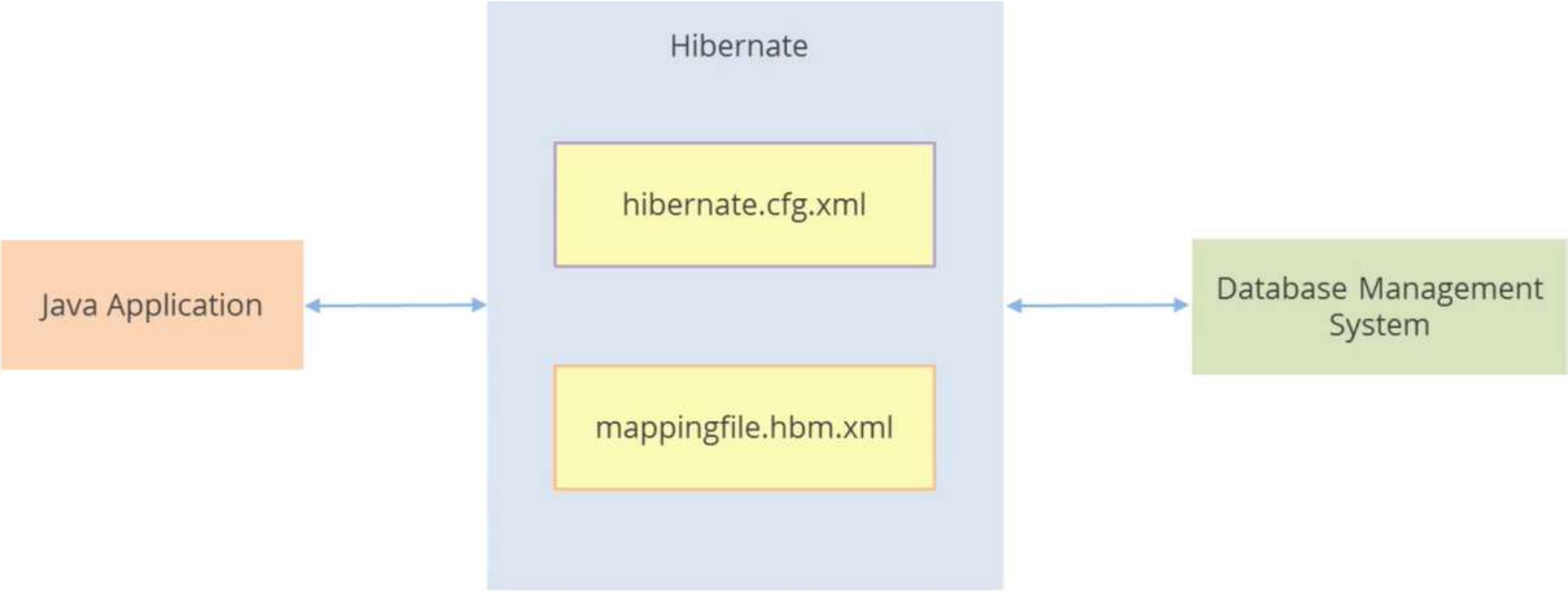
"As an Object/Relational Mapping (ORM) framework, Hibernate is concerned with data persistence as it applies to relational databases (via JDBC)."

©Simplilearn. All rights reserved.

Reference : http://hibernate.org/orm/

Hibernate Configuration Files

DBMS-specific details and mapping file details are specified in hibernate.cfg.xml and mappingfile.xml. Hibernate engine uses these files to generate DBMS-specific SQL syntax.



©Simplilearn. All rights reserved.

Features of Hibernate ORM

- Hibernate ORM provides its own native API, in addition to full **JPA (Java Persistence API)** supports.
- It maps Java POJO's (Plain Old Java Object) to relational database.
- It provides rich tool set.
- Performance: Fetch strategies, caching, byte code enhancement
- It is part of JBoss community.

You can download the latest version of Hibernate from: http://www.hibernate.org/downloads

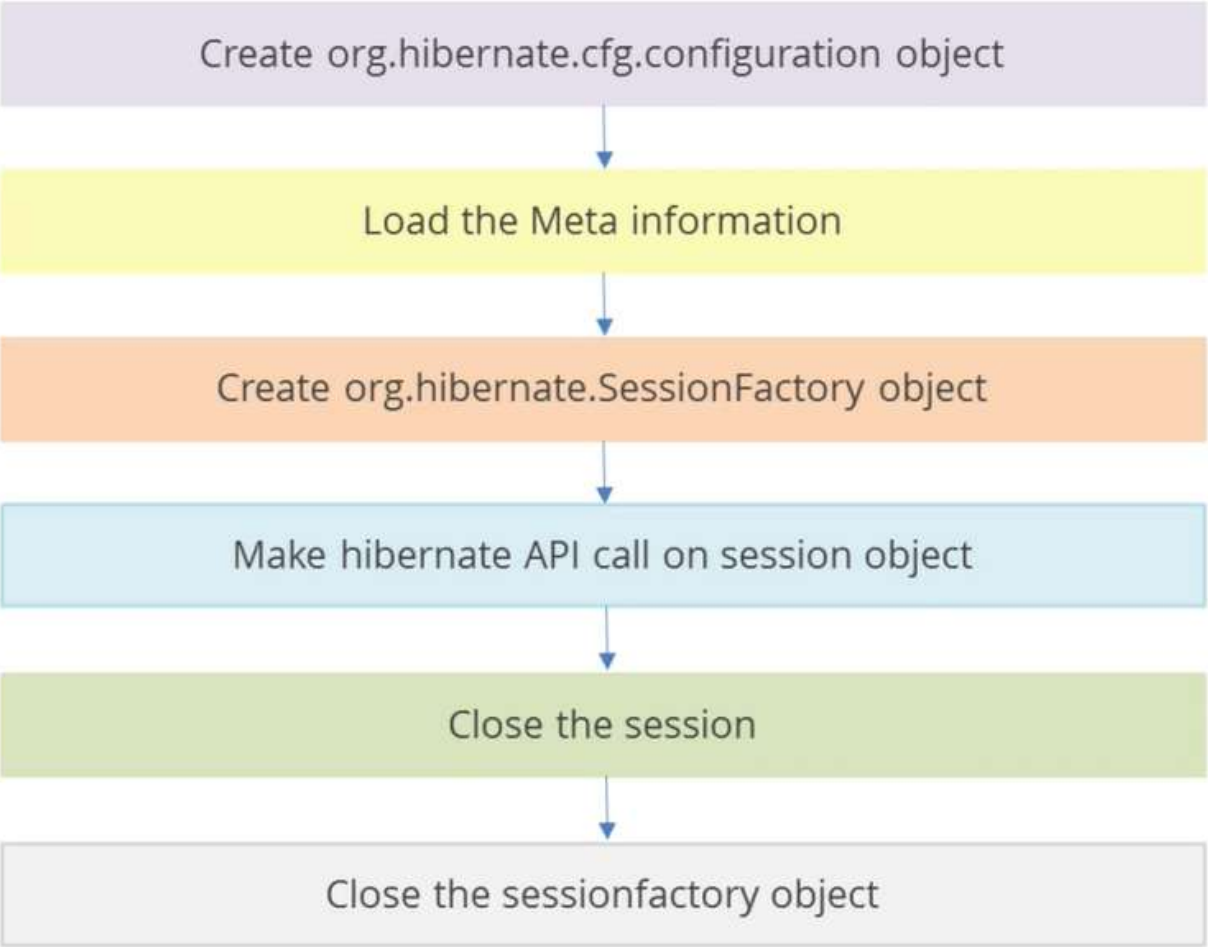
©Simplilearn. All rights reserved.

JPA (Java Persistence Application Programming Interface)

- Java EE 5.0 platform provides a standard persistence API named JPA. As part of the EJB 3.0 specification effort, it is supported by all major vendors of the Java industry.
- Hibernate model implements Java Persistence object relational API.
- Hibernate persistence provider can be used in any environment of Java platform: Java SE or Java EE.

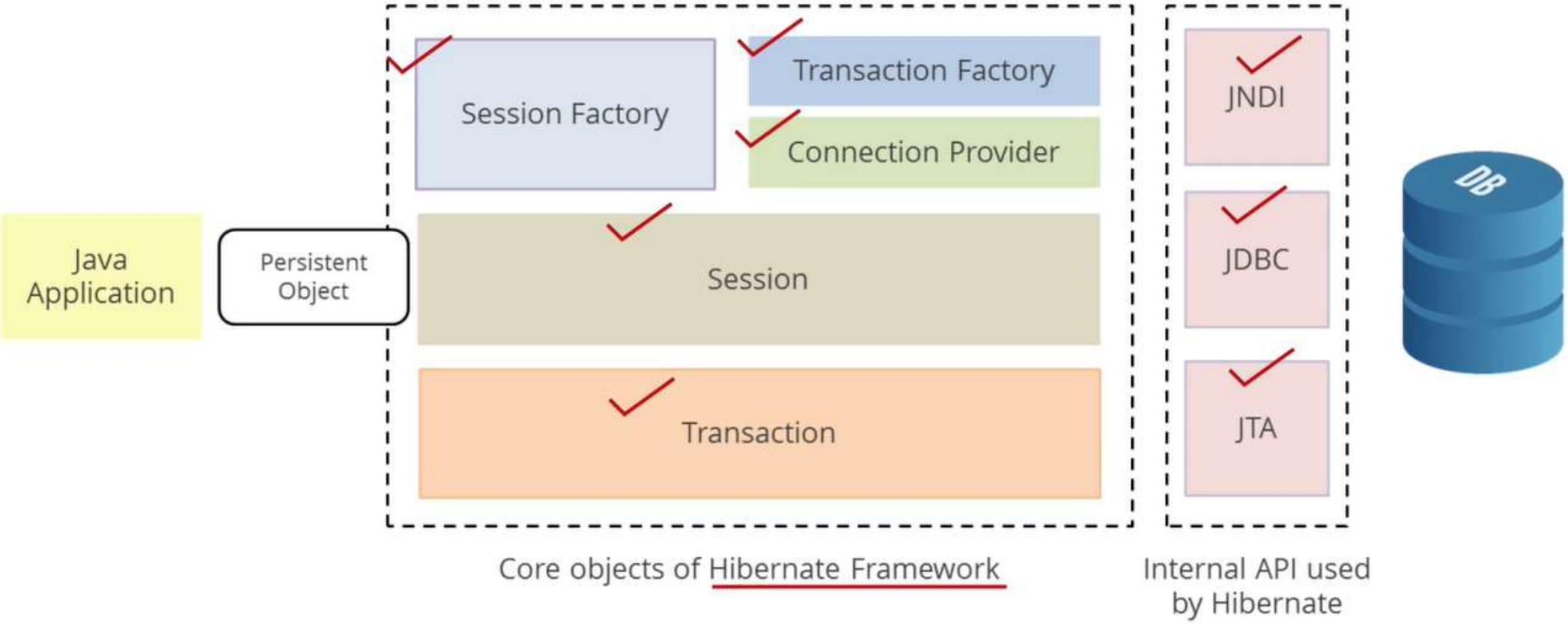
©Simplilearn. All rights reserved.

Hibernate Programming Model



©Simplilearn. All rights reserved.

Hibernate Architecture



©Simplilearn. All rights reserved.

Persistent Object

Session object is created within the Database Layer in every DAO method. It is known as persistence object.

It has three lifecycle states:

1. Transient: An object is transient if it has just been instantiated using the new operator, and it is not associated with a Hibernate Session.

2. Persistent: A persistent instance has a representation in the database and an identifier value. It might just have been saved or loaded; however, it is by definition in the scope of a Session.

3. Detached: A detached instance is an object that has been persistent, but its Session has been closed. The reference to the object is still valid, of course, and the detached instance might even be modified in this state.

©Simplilearn. All rights reserved.

Hibernate Core APIs

1. Configuration (org.hibernate.cfg): It gives two services in hibernate application:

   - Loads mapping file and configuration file into memory and makes them available to hibernate engine
   - Acts as factory to create the SessionFactory

2. SessionFactory (org.hibrnate.SessionFactory)

   - Hibernate (engine) implements SessionFactory interface
   - SessionFactory is one per DBMS (mostly one per application)
   - SessionFactory is Thread safe
   - SessionFactory is not a Singleton
   - It creates Session object, SessionFactory encapsulates, second level cache, connection pool, meta information cache, and pool of session

©Simplilearn. All rights reserved.

Hibernate Interface

3. Session (org.hibernate.Session)

- Hibernate engine implements Session interface
- Session object acts as persistent manager
- Session object is a light weight object
- It encapsulates connection and first-level cache
- Session object is not thread-safe
- In every dao method, Session object is created
- It is used for CRDD operation

4. Transaction (org.hibernate.Transaction)

- Hibernate engine implements this interface
- When database connection is created by hibernate, connection associated with session is in autocommit disable mode.
- Whenever any CRUD operation is performed, the changes will not be reflected in the database unless connection is maintained in auto-commit enabled mode.

©Simplilearn. All rights reserved.

Hibernate Interface

IdentifierGenerator

All the generator classes implement the **org.hibernate.id.IdentifierGenerator interface**.

The application programmer may create one's own generator classes by implementing the IdentifierGenerator interface.

Hibernate framework provides many built-in generator classes:

- assigned: Default, value has to be explicitly assigned to persistent object before persisting it. <generator class = assigned/>
- increment: It increments value by 1. It generates **short**, **int,** or **long** type identifier.
- native: It uses identity, sequence, or hilo, depending on the database vendor.
- sequence: It uses the sequence of the database. If there is no sequence defined, it creates a sequence automatically. For example, in case of Oracle database, it creates a sequence named HIBERNATE_SEQUENCE.
- hilo: It uses high and low algorithm to generate the id of type short, int, and long.
- identify: It is used in Sybase, My SQL, MS SQL Server, DB2, and SQL to support the id column. The returned id is of type short, int, or long.

©Simplilearn. All rights reserved.

Setting Up a Project with Hibernate

1. Create a Java project and add required jars from the required folder of hibernate downloads
2. Add ojdbc14.jar or mysqlconnector.jar depending on the SQL vendor you are connecting to
3. Create the POJO class and save it in src folder
4. Create the hibernate.cfg.xml file in src folder(configuration file) with specific RDBMS dialect

RDBMS	Dialect
Oracle (any version)	org.hibernate.dialect.OracleDialect
Oracle9i	org.hibernate.dialect.Oracle9iDialect
Oracle10g	org.hibernate.dialect.Oracle10gDialect
MySQL	org.hibernate.dialect.MySQLDialect
DB2	org.hibernate.dialect.DB2Dialect

5. Create the ClassName.hbm.xml file in src folder (mapping file)
6. Create the class that retrieves or stores the persistent object and save it in src folder
7. Run the application

©Simplilearn. All rights reserved.

Hibernate Annotation

You have Learned how Hibernate uses XML mapping file for the transformation of data from POJO to database tables and vice versa

Hibernate annotation is the newest way to define mappings without the use of XML file. You can use annotations in addition to or as a replacement of XML mapping metadata.

Creating POJO Class With Annotation

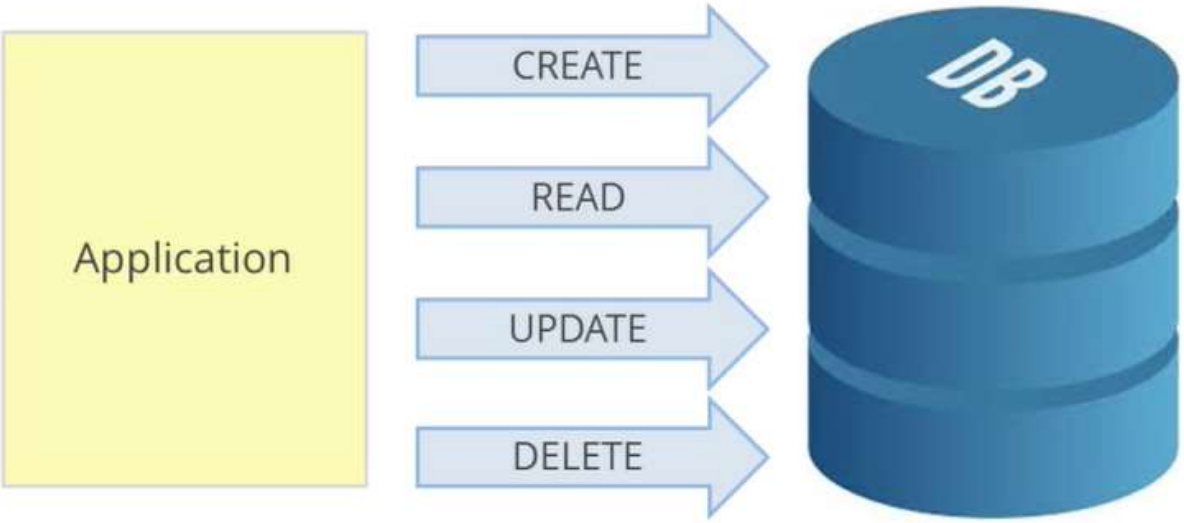
```
@Entity // Every persistent POJO class is an entity and is declared using the @Entity
annotation (at the class level):

@Table // @Table annotation, hibernate will use the class name as the table name by default

public class Student
 {
@Id  // @Id declares the identifier property of this entity.
    public int  id ;

private String name; // @Column annotation specifies the details of the column for this
property or field.
If // if it is  is not specified, property name will be used as the column name by default.

// setter and getter methods
}
```

Use POJO class student, hibernate.cfg.xml file, and annotations instead of hibernate mapping file.

CRUD Operation

The acronym CRUD stands for Create, Read, Update, and Delete

They are four basic operations that any data-driven application performs often



Hibernate also supports CRUD operation by means of Session interface.

©Simplilearn. All rights reserved.

Hibernate CRUD Operation

LIMITATIONS OF USING SESSION METHODS

1. Multiple persistent objects can't be retrieved
2. Multiple delete and update operations are not possible
3. Required criteria can't be expressed to perform some complex CRUD operation

To overcome the above limitations, the following are used:

a. HQL (Hibernate Query Language)

b. Criteria API

c. Native SQL (Structured Query Language)

©Simplilearn. All rights reserved.

HQL

HQL
CRITERIA API
NATIVE SQL

- It gives query language provided by Hibernate.
- It is same as SQL (Structured Query Language), but it doesn't depend on the table of the database. It uses classes.

©Simplilearn. All rights reserved.

Criteria

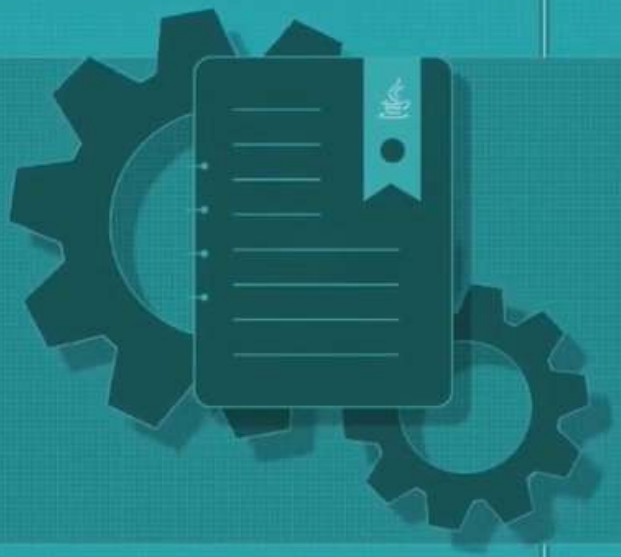
HQL
CRITERIA API
NATIVE SQL

- It is one way of querying the DB in the hibernate application
- It can be used for DML (Database Manipulation Language) operations
- Criteria API is an object oriented after native for HQL to read data from database
- Criteria API supports compile time checking for the query that we build, unlike HQL

Native SQL will be discussed in detail in the next lesson.

©Simplilearn. All rights reserved.

Press Esc to exit full screen



Advanced Java

Hibernate Queries and Relationships

Why HQL: Example

Analyze the following code to retrieving student record from student table:

```
// POJO class


class Student
{
private int id;
private String name;
// getter and setter
methods
}
```

```
hibernate.cfg.xml
```

```
public class Test
{
org.hibernate.cfg.Configuration cfg= new
org.hibernate.cfg.Configuration().config
ure("onetomany/hibernate.cfg.xml");


      StandardServiceRegistryBuilder
builder= new
StandardServiceRegistryBuilder().applySe
ttings(cfg.getProperties());
             SessionFactory
factory=cfg.buildSessionFactory(builder.
build());
             Session s=
factory.openSession();
             s.beginTransaction();
Person user=new Person();
Student s1=user.get(Person.class,1);

}
```

Student table

id	name
1	John

©Simplilearn. All rights reserved.

Why HQL?

To retrieve the student's data, we provide primary key value for that Student: user.get(Person.class,1);

If one table doesn't have a primary key, the records cannot be retrieved.

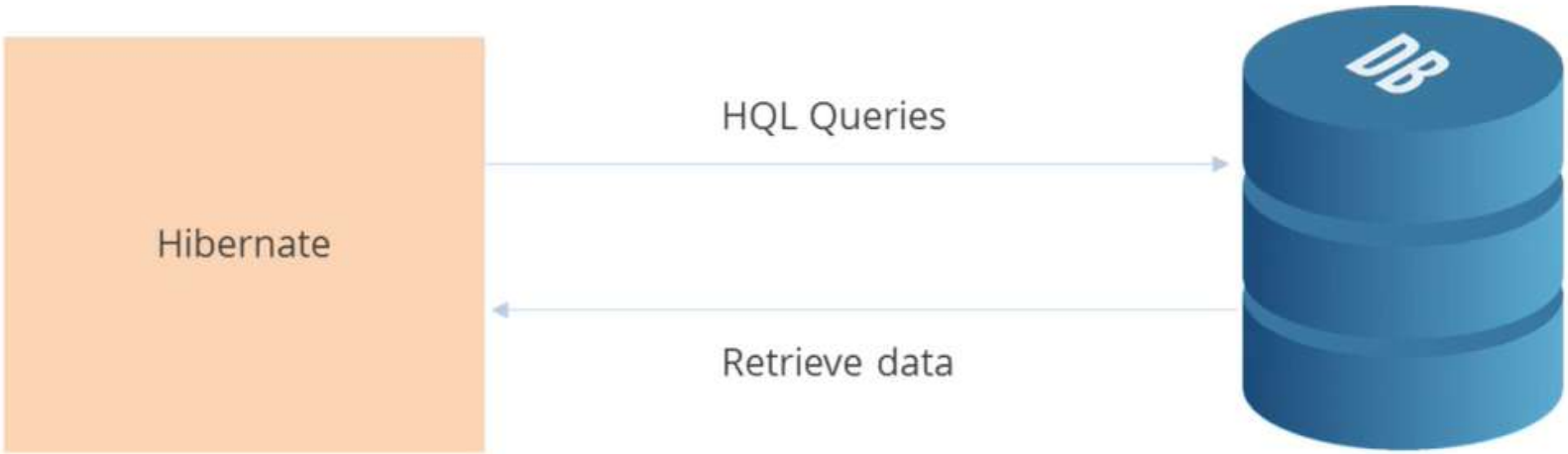
Hibernate provides its own language known as Hibernate Query Language, to solve such problems. The syntax is quite similar to database SQL language,

©Simplilearn. All rights reserved.

What Is HQL?

HQL is used to communicate with a database.

It is a Hibernate language for relational database management systems. HQL statements are used to perform tasks such as updating data on a database or retrieving data from a database.



©Simplilearn. All rights reserved.

HQL vs. SQL

<table>
<tr><th>HQL</th><th>SQL</th></tr>
<tr><td>It is related to hibernate framework</td><td>It is related to a specific database</td></tr>
<tr><td>HQL queries are object queries</td><td>SQL queries are table queries</td></tr>
<tr><td>Example:

<pre>class Student
{
String name;
int id;
}</pre>

<pre>HQL Query :- from Student
where Student is one class name</pre>

HQL Query: from Student, where Student is one class name</td><td>Example: Student table

<table><tr><td></td><td></td></tr><tr><td>Id</td><td>name</td></tr></table>

SQL Query: Select * from Student, where Student is one table name</td></tr>
</table>

©Simplilearn. All rights reserved.

Features of HQL

- HQL is object-oriented
- It understands notions like inheritance, polymorphism, and association
- Its queries are case-insensitive, unlike queries in Java classes.

Example: For HQL queries, SeLeCT is the same as sELEct
but for Java class org.hibernate.eg.STUDENT is not equal to org.hibernate.eg.Student.

©Simplilearn. All rights reserved.

Uses of HQL

Using HQL, we can create required CRUD operations:

- UPDATE
- DELETE
- SELECT

Using HQL, we can move table data to another table. However, we cannot insert data using HQL.

©Simplilearn. All rights reserved.

Creating HQL Query

1. Create configuration object and hibernate.cfg.xml file

```
StandardServiceRegistry standardRegistry = new
StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
```

2. Create SessionFactory object

```
Metadata metaData = new
MetadataSources(standardRegistry).getMetadataBuilder().build();
sessionFactory sf =
metaData.getSessionFactoryBuilder().build();
```

3. Open Session

```
Session sn = sf.openSession();
```

4. Create query

```
Query query = sn.createQuery("Here you can write HQL Query");
```

©Simplilearn. All rights reserved.

Methods of Query Interface

1. getReturnType() - Return the Hibernate types of the query result set
1. iterate() - Return the query results as an Iterator
1. list() - Return the query results as a List

©Simplilearn. All rights reserved.

Hibernate Parameter Building

Named parameters: This is the most common way followed by a parameter name (:example) to define a named parameter. Suppose we have Stock table with stockCode column.

setParameter: The setParameter discovers the parameter data type

```
String hql = "from Stock s where s.stockCode = :stockCode";
List result = session.createQuery(hql) .setParameter("stockCode",
"1111") .list();
```

setString: This informs Hibernate that the parameter date type is String

```
String hql = "from Stock s where s.stockCode = :stockCode";
List result = session.createQuery(hql) .setString("stockCode", "1111") .list();
```

setProperties: This can be used to pass an object to the parameter binding. Hibernate will automatically check the object's properties and match them with the colon parameter

```
Stock stock = new Stock(); stock.setStockCode("1111");
String hql = "from Stock s where s.stockCode = :stockCode";
List result = session.createQuery(hql) .setProperties(stock) .list();
```

©Simplilearn. All rights reserved.

HQL Syntax

1. from clause
2. Aliasing
3. Aggregate functions and select clause
4. where clause
5. Expressions
6. order by clause
7. Associations and Joins

©Simplilearn. All rights reserved.

Why Mapping?

Hibernate mapping is used for the following:

- To map inheritance hierarchy classes with the table of the database
- To meet the need of mapping collection elements of Persistent class in Hibernate
- To map dependent object as a component
- Hibernate mapping is used to map Java data type to SQL data type and vice versa. Hibernate uses Hibernate Type, which is different from Java data type and SQL data type

©Simplilearn. All rights reserved.

Hibernate Types

1. Value type has the following types:
   - Basic value Type: Hibernate provides a number of built-in basic types
   - Composite type: Components represent aggregation of value into a single Java type
   - Collection type: To map collection, Hibernate provides collection type
2. Entity type: Entities are classes that correlate with row in table
3. Custom type: Hibernate allows you to create your own value types

©Simplilearn. All rights reserved.

Types of Mapping in Hibernate

1. Inheritance Mapping
2. Association Mapping
3. Collection Mapping
4. Component Mapping

©Simplilearn. All rights reserved.

Press Esc to exit full screen

Advanced Java

Introduction to Spring

00:03 / 15:42

Shortcomings of the Java EE Architecture

- Java EE specifications don't help in understanding the implementation of the specification and the environment.
- The components in the architecture are written in a way that only the container can understand or handle them.

©Simplilearn. All rights reserved.

Why Spring?

- Spring is a framework that can help simplify the Java EE architecture by reducing the complexity involved in enterprise application development by using Java EE technologies directly.
- Its gives abstract layer for all the layers involved in Java EE architecture.

©Simplilearn. All rights reserved.

Spring Basics

- Spring was Invented by Rod Johnson
- The latest version of spring is Spring 4.0

Spring 4.0 can be downloaded from: http://maven.springframework.org/release/org/springframework/spring/4.2.0.RELEASE/

Link to download common loggings: http://www.java2s.com/Code/Jar/o/Downloadorgapachecommonsloggingjar.htm

©Simplilearn. All rights reserved.

What is Spring?

"Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure to ensure focus on your application."

Spring framework :

- Is an open source framework
- Is a lightweight framework
- Is developed in Java
- Eliminates common code
- Is from SpringOpensource community

©Simplilearn. All rights reserved.

What's New in Spring 4.0?

Spring 3.0 (Version before 4.0)	Spring 4.0
It does not give comprehensive support for JDK8.	Spring Framework 4.0 provides support for several Java 8 features. You can make use of lambda expressions and method references with Spring's call-back interfaces.
It provides Spring MVC Test Framework	Spring Framework 4.0 introduces several new features for use in unit and integration testing.
There is no support for Date-Time	JSR-310 provides Date-Time value types for Spring data binding and formatting.

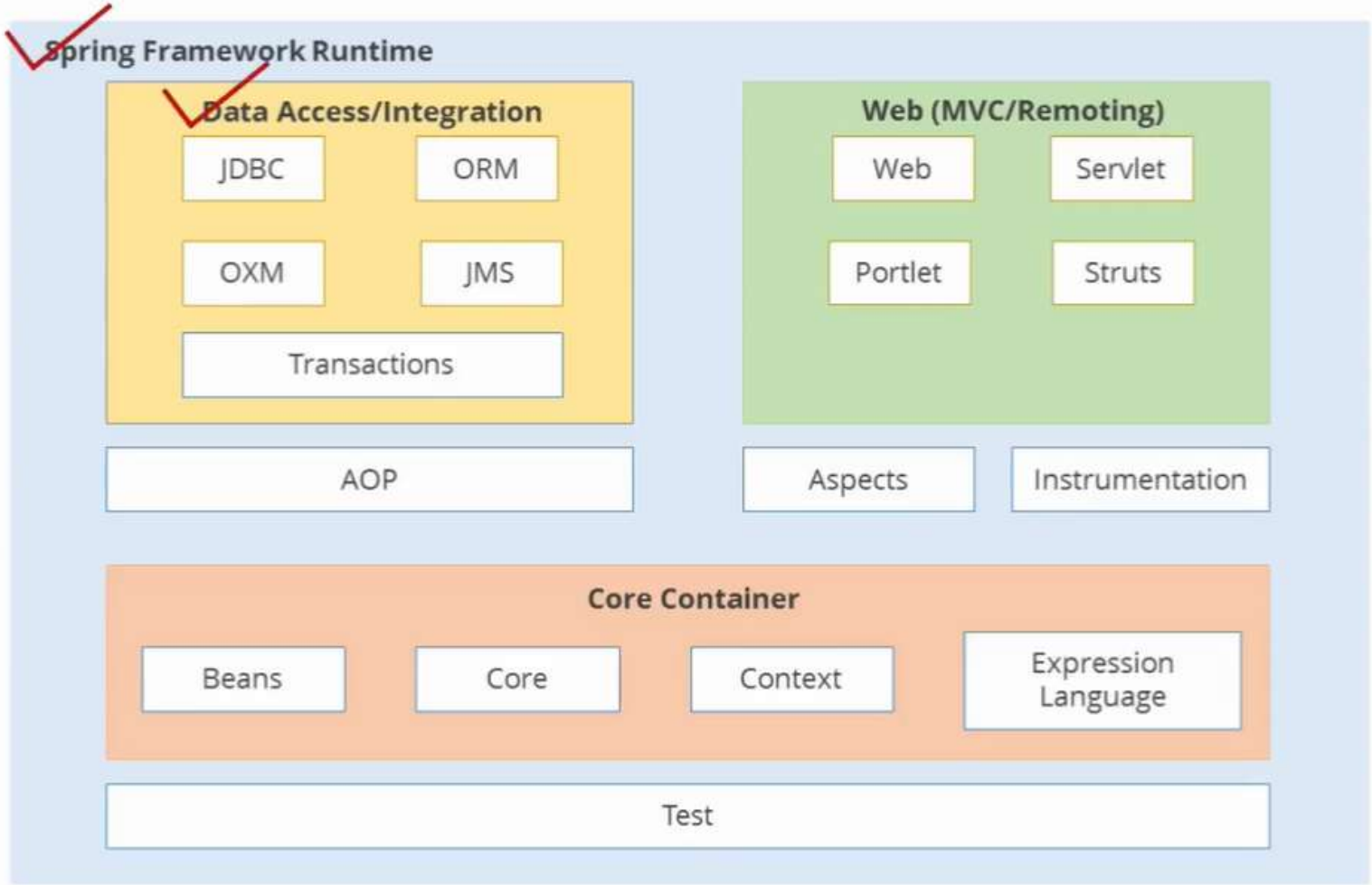
©Simplilearn. All rights reserved.

Features of Spring

- It supports multiple transaction environments such as JTA, Hibernate, and JDBC (Java Database Connectivity).
- It encourages good object-oriented design practices (e.g., interfaces, layers, separation of concerns).
- It provides IOC/DI (Inversion of Control/Dependency Injection) container.
- It supports annotation.
- POJOs (Plain Old Java Object) are faster than XML (Extensible Markup Language) Descriptor. Spring supports POJO.

©Simplilearn. All rights reserved.

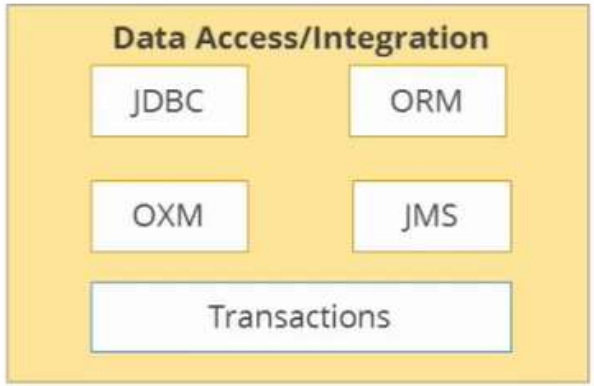
Spring Architecture



©Simplilearn. All rights reserved.

Spring Architecture

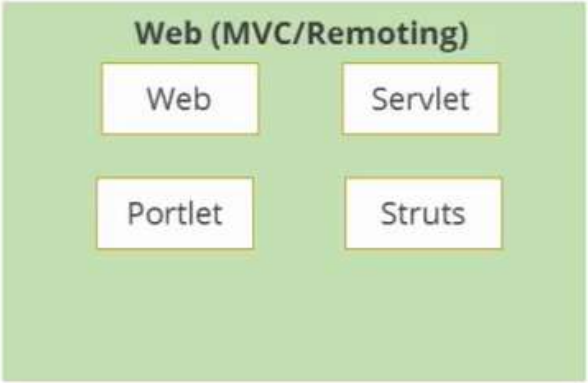
DATA ACCESS/INTEGRATION LAYER



- JDBC (Java Database Connectivity) provides a JDBC-abstraction layer that removes the need for traditional JDBC coding.
- ORM (Object XML Mappers) provides integration layers for popular object-relational mapping APIs.
- OXM provides an abstraction layer that supports Object/XML mapping implementation.
- Spring provides Java Message Service integration to simplify the use of the JMS API.
- Spring framework provides an abstract layer on top of different underlying transaction management APIs in transaction layer of spring.

©Simplilearn. All rights reserved.

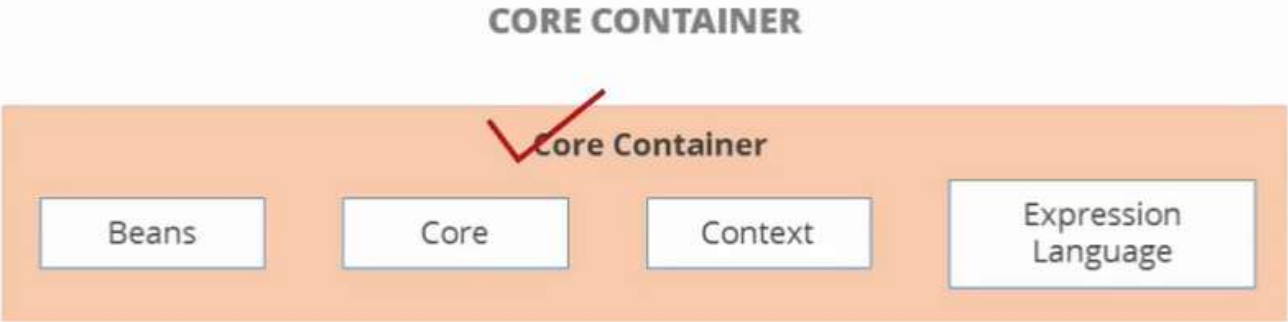
Spring Architecture



- Web: Web module provides basic web-oriented integration features.
- Web-Servlet: The Web-Servlet module contains Spring's model-view-controller (MVC) implementation.
- Web-Portlet modules: The Web-Portlet module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.
- Web-Struts: The Web-Struts module contains the support classes for integrating a classic Struts web tier within a Spring application.

©Simplilearn. All rights reserved.

Spring Architecture



- Beans: It provides BeanFactory (factory pattern). This allows you to decouple the configuration and specification of dependencies from your actual program logic.
- Core: It provides Inversion of Control and Dependency Injection features.
- Context: The Context module inherits its features from the Beans module and adds support for internationalization (using, for example, resource bundles), event-propagation, and resource-loading.
- Expression Language modules: These provide a powerful expression language for querying and manipulating an object graph at runtime.

©Simplilearn. All rights reserved.

Spring Architecture

MODULES

AOP (Aspect-oriented programming): It develops proxies for business layer and works as a solution for cross cutting problem.

Test: This module supports the testing of Spring components with JUnit or TestNG.

Design Pattern: Includes Singleton, MVC, Factory, etc.

©Simplilearn. All rights reserved.

What is IOC?

- IOC is a design principle that explains how object creation, dependency injection, object life cycle management, object destruction, etc., can be managed using an external entity.
- Spring Container is the external entity that implements the principles of IOC.

©Simplilearn. All rights reserved.

Spring IOC Container

It is the core container of the Spring Framework. It performs the following operations:

1. Create the objects
2. Wire them together
3. Configure them
4. Manage their lifecycle till destruction

Java classes that are handled by this container are called spring beans.

©Simplilearn. All rights reserved.

Uses of Spring IOC Container

Containers are used for the following functions:

1. Instantiation of object
2. Dependency injection
3. Initialization to read configuration metadata

©Simplilearn. All rights reserved.

Spring Framework's IOC Container Package

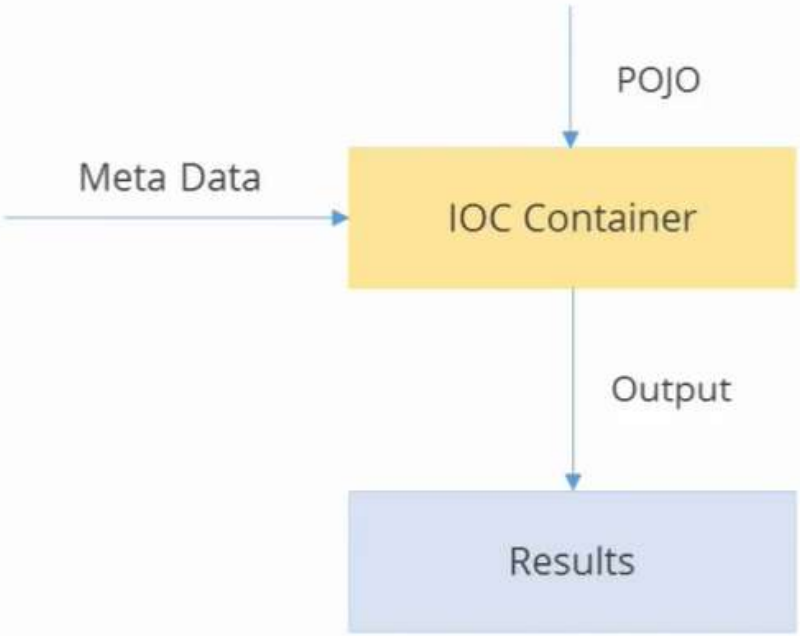
There are two IOC container packages:

- org.springframework.beans
- org.springframework.context packages

©Simplilearn. All rights reserved.

IOC Container: Working

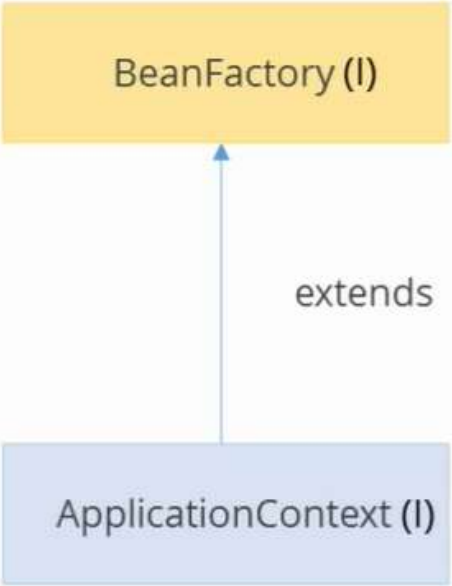
Spring container uses POJO classes and configures metadata to produce fully configured and executable beans.



©Simplilearn. All rights reserved.

Types of IOC Container

There are two types of Spring Container, which are based on the following interface:

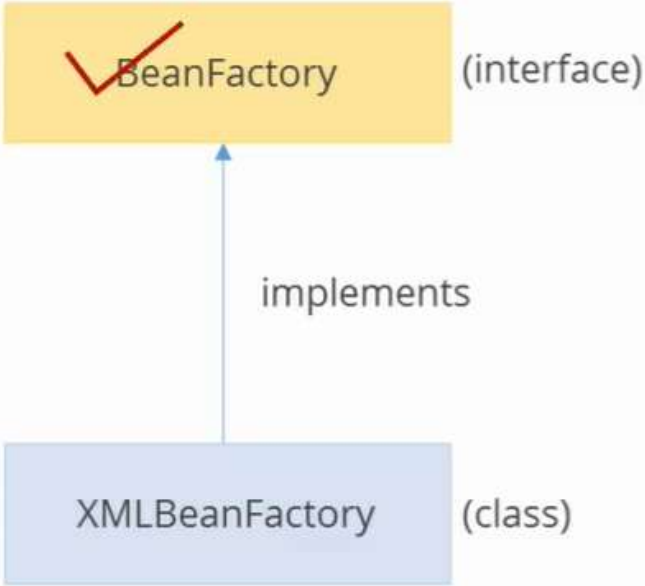


BeanFactory and ApplicationContext are the interface for container.

©Simplilearn. All rights reserved.

Implementation for XMLBeanFactory

XMLBeanFactory class gives the implementation for BeanFactory interface.



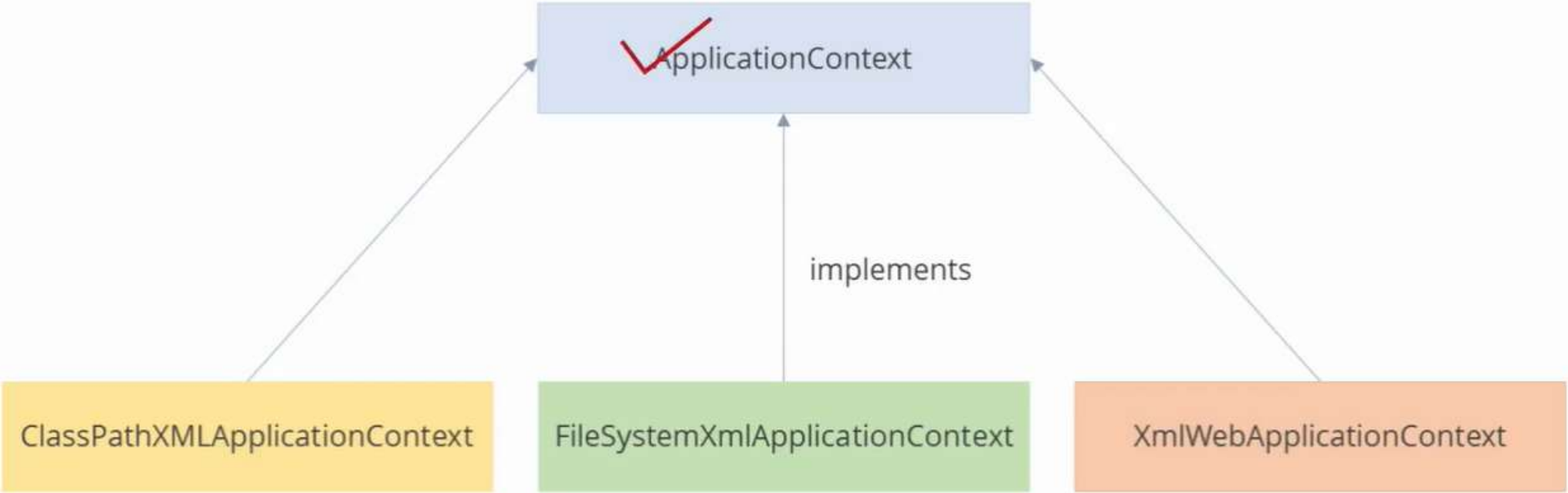
©Simplilearn. All rights reserved.

Containers based on BeanFactory Interface

- Spring bean is configured in configuration file. Beanfactory container loads that bean. It is a lightweight container.
- It manages the life cycle of spring bean.
- The lifecycle of Bean starts from calling the getBean(" ") method.
- It is generally used in the application where data volume and speed is significant, for example, mobile devices or applet-based applications.

©Simplilearn. All rights reserved.

Implementation for ApplicationContext



©Simplilearn. All rights reserved.

Container Based on ApplicationContext Interface

- If you use FileSystemXmlApplicationContext container, you need to pass the spring configuration file to this constructor.
- If you change the project location once, you have to change it every time.
- If you use ClassPathXmlApplicationContext, you don't have to specify the complete path of configuration file.
- If you change project location, your file will work without modification.
- If you use XmlWebApplicationContext, you can use Spring MVC module.
- Container based on ApplicationContext provides all the features of BeanFactory. In addition to this, it provides the following features:
  - Event-Handling
  - EJB integration
  - Transaction
  - Securities

©Simplilearn. All rights reserved.

Creating Bean Container using XmlBeanFactory Class

Creating Spring Bean Container involves creating an object of any implemented class in XmlBeanFactory.

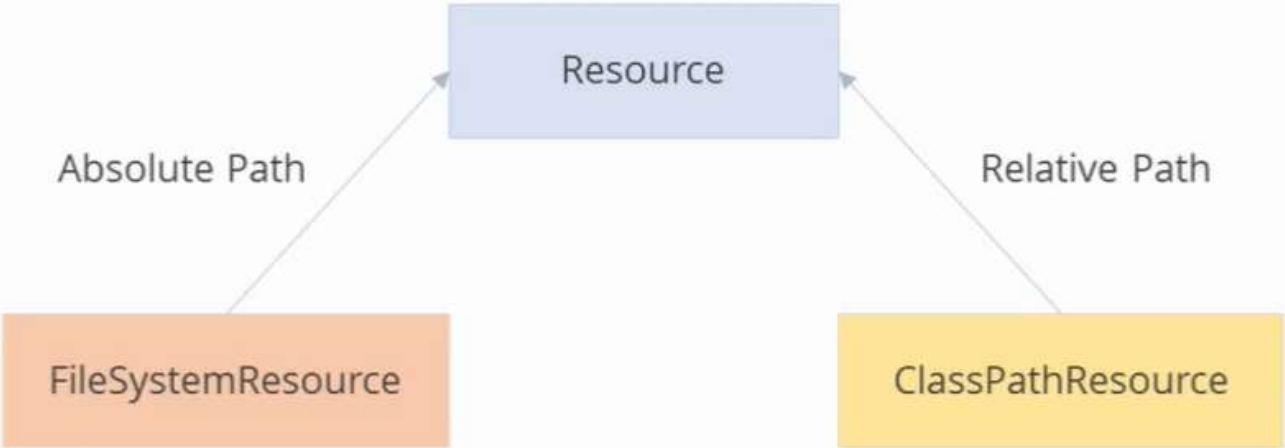
```
Resource resource=new ClassPathResource("applicationContext.xml");

OR

Resource resource=new ClassPathResource("path/applicationContext.xml");
BeanFactory factory=new XmlBeanFactory(resource);
```

©Simplilearn. All rights reserved.

Implementation of Resource Interface

Constructor of XmlBeanFactory class takes Resource object as an argument. This represents spring configuration file.

There are two implementations of Resource interface:



©Simplilearn. All rights reserved.

Creating SpringBean Container

Creating Spring Bean Container involves creating an object of any implemented class.

Creating Spring Bean Container using FileSystemXmlApplicationContext class

```
ClassPathXmlApplicationContext .

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

Creating Spring Bean Container using ClassPathXmlApplicationContext class

```
ClassPathXmlApplicationContext .

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

©Simplilearn. All rights reserved.

Creating Spring Applications in Eclipse

1. Create a Java project in eclipse and add a name to it: Click New → Click Project → Click Java Project, Add a name → Click Next

2. Add spring jar files: Right click on project → Click Buildpath → Click Library → Click on add external library and add spring libraries. You can also Click on user library and add spring jar to this user-defined library folder.

3. Create the class: Right click project → Create Java class in src folder → Add a name→ Write private properties inside Java class → Select properties and click on right → Click Source → Generate setter and getter method.

   This class is called spring bean. The syntax is provided below:

```
class ClassName
{
//public constructor
//private variable ;
//public getter and setter methods for variable;
}
```

©Simplilearn. All rights reserved.

Configuration File Tags

- <beans
- xmlns="http://www.springframework.org/schema/beans"
- xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
- xmlns:p="http://www.springframework.org/schema/p"
- xsi:schemaLocation="http://www.springframework.org/schema/beans
- http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
- <bean id="refernce_id_for_class_object" class="ClassName">
- </bean>
- </beans>

©Simplilearn. All rights reserved.

Types of DI

In Spring, DI is achieved in two ways:

- Setter injection: If dependency is injected to dependent object via setter method, it is known as setter injection.
- Constructor injection: If dependency is injected to dependent object via constructor, it is known as constructor injection.

©Simplilearn. All rights reserved.

Advantages of Using DI

- The dependency object can be changed dynamically.
- Provide loose coupling between dependent and dependencies.
- The codes appear clean and readable.
- DI and IOC make managing dependencies between objects simple.

©Simplilearn. All rights reserved.

Implementing Setter Injection

1. Create Spring bean class
2. Declare member variable
3. Define setter methods
4. Use <property> tag in bean configuration file
5. Provide values for member variable in configuration file

©Simplilearn. All rights reserved.

Implementing Constructor Injection

1. Create Spring bean class
2. Declare member variable
3. Define constructor, which will take variable as parameter
4. Use <constructor-arg> tag in bean configuration file
5. Provide values for member variable in configuration file

©Simplilearn. All rights reserved.

Spring Configuration File

SETTER INJECTION

```
<beans…>
<bean id=”  “  class=”  “ >
<property name=”nameOfClassMemberVariable” value =” “ >
</property>

or

<property name=”nameOfClassMemberVariable” value =” “ />

or

<property name =”nameOfClassMemberVariable”> <value> </value>
</property>
</bean>
</beans>
```

©Simplilearn. All rights reserved.

Spring Configuration File

SETTER INJECTION WHERE DEPENDENCY IS OBJECT TYPE

```
<beans…>
<bean id=”  “  class=”  “ >
<property name=”nameOfClassMemberVariable” ref =”dependency_class_id
“ > </property>

or

<property name=”nameOfClassMemberVariable” ref =”dependency_class_id
“ />

or

<property name =”nameOfClassMemberVariable”> <ref local/bean/pparent
”dependency_class_id” />
</property>
</bean>
</beans>
```

©Simplilearn. All rights reserved.

Implementing Constructor Injection

```
Spring configuration file
<beans…>
<bean id=”  “  class=”  “ >
<constructor-arg  value =” “ type=” data_type”> </constructor-arg >

or

<constructor-arg  value =” “ />

or

<constructor-arg>
<value> </value>
</constructor-arg>
</property>
</bean>
</beans>
```

©Simplilearn. All rights reserved.

Implementing Constructor Injection

IMPLEMENTATION OF CONSTRUCTOR INJECTION WHERE DEPENDENCY IS OBJECT TYPE

```
Spring configuration file
<beans…>
<bean id=”  “  class=”  “ >
<constructor-arg  ref=”  ”> </constructor-arg >

or

<constructor-arg  ref =” “ />

or

<constructor-arg>
<ref local/bean/parent=”providerRef”/>
</constructor-arg>
</property>
</bean>
</beans>
```

©Simplilearn. All rights reserved.

Injecting Collection Object

To pass multiple values to bean, we use collection injection. Spring offers four types of collection configuration elements:

Element	Description
<list>	inject list of values, allowing duplicates
<set>	inject set of values without duplication
<map>	inject name-value pair
<props>	inject name-value pair where name and value are both Strings

©Simplilearn. All rights reserved.

Configuration File for Injecting Collection Injection

```
<beans…>
<bean id=”  “  class=”  “ >
<property name=”nameOfClassMemberVariable” >
<list>
<value> …...    < /value>
<value>.......</value>
<value>......</value>
</list>
</property>
</bean>
</beans>
```

In case of set, replace <list> tag with <set> tag

©Simplilearn. All rights reserved.

Constructor Injection vs. Setter Injection

Constructor Injection	Setter Injection
In constructor injection, partial injection of dependencies is not possible. This is because we must pass all the arguments to call constructor.	In setter injection, partial injection of dependencies is possible
Constructor injection cannot override the values injected by setter.	Setter injection overrides the values injected by constructor.
For dependencies > 15, setter method is not recommended.	Ideal for dependencies > 15
Constructor injection makes bean class object immutable (cannot be changed).	Setter injection makes bean class object mutable (can be changed).

©Simplilearn. All rights reserved.

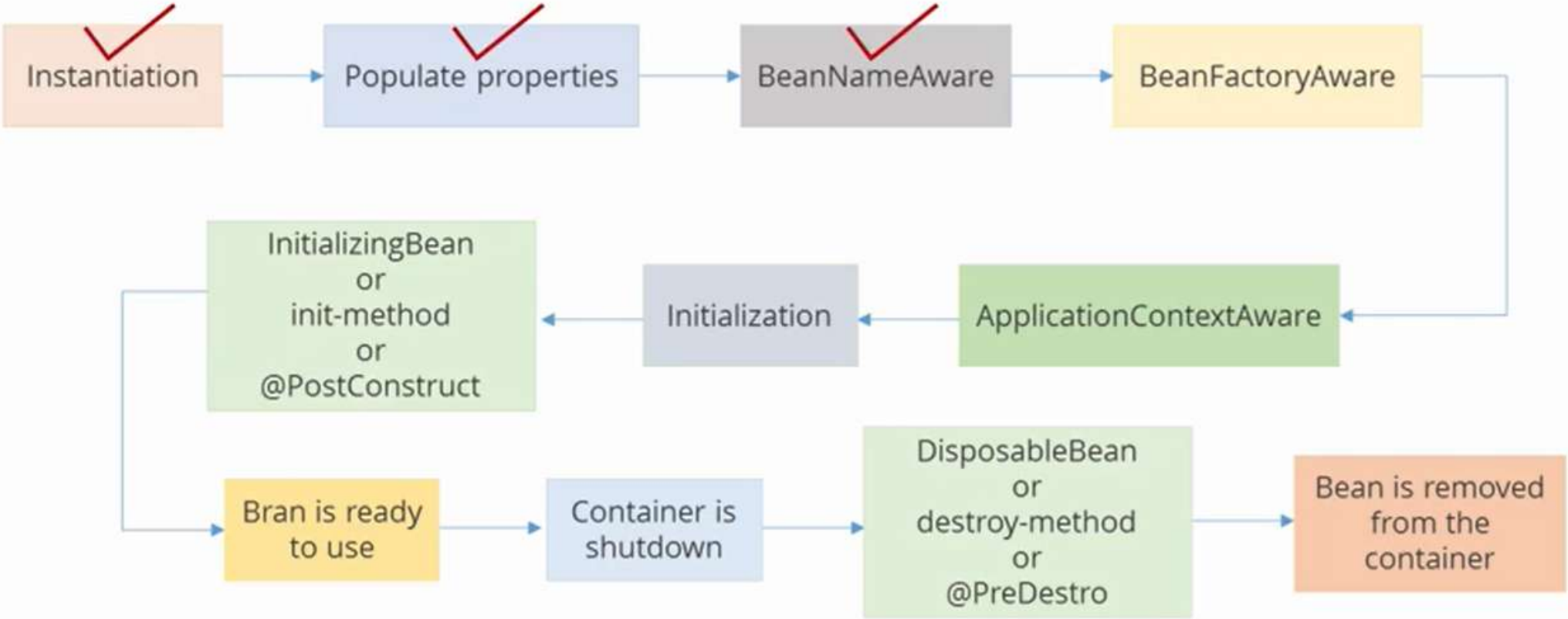
Spring Container

Recall: Springbean factory is responsible for managing the life cycle of beans created through spring container.

Spring container controls the life cycle of spring bean. It creates the object for bean, injects dependencies, performs initialization operation, calls business methods, and removes it from the container (when bean is no longer required).

©Simplilearn. All rights reserved.

SpringBean Lifecycle



©Simplilearn. All rights reserved.

simplilearn

SpringBean Lifecycle Components

1. Instantiate: Creation of object is done by spring container
2. Populate properties: Spring bean properties are populated by dependency injection
3. BeanNameAware: An interface used to override its method setBeanName(String) and get bean name
4. BeanFactoryAware: It is used to get container information in which spring bean is available
5. Initialization: There are three options for controlling Initialization;
   1. InitializingBean
   2. custom init()
   3. @PostConstruct annotations
6. Method Ready State: In this state, object is ready for use and business methods can be called.
7. Destruction: Three options for controlling Initialization
   1. DisposibleBean
   2. custom destroy()
   3. @PreDestroy annotations

©Simplilearn. All rights reserved.

Initialization

InitializingBean

It is an interface used to implement its method afterPropertiesSet(). Initialization can be written inside this method.

```
Syntax:

class SpringBeanExample implements InitializingBean
{
public void afterPropertiesSet()
{
}
}
```

InitializingBean

custom init()

@PostConstruct annotations

©Simplilearn. All rights reserved.

Initialization

custom init()

In the case of XML-based configuration metadata, you can use the init-method attribute to specify the name of the method.

```
Syntax:

<bean id="idName" class="SpringBeanExample" init-method="init"/>
```

InitializingBean	custom init()	@PostConstruct annotations

©Simplilearn. All rights reserved.

Initialization

@PostConstruct annotations

In the case of Annotation–based configuration metadata , you can use @PostConstruct annotation of JDK at the top of the method to specify the method as initialization method.

```
Syntax:

public class SpringExampleBean
{
@PostConstruct
public void myinit()
{
// do some initialization work
}
}
```

InitializingBean	custom init()	@PostConstruct annotations

©Simplilearn. All rights reserved.

What is Bean Wiring?

The Spring container can **autowire** relationships between collaborating beans without using <constructor-arg> and <property> elements. This is known as Bean Wiring.

©Simplilearn. All rights reserved.

AutoWiring

- AutoWiring refers to injecting the dependencies automatically
- It is used for object injection as it provides one way of injecting object dependency without<property> and <constructor-arg> injection.
- To implement autowiring, use autowire attribute of <bean> tag

©Simplilearn. All rights reserved.

Types of AutoWiring

1. constructor
2. byName
3. byType
4. autodetect

©Simplilearn. All rights reserved.

Bean Scope

When defining a <bean>, you have the option of declaring a scope for that bean. There can be two choices.

1) You want Spring to return the same bean instance each time one is needed

2) You want to force Spring to produce a new bean instance each time one is needed

©Simplilearn. All rights reserved.

Scope Attributes

Scope attribute has following values:

1) singleton: Create the configured spring bean only once in the container
2) prototype: Create the configured spring bean each time it is requested
3) request: Create the configure spring bean once per web request. It is applicable only in web module
4) session: Create the configured spring bean once per HttpSession. It is applicable only in web module
5) thread: Create the configured spring ben once per each thread

©Simplilearn. All rights reserved.

Advanced Java

Introduction to Spring AOP (Aspect-Oriented Programming)

Why AOP?

AOP is used in applications that have cross cutting **concerns,** that is, pieces of logic or code that are written in multiple classes/layers as per the requirements.

Common examples:

- Transaction Management
- Logging
- Exception Handling (especially when you may want to have detailed traces or have some plan of recovering from exceptions)
- Security aspects
- Instrumentation

©Simplilearn. All rights reserved.

Understanding Concern

A concern is a piece of code that performs a specific task. There are two types of concerns:

1) "Core" concerns are codes used for business logic.

2) "Cross concerns" are functions that are conceptually separate from the application's business logic but affect the entire service layer.

   Example: logging, auditing, declarative transactions, security, caching

Aspect-Oriented Programming provides one way to dynamically decouple core concern from cross cutting concern.

©Simplilearn. All rights reserved.

What Is AOP?

1) It is one of the basic components of Spring framework.

2) The main idea of AOP (Aspect-Oriented Programing) is to isolate the cross cutting concerns from the application code, thereby modularizing them as a different entity.

3) Services layer deals with two types of concerns: Core concerns and cross cutting concerns.

©Simplilearn. All rights reserved.

Uses of AOP

- Spring AOP provides declarative enterprise services such as declarative transaction management.
- It allow users to implement custom aspects.
- Spring AOP is used to track user activity in large business applications.
- It uses Proxy as a mechanism to implement cross cutting concerns in a non-intrusive way.

©Simplilearn. All rights reserved.

AOP Terminologies

- Aspect: It is a feature or functionality that cross-cuts over objects. It has a set of APIs (Application Programming Interface) that provides cross cutting requirements.
- JoinPoint: It defines the various execution points where an Aspect can be applied.
- Pointcut: A Pointcut tells us about the Join Points where Aspects will be applied.
- Advice: An advice provides concrete code implementation for the Aspect.
- Target: It is a business object that comprises of Aspect business core concern.

©Simplilearn. All rights reserved.

AOP Terminologies

- Weaving: It represents a mechanism of associating cross cutting concern to core concern dynamically.
- Proxy: Proxy is an object produced through weaving. There are two types: static proxy and dynamic proxy.
  - In static proxy, static method is used to develop and maintain proxy classes for each business method.
  - In dynamic proxy, proxy is developed at run time.
- Weaver: Code that produces proxies
- Proxy design pattern: Actual object (business object) is wrapped into another object known as proxy and substitutes that object in place of actual object

We will discuss JoinPoint, Pointcut, and Proxy in this lesson.

©Simplilearn. All rights reserved.

AOP Terminologies

Pointcut

Pointcut defines where exactly the Advices have to be applied in various Join Points.

Generally, it acts as a Filter for the application of various Advices into the real implementation.

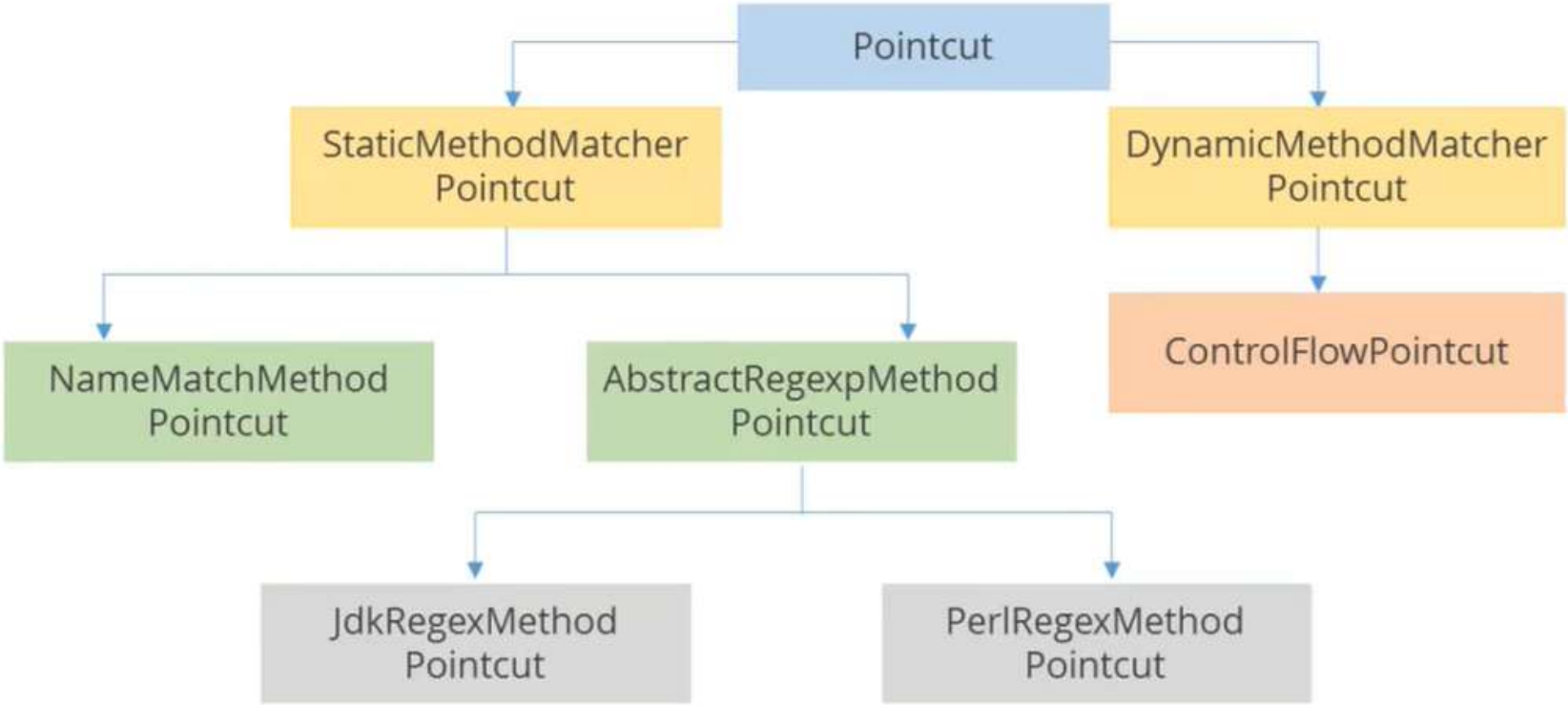
©Simplilearn. All rights reserved.

AOP Terminologies

Pointcut: TYPES

Springs defines two types of Pointcut:

1. Static: It verifies whether the join point has to be advised or not. It does this once the result is catched @reused.
2. Dynamic: It verifies every time as it has to decide the Join Point based on the argument passes to method call.



©Simplilearn. All rights reserved.

AOP Terminologies

StaticMethodMatcherPointcut

StaticMethodMatcherPointcut has two types of Pointcuts:

1. NameMatchMethodPointcut: It interrupts a method via 'pointcut' and 'advisor'

2. AbstractRegexpMethodPointcut: Besides matching method by name, it can match the method's name by using regular expression pointcut.

   This pointcut is used to verify join point based on pattern of the method name instead of name.

©Simplilearn. All rights reserved.

AOP Terminologies

AbstractRegexpMethodPointcut

1. JdkRegexMethod Pointcut: It is represented by org.springframework.aop.JdkRegexpMethodPointcut
2. PerlRegexMethodPointcut: It is represented by org.springframework.aop.support.Perl5RegexpMethodPointcut

©Simplilearn. All rights reserved.

AOP Terminologies

DynamicMethodMatcherPointcut

This pointcut is used to verify the context from which business method call is made. If method call is made in specific flow, it will be advice; otherwise, it won't.

©Simplilearn. All rights reserved.

AOP Terminologies

POINTCUT INTERFACE

Pointcut is an interface; it is represented by org.springframework.aop.Pointcut.

```
public interface Pointcut
{
ClssFilter getClassFilter()
MethodMatcher getMethodMatcher()
}
```

The getClassFilter() method returns a ClassFilter object which determines whether the classObject argument passed to the matches() method should be considered for giving Advices

©Simplilearn. All rights reserved.

AOP Terminologies

Joinpoint

A Joinpoint is a candidate point in the **Program Execution** of the application where an aspect can be plugged in.

This point could be a method being called, an exception being thrown, or even a field being modified.

These are the points where your aspect's code can be inserted into the normal flow of your application to add new behavior.

A pointcut defines the Joinpoints where associated Advice should be applied.

©Simplilearn. All rights reserved.

AOP Terminologies

Joinpoint METHODS

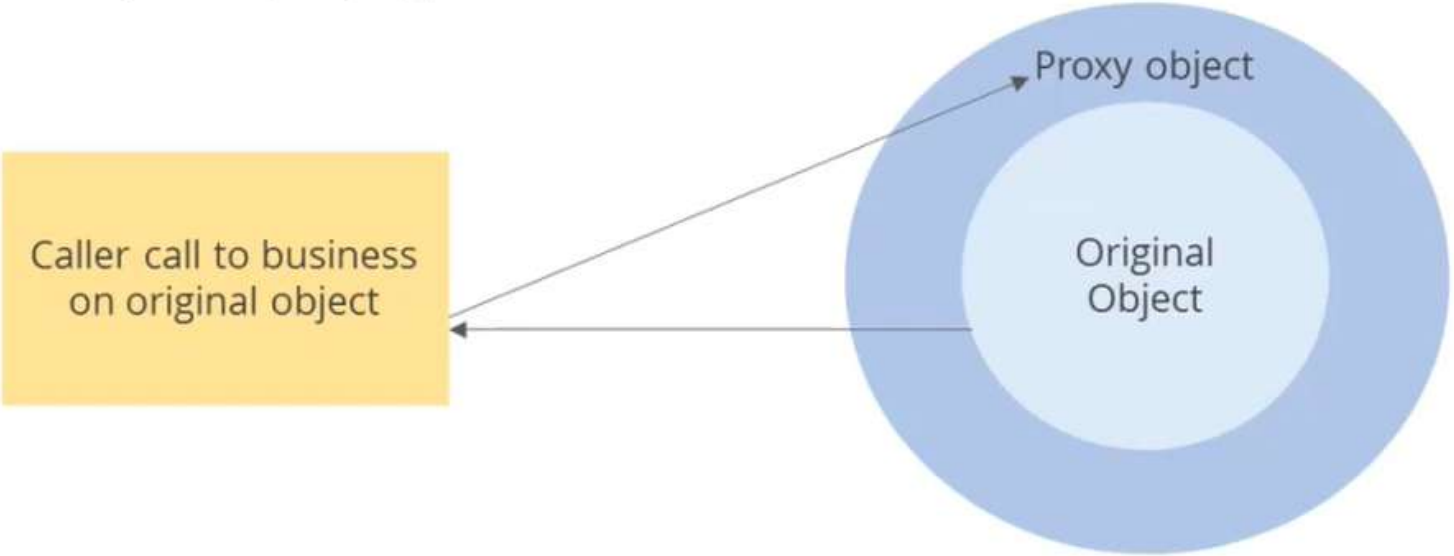
Return Type	Method Name	Description
AccessibleObject	getStaticPart()	Return the static part of Joinpoint
Object	getThis()	Return the object that holds the current Joinpoint static part
Object	proceed()	Proceed to the next interceptor in the chain

©Simplilearn. All rights reserved.

How Does AOP Work?

Proxy interrupts the call made by caller to original object. It will have chance to decide whether and when to pass on the call to original object. In the meantime, any additional code can be executed.

Spring uses the dynamic proxy approach.



A dynamic proxy class is a class that implements a list of interfaces specified at runtime so that a method invocation through one of the interfaces on an instance of the class will be encoded and dispatched to another object through a uniform interface.

©Simplilearn. All rights reserved.

AOP Terminologies

Proxy

- In proxy pattern, a class represents functionality of another class. This type of design pattern is a structural pattern.
- It creates an object that has original object to interface its functionality to outer world. Proxy design pattern is used when we want to provide controlled access to a functionality.
- It is also used to save on the amount of memory used. Similarly, if you want to control access to an object, the pattern becomes useful.

©Simplilearn. All rights reserved.

Ways to Generate Dynamic Proxy

1. jdk approach: When the business class implements interface, jdk creates proxy for the business object
2. cglib approach: cglib.jar is used when a business class fails to implement any interface, and jdk is unable to create proxy

©Simplilearn. All rights reserved.

AOP Terminologies

Advice

Advice refers to the actual implementation code for an Aspect. Spring supports Method Aspect.

Different types of aspects available in Spring:

1. Before advice: It executes before a Joinpoint but does not have the ability to prevent execution flow proceeding to the Joinpoint.
2. After advice: It is executed after business method call.
3. Around advice: It surrounds a Joinpoint such as a method invocation. It can perform custom behavior before and after method invocation.
4. Throws Advice is executed if actual method throws exception.

©Simplilearn. All rights reserved.

AOP Terminologies

Advice

Advice refers to the actual implementation code for an Aspect. Spring supports Method Aspect.

Different types of aspects available in Spring:

1. Before advice: It executes before a Joinpoint but does not have the ability to prevent execution flow proceeding to the Joinpoint.
2. After advice: It is executed after business method call.
3. Around advice: It surrounds a Joinpoint such as a method invocation. It can perform custom behavior before and after method invocation.
4. Throws Advice is executed if actual method throws exception.

©Simplilearn. All rights reserved.

AOP Terminologies

Steps to Configure Advice

1. To configure these advices, we have to depend on ProxyFactoryBean. This Bean is used to create Proxy objects for the implementation class along with the Advice implementation.

2. The property 'proxyInterface' contains the Interface Name for which the proxy class has to be generated.

3. The 'interceptorNames' property takes a list of Advices to be applied to the dynamically generated proxy class.

4. Finally, the implementation class is given in the 'target' property.

©Simplilearn. All rights reserved.

Before Advice

- It is used to intercept before the method execution starts. For example, a system may need to perform some logging operations before allowing users to access resources
- Whenever caller calls the business method, all before advice code is executed prior to the business method.

If exception occurs in the method, afterReturning() method will never be called.

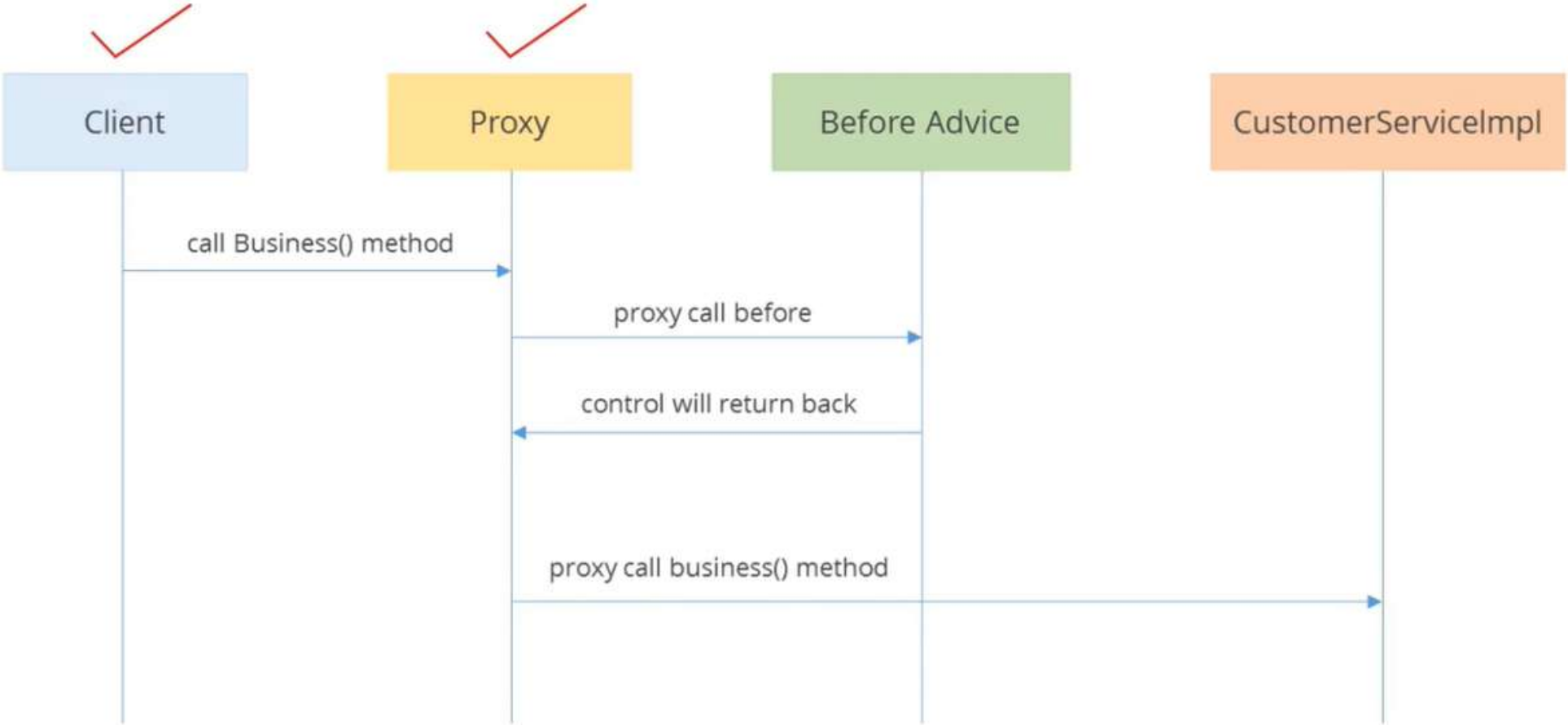
©Simplilearn. All rights reserved.

Configuring Before Advice

1) Develop a business interface and its implemented class
2) Develop a class that implements AfterReturningAdvice interface
3) Override its afterReturning() method[(Method method, Object[] args, Object target)]. Cross cutting code is written here.
   - afterReturningMethod method - target method to be invoked
   - Object[] args - various arguments that are passed on to the method
   - Object target - target reference to the object that is calling the method
4) Get a proxy
5) Make a method call

©Simplilearn. All rights reserved.

Steps Diagram: Before Advice



©Simplilearn. All rights reserved.

After Advice

- It is used to intercept after the method execution. For example, a system needs to perform some delete operations after logging out.
- After Advice is executed after business method.

©Simplilearn. All rights reserved.

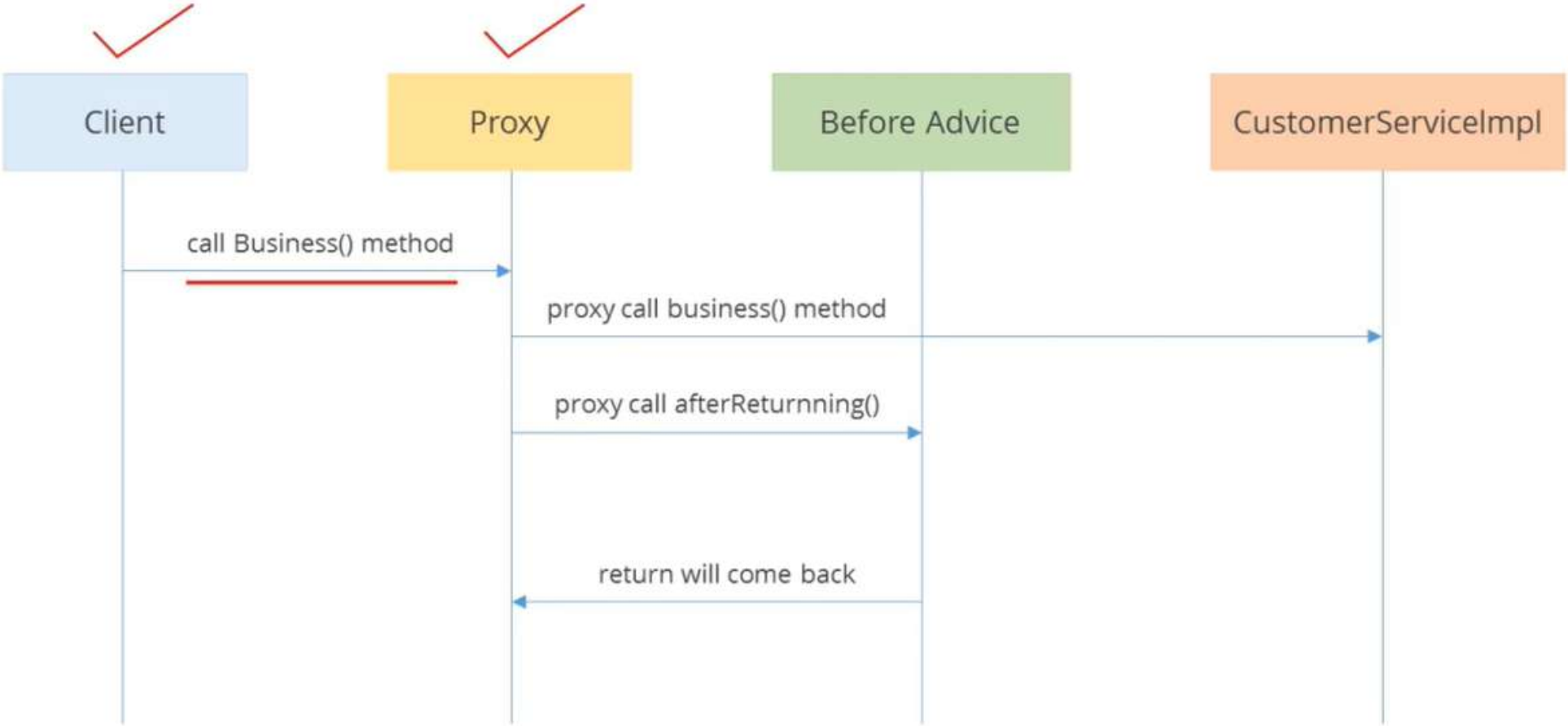
Configuring After Advice

1. Develop a business interface and its implemented class
2. Develop a class that implements AfterReturningAdvice interface
3. Override its afterReturning (Method method, Object[] args, Object target) method. Cross cutting code is written here.
   - Method method - target method is invoked
   - Object[] args - various arguments that are passed to the method
   - Object target - target reference to the object that is calling the method
4. Get a proxy
5. Make a method call

If exception occurs in the method, afterReturning() method will never be called.

©Simplilearn. All rights reserved.

Steps Diagram: After Advice



©Simplilearn. All rights reserved.

Around Advice

- It can change the return argument of the method call.
- It provides finer control regarding whether the target method has to be called or not.
- It can be performed before business logic or after business logic.

©Simplilearn. All rights reserved.

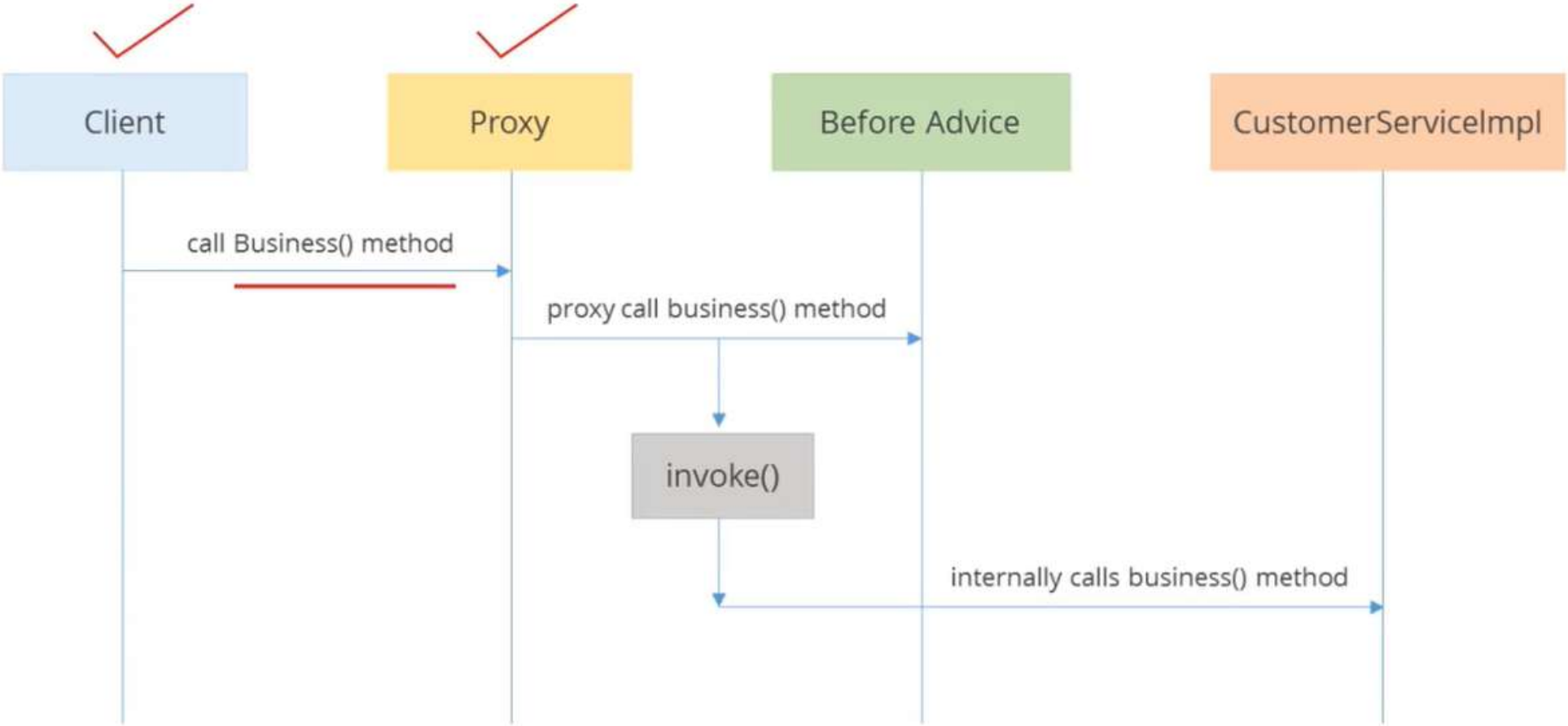
Configuring Around Advice

- Develop a class that implements MethodInterceptor interface
- Override its invoke method. Write cross cutting code
- Get a proxy
- Make a method call
- Parameter of invoke method()

```
public Object invoke(MethodInvocation methodInvocation)
```

©Simplilearn. All rights reserved.

Steps Diagram: Around Advice



©Simplilearn. All rights reserved.

Throws Advice

When an exception happens during the execution of a method, Throws Advice can be used through the means of org.springframwork.aop.ThrowsAdvice to handle the exception.

- This advice is executed only when exception is raised
- Advice has to implement org.springframework.aop.ThrowsAdvice. It is just a marker interface
- Exception handling code is considered cross cutting code and can be done for entire service layer in Throwsadvice with different kinds of exception parameters.

©Simplilearn. All rights reserved.

Configuring Throws Advice

1. Develop a business interface and its implemented class
2. Develop a class that implements ThrowsAdvice interface
3. Override its afterThrowing method. Cross cutting code is written here.
4. Get a proxy
5. Make a method call
6. Throws Advice can take any of the following forms:

```
public void afterThrowing(Exception ex)

public void afterThrowing(Method method,Object[] args,Object target,Exception
exception)
```

©Simplilearn. All rights reserved.

AspectJ Based AOP

There are two ways to use Spring AOP AspectJ implementation:

1. By annotation
2. By XML configuration

AspectJ jar files (aspectjrt.jar, aspectjweaver.jar, aspectj.jar, aopalliance.jar) can be downloaded from https://eclipse.org/aspectj/downloads.php

©Simplilearn. All rights reserved.

Annotation Based AOP

1. @Before declares the before advice. It is applied before calling the actual method.
2. @After declares the after advice. It is applied after calling the actual method and before returning result.
3. @Around declares the around advice. It is applied before and after calling the actual method.
4. @AfterReturning declares the after returning advice. It is applied after calling the actual method and before returning result. But, you can get the result value in the advice.
5. @AfterThrowing declares the throws advice. It is applied if actual method throws exception.
6. The @Pointcut annotation is used to define the pointcut.
   - @Pointcut("execution(public * *(..))") is applied on all the public methods.
   - @Pointcut("execution(public Operation.*(..))") is applied on all the public methods of Operation class.
   - @Pointcut("execution(* Operation.*(..))") is applied on all the methods of Operation class.
   - @Pointcut("execution(public Employee.set*(..))") is applied on all the public setter methods of Employee class.
   - @Pointcut("execution(* Operation.*(..))")
   - private void doSomething() {}
   - The name of the pointcut expression is doSomething().
7. @Aspect declares the class as aspect

@Before, @After, and @Around will be covered in the scope of this lesson.

©Simplilearn. All rights reserved.

@Before

Let's look at Perform and TrackPerformance classes. Assume that Perform class contains actual business methods.

The AspectJ Before Advice is applied before the actual business logic method USING @Before advice.

Perform class

```
public  class Perform{
    public void msgone{System.out.println("msg method invoked");}
    public int msgtwo(){System.out.println("m method invoked");return 2;}
    public int msgthree(){System.out.println("k method invoked");return 3;}
}
```

TrackPerformance class

```
@Aspect
public class TrackPerformance{
    @Pointcut("execution(* Operation.*(..))")
    public void k(){}//pointcut name

    @Before("k()")//applying pointcut on before advice
    public void myadvice(JoinPoint jp)//it is advice (before advice)
    {
        System.out.println("Before advice is called");
            }
}
```

©Simplilearn. All rights reserved.

@After

For @After annotation, use the same Person class and TrackPerformance.

Person class is the same, but @After annotation in TrackPerformance class is used in this case.

TrackPerformance class

```
@Aspect
public class TrackPerformance{
    @Pointcut("execution(* Operation.*(..))")
    public void k(){}//pointcut name

    @After("k()")//applying pointcut on before advice
    public void myadvice(JoinPoint jp)//it is advice (before advice)
    {
        System.out.println("After advice is called");
            }
}
```

@After("k()")//apply Pointcut on after advice

©Simplilearn. All rights reserved.

XML Configuration Based AOP

It uses the following XML elements to define advice:

- **aop:before:** It is applied before calling the actual business logic method.
- **aop:after:** It is applied after calling the actual business logic method.
- **aop:after-returning:** It is applied after calling the actual business logic method. It can be used to intercept the return value in advice.
- **aop:around:** It is applied before and after calling the actual business logic method.
- **aop:after-throwing:** It is applied if actual business logic method throws exception.

©Simplilearn. All rights reserved.

Advice vs. Aspectj

Advice	Aspectj
Spring AOP is best used for application-specific tasks such as security, logging, transactions, etc.	AspectJ contains friendly design-patterns.
This is proxy-based AOP. You can use method-execution Pointcut only.	This supports all Pointcuts.
There can be a little runtime overhead.	There is less runtime overhead than that of Spring AOP.
It needs Spring jar files only.	You need extra build process with AspectJ Compiler or have to setup LTW (load-time weaving).

©Simplilearn. All rights reserved.

Press Esc to exit full screen

Advanced Java

Spring JDBC Implementation in an Application

Spring JDBC /DAO (Data Access Object)

- It is used to build Data Access Layer of enterprise application.
- It is a mechanism to connect to the database and execute SQL queries.
- The database-related logic has to placed in separate classes called DAO classes.

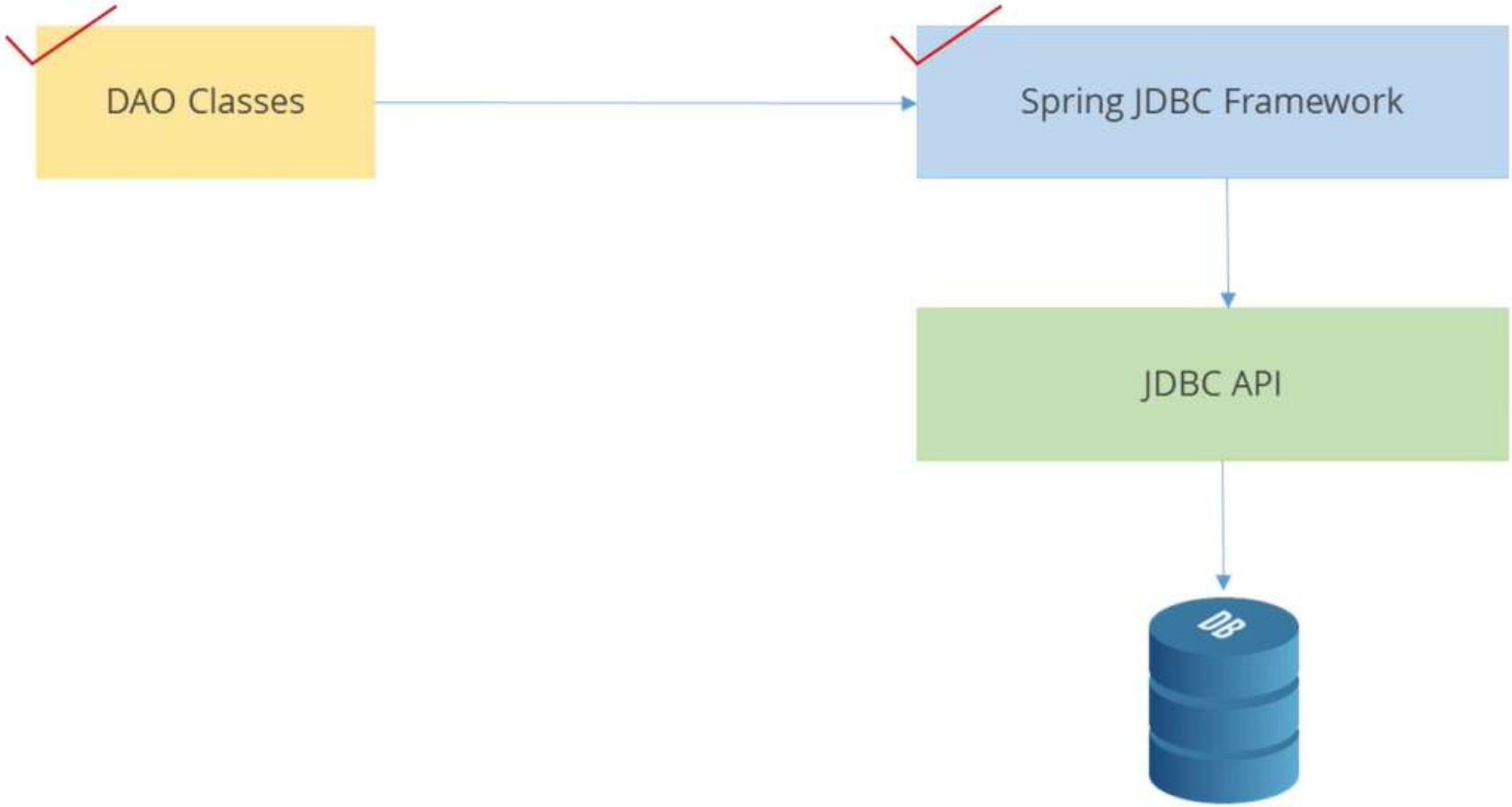
©Simplilearn. All rights reserved.

JDBC vs. Spring JDBC

Difference based on	JDBC	Spring JDBC
Transaction Management Logic: (enable or disable transactions declaratively)	It doesn't support	Spring JDBC supports
Distribution based Transaction Management	It doesn't support	Spring JDBC supports
Annotation based transaction management	It doesn't support	Spring JDBC supports
Resource Allocating Logic	Connection, Statement, and PreparedStatement have to be allocated explicitly by the developer in JDBC.	Spring provides org.springframework.jdbcTemplate object. It abstracts the Database Resource Allocation Logic.
Resource Releasing Logic	In JDBC, developer explicitly writes the logic to release the database resources.	Spring provides org.springframework.jdbc.core.jdbcTemplate object.
Exception Handling	JDBC throws java.sql.SQLException. There won't be specific exceptions for specific problems.	Spring provides Fine Grind Exception Handling mechanism to deal with Database.

©Simplilearn. All rights reserved.

Spring JDBC Architecture



©Simplilearn. All rights reserved.

Advantage of Spring JDBC API

- It provides the capability to establish the connection and interact with database
- It simplifies the development of JDBC, Hibernate, JPA ,and JDO
- It provides multiple templates to interact with DB:

    1. JdbcTemplate
    2. HibernateTemplate
    3. JdoTemplate
    4. JpaTemplate

©Simplilearn. All rights reserved.

Terminologies of Spring JDBC API

CONNECTION POOLING

When a connection is opened and closed (when required) in an application, object utilization becomes difficult and the cost of implementation increases. In such cases, connection pooling is used.

It is a mechanism of pre-creating a group of database connections and keeping them in cache memory for use and reuse. It provides high performance and efficient resource management.

©Simplilearn. All rights reserved.

Terminologies of Spring JDBC API

java.sql.DataSource

It is an interface that is an object-oriented representation of connection pooling. It a connection factory for Java application. It is an alternative for DriverManager and provides Database connections to the Java application.

Its implementation classes are provided by the following connection pooling mechanisms:

1. >Apache BasicDatasource
2. >Spring DriverManagerDatasource
3. >C3p0ComboPoolDatasource

©Simplilearn. All rights reserved.

Spring JDBC Template

Spring JDBC provides a class called "JdbcTemplate" to handle the common logic.

JdbcTemplate is the abstraction of JDBC technology.

Methods provided by JdbcTemplate

1. For insert(), update(), and delete() methods of JDBC, Spring JdbcTemplate provides update() method
2. For findByAccno(), findByBalance(), findAll(), rowCount() methods of JDBC, Spring Template provides query(), queryForInt(), queryObject(), queryForMap(), and queryForList()
3. To handle batch updates, JdbcTemplate has convenient methods

©Simplilearn. All rights reserved.

Spring JDBC Template Methods

`public int update (String query)`	public int update (String query)
`public int update (String query)`	inserts, updates, and deletes records using PreparedStatement using given arguments
`public void execute (String query)`	executes DDL query
`public T execute (String sql, PreparedStatementCallback action)`	executes the query by using PreparedStatement callback
`public T query (String sql, ResultSetExtractorrse)`	fetches records using ResultSetExtractor
`public List query (String sql, RowMapper rse)`	fetches records using RowMapper

©Simplilearn. All rights reserved.

Implementing Spring JDBC in an Application

1. Create database and table in the database
2. Create Java project in eclipse
3. Create Spring Bean class (POJO class)
4. Create a DAO class
5. Configure DataSource and Spring Bean spring configuration XML file
6. Create Spring container either by XmlBeanFactory or ApplicationContext

©Simplilearn. All rights reserved.

Why Integrate Spring with Hibernate?

Spring Framework provides integration with Hibernate, Toplink, Ibatis, and JPA.

Reasons to integrate Spring with Hibernate:

- In large applications, this integration makes the maintenance very easy.
- When integrated, caching and other performance tuning is handled automatically.
- Database can be changed easily without affecting the application code.
- Integrating hibernate application with spring avoids creating the hibernate.cfg.xml file. All information can be contained in the applicationContext.xml file.

©Simplilearn. All rights reserved.

Integrating Spring with Hibernate

1. Configure application.xml.
2. Configure session factory using the class AnnotationSessionFactoryBean.
3. Define HibernateTemplate by passing session factory.
4. For the database transaction, define HibernateTransactionManager. Using the transaction manager, configure AOP.

©Simplilearn. All rights reserved.

HibernateTemplate Class

HibernateTemplate is a helper class that simplifies Hibernate data access code.

It automatically converts HibernateExceptions into DataAccessExceptions by following the org.springframework.dao exception hierarchy.

Structure of HibernateTemplate class:

```
org.springframework.orm.hibernate3
public class HibernateTemplate
extends HibernateAccessor
implements HibernateOperations
```

©Simplilearn. All rights reserved.

Methods of HibernateTemplate Class

Method	Description
`void persist(Object entity)`	Persists the given object.
`Serializable save(Object entity)`	Persists the given object and returns id.
`void saveOrUpdate(Object entity)`	Persists or updates the given object. If id is found, it updates the record; otherwise, it saves the record.
`void update(Object entity)`	Updates the given object
`void delete(Object entity)`	Deletes the given object on the basis of id.
`Object get(Class entityClass, Serializable id)`	Returns the persistent object on the basis of given id.
`Object load(Class entityClass, Serializable id)`	Returns the persistent object on the basis of given id.
`List loadAll(Class entityClass)`	Returns all the persistent objects.

©Simplilearn. All rights reserved.

Creating Application using Spring Hibernate Template

1. Create database.
2. Create Java project in eclipse.
3. Create User.java file. It is the persistent class.
4. Create UserDao.java file. It is the DAO class that uses HibernateTemplate.
5. Create applicationContext.xml file. It contains information of DataSource, SessionFactory, etc.
6. Create Test.Java file. It calls methods of UserDao class.

©Simplilearn. All rights reserved.

Spring JDBC Transaction Management

Spring provides extensive support for transaction management. This helps developers to focus on business logic rather than worrying about the integrity of data in case of any system failures.

Spring provides an abstract layer on top of different transaction management APIs.

It helps application developers to focus on the business problem, without having to know much about the underlying transaction management APIs.

©Simplilearn. All rights reserved.

ACID

The concept of transactions can be described using the following four key properties, known as ACID.

A	**Atomicity** requires that each transaction be 'all or nothing'. If one part of the transaction fails, then the entire transaction fails, and the database state is left unchanged.
C	**Consistency** property ensures that any transaction will bring the database from one valid state to another.
I	**Isolation** property ensures that the concurrent execution of transactions result in a system state that would be obtained if transactions were executed serially, that is, one after the other.
D	**Durability** property ensures that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors.

©Simplilearn. All rights reserved.

Transaction Management Interface

PlatformTransactionManager is a general interface for all Spring transaction managers:

```
public interface PlatformTransactionManager {
   TransactionStatus getTransaction(TransactionDefinition definition);
   throws TransactionException;

   void commit(TransactionStatus status) throws TransactionException;
   void rollback(TransactionStatus status) throws TransactionException;
}
```

Spring has several built-in implementations of this interface that can be used with different transaction management APIs like DataSourceTransactionManager, HibernateTransactionManager, and JpaTransactionManager

©Simplilearn. All rights reserved.

Methods of PlatformTransactionManage

Method	Description
`TransactionStatus getTransaction(TransactionDefinition definition)`	This method returns a currently active transaction or creates a new one, according to the specified propagation behavior.
`void commit(TransactionStatus status)`	This method commits the given transaction, with regard to its status.
`void rollback(TransactionStatus status)`	This method performs a rollback of the given transaction.

©Simplilearn. All rights reserved.

TransactionDefinition Interface

TransactionDefinition is the core interface of the transaction support in Spring, and it is defined as follows:

```
public interface TransactionDefinition
{
   int getPropagationBehavior();
   int getIsolationLevel();
   String getName();
   int getTimeout();
   boolean isReadOnly();
}
```

©Simplilearn. All rights reserved.

TransactionDefinition Interface

TransactionDefinition is the core interface of the transaction support in Spring, and it is defined as follows:

```
public interface TransactionDefinition
{
   int getPropagationBehavior();
   int getIsolationLevel();
   String getName();
   int getTimeout();
   boolean isReadOnly();
}
```

©Simplilearn. All rights reserved.

Methods of TransactionDefinition

Method	Description
`int getPropagationBehavior()`	This method returns the propagation behavior. Spring offers all of the transaction propagation options offered by EJB CMT.
`int getIsolationLevel()`	This method returns the degree to which this transaction is isolated from the work of other transactions.
`String getName()`	This method returns the name of the transaction.
`int getTimeout()`	This method returns the time (in seconds) in which the transaction must complete.
`boolean isReadOnly()`	This method queries whether the transaction is read only.

©Simplilearn. All rights reserved.

TransactionStatus Interface

TransactionStatus interface provides a simple way for transactional code to control transaction execution and query transaction status.

```
public interface TransactionStatus extends SavepointManager {
   boolean isNewTransaction();
   boolean hasSavepoint();
   void setRollbackOnly();
   boolean isRollbackOnly();
   boolean isCompleted();
}
```

©Simplilearn. All rights reserved.

Methods of TransactionStatus

Method	Description
`boolean hasSavepoint()`	This method returns whether this transaction internally carries a savepoint, i.e., has been created as nested transaction based on a savepoint.
`boolean isCompleted()`	This method returns whether this transaction is completed, i.e., whether it has already been committed or rolled back.
`boolean isNewTransaction()`	This method returns true in case the present transaction is new.
`boolean isRollbackOnly()`	This method returns whether the transaction has been marked as rollback-only.
`void setRollbackOnly()`	This method sets the transaction as rollback-only.

©Simplilearn. All rights reserved.

Types of Transaction Management

PROGRAMMATIC TRANSACTION MANAGEMENT

Programmatic transaction management

Declarative transaction management

- It manages transactions with the help of programming.
- It uses PlatformTransactionManager directly to implement the programmatic approach, to implement transactions, and to create instance of TransactionDefinition with the appropriate transaction attributes.
- Once the TransactionDefinition is created, you can start your transaction by calling getTransaction() method, which returns an instance of TransactionStatus.
- The TransactionStatus objects helps in tracking the current status of the transaction.
- Programmatic Transaction Management uses commit() method of PlatformTransactionManager to commit the transaction.

©Simplilearn. All rights reserved.

Types of Transaction Management

DECLARATIVE TRANSACTION MANAGEMENT

Programmatic transaction management	
Declarative transaction management	

- Separates transaction management from the business code.
- Allows you to manage the transaction with the help of configuration instead of hard coding in your source code.
- Uses annotations or XML-based configurations to manage the transactions.
- Uses <tx:advice /> tag. This creates a transaction-handling advice and defines a Pointcut that matches all methods needed for the transaction at the same time.

©Simplilearn. All rights reserved.

Press Esc to exit full screen

Advanced Java

Spring MVC Architecture, Components, and Framework

00:03 / 09:57

Why Spring MVC ?

Problems in Enterprise Application development with Servlet and JSP Technology

1. Servlet and code are not reusable
2. Web designing tools can't be used in case of servlet
3. Parallel development is not possible

Solution

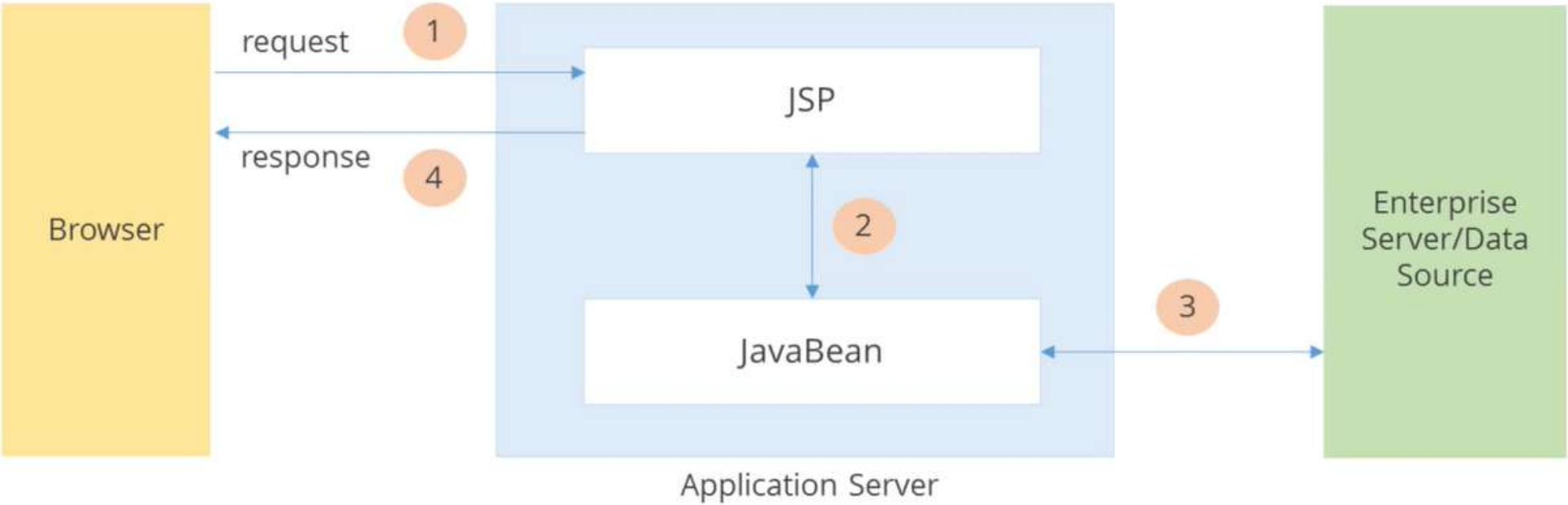
To solve the above problems, Sun microsystem introduced two design patterns for developing enterprise applications:

1. Model 1 (Page Centric Model)
2. Model 2 (MVC)

©Simplilearn. All rights reserved.

Model 1: Page Centric Model

ARCHITECTURE



©Simplilearn. All rights reserved.

Model 1: Page Centric Model

WORKFLOW

1. Request is sent by the browser and received by JSP.
2. Java bean object is created and business methods are called. If required, it communicates with the database to get the required data
3. JSP displays the processed data to the end user

©Simplilearn. All rights reserved.

Model 1: Page Centric Model

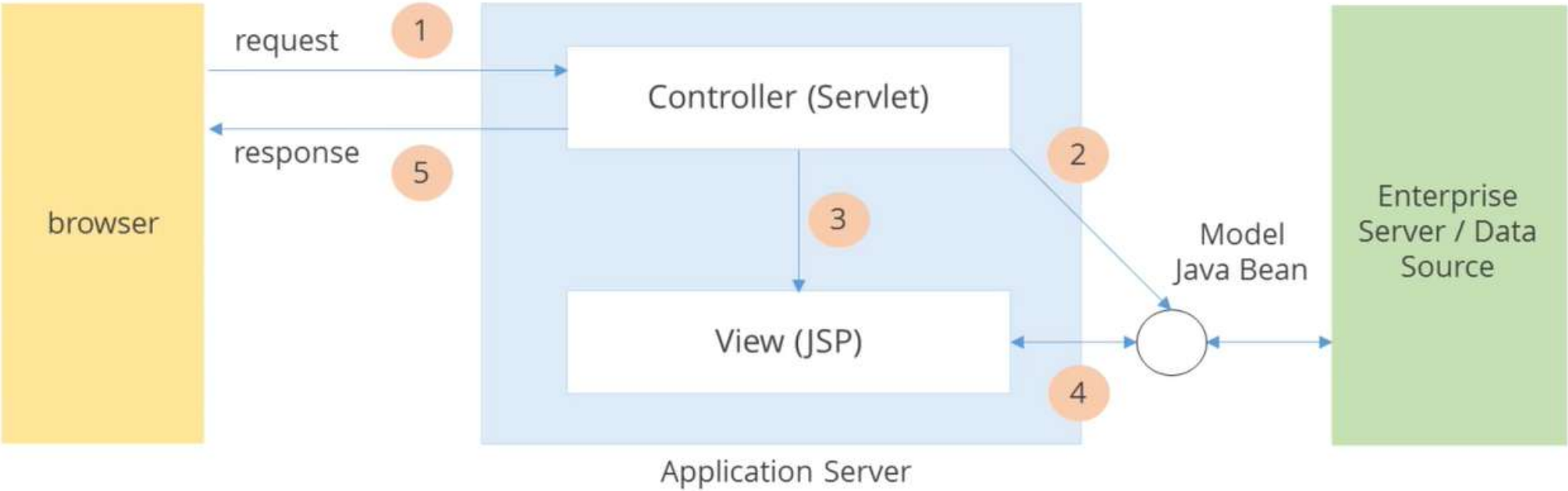
LIMITATIONS

- There is no clear separation of responsibilities
- JSP acts as both controller and view
- Business logic is separated using Java Bean

©Simplilearn. All rights reserved.

Model 2: MVC Model

ARCHITECTURE



©Simplilearn. All rights reserved.

Model 2: MVC Model

WORKFLOW

1. Controller or Servlet receives request from browser and captures the input
2. Controller invokes the business method of the model or Java bean
3. Model connects with the database and gets business data
4. Model sends response to controller (Keeps the process data in heap memory request, session, and ServletContext)
5. Controller switches the control to appropriate view of the application

©Simplilearn. All rights reserved.

Model 2: MVC Model

LIMITATIONS

- Clear separation of responsibilities
- Code reusability
- Single point entry for the application
- Support for multiple view technologies

©Simplilearn. All rights reserved.

Advantages of Spring MVC

- Spring MVC is used to develop the web application that uses MVC design pattern
- Spring MVC is meant to make web application development faster, cost-effective, and flexible

©Simplilearn. All rights reserved.

Components of Spring MVC

DispatcherServlet

- DispatcherServlet
- HandlerMapping
- Controller
- ModelAndView
- ViewResolver

1. It is given by org.springframework.web.DispatcherServlet
2. It follows FrontController Design Pattern
3. Whatever URL comes from the Client, Servlet intercepts the Client Request before passing the Request object to the Controller
4. In web configuration file, write <servlet-mapping> in such a way that Dispatcher Servlet is invoked for ClientRequest

```
<servet>
<servlet-name> front-controller</servlet-name>
<servlet-class>
org.springframework.web.servlet.DispatcherServlet
</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name> front-controller </servlet-name>
<url-pattern>*.extensionname</url-pattern>
</servlet-mapping>
```

Definition given in the web.xml to invoke Spring's Dispatcher

©Simplilearn. All rights reserved.

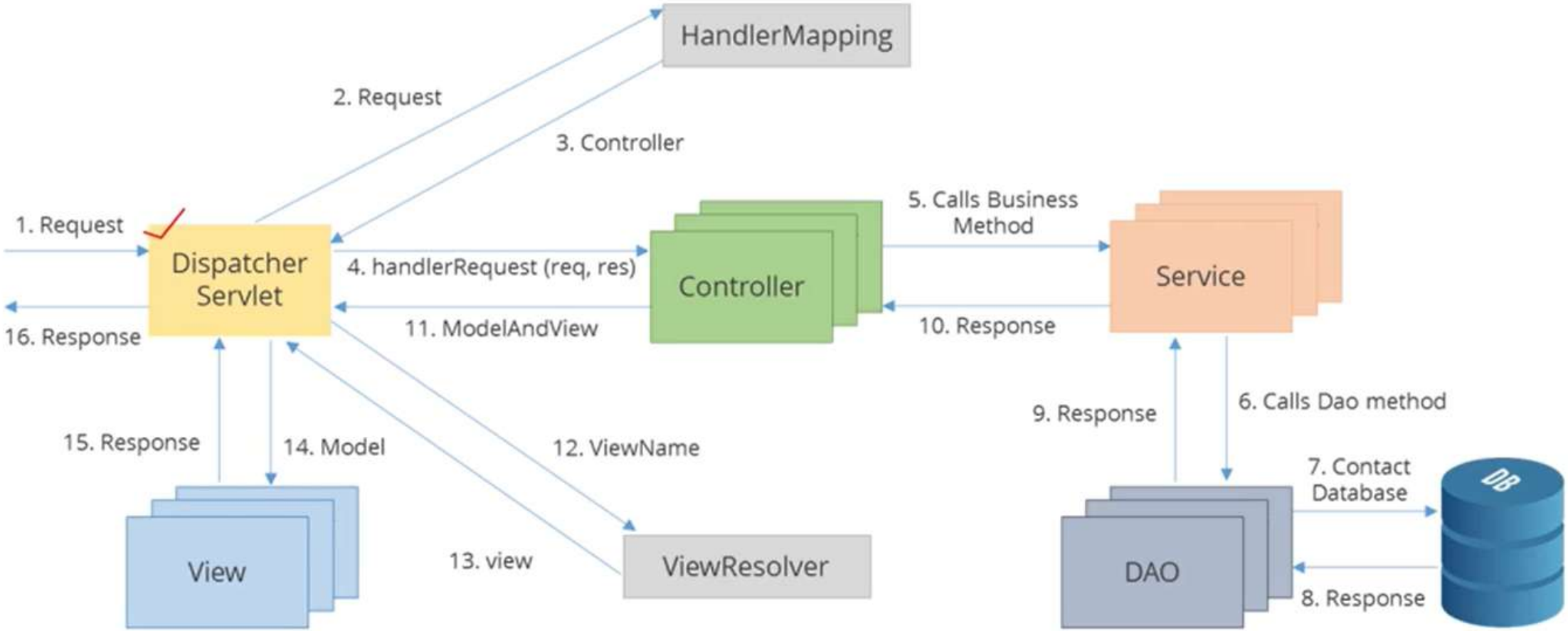
What is Spring MVC Framework?

The Spring web MVC framework provides model-view-controller architecture that can be used to develop flexible and loosely coupled web applications.

Workflow is described in 16 steps in the next slide.

©Simplilearn. All rights reserved.

Spring MVC Framework



©Simplilearn. All rights reserved.

simplilearn

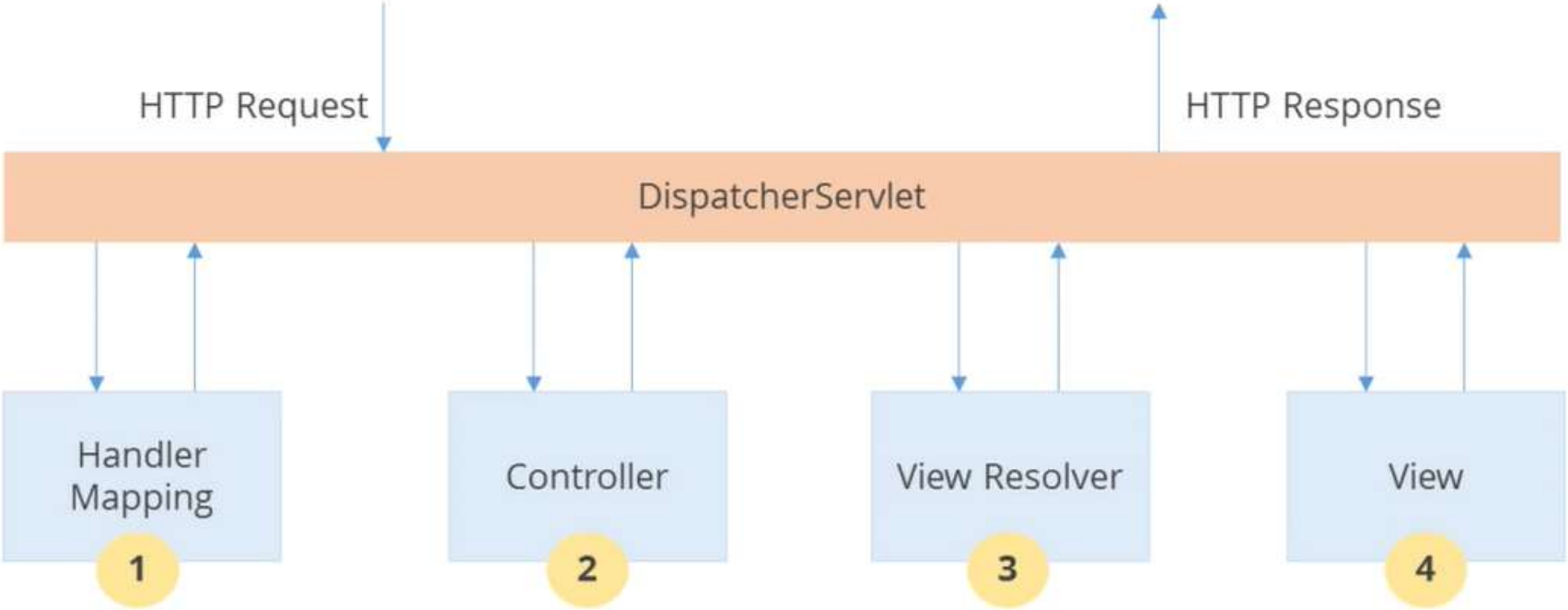
Spring MVC Framework

1. Client requests a resource in the Web Application
2. The Spring Front Controller (DispatcherServlet) requests the HandlerMapping to identify the particular controller for the given URL
3. HandlerMapping identifies the controller and sends it to the DispatcherServlet
4. DispatcherServlet calls handleRequest (req, res) method and passes these two objects to that controller
5. Controller calls the business method
6. Service class calls the DAO method for business data
7. DAO interacts with the database to get the data
8. Database shares the result
9. DAO returns this result data to service
10. DAO data is processed according to business requirement and returns to Controller
11. Controller returns ModelAndView object back to front Controller
12. Dispatcher resolves actual View by consulting the View Resolver object
13. ViewResolver renders View to the Dispatcher
14. DispatcherServlet consults the View to the DispatcherServlet
15. View executes and returns HTML to the DispatcherServlet
16. DispatcherServlet sends output to the Browser

© simplilearn. All rights reserved.
06:43 / 09:57

Spring Web MVC DispatcherServlet Workflow

The request processing workflow of the Spring Web MVC DispatcherServlet is illustrated in the following diagram:



©Simplilearn. All rights reserved.

Spring Web MVC DispatcherServlet Workflow

DESCRIPTION

- After receiving an HTTP request, DispatcherServlet consults the HandlerMapping to call the appropriate Controller.
- The Controller takes the request and calls the appropriate service method based on used GET or POST method. The service method will set model data based on defined business logic and return view name to the DispatcherServlet.
- The DispatcherServlet will take help from ViewResolver to pick up the defined view for the request.
- Once view is finalized, the DispatcherServlet passes the model data to the view; this is finally rendered on the browser

©Simplilearn. All rights reserved.

Configurations Required for Spring MVC

- Map requests that you want the DispatcherServlet to handle using URL mapping in the web.xml file
- Create [servlet-name]-servlet.xml file
- Define a controller
- Create view

© simplilearn. All rights reserved.
08:24 / 09:57

Advanced Java

Basics of SOA Architecture and Web Services

Phases of Software Evolution

The software evolution has distinct phases. These layers are built up one by one over many years.

Software evolution begins with understanding the concept of 1 and 0 (bits) that gives rise to machine language. This is followed by assembly, procedure oriented, object oriented, component oriented, and service oriented languages.

©Simplilearn. All rights reserved.

https://www.researchgate.net/publication/215571061

simplilearn

What is SOA?

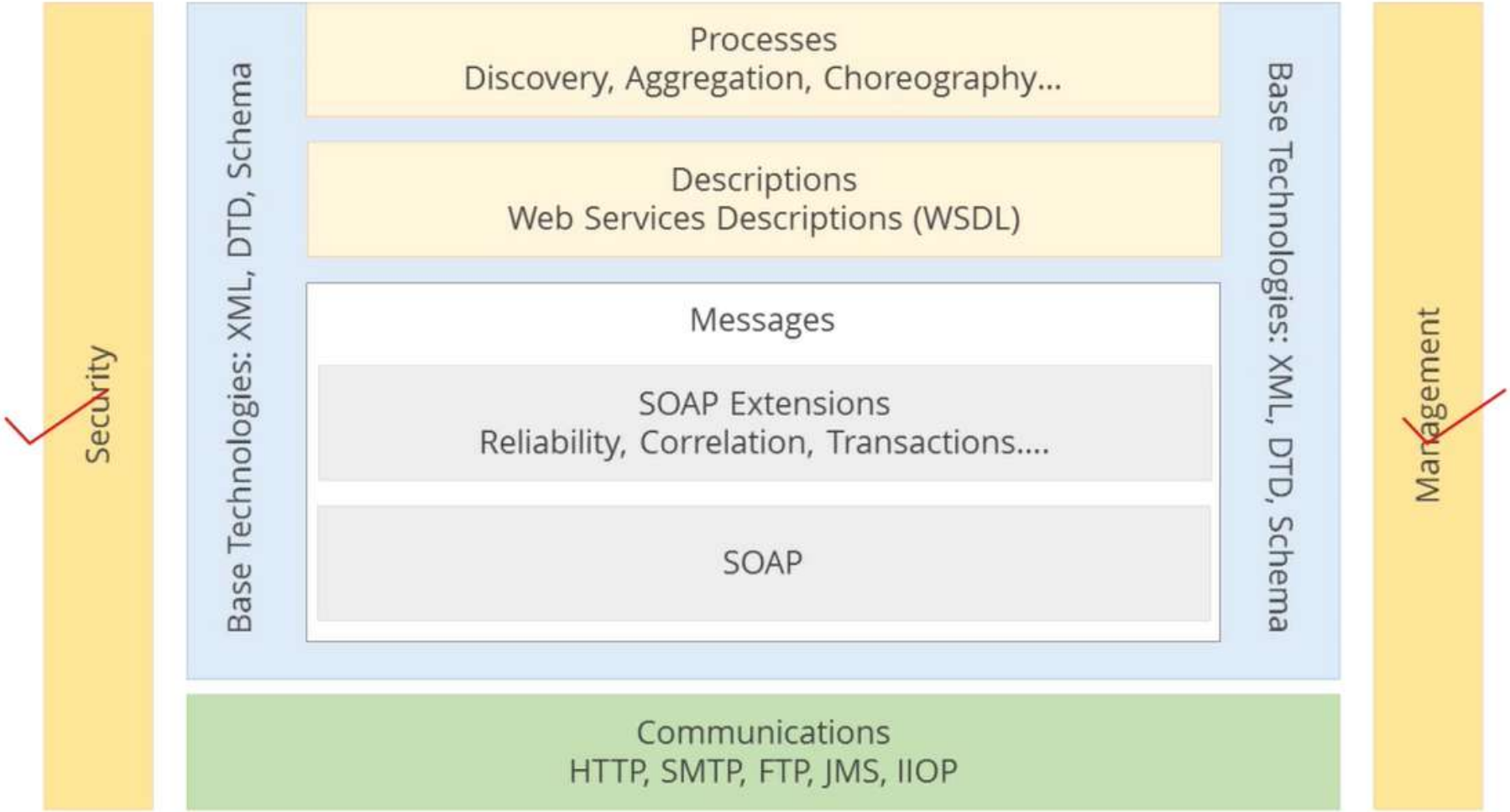
AN ARCHITECTURAL SOLUTION

SOA is the architectural solution for integrating diverse systems by providing an architectural style that promotes loose coupling reuse.

It is a programming model or paradigm where web service and contracts becomes a dominant design for interoperability

©Simplilearn. All rights reserved.

SOA Architecture



©Simplilearn. All rights reserved.

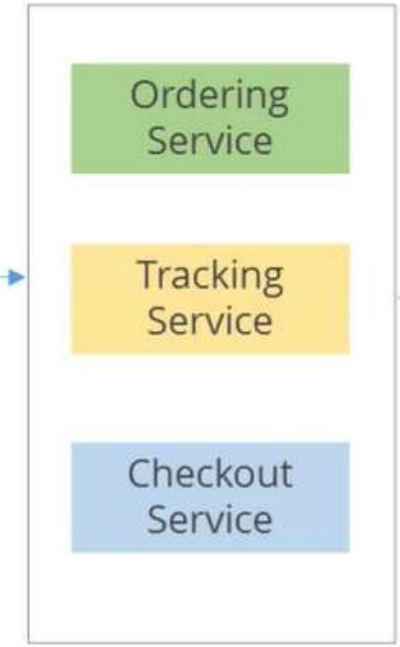
What Is SOA?

A COLLECTION OF SERVICES

SOA is a collection of Services.

The following diagram shows how an Online Shopping website selling gadgets is involved with different services like Ordering, Tracking, and Checkout to give service to its customers.

Online shopping website selling gadget



Customers

©Simplilearn. All rights reserved.

What Is SOA?

ESTABLISHING COMMUNICATION

It helps in establishing communication between all services (applications).

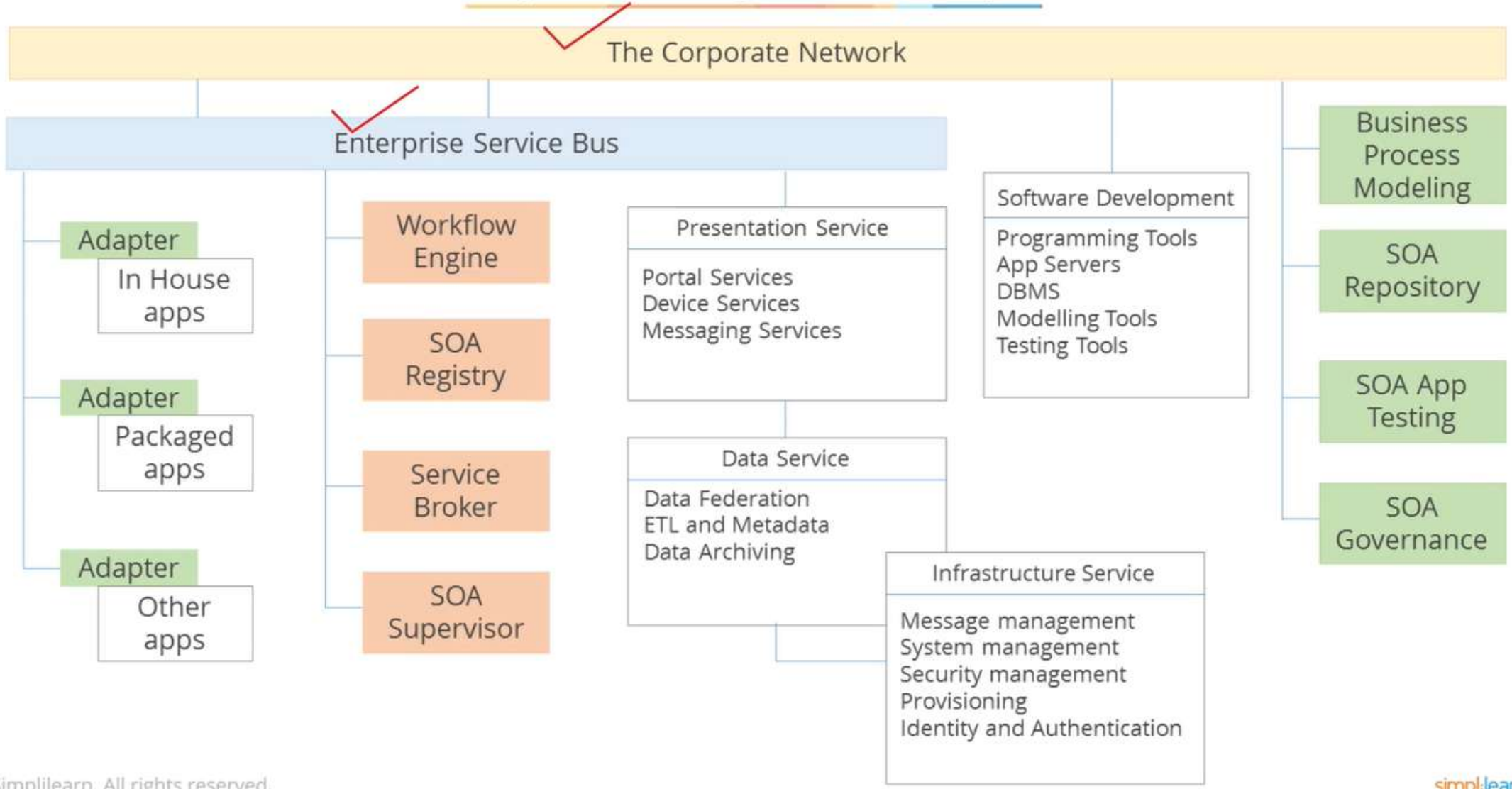
©Simplilearn. All rights reserved.

Advantages of SOA

SOA Services	Uses
Service	• Improved information flow • Ability to expose internal functionality • Organizational flexibility
Service Re-use	Lower software development and management costs
Messaging	Configuration flexibility
Message Monitoring	• Business intelligence • Performance measurement • Security attack detection
Message Control	• Application of management policy • Application of security policy
Message Transformation	Data translation
Message Security	Data confidentiality and integrity
Complex Event Processing	• Simplification of software structure • Ability to adapt quickly to different external environments • Improved manageability and security

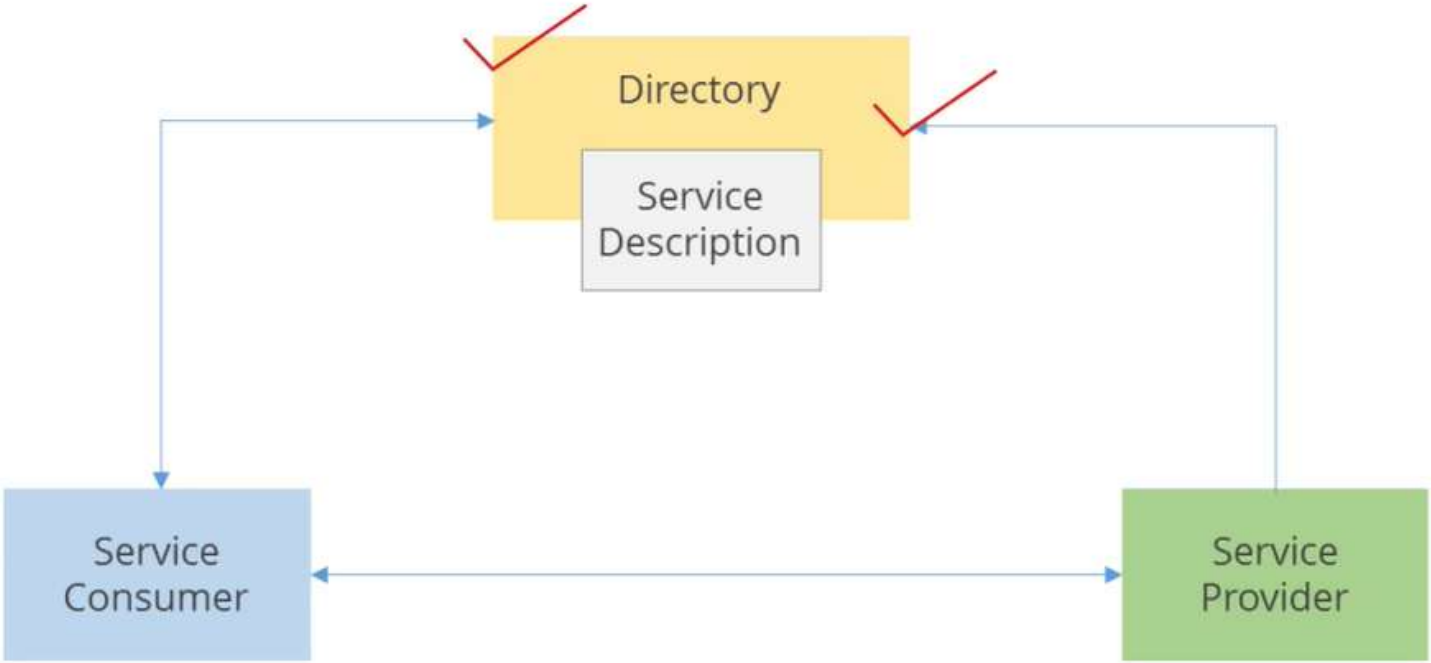
©Simplilearn. All rights reserved.

Implementing SOA: Example



©Simplilearn. All rights reserved.

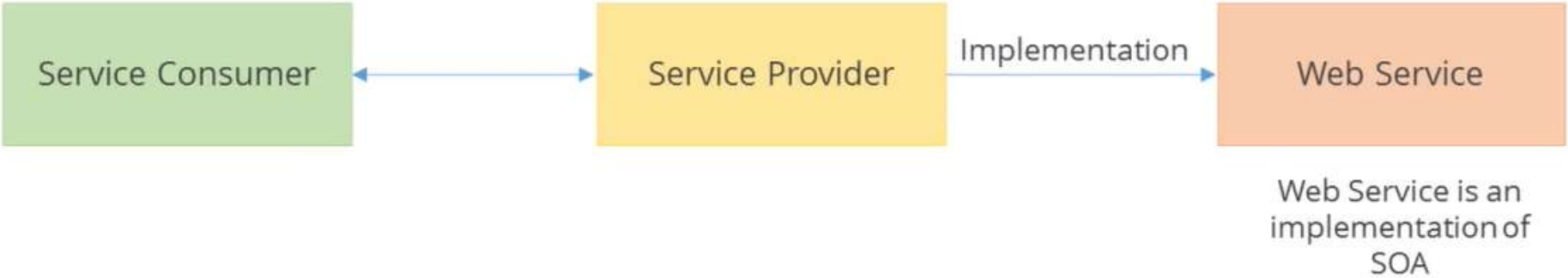
SOA Service Description



Consumer queries the directory to locate a service and communicate with the provider

©Simplilearn. All rights reserved.

Web Service as Implementation of SOA



©Simplilearn. All rights reserved.

Why Use Web Services?

1. Web applications enable interaction between an end user and a website. Web services are service-oriented and enable application-to-application communication over the Internet and easy accessibility to heterogeneous applications and devices.

2. Web services can be invoked through XML-based RPC mechanisms across firewalls.

3. Web services provide a cross-platform, cross-language solution based on XML messaging.

4. Web services facilitate ease of application integration using a lightweight infrastructure without affecting scalability.

5. Web services enable interoperability among heterogeneous applications.

©Simplilearn. All rights reserved.

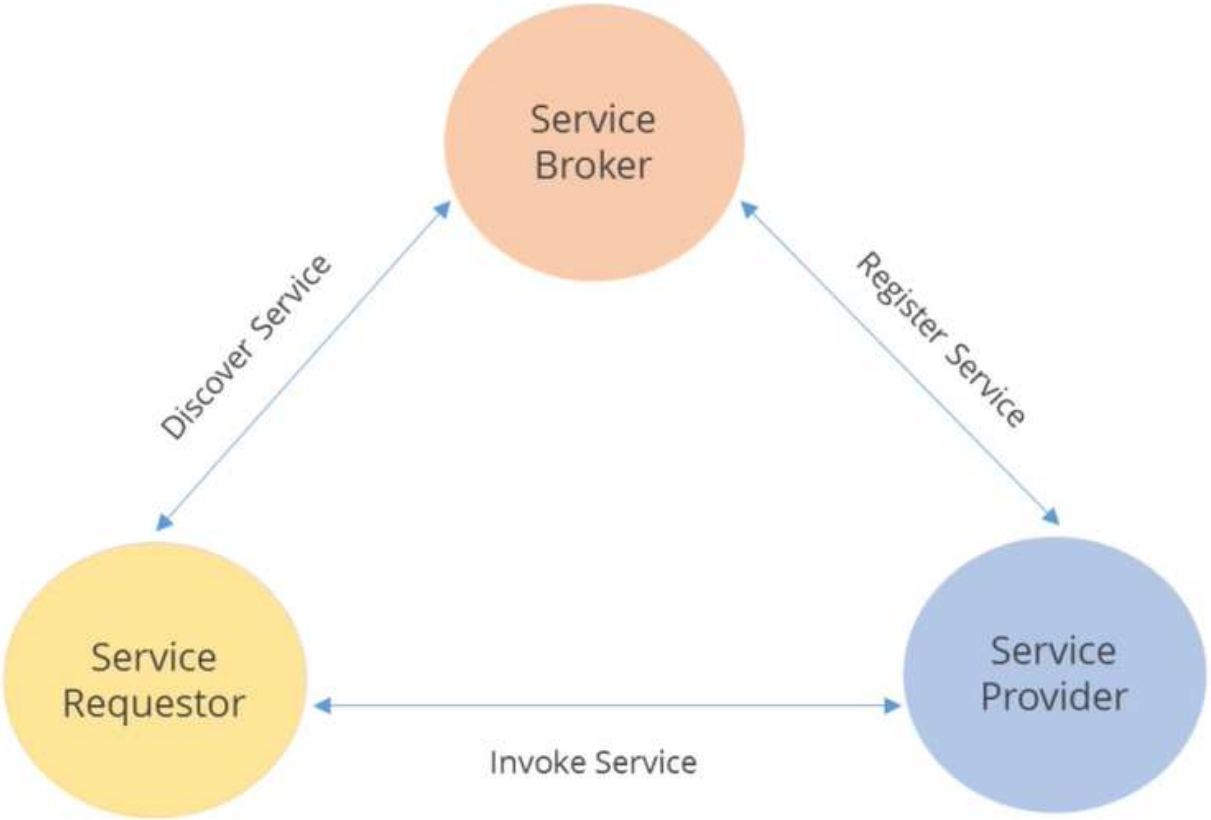
What Are Web Services?

- Web services are based on the concept of service-oriented architecture (SOA).
- According to Gartner research (June 15, 2001), "Web services are loosely coupled software components delivered over Internet standard technologies."
- Web services are self-describing and modular business applications that expose the business logic as services over the Internet. This is done through programmable interfaces and Internet protocols to provide ways to find, subscribe, and invoke those services.

©Simplilearn. All rights reserved.

Web Services Operational Model

Web Service operational model can be conceptualized as a simple operational model that has a lot in common with a standard communication model:



Operations are conceived as involving three distinct roles and relationships that define the Web Services provider and users.

©Simplilearn. All rights reserved.

Web Services: Example

Consider the following scenario:

1. A Travel service provider deploys its Web services by exposing the business applications obtained from different travel businesses like airlines, car-rental, hotel accommodation, credit card payment, and so forth.

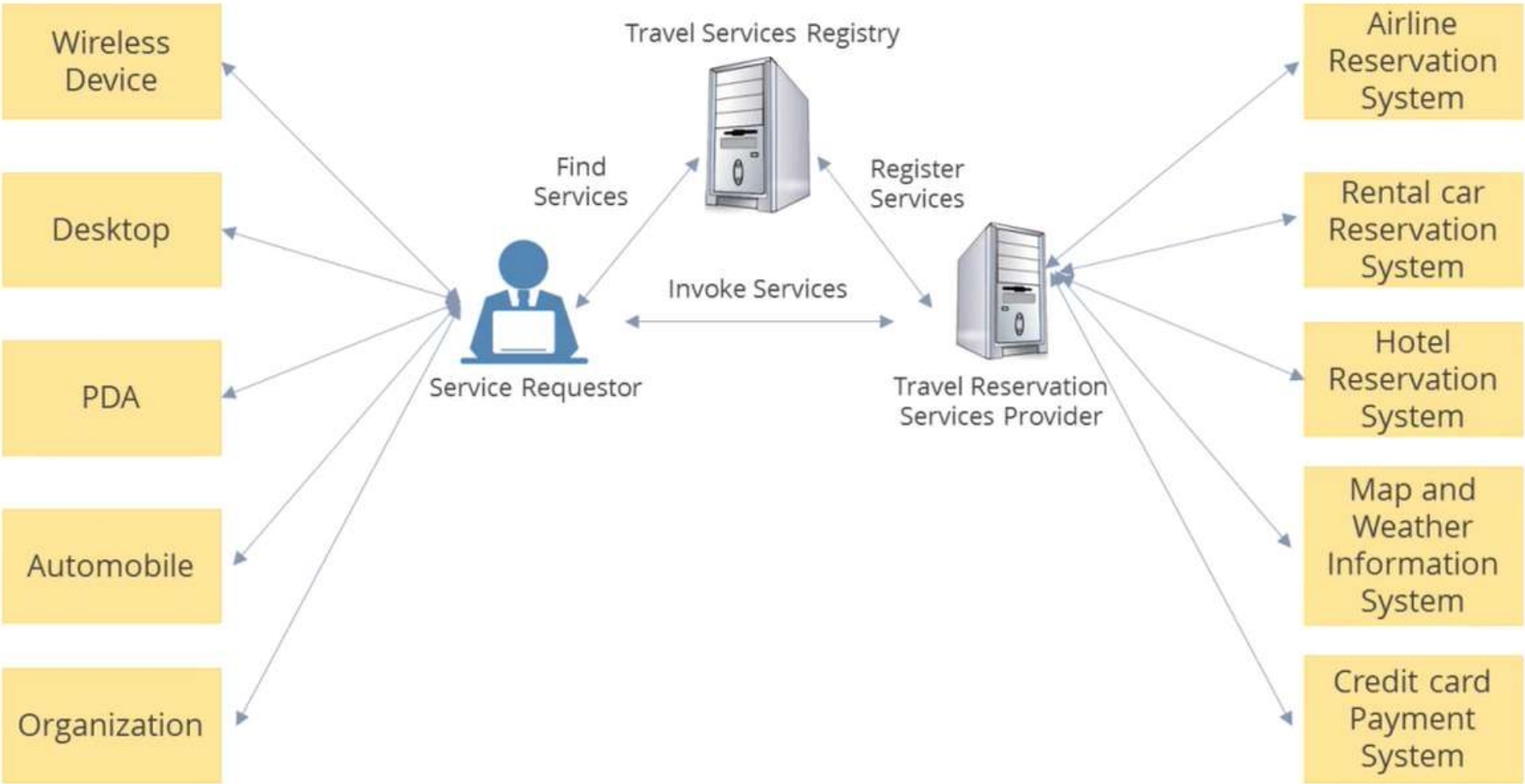
2. The service provider registers its business services with descriptions using a public or private registry. The registry stores the information about the services exposed by the service provider.

3. The customer discovers the Web services using a search engine or by locating it directly from the registry and then invokes the Web services to perform travel reservations and other functions over the Internet using any platform or device.

4. In the case of large-scale organizations, the business applications consume these Web services to provide travel services to their own employees through the corporate intranet.

©Simplilearn. All rights reserved.

Web Services: Example



©Simplilearn. All rights reserved.

Types of Web Services

Web services are based on the following key standards:

1. Extensible Markup Language (XML) based messaging
2. Universal Description, Discovery, and Integration (UDDI)
3. Web Services Description Language (WSDL)
4. Simple Object Access Protocol (SOAP)

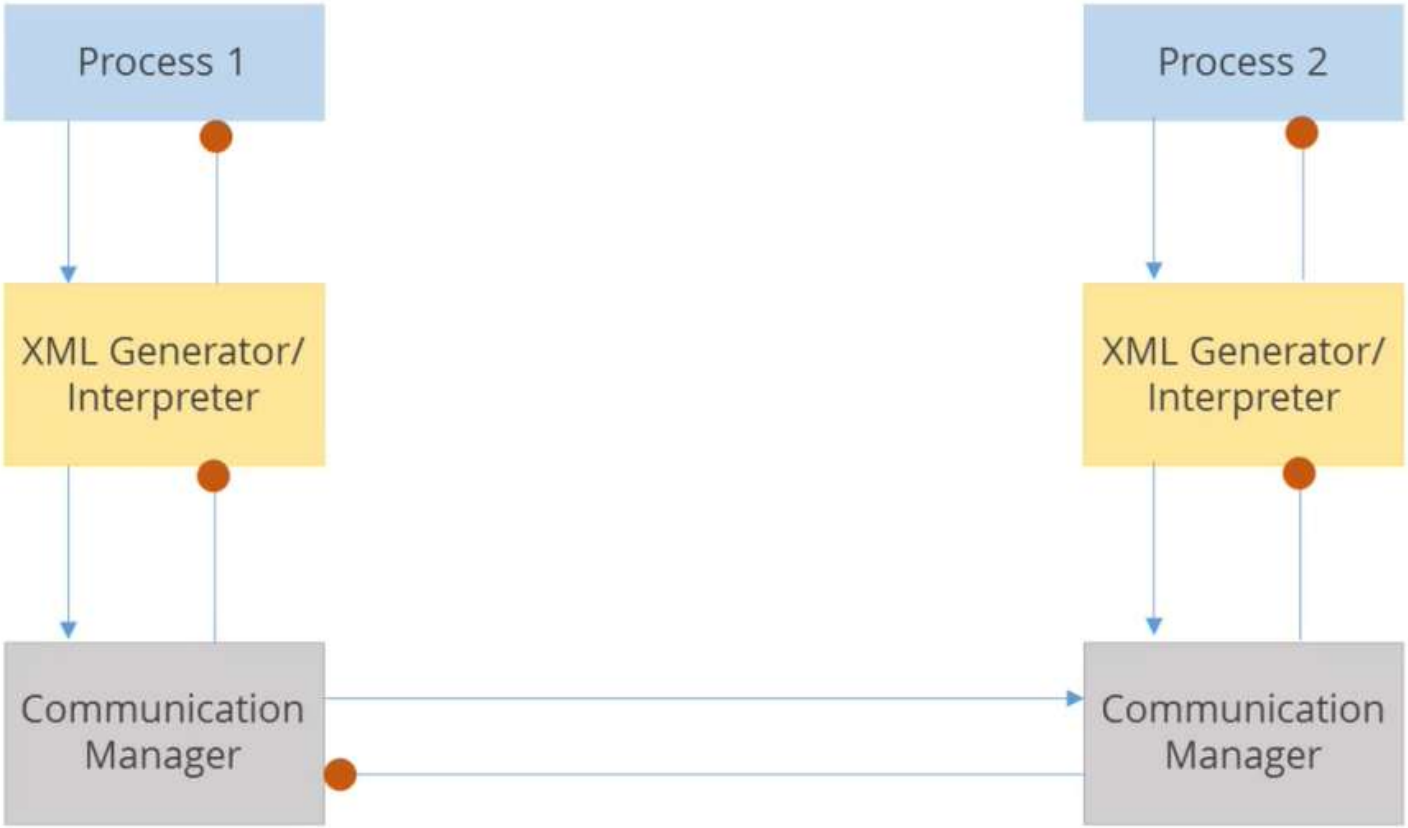
©Simplilearn. All rights reserved.

Types of Web Services

EXTENSIBLE MARKUP LANGUAGE (XML)

These technologies are interoperable at a core level as they use XML as data representation layer for all web service protocols and technologies that are created

Extensible Markup Language (XML)
Universal Description, Discovery and Integration (UDDI)
Web Services Description Language (WSDL)
Simple Object Access Protocol (SOAP)

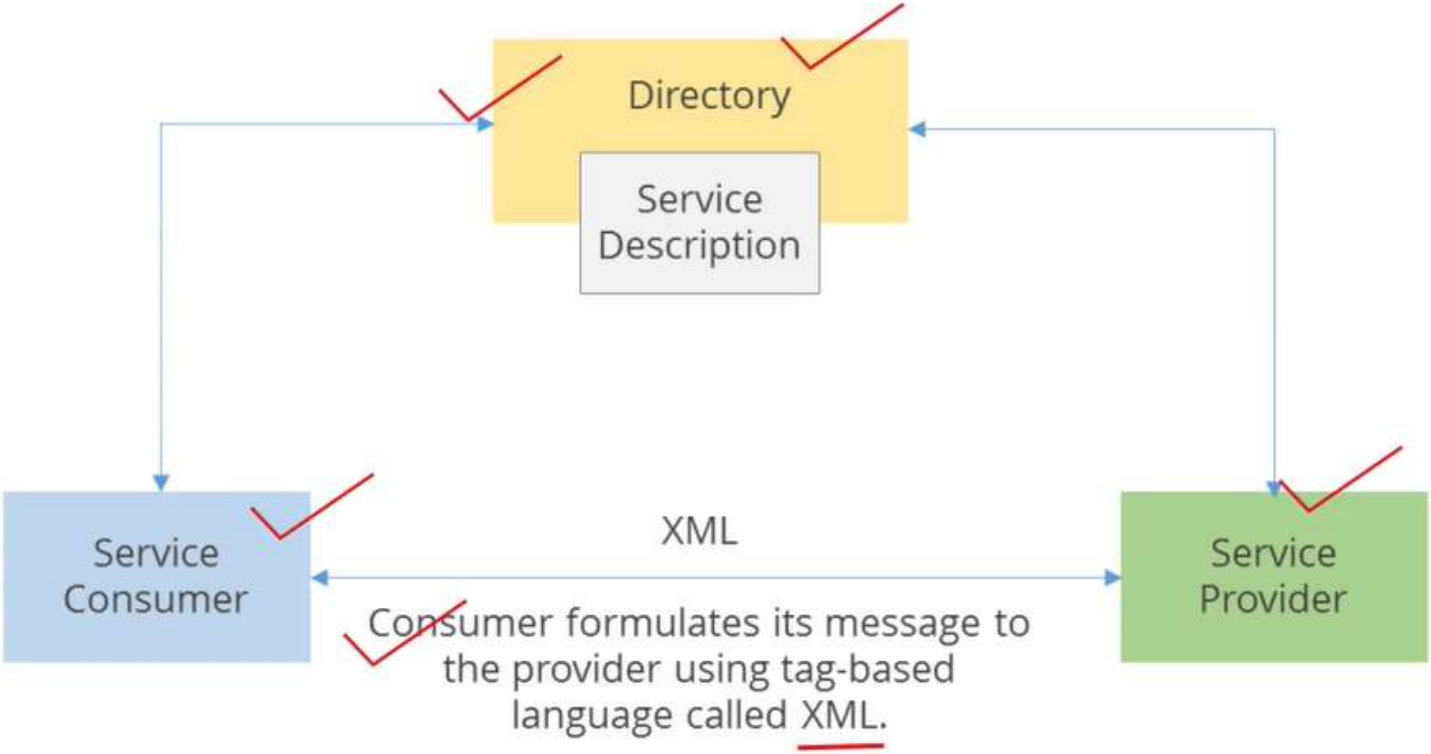


©Simplilearn. All rights reserved.

Types of Web Services

Extensible Markup Language (XML)
Universal Description, Discovery and Integration (UDDI)
Web Services Description Language (WSDL)
Simple Object Access Protocol (SOAP)

WEB SERVICES WITH XML



©Simplilearn. All rights reserved.

Types of Web Services

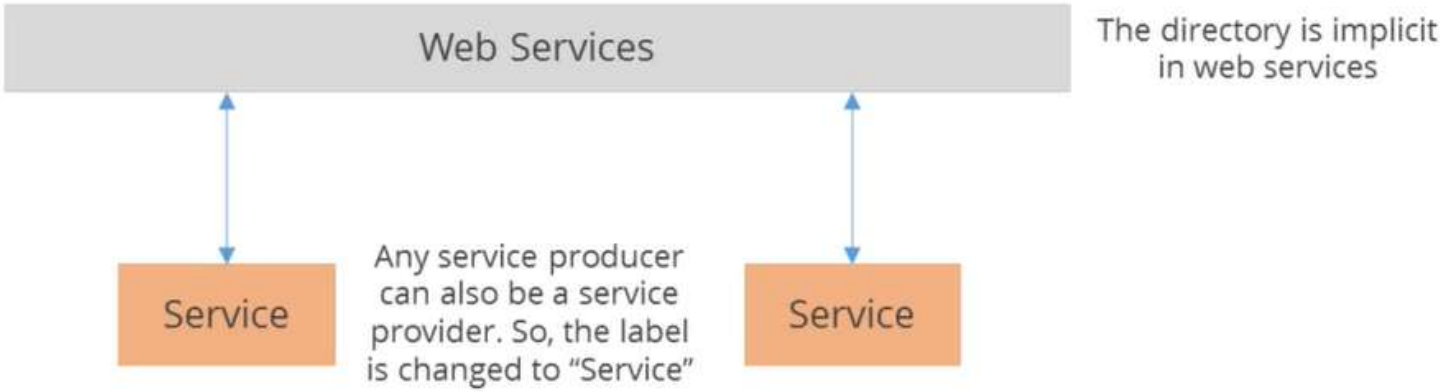
UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION (UDDI)

Extensible Markup Language (XML)
Universal Description, Discovery and Integration (UDDI)
Web Services Description Language (WSDL)
Simple Object Access Protocol (SOAP)

UDDI provides a world-wide registry of web services for advertisements, discovery, and integration purposes.

UDDI could be dynamic or static; Business Analysts and technologists could use UDDI to search for available web services.

The UDDI Business Registry provides a place for a company to programmatically describe its services and business processes and its preferred methods for conducting business.



©Simplilearn. All rights reserved.

Types of Web Services

WEB SERVICES DESCRIPTION LANGUAGE (WSDL)

Extensible Markup Language (XML)
Universal Description, Discovery and Integration (UDDI)
Web Services Description Language (WSDL)
Simple Object Access Protocol (SOAP)

WSDL describes the interface of the Web Service. It is comparable to the concept of "Remote Interface" of Java RMI or the "Interface Definition Language File (IDL)" of RPC.

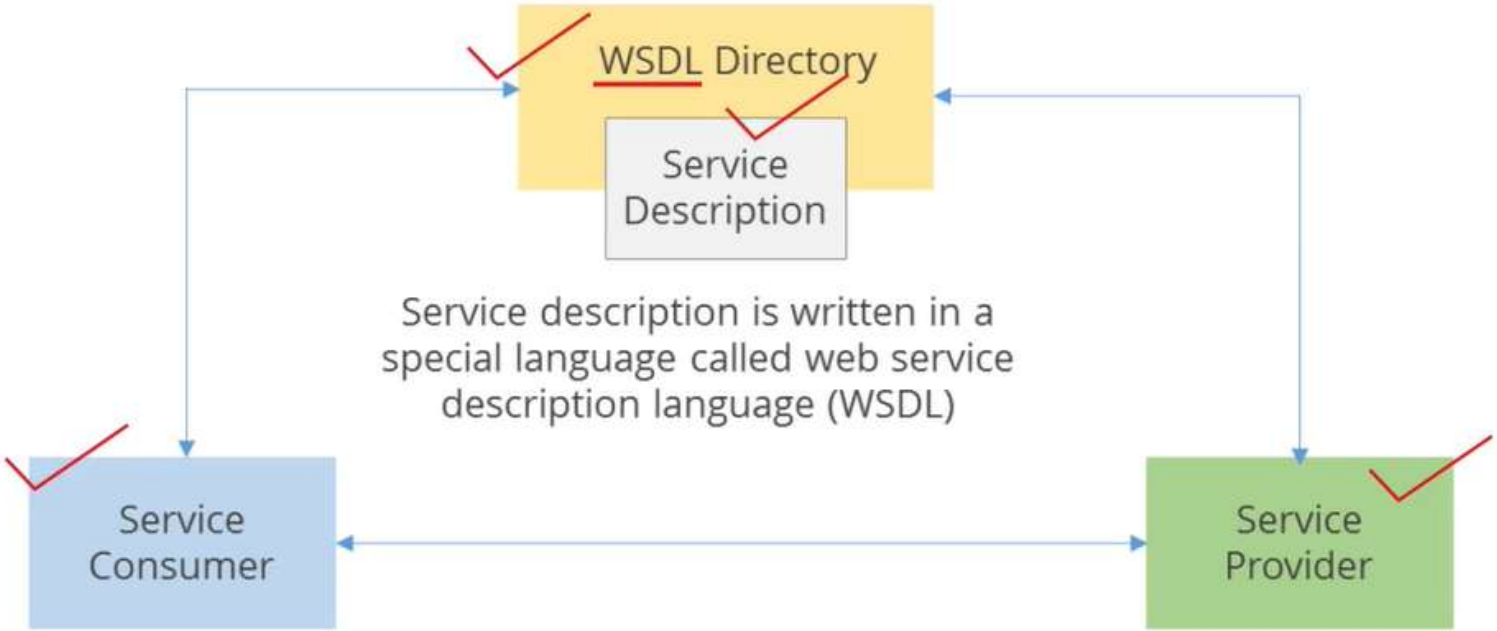
It standardizes how a web service represents the input and output parameters of an invocation externally, that is, the function structure, the nature of invocation (in, in/out,etc), and the service protocol binding.

©Simplilearn. All rights reserved.

Types of Web Services

WEB SERVICES WITH WSDL

Extensible Markup Language (XML)
Universal Description, Discovery and Integration (UDDI)
Web Services Description Language (WSDL)
Simple Object Access Protocol (SOAP)



©Simplilearn. All rights reserved.

Types of Web Services

WHY USE SIMPLE OBJECT ACCESS PROTOCOL (SOAP)?

Extensible Markup Language (XML)
Universal Description, Discovery and Integration (UDDI)
Web Services Description Language (WSDL)
Simple Object Access Protocol (SOAP)

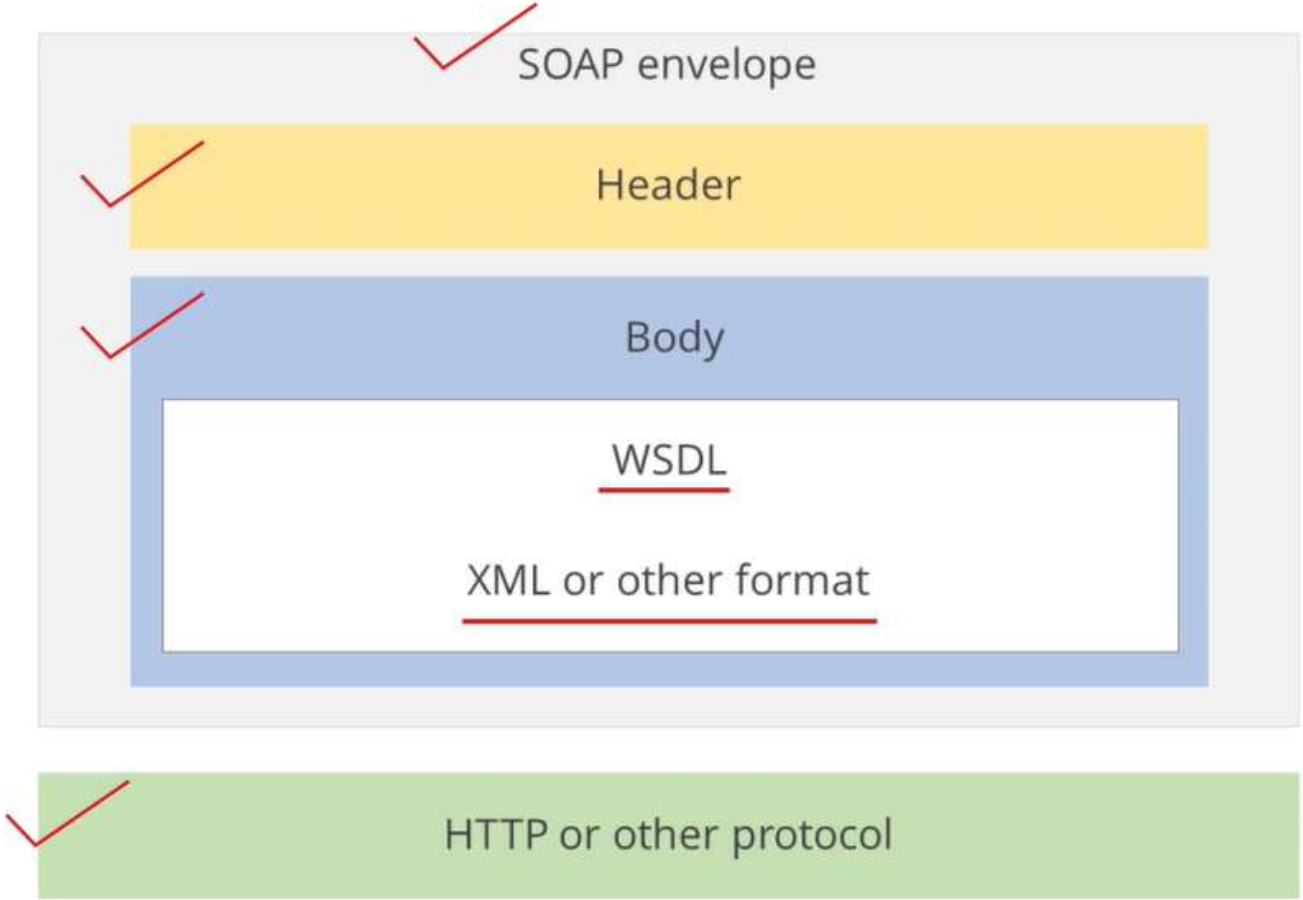
- It is important to have Internet communication between programs for application development.
- Firewalls and proxy servers generally block traffic that creates a compatibility and security problem, even if they are used for application communication like RPC (Remote Procedure Call).
- HTTP is supported by all Internet browsers and servers. However, SOAP was created to accomplish better communication between applications.
- SOAP supports different technologies and programming languages.

©Simplilearn. All rights reserved.

Types of Web Services

Extensible Markup Language (XML)
Universal Description, Discovery and Integration (UDDI)
Web Services Description Language (WSDL)
Simple Object Access Protocol (SOAP)

SOAP ARCHITECTURE



©Simplilearn. All rights reserved.

Types of Web Services

SOAP ELEMENTS

Extensible Markup Language (XML)
Universal Description, Discovery and Integration (UDDI)
Web Services Description Language (WSDL)
Simple Object Access Protocol (SOAP)

- Envelope: It identifies the XML document as a SOAP message.
- Header: It is an optional element that contains header information.
- Body: It contains call and response information.
- Fault: It provides information about errors that occurred while processing the message.

©Simplilearn. All rights reserved.

Java APIs for Web Service

There are two main APIs defined by Java for developing web service applications (since Java EE 6):

1. JAX-WS: for SOAP web services.
2. JAX-RS: for RESTful web services.

©Simplilearn. All rights reserved.

Axis

Axis facilitates easy deployment and undeployment of services using XML-based Web services deployment descriptors (WSDDs).

- It enables deploying and undeploying services and also Axis-specific resources like handlers and chains using an administration utility 'AdminClient' provided as part of the Axis toolkit.
- It helps deploy a service, ensures that the AXIS CLASSPATH is set, and runs the following command:

```
java org.apache.axis.client.AdminClient deploy.wsdd
```

- It helps undeploy a service, ensures that the AXIS CLASSPATH is set, and then runs the following command:

```
java org.apache.axis.client.AdminClient undeploy.wsdd
```

©Simplilearn. All rights reserved.

Web Service in an Axis Environment

To create a Web service in an Axis environment, the following Models are used:

1. Create the service provider (server application)
2. Create the service requester (client applications)

©Simplilearn. All rights reserved.

Creating a SOAP based Web Service

1. Create a new dynamic web project and name it "SimpleSOAPExample"
2. Create a new package named "com.webservice"
3. Create a simple Java class named "HelloWorld.java" which works as Service
4. Right click project → new → web service
5. Click next

   In service implementation text box, write the qualified class name of created class (HelloWorld.java), move both above slider to maximum level (Test service and Test Client level), and click finish. You are done. A new project named "SimpleSOAPExampleClient" will be created in your work space.

6. Click start server
7. After clicking start server, eclipse will open test web service API. With this test API, you can test your web service

©Simplilearn. All rights reserved.

Introduction to RESTful Web Services

RESTful web services are lightweight, highly scalable, and maintainable Web Services. They are basically based on REST (Representational State Transfer) Architecture.

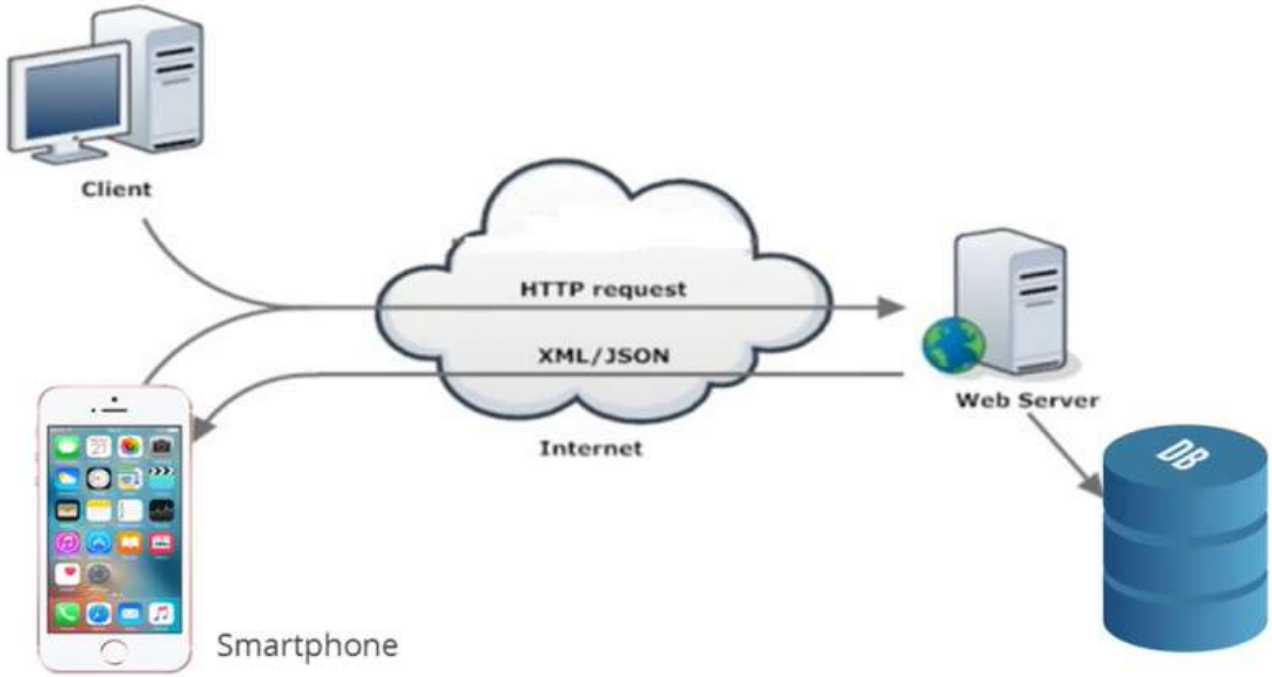
REST architecture is basically focuses on two things:

1. Resources: Its application functionality, which is represented by a unique URL.
2. Interface: Its HTTP method works as an interface to access the resources.

©Simplilearn. All rights reserved.

RESTful Web Services: Working

- A RESTful web service usually defines a URI (Uniform Resource Identifier), which is a service that provides resource representation such as JSON and a set of HTTP Methods.
- There is no need to use XML data interchange format for request and response.
- The REST web services can be return XML, JSON, or even HTML format response.



©Simplilearn. All rights reserved.

Creating a RESTful Web Service

FUNCTIONALITIES

Create a web service called UserLog Management with the following functionalities:

HTTP Method	URI	Operation	Operation Type
GET	/UserLogService/users	Get list of users	Read Only
GET	/UserLogService/users/1	Get User with Id 1	Read Only
PUT	/UserLogService/users/2	Insert User with Id 2	Idempotent
POST	/UserLogService/users/2	Update User with Id 2	N/A
DELETE	/UserLogService/users/1	Delete User with Id 1	Idempotent
OPTIONS	/UserLogService/users	List the supported operations in web service	Read Only

©Simplilearn. All rights reserved.

Creating a RESTful Web Service

ENVIRONMENT SETUP

1. Set up Java Development Kit (JDK)
2. Set up Eclipse IDE
3. Set up Jersey Framework Libraries
4. Download the latest version of Jersey framework binaries from the following link: https://jersey.java.net/download.html
5. Set up Apache Tomcat

©Simplilearn. All rights reserved.

Creating a RESTful Web Service

WRITING RESTful WEB SERVICE WITH JERSEY FRAMEWORK

1. Create a Dynamic Web Project named UserLogManagement using Eclipse IDE.
2. Add Jersey Framework and its dependencies (libraries) in your project.
3. Create UserLogService.java and User.java,UserLogDao.java files under your dynamic project
4. Create a Web XML Configuration file to specify Jersey framework servlet for your application.
5. Export your application as a war file and deploy the same in tomcat.

©Simplilearn. All rights reserved.

SOAP vs. RESTful Web Services

SOAP	RESTful
The main advantage of SOAP is that it provides a mechanism for services to describe themselves to clients and to advertise their existence.	In RESTful Web Services, clients have to know what to send and what to expect.
SOAP brings its own protocol and focuses on exposing pieces of application logic (not data) as services.	REST is focused on accessing named resources through a single consistent interface.
SOAP-based reads cannot be cached.	REST has better performance and scalability. REST reads can be cached,
SOAP only permits XML.	REST permits many different data formats

©Simplilearn. All rights reserved.