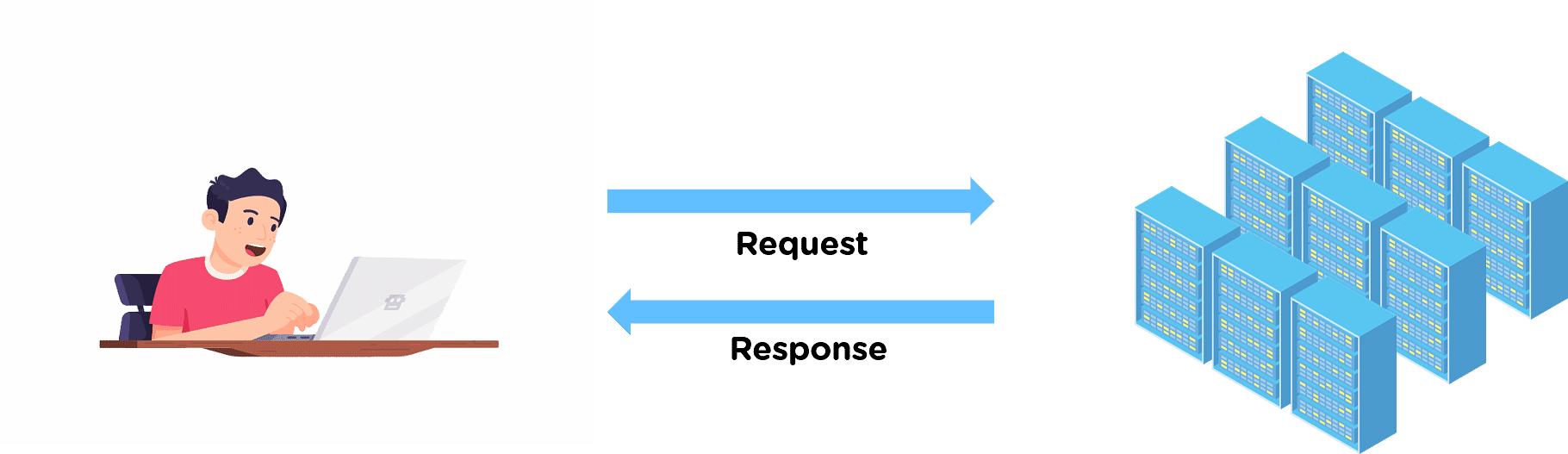
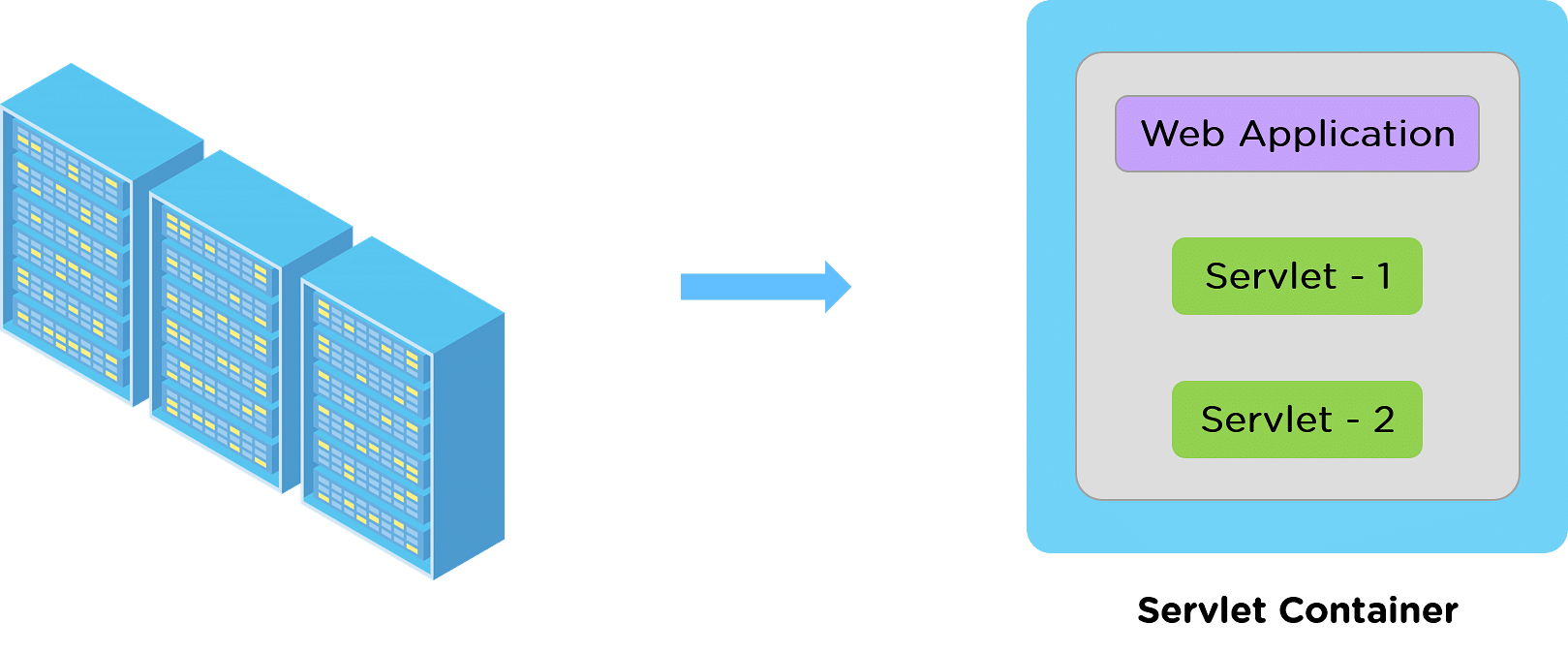
**SERVLETS:**

* A Java program which resides and executes on a server to provide functionality to the server or processing of data on the server.
* They are used to handle the request obtained from the webserver, process the request, produce the response, then send a response back to the webserver.



**Lifecycle:**



* The life cycle 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.

1. If an instance of the servlet does not exist, the Web container

a. Loads the servlet class.

b. Creates an instance of the servlet class.

c. Initializes the servlet instance by calling the init method. Initialization is covered

in Initializing a Servlet.

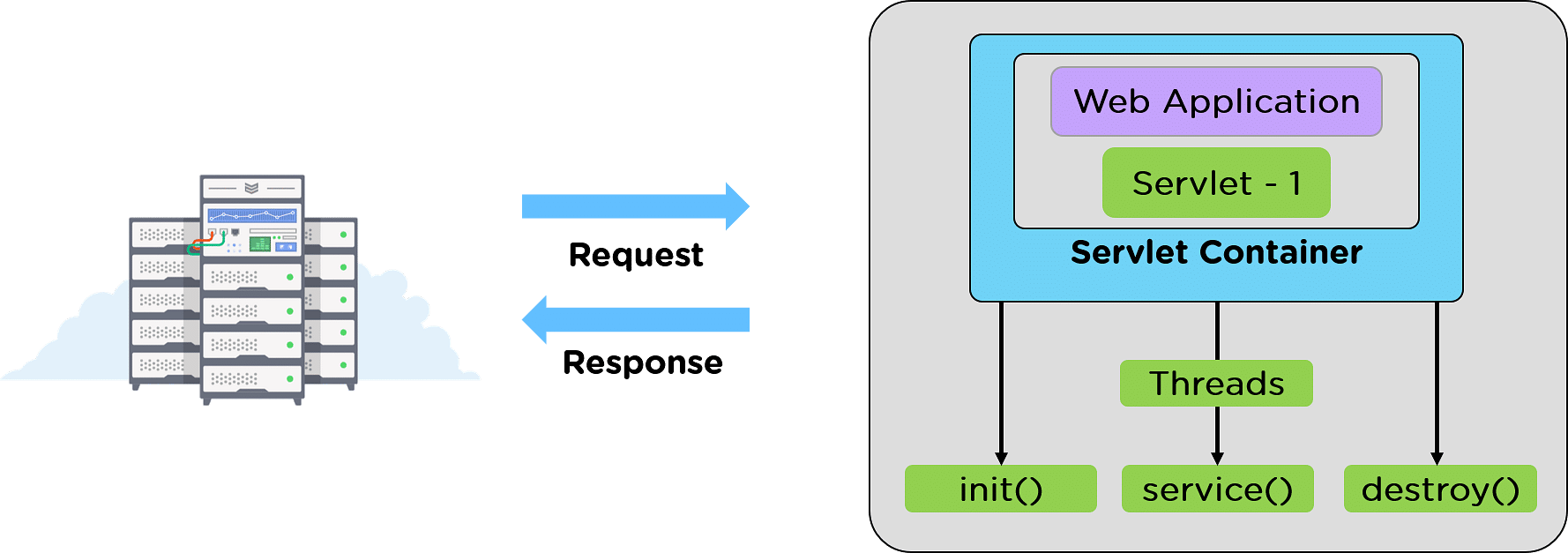
2. Invokes the service method, passing a request and response object. Service methods are

discussed in the section Writing Service Methods.

3. If the container needs to remove the servlet, it finalizes the servlet by calling the

servlet's destroy method.

**The Java Servlet Life cycle includes three stages right from its start to the end until the Garbage Collector clears it.**



**1.init()**

**2.service()**

**3.destroy()**

**1. init()**

* The init() is the germinating stage of any Java Servlet.
* When a URL specific to a particular servlet is triggered, the init() method is invoked.
* Another scenario when the init() method gets invoked is when the servers are fired up.
* With every server starting up, the corresponding servlets also get started, and so does the init() method.
* One important specialty of the init() method is the init() method only gets invoked once in the entire life cycle of the Servlet, and the init() method will not respond to any of the user's commands.

Syntax:

public void init() throws ServletException {

//init() method initializing

}

**2. service()**

* The service() method is the heart of the life cycle of a Java Servlet.
* Right after the Servlet's initialization, it encounters the service requests from the client end.
* The client may request various services like:
* GET
* PUT
* UPDATE
* DELETE
* The service() method takes responsibility to check the type of request received from the client and respond accordingly by generating a new thread or a set of threads per the requirement and implementing the operation through the following methods.
* **doGet() for GET**
* **doPut() for PUT**
* **doUpdate() for UPDATE**
* **doDelete() for DELETE**

service() method Syntax:

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException {

}

**3. destroy()**

* Like the init() method, the destroy() method is also called only once in the Java Servlet's entire life cycle.
* When the destroy() method is called, the Servlet performs the cleanup activities like,Halting the current or background threads.
* Making a recovery list of any related data like cookies to Disk.
* After that, the Servlet is badged, ready for the Garbage collector to have it cleared.

**Syntax:**

public void destroy() {

//destroy() method finalizing

}

**Example:**

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class ex1 extends HttpServlet

{

public void doGet(HttpServletRequest request, HttpServletResponse response) throws

ServletException, IOException

{

PrintWriter pw = response.getWriter();

pw.println("Hello SKCT");

}

}

**Servlet API**

* Two packages contain the classes and interfaces that are required to build servlets.
* They are

**javax.servlet and javax.servlet.http.**

* They constitute the Servlet API.
* They are standard extensions. Therefore, they are not included in the Java Software Development Kit.
* Download Tomcat to obtain their functionality.
* The Servlet API has been in a process of ongoing development and enhancement.
* The javax.servlet package contains a number of interfaces and classes that establish

the framework in which servlets operate.

* The most significant of these is Servlet.
* All servlets must implement this interface or extend a class that implements the interface.

**Interface Description**

Servlet Declares life cycle methods for a servlet.

ServletConfig Allows servlets to get initialization parameters.

ServletContext Enables servlets to log events and access information about their environment.

ServletRequest Used to read data from a client request.

ServletResponse Used to write data to a client response.

SingleThreadModel Indicates that the servlet is thread safe.

**Classes in the javax.servlet package**

**Class Description**

GenericServlet Implements the Servlet and ServletConfig

interfaces.

ServletInputStream Provides an input stream for reading requests from

a client.

ServletOutputStream Provides an output stream for writing responses to

a client.

ServletException Indicates a servlet error occurred.

UnavailableException Indicates a servlet is unavailable.

**Deployment of servlet:**

* Write the servlet class.
* Compile the Servlet class.Extract the HTML form parameters from HttpServletRequest.
* web.xml deployment descriptor file.
* Create the war (web application archive) file.
* Deploy and run the sample web application in tomcat web container.

**Define servlet in deployment descriptor (web.xml) :**

A servlet must be defined into web.xml deployment descriptor.

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app

PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"

"http://java.sun.com/j2ee/dtds/web-app\_2.2.dtd">

<web-app>

<servlet>

<servlet-name> welcomeServlet </servlet-name>

<servlet-class> servletexample.WelcomeServlet

</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name> welcomeServlet </servlet-name>

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

</servlet-mapping>

<welcome-file-list>

<welcome-file>/ welcomeServlet </welcome-file>

</welcome-file-list>

</web-app>

**Web Server**

* A web server is a computer programs that delivers (serves) content, such as web pages, using the Hypertext Transfer Protocol (HTTP), over the World Wide Web.
* The term web server can also refer to the computer or virtual machine running the program.
* Apache Tomcat (or Jakarta Tomcat or simply Tomcat) is an open source servlet container developed by the Apache Software Foundation (ASF).
* Tomcat implements the Java Servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems, and provides a "pure Java"
* HTTP web server environment for Java code to run.

Other examples of web server:Java Web Server,WebLogic

**HTTP GET and POST requests**

* In HTML, two different submission methods for a form can be specified.
* The method is specified inside a FORM element, using the METHOD attribute. The difference between METHOD="GET" (the default) and METHOD="POST" is primarily defined in terms of form data encoding. The official recommendations say that "GET" should be used if and only if the form processing is idempotent, which typically means a pure query form.
* Problems related to long URLs and non-ASCII character repertoires which can make it necessary to use "POST" even for idempotent processing.

**GET**

* The Get is one the simplest Http method. Its main job is to ask the server for the resource.
* If the resource is available then it will given back to the user on your browser.
* That resource may be a HTML page, a sound file, a picture file (JPEG) etc.
* We can say that get method is for getting something from the server.
* It doesn't mean that you can't send parameters to the server.
* But the total amount of characters in a GET is really limited.
* In get method the data we send get appended to the URL so whatever you will send will be seen by other user so can say that it is not even secure.

**POST**

* The Post method is more powerful request.
* By using Post we can request as well as send some data to the server.
* We use post method when we have to send a big chunk of data to the server, like when we have to send a long enquiry form then we can send it by using the post method.

Example

<html>

<head><title>Using Post Method in Form.</title></head>

<body>

<form method="post">

Enter your name: <input type="text” size="20" name="txtName" />

<input type="submit" name="B1" value="Submit" />

<input type="reset" name="B2" value="Reset" />

</form>

</body>

</html>

**Session Tracking**

* A Session refers to the entire request that a single client makes to a server.
* A session is specific to the user and for each user a new session is created to track all the request from that user.
* Every user has a separate session and separate session variable is associated with that session.
* HTTP is stateless protocol and it does not maintain the client state.
* But there exist a mechanism called "Session Tracking" which helps the servers to maintain the state to track the series of requests from the same user across some period of time.
* A session ID is an unique identification string usually a long, random and alpha-numeric string, that is transmitted between the client and the server.
* Session IDs are usually stored in the cookies, URLs (in case url rewriting) and hidden fields of Web pages.

**The Need for Session Tracking**

* HTTP is a stateless protocol.
* Each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server does not automatically maintain contextual information about the client.
* Even with servers that support persistent (keep-alive) HTTP connections and keep sockets open for multiple client requests that occur in rapid succession, there is no built-in support for maintaining contextual information.
* This lack of context causes a number of difficulties.

For example, when clients at an online store add an item to their shopping carts, how does the server know what‘s already in the carts? Similarly, when clients decide to proceed to checkout, how can the server determine which previously created shopping carts are theirs?

There are three typical solutions to this problem:

* cookies,
* URL rewriting
* Hidden form fields.
* User Authorization

**public interface HttpSession**

-Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

-The servlet container uses this interface to create a session between an HTTP client and an HTTP server.

-The session persists for a specified time period, across more than one connection or page request from the user.

-A session usually corresponds to one user, who may visit a site many times. The server can maintain a session in many ways such as using cookies or rewriting URLs.

This interface allows servlets to

* View and manipulate information about a session, such as the session identifier, creation

time, and last accessed time.

* Bind objects to sessions, allowing user information to persist across multiple user connections.

**Methods**

1. **public void setAttribute(java.lang.String name, java.lang.Object value)**

Binds an object to this session, using the name specified. If an object of the same name is already bound to the session, the object is replaced.

2. **public java.lang.Object getAttribute(java.lang.String name)**

Returns the object bound with the specified name in this session, or null if no object is bound under the name

3. **public java.util.Enumeration getAttributeNames()**

An Enumeration of String objects containing the names of all the objects bound to this session.

4. **public void removeAttribute(java.lang.String name)**

Removes the object bound with the specified name from this session. If the session does not have an object bound with the specified name, this method does nothing.

5. **public long getCreationTime()**

Returns the time when this session was created, measured in milliseconds

6. **public long getLastAccessedTime()**

Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT, and marked by the time the container recieved the request.

7. **public int getMaxInactiveInterval()**

Returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses. After this interval, the servlet container will invalidate the session. A negative time indicates the session should never timeout.

8. **public void setMaxInactiveInterval(int interval**)

Specifies the time, in seconds, between client requests before the servlet container will invalidate this session. A negative time indicates the session should never timeout

9. **public void invalidate()**

Invalidates this session then unbinds any objects bound to it.

10. **public java.lang.String getId()**

Returns a string containing the unique identifier assigned to this session. The identifier is assigned by the servlet container.

**Cookies**



* A cookie is a small piece of information that is persisted between multiple client requests.
* A cookie incorporates a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.
* The user logins to the website and surfs through the Website.
* User activities and login credentials will be saved in the form of cookies to help user through easy login or provide with recent browsing details that may be considered as favorites.