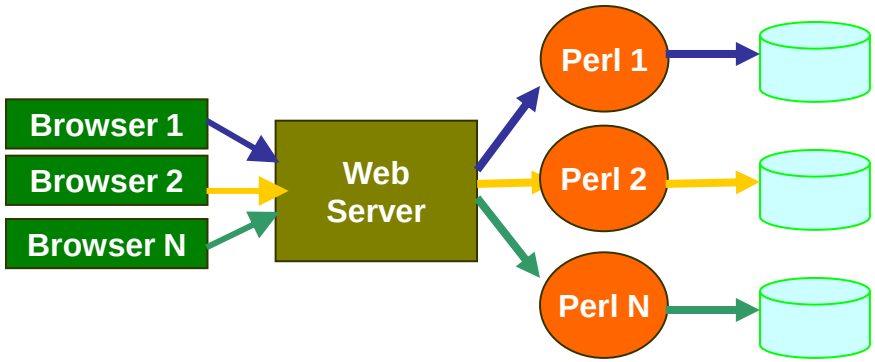
Servlet Fudamentals

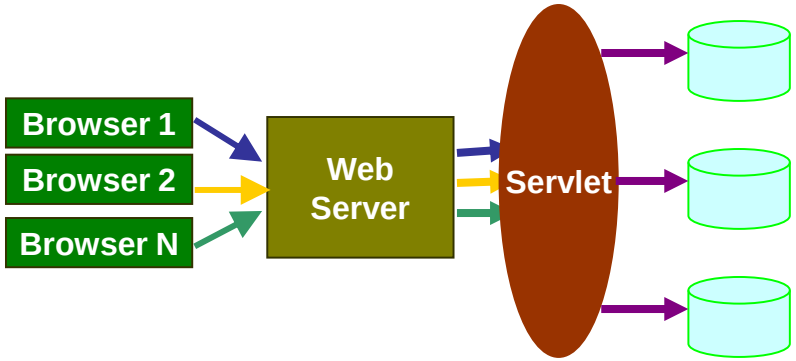
Celsina Bignoli
bignolic@smccd.net

What can you build with Servlets?

- Search Engines
- E-Commerce Applications
- Shopping Carts
- Product Catalogs
- Intranet Applications
- Groupware Applications:
  - bulletin boards
  - file sharing

Servlets vs. CGI





- A Servlet does not run in a separate process.
- A Servlet stays in memory between requests.
- A CGI program needs to be loaded and started for each CGI request.
- There is only a single instance of a servlet which answers all requests concurrently.

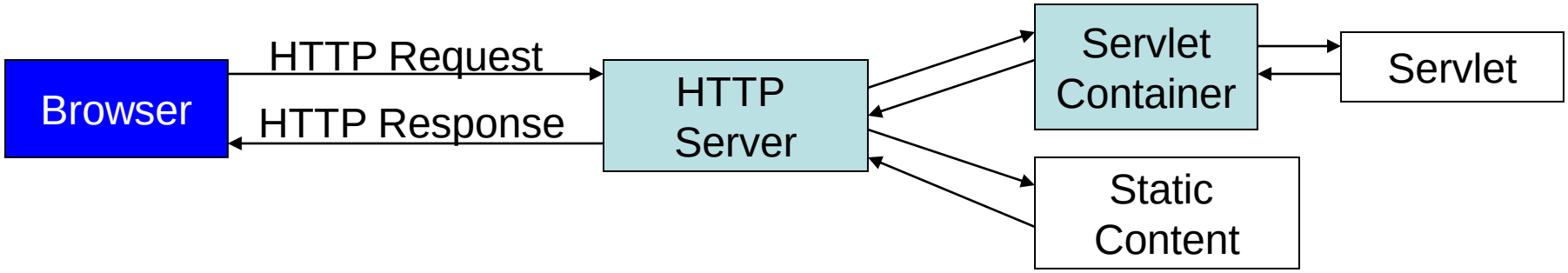
Benefits of Java Servlets

- **Performance**
  - –The performance of servlets is superior to CGI because there is no process creation for each client request.
  - –Each request is handled by the servlet container process.
  - –After a servlet has completed processing a request, it stays resident in memory, waiting for another request.
- **Portability**
  - –Like other Java technologies, servlet applications are portable.
- **Rapid development cycle**
  - –As a Java technology, servlets have access to the rich Java library that will help speed up the development process.
- **Robustness**
  - –Servlets are managed by the Java Virtual Machine.
  - –Don't need to worry about memory leak or garbage collection, which helps you write robust applications.
- **Widespread acceptance**
  - –Java is a widely accepted technology.

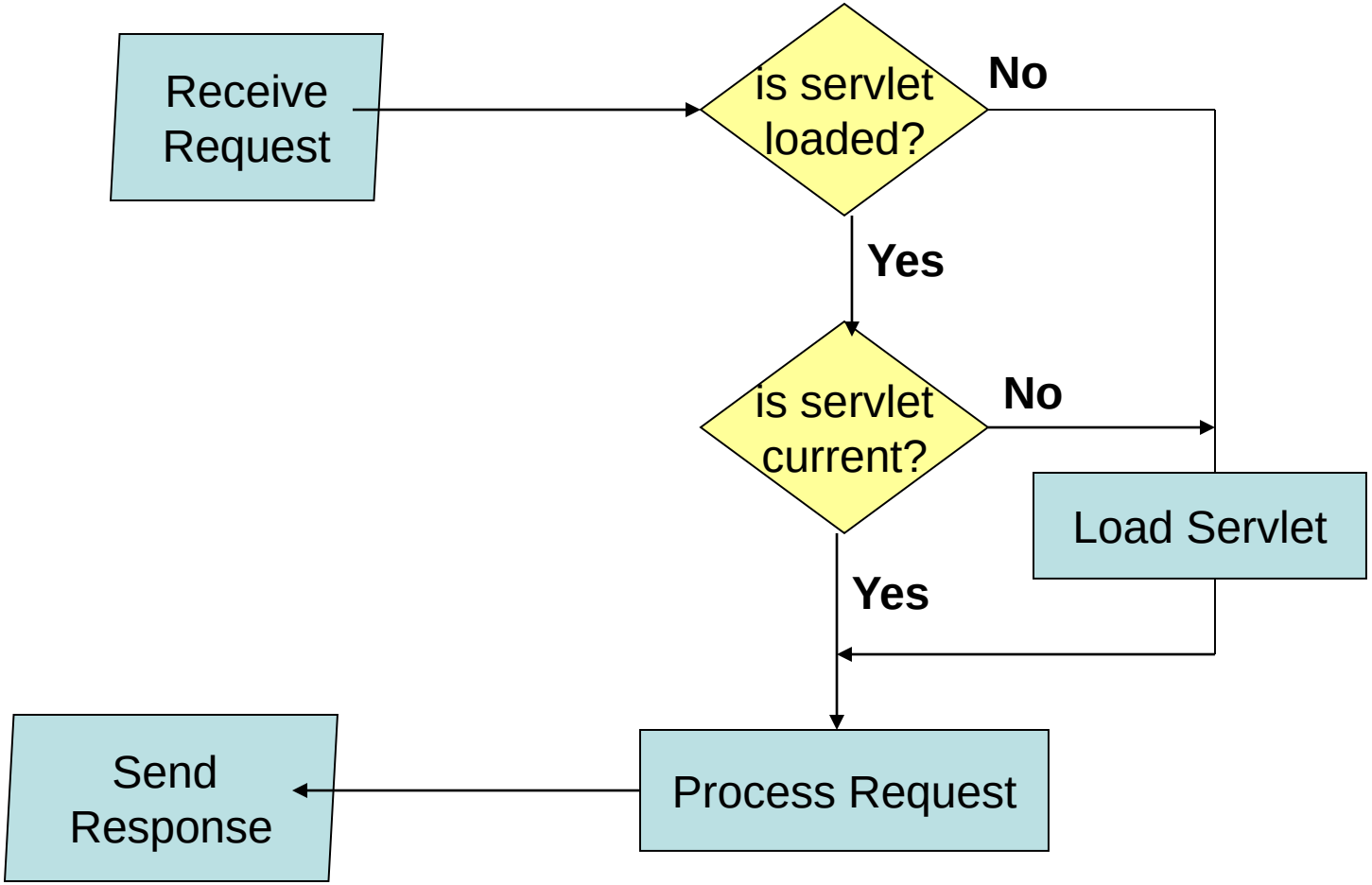
Definitions

- A ***servlet*** is a Java class that can be loaded dynamically into and run by a special web server.
- This servlet-aware web server, is known as ***servlet container***.
- Servlets interact with clients via a request-response model based on HTTP.
- Therefore, a servlet container must support HTTP as the protocol for client requests and server responses.
- A servlet container also can support similar protocols such as HTTPS (HTTP over SSL) for secure transactions.

Servlet Container Architecture



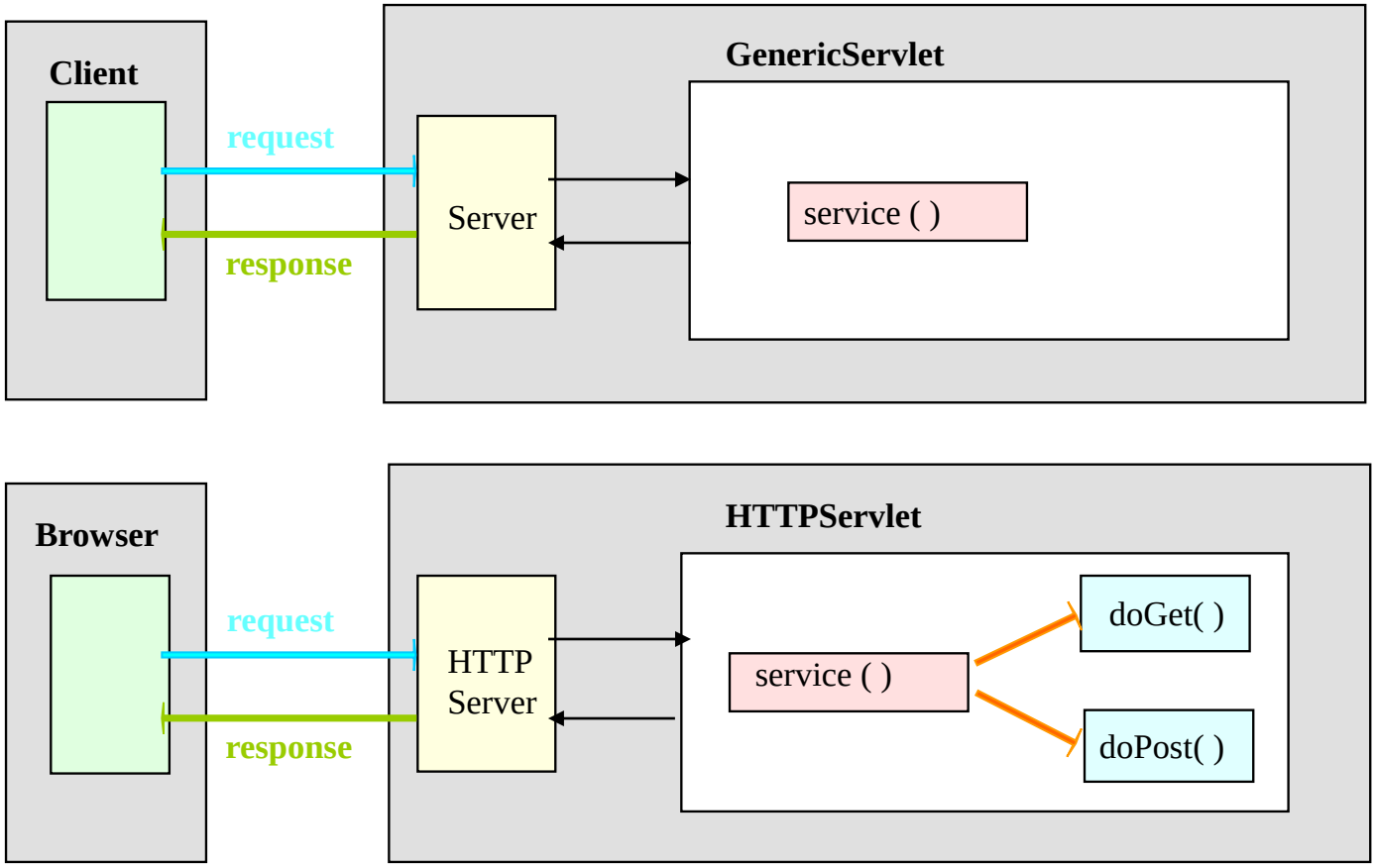
How Servlets Work



Servlet APIs

- Every servlet must implement javax.servlet.Servlet interface
- Most servlets implement the interface by extending one of these classes
  - javax.servlet.GenericServlet
  - javax.servlet.http.HttpServlet

Generic Servlet & HTTP Servlet



Interface javax.servlet.Servlet

- The Servlet interface defines methods
  - –to initialize a servlet
  - – to receive and respond to client requests
  - –to destroy a servlet and its resources
  - – to get any startup information
  - – to return basic information about itself, such as its author, version and copyright.

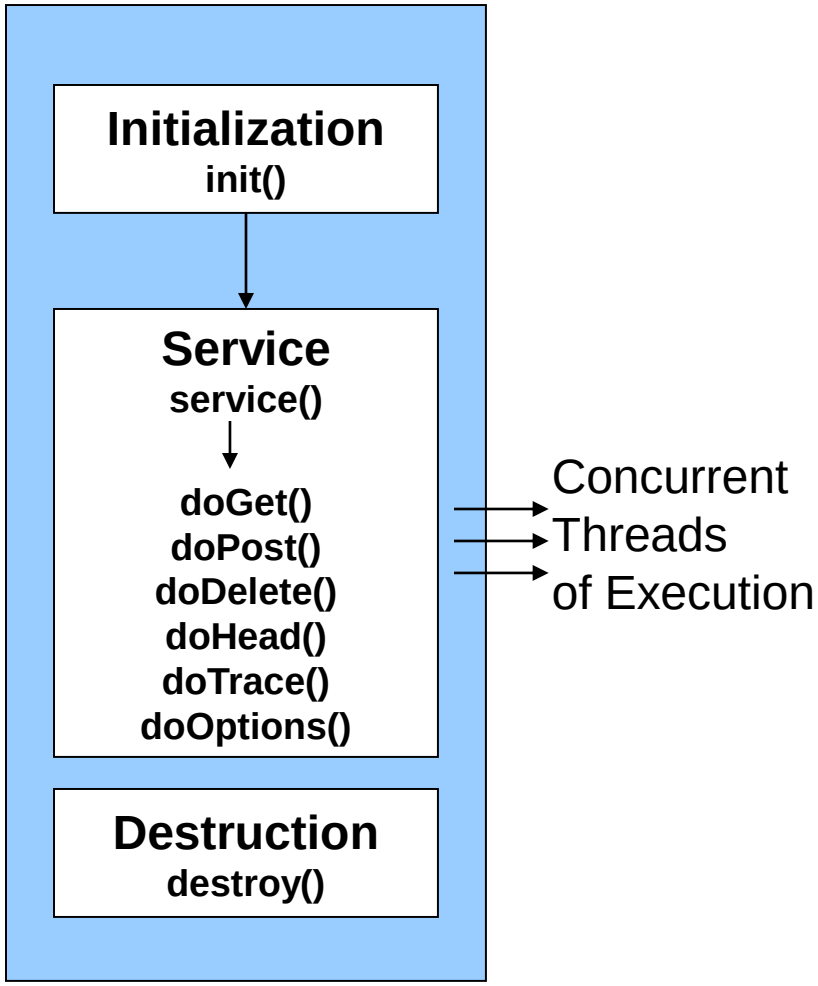
Life Cycle Methods

- Developers need to directly implement this interface only if their servlets cannot (or choose not to) inherit from GenericServlet or HttpServlet.

GenericServlet - Methods

- **void init**(ServletConfig config)
  - –Initializes the servlet.
- void **service**(ServletRequest req, ServletResponse res)
  - –Carries out a single request from the client.
- **void destroy**()
  - –Cleans up whatever resources are being held (e.g., memory, file handles, threads) and makes sure that any persistent state is synchronized with the servlet's current in-memory state.
- ServletConfig **getServletConfig**()
  - –Returns a servlet config object, which contains any initialization parameters and startup configuration for this servlet.
- String **getServletInfo**()
  - –Returns a string containing information about the servlet, such as its author, version, and copyright.

Servlet Life Cycle



HttpServlet - Methods

- void doGet (HttpServletRequest request,
  HttpServletResponse response)
  - handles GET requests
- void doPost (HttpServletRequest request,
  HttpServletResponse response)
  - handles POST requests
- void doPut (HttpServletRequest request,
  HttpServletResponse response)
  - handles PUT requests
- void doDelete (HttpServletRequest request,
  HttpServletResponse response)
  - handles DELETE requests

Servlet Request Objects

- provides client request information to a servlet.
- the servlet container creates a servlet request object and passes it as an argument to the servlet's service method.
- the ServletRequest interface define methods to retrieve data sent as client request:
  - parameter name and values
  - attributes
  - input stream
- HTTPServletRequest extends the ServletRequest interface to provide request information for HTTP servlets

HttpServletRequest - Methods

Enumeration	**getParameterNames**() an Enumeration of String objects, each String containing the name of a request parameter; or an empty Enumeration if the request has no parameters
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 null if the parameter does not exist.
java.lang.String	**getParameter** (java.lang.String name) Returns the value of a request parameter as a String, or null if the parameter does not exist.

HttpServletRequest - Methods

Cookie[]	**getCookies**() Returns an array containing all of the Cookie objects the client sent with this request.
java.lang.String	**getMethod**() Returns the name of the HTTP method with which\thi request was made, for example, GET, POST, or PUT.
java.lang.String	**getQueryString**() Returns the query string that is contained in the request URL after the path.
HttpSession	**getSession**() Returns the current session associated with this request, or if the request does not have a session, creates one.

Servlet Response Objects

- Defines an object to assist a servlet in sending a response to the client.
- The servlet container creates a ServletResponse object and passes it as an argument to the servlet's service method.

HttpServletResponse - Methods

java.io.PrintWriter	**getWriter**() Returns a PrintWriter object that can send character text to the client
void	**setContentType** (java.lang.String type) Sets the content type of the response being sent to the client. The content type may include the type of character encoding used, for example, text/html; charset=ISO-8859-4
int	**getBufferSize**() Returns the actual buffer size used for the response

Steps to Running a Servlet

- Create a directory structure under Tomcat for your application.
- Write the servlet source code.
- Compile your source code.
- deploy the servlet
- Run Tomcat
- Call your servlet from a web browser

Create a Directory Structure

•The webapps directory is the Tomcat installation dir (CATALINA_HOME) is where you store your web applications.
• A *web application* is a collection of servlets and other contents installed under a specific subset of the server's URL namespace.
•A separate directory is dedicated for each servlet application.
•Create a directory called myApp under the webapps directory.
•Create the src and WEB-INF directories under myApp, and create a directory named classes under WEB-INF.
  – The src directory is for your source files, and the classes directory under WEB-INF is for your Java classes.
  – If you have html files, you put them directly in the myApp directory.
•The admin, ROOT, and examples directories are for applications created automatically when you install Tomcat

Write the Servlet Code

- Servlets implement the javax.servlet.Servlet interface.
- Because most servlets extend web servers that use the HTTP protocol to interact with clients, the most common way to develop servlets is by specializing the javax.servlet.http.HttpServlet class.
- The HttpServlet class implements the Servlet interface by extending the GenericServlet base class, and provides a framework for handling the HTTP protocol.
- Its service() method supports standard HTTP requests by dispatching each request to a method designed to handle it.

- In myApp/src, create a file called TestingServlet.java

Servlet Example

```
 1:  import java.io.*;
 2:  import javax.servlet.*;
 3:  import javax.servlet.http.*;
 4:
 5:  public class MyServlet extends HttpServlet
 6:  {
 7:    protected void doGet(HttpServletRequest req,
 8:                         HttpServletResponse res)
 9:    {
10:      res.setContentType("text/html");
11:      PrintWriter out = res.getWriter();
12:      out.println( "<HTML><HEAD><TITLE> Hello You!" +
13:                   "</Title></HEAD>" +
14:                   "<Body> HelloYou!!!</BODY></HTML>" );
14:      out.close();
16:    }
17:   }
```

Compile the Servlet

- Compile the Servlet class
- The resulting TestingServlet.class file should go under myApp/WEB-INF/classes

Deploy the Servlet

- In the Servlet container each application is represented by a servlet context
- each servlet context is identified by a unique path prefix called context path
  - For example our application is identified by /myApp which is a directory under webapps.
- The remaining path is used in the selected context to find the specific Servlet to run, following the rules specified in the deployment descriptor.

Deployment Descriptor

- The deployment descriptor is a XML file called web.xml that resides in the WEB-INF directory whitin an application.

```
<web-app xmlns=http://java.sun.com/xml/ns/j2ee……>

        <display-name>test</display-name>
        <description>test example</description>

   <servlet>
       <servlet-name>Testing</servlet-name>
       <servlet-class>TestingServlet</servlet-class>
   </servlet>

        <servlet-mapping>
                <servlet-name>Testing</servlet-name>
                <url-pattern>/servlet/TestingServlet</url-pattern>
        </servlet-mapping>
</web-app>
```

Run the Servlet

- To execute your Servlet, type the following URL in the Browser's address field:
- http://localhost/myApp/servlet/myServlet

Running service() on a single thread

- By default, servlets written by specializing the HttpServlet class can have multiple threads concurrently running its service() method.
- If you would like to have only a single thread running a service method at a time, then your servlet should also implement the SingleThreadModel interface.
  - This does not involve writing any extra methods, merely declaring that the servlet implements the interface.

Example

```
public class SurveyServlet extends HttpServlet
implements SingleThreadModel
{
/* typical servlet code, with no threading concerns
 * in the service method. No extra code for the
 * SingleThreadModel interface. */
 ...
}
```