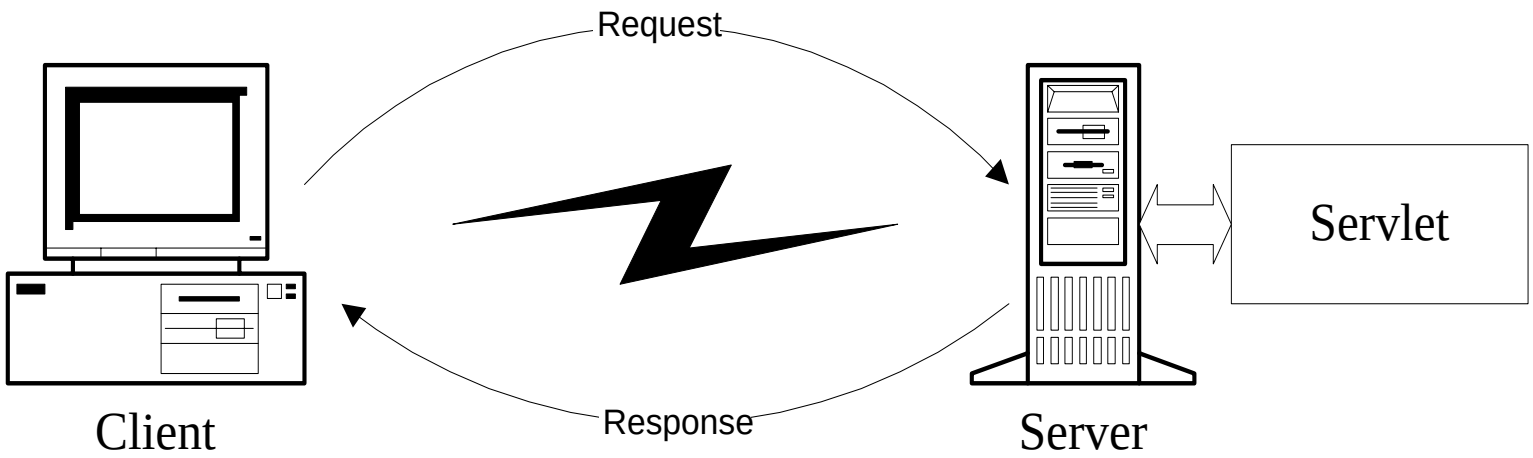
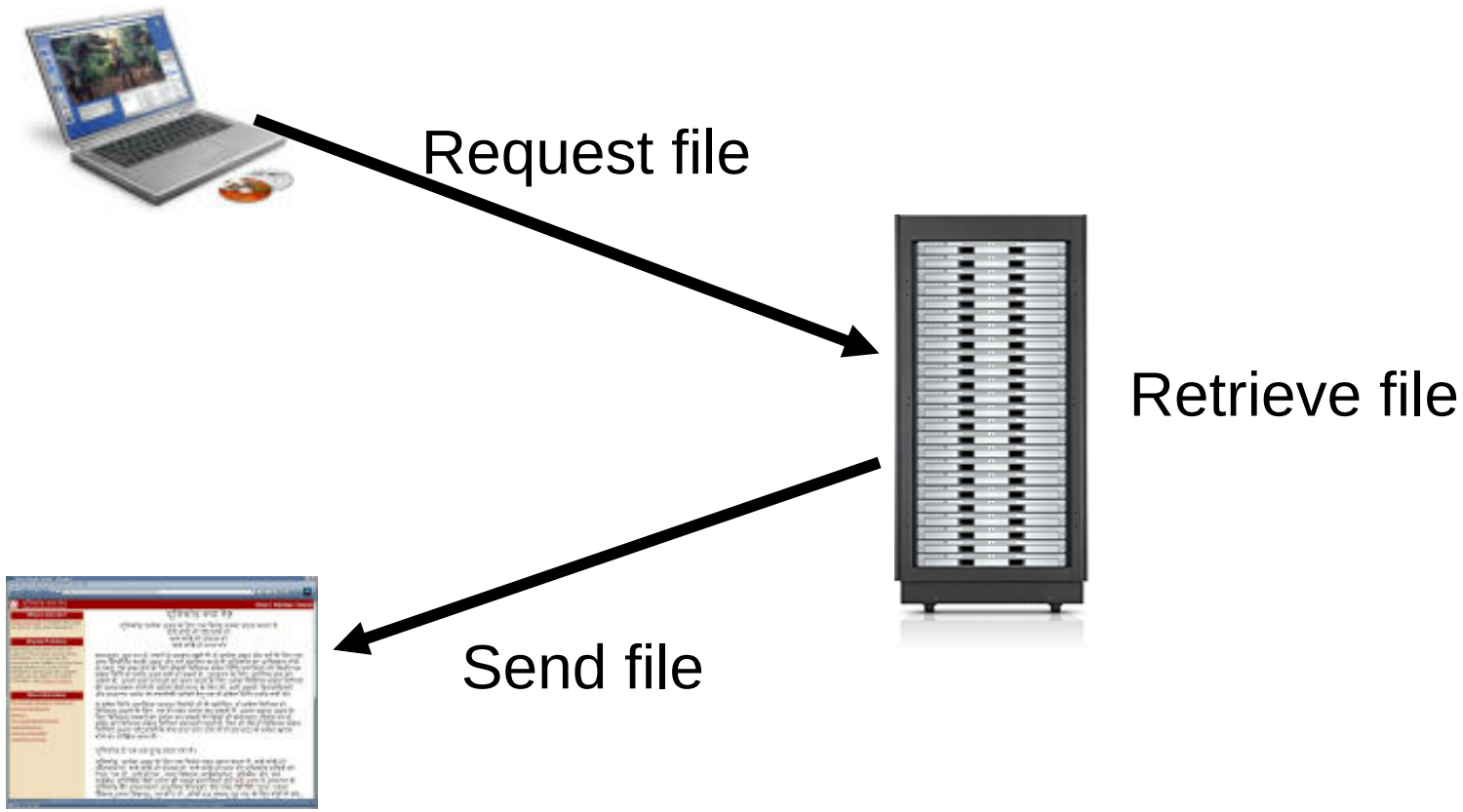
Introduction to Java Servlets

What is java servlet ?

A servlet is a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol. Servlet is an opposite of applet as a server-side applet.



Static Pages



Dynamic Pages

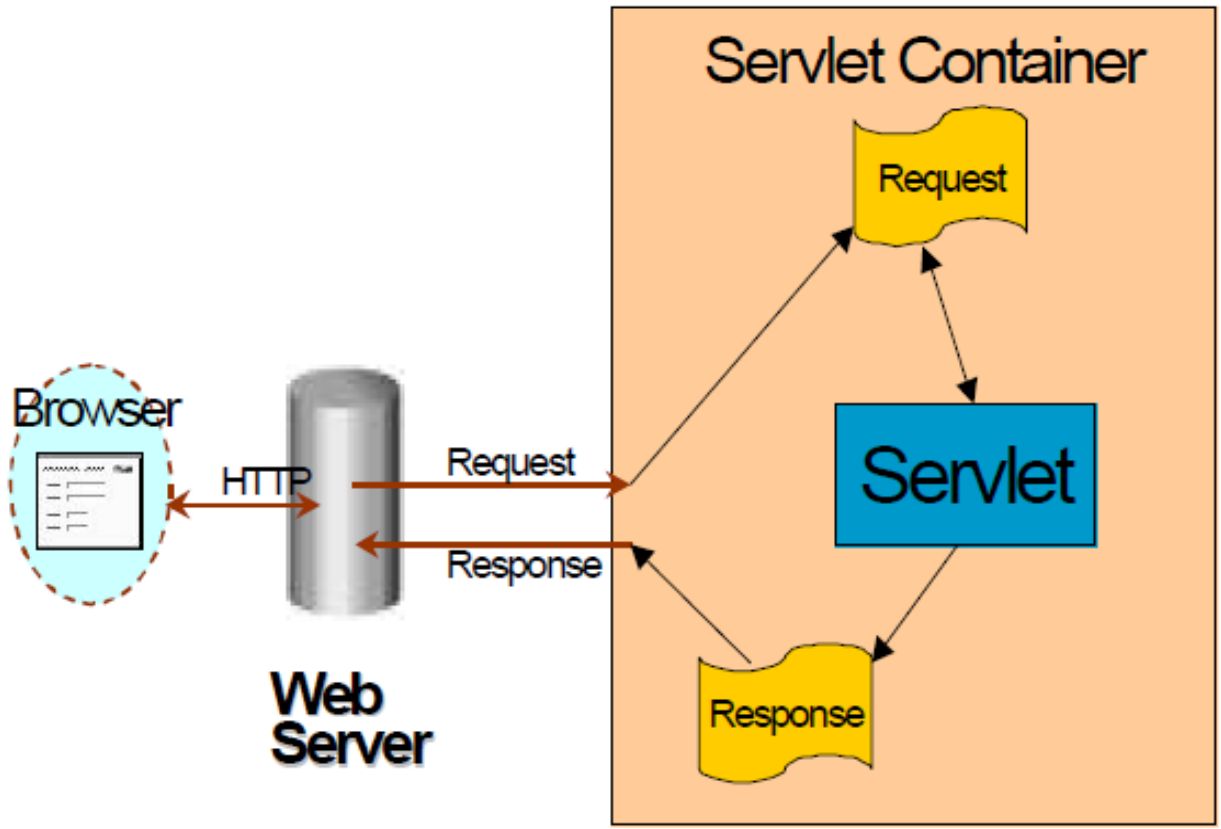
Request service

Do Computation

Generate HTML page with results of computation

Return dynamically generated HTML file

Servlets Request and Response



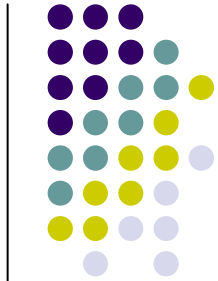
Java Servlet

- Servlets are generic extensions to Java-enabled servers
- Servlets are secure, portable, and easy to use replacement for CGI
- ***Servlet is a dynamically loaded module that services requests from a Web server***
- Servlets are executed within the Java Virtual Machine
- Because the servlet is running on the server side, it does not depend on browser compatibility

Java Servlet Alternatives

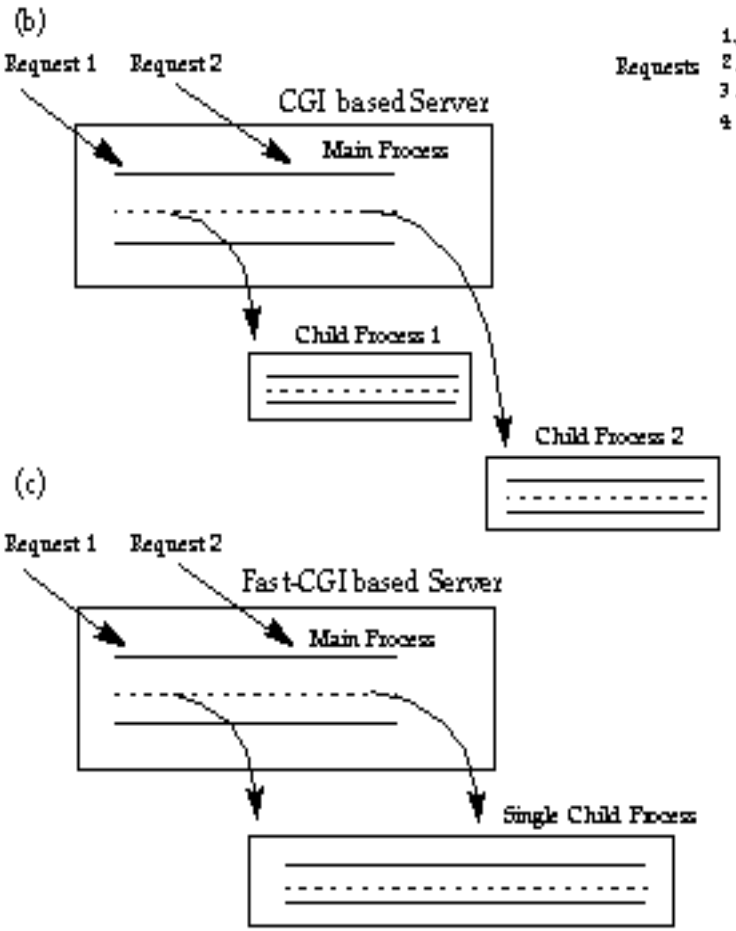
- CGI – Common Gateway Interface
  - New process for every cgi request
    - Slow response time
    - If cgi program terminates before responding to web server, the browser just waits for a response until it times out
- Proprietary APIs
  - ASP
    - Microsoft

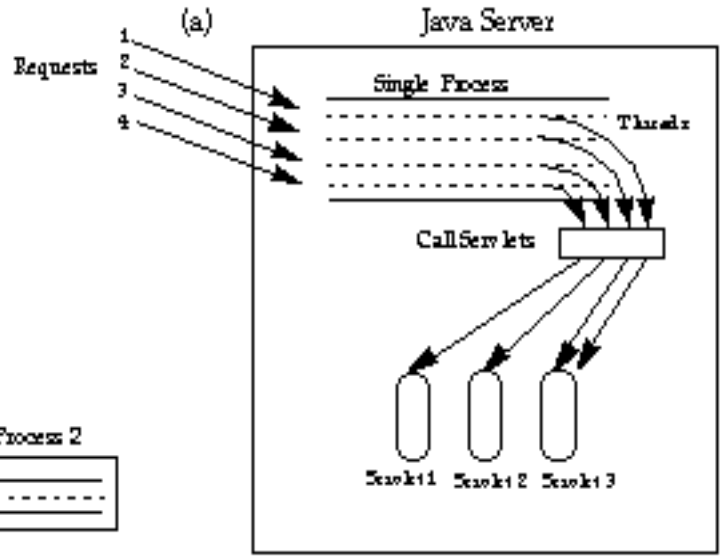
Servlets Vs Others



- vs. Common Gateway Interface (CGI)
  - create new process for each request
    - most platform independent CGI language - Perl
      - start a new instance of interpreter for every request
  - CGI runs in a completely separate process from the Web server
- vs. Server-Side JavaScript
  - only available on certain web servers
- vs. Active Server Pages (ASP)
  - only available on certain web servers

Servlet vs CGI

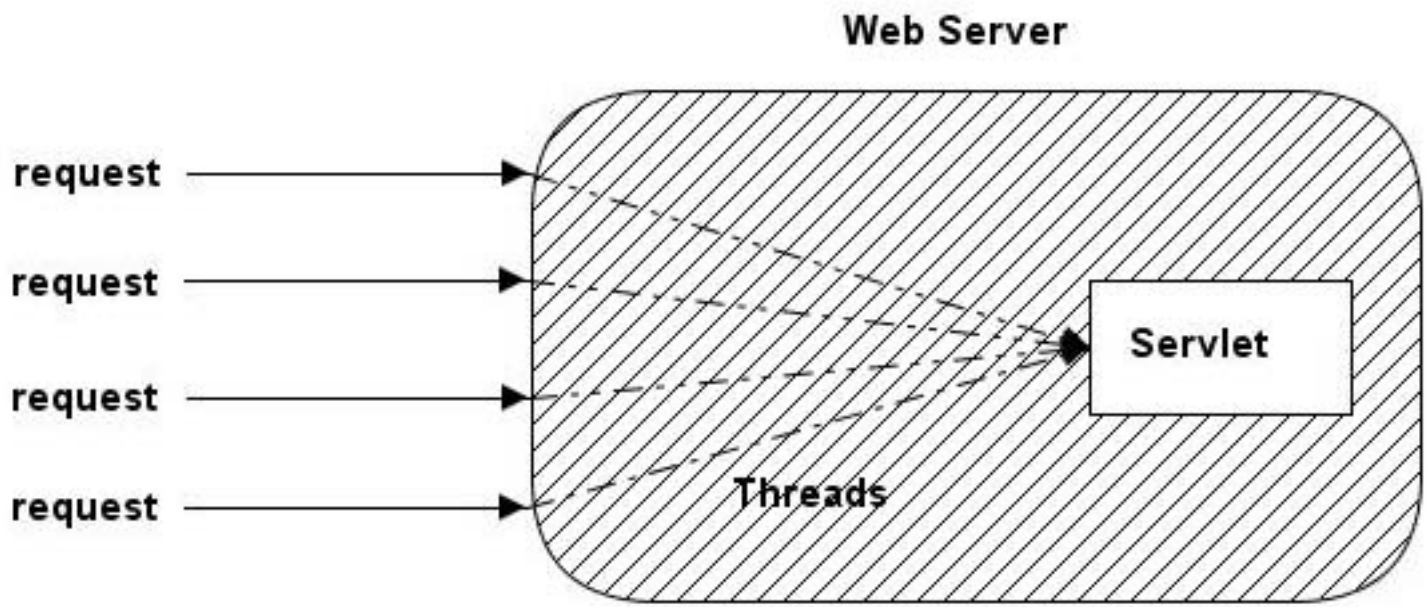




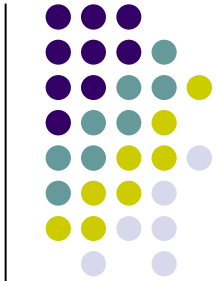
Servlet run as light weight thread in process.

CGI run as heavy weight process.

The Basic Servlet Architecture



Advantage of servlet over CGI



- **PlatForm Independence**

  Servlets can run on any platform. PERL also can be moved from platform to platform while CGI such as C are not portable.

- **Performance**

  Servlets only needs be loaded once, while CGI programs needs to be load for every request so that servlet should performs faster than CGI

- **Security**

  While CGI can do many insecure things, java provided security in language level.

Advantages of Servlets

- Efficiency
  - More efficient – uses lightweight java threads as opposed to individual processes
- Persistency
  - Servlets remain in memory
  - Servlets can maintain state between requests
- Portability
  - Since servlets are written in Java, they are platform independent
- Robustness
  - Error handling, Garbage collector to prevent problems with memory leaks
  - Large class library – network, file, database, distributed object components, security, etc.

Advantages of Servlets

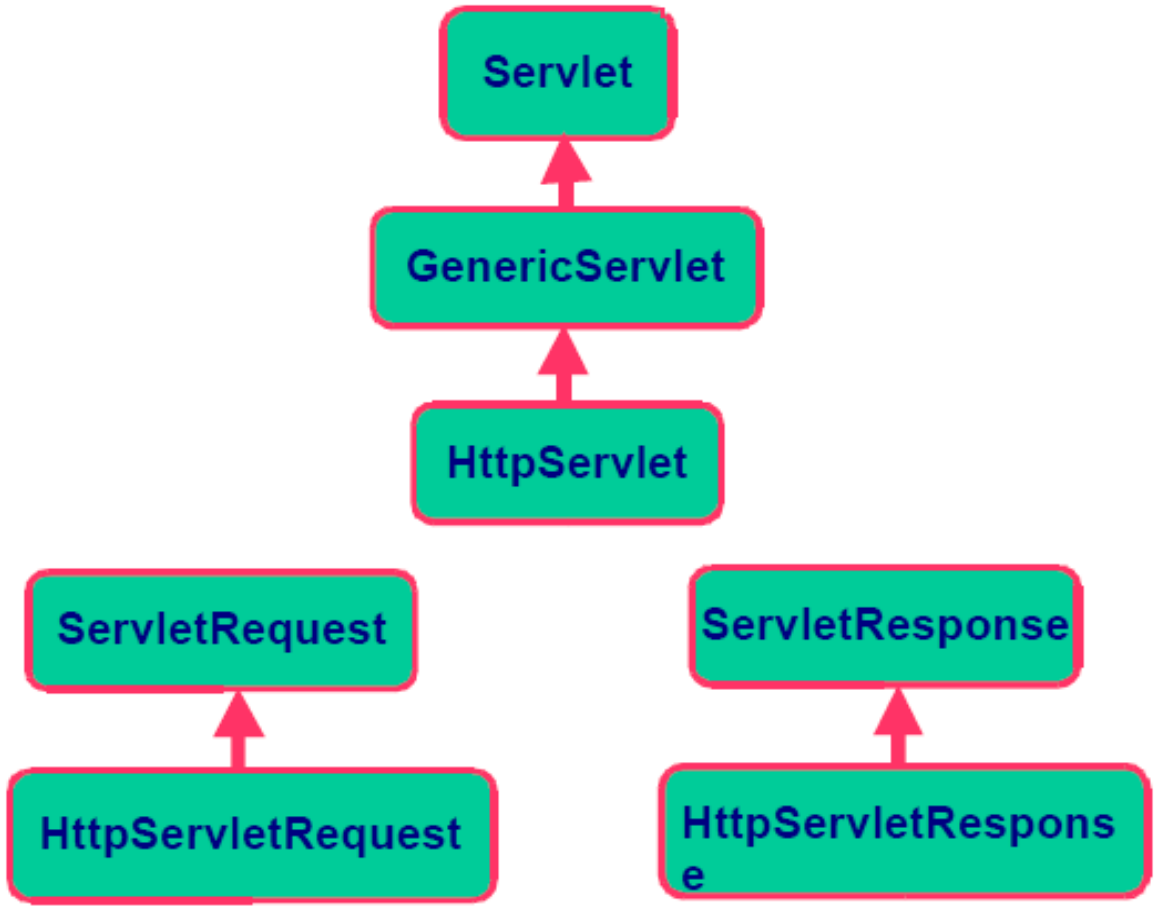
- Extensibility
  - Creating new subclasses that suite your needs
    - Inheritance, polymorphism, etc.
- Security
  - Security provided by the server as well as the Java Security Manager
  - Eliminates problems associated with executing cgi scripts using operating system “shells”
- Powerful
  - Servlets can directly talk to web server
  - Facilitates database connection pooling, session tracking etc.
- Convenient
  - Parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, etc.

Java Servlet Architecture

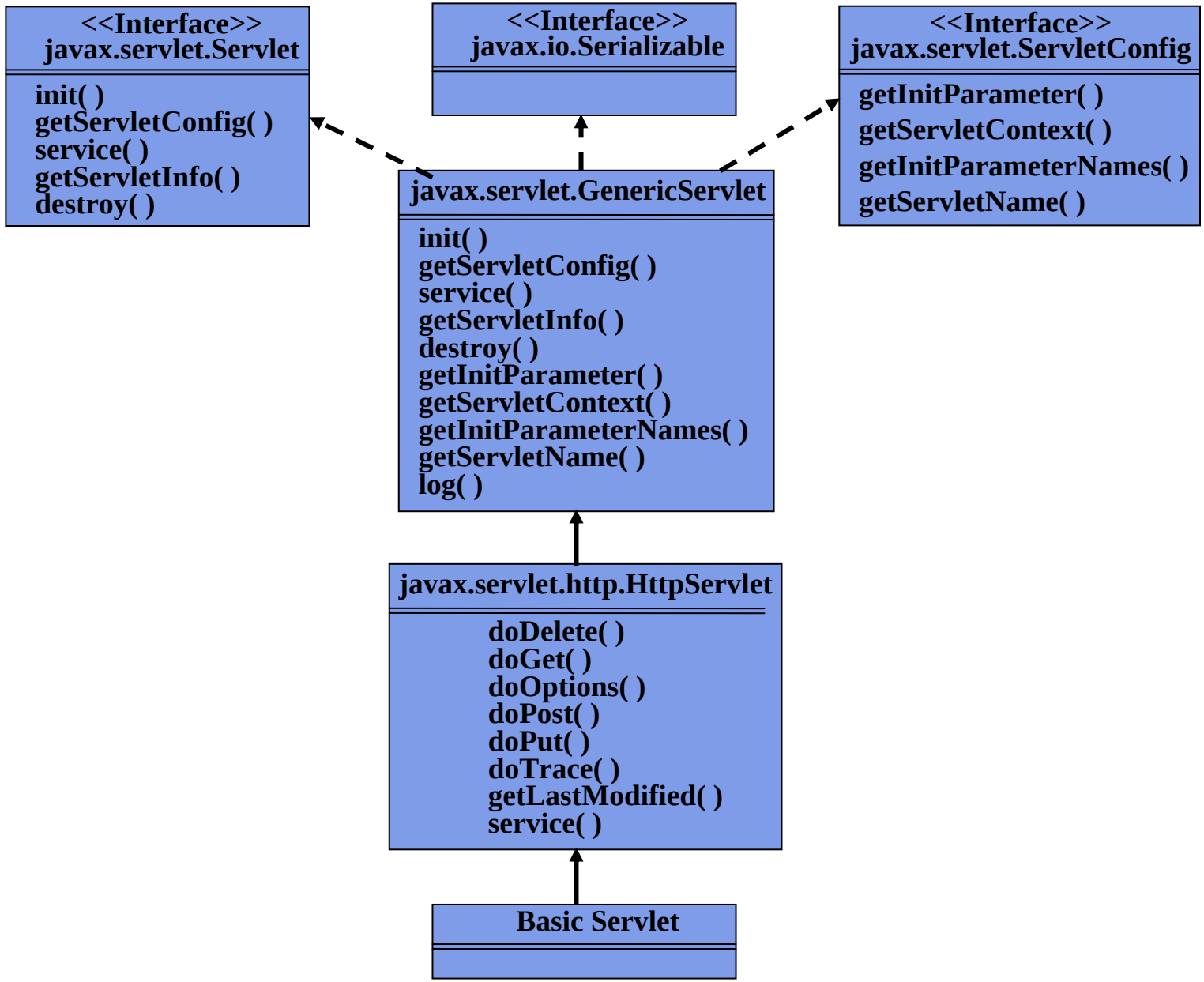
- Two packages make up the servlet architecture
  - ***javax.servlet***
    - Contains generic interfaces and classes that are implemented and extended by all servlets
  - ***javax.servlet.http***
    - Contains classes that are extended when creating HTTP-specific servlets

- The heart of servlet architecture is the interface ***javax.servlet.Servlet***
- It provides the framework for all servlets
- Defines five basic methods – init, service, destroy, getServletConfig and getServletInfo

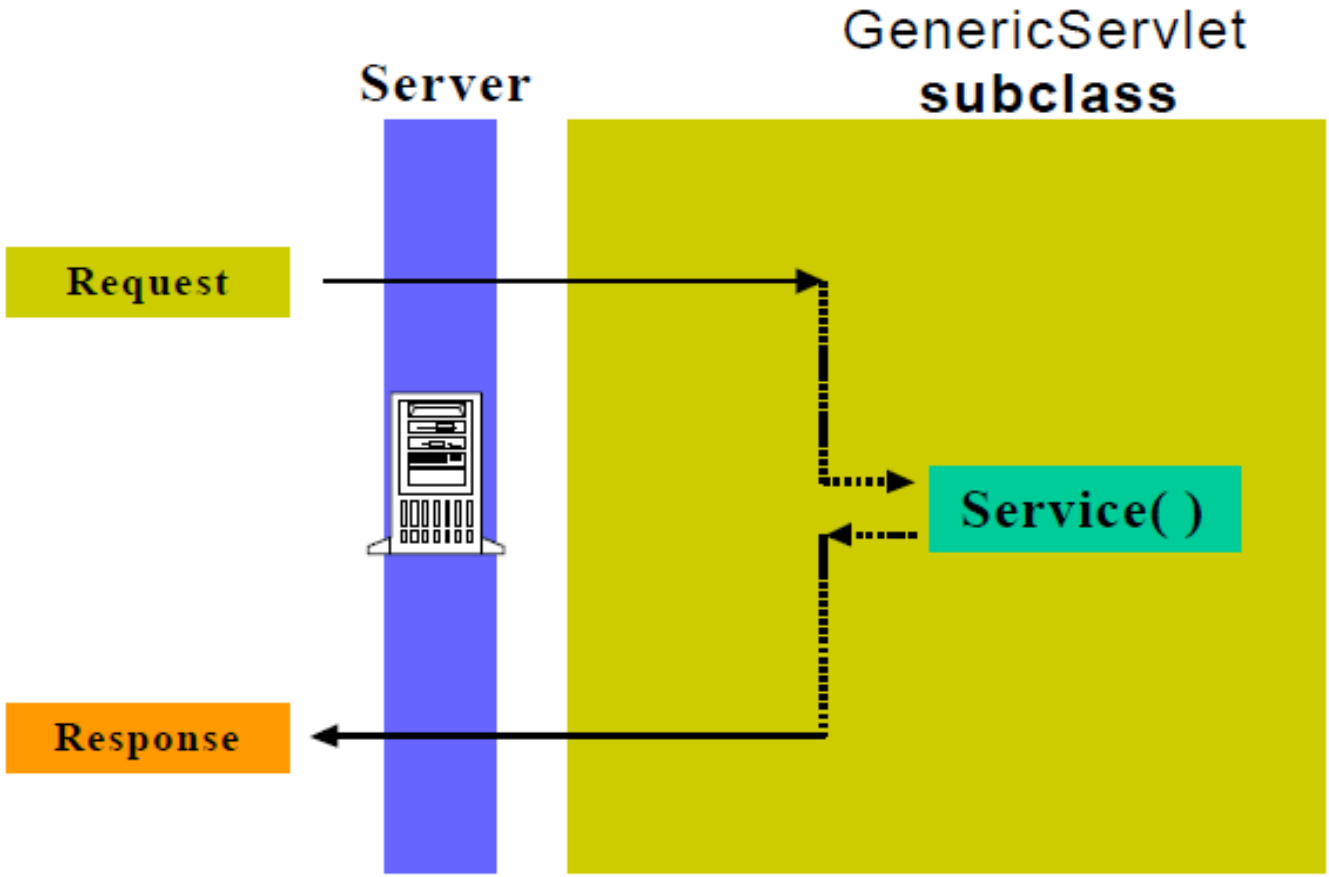
Servlet Classes



Object model of Servlet Framework



Service() method



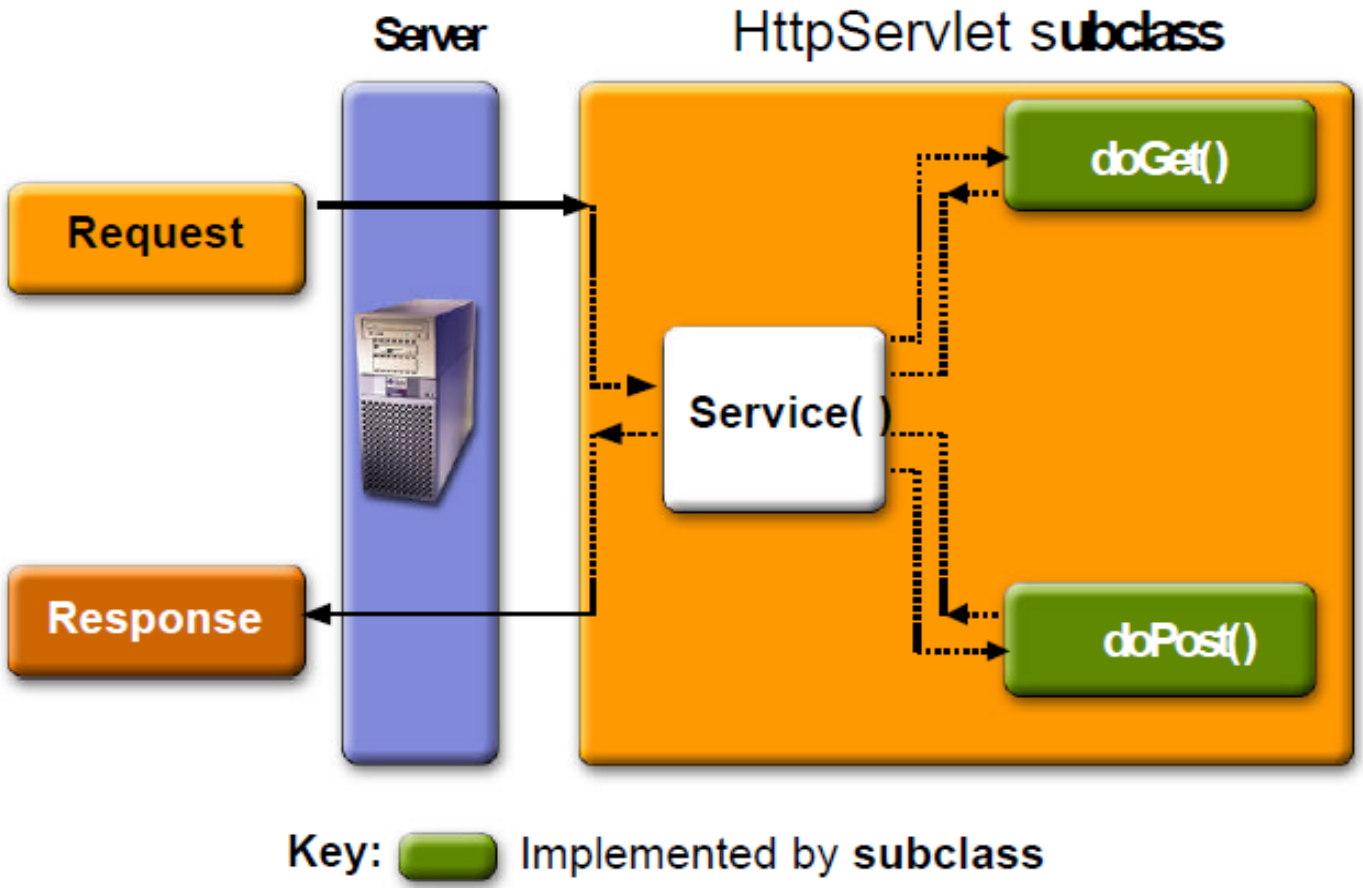
GenericServlet & HttpServlet

- *HttpServlet* class is extended from *GenericServlet* class
- *GenericServlet.service()* method has been defined as an abstract method
- The two objects that the service() method receives are ***ServletRequest* and *ServletResponse***
- ServletRequest Object
  - Holds information that is being sent to the servlet
- ServletResponse Object
  - Holds data that is being sent back to the client

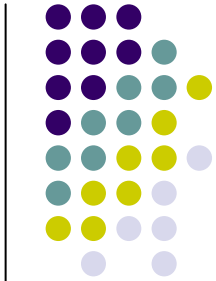
GenericServlet & HttpServlet

- Unlike the *GenericServlet*, when extending *HttpServlet*, don't have to implement the *service()* method. It is already implemented for you
- When *HttpServlet.service( )* is invoked, it calls *doGet( )* or *doPost( ),* depending upon how data is sent from the client
- *HttpServletRequest* and *HttpServletResponse* classes are just extensions of *ServletRequest* and *ServletResponse* with HTTP-specific information stored in them

GET and POST

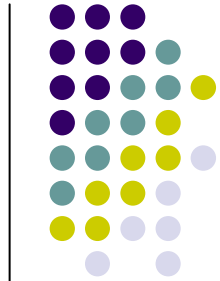

Server
HttpServlet subclass
Request
Response
Service( )
doGet()
doPost()
Key:
Implemented by subclass

GET Method



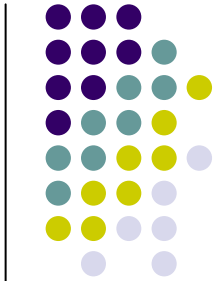
- The form data is encoded and then appended to the URL after ? mark
- The information contained in the part of the URL after the ? mark is called the QUERY_STRING, which consists of a string of name=value pairs separated by ampersands (&)
- **GET http://www.vit.ac.in/cgi-bin/example/simple.pl?**first=anita&last=kumar
-

GET vs. POST



- Above examples used the `GET` method to handle the data from the form.
- The form data was concatenated to the CGI URL
- In the `POST` method the data is sent to the CGI separately, in the request body.
- `GET` method is not secure, the data is visible in URL.
- `GET` is suitable for small amounts of data (limited to 1K), but not for larger amounts.

HTTP request methods



- Syntax of using doGet()

```
public void doGet (HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
...servlet code goes here...
}
```

Syntax of using doPost()

```
public void doPost (HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException
{
...servlet code goes here...
}
```

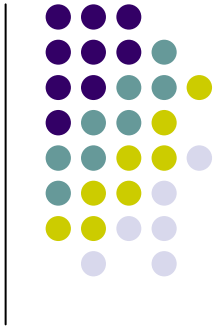
Life Cycle of a Servlet

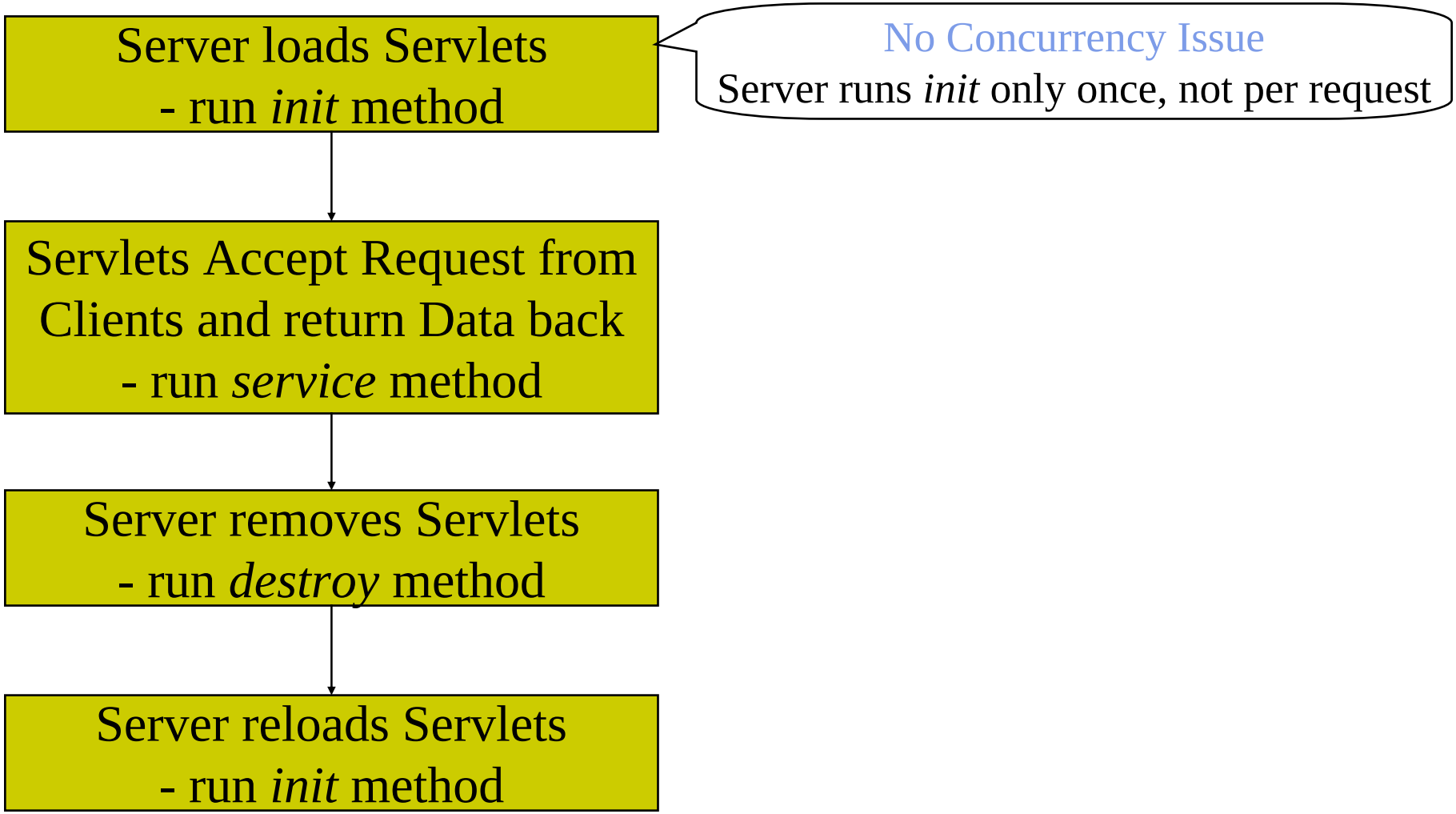
- Servlets operate in the context of a request and response model managed by a servlet engine

- The engine does the following

  - Loads the servlet when it is first requested
  - Calls the servlet's ***init( )*** method
  - Handles any number of requests by calling the servlet's ***service( )*** method
  - When shutting down, calls each servlet's ***destroy( )*** method

Servlet Lifecycle





Life Cycle – *init( )* method

- **Request for a servlet received by the servlet engine**
- **Checks to see if the servlet is already loaded**
- **If not, uses a class loader to get the required servlet class and instantiates it by calling the constructor method**
- **After the servlet is loaded, but before it services any requests, the *init ( )* method is called**
- **Inside *init( ),* the resources used by the servlet are initialized. E.g: establishing database connection**
- **This method is called only once just before the servlet is placed into service**
- **The init( ) method takes a *ServletConfig* object as a parameter**

Life Cycle – *service( )* method

- The *service( )* method handles all requests sent by a client
- It cannot start servicing requests until the *init( )* method has been executed
- Only a single instance of the servlet is created and the servlet engine dispatches each request in a single thread
- The *service( )* method is used only when extending *GenericServlet* class
- Since servlets are designed to operate in the HTTP environment, the *HttpServlet* class is extended
- The ***service(HttpServletRequest, HttpServletResponse)*** method examines the request and calls the appropriate *doGet()* or *doPost()* method.
- A typical Http servlet includes overrides to one or more of these subsidiary methods rather than an override to *service()*

Life Cycle – *destroy( )* method

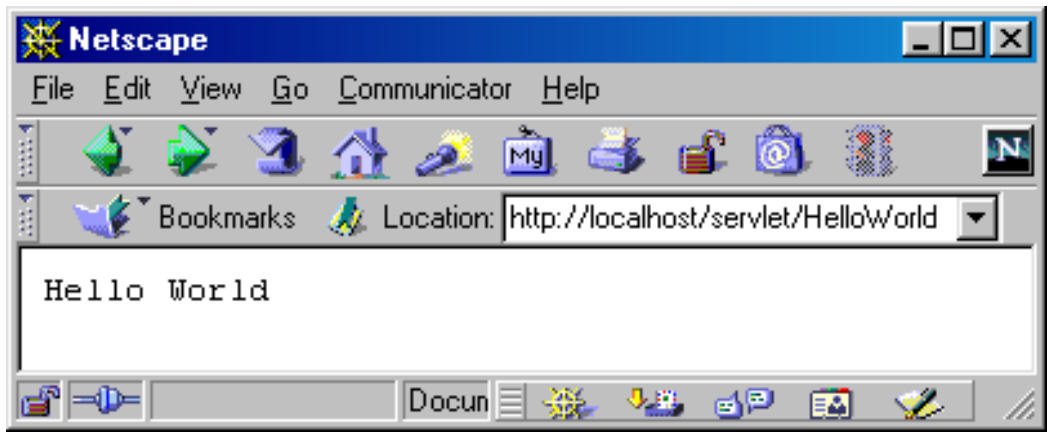
- This method signifies the end of a servlet's life
- The resources allocated during init( ) are released
- Save persistent information that will be used the next time the servlet is loaded
- The servlet engine unloads the servlet
- Calling destroy( ) yourself will not acutally unload the servlet. Only the servlet engine can do this

Simple Servlet Template

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletTemplate extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {


    PrintWriter out = response.getWriter();
    // Use "out" to send content to browser
  }
}
```

A Simple Servlet That Generates Plain Text

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("Hello World");
  }
}
```



A Servlet That Generates HTML

```
import java.io.*;
import javax.servlet.*;
 import javax.servlet.http.*;
public class HelloWorld extends HttpServlet
    {
     public void doGet(HttpServletRequest request,
    HttpServletResponse response) throws IOException,
    ServletException
    {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>"); out.println("<head>");
    out.println("<title>Hello World!</title>");
    out.println("</head>"); out.println("<body>");
    out.println("<h1>Hello World!</h1>");
    out.println("</body>"); out.println("</html>");
} }
```

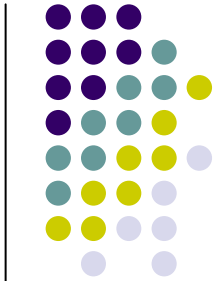
Displaying Date in Servlet

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class DisplayingDate extends HttpServlet{
   public void doGet(HttpServletRequest request, HttpServletResponse
  response) throws ServletException, IOException{
  PrintWriter pw = response.getWriter();
  Date today = new Date();
  pw.println("<html>"+"<body><h1>Today Date is</h1>");
  pw.println("<b>"+ today+"</b></body>"+ "</html>");
  }
}
```

HTML Form

- <html>
  <head>
  <title>Introductions</title>
  </head>
  <body>
  <form method=GET action="/servlet/name">
  <input type=text name="name1"><P>
  <input type=submit>
  </form>
  </body>
  </html>

The corresponding Servlets Page

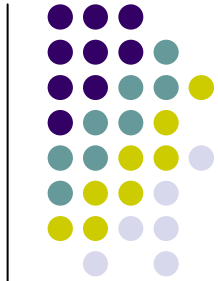


```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class name extends HttpServlet
{
public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
{
res.setContentType("text/html");
PrintWriter out = res.getWriter();
String name = req.getParameter("name1");
out.println("<html>");
out.println("<head><title>Hello, " + name + "</title></head>");
out.println("<body>");
out.println("Hello, " + name);
out.println("</body></html>");
}
}
```

Servlet Life Cycle Summary

- init
  - Executed once when the servlet is first loaded.
    Not called for each request.
- service
  - Called in a new thread by server for each request.
    Dispatches to doGet, doPost, etc.
    Do not override this method!
- doGet, doPost
  - Handles GET, POST, etc. requests.
  - Override these to provide desired behavior.
- destroy
  - Called when server deletes servlet instance.
    Not called after each request.

Form Processing



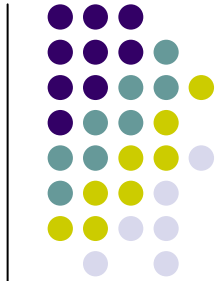
- **Reading Form Data using Servlet:**
- Servlets handles form data parsing automatically using the following methods depending on the situation:
- **getParameter():** You call request.getParameter() method to get the value of a form parameter.
- **getParameterValues():** Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- **getParameterNames():** Call this method if you want a complete list of all parameters in the current request.

Get method

```
<html>
<body>
<form action="HelloForm" method="GET">
First Name: <input type="text" name="first_name"><br
    />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloForm extends HttpServlet
{
public void doGet(HttpServletRequest request,    HttpServletResponse response)
       throws ServletException, IOException
        {
       response.setContentType("text/html");
      PrintWriter out = response.getWriter();

     out.println("<html>\n" +
              "<head><title>" + title + "</title></head>\n" +
             "<body bgcolor=\"#f0f0f0\">\n" +
               “<b>First Name</b>: “+

           request.getParameter("first_name") + "\n" +

            "  <b>Last Name</b>: "    +

               request.getParameter("last_name") + "\n" +

            "</body>                    </html>");
                }
               }
```

Using Post method



```
<html>
<body>
<form action="HelloForm" method=“POST">
First Name: <input type="text" name="first_name"><br
    />
Last Name: <input type="text" name="last_name" />
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

```
// Method to handle POST method request.
  public void doPost(HttpServletRequest
   request, HttpServletResponse response)
   throws ServletException, IOException
   {     doGet(request, response);  }
```

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloForm extends HttpServlet
{
public void doGet(HttpServletRequest request,    HttpServletResponse response)
       throws ServletException, IOException
        {
      response.setContentType("text/html");
     PrintWriter out = response.getWriter();

    out.println("<html>\n" +
              "<head><title>" + title + "</title></head>\n" +
             "<body bgcolor=\"#f0f0f0\">\n" +
               “<b>First Name</b>: “+

          request.getParameter("first_name") + "\n" +

           "  <b>Last Name</b>: "    +

              request.getParameter("last_name") + "\n" +

           "</body>                    </html>");
               }

public void doPost(HttpServletRequest request, HttpServletResponse response)      throws
     ServletException, IOException
     {    doGet(request, response);  }
}
```

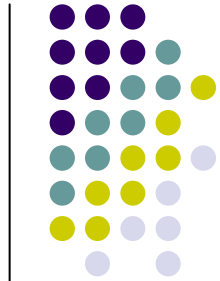
Redirection

Page redirection is generally used when a document moves to a new location and we need to send the client to this new location

The simplest way of redirecting a request to another page is using method **sendRedirect()** of response object

```
response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    String name = request.getParameter("username");
    String password = request.getParameter("password");
    if(name.equals(“VIT")&& password.equals(“student"))
    {
    response.sendRedirect(“http://vit.ac.in/stu.asp");
    }
    else
    {
    pw.println("u r not a valid user");
    }
```

Request related methods



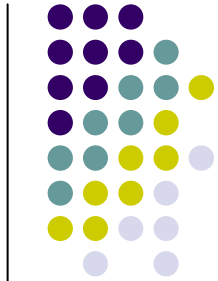
CGI Environment variables and the corresponding Servlet Methods

SERVER_NAME	request.getServerName()
SERVER_PROTOCOL	request.getProtocol()
SERVER_PORT	request.getServerPort()
REQUEST_METHOD	request.getMethod()
QUERY_STRING	request.getQueryString()
CONTENT_TYPE	request.getContentType()
CONTENT_LENGTH	request.getContentLength()

Response related methods

- response.setContentType()
- response.addCookie(Cookie cookie)
- response.addHeader(String name, String value)
- response.setHeader(String name, String value)
- response.sendRedirect(String)

Multi tier applications



Using JDBC from Servlets

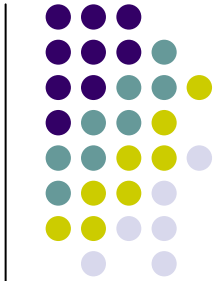
Three-tier distributed applications

- User interface
- Business logic
- Database

Web servers often represent the middle tier

Three-tier distributed application example

- Servlet - Business Logic
- Html – User Interface
- database - DB



```
Try
    {
     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

  Connection theConnection = DriverManager.getConnection("jdbc:odbc:student",
       "admin", "");

   Statement theStatement=theConnection.createStatement();

   ResultSet theResult=theStatement.executeQuery("select * from stu");

  while(theResult.next())
       {
       out.println(theResult.getString(1));
       out.println(theResult.getString(2));
       }

       theResult.close();
       theStatement.close();
       theConnection.close();
       }

catch(Exception e)
       {
          out.println(e.getMessage());
        }
```