Communication

Redirect: Servlet- browser communication

- Sometime the resource requested may be moved to another server either temporarily or permanently. In such case, we may want to send back a message to browser to redirect the page to another location.
- **`HttpServletResponse`** method

  `void sendRedirect(java.lang.String location) throws java.io.IOException`

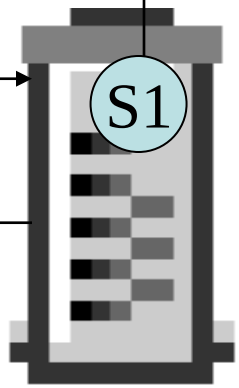
  This method sends a temporary redirect response to client with status code 3xx. Location specifies the URL page to which the redirection is going to be.
- Usage:

  `response.sendRedirect("http://localhost:8080/direct/s?name='Potter'")`

Status 3xx

```
public void doGet(){
response.sendRedirect("http://pondser/app2);
…
}
```

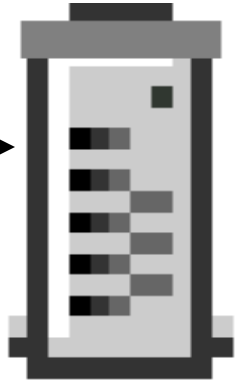
Request for servlet S1



Server

Response packet
HTTP/1.1 302
…

Request for http://pondser/app2



PondServer

Status code (302) indicating that the resource reside temporarily under a different URI

More on `sendRedirect`

- If the response has already been committed, **`sendRedirect()`** method throws an **`IllegalStateException`**
- The URL can be relative . The servlet container converts the relative URL to an absolute URL before sending the response to the client.

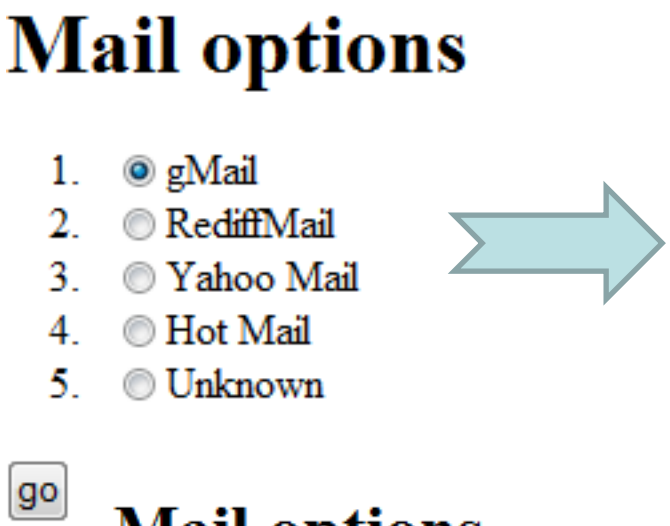
Sending Error: Servlet- browser communication

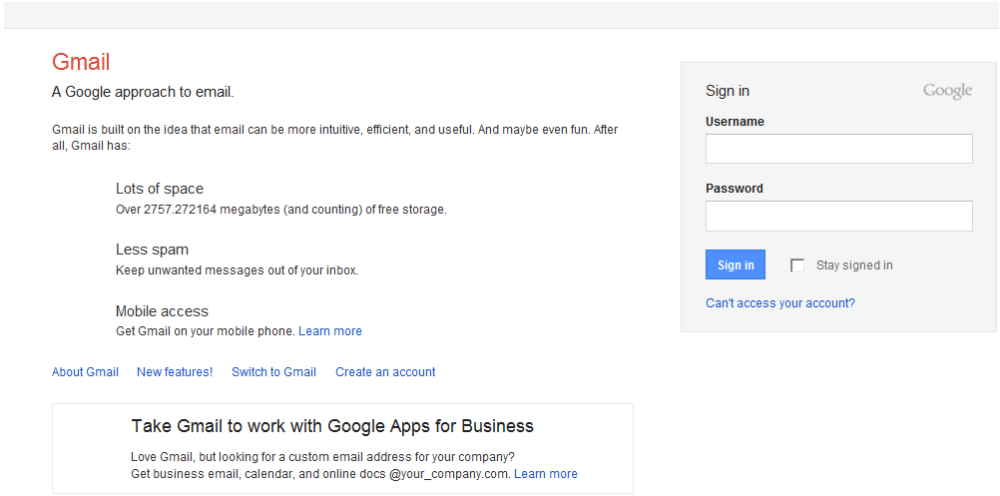
- **`HttpServletResponse`** method
  - **`void sendError(int sc,String msg) throws IOException`**
  - **`void sendError(int sc) throws IOException`**

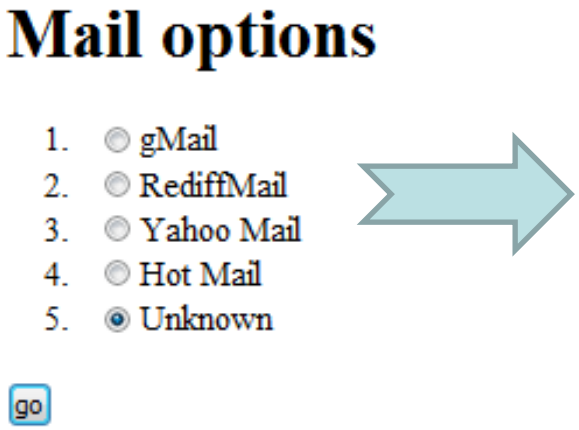
  is used to send an error response to the client using the specified status code represented by int sc and and clearing the buffer
- For sc static constant defined in this class that represents the HTTP response Status code is used.
- The first method allows sending a descriptive message as well.
- Example: **`response.sendError(response.SC_NOT_FOUND, "No recognized search engine specified.");`**
- The advantage of **`sendError`** over **`setStatus`** is that, with **`sendError`**, the server automatically generates an error page showing the error message.

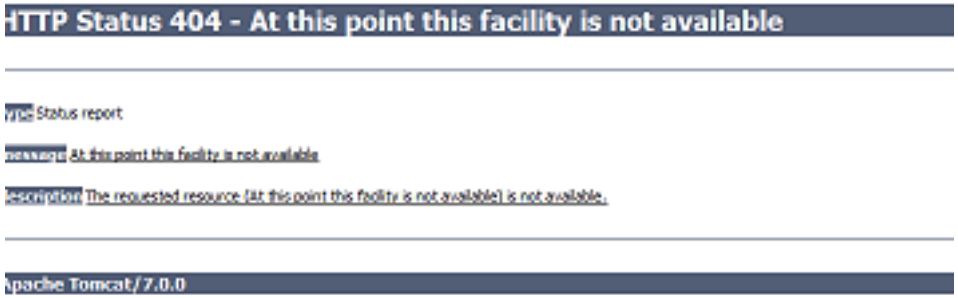
Example: sendRedirect() and sendError()

- HTML page displays email options of which one is unknown
- If unknown option is an 404 error page is displayed.









index.html

```
<html><head>
<title>Mail</title>
</head><body>
<h1>Mail options</h1>
<form action="GoMailServlet">
<ol>
<li><input type="radio" name="mail"
value="http://www.gmail.com">gMail
<li><input type="radio" name="mail"
value="http://www.rediffmail.com">RediffMail
<li><input type="radio" name="mail"
value="http://www.yahoo.com">Yahoo Mail
<li><input type="radio" name="mail"
value="http://www.hotmail.com">Hot Mail
<li><input type="radio" name="mail"
value="unknown">Unknown
</ol>
<input type=submit value="go">
</form></body></html>
```

```
@WebServlet("/GoMailServlet")
public class GoMailServlet extends HttpServlet {

protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
   String mail=request.getParameter("mail");
   if (mail.equals("unknown") )
   response.sendError(response.SC_NOT_FOUND,"At this
   point this facility is not available");
   else response.sendRedirect(mail);

}
}
```

Inter- servlet communication

- Many times it becomes useful for a servlet to partly process the request and then to pass on the request to another servlet for further processing.
- Two servlets (or JSPs) can communicate with each other using a **`RequestDispatcher`** object.
- The servlet container creates the **`RequestDispatcher`** object,.
- This object can be obtained from **`ServletContext`** using methods:

  1. **`RequestDispatcher getRequestDispatcher(java.lang.String path)`**
  2. **`RequestDispatcher getNamedDispatcher(java.lang.String name)`**

  The first method is provided by **`Servlet Request`** also.

  These methods return a **`RequestDispatcher`** object that acts as a wrapper for the resource located at the given path (url pattern) or servlet name.

Different `getRequestDispatcher()`

- **`RequestDispatcher getRequestDispatcher(java.lang.String path)`**
- The path must be relative to the root of the application when this method when used using **`ServletContext`** .
- Relative path with respect to the servlet can be used with when this method when used using **`Servlet Request`** .

Tell me what

- **`RequestDispatcher getNamedDispatcher(java.lang.String name)`**

  This method uses servlet name to get **`RequestDispatcher ?`** What is servlet name? Is the servlet name same as servlet class name?

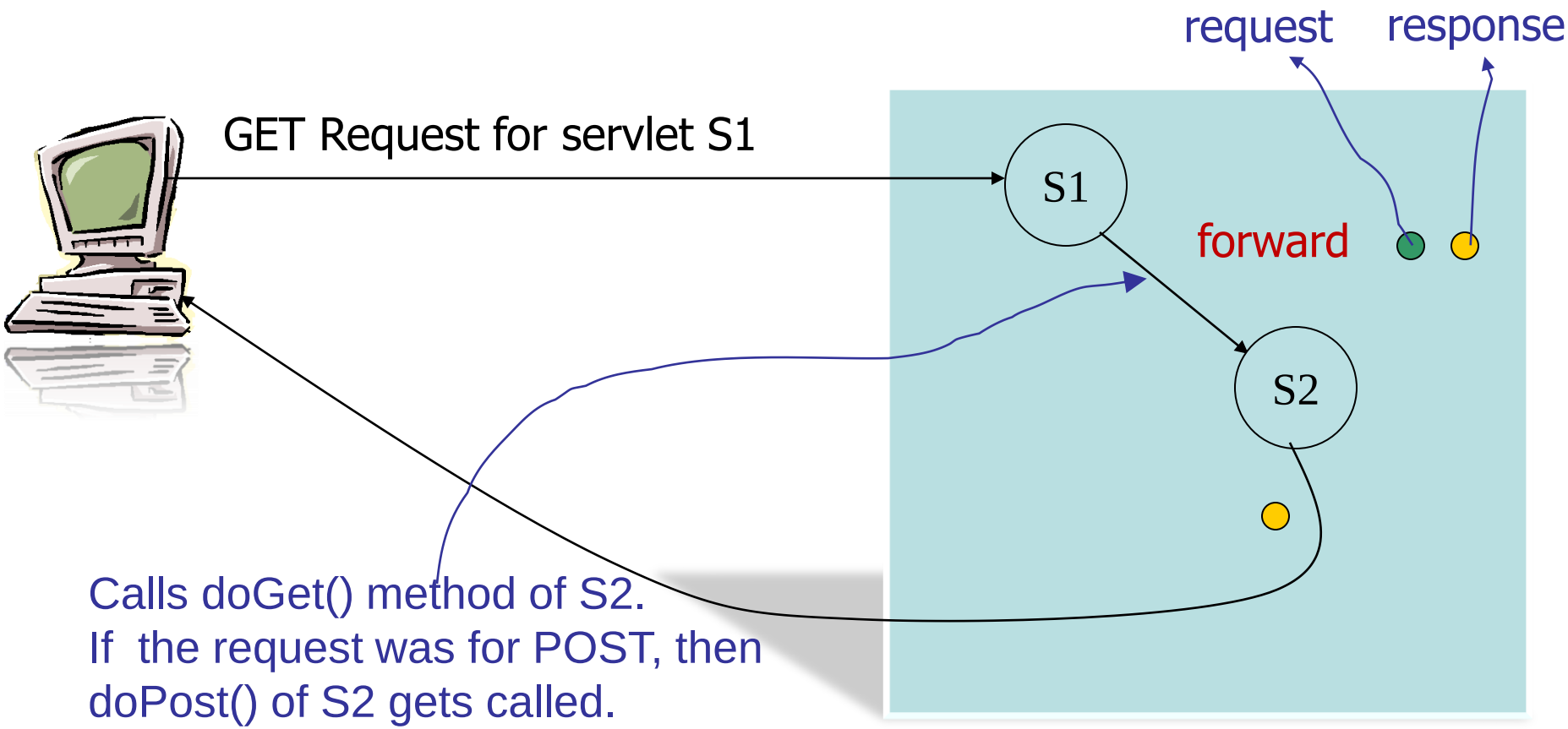
- Servlet name is not same as servlet class name. This name is usually used for configuration information by the server and is provided to decouple the class name from name that can be used for configuration.
- Servlet name can be provided in two ways :
  - Through annotation:
    - `@WebServlet(name="MySer",urlPatterns= {"/my"})`
  - Though **`web.xml`**:

```
<web-app>
 <servlet>
    <servlet-name>Register</servlet-name>
    <servlet-class>Register</servlet-class>
 </servlet>        …</web-app>
```

javax.servlet.RequestDispatcher

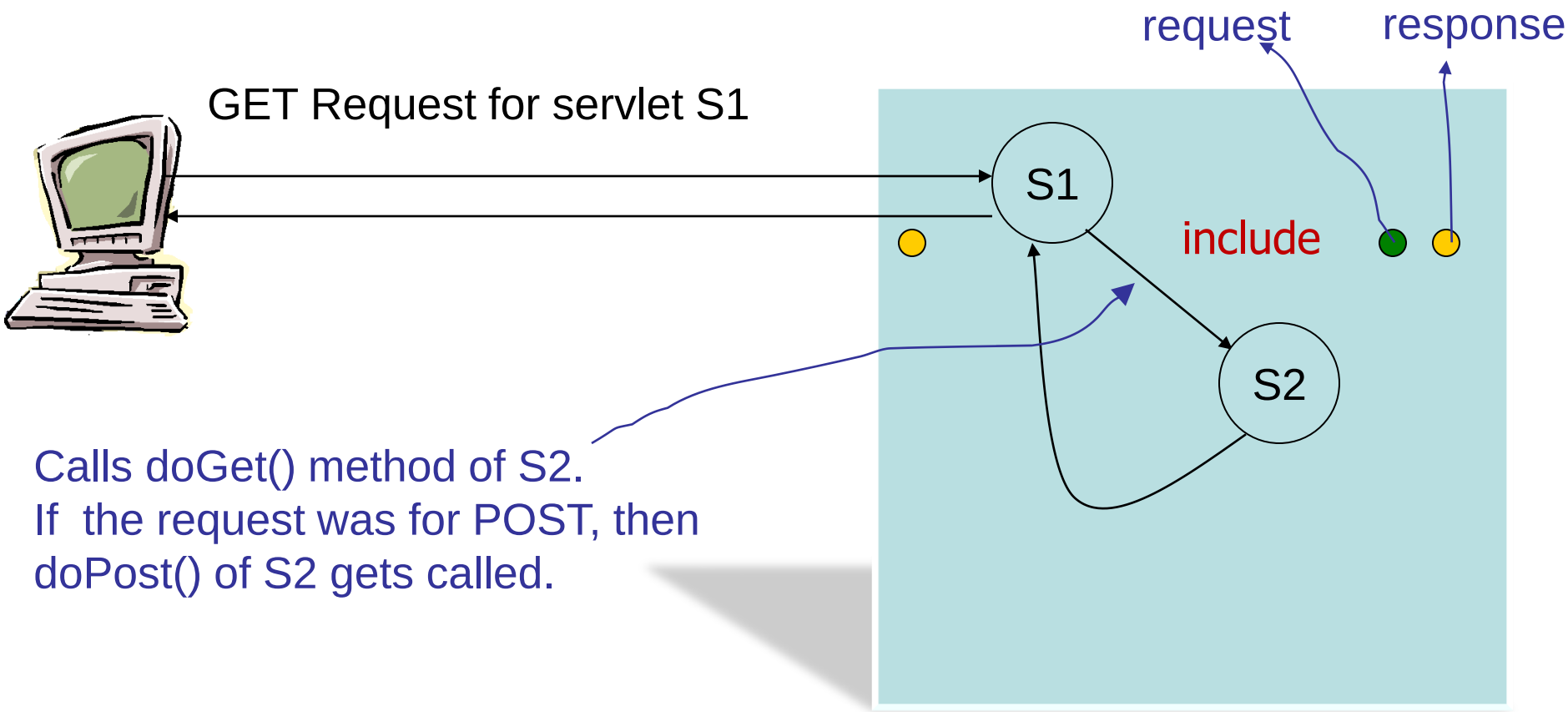
- This interface receives requests from the client and sends them to any resource (such as a servlet, HTML file, or JSP file) on the server.
- There are two ways in which the communicate can happen based on where the control is finally going to be.
- These are the only 2 methods that this interface declares. Note that the 2 request and response objects are also passed.
  - forward:
    - controls goes to the forwarded servlet
    - **`void forward(ServletRequest request, ServletResponse response)`**
  - include:
    - control comes back to the original servlet
    - **`void include(ServletRequest request, ServletResponse response)`**

forward



Control finally is in S2. Response is generated by S2

include



Control finally is in S1. Response is generated by S1

Test your understanding

- Servlets are java classes. A java class method can call another java class method. Then why do you need to **`RequestDispatcher`**?

Request Attributes

- Request and Response objects are shared between the servlets when they communicate. Needless to say that form parameters etc. will be accessible by both the servlets.
- In addition to this, request object can be used to send additional data to the a servlet or a JSP while forwarding (or including).
- **`HttpSessionRequest`** methods that allow this:
  - **`Object getAttribute(String name)`**
  - **`void setAttribute(String name , Object obj)`**
  - **`void removeAttribute(String name)`**

Example: using forward and include

- Scenario:
  1. A HTML form is displayed where user enters some data.
  2. On submitting this form, a servlet is invoked that checks for the validity of data.
  3. If the data is invalid, the same form is displayed with an error message → using include
  4. Otherwise another servlet is invoked that displays a friendly message→ using forward

1. HTML form

```
<html><head><title>Register</title></head>
<body>
<h2>Register</h2>
<table border=1><tr><td>
<form method=post action="Register">
First Name:<input type=text name="fname">
Last Name:<input type=text name="lname">
<input type=submit></form></td></tr> </table>
</body>
</html>
```

2. Register Servlet

```
// assume imports
public class Register extends HttpServlet {
@WebServlet("/Register")
..
protected void doPost(HttpServletRequest request,
  HttpServletResponse response) throws
  ServletException, IOException {
  String fname=request.getParameter("fname");
  String lname=request.getParameter("lname");
  if( (fname!=null && fname.trim().length()!=0) &&
  (lname!=null && lname.trim().length()!=0))
  request.getRequestDispatcher("Success").forward(req
  uest,response);
```

```
else{
request.setAttribute("error",“first name or last name
not entered");
getServletContext().getNamedDispatcher("error").include
(request,response);
request.getRequestDispatcher("index.html").include(requ
est,response);
}
}
```

3. Success Servlet

```
//assume imports
@WebServlet("/Success")
public class Success extends HttpServlet {
  protected void doPost(HttpServletRequest request,
  HttpServletResponse response) throws
     ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html><head><title>Success</title>");
    out.println("<head><body>");
    out.println("Thanks <I>
      "+request.getParameter("fname")+"</I>" );
    out.println("</body></html>");}
…}
```

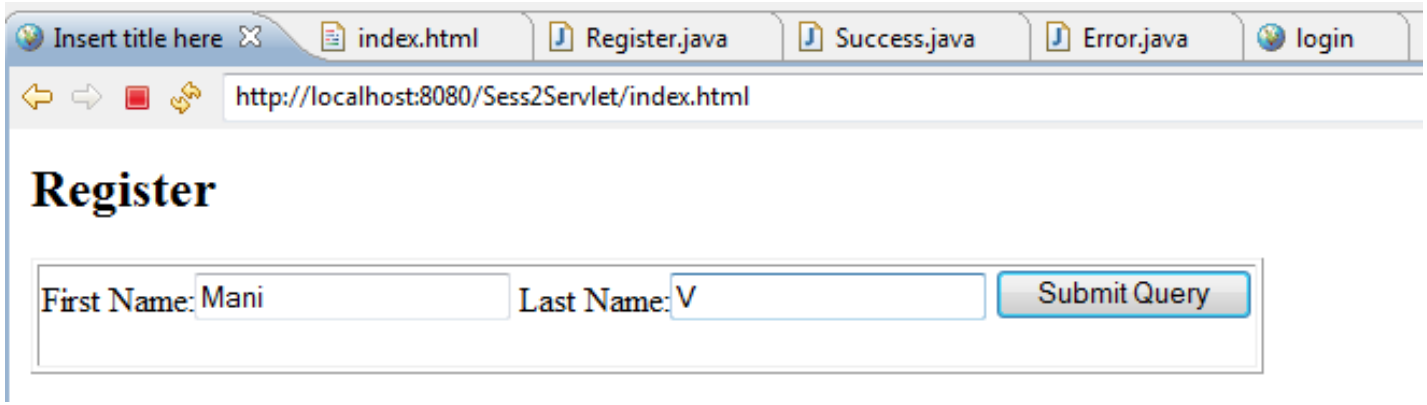
4. Error Servlet

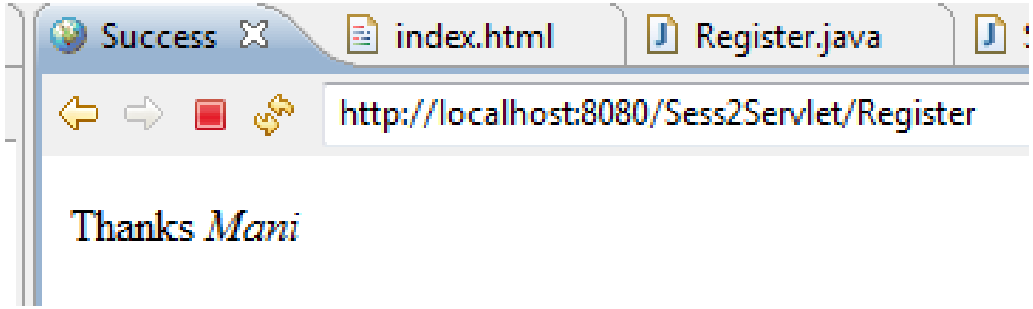
```
//assume imports
@WebServlet(name = "error",  urlPatterns ="/Error")
public class Error extends HttpServlet {
…
protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
PrintWriter out = response.getWriter();
response.setContentType("text/html");
out.println("<html><head><title>login </title>");
out.println("<head><body><font color=red><b>");

String err=(String)request.getAttribute("error");
if(err!=null)
out.println(err);
out.println("</font><b><br></body></html>");
}
…}
```

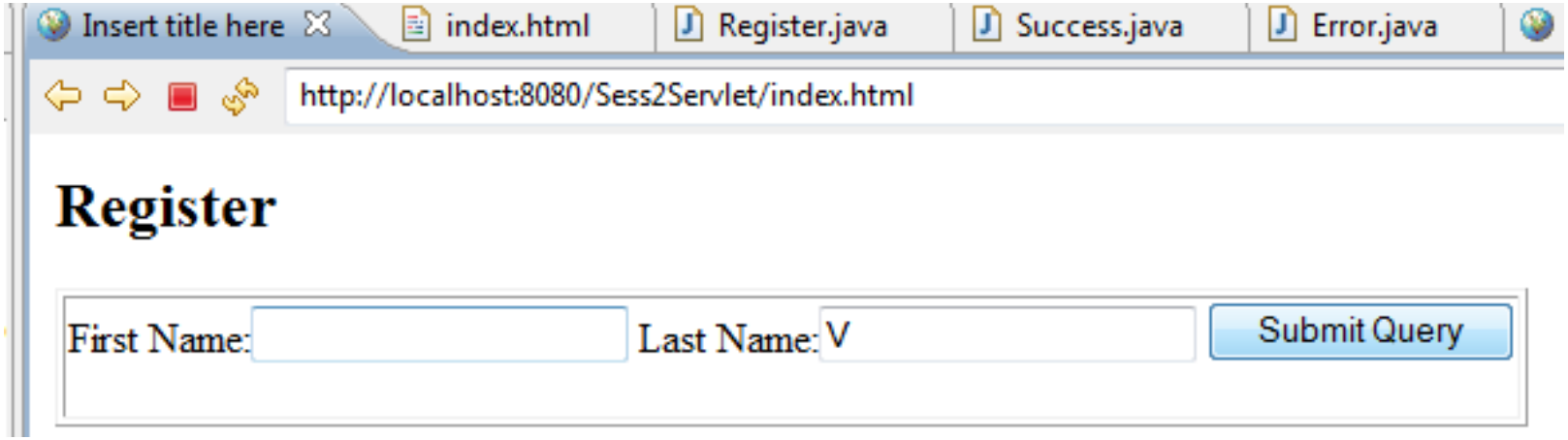
```
String err=(String)request.getAttribute("error");
      String login=request.getParameter("login");
      if(err!=null)
      out.println(err);
      out.println("</font><b><br></body></html>");
}catch(Exception e){out.println(e.toString());
}
}
}
```

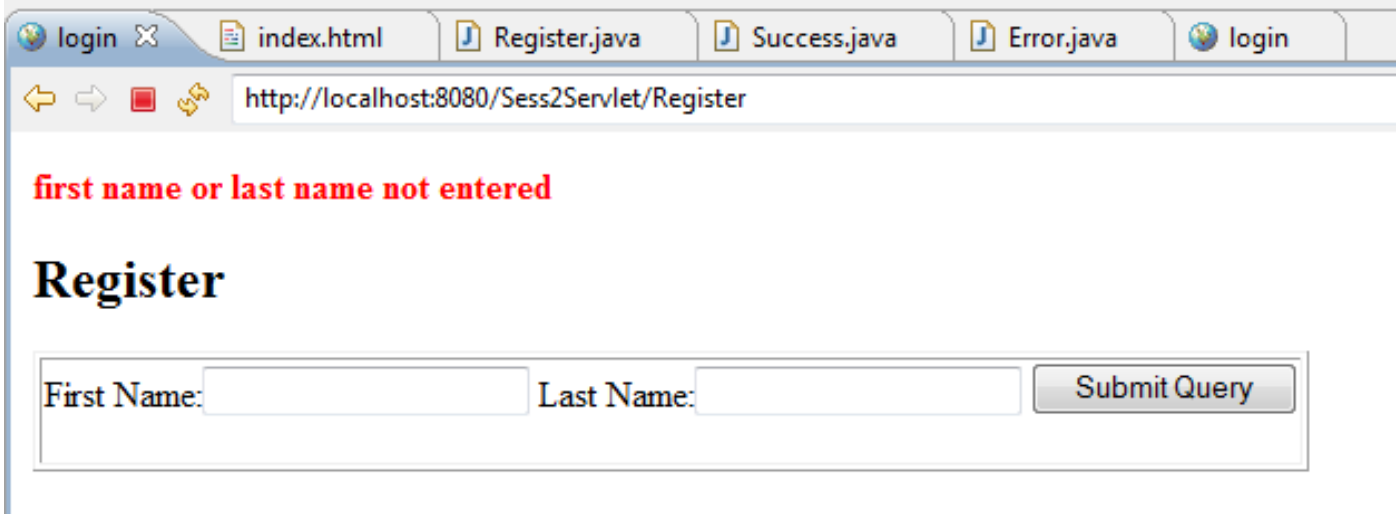
Executing: scenario 1





Executing: scenario 2





Tell me how

- It would be nice if the form page retains the data already entered? How can we achieve this?

- This can be easily done using by using a JSP page instead of HTML page.
- We will do this in JSP session.

More on `include` method

- The **`include()`** may be called at any time.
- Though the request object passed to the included/target servlet can be used just in the same way as the original servlet, there are some restrictions on the usage of response object.
- The included/target servlet can only write information to the ServletOutputStream or Writer of the response object and commit a response by writing content.
- It cannot set headers or call any method that affects the headers of the response (except for setting session parameters, which we will see later).
- Any attempt to set headers will be ignored.
- If the response is committed, then setting session parameters will throw **`IllegalStateException`**.

More on `forward` method

- The **`forward()`** can be called by any servlet only when no output has been committed to the client.
- If the response has been committed, an **`IllegalStateException`** is thrown.
- If output data exists in the response buffer that has not been committed then the content must be flushed using **`flushBuffer()`** of **`ServletResponse`** before the target servlet is called.
- The path elements of the request object has target servelt's path incase the **`RequestDispatcher`** is obtained using URL.
- In case **`RequestDispatcher`** is obtained using **`getNamedDispatcher`**, then path elements of the request object has path of original request.

Query Strings

- Query Strings can also be specified with the path that is `getRequestDispatcher(java.lang.String path).`
- `String path = "/error?code=5";`

  `RequestDispatcher rd = getServletContext().getRequestDispatcher(path);`

  `rd.include(request, response);`
- This parameter can be obtained using

  `request.getParameter("error");`

Tell me what

- What if there is clash between names of form parameter and query strings? That is if in the previous example

```
String path = "/error?code=5&fname=bob";
```

- Parameters specified in the query string used to create the RequestDispatcher take precedence over other parameters of the same name passed to the included servlet.
- Also the scope of the parameters associated with a RequestDispatcher are only for the duration of the include or forward call.