Servlets: Introduction to Servlets

Simon Xu

Contents

- Servlet Overview
- First Servlet Program
- Servlet Life Cycle
- Browser/Servlet Communication
- Servlet Session
- Other advanced Topics

5/10/02

Contents

- Servlet Overview
- First Servlet Program
- Servlet Life Cycle
- Browser/Servlet Communication
- Servlet Session
- Other advanced Topics

5/10/02

What is a Servlet?

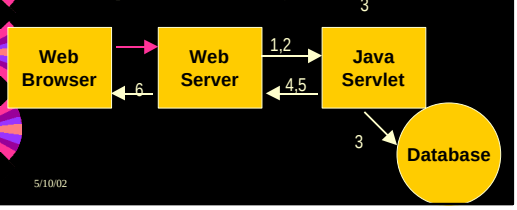
- Java's answer to the Common Gateway Interface (CGI).
- Applet: a java program that runs within the web browser.
- Servlet: a java program that runs within the web server.
  - Rapidly becoming the standard for building web applications.

5/10/02

Life of a Servlet

Regardless of the application, servlets usually carry out the following routine:

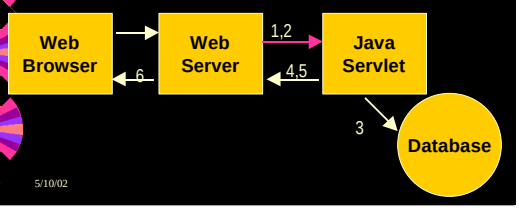
1. Read any data sent by the user
   - Capture data submitted by an HTML form.



5/10/02

Life of a Servlet

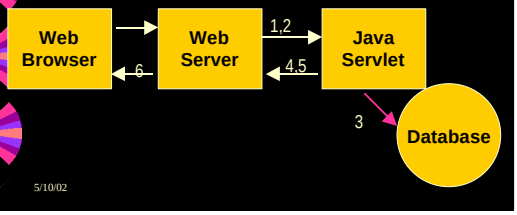
2. Look up any HTTP information
   - Determine the browser version, host name of client, cookies, etc.



5/10/02

Life of a Servlet

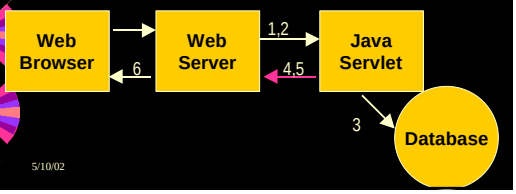
3. Generate the Results
   - Connect to databases, connect to legacy applications,



5/10/02

Life of a Servlet

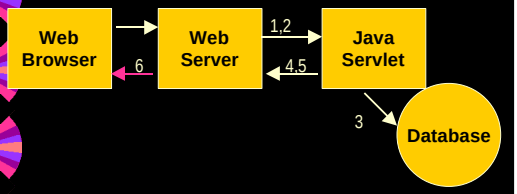
4. Format the Results
   - Generate HTML on the fly 3.
5. Set the Appropriate HTTP headers
   - Tell the browser the type of document being returned or set any cookies.



5/10/02

Life of a Servlet

6. Send the document back to client



5/10/02

What can we build with Servlets?

- Search Engines
- Personalization Systems
- E-Commerce Applications
- Shopping Carts
- Product Catalogs
- Intranet Applications

5/10/02

Server Side Options

- There are many options for creating server side applications.
  - We will examine CGI briefly only.
- This better enables us to understand servlets within the broader context of web development.
- Also enables us to better understand the advantages and disadvantages of servlets.

5/10/02

Server Side Options

- Common Gateway Interface (CGI)
- Fast CGI
- Mod Perl
- Server Extensions
  - NSAPI
  - ISAPI
- ASP
- PHP
- Cold Fusion

5/10/02

Common Features

- All server side frameworks share a common set of features:
  - Read data submitted by the user
  - Generate HTML dynamically based on user input
  - Determine information about the client browser
  - Access Database systems
  - Exploit the HTTP protocol

5/10/02

Decision Points

- When evaluating which server side framework to use, you need to consider a number of critical factors:
  - Ease of development:
    - How easily can you build new applications?
  - Performance:
    - How fast can the framework respond to queries?
  - Scalability:
    - Can the framework scale to thousands, millions of users?
  - Security:
    - Are there any inherent security vulnerabilities?

5/10/02

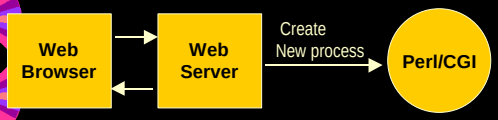
CGI

- Represents one of the earliest, practical methods for generating web content.
- Primarily written in the Perl programming language.
- Unfortunately, traditional CGI programs suffer from scalability and performance problems.
- Let's examine these two problems…

5/10/02

CGI Architecture

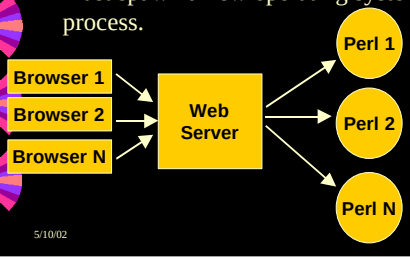
- Browser initiates request
- Web server receives the request.
- For each request, web server spawns a new operating system process to execute the CGI/Perl Program.



5/10/02

CGI Architecture

- For each browser request, the web server must spawn a new operating system process.



5/10/02

CGI Architecture

- Creating a new operating system process for each request takes time and memory.
- Hence, traditional CGI programs have inherent performance and scalability problems.
- Every other server architecture tries to address these problems.

5/10/02

Advantages of Servlets

- Servlets have six main advantages:
  - Efficient
  - Convenient
  - Powerful
  - Portable
  - Secure
  - Inexpensive

5/10/02

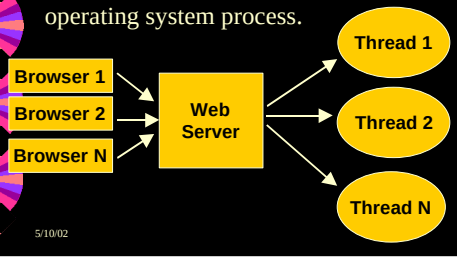
Advantage 1: Efficient

- For each browser request, the servlet spawns a light weight thread.
- This is faster and more efficient that spawning a new operating system process.
- Hence, servlets have better performance and better scalability than traditional CGI.

5/10/02

Servlets Architecture

- For each browser request, the web server only spawn a new thread, not a new operating system process.



5/10/02

Advantage 2: Convenient

- Servlets include built-in functionality for:
  - Reading HTML form data
  - Handling cookies
  - Tracking user sessions
  - Setting HTTP headers
- Java is object oriented

5/10/02

Advantage 3: Powerful

- Servlets can talk directly to the web servers.
- Multiple servlets can share data:
  - Particularly important for maintaining database connections.
- Includes powerful techniques for tracking user sessions.

5/10/02

Advantage 4: Portable

- One of the advantages of Java is its portability across different operating systems.
- Servlets have the same advantages.
- You can therefore write your servlets on Windows, then deploy them on UNIX.
- You can also run any of your servlets on any Java-enabled web server, with no code changes.

5/10/02

Advantage 5: Secure

- Traditional CGI programs have a number of known security vulnerabilities.
- Hence, you usually need to include a separate Perl/CGI module to supply the necessary security protection.
- Java has a number of built-in security layers.
- Hence, servlets are considered more secure than traditional CGI programs.

5/10/02

Advantage 6: Inexpensive

- You can download free servlet kits for development use.
- You can therefore get started for free!
- Nonetheless, production strength servlet web servers can get quite expensive.

5/10/02

Review of Servlet Introduction

- Servlets: a java program that runs within the web server.
- The simple life of Servlets
- The applications using servlets
- Advantages of Servlets

5/10/02

Contents

- Servlet Overview
- First Servlet Program
- Servlet Life Cycle
- Browser/Servlet Communication
- Servlet Session
- Other advanced Topics

5/10/02

Servlet Template

- First, let's take a look at a generic servlet template.
- All your future templates will follow this general structure.
- The most important pieces are noted in **Red**

5/10/02

```
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 {

    // Use "request" to read incoming HTTP headers
    // (e.g. cookies) and HTML form data (e.g. data the user
    // entered and submitted).

    // Use "response" to specify the HTTP response status
    // code and headers (e.g. the content type, cookies).

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

5/10/02

Generic Template

- Import the Servlet API:

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

- To create servlets, you must remember to always use these two import statements.

5/10/02

```
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 {

    // Use "request" to read incoming HTTP headers
    // (e.g. cookies) and HTML form data (e.g. data the user
    // entered and submitted).

    // Use "response" to specify the HTTP response status
    // code and headers (e.g. the content type, cookies).

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

5/10/02

Generic Template

- All your servlets must extend HTTPServlet.
- HTTPServlet represents the base class for creating Servlets within the Servlet API.
- Once you have extended HTTPServlet, you must override one or both:
  - doGet: to capture HTTP Get Requests
  - doPost: to capture HTTP Post Requests

5/10/02

```
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 {

    // Use "request" to read incoming HTTP headers
    // (e.g. cookies) and HTML form data (e.g. data the user
    // entered and submitted).

    // Use "response" to specify the HTTP response status
    // code and headers (e.g. the content type, cookies).

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

5/10/02

doGet doPost

- doGet and methods each take two parameters:
  HTTPServletRequest: encapsulates all information
  - Form data, client host name, HTTP request headers.

  HTTPServletResponse: encapsulate all information servlet response.
  HTTP Return status, outgoing cookies, HTML response.
- servlet to handle both GET doGet call or vice versa.

5/10/02

```
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 {

    // Use "request" to read incoming HTTP headers
    // (e.g. cookies) and HTML form data (e.g. data the user
    // entered and submitted).

    // Use "response" to specify the HTTP response status
    // code and headers (e.g. the content type, cookies).

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

5/10/02

Getting an OutputStream

- The HTTPResponse object has a getWriter() method.
- This method returns a java.io.PrintWriter object for writing data out to the Web Browser.

```
PrintWriter out = response.getWriter();
```

5/10/02

Hello World!

- We are finally ready to see our first real servlet.
- This servlet outputs “Hello World!” as plain text, not HTML.
- Let’s take a look at the code, and then see the servlet in action.

5/10/02

```
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");
  }
}
```

5/10/02

Output Stream

- Once you have an OutputStream object, you just call the println() method to output to the browser.
- Anything you print will display directly within the web browser.
- As we will now see, you can also output any HTML tags.

5/10/02

Generating HTML

- To generate HTML you need to add two steps:
  - Tell the browser that you are sending back HTML.
  - Modify the println() statements to return valid HTML.

5/10/02

HelloWorld.java

```
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 {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<HTML>\n" +
                  "<HEAD><TITLE>Hello World</TITLE></HEAD>\n" +
                  "<BODY>\n" +
                  "<H1>Hello World</H1>\n" +
                  "</BODY></HTML>");
  }
}
```

5/10/02

Generating HTML

- To return HTML, you must set the content MIME type to text/html:
  - `response.setContentType("text/html");`
- Remember that you must set the content type *before* you output any content.
- Once you have set the MIME type, you can return any HTML document you want.

5/10/02

How to compile your program?

- Hence, you will need to do development on your own machines.
- This is fairly straightforward, but it takes some time.

5/10/02

The Software

- To develop servlets, you will need three pieces of software:
  - Text Pad: a simple, text editor.
  - Java 2 Software Development Kit (JDK), Version 1.3
  - Java Servlet Development Kit (JSDK)
    - Contains the Servlet Runner for running Servlets on your own machine.
- You may use other replacement

5/10/02

Servlet Runner

- 1. Starting Servlet Runner
  - Open an MS-DOS Window
  - Go to the JSDK2.1 root directory: CD c:\jsdk2.1
  - Run the startserver command: startserver
    By default, Servlet Runner will run on Port 8080.
  - Open your web browser and go to: http://localhost:8080/
  - In your browser, you should see an index page of Sample servlets. Click any one of the "Execute" links to run the servlet.
- 2. Stopping Servlet Runner
  - To stop Servlet Runner, run the stopserver command: stopserver

5/10/02

- Once you have verified that Servlet Runner is able

  compile and run your own servlets.
  - servlet.jar file
- Generate your .class file

  % javac HelloWorld java
- Helloworld.class file to servlets directory

type: http://localhost servlet/

5/10/02

Review of First Servlet Program

```
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 {
    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
    out.println("<HTML>\n" +
                "<HEAD><TITLE>Hello World</TITLE></HEAD>\n" +
                "<BODY>\n" +
                "<H1>Hello World</H1>\n" +
                "</BODY></HTML>");
      }
}
```

5/10/02

Contents

- Servlet Overview
- First Servlet Program
- Servlet Life Cycle
- Servlet Communication
- Servlet Session
- Other advanced Topics

5/10/02

Life of a Servlet

- Birth: Create and initialize the servlet
  - Important method: init()
- Life: Handle 0 or more client requests
  - Important method: service()
- Death: Destroy the servlet
  - Important method: destroy()

5/10/02

Birth: The init() method

- The init() method is called when the servlet is first requested by a browser request.
- It is not called again for each request.
  - Used for one-time initialization.
  - No concurrency issues during servlet initialization
- There are two versions of the init() method:
  - Version 1: takes no arguments
  - Version 2: takes a **servletConfig** object as an argument.

5/10/02

Property Files

- To understand the difference between the two init() options, you need to first understand property files.
- All Web Servers/Servlet Runners maintain a central properties file for storing constants.
- You can add your own properties here. For example:
  - Database settings, user names, passwords, URLs, etc.

5/10/02

Servlet Runner

- Our Servlet Runner maintains a properties file at:
  C:\jsdk2.1\examples\WEB-INF>servlets.properties
- All initialization parameters go here.
- Remember that the location and format of the properties file is different for each web server.
- The next slide shows a sample servlets.properties file…

5/10/02

servlet.properties file

```
# $Id: servlets.properties,v 1.2 $

# Define servlets here

# <servletname>.code=<servletclass>
# <servletname>.initparams=<name=value>,<name=value>

snoop.code=SnoopServlet
snoop.initparams=initarg1=foo,initarg2=bar
```

Example Initialization Parameters

5/10/02

servlet.properties Rules

- To add your own properties, you need to follow the servlet.properties rules.
- You first need to register your servlet within the property file:
  <servletname>.code=<servletclass>

5/10/02

servlet.properties Rules

- You can then add your own properties:
  <servletname>.initparams=<name=value>,<name=value>
- For example, the following registers the Birth servlet, and sets its password parameter to "bluemoon":

```
Birth.code=Birth
Birth.initparams=password=bluemoon
```

5/10/02

Version 1: init() method

- No parameters
- Used when the servlet does not need to read any property files.
- Here's an example:

```
public void init() throws ServletException {
…
}
```

5/10/02

Version 2: init() method

- Used when the servlet needs to read from a
- Here's an example:

```
public void init (ServletConfig config)
  throws ServletException {
  super.init (config);
  …
 }
```

5/10/02

ServletConfig Object

- Provides access to the servlet properties file.
- Has a getInitParameter() method for retrieving specific properties.
- For example:

```
String message = config.getParameter ("message");
String password = config.getInitParameter ("password");
```

5/10/02

Version 2: init() method Cont.

- Let's examine version 2 again:

```
public void init (ServletConfig config)
  throws ServletException {
  super.init (config);
  …
 }
```

- It is important to call super.init().
- The init() method of the superclass registers the ServletConfig object so you can access it later.
- Therefore, if you do not call super.init(), you will never have access to the ServletConfig object.

5/10/02

Example

- Let's examine a simple example.
- This example uses the 2nd init() option.
- In this case, we have hard coded one parameter, and read one parameter from the properties file.
- Once initialized, this program echos out its initialization parameters.

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

public class Birth extends HttpServlet {
  String projectCodeName;
  String password;
  //  init() is called first
  public void init(ServletConfig config) throws ServletException {
          super.init (config);
          projectCodeName = new String ("Xerces");
          password = config.getInitParameter ("password");
  }
  //  Handle an HTTP GET Request
  public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws IOException, ServletException {
          response.setContentType("text/plain");
          PrintWriter out = response.getWriter();
          out.println ("Project Code Name:  "+projectCodeName);
          out.println ("Password:  "+password);
          out.close();
  }
}
```

Password is retrieved from the properties file.

5/10/02

serlvet.properties

```
# $Id: servlets.properties,v 1.2 2000/04/02 02:04:01 duncan Exp $

# Define servlets here

# <servletname>.code=<servletclass>
# <servletname>.initparams=<name=value>,<name=value>

#snoop.code=SnoopServlet
#snoop.initparams=initarg1=foo,initarg2=bar
cookie.code=CookieExample
cookie.initparams=foo
jsp.code=com.sun.jsp.runtime.JspServlet

Birth.code=Birth
Birth.initparams=password=bluemoon
```

5/10/02

Life of a Servlet---init()

- The first time a servlet is called, the Servlet is instantiated, and its init() method is called.
- Only one instance of the servlet is instantiated.
- This one instance handles all browser requests.

5/10/02

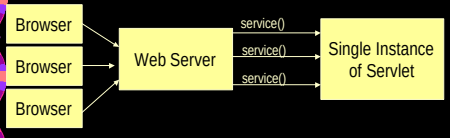
Life of a Servlet

- Birth: Create and initialize the servlet
  - Important method: init()
- Life: Handle 0 or more client requests
  - Important method: service()
- Death: Destroy the servlet
  - Important method: destroy()

5/10/02

Service() Method

- Each time the server receives a request for a servlet, the server spawns a new thread and calls the servlet's service () method.



5/10/02

Let's Prove it…

- To prove that only one instance of a servlet is created, let's create a simple example.
- The Counter Servlet keeps track of the number of times it has been accessed.
- This example maintains a single instance variable, called count.
- Each time the servlet is called, the count variable is incremented.
- If the Server created a new instance of the Servlet for each request, count would always be 0!

5/10/02

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

public class Counter extends HttpServlet {
        //  Create an instance variable
        int count = 0;

        //  Handle an HTTP GET Request
        public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws IOException, ServletException   {
                response.setContentType("text/plain");
                PrintWriter out = response.getWriter();
                count++;
                out.println ("Since loading, this servlet has "
                        + "been accessed "+ count + " times.");
                out.close();
        }
}
```

Only one instance of the counter Servlet is created. Each browser request is therefore incrementing the same count variable.

5/10/02

The Service Method

- By default the service() method checks the HTTP Header.
- Based on the header, service calls either doPost() or doGet().
- doPost and doGet is where you put the majority of your code.
- If your servlets needs to handle both get and post identically, have your doPost() method call doGet() or vice versa.

5/10/02

Thread Synchronization

- By default, multiple threads are accessing the same servlet at the same time.
- You therefore need to be careful to synchronize access to shared data.
  - For example, what happens if two browsers request a stock trade for the same account at the same time.
- Synchronization is, however a large topic in itself, and I will skip it here.
- Nonetheless, there is an option called the SingleThreadModel…

5/10/02

SingleThreadModel Interface

- To prevent multi-threaded access, you can have your servlet implement the SingleThreadModel:

```
public class FormServlet extends HttpServlet implements
    SingleThreadModel {
    ...
}
```

- This will guarantee that your servlet will only process one browser request at a time.
- It therefore addresses most synchronization issues.
- Unfortunately, this can result in severe slowing of performance.

5/10/02

Life of a Servlet

- Birth: Create and initialize the servlet
  - Important method: init()
- Life: Handle 0 or more client requests
  - Important method: service()
- Death: Destroy the servlet
  - Important method: destroy()

5/10/02

Death of a Servlet

- Before a server shuts down, it will call the servlet's destroy() method.
- You can handle any servlet clean up here. For example:
  - Updating log files.
  - Closing database connections.
  - Closing any socket connections.

5/10/02

Example: Death.java

- This next example illustrates the use of the destroy() method.
- While alive, the servlet will say "I am alive!".
- When the server is stopped, the destroy() method is called, and the servlet records its time of death in a "rip.txt" text file.

5/10/02

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

public class Death extends HttpServlet {

// Handle an HTTP GET Request
public void doGet(HttpServletRequest request, HttpServletResponse
        response)  throws IOException, ServletException
{
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        out.println ("I am alive!");
        out.close();
}
```

Continued….

5/10/02

```
// This method is called when one stops the Java Web Server

public void destroy() {
        try {
                FileWriter fileWriter = new FileWriter ("rip.txt");
                Date now = new Date();
                String rip = "I was destroyed at:  "+now.toString();
                fileWriter.write (rip);
                fileWriter.close();
        } catch (IOException e) {
                e.printStackTrace();
        }
}
```

5/10/02

Example rip.txt file

I was destroyed at: Thu Aug 24 11:10:58 CDT 2000

5/10/02

Putting it all together

A Persistent Counter

- Now that we know all about the birth, life and death of a servlet, let's put this knowledge together to create a persistent counter.
- The Counter.java example we covered earlier has a big problem:
  - When you restart the web server, counting starts all over at 0.
  - It does not retain any persistent memory.

5/10/02

Persistent Counter

- To create a persistent record, we can store the count value within a "counter.txt" file.
  - init(): Upon start-up, read in the current counter value from counter.txt.
  - destroy(): Upon destruction, write out the new counter value to counter.txt

5/10/02

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

public class CounterPersist extends HttpServlet {
  String fileName = "counter.txt";
  int count;

public void init () {
        try {
          FileReader fileReader = new FileReader (fileName);
          BufferedReader bufferedReader = new BufferedReader (fileReader);
          String initial = bufferedReader.readLine();
          count = Integer.parseInt (initial);
        } catch (FileNotFoundException e) { count = 0; }
        catch (IOException e) {  count = 0; }
        catch (NumberFormatException e) { count = 0; }
}
```

At Start-up, load the counter from file. In the event of any exception, initialize count to 0.

Continued….

5/10/02

```
// Handle an HTTP GET Request
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException  {
      response.setContentType("text/plain");
      PrintWriter out = response.getWriter();
      count++;
      out.println ("Since loading, this servlet has "
                  +"been accessed "+ count + " times.");
      out.close();
}
```

Each time the doGet() method is called, increment the count variable.

Continued….

5/10/02

```
// At Shutdown, store counter back to file
public void destroy() {
      try {
            FileWriter fileWriter = new FileWriter (fileName);
            String countStr = Integer.toString (count);
            fileWriter.write (countStr);
            fileWriter.close();
      } catch (IOException e) {
            e.printStackTrace();
      }
}
```

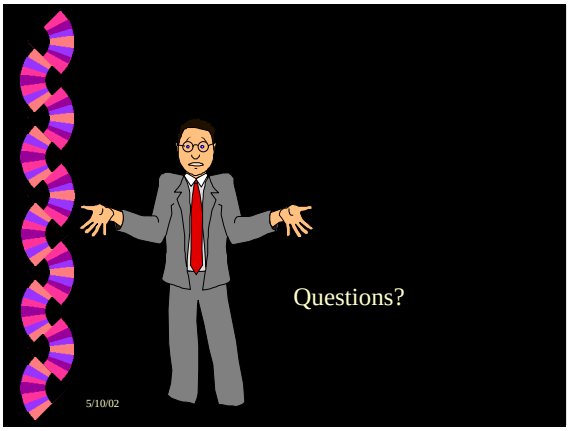
When destroy() is called, store new counter variable back to counter.txt.

5/10/02

Review of Servlet Lifecycle

- Loading servlets using class loader
- Birth: Create and initialize the servlet, by overriding init()
- Life: Handle 0 or more client requests by invocation of method service()
- Death: Destroy the servlet by calling destroy() method

5/10/02


Questions?
5/10/02