Midterm is NEXT WEEK!

• Nov 3
  • Starts at 7:05PM – ends at 9:15PM
    – I will hand out the tests at 7:05PM
    – The test will start at 7:15PM
  • Midterm will be next week instead of lecture. In this lecture hall.
  • NO LAPTOPS/no computers including PDAs! Just slides, notes, books and printouts.
Midterm is NEXT WEEK!

• Bring a pen or pencil.
• 50 - 60 questions.
• Mostly multiple choice a) - e).
• Covers up to and including this lecture.
• Questions from lecture, book (LJ) and homework ONLY.
Sample question format

Q: Object Orientation consists of the following basic characteristics:

a) Encapsulation, Polymorphism, Classes, Unique Object Identity
b) Incrementing, decrementing, looping, function calling, compiling
c) Exposing data and code publicly
d) Defining classes and operators
e) Creating functions and data structures
Homework #2

• HW#2 due by 5:00 PM EST TODAY.
• Grades on homework #1 will be posted to class website as soon as available
• HW#3 will be assigned tonight. Due Nov 10.
• Questions should be addressed to the TAs and me to speed answers and avoid delays.
HW #2

• Why a double and not a string?
  – Use the dataStream
  – avoid conversions to/from string from numeric and even complex objects
  – go to/from string means you need to do more error checking on your own like is the String “-0.0” a legal value as a double in java?
  – Inefficient and error prone converting to/from strings

• Why a RMI Inventory?
  – Inventory is good target for remoting. It has a lifecycle independent of the customer or PO. It could be started up and wait for clients to use it remotely.
  – RMI isn’t just the same design as a local object in many situations
  – Could have created a manager that controlled the lifecycle of a PO or created a remote factory that returned a PO to the client that then manipulated it.
Lecture 8

Java Reflection and Servlets
Chapter 7

• Read it again!
java.lang.reflect package

• allows looking at a class in runtime
• allows calling methods “discovered dynamically” in runtime
• allows walking up the class hierarchy
• allows dynamically instantiating them!
the Class class

• Represents all the classes and objects in the running application
Class methods

Class        forName(String className)
    //Returns the Class object associated with the
    //class or interface with the given string name.

static       Class forName(String name, boolean initialize,
    ClassLoader loader)
    //Returns the Class object associated with the
    //class or interface with the given string name,
    //using the given class loader.

Class[]       getClasses()
    //array class objs for the public classes and
    //interfaces that are members of the class
    //represented by this Class object.

ClassLoader   getClassLoader()
    //class loader for the class.

Class         getComponentType()
    //Returns the Class representing the component type
    //of an array.
Class methods (2)

Constructor getConstructor(Class[] parameterTypes)
//Returns a Constructor object that reflects the
//specified public constructor

Constructor[] getConstructors()
//Returns an array containing Constructor objects
//reflecting all the public constructors

Class[] getDeclaredClasses()
//Returns an array of Class objects reflecting all
//the classes and interfaces declared as members
//of the class represented by this Class object.

Constructor getDeclaredConstructor(
    Class[] parameterTypes)
//Any constructor – pub or private?

Constructor[] getDeclaredConstructors()
//all the constructors declared
Class methods (3)

Field    getDeclaredField(String name)
Field[]  getDeclaredFields()
Field    getField(String name)
          //Field object of a public member field
Field[]  getFields()
          //array of all the accessible public fields
Method   getDeclaredMethod(String name,
                           Class[] parameterTypes)
Method[] getDeclaredMethods()
Class methods (4)

Class      getDeclaringClass()
    //If the class or interface represented by this
    //Class object is a member of another class,
    //returns the Class object representing the class
    //in which it was declared.

Class[]    getInterfaces()

Method     getMethod(String name, Class[] parameterTypes)
    //public only

Method[]   getMethods()
    //including those declared by the class or interface and
    //and those inherited from supers

int        getModifiers()
    //Returns the Java language modifiers for this class or
    //interface, encoded in an integer.
<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td><code>getName()</code></td>
</tr>
<tr>
<td>Package</td>
<td><code>getPackage()</code></td>
</tr>
<tr>
<td>URL</td>
<td><code>getResource(String name)</code></td>
</tr>
<tr>
<td>InputStream</td>
<td><code>getResourceAsStream(String name)</code></td>
</tr>
<tr>
<td>Class</td>
<td><code>getSuperclass()</code></td>
</tr>
<tr>
<td>boolean</td>
<td><code>isArray()</code></td>
</tr>
<tr>
<td>boolean</td>
<td><code>isAssignableFrom(Class cls)</code></td>
</tr>
<tr>
<td>boolean</td>
<td><code>isInstance(Object obj)</code></td>
</tr>
<tr>
<td>boolean</td>
<td><code>isInterface()</code></td>
</tr>
<tr>
<td>boolean</td>
<td><code>isPrimitive()</code></td>
</tr>
<tr>
<td>Object</td>
<td><code>newInstance()</code></td>
</tr>
<tr>
<td>String</td>
<td><code>toString()</code></td>
</tr>
</tbody>
</table>

`getResource(String name)` returns an InputStream representing the resource with the given name. It is useful for loading resources from files or class loaders.

`getSuperclass()` returns the superclass of the class. This method is useful for understanding the inheritance hierarchy of a class.

`isArray()` returns `true` if the class is an array class.

`isAssignableFrom(Class cls)` returns `true` if the current class can be assigned to `cls` or if both the current class and `cls` are array classes.

`isInstance(Object obj)` returns `true` if `obj` is an instance of the current class.

`isInterface()` returns `true` if the class is an interface.

`isPrimitive()` returns `true` if the class is a primitive class.

`newInstance()` creates a new instance of the class. This method is useful for creating new objects of the class.

`toString()` returns the string representation of the object. It is useful for debugging and logging purposes.
RefelectionDemoGUI.java

- Shows using reflection to inspect different objects
Class name of the StringBuffer is = java.lang.StringBuffer
Class name = Lecture8.DemoObjectType1
Method => SampleClassMethod () returns => java.lang.String
Method => hashCode () returns => int
Method => wait () returns => void
Method => wait () returns => void
Method => wait () returns => void
Method => getClass () returns => java.lang.Class
Method => equals () returns => boolean
Method => notify () returns => void
Method => notifyAll () returns => void
Method => toString () returns => java.lang.String
Method => sample/instanceMethod () returns => java.lang.String
Field => instanceString1 of Type => java.lang.String
Field => instanceInt1 of Type => int
Field => classString1 of Type => java.lang.String
Field => classString2 of Type => java.lang.String
Constructor => Lecture8.DemoObjectType1
Constructor => Lecture8.DemoObjectType1

Class Lecture8.DemoObjectType1
is a child of Super Class java.lang.Object
newly create object.toString() = The class string 1 | The class string 2 | Hello | There | First | Second | Third | 100 | 200
Class name of the StringBuffer is java.lang.StringBuffer
Class name = Lecture8.DemoObjectType2
Method => hashCode () returns => int
Method => wait () returns => void
Method => wait () returns => void
Method => wait () returns => void
Method => getClass () returns => java.lang.Class
Method => equals () returns => boolean
Method => toString () returns => java.lang.String
Method => notify () returns => void
Method => notifyAll () returns => void
Constructor => Lecture8.DemoObjectType2

Class Lecture8.DemoObjectType2
is a child of Super Class java.lang.Object
newly create object.toString() = Lecture8.DemoObjectType2@30ce8f
Examine the method in the code

- public static String GetStringWithClassInfo(Object objectToReflect)

- Shows getting:
  - *methods*
  - *constructors*
  - *fields*
  - *parent class*

- Shows creating a new *instance* of the class via reflection instead of the *new* keyword
• Questions
• Comments

• 7.5
Servlets

an actual WWW technology!
run welcome servlet
Servlet

- A servlet is a java “program” that runs in a servlet engine and responds to requests from a webserver.
Servlets vs CGI

• CGI is a program that gets executed every time you request a page that is the CGI application
  – A process is started
  – The information on the request is passed into the process via STDIN
  – The program executes and returns response via STDOUT
  – The process terminates
• Can implement A CGI program in Java! (but why), C, C++, PERL, VB, Python, …
• All HTTP request types are passed in through the STDIN stream of characters
Servlets vs CGI (2)

- Java only.
- A servlet is a class that extends `HttpServlet` class
- Gets compiled and loaded into the servlet engine.
- Stays in memory
- Methods on the object get invoked for different types HTTP requests e.g. GET, POST, ...
- Reentrant!! The same code may be executed simultaneously by multiple threads serving multiple requesters.
  - Avoid class variables and even instance variables as they are shared across the threads because a single instance is in memory.
  - Use synchronized!
Servlets vs CGI

• CGI is a program that gets executed every time you request a page that is the CGI application
  – A process is started
  – The information on the request is passed into the process via STDIN
  – The program executes and returns response via STDOUT
  – The process terminates
Servlet & CGI

• Both return HTML to the web server that then returns the HTML to your browser!
HTTP Commands

- GET
- HEAD
- PUT
- POST
- DELETE
- OPTIONS
- TRACE
Servlet class methods

protected void doDelete(HttpServletRequest req, HttpServletResponse resp)
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
protected void doHead(HttpServletRequest req, HttpServletResponse resp)
protected void doOptions(HttpServletRequest req, HttpServletResponse resp)
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
protected void doPut(HttpServletRequest req, HttpServletResponse resp)
protected void doTrace(HttpServletRequest req, HttpServletResponse resp)
protected long getLastModified(HttpServletRequest req)

protected void service(HttpServletRequest req, HttpServletResponse resp)
    // Receives standard HTTP requests from the public service
    // method and dispatches them to the doXXX methods defined in
    // this class.

void service(ServletRequest req, ServletResponse res)
    // Dispatches client requests to the protected service method.
inherited from GenericServlet

void init(ServletConfig config)  //Called by the servlet container to
    //indicate to a servlet that the servlet is
    //being placed into service.

void init()   //A convenience method which can be overridden so that
    //there's no need to call super.init(config).

void destroy()  //Called by the servlet container to indicate to a servlet
    //that the servlet is being taken out of service.

ServletContext getServletContext()   //Returns a reference to the
    //ServletContext in which this servlet
    //is running.
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, java.io.IOException
{
    System.out.println("inside processRequest()");
    response.setContentType("text/html");
    java.io.PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("It is a great day to be a servlet isn't it?<body>");
    out.println("</body>");
    out.println("</html>");
    out.close();
}
Servlets in NetBeans
Choose Target

Specify the absolute path to the directory for the web module.

Directory: c:\Lectures\Lecture8\MyWebModule
out.println("<p>Invoke servlet 2 </p>");
out.println("<p>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&n
out.println("<p>Invoke servlet 2 </p>");
out.println("<p>&nbsp;&nbsp;</p>");
out.println("<p>&nbsp;&nbsp;</p>");
out.println("<p>&lt;Bold&gt;</p>");
out.println("<p>&lt;font size="7"&gt;LARGE&lt;/font&gt;</p>");
out.println("<p>&lt;font color="#FF0000"&gt;RED &lt;/font&gt;&lt;font color="#0000FF"&gt;BLUE&lt;/font&gt;&lt;font size="10000FF"&gt;BLUE</font>&lt;/font&gt;";
out.println("&lt;form method="POST" name="MyTestForm" action="http://192.168.0.4:8080/servlet/MyNewServlet"&gt;
	&lt;input type="text" name="YourNameFromTheForm" size="20"&gt;&lt;/p&gt;");
out.println("&lt;input type="submit" value="Submit" name="SubmitButton"&gt;&lt;/input type="&lt;/form"&gt;");
out.println("&lt;nbsp;&lt;/p&gt;");
out.println("&lt;/body&gt;");
New Wizard - Servlet

Steps

1. Choose Template
2. New Object Name

New Object Name

Name: WelcomeServlet
Add a line to the generated method processRequest()
then execute it via the little green triangle button

- Starts up the tomcat servlet engine built into NetBeans and HTTP server on TCP port 8080 (not standard HTTP 80)
- In a browser you should see:
• Questions
• Comments

• 8.0
“Fixing” the double quotes in a standard HTML page to work in a Java printStream

• You must escape a “ if you want it in the text and not to be seen as a delimiter by Java
• You do this by preceding it with a \ as in \”
• You can use notepad.exe search and replace to do this on a complete HTML source file for you.
• Just find ” and replace with \” through out the whole doc
• Then cut and paste into your servlet source code
In an HTML editor I created a form. Then copied the HTML into my servlet code.
I set certain form related values in FrontPage as so
Getting a servlet to be a HTML form

• Copied the code into my doPost() method of the servlet
• Created a bunch of out.println();
protected boolean createFirstPage(java.io.PrintWriter out) throws ServletException, java.io.IOException {
    out.println("<html>");
    out.println("<head>");
    out.println("<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">");
    out.println("<title>Customer Order Dialog</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<form method="POST" name="CustomerOrderDialog" action="http://localhost:8080/servlet/GVBooksOnlineServlet">");
    out.println("  <p>First Name&nbsp; <input type="text" name="FirstName" size="49"><br>");
    out.println("  Last Name&nbsp; <input type="text" name="LastName" size="49"><br>");
    out.println("  Address&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; <input type="text" name="Address" size="49"><br>");
    out.println("  City&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; <input type="text" name="City" size="20"><br>");
    out.println("  State&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; <select size="1" name="State">");
    out.println("  <option>AZ</option>");
    out.println("  <option>NJ</option>");
    out.println("  <option selected>NY</option>");
    out.println("  </select><br>");
    out.println("  Zipcode&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; <input type="text" name="Zipcode" size="15"><br>");
    out.println("  Phone&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; <input type="text" name="Phone" size="15"><br>");
    out.println("  Email&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; ");
Compiled the servlet and executed (loaded into the servlet engine) then referenced in a browser -
• Questions
• Comments

• 8.5
HTTP headers that are “visible” through the request object. 
request.getHeaders() returns an Enumeration of Strings

In doPost()
Header = Cache-Control
Header = Host
Header = Accept
Header = User-Agent
Header = Content-Length
Header = Accept-Language
Header = Accept-Encoding
Header = Content-Type
Header = Connection
Header = Referer
Get value of the headers
request.getHeader()

In doPost()
Header is  Cache-Control = no-cache
Header is  Host = 192.168.0.4:8080
Header is  Accept = image/gif, image/x-xbitmap, image/jpeg,
           image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*
Header is  User-Agent = Mozilla/4.0 (compatible; MSIE 6.0;
                 Windows NT 5.0; .NET CLR 1.0.3705)
Header is  Content-Length = 77
Header is  Accept-Language = en-us
Header is  Accept-Encoding = gzip, deflate
Header is  Content-Type = application/x-www-form-urlencoded
Header is  Connection = Keep-Alive
Header is  Referer =  
            http://lpoelman:8080/servlet/GVBooksOnlineServlet
What happens when you do a HTTP GET for a servlet?

- Browser sends a GET to the web server
- Web server passes the request to the servlet engine
- Servlet engine loads servlet if not loaded
- Spawns a thread (or gets one from the pool, actually)
- Invokes the `service()` method with a request and response object
What happens when you do a HTTP get for a servlet? (2)

• Browser sends a GET to the web server
• Web server passes the request to the servlet engine
• Servlet engine loads servlet if not loaded
• Spawns a thread (or gets one from the pool, actually)
• Invokes the service() method with a request and response object
Getting the form values from a POST

Parameter Load = Load
Parameter Phone = 212-222-2222
Parameter Zipcode = 11111
Parameter Email = poelman@cs.nyu.edu
Parameter Address = 43 Mercer
Parameter LastName = Poelman
Parameter City = Manhattan
Parameter FirstName = Logan
Parameter State = NY
See the GVBooksOnlineWithDB.pdf in the Lectures\Lecture7 directory

• print it out and look at it.
• this is a powerful template of a servlet that reads/writes data from a form to a file
Testing writing to a file

First Name: Logan
Last Name: Poelman
Address: 1000 Broadway
City: Manhattan
State: NY
Zipcode: 10006
Phone: 212-212-2222
Email: poelman@cs.nyu.edu
You pressed LOAD

Go to main page
Testing writing to a file

First Name: Logan  
Last Name: Poelman  
Address: 1000 Broadway  
City: Manhattan  
State: NY  
Zipcode: 10006  
Phone: 212-212-2222  
Email: poelmon@cs.nyu.edu

Load  Save  Create Order  Reset
Created a customer file on the server
Customer toString()
3 Logan Poelman 212-212-2222 1000 Broadway Manhattan NY 10006
poelman@cs.nyu.edu
Each POST sends the fields in the form to the HTTP server

- Then the servlet engine invokes the `doPost()` method
- Examining the parameters we can find the ones that are “ActionToDoBUTTON”
- Then get the parameter value and check for “Load”, “Save”, or “Create Order”
- Based on which one, create an HTML page and return that through the `outputStream` to the servlet
- That gets returned to the browser
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Customer Order Dialog</title>
</head>
<body>

Testing writing to a file<br>

<form method="POST" name="CustomerOrderDialog" action="http://localhost:8080/servlet/GVBooksOnlineWithDB">
<p>First Name<br>
<input type="text" name="FirstName" size="49"><br></p>
<p>Last Name<br>
<input type="text" name="LastName" size="49"><br>
</p>
<p>Address<br>
<input type="text" name="Address" size="49"><br>
</p>
<p>City<br>
<input type="text" name="City" size="20"><br>
</p>
<p>State<br>
<select size="1" name="State">
<option>AZ</option>
<option>NJ</option>
<option selected>NY</option>
<option>TX</option>
<option>WI</option>
</select><br>
</p>
<p>Zipcode<br>
<input type="text" name="Zipcode" size="15"><br>
</p>
<p>Phone<br>
<input type="text" name="Phone" size="15"><br>
</p>
<p>Email<br>
<input type="text" name="Email" size="49"><br>
</p>
<p>&nbsp;</p>
<p><input type="submit" value="Load" name="ActionToDoBUTTON">
<input type="submit" value="Save" name="ActionToDoBUTTON">
<input type="submit" value="Create Order" name="ActionToDoBUTTON">
<input type="reset" value="Reset" name="Reset"></p>
</form>
</body>
</html>
• Remember to refresh your page in the browser with changes!!
• Bug like symptoms
• Questions
• Comments

• 8.75
Summary

• Reflection allow you to examine the classes in the runtime
• See the methods, fields, constructors, etc.
• Servlets are a Java based web page that runs on the server
• POST, GET, …
• doPost(), doGet()
• Multithreaded and multiply reentrant
• Careful using class and instance variables, stack variable are safe
Resources

• http://www.apache.org/
• http://jakarta.apache.org/tomcat/index.html
• http://www.novocode.com/doc/servlet-essentials/
HW #3

• Create a Servlet that loads and saves a customer object to a file called customerfile.txt in the current working directory. You should be able to enter data into a web page and it gets saved to a file on the server.
• Use the code created to handle a customer.
• Create a Servlet that returns a HTML form initially. (GET).
HW#3 (2)

• Create one servlet or multiple servlets, your choice.
• Should look almost exactly like the screen shot in the homework page.
• Use the HTML in the homework directory to create the visual part of the servlet (HTML).
• Use the code you already wrote for HW#1 for the Customer class.