

## 1. Basic Servlet Structure

Here's the outline of a basic servlet that handles GET requests. GET requests, for those unfamiliar with HTTP, are requests made by browsers when the user types in a URL on the address line, follows a link from a Web page, or makes an HTML form that does not specify a METHOD. Servlets can also very easily handle POST requests, which are generated when someone creates an HTML form that specifies METHOD="POST". We'll discuss that in later sections.

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

public class SomeServlet 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 line and headers
        // (e.g. specifying the content type, setting cookies).

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

(Download [template source code](#) -- click with the right mouse on the link or hold down SHIFT while clicking on the link.)

To be a servlet, a class should extend [HttpServlet](#) and override `doGet` or `doPost` (or both), depending on whether the data is being sent by GET or byPOST. These methods take two arguments: an [HttpServletRequest](#) and an [HttpServletResponse](#). The `HttpServletRequest` has methods that let you find out about incoming information such as FORM data, HTTP request headers, and the like. The `HttpServletResponse` has methods that lets you specify the HTTP response line (200, 404, etc.), response headers (`Content-Type`, `Set-Cookie`, etc.), and, most importantly, lets you obtain a [PrintWriter](#) used to send output back to the client. For simple servlets, most of the effort is spent in `println` statements that generate the desired page. Note that `doGet` and `doPost` throw two exceptions, so you are required to include them in the declaration. Also note that you have to import classes in `java.io` (for `PrintWriter`, etc.), `javax.servlet` (for `HttpServlet`, etc.), and `javax.servlet.http` (for `HttpServletRequest` and `HttpServletResponse`). Finally, note that `doGet` and `doPost` are called by the `service` method, and sometimes you may want to override `service` directly, e.g. for a servlet that handles both GET and POST request.

## 2. A Simple Servlet Generating Plain Text

Here is a simple servlet that just generates plain text. The following section will show the more usual case where HTML is generated.

## 2.1 HelloWorld.java

---

You can also [download the source](#) or [try it on-line](#).

```
package hall;

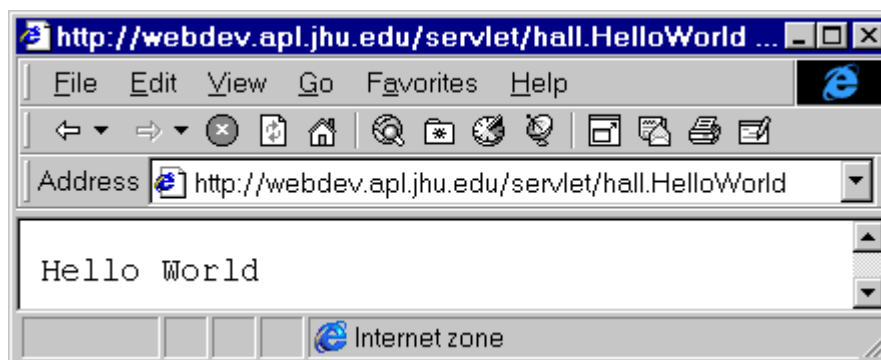
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

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

## 2.3 Running the Servlet

---

With the Java Web Server, servlets are placed in the `servlets` directory within the main JWS installation directory, and are invoked via `http://host/servlet/ServletName`. Note that the directory is `servlets`, plural, while the URL refers to `servlet`, singular. Since this example was placed in the `hall` package, it would be invoked via `http://host/servlet/hall.HelloWorld`. Other Web servers may have slightly different conventions on where to install servlets and how to invoke them. Most servers also let you define aliases for servlets, so that a servlet can be invoked via `http://host/any-path/any-file.html`. The process for doing this is completely server-specific; check your server's documentation for details.



## Servlets and HTML

Most servlets generate HTML, not plain text as in the previous example. To do that, you need two additional steps: tell the browser that you're sending back HTML, and modify the `println` statements to build a legal Web page. The first step is done by setting the `Content-Type` response header. In general, headers can be set via the `setHeader` method of `HttpServletResponse`, but setting the content type is such a common task that there is also a special `setContentType` method just for this purpose. Note that you need to set response headers *before* actually returning any of the content via the `PrintWriter`. Here's an example:

### 3.1 HelloWWW.java

---

You can also [download the source](#) or [try it on-line](#).

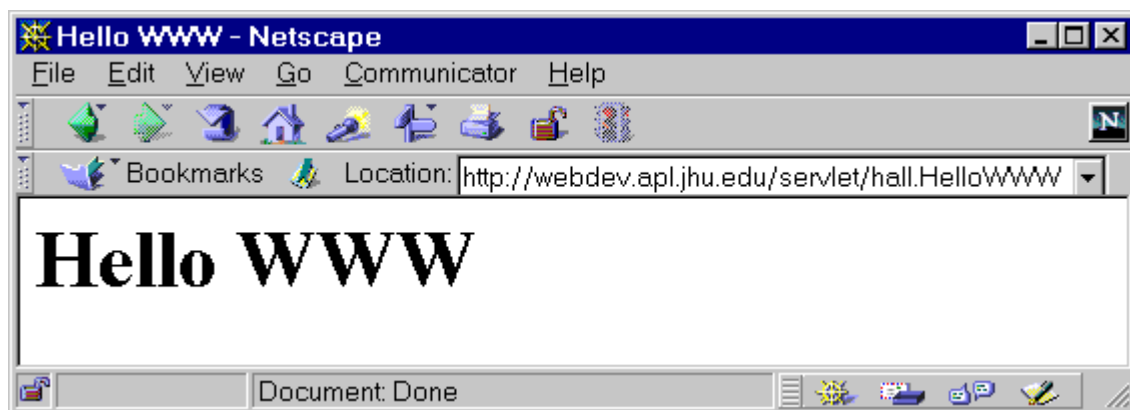
```
package hall;

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

public class HelloWWW extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 \" +
                    \"Transitional//EN\">\n" +
                    "<HTML>\n" +
                    "<HEAD><TITLE>Hello WWW</TITLE></HEAD>\n" +
                    "<BODY>\n" +
                    "<H1>Hello WWW</H1>\n" +
                    "</BODY></HTML>");
    }
}
```

### 3.2 HelloWWW Result

---



# Servlets & Form Data

## 1. Introduction

If you've ever used a Web search engine, visited an on-line bookstore, tracked stocks on-line, or asked a Web-based site for quotes on plane tickets, you've probably seen funny looking URLs like `http://host/path?user=Marty+Hall&origin=bwi&dest=lax`. The part after the question mark (i.e. `user=Marty+Hall&origin=bwi&dest=lax`) is known as *form data*, and is the most common way to get data from a Web page to a server-side program. It can be attached to the end of the URL after a question mark (as above), for GET requests, or sent to the server on a separate line, for POST requests.

Extracting the needed information from this form data is traditionally one of the most tedious parts of CGI programming. First of all, you have to read the data one way for GET requests (in traditional CGI, this is usually via the `QUERY_STRING` environment variable), and another way for POST requests (usually by reading the standard input). Second, you have to chop the pairs at the ampersands, then separate the parameter names (left of the equals signs) from the parameter values (right of the equals signs). Third, you have to URL-decode the values. Alphanumeric characters get sent unchanged, but spaces get converted to plus signs and other characters get converted to `%XX` where `XX` is the ASCII (or ISO Latin-1) value of the character, in hex. For example, if someone entered a value of "`~hall, ~gates, and ~mcnealy`" into a textfield with the name "users" in an HTML form, the data would get sent as "`users=%7Ehall%2C+%7Egates%2C+and+%7Emcnealy`". Finally, the fourth reason that parsing form data is tedious is that values can be omitted (e.g. `param1=val1&param2=&param3=val3`) and a parameter can have more than one value in that the same parameter can appear more than once (e.g. `param1=val1&param2=val2&param1=val3`).

One of the nice features of Java servlets is that all of this form parsing is handled automatically. You simply call the `getParameter` method of the `HttpServletRequest`, supplying the parameter name as an argument. Note that parameter names are case sensitive. You do this exactly the same way when the data is sent via GET as you do when it is sent via POST. The return value is a `String` corresponding to the udecoded value of the first occurrence of that parameter name. An empty `String` is returned if the parameter exists but has no value, and `null` is returned if there was no such parameter. If the parameter could potentially have more than one value, as in the example above, you should call `getParameterValues` instead of `getParameter`. This returns an array of strings. Finally, although in real applications your servlets probably have a specific set of parameter names they are looking for, for debugging purposes it is sometimes useful to get a full list. Use `getParameterNames` for this, which returns an `Enumeration`, each entry of which can be cast to a `String` and used in a `getParameter` call.

## 2. Example: Reading Three Parameters

Here's a simple example that reads parameters named `param1`, `param2`, and `param3`, listing their values in a bulleted list. Note that, although you are required to specify response settings (content type, status line, other HTTP headings) before beginning to generate the content, there is no requirement that you read the request parameters at any particular time.

Also note you can easily make servlets that can handle both `GET` and `POST` data, simply by having its `doPost` method call `doGet` or by overriding `service` (which calls `doGet`, `doPost`, `doHead`, etc.). This is good standard practice, since it requires very little extra work and permits flexibility on the part of the client. If you're used to the traditional CGI approach where you read `POST` data via the standard input, you should note that there is a similar way with servlets by first calling `getReader` or `getInputStream` on the `HttpServletRequest`. This is a bad idea for regular parameters, but might be of use for uploaded files or `POST` data being sent by custom clients rather than via `HTML` forms. Note, however, that if you read the `POST` data in that manner, it might no longer be found by `getParameter`.

### 2.1 ThreeParams.java

You can also [download the source](#) or [try it on-line](#). Note: also uses [ServletUtilities.java](#), shown earlier.

```
package hall;

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

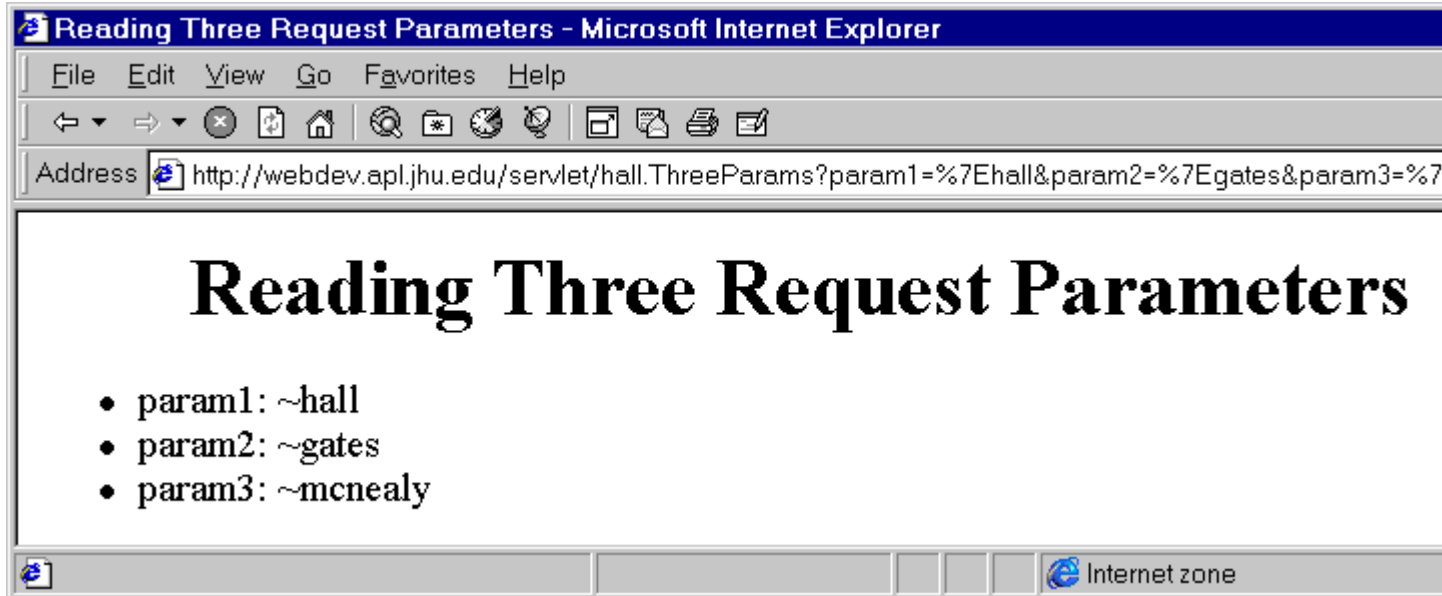
public class ThreeParams extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Reading Three Request Parameters";
        out.println(ServletUtilities.headWithTitle(title) +
            "<BODY>\n" +
            "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
            "<UL>\n" +
            "  <LI>param1: "
            + request.getParameter("param1") + "\n" +
            "  <LI>param2: "
            + request.getParameter("param2") + "\n" +
            "  <LI>param3: "
            + request.getParameter("param3") + "\n" +
            "</UL>\n" +
            "</BODY></HTML>");
    }
}
```

```

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

```

## 2.2 ThreeParams Output



## 3. Example: Listing All Form Data

Here's an example that looks up *all* the parameter names that were sent and puts them in a table. It highlights parameters that have zero values as well as ones that have multiple values. First, it looks up all the parameter names via the `getParameterNames` method of `HttpServletRequest`. This returns an `Enumeration`. Next, it loops down the `Enumeration` in the standard manner, using `hasMoreElements` to determine when to stop and `nextElement` to get each entry. Since `nextElement` returns an `Object`, it casts the result to a `String` and passes that to `getParameterValues`, yielding an array of `Strings`. If that array is one entry long and contains only an empty string, then the parameter had no values, and the servlet generates an italicized "No Value" entry. If the array is more than one entry long, then the parameter had multiple values, and they are displayed in a bulleted list. Otherwise the one main value is just placed into the table.

### 3.1 ShowParameters.java

You can also [download the source](#) or [try it on-line](#). Note: also uses [ServletUtilities.java](#), shown earlier.

```
package hall;
```

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

```
/** Shows all the parameters sent to the servlet via either
 * GET or POST. Specially marks parameters that have no values or
 * multiple values.
 *

```

```
 * Part of tutorial on servlets and JSP that appears at
 * http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/
 * 1999 Marty Hall; may be freely used or adapted.
 */
```

```
public class ShowParameters extends HttpServlet {

    public void doGet(HttpServletRequest request,

                       HttpServletResponse response)

        throws ServletException, IOException {

        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        String title = "Reading All Request Parameters";

        out.println(ServletUtilities.headWithTitle(title) +

                    "<BODY BGCOLOR=#FDF5E6>\n" +

                    "<H1 ALIGN=CENTER>" + title + "</H1>\n" +

                    "<TABLE BORDER=1 ALIGN=CENTER>\n" +
```

```

        "<TR BGCOLOR=\"#FFAD00\">\n" +
        "<TH>Parameter Name<TH>Parameter Value(s)");

Enumeration paramNames = request.getParameterNames();
while(paramNames.hasMoreElements()) {

    String paramName = (String)paramNames.nextElement();

    out.println("<TR><TD>" + paramName + "\n<TD>");

    String[] paramValues = request.getParameterValues(paramName);

    if (paramValues.length == 1) {

        String paramValue = paramValues[0];

        if (paramValue.length() == 0)

            out.print("<I>No Value</I>");

        else

            out.print(paramValue);

    } else {

        out.println("<UL>");

        for(int i=0; i<paramValues.length; i++) {

            out.println("<LI>" + paramValues[i]);

        }

        out.println("</UL>");

    }

}

out.println("</TABLE>\n</BODY></HTML>");
}

```

```

public void doPost(HttpServletRequest request,

```

```

        HttpServletResponse response)

    throws ServletException, IOException {

    doGet(request, response);

    }

}

```

## 3.2 Front End to ShowParameters

Here's an HTML form that sends a number of parameters to this servlet. Right click on the [source code link](#) to download the HTML. Left click on the link to try it out on-line. It uses POST to send the data (as should *all* forms that have PASSWORD entries), demonstrating the value of having servlets include both a `doGet` and a `doPost`. However, just for the sake of illustration, a [version using GET](#) can also be downloaded or tried out on-line.

### PostForm.html

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>A Sample FORM using POST</TITLE>
</HEAD>

<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">A Sample FORM using POST</H1>

<FORM ACTION="/servlet/hall.ShowParameters"
  METHOD="POST">
  Item Number:
  <INPUT TYPE="TEXT" NAME="itemNum"><BR>
  Quantity:
  <INPUT TYPE="TEXT" NAME="quantity"><BR>
  Price Each:
  <INPUT TYPE="TEXT" NAME="price" VALUE="$"><BR>
  <HR>
  First Name:
  <INPUT TYPE="TEXT" NAME="firstName"><BR>
  Last Name:
  <INPUT TYPE="TEXT" NAME="lastName"><BR>
  Middle Initial:
  <INPUT TYPE="TEXT" NAME="initial"><BR>
  Shipping Address:
  <TEXTAREA NAME="address" ROWS=3 COLS=40></TEXTAREA><BR>
  Credit Card:<BR>
  <INPUT TYPE="RADIO" NAME="cardType"
    VALUE="Visa">Visa<BR>
  <INPUT TYPE="RADIO" NAME="cardType"
    VALUE="Master Card">Master Card<BR>

```

```
<INPUT TYPE="RADIO" NAME="cardType"
      VALUE="Amex">American Express<BR>
<INPUT TYPE="RADIO" NAME="cardType"
      VALUE="Discover">Discover<BR>
<INPUT TYPE="RADIO" NAME="cardType"
      VALUE="Java SmartCard">Java SmartCard<BR>
Credit Card Number:
<INPUT TYPE="PASSWORD" NAME="cardNum"><BR>
Repeat Credit Card Number:
<INPUT TYPE="PASSWORD" NAME="cardNum"><BR><BR>
<CENTER>
  <INPUT TYPE="SUBMIT" VALUE="Submit Order">
</CENTER>
</FORM>

</BODY>
</HTML>
```

**A Sample FORM using POST** - Netscape

File Edit View Go Communicator Help

Bookmarks Location: <http://webdev.apl.jhu.edu/~hall/servlets/PostForm.html>

## A Sample FORM using POST

Item Number:

Quantity:

Price Each:

---

First Name:

Last Name:

Middle Initial:

Shipping Address:

Johns Hopkins Univ. Applied Physics Lab  
11100 Johns Hopkins Rd.  
Laurel, MD 20723

Credit Card:

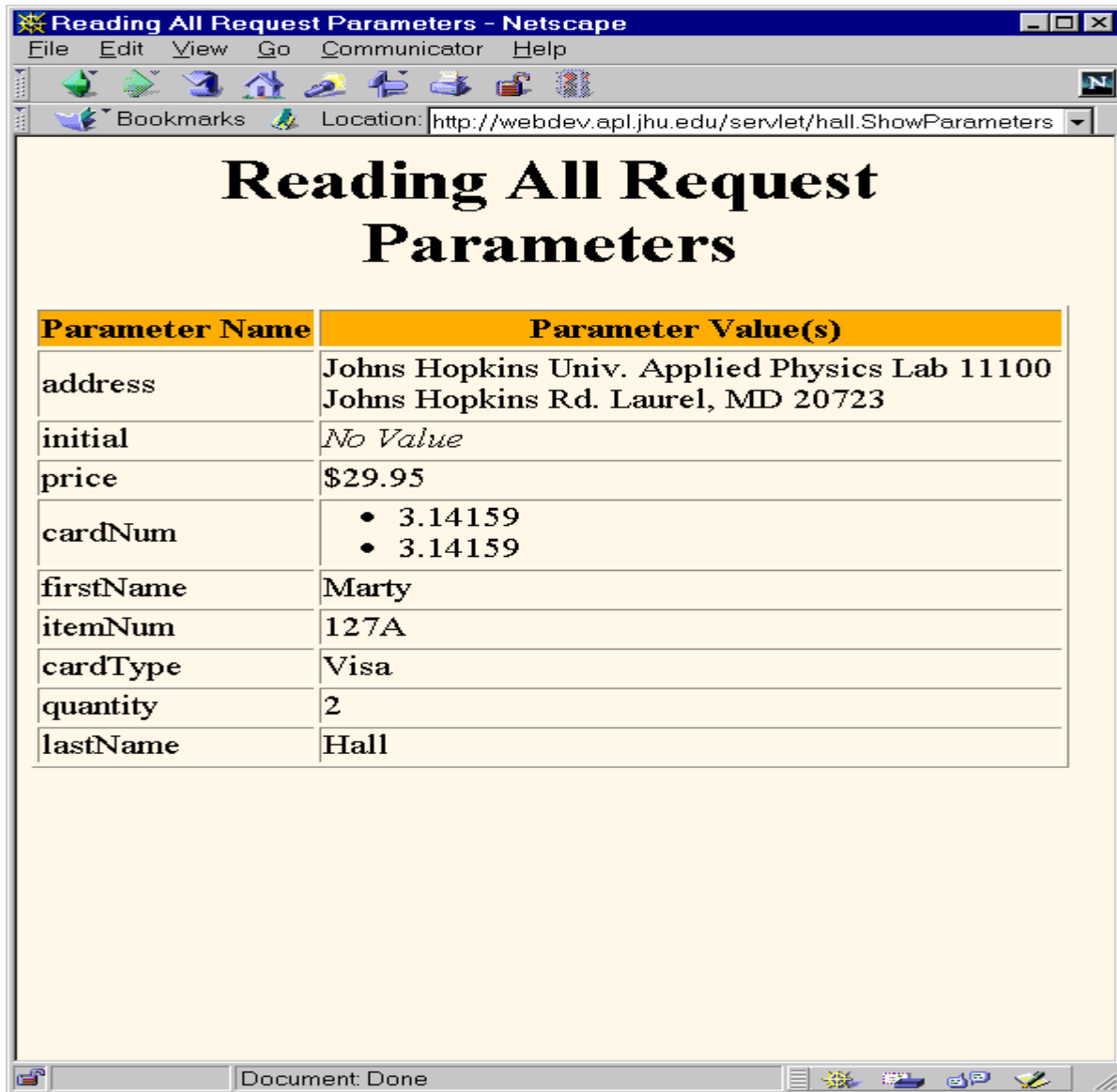
- Visa
- Master Card
- American Express
- Discover
- Java SmartCard

Credit Card Number:

Repeat Credit Card Number:

Document Done

### 3.3 Submission Result



The screenshot shows a Netscape browser window with the title "Reading All Request Parameters - Netscape". The address bar displays the URL "http://webdev.apl.jhu.edu/servlet/hall.ShowParameters". The main content area features a large heading "Reading All Request Parameters" and a table listing request parameters and their values.

Parameter Name	Parameter Value(s)
address	Johns Hopkins Univ. Applied Physics Lab 11100 Johns Hopkins Rd. Laurel, MD 20723
initial	<i>No Value</i>
price	\$29.95
cardNum	<ul style="list-style-type: none"><li>• 3.14159</li><li>• 3.14159</li></ul>
firstName	Marty
itemNum	127A
cardType	Visa
quantity	2
lastName	Hall