10.1 Overview of Ajax

- ***History***
  - **Possibility began with the nonstandard `iframe` element, which appeared in IE4 and Netscape 4**
    - **An `iframe` element could be made invisible and could be used to send asynchronous requests**
  - **Microsoft introduced `XmlDocument` and `XMLHTML` ActiveX objects in IE5 – for asynchronous requests**
  - **Two events ignited widespread interest in Ajax:**
    1. **The appearance of Google Maps and Google Mail**
    2. **Jesse James Garrett named the new technology Ajax**
  - **Goal of Ajax is to provide Web-based applications with responsiveness approaching that of desk-top applications**

10.1 Overview of Ajax (continued)

- Specific kind of Web applications that benefit from Ajax are those that have frequent interactions between the client and the server

- Goals are achieved with two different approaches:

1. Client requests are handled asynchronously

2. Only small parts of the current document are updated

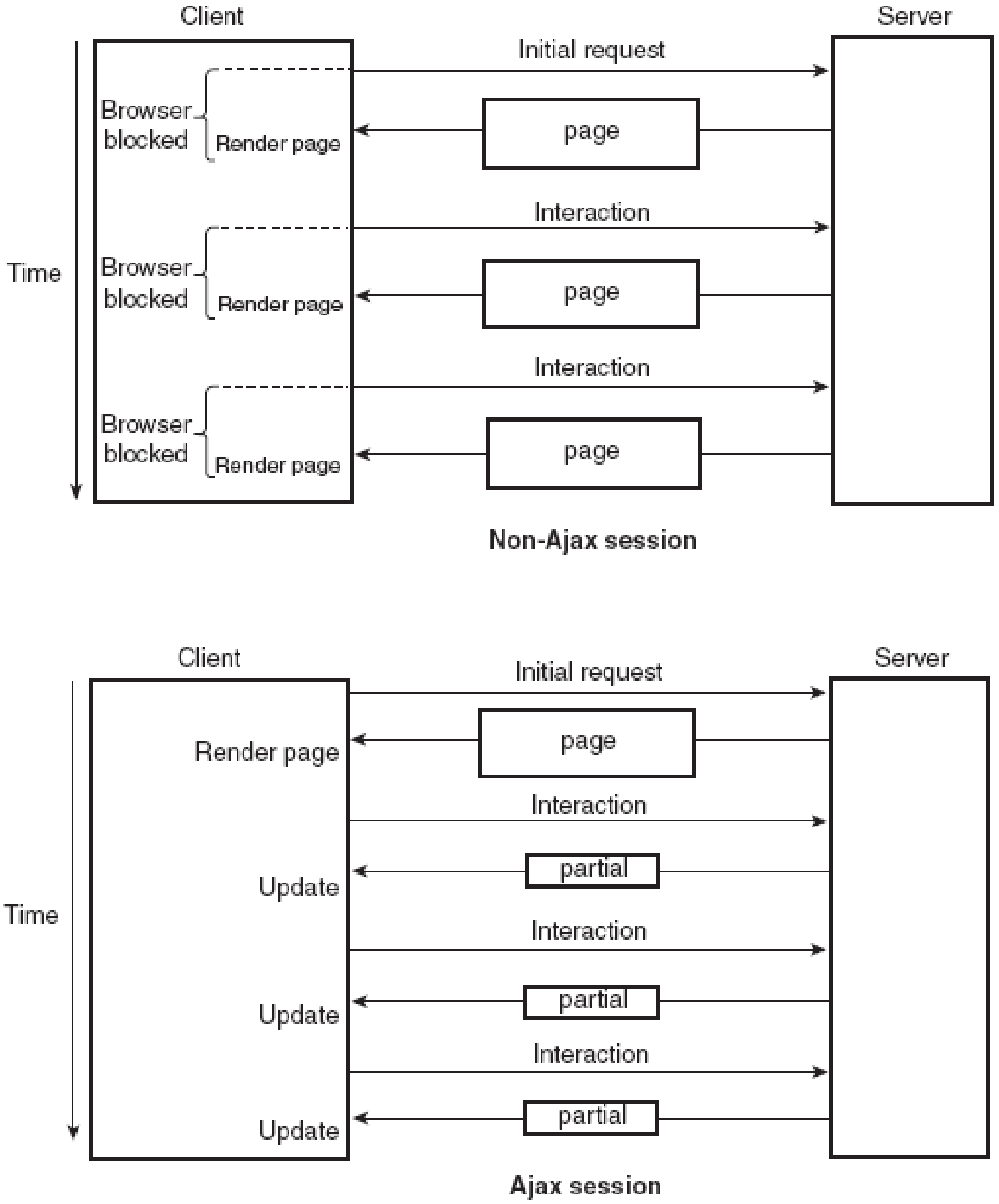


Figure 10.1 Traditional and Ajax browser–server interactions

- Ajax does not use any new programming languages or markup languages

    - Client side: JavaScript, XML, XHTML, DOM, CSS

    - Server side: any (PHP, servlets, ASP.NET, etc.)

- Rather than the original `XMLHTML` and `XmlDocument` objects, now the `XMLHttpRequest` object is used

- Toolkits are now often used to create Ajax applications, e.g., Prototype and Dojo
- Also, frameworks, such as ASP.NET, JavaServer Faces, and Rails

10.2 The Basics of Ajax

- **Described through a very simple application**

- ***The application*: Helps the user fill a form**

  - **The form gathers client information; asks for the zip code before the names of the city and state**

  - **As soon as the zip code is entered, the application sends a request to the server, which looks up the city and state for the given zip code and returns them to the form**

  - **Uses JavaScript to put the city and state names in the form**

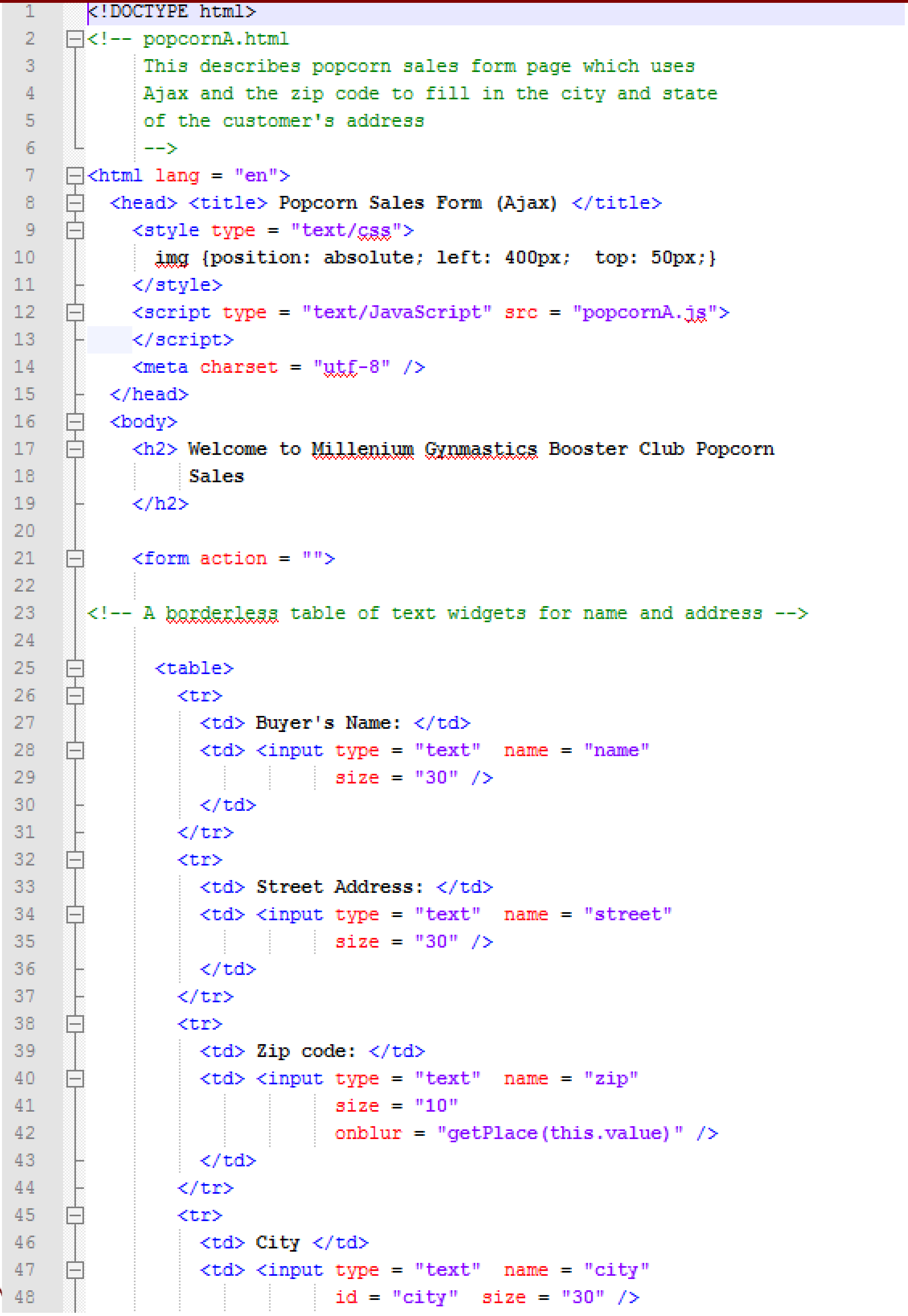
  - **Uses PHP on the server to look up the city and state**

- ***The form***

  - **Must reference the JavaScript code file in its head**

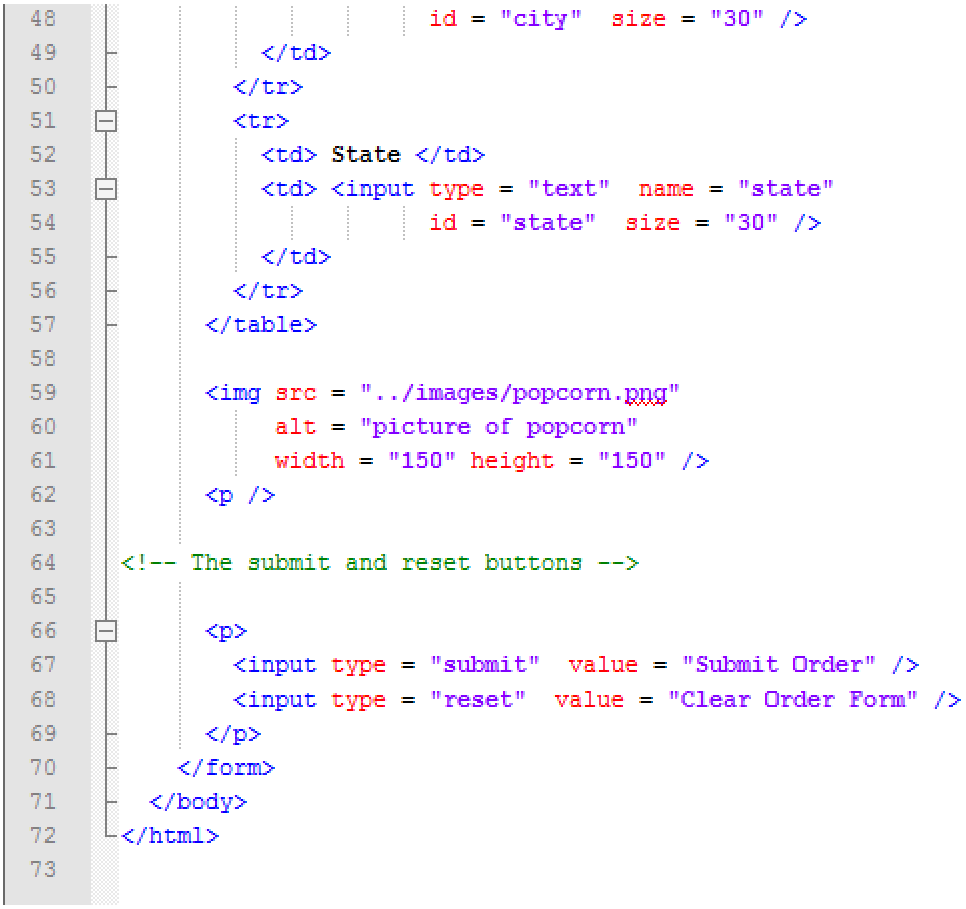
  - **Must register an event handler on the `blur` event of the zip code text box**

Example: popcornA.html

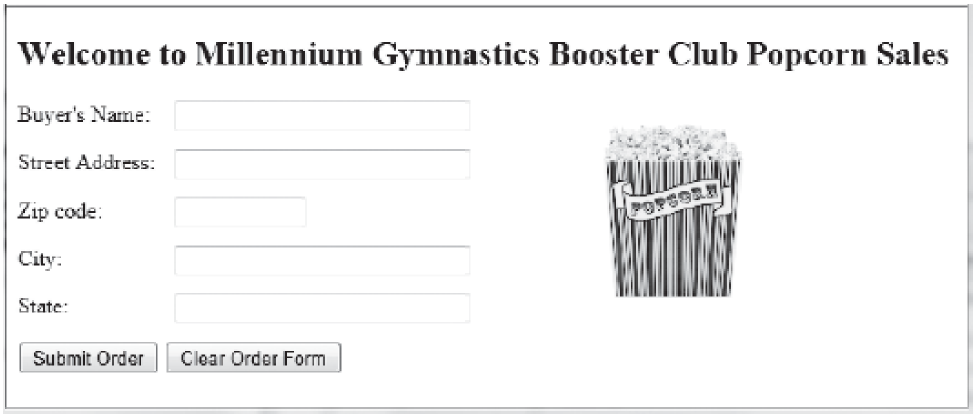


```
<!DOCTYPE html>
<!-- popcornA.html
     This describes popcorn sales form page which uses
     Ajax and the zip code to fill in the city and state
     of the customer's address
     -->
<html lang = "en">
  <head> <title> Popcorn Sales Form (Ajax) </title>
    <style type = "text/css">
      img {position: absolute; left: 400px;  top: 50px;}
    </style>
    <script type = "text/JavaScript" src = "popcornA.js">
    </script>
    <meta charset = "utf-8" />
  </head>
  <body>
    <h2> Welcome to Millenium Gymnastics Booster Club Popcorn
         Sales
    </h2>

    <form action = "">

<!-- A borderless table of text widgets for name and address -->

      <table>
        <tr>
          <td> Buyer's Name: </td>
          <td> <input type = "text"  name = "name"
                      size = "30" />
          </td>
        </tr>
        <tr>
          <td> Street Address: </td>
          <td> <input type = "text"  name = "street"
                      size = "30" />
          </td>
        </tr>
        <tr>
          <td> Zip code: </td>
          <td> <input type = "text"  name = "zip"
                      size = "10"
                      onblur = "getPlace(this.value)" />
          </td>
        </tr>
        <tr>
          <td> City </td>
          <td> <input type = "text"  name = "city"
                      id = "city"  size = "30" />
```

Example: popcornA.html



```
                      id = "city"  size = "30" />
          </td>
        </tr>
        <tr>
          <td> State </td>
          <td> <input type = "text"  name = "state"
                      id = "state"  size = "30" />
          </td>
        </tr>
      </table>

      <img src = "../images/popcorn.png"
           alt = "picture of popcorn"
           width = "150" height = "150" />
      <p />

<!-- The submit and reset buttons -->

      <p>
        <input type = "submit"  value = "Submit Order" />
        <input type = "reset"  value = "Clear Order Form" />
      </p>
    </form>
  </body>
</html>

```



Welcome to Millennium Gymnastics Booster Club Popcorn Sales

Buyer's Name: ______

Street Address: ______

Zip code: ______

City: ______

State: ______

Figure 10.2 A display of the `popcornA.html` document

10.2 The Basics of Ajax (continued)

- Two functions are required by the application:

  1. The `blur` handler

  2. A function to handle the response

-The Request Phase (The `blur` handler)

- The communication to the server for the asynchronous request must be made through the `XMLHttpRequest` object, so one must be created

```
var xhr = new XMLHttpRequest();
```

- When the server receives an asynchronous request, it sends a sequence of notices, called *callbacks*, to the browser (0, …, 4)

  - Only the last one is of interest, 4, which indicates that the response is complete

  - The response function is what is called in the callbacks

  - The response function must be registered on the `onreadystatechange` property of the XHR object

```
xhr.onreadystatechange = receivePlace;
```

10.2 The Basics of Ajax (continued)

- *The Request Phase* (continued)

- Next, the handler must call the `open` method of the XHR object

- Parameters to `open`:

1. HTTP method, `GET` or `POST`, quoted

2. The URL of the response document on the server

3. A Boolean literal to indicate whether the request is to be asynchronous (`true`) or synchronous (`false`)

- The parameter (the zip code) must be attached to the URL (because `GET` will be used)

```
xhr.open("GET",
     "getCityState.php?zip=" + zip, true);
```

(`getCityState.php` is the response document)

- The request is sent with the `send` method

```
xhr.send(null);
```

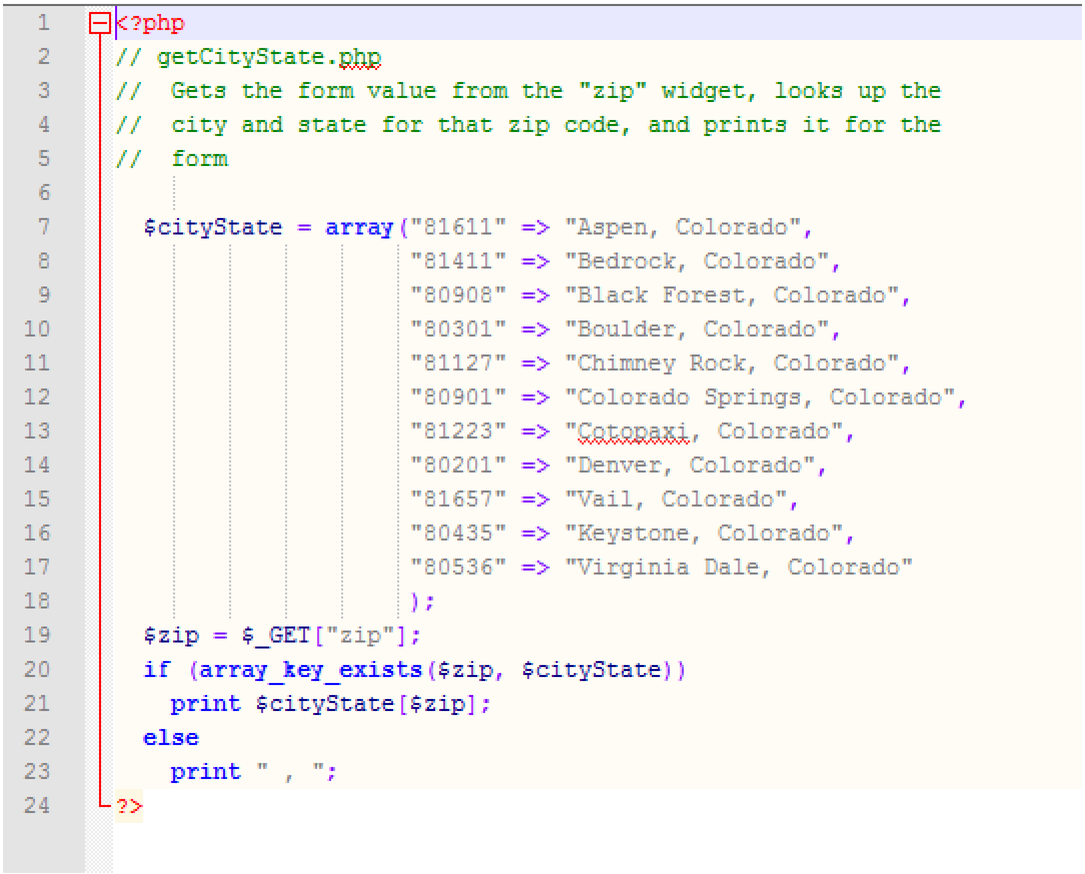
10.2 The Basics of Ajax (continued)

- *The Response Document*

- We'll use a simple hash of zip codes and names of cities and states, so this will be very simple

- The response data is produced with a `print` statement

→ `getCityState.php`



```php
<?php
// getCityState.php
//  Gets the form value from the "zip" widget, looks up the
//  city and state for that zip code, and prints it for the
//  form

  $cityState = array("81611" => "Aspen, Colorado",
                     "81411" => "Bedrock, Colorado",
                     "80908" => "Black Forest, Colorado",
                     "80301" => "Boulder, Colorado",
                     "81127" => "Chimney Rock, Colorado",
                     "80901" => "Colorado Springs, Colorado",
                     "81223" => "Cotopaxi, Colorado",
                     "80201" => "Denver, Colorado",
                     "81657" => "Vail, Colorado",
                     "80435" => "Keystone, Colorado",
                     "80536" => "Virginia Dale, Colorado"
                     );
  $zip = $_GET["zip"];
  if (array_key_exists($zip, $cityState))
    print $cityState[$zip];
  else
    print " , ";
?>
```

- *The Receiver Phase*

    - A JavaScript function with no parameters

        - Fetch the server response (text), split it into its two parts (city and state), and set the corresponding text boxes to those values

    - The receiver function must be able to access the XHR

        - If it is global, it would be accessible, but it could be corrupted by simultaneous requests and responses

        - The alternative is to register the actual code of the receiver, rather than its name

10.2 The Basics of Ajax (continued)

- *The Receiver Phase* (continued)

  - Actions of the receiver function:

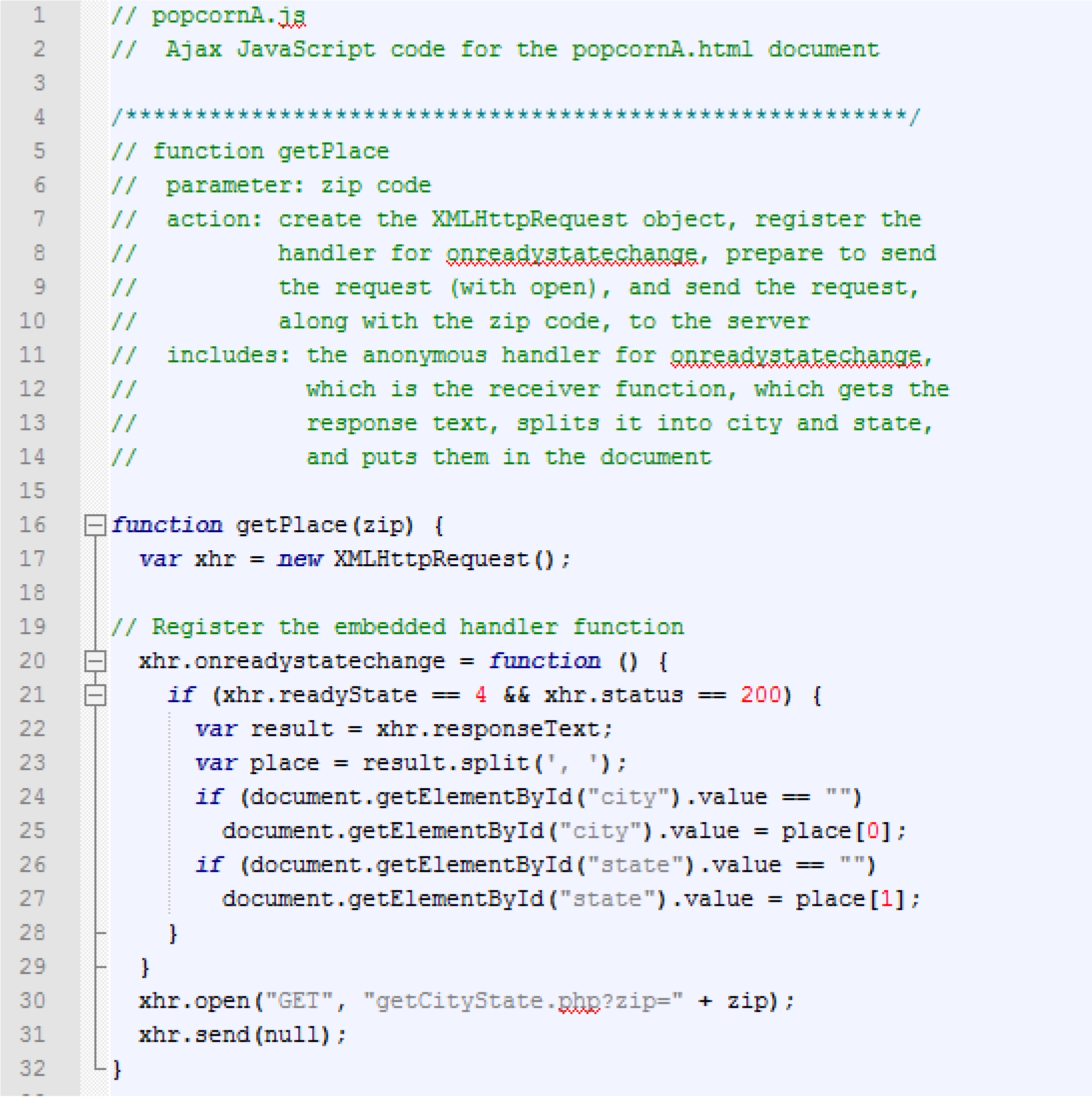
    1. Put all actions in the then clause of a selector that checks to see if `readyState` is `4`

    2. Get the response value from the `responseText` property of the XHR object

    3. Split it into its two parts

    4. Set the values of the city and state text boxes

→ popcornA.js



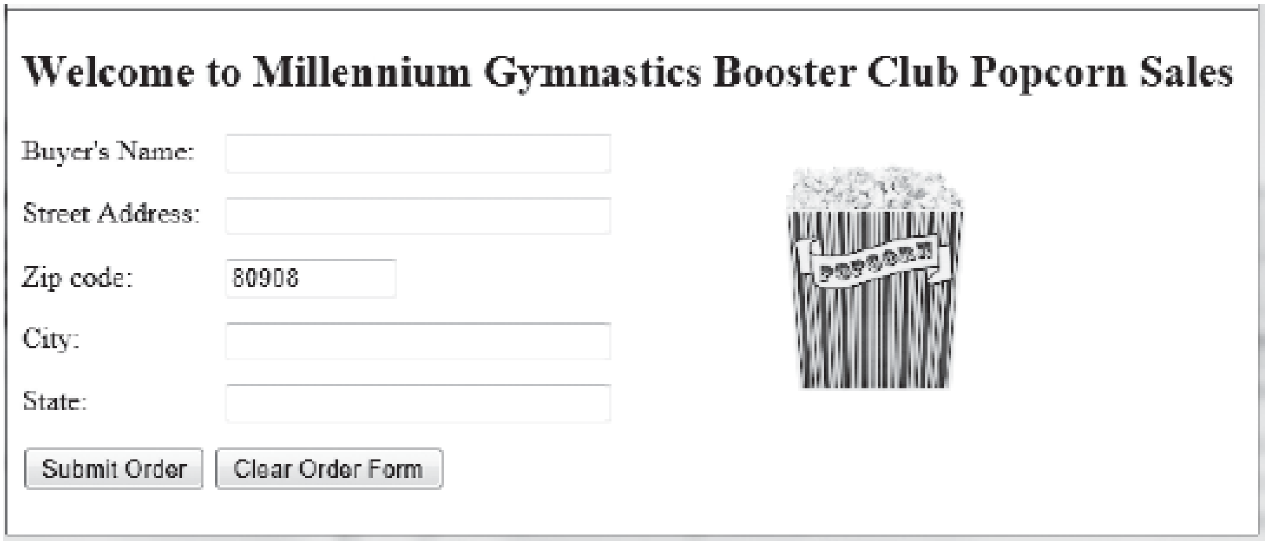
```
// popcornA.js
//  Ajax JavaScript code for the popcornA.html document

/********************************************************/
// function getPlace
//  parameter: zip code
//  action: create the XMLHttpRequest object, register the
//          handler for onreadystatechange, prepare to send
//          the request (with open), and send the request,
//          along with the zip code, to the server
//  includes: the anonymous handler for onreadystatechange,
//            which is the receiver function, which gets the
//            response text, splits it into city and state,
//            and puts them in the document

function getPlace(zip) {
  var xhr = new XMLHttpRequest();

// Register the embedded handler function
  xhr.onreadystatechange = function () {
    if (xhr.readyState == 4 && xhr.status == 200) {
      var result = xhr.responseText;
      var place = result.split(', ');
      if (document.getElementById("city").value == "")
        document.getElementById("city").value = place[0];
      if (document.getElementById("state").value == "")
        document.getElementById("state").value = place[1];
    }
  }
  xhr.open("GET", "getCityState.php?zip=" + zip);
  xhr.send(null);
}
```

- Cross-Browser Support

  - What we have works with FX3 and IE9, but not IE browsers before IE7

  - IE5 and IE6 support an `ActiveXObject` named `Microsoft.XMLHTTP`

```
 xhr = new
ActiveXObject("Microsoft.XMLHTTP");
```



Welcome to Millennium Gymnastics Booster Club Popcorn Sales

Buyer's Name:

Street Address:

Zip code: 80908

City:

State:

Figure 10.3 Display of the form after the zip code has been entered

10.3 Return Document Forms

1. *HTML*

- Most common approach is to place an empty div element in the original document

  - The innerHTML property of the div element is assigned the new content

```
<div id = "replaceable_list">
  <h2> 2010 US Champion/Runnerup – baseball </h2>
    <ul>
      <li> Texas Rangers </li>
      <li> San Francisco Giants </li>
    </ul>
</div>
```

Now, if the user selects a different sport, say football, the HTML response fragment could have the following:

```
  <h2> 2011 US Champion/Runnerup – football </h2>
  <ul>
    <li> Green Bay Packers </li>
    <li> Pittsburgh Steelers </li>
  </ul>
```

10.3 Return Document Forms (continued)

1. *HTML* (continued)

Now, the returned fragment can be inserted in the `div` element with

```
var divDom = document.getElementById(
                                 "replaceable_list");
divDom.innerHTML = xhr.responseText;
```

- The disadvantage of using HTML for the return document is it works well only if markup is what is wanted.

2. *XML*

- For the previous example, the following would be returned:

```
<header> 2007 US Champion/Runnerup – football
</header>
<list_item> New York Giants </list_item>
<list_item> New England Patriots </list_item>
```

10.3 Return Document Forms (continued)

2. *XML* (continued)

- Problem: the XML returned must also be parsed
  - Two approaches:
    - A. Use the DOM binding parsing methods
      - Two disadvantages:
        - i. Writing the parsing code is tedious
        - ii. Support for DOM parsing methods is a bit inconsistent over various browsers
    - B. Use XSLT style sheets
      - For the example, see next page

10.3 Return Document Forms (continued)

2. *XML* (continued)

```
<xsl:stylesheet version = "1.0"
  xmlns:xsl =
      "http://www.w3.org/1999/XSL/Transform"
  xmlns = "http://www.w3.org/1999/xhtml" >
  <xsl:template match = "/">
  <h2> <xsl:value-of select = "header" />
  </h2> <br /> <br />
    <ul>
      <xsl:for-each select = "list_item">
        <li> <xsl:value-of select = "list_item"/>
          <br />
        </li>
      </xsl:for-each>
    </ul>
  </xsl:template>
</xsl:stylesheet>
```

3. *JavaScript Object Notation (JSON)*

- Part of the JavaScript standard, 3rd edition

- A method of representing objects as strings, using two structures
- Easy for people to read and write and easy for machines to parse and generate

A. Collections of name/value pairs
B. Arrays of values

10.3 Return Document Forms (continued)

3. *JavaScript Object Notation (JSON)* (continued)

```
{"employees" :
  [
    {"name" : "Dew, Dawn", "address" :
      "1222 Wet Lane"},
    {"name" : "Do, Dick", "address" :
      "332 Doer Road"},
    {"name" : "Deau, Donna", "address" :
      "222 Donne Street"}
  ]
}
```

This object consists of one property/value pair, whose value is an array of three objects, each with two property/value pairs

Array element access can be used to retrieve the data elements

```
var address2 = myObj.employees[1].address;
```

puts `"332 Doer Road"` in `address2`

- **JSON objects are returned in `responseText`**
  - **How does one get the object, `myObj`?**

10.3 Return Document Forms (continued)

3. *JavaScript Object Notation (JSON)* (continued)

- The object could be obtained by running `eval` on the response string

- It is safer to get and use a JSON parser

```
var response = xhr.responseText;
var myObj = JSON.parse(response);
```

- *JSON has at least three advantages over XML*

1. JSON representations are smaller

2. `parse` is much faster than manual parsing or using XSLT

3. `parse` is much easier than manual parsing or using XSLT

- XML is better if the returned data is going to be integrated with the original document – use XSLT

10.3 Return Document Forms (continued)

3. *JavaScript Object Notation (JSON)* (continued)

- Example return document:

```
{"top_two":
   [
     {"sport": "football", "team":
                         "Green Bay Packers"},
     {"sport": "football", "team":
                         "Pittsburgh Steelers"},
   ]
}
```

- The processing to put it in the HTML document:

```
var myObj = JSON.parse(response);
document.write("<h2> 2010 US Champion/Runnerup"
       + myObj.top_two[0].sport + "</h2>");
document.write("<ul> <li>" +
       myObj.top_two[0].team + "</li>");
document.write("<li>" + myObj.top_two[1].team
       + "</li></ul>");
```

10.4 Ajax Toolkits

- There are many toolkits to help build Ajax applications, for both server-side and client-side

- Client-side toolkits:

1. *Dojo*

    - A free JavaScript library of modules, for Ajax and other parts of Web site software

    - Provides commonly needed code and hides the differences among browsers

    - We will use only one function, `bind`, which creates an XHR object and builds an Ajax request

        - `bind` is part of the `io` module

    - To gain access to Dojo module, if `dojo.js` is in the `dojo` subdirectory of where the markup resides

```
<script type = "text/javascript"
  src = "dojo/dojo.js">
</script>
```

10.4 Ajax Toolkits (continued)

1. *Dojo* (continued)

- The `bind` function takes a single literal object parameter
  - a list of property/value pairs, separated by commas and delimited by braces
    - properties are separated from their values by colons
- The parameter must include `url` and `load` properties
  - The value of the `url` property is the URL of the server
  - The value of the `load` property is an anonymous function that uses the returned data
- It also should have `method`, `error` , and `mimetype` properties

The `getPlace` function, rewritten with Dojo's `bind`:

→ SHOW `dojo.io.bind`

10.4 Ajax Toolkits (continued)

1. *Dojo* (continued)

- An example – ordering a shirt on-line

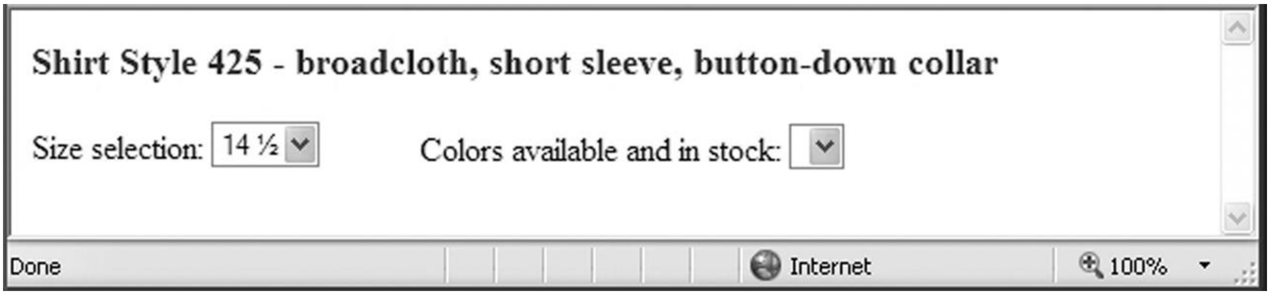
  - After the user selects a size, present the user with the colors in that size that are now in stock

    - Use Ajax to get the colors for the chosen size

  - The original document is for one particular style of shirt, including a menu for sizes and an empty menu for colors

→ SHOW `shirt.html`

→ SHOW `shirtstyles.css`



Shirt Style 425 - broadcloth, short sleeve, button-down collar

Size selection: 14 ½ Colors available and in stock:

10.4 Ajax Toolkits (continued)

1. *Dojo* (continued)

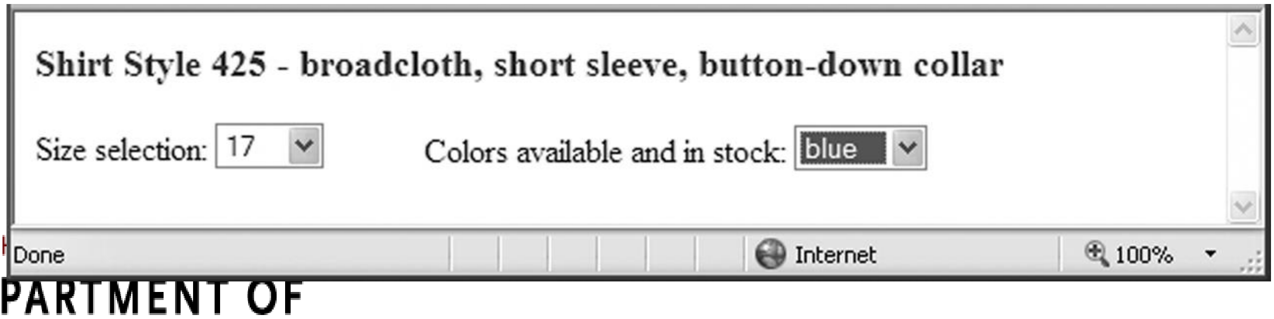
- The required JavaScript must define two functions

A. `buildMenu` – the callback function to build the menu of colors

- Get the DOM address of the empty select
- If it is not the first request, set `options` property to zero
- Split the returned value (a string of colors separated by commas and spaces)
- Build the `Options` of the menu and add them to the menu with `add`
  - The second parameter to `add` is browser-dependent; for IE, it is `-1`; for others, it is `null`

B. `getColors` – a wrapper function that calls `bind` to create the Ajax request

→ SHOW `shirt.js`



Shirt Style 425 - broadcloth, short sleeve, button-down collar

Size selection: 17

Colors available and in stock: blue

10.4 Ajax Toolkits (continued)

2. ***Prototype***

- A toolkit that extends JavaScript and provides tools for Ajax applications

- Includes a large number of functions and abbreviations of commonly needed JavaScript code

```
$("name") is an abbreviation for
document.getElementById("name")
```

- In Prototype, all of the Ajax functionality is encapsulated in the `Ajax` object

- A request is created by creating an object of `Ajax.Request` type, sending the parameters to the constructor

  - The first parameter is the URL of the server

  - The second parameter is a literal object with the other required information:

    - `method` – `"get"` or `"post"`
    - `parameters` – what to attach to the `get`
    - `onSuccess` – the anonymous callback function to handle the return
    - `onFailure` – the anonymous callback function for failure

→ SHOW the `Ajax.request` object creation

10.5 Security and Ajax

- *Issues*:

  1. In many cases, Ajax developers put security code in the client code, but it also must be included in the server code, because intruders can change the code on the client

  2. Non-Ajax applications often have just one or only a few server-side sources of responses, but Ajax applications often have many server-side programs that produce small amounts of data. This increases the attack surface of the whole application.

  3. Cross-site scripting – servers providing JavaScript code as an Ajax response. Such code could be modified by an intruder before it is run on the client
     - All such code must be scanned before it is interpreted

     - Intruder code could also come to the client from text boxes used to collect return data
       - It could include script tags with malicious code

Extra Credit Lab (10 points)

- Modify PopcornA example application to use Dojo
- Code is available under D2L-Week 16 module