Lecture 8
Session Handling

Lecture Agenda
Applied

1. Implementing Session Tracking from Scratch.
2. Using basic session tracking.
3. Understanding the session-tracking API.
4. Differentiating between server and browser sessions.
5. Encoding URLs.
6. Implementing an online shopping cart.

What is a Session?

What is a Session?

Definition

- Session:
  - In computer science, a session is semi-permanent interactive information interchange (a dialogue or conversation) between two or more communicating devices.

What is Session Tracking?

What is Session Tracking?

Definition

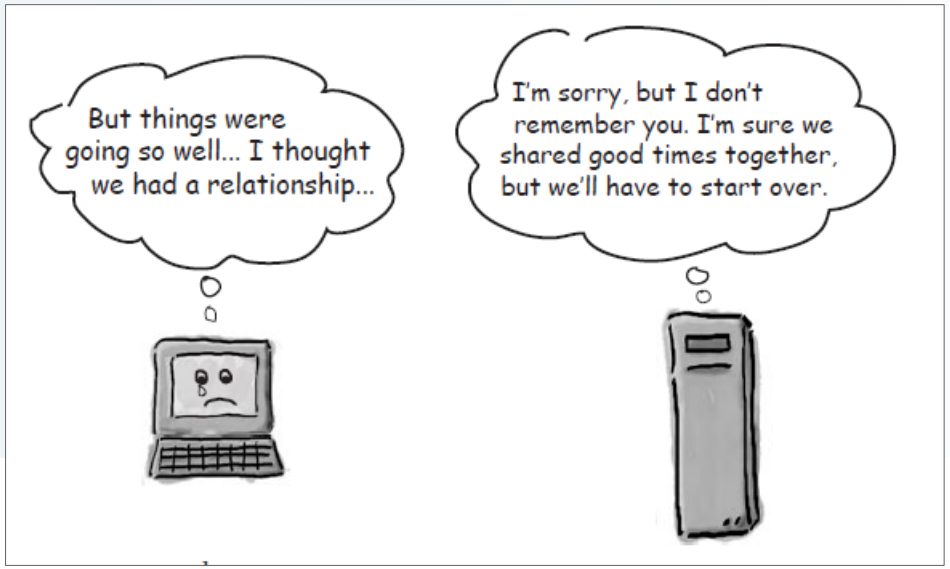
- Session Tracking:
  - Is the capability of a server to maintain the current conversational state of a single clients sequential requests.

Why Session Tracking?

Why Session Tracking?

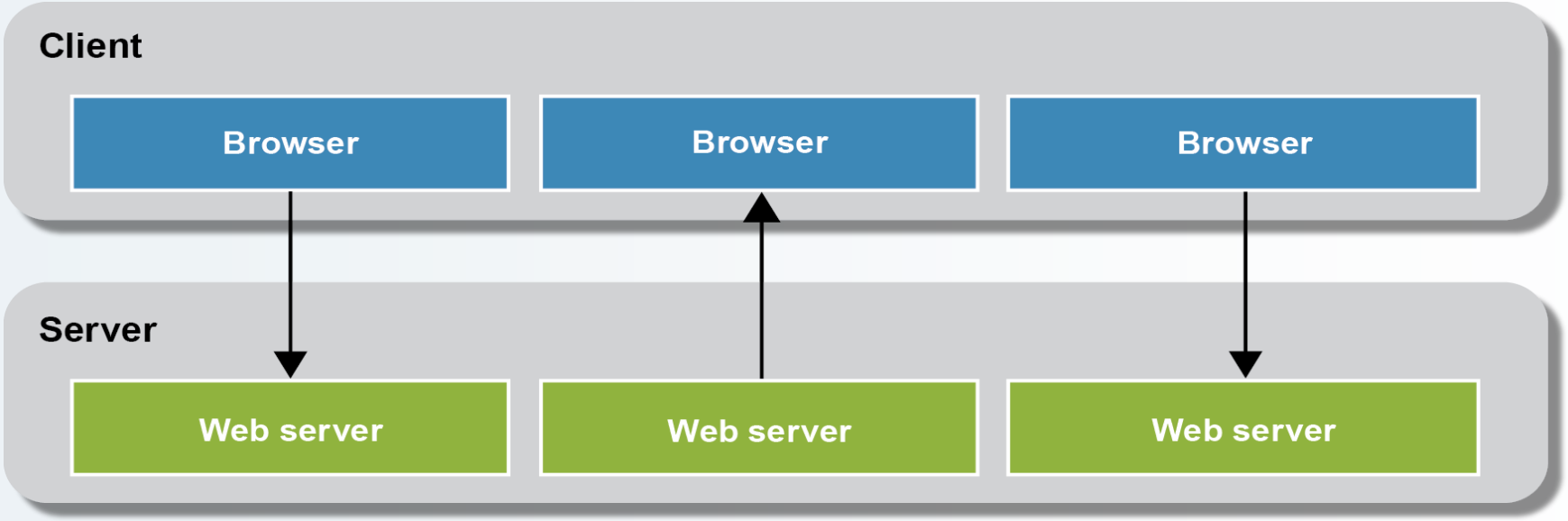
Why is it needed?

- Purpose?
  - HTTP is a **stateless protocol** which means each time a client retrieves a Web page, the client opens a separate connection to the server, and the server automatically does not keep any record of the previous request.
  - The stateless nature of HTTP becomes a problem when you need to know the sequence of actions a client has performed while on a site.



Why Session Tracking is difficult with HTTP

Diagram Overview



First HTTP Request:
The browser requests a page.

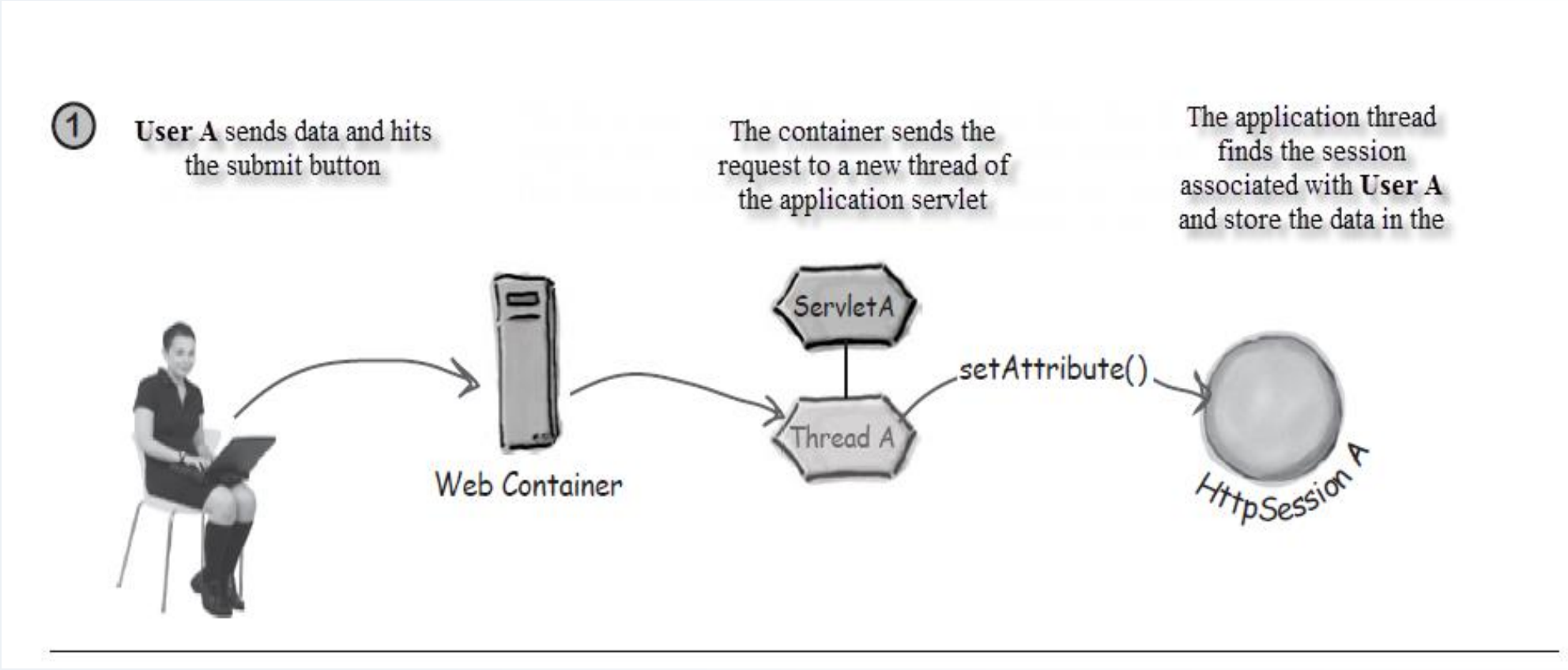
First HTTP Response:
The server returns the requested page and drops the connection.

Following HTTP Requests:
The browser requests a page. The web server has no way to associate the browser with its previous request.

How Session Tracking Works

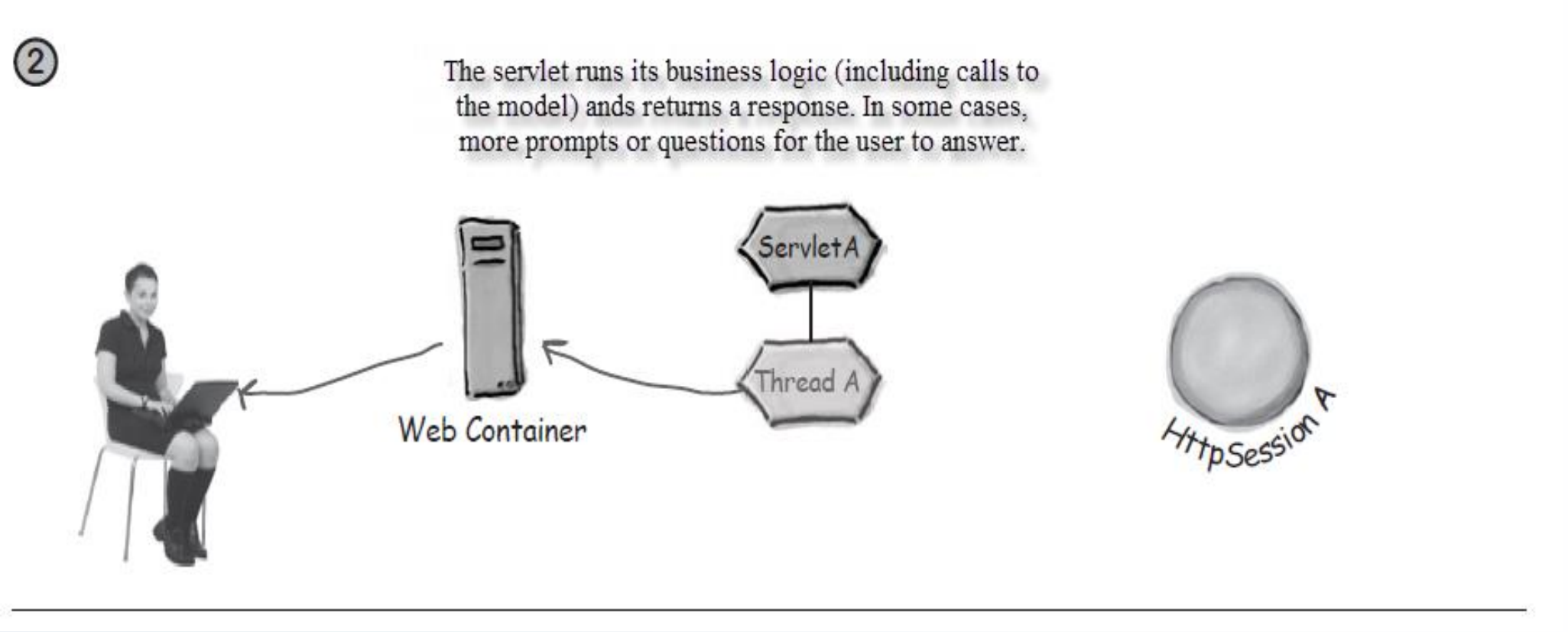
How Session Tracking Works

Conversation Scenario: (Part 1 of 5)



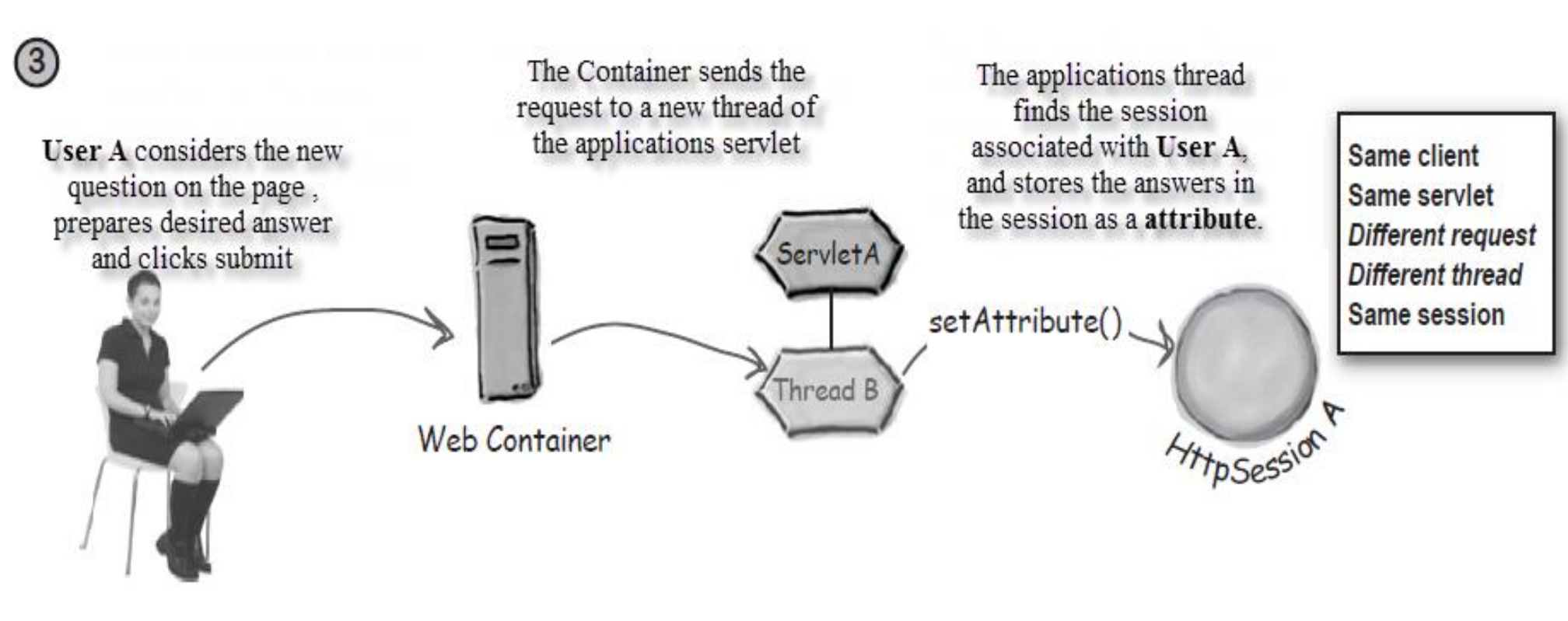
How Session Tracking Works

Conversation Scenario: (Part 2 of 5)



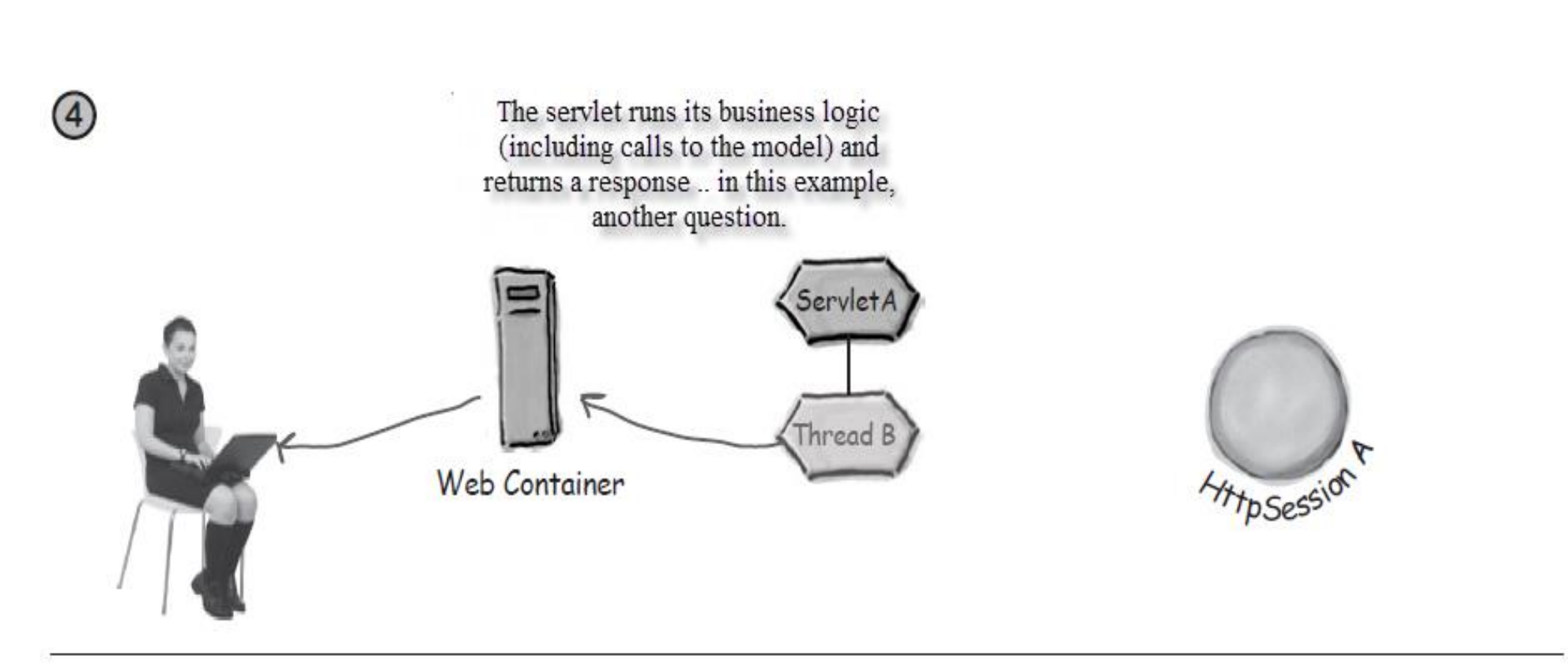
How Session Tracking Works

Conversation Scenario: (Part 3 of 5)



How Session Tracking Works

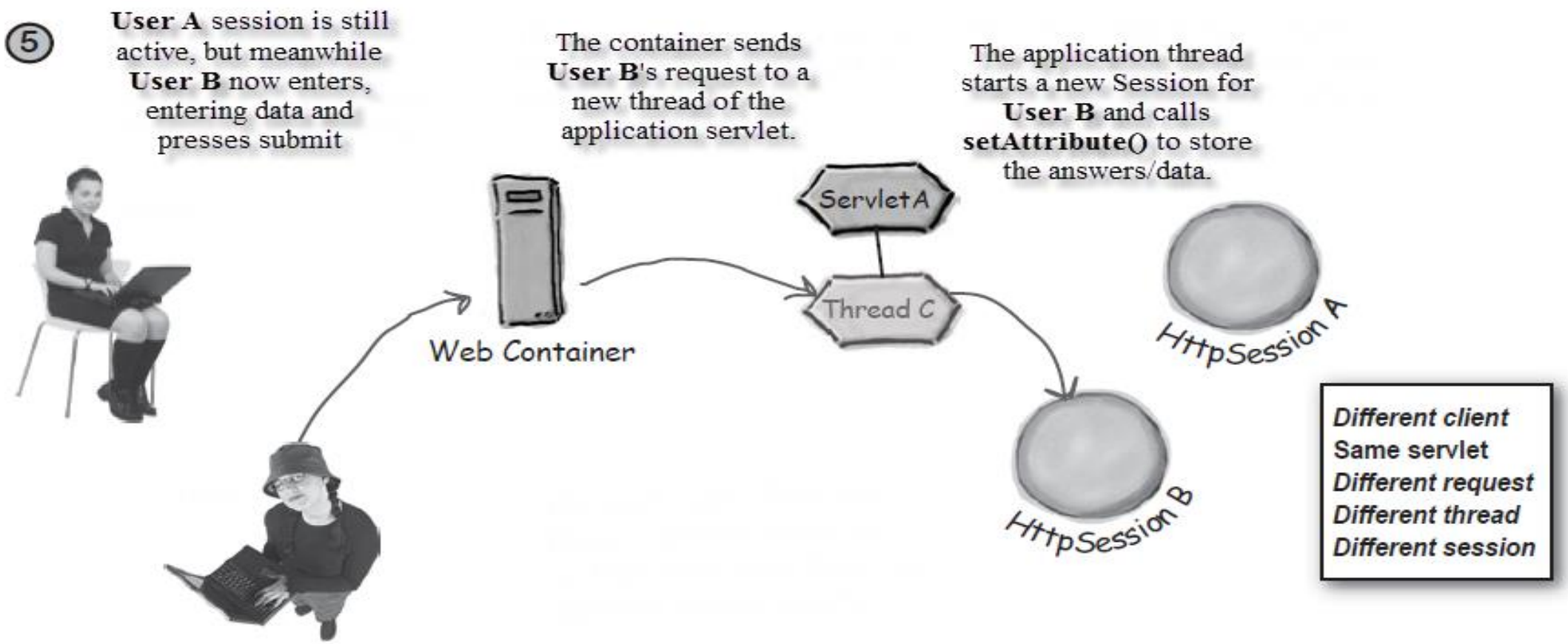
Conversation Scenario: (Part 4 of 5)



How Session Tracking Works

Conversation Scenario: (Part 5 of 5)

Meanwhile, imagine ANOTHER client goes to the same site ....



How to maintain client/server session state

Session Tracking Strategies

Strategy	Description
Cookies	• Server assigns a unique session Id as a cookie to each client. • Subsequent requests from client can be recognized utilizing received cookie. • Problem: Not all browsers support cookies.
Hidden Form Fields	• Server sends a hidden HTML form field along with unique session Id. `<input type="hidden" name="sessionid" value="1234">` • This means, when a form is submitted, name and value are in the POST/GET. • Each time the client sends a request back, Id value can be tracked. • Works even if cookies are disabled. • Problem: Not all HTML elements result in form submission (ex: <A HREF>)
URL Rewriting	• Append extra data on the end of each URL that identifies the session. • Server can associate that Id with the stored data http://localhost:8080/servletlabs/index.html;sessionId=1234 • Works even if cookies are disabled • Problem: Required to generate every URL dynamically.
HttpSession Object	• Java Servlet API provides HttpSession Interface • The Interface provides a way to identify a user across multiple pages. • The container uses this interface to associate a session with a client. • The session persists for a specified period of time. `HttpSession session = request.getSession()`

Implementing Your **Own** Session Tracking

Possibly: Associate Cookie with data on the server

```
String sessionID = makeUniqueString();
HashMap sessionInfo = new HashMap();
HashMap globalTable = findTableStoringSessions();
globalTable.put(sessionID, sessionInfo);

//store identifier in cookie
Cookie sessionCookie = new Cookie("SESSIONID", sessionID);
sessionCookie.setPath("/");
response.addCookie(sessionCookie);
```

Create a unique identifier and in session information globally for lookup.

Create cookie storing session identifier.

Implementing Your Own Session Tracking

Difficulty

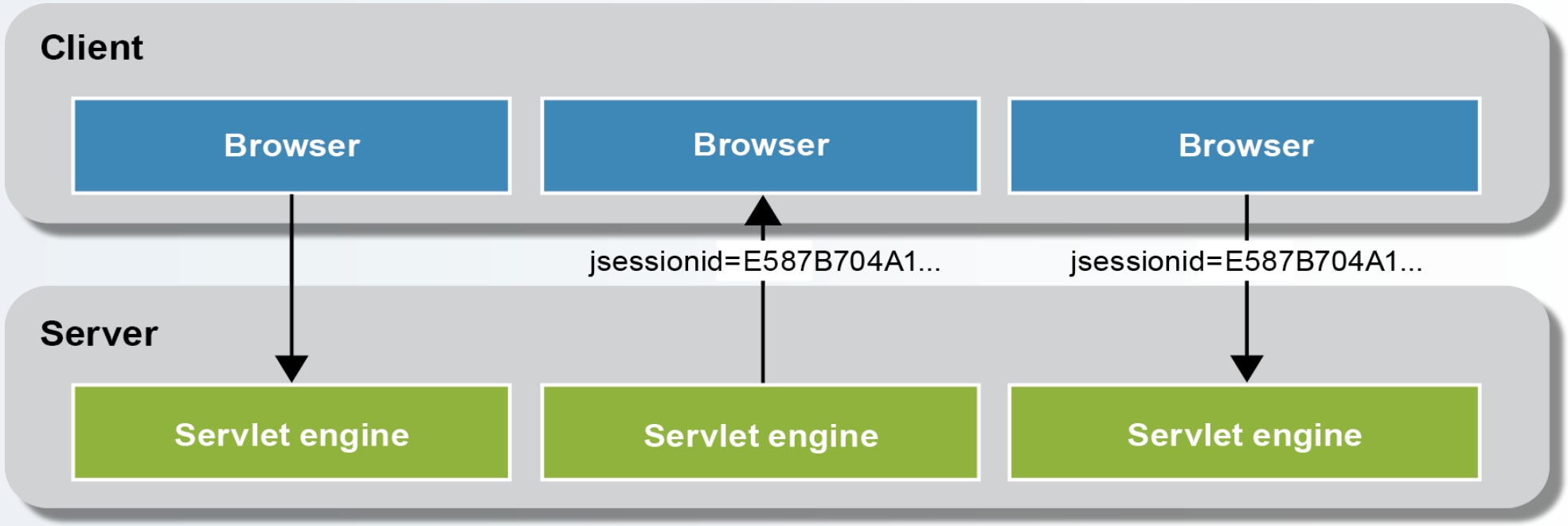
STILL TO BE DONE!

- Extract cookie that stores session identifier
- Set appropriate expiration time for the cookie
- Associate the hash tables with each request
- Generate the unique session identifier(s).

Java Session-Tracking

Java Session Management

How Java keeps track of Session



First HTTP Request:
The browser requests a JSP or servlet. The servlet engine creates a session object and assigns an ID for the session.

First HTTP Response:
The server returns the requested page and the ID for the session.

Following HTTP Requests:
The browser requests a JSP or servlet. The servlet engine uses the session ID to associate the browser with its session object.

Session Tracking Basics

Step	Description
1. Access the session object	• Call request.getSession() • Returns HttpSession object or if does not exist, creates one.
2. Look Up Information associated with session	• Call session.getAttribute() • Cast the returned object value to the appropriate type.
3. Store information in a session.	• Call session.setAttribute() • Use setAttribute with a key and value.
4. Discard session data	• Call session.removeAttribute() or session.invalidate() • Call removeAttribute() discards a specific value • Call invalidate to discard an entire session

Servlet Thread Safety

Thread Safety

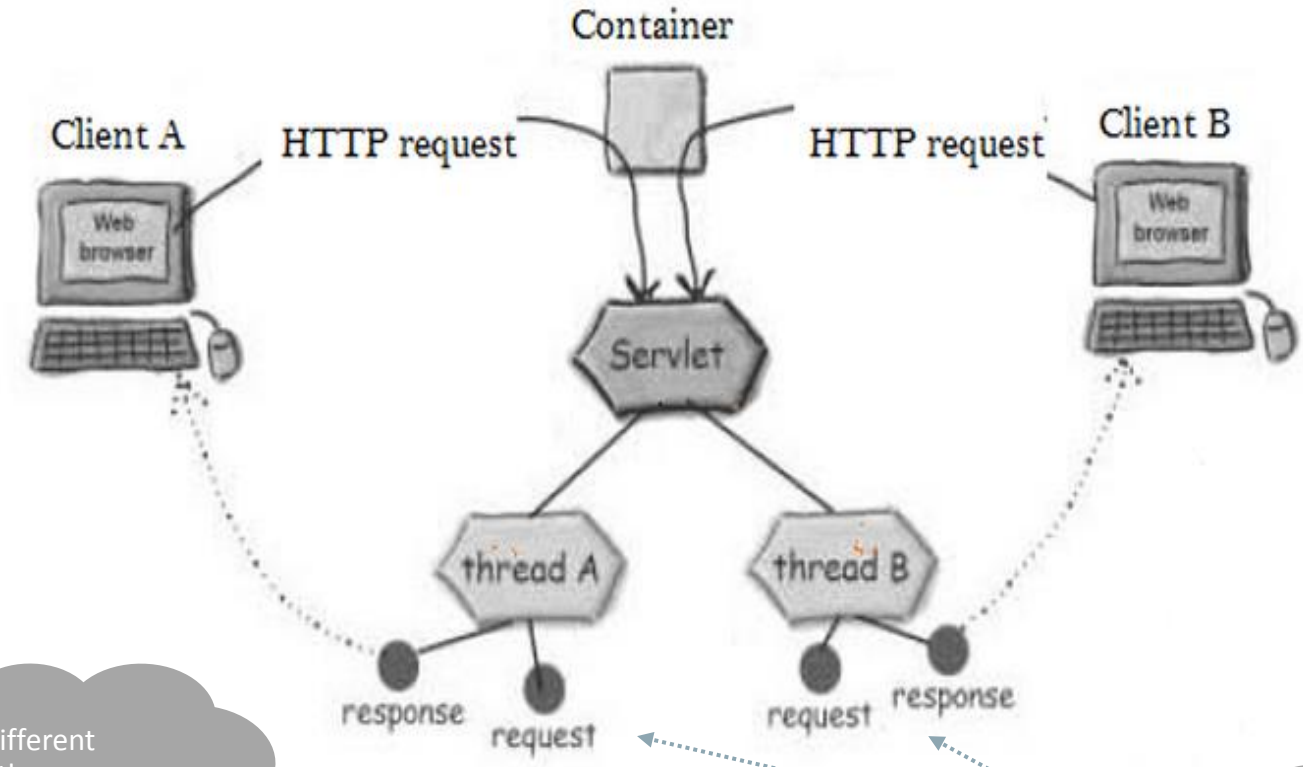
A Word on Thread Safety

- Each servlet creates one session object that exists for multiple requests that come from a single client.
- If the client has **one** browser window open, access to the session object is thread-safe.
- If the client has **multiple** browser windows open, two threads from the same client **could** access the same session object at the same time. As a result, the session object is **not** thread-safe.
- Since the servlet specification doesn’t guarantee that it will always return the same session object , you can’t make the session object thread-safe by synchronizing on it. Instead you can synchronize on the session Id string for the session object.

Thread Safety

A word on Thread Safety

Each client gets a separate thread for each request and the container allocates a new request and response object



It's feasible that different threads will access the same HttpSession object. Its even possible that serial requests from a client, a different thread will service each call.

Since the server hands out requests arbitrarily to the next available thread, we need to be sure that the different threads sees a consistent view of a given HttpSession object.

Synchronization Session Tracking Basics

Typical Sample Code

```
HttpSession session = request.getSession();
synchronized( session ){
        SomeClass value = (SomeClass) session.getAttribute("someID");
      if( value == null) {
          value = new SomeClass( … );
      }
      doSomething( value );
      session.setAttribute("someID", value);
}
```

1. Access Session Object

2. Thread Safety

3. Look Up Attribute

4. Store Information

Thread Safety

A word on Thread Safety

- Why?
  - A Java servlet container is typically **multithreaded**. That means, multiple requests to the same servlet may be executed at the same time.
  - When we start two or more threads within a program, there may be a situation when multiple threads try to access the same resource and produce an unforeseen result due to **concurrency** issues.
- How?
  - The Java programming language provides a very handy way of creating threads and synchronizing their tasks by using synchronized blocks. You keep shared resource within this block.

```
synchronized( objectIdentifier ) {
    // Access shared variables and other shared resources
}
```

Java Session API

HttpSession Methods

Step	Description
getAttribute()	• Extracts a previously stored value from a session object. • Returns **null** if no value is associated with given name.
setAttribute()	• Associates a value with a attribute name.
removeAttribute()	• Removes values associated with an attribute name.
getAttributeNames()	• Returns names of all attributes in the session.
getId()	• Returns the unique identifier.

HttpSession Methods Continued

Step	Description
isNew()	• Determines if session is new to client (session id supplied or not).
getCreationTime()	• Returns time at which session was first created.
getLastAccesseTime()	• Returns time at which session was last sent from client.
getMaxInactiveInterval(), setMaxInactiveInterval()	• Get/Sets the amount of time session should go without access before being invalidated.
invalidate()	• Invalidates current session.

Storing Simple Session Values

Storing Simple Session Values

Servlet that Shows Per-Client Access Count

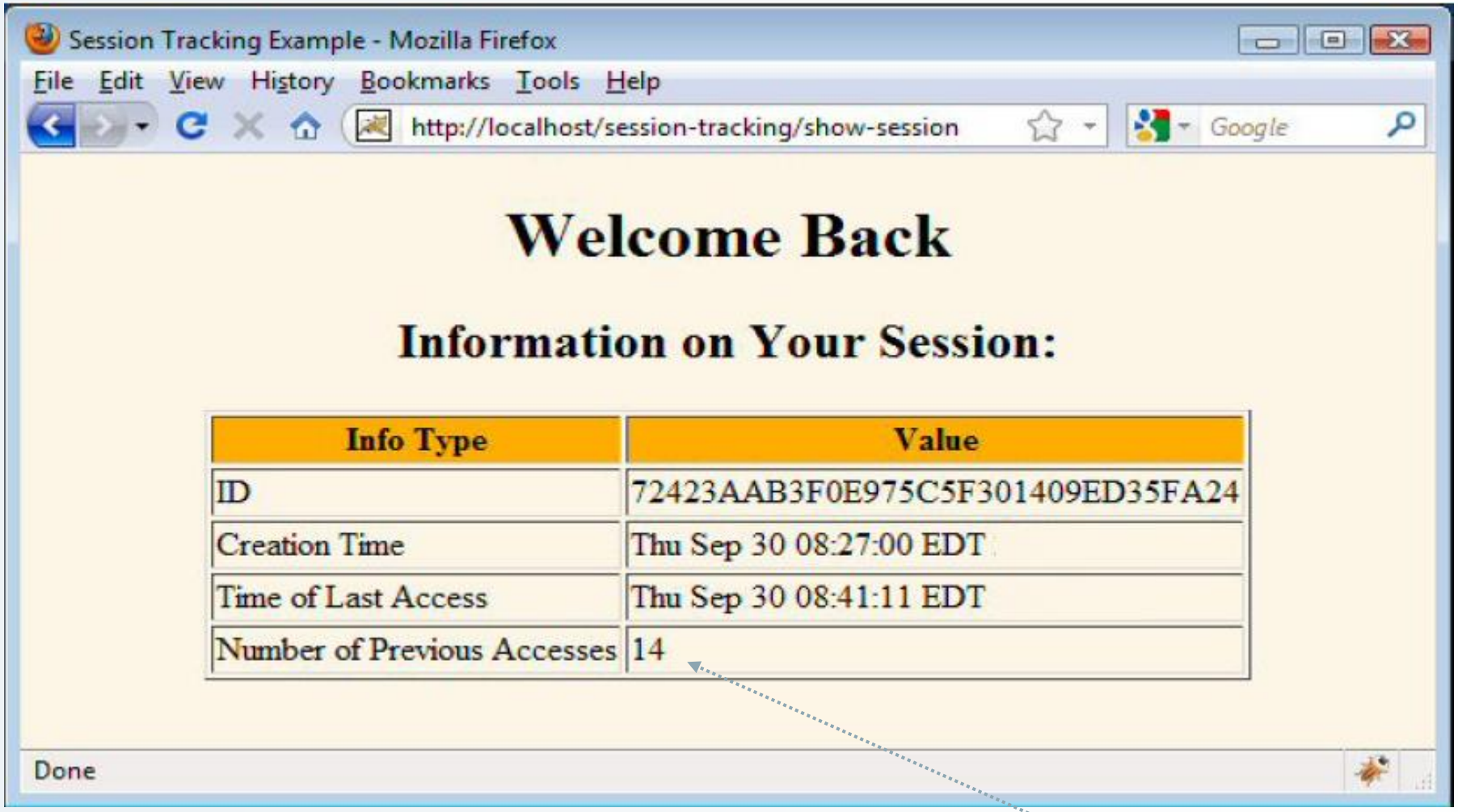
```
@WebServlet("/show-session")

public class ShowSession extends HttpServlet {

@Override

public doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        response.setContentType("text/html");

        HttpSession session = request.getSession();

         synchronized( session ){

            String heading;

            Integer accessCount = (Integer) session.getAttribute("accessCount");

            if( accessCount == null ) {

               heading = "Welcome, Newcomer";

                  }  else{

                        heading = "Welcome, Back";

                        accessCount  =  accessCount +1;

                  }

                  session.setAttribute("accessCount", accessCount);

         }

  }

} //OUTPUT HTML
```

Get session attribute "**accessCount**"

Increment "**accessCount**" based on session value stored

Storing Simple Session Values

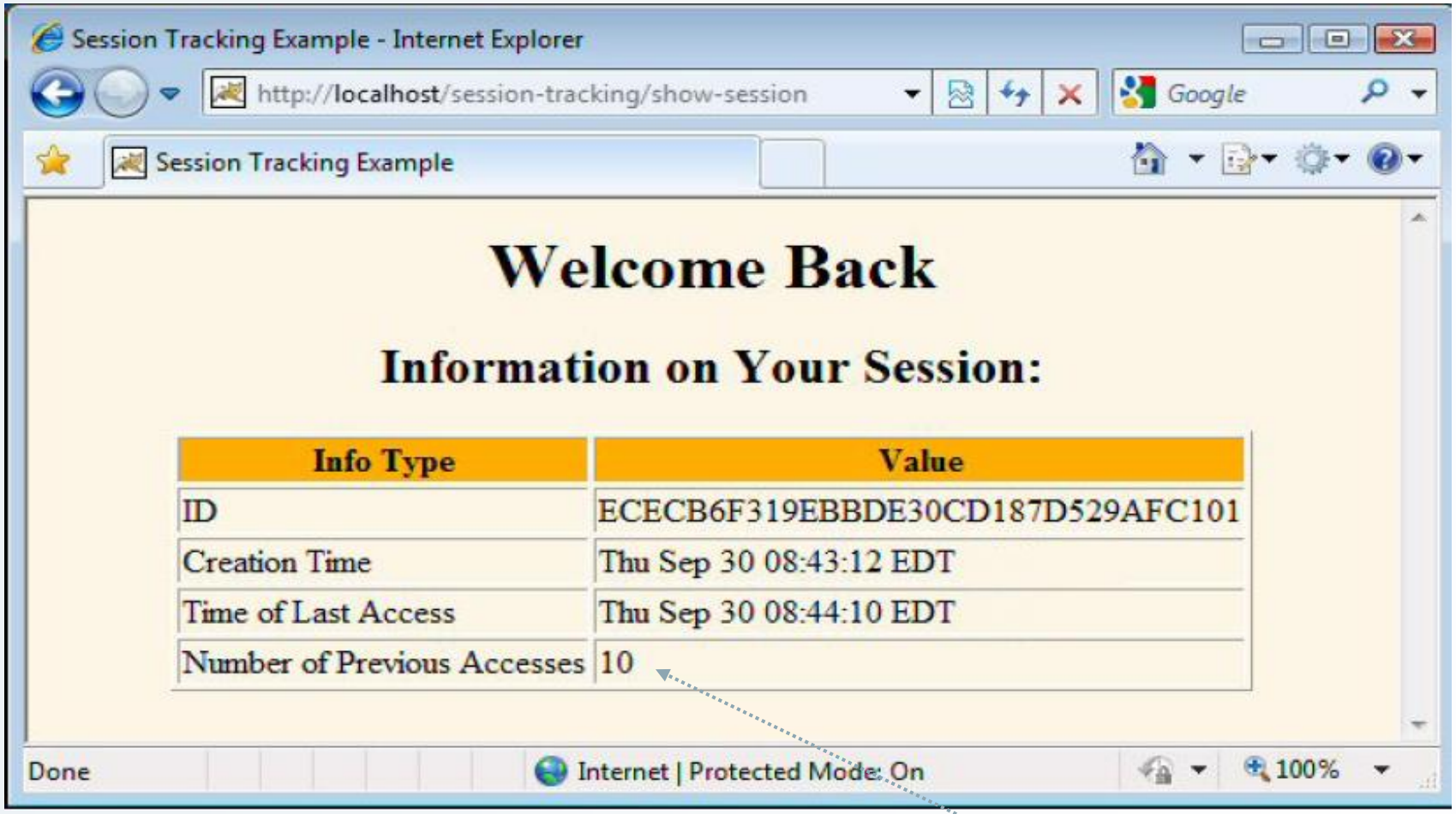
Result: User 1



Session Count
(accessCount)

Storing Simple Session Values

Result: User 2



Session Count
(accessCount)

Storing Lists of Values in Session

Storing Lists of Values in Session

Example: Accumulating a List of User Data

```
HttpSession session = request.getSession();
synchronized( session ) {
    @SuppressWarnings("unchecked")
    List<String> previousItems = (List<String>) session.getAttribute("previousItems");
    if( previousItems == null ) {
        previousItems = new ArrayList<String>();
    }
    String newItem = request.getParameter("newItem");
    if ( ( newItem  != null ) && (!newItem.trim().equals("")) ){
        previousItems.add( newItem );
    }
    session.setAttribute("previousItems", previousItems);
}
//CREATE HTML OUTPUT
```

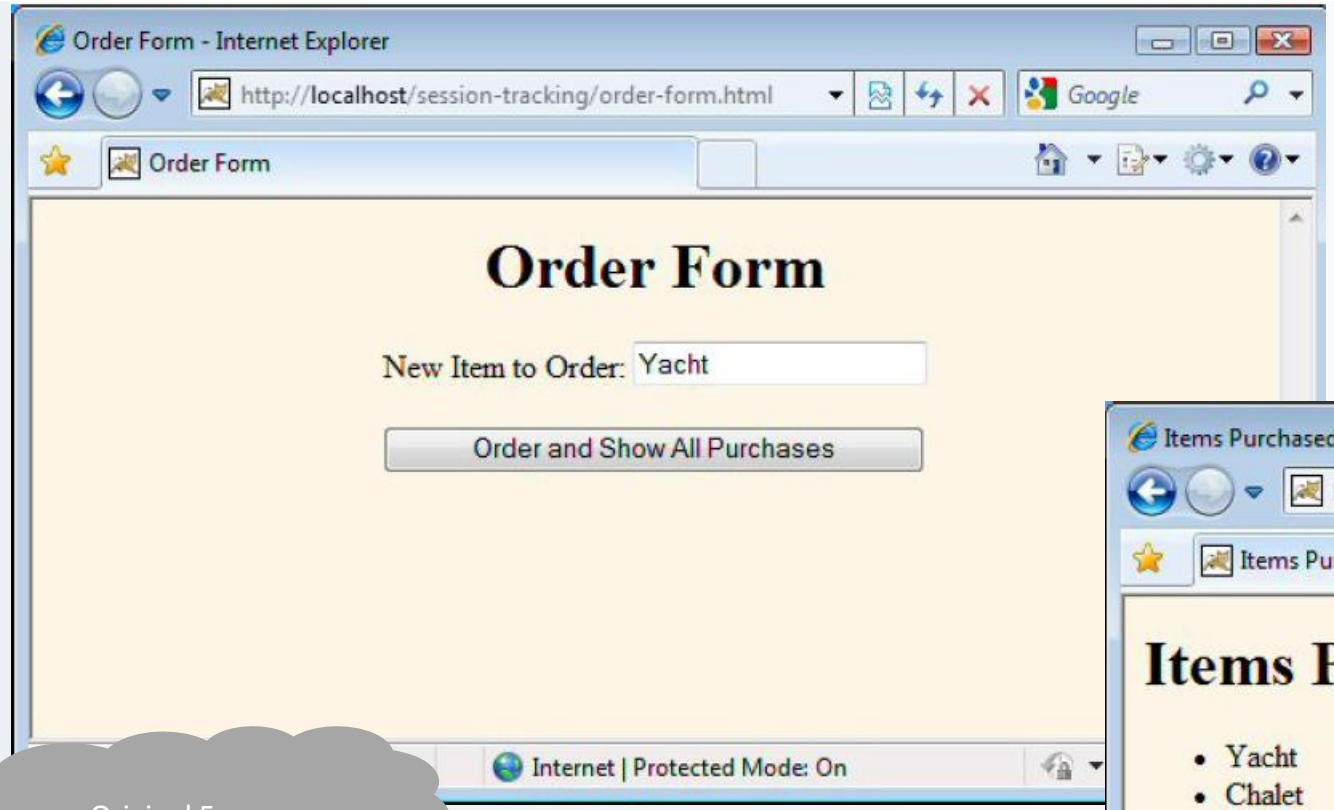
Get session attribute "previousItems"

Read/Get parameter input from page

Set updated value for "previousItems"

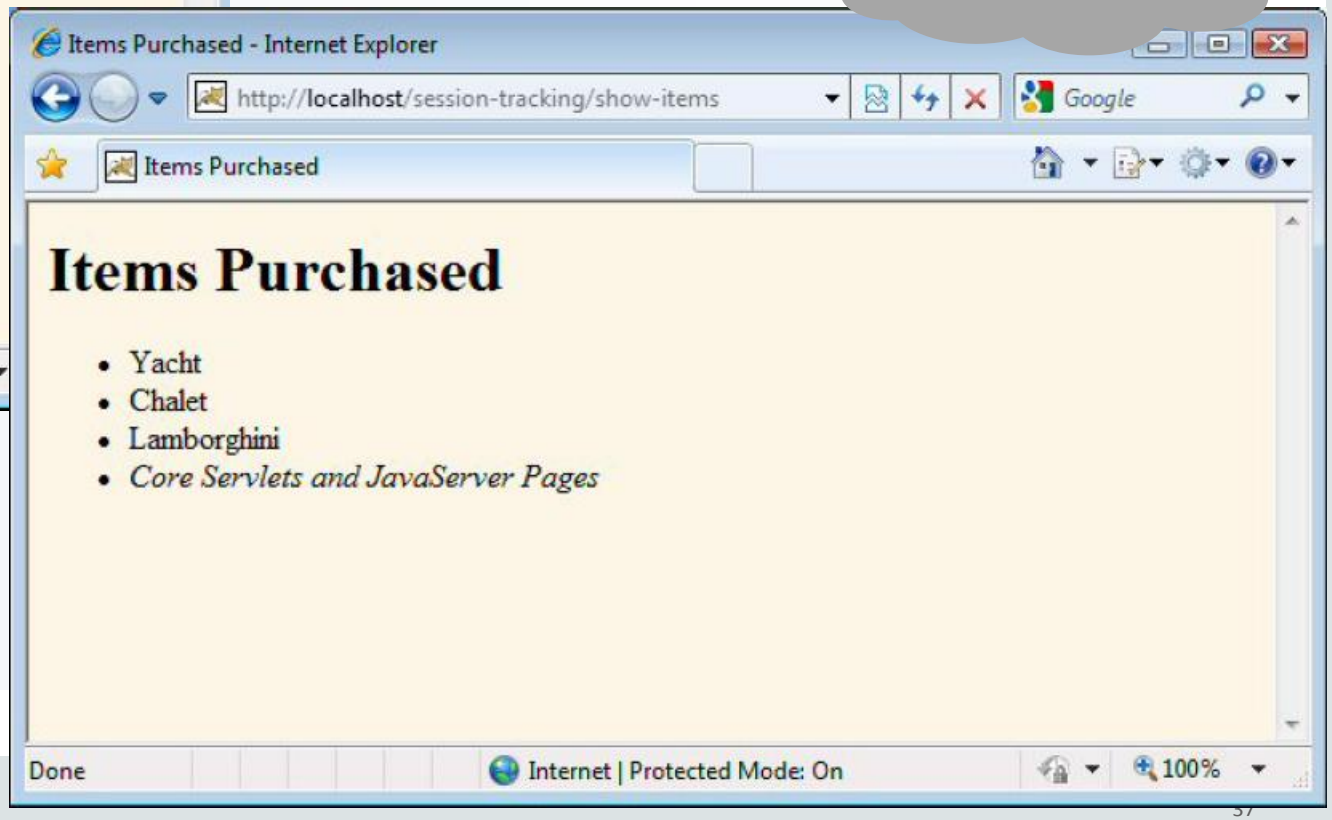
Storing List of Values in Session

Result



Original Form

Result



Pizza Website: Shopping Cart

Shopping Cart

Client Side: The HTML Form

```
<table>
<form action="addtocart">
    <th>Pizza Name</th> <th>Price</th>  <th>Add to Cart</th>
    <tr><td>Muffuleta</td><td>$20</td><td>
    <input type="hidden" name="name" value="Muffuleta">
    <input type="hidden" name="price" value="20">
    <input type="submit" value="Add to cart"></td>
    </tr>
</form>
<form action="addtocart">
    <tr><td>Veggie Delight</td><td>$40</td><td>
    <input type="hidden" name="name" value="Veggie Delight">
    <input type="hidden" name="price" value="40">
    <input type="submit" value="Add to cart"></td>
    </tr>
</form>
 <form action="addtocart">
    <tr><td>Margherita</td><td>$10</td><td>
    <input type="hidden" name="name" value="margherita">
    <input type="hidden" name="price" value="10">
    <input type="submit" value="Add to cart"></td>
    </tr>

</form>
</table>
```

Pizza Name	Price	Add to Cart
Muffuleta	$20	Add to cart
Veggie Delight	$40	Add to cart
Margherita	$10	Add to cart

In this example we use a basic HTML form for each item. Utilizing hidden fields to send data.

Shopping Cart

Server Side: The Cart Class

Create a new java class:

```
public class cart() { … }
```

Create a HashMap to hold cart item details:

```
HashMap<String, Integer> cartItems;
```

Constructor of cart class:

```
public cart() {
   cartItems = new HashMap<>();
}
```

Function to retrieve cart items:

```
public HashMap getCartItems(){
    return cartItems;
}
```

Add new Item to shopping cart:

```
public void addToCart(String itemId, int price){
   cartItems.put(itemId, price);
}
```

```
import java.util.HashMap;
/**
 *
 * @author rajat
 */
public class cart {
    HashMap<String, Integer> cartItems;
    public cart(){
     cartItems = new HashMap<>();

    }
    public HashMap getCartItems(){
        return cartItems;
    }
    public void addToCart(String itemId, int price){
        cartItems.put(itemId, price);
    }

}
```

Shopping Cart

Server Side: Session Handling points

Create a session object:

```
cart shoppingCart;
```

Retrieve the cart attribute from session object:

```
HttpSession session = request.getSession();
shoppingCart = (cart) session.getAttribute("cart");
```

If session for cart doesn't exist, then set a new attribute:

```
if( shoppingCart == null ) {
    shoppingCart = new Cart();
    session.setAttribute("cart", shoppingCart);
}
```

Shopping Cart

Server Side: Add Item to Cart

Fetched Selected Data:

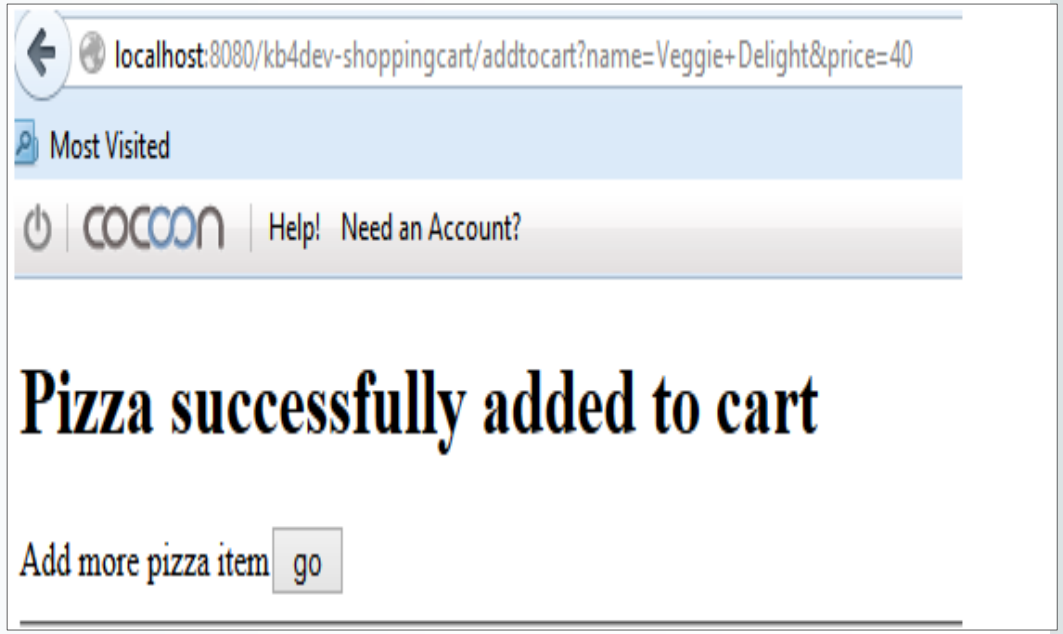
```
String name = request.getParameter("name");
Integer price = Integer.parseInt(request.getParameter("price"));
```

Put Data in HashMap:

```
shoppingCart.addToCart(name, price);
```

Update cart:

```
session.setAttribute("cart", shoppingCart);
```



Shopping Cart

Server Side: Display Items From Shopping Cart

Get value from shopping cart and assign to HashMap:

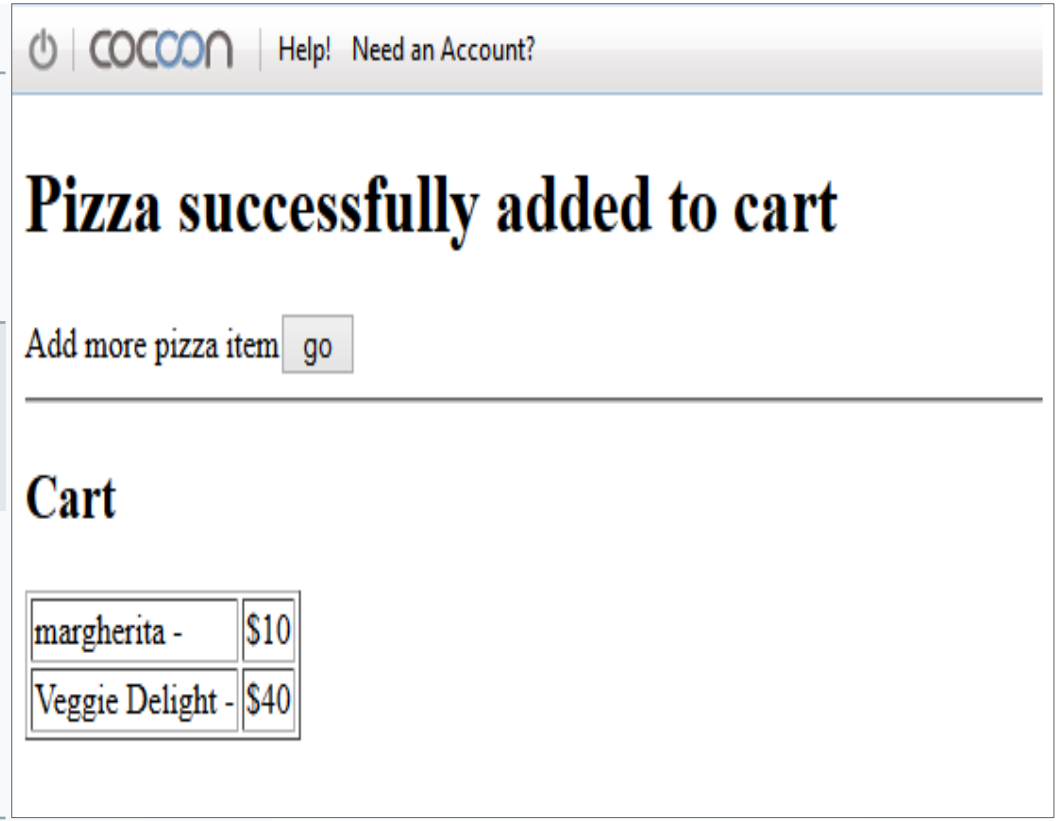
```
HashMap<String, Integers> items =
                                shoppingCart.getCartItems();
```

Create a new table to display pizza cart:

```
out.println("<table border='1px'>")
```

Each Item from HashTable will be different row in HTML table:

```
for( String key: items.keyset() ){
    out.println("<tr><td>"+ key +"- </td><td>"+
                "$"+ items.get(key) );
};
```



Shopping Cart

Server Side: Servlet Example Full Code

```
public class addtocart extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletRespons
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        HttpSession session = request.getSession();
        cart shoppingCart;
        shoppingCart = (cart) session.getAttribute("cart");
        if(shoppingCart == null){
          shoppingCart = new cart();
          session.setAttribute("cart", shoppingCart);
        }
        String name = request.getParameter("name");
        Integer price = Integer.parseInt(request.getParameter("price"));
        shoppingCart.addToCart(name, price);
        session.setAttribute("cart", shoppingCart);
        try (PrintWriter out = response.getWriter()) {
            /* TODO output your page here. You may use following sample code. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>result</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Pizza successfully added to cart </h1>");
            out.println("<form action='index.html'>Add more pizza item<input typ
            out.println("<hr>");
            out.println("<h2>Cart</h2>");
            HashMap<String, Integer> items = shoppingCart.getCartItems();
            out.println("<table border='1px'>");

            for(String key: items.keySet()){
                out.println("<tr><td>"+key+" - </td><td>"+"$"+items.get(key)+"</
            }
            out.println("<table>");
            out.println("</body>");
            out.println("</html>");

        }
    }
```

Set Session Attributes

Note: You can use the similar logic to implement delete (delete item etc…)

Questions?