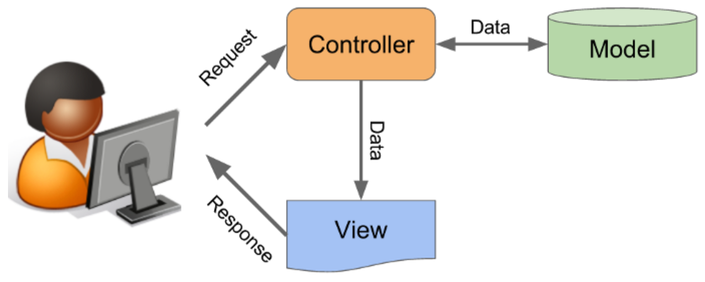
MVC PATTERN AND SHOPPING CART

1

MVC Pattern

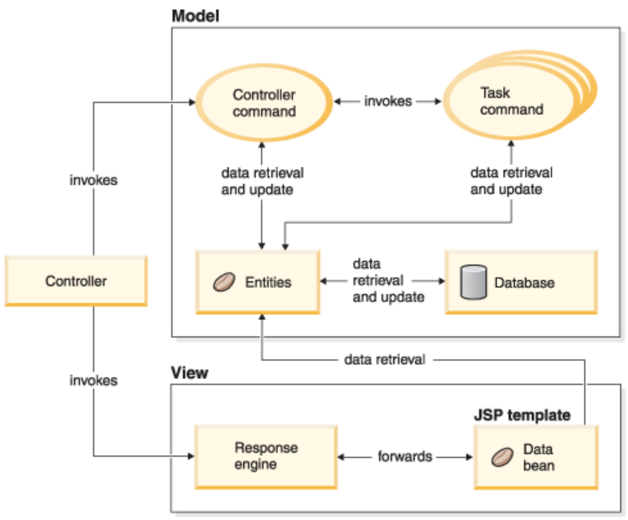
- The model-view-controller (MVC) design pattern specifies that an application consist of a data model, presentation information, and control information.



2

MVC Pattern

- The **Model** contains only the pure application data, it contains no logic describing how to present the data to a user.
- The **View** presents the model's data to the user. The view knows how to access the model's data, but it does not know what this data means or what the user can do to manipulate it.
- The **Controller** exists between the view and the model. It listens to events triggered by the view and executes the appropriate reaction to these events.
- In most cases, the reaction is to call a method on the model. Since the view and the model are connected through a notification mechanism, the result of this action is then automatically reflected in the view.



Controllers

- Controllers are what the user **interacts** with.
- They receive a **request** from the user, decide what to do, and send a **response** back.
- It's the **only** component that interacts with the models.

In JSP/Servlet, controller is servlet page. The servlet receipt data in web browser and do something with them.

With data receipt, The servlet decides what business logic code applies and which JSP page should present the results

Model

- Models are where an application's **data** are stored, usually in a database.
- Responsible for **storing** and **retrieving** data.
- They know **nothing** about the user interface.

The model is a java class with some method validate data, is where define business rules of data.

Should use model to connect database.

Model data is usually the same as the database data.

View

- Views are what the user **sees on the screen** (i.e. the HTML).
- They **present** the data to the user.

Displaying the Model data fetched by the Controller.

SHOPPING CART

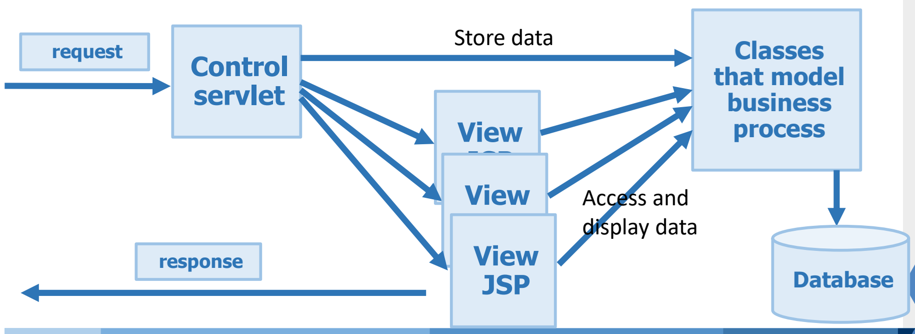
Shopping cart

- Shopping cart in eCommerce assist to visitors make **purchases online**. Upon checkout, the software **calculates** the total of the order, including **shipping fee** and payment information, etc.
- Shoping cart:
  - Add item.
  - Update item (color, size, quantity …)
  - Delete item
  - Calcutate bill.
  - …

Modeling the Business Process

- What **information** should a session deal with?
  - What information must be gathered via forms?
    - Items to purchase
    - Customer information, etc.
  - What information must be displayed to the user in response pages?
    - Shopping cart contents, etc.
  - What information must be stored **long term**?
    - Throughout **session**
    - Longer (in databases, etc.)
- **Model** of business process
  - Defined by **organization** you develop site for
  - Defined structure for each modeL

- **MODEL**: software classes that store/manipulate information gathered in session
  - Usually **separate** from servlets/JSPs
  - Servlets/JSPs **interact** with those classes
  - Usually working with **databases**

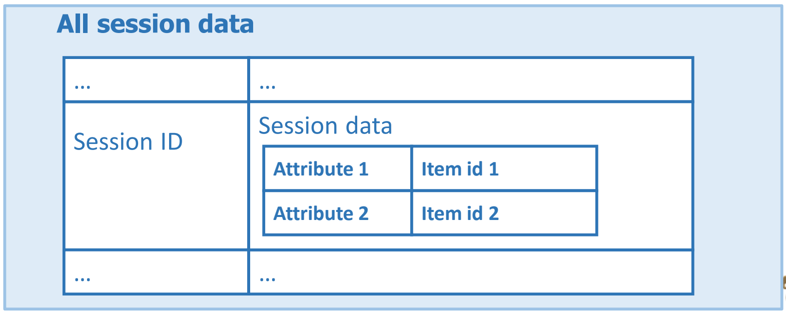


11

Store data solution

- Bad solution:

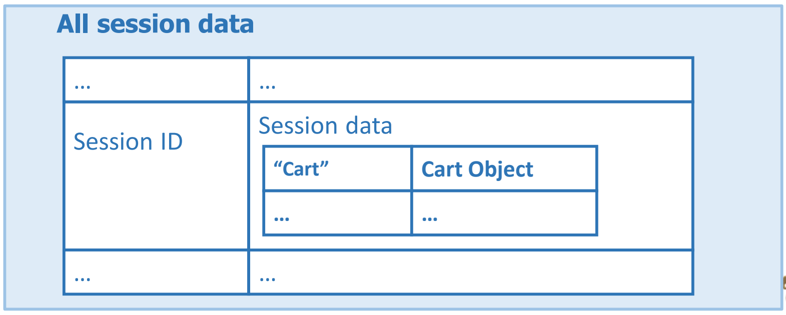
  Store **all information** to session as **separate attributes**
  - We need multiple attribute to store
  - Difficulties in data calculating.



12

Store data solution

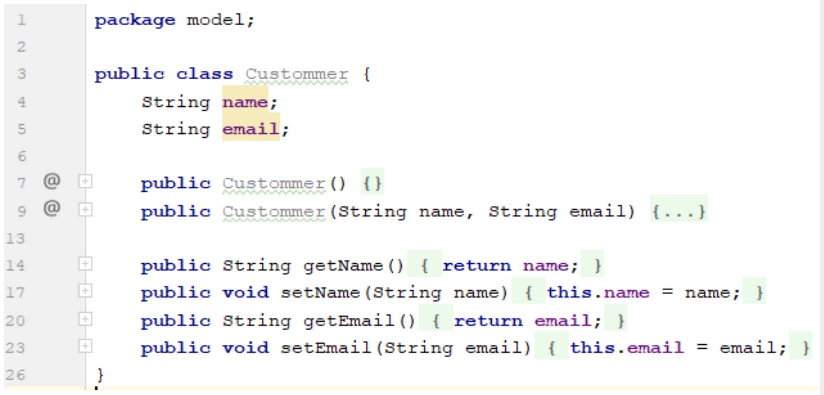
- Better solution:
  - Create classes and use data structure to store data
    - Data stored in session as a object.
    - Class have all method to **store/access/manipulate** data.



Model Class

- All information is stored as the variables
- Each variable has getter and setter method:
  - Setter: Takes value as parameter and stores it in state variable, validate all value put in
  - Getter: return current value of variable

Example Customer Class

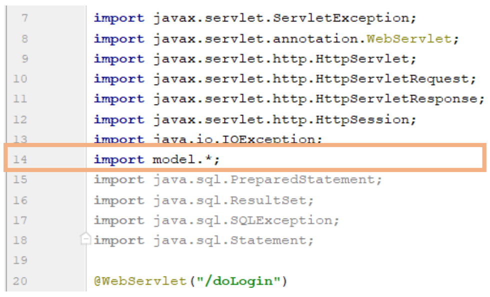


```
package model;

public class Custommer {
    String name;
    String email;

    public Custommer() {}
    public Custommer(String name, String email) {...}

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
}
```

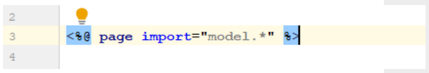
15

Using Model Classes

- We must include package name at top of file **package *packagename;***
- Other classes that use class must **import** the package
  - Servlets use <u>import</u> statement
  - JSPs use <u><@ page import</u> tag



```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.io.IOException;
import model.*;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

@WebServlet("/doLogin")
```

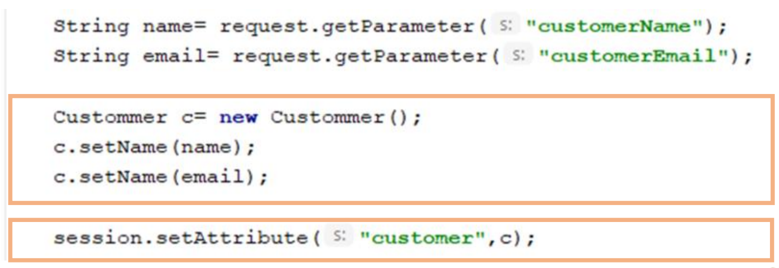


```
<%@ page import="model.*" %>
```

16

Using Model Classes

- In servlet:
  - Construct a new instance of the model object
  - Use its ***set*** methods to store parameters from form
  - Store the object as a session attribute

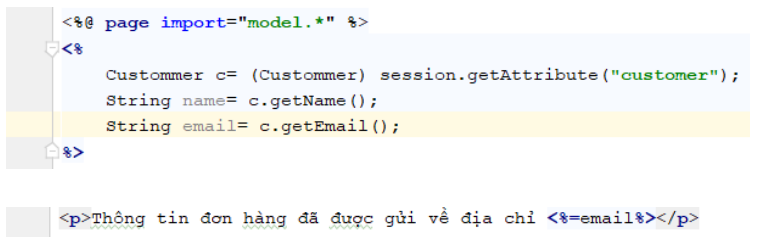


```
String name= request.getParameter( s: "customerName");
String email= request.getParameter( s: "customerEmail");

Custommer c= new Custommer();
c.setName(name);
c.setName(email);

session.setAttribute( s: "customer",c);
```

17

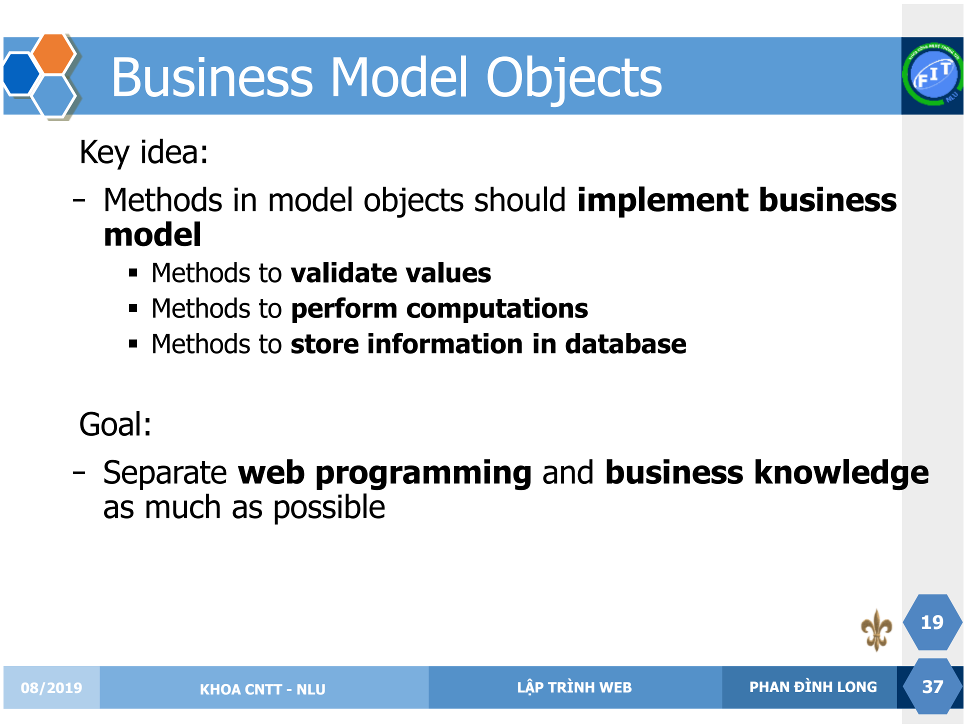
Using Model Classes

- In JSP:
  - Retrieve the object from the session attributes
    - Must cast back to its original type
  - Use its ***get*** methods to retrieve data
  - Display the data on the page



```
<%@ page import="model.*" %>
<%
    Custommer c= (Custommer) session.getAttribute("customer");
    String name= c.getName();
    String email= c.getEmail();
%>

<p>Thông tin đơn hàng đã được gửi về địa chỉ <%=email%></p>
```

18



Business Model Objects

Key idea:

- Methods in model objects should **implement business model**
  - Methods to **validate values**
  - Methods to **perform computations**
  - Methods to **store information in database**

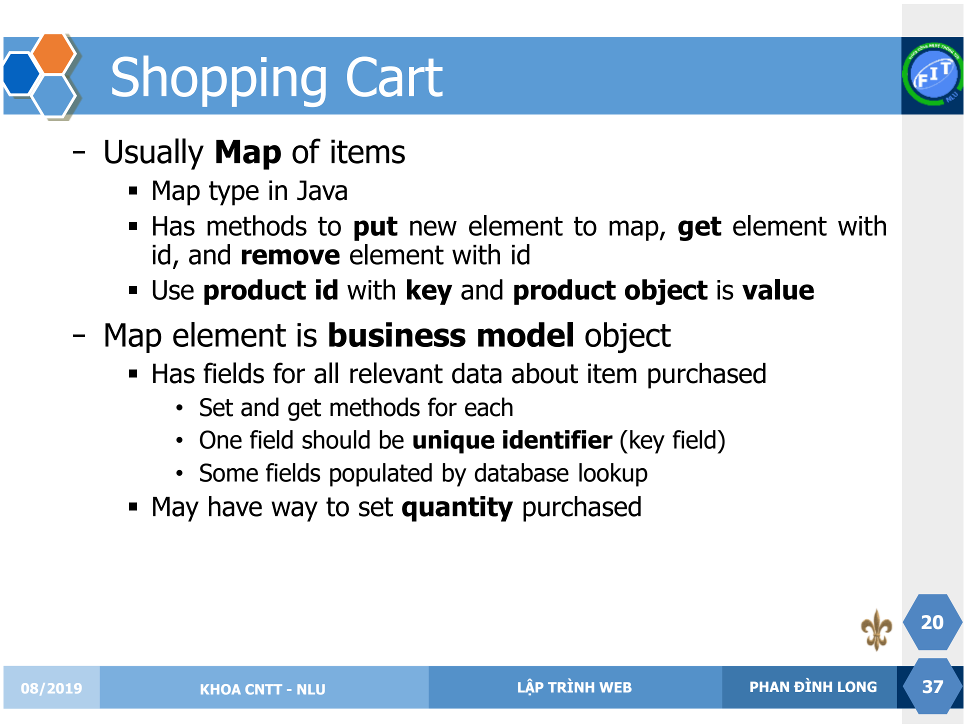
Goal:

- Separate **web programming** and **business knowledge** as much as possible

19

08/2019 KHOA CNTT - NLU LẬP TRÌNH WEB PHAN ĐÌNH LONG 37

19



Shopping Cart

- Usually **Map** of items
  - Map type in Java
  - Has methods to **put** new element to map, **get** element with id, and **remove** element with id
  - Use **product id** with **key** and **product object** is **value**
- Map element is **business model** object
  - Has fields for all relevant data about item purchased
    - Set and get methods for each
    - One field should be **unique identifier** (key field)
    - Some fields populated by database lookup
  - May have way to set **quantity** purchased

20

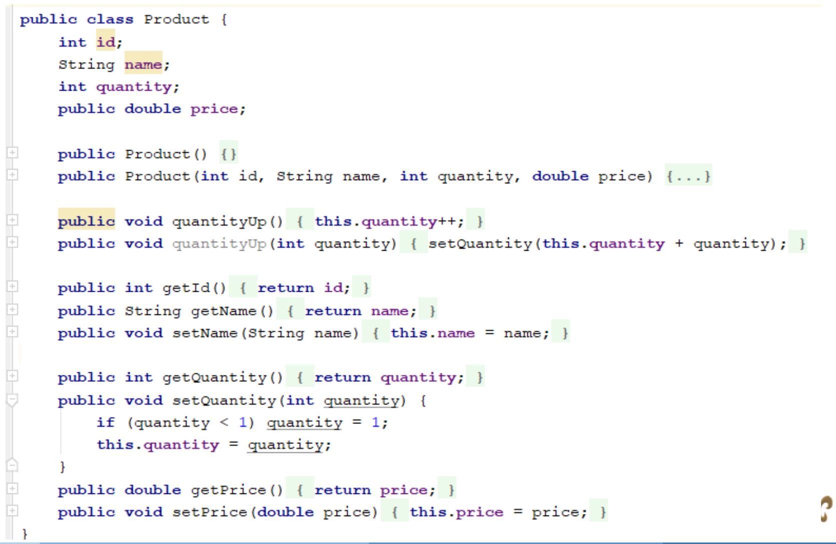
08/2019 KHOA CNTT - NLU LẬP TRÌNH WEB PHAN ĐÌNH LONG 37

20

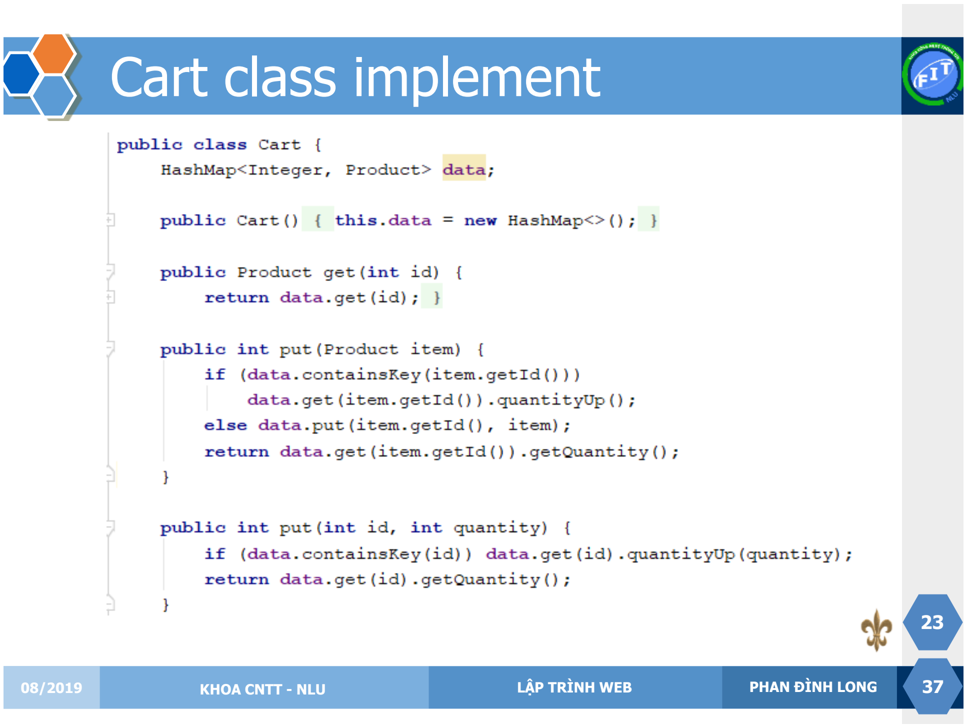
Shopping Cart API

- **`Product get(int id)`**
  - Lookup and return the Item in the list with the given ID
- **`int put(Product item)`**
  - add a new item to the cart with the given ID
- **`boolean remove(int id)`**
  - Delete the Item in the list with the given ID
- **`double total()`**
  - Return sum of cost of Cart
- **`Collection<Product> list()`**
  - Return list of Product in Cart

Product class sample



```
public class Product {
    int id;
    String name;
    int quantity;
    public double price;

    public Product() {}
    public Product(int id, String name, int quantity, double price) {...}

    public void quantityUp() { this.quantity++; }
    public void quantityUp(int quantity) { setQuantity(this.quantity + quantity); }

    public int getId() { return id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public int getQuantity() { return quantity; }
    public void setQuantity(int quantity) {
        if (quantity < 1) quantity = 1;
        this.quantity = quantity;
    }
    public double getPrice() { return price; }
    public void setPrice(double price) { this.price = price; }
}
```



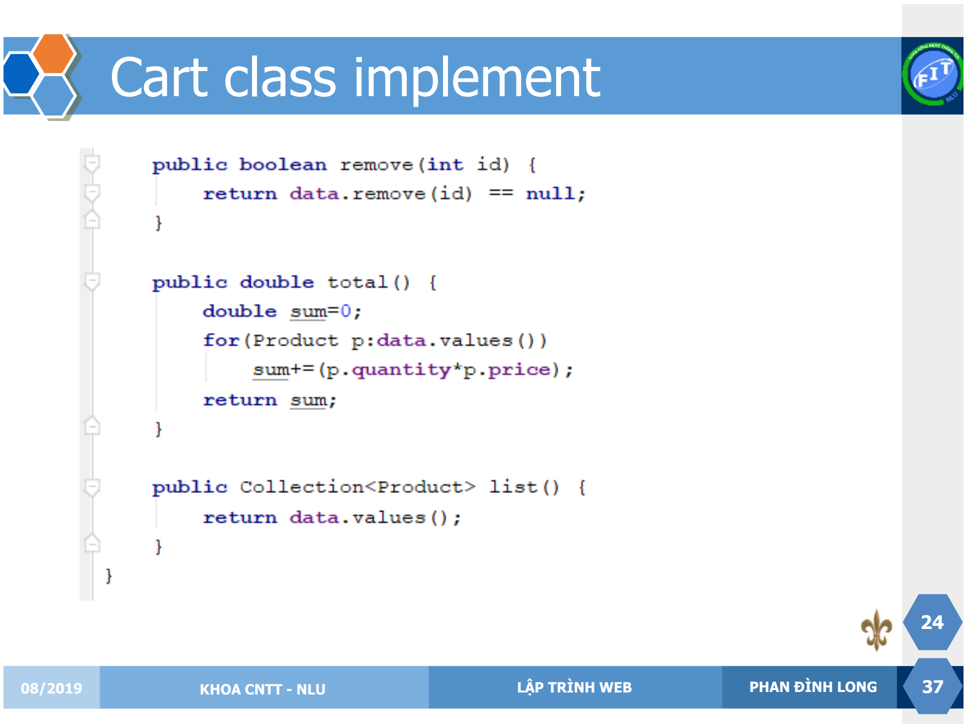
Cart class implement

```
public class Cart {
    HashMap<Integer, Product> data;

    public Cart() { this.data = new HashMap<>(); }

    public Product get(int id) {
        return data.get(id); }

    public int put(Product item) {
        if (data.containsKey(item.getId()))
            data.get(item.getId()).quantityUp();
        else data.put(item.getId(), item);
        return data.get(item.getId()).getQuantity();
    }

    public int put(int id, int quantity) {
        if (data.containsKey(id)) data.get(id).quantityUp(quantity);
        return data.get(id).getQuantity();
    }
```

23

08/2019 | KHOA CNTT - NLU | LẬP TRÌNH WEB | PHAN ĐÌNH LONG | 37

23



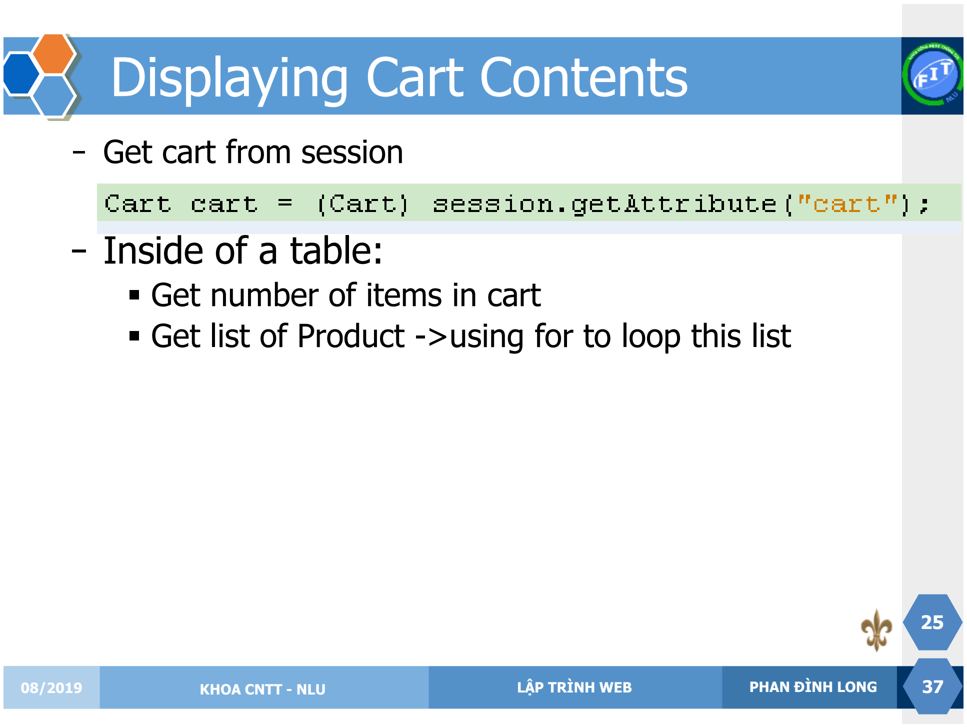
Cart class implement

```
    public boolean remove(int id) {
        return data.remove(id) == null;
    }

    public double total() {
        double sum=0;
        for(Product p:data.values())
            sum+=(p.quantity*p.price);
        return sum;
    }

    public Collection<Product> list() {
        return data.values();
    }
}
```

24

08/2019 | KHOA CNTT - NLU | LẬP TRÌNH WEB | PHAN ĐÌNH LONG | 37

24



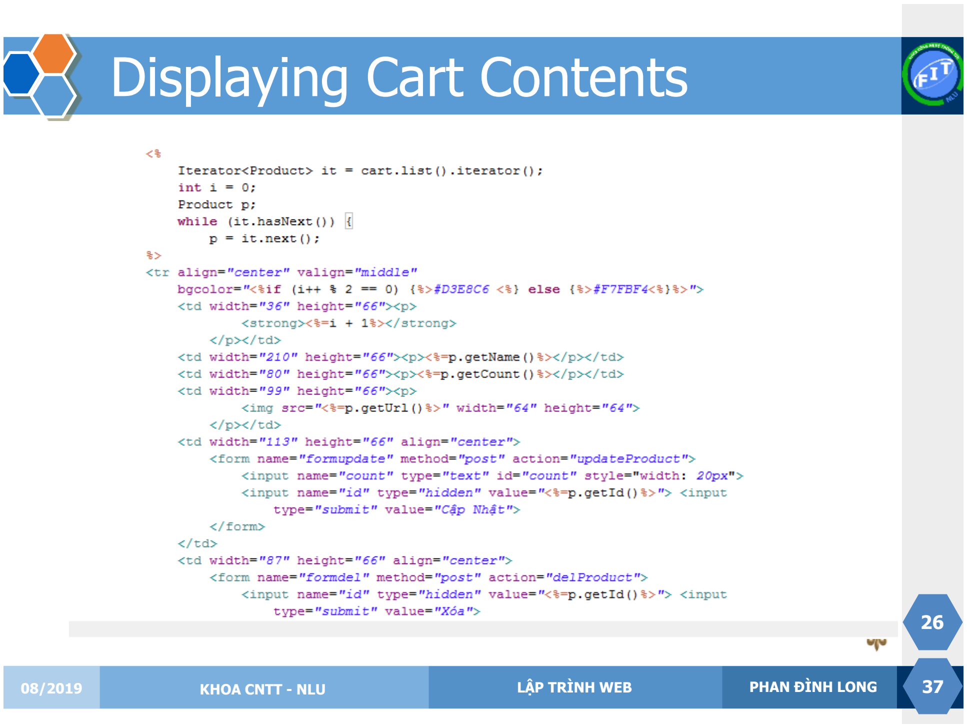
Displaying Cart Contents

- Get cart from session

```
Cart cart = (Cart) session.getAttribute("cart");
```

- Inside of a table:
  - Get number of items in cart
  - Get list of Product ->using for to loop this list

25

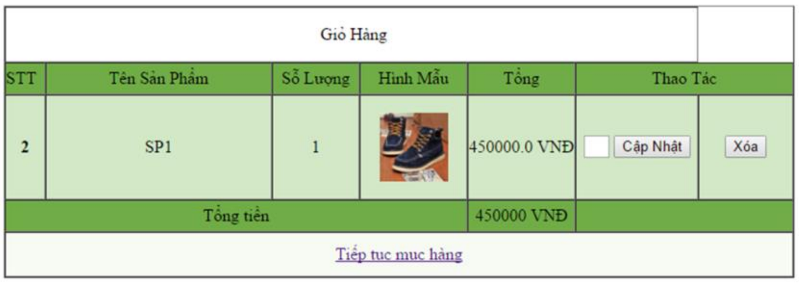


Displaying Cart Contents

```
<%
    Iterator<Product> it = cart.list().iterator();
    int i = 0;
    Product p;
    while (it.hasNext()) {
        p = it.next();
%>
<tr align="center" valign="middle"
    bgcolor="<%if (i++ % 2 == 0) {%>#D3E8C6 <%} else {%>#F7FBF4<%}%>">
    <td width="36" height="66"><p>
            <strong><%=i + 1%></strong>
        </p></td>
    <td width="210" height="66"><p><%=p.getName()%></p></td>
    <td width="80" height="66"><p><%=p.getCount()%></p></td>
    <td width="99" height="66"><p>
            <img src="<%=p.getUrl()%>" width="64" height="64">
        </p></td>
    <td width="113" height="66" align="center">
        <form name="formupdate" method="post" action="updateProduct">
            <input name="count" type="text" id="count" style="width: 20px">
            <input name="id" type="hidden" value="<%=p.getId()%>"> <input
                type="submit" value="Cập Nhật">
        </form>
    </td>
    <td width="87" height="66" align="center">
        <form name="formdel" method="post" action="delProduct">
            <input name="id" type="hidden" value="<%=p.getId()%>"> <input
                type="submit" value="Xóa">
```

26

Displaying Cart Contents



Giỏ Hàng						
STT	Tên Sản Phẩm	Số Lượng	Hình Mẫu	Tổng	Thao Tác	
2	SP1	1		450000.0 VNĐ	Cập Nhật	Xóa
Tổng tiền				450000 VNĐ		
Tiếp tục mua hàng						

27

Add product to Cart

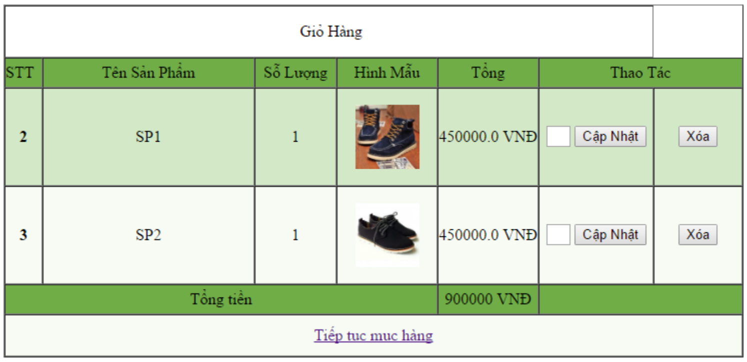
- Get current Cart object from session using getAttribute
  - If null, no Cart exists yet
  - In that case, construct one
- Get data from request and pass to Cart
  - Cart will construct and store a new Item
  - Will need to validate request first
- Will need to check whether item already in Cart
  - Need to avoid duplicate entries in Cart
  - Business model defines how handled
    - Error message
    - Change quantity,
    - Add to quantity, etc.

28

Add product to Cart

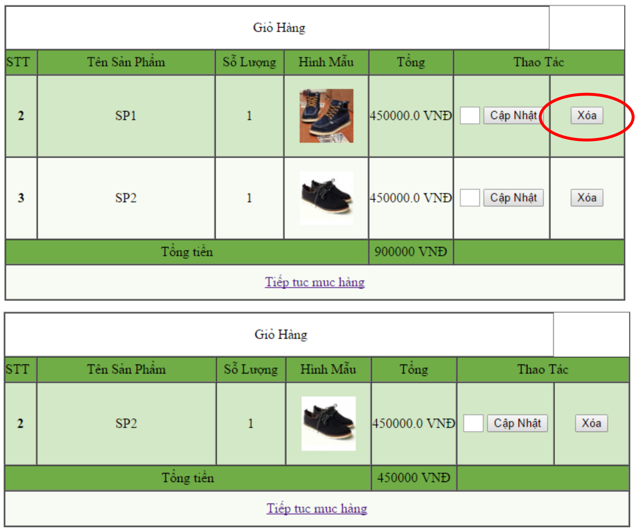
```
HttpSession session = request.getSession();        //get Session Object
int id = Integer.parseInt(request.getParameter( s: "id"));
                                                    // Get ID of Product
int quantity = Integer.parseInt(request.getParameter( s: "quantity"));
                                                    // Get quantity
Cart c = (Cart) session.getAttribute( s: "Cart");// Get Cart Object in Session
if (c == null) c = new Cart();
if (c.get(id) != null)
    c.put(id, quantity);
else {
    Product p = Product.find(id);                   //find product in DB
    if(p!=null)c.put(p);
}
session.setAttribute( s: "Cart", c);                //Put Cart to session again
```

Embedded Forms in Cart Pages



Giỏ Hàng						
STT	Tên Sản Phẩm	Số Lượng	Hình Mẫu	Tổng	Thao Tác	
2	SP1	1		450000.0 VNĐ	[ ] Cập Nhật	Xóa
3	SP2	1		450000.0 VNĐ	[ ] Cập Nhật	Xóa
Tổng tiền				900000 VNĐ		
Tiếp tục mua hàng						

Embedded Forms in Cart Pages



Giỏ Hàng						
STT	Tên Sản Phẩm	Số Lượng	Hình Mẫu	Tổng	Thao Tác	
2	SP1	1		450000.0 VNĐ	Cập Nhật	Xóa
3	SP2	1		450000.0 VNĐ	Cập Nhật	Xóa
Tổng tiền				900000 VNĐ		
Tiếp tục mua hàng						

Giỏ Hàng						
STT	Tên Sản Phẩm	Số Lượng	Hình Mẫu	Tổng	Thao Tác	
2	SP2	1		450000.0 VNĐ	Cập Nhật	Xóa
Tổng tiền				450000 VNĐ		
Tiếp tục mua hàng						

31

Simple Removal Servlet

- Delete servlet

```
HttpSession session = request.getSession(); //Get Session Object
int id = Integer.parseInt(request.getParameter( s: "id"));
                                            //Get ID from jsp
Cart c=(Cart)session.getAttribute( s: "Cart");//Get Cart Object
if(c==null)c= new Cart();                     //Check Cart is null
c.remove(id);                                 //Remove product
session.setAttribute( s: "Cart",c);           //Put Cart to session
```

- Remove method in cart object

```
public boolean remove(int id) {
    return data.remove(id) == null;
}
```

32

Hidden Form Elements

- Must submit product code for remove to work
- Product code not displayed on page inside a form element
  - Common for most ecommerce pages

Can use hidden form element

- Not shown by browser
- Can store product code or other information that we need to send to server

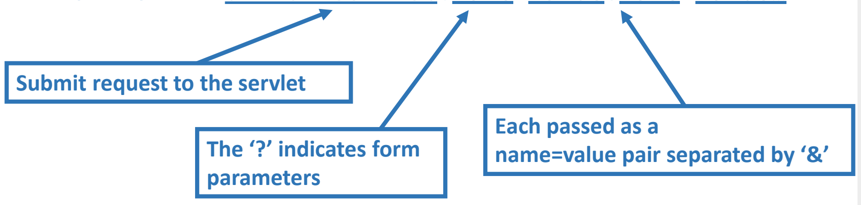
```
<input type="hidden" name="parameter name"
                     value="<%= product code %>"
```

Passing Data using Links

- Can append "form data" directly to URL in link
  - Result similar to "get" method in form

- Syntax:

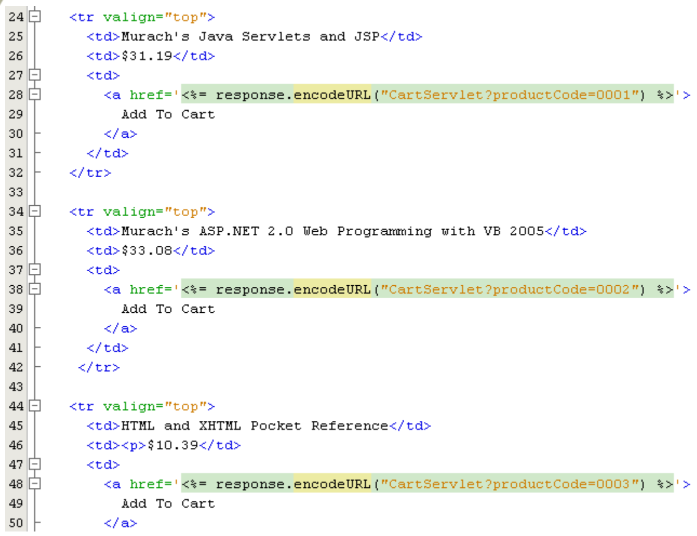
```
<a href = "url of servlet?name=value&name=value…">
```


Submit request to the servlet
The '?' indicates form parameters
Each passed as a name=value pair separated by '&'

- Note: will need to use `response.encodeURL` to insure this works if cookies not enabled

```
<a href = "<%=response.encodeURL(
                 'servleturl?name=value&…')%>">
```

Passing Data using Links



```
<tr valign="top">
  <td>Murach's Java Servlets and JSP</td>
  <td>$31.19</td>
  <td>
    <a href='<%= response.encodeURL("CartServlet?productCode=0001") %>'>
      Add To Cart
    </a>
  </td>
</tr>

<tr valign="top">
  <td>Murach's ASP.NET 2.0 Web Programming with VB 2005</td>
  <td>$33.08</td>
  <td>
    <a href='<%= response.encodeURL("CartServlet?productCode=0002") %>'>
      Add To Cart
    </a>
  </td>
 </tr>

<tr valign="top">
  <td>HTML and XHTML Pocket Reference</td>
  <td><p>$10.39</td>
  <td>
    <a href='<%= response.encodeURL("CartServlet?productCode=0003") %>'>
      Add To Cart
    </a>
```

35

DEMO

36

Q & A