# Using JavaScript for Client-Side Behavior

## Internet Applications, ID1354

# Contents

# Section

- The Document Object Model, DOM
- The Browser Object Model, BOM
- The jQuery JavaScript Library
- The Knockout JavaScript Framework

# The Document Object Model, DOM

- The W3C Document Object Model, DOM, is an API that allows programs to access and update document content.

# The Document Object Model, DOM

- The W3C Document Object Model, DOM, is an API that allows programs to access and update document content.
- Defines objects representing HTML elements, methods to access HTML elements, and events generated by HTML elements.

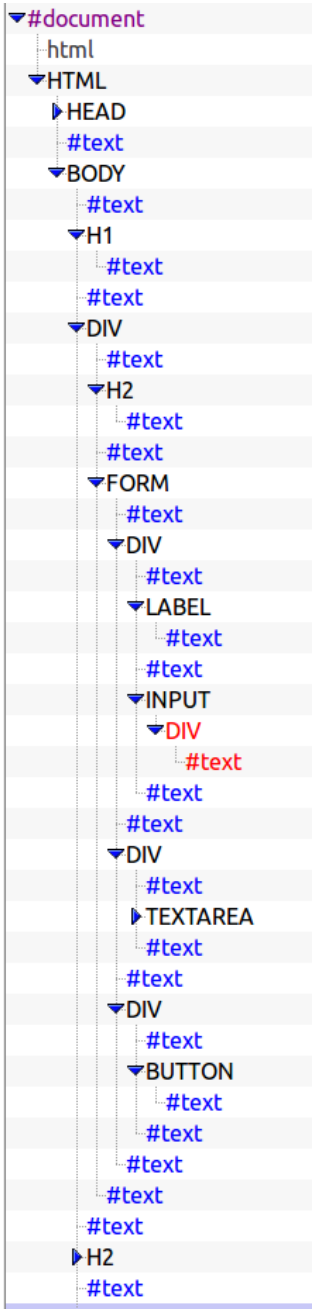
# The Document Object Model, DOM

- The W3C Document Object Model, DOM, is an API that allows programs to access and update document content.
- Defines objects representing HTML elements, methods to access HTML elements, and events generated by HTML elements.
- The best that can be said about browser support is that it varies.

# The Document Object Model, DOM

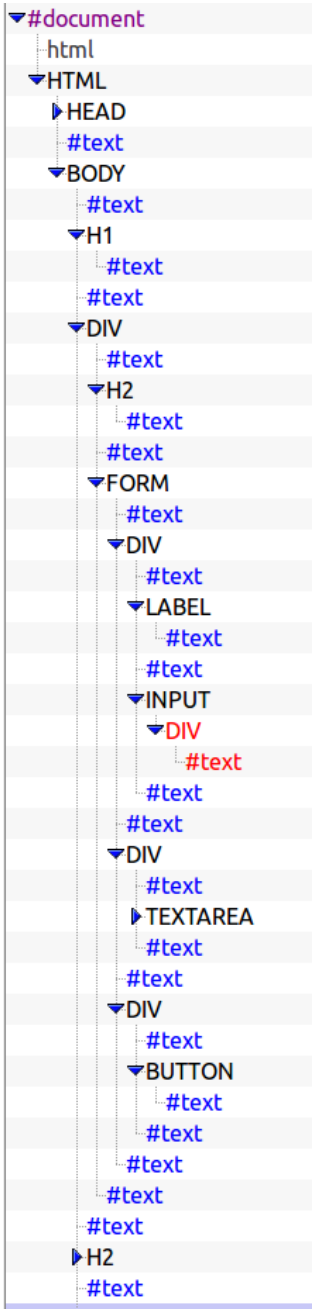
- The W3C Document Object Model, DOM, is an API that allows programs to access and update document content.
- Defines objects representing HTML elements, methods to access HTML elements, and events generated by HTML elements.
- The best that can be said about browser support is that it varies.
  - Try the features you want to use in all relevant browsers, check **`caniuse.com`**, etc

# The DOM Tree




- The DOM objects are organized in a tree.

# The DOM Tree




- The DOM objects are organized in a tree.
- The picture to the left is a part of the DOM tree for the course's chat program.

# The DOM API

- All HTML elements are represented by objects.

# The DOM API

- All HTML elements are represented by objects.
- The HTML objects have properties you can get or set, to read or update the objects.

# The DOM API

- All HTML elements are represented by objects.
- The HTML objects have properties you can get or set, to read or update the objects.
- The HTML objects have methods, for example for adding and deleting elements.

# The DOM API

- All HTML elements are represented by objects.
- The HTML objects have properties you can get or set, to read or update the objects.
- The HTML objects have methods, for example for adding and deleting elements.
- An example is the statement

```
document.getElementById("demo").innerHTML =
    "Hello World!";
```

that uses the method **`getElementById`** to find the HTML object for the element with id **`demo`**, and sets the HTML of that object to **`"Hello World!"`**, using the **`innerHTML`** property.

# The `document` Object

- The implicit object **`document`** represents the entire web page and is the entry point to the DOM API.

# The `document` Object

- The implicit object **`document`** represents the entire web page and is the entry point to the DOM API.
- Sample methods in **`document`** are

  Find HTML elements **`getElementById`**, **`getElementsByTagName`**, **`getElementsByClassName`**

# The `document` Object

- The implicit object **`document`** represents the entire web page and is the entry point to the DOM API.
- Sample methods in **`document`** are

  Find HTML elements `getElementById`, `getElementsByTagName`, `getElementsByClassName`

  Properties of HTML elements **`innerHTML`**, **`attribute`**

# The `document` Object

- The implicit object **`document`** represents the entire web page and is the entry point to the DOM API.
- Sample methods in **`document`** are

  Find HTML elements **`getElementById`**, **`getElementsByTagName`**, **`getElementsByClassName`**

  Properties of HTML elements **`innerHTML`**, **`attribute`**

  Add or delete elements **`createElement`**, **`removeChild`**, **`appendChild`**

# The `document` Object

- The implicit object **`document`** represents the entire web page and is the entry point to the DOM API.
- Sample methods in **`document`** are

  Find HTML elements **`getElementById`**, **`getElementsByTagName`**, **`getElementsByClassName`**

  Properties of HTML elements **`innerHTML`**, **`attribute`**

  Add or delete elements **`createElement`**, **`removeChild`**, **`appendChild`**

  Collections of HTML elements **`cookie`**, **`URL`**, **`elements`**, **`forms`**

# Change CSS Rules

- To change CSS rules use the following type of statement:

```
document.getElementById(id).style.<property>
    = <new style>
```

# Change CSS Rules



- To change CSS rules use the following type of statement:

```
document.getElementById(id).style.<property>
    = <new style>
```

- For example, the statement

```
document.getElementById("p2").style.color =
    "blue";
```

changes the font color of element with id **p2** to blue.

# Events

- The DOM defines many events, for example **`onClick`**, which is fired by an element when the user clicks on it.

# Events

- The DOM defines many events, for example `onClick`, which is fired by an element when the user clicks on it.
- To react to an event, add JavaScript code to the corresponding attribute of the event's source element.

# Events

- The DOM defines many events, for example **`onClick`**, which is fired by an element when the user clicks on it.
- To react to an event, add JavaScript code to the corresponding attribute of the event's source element.
- For example, to change text in the paragraph when the user clicks it:

**HTML:**

```
<p onclick="clicked(this)">Click Me!</p>
```

**JavaScript:**

```
function clicked(source) {
    source.innerHTML = "You clicked!";
}
```

# Examples of Events

Mouse events **`onclick`**, **`ondblclick`**, **`onmousedown`**, **`onmouseover`**

# Examples of Events

Mouse events **`onclick`**, **`ondblclick`**, **`onmousedown`**, **`onmouseover`**

Keyboard events **`onkeydown`**, **`onkeypress`**, **`onkeyup`**, fired in that order.

# Examples of Events

Mouse events **onclick**, **ondblclick**, **onmousedown**, **onmouseover**

Keyboard events **onkeydown**, **onkeypress**, **onkeyup**, fired in that order.

Object events **onload**, **onunload**

# Examples of Events

Mouse events **onclick**, **ondblclick**, **onmousedown**, **onmouseover**

Keyboard events **onkeydown**, **onkeypress**, **onkeyup**, fired in that order.

Object events **onload**, **onunload**

Form events **onchange**, **onselect**

# Event Listeners

- **addEventListener()** attaches an event listener to the specified element.

```
<button id="myBtn">Try it</button>
<p id="demo"></p>
```

```
document.getElementById("myBtn").
   addEventListener("click", displayDate);

function displayDate()  {
  document.getElementById("demo").
     innerHTML = Date();
}
```

# Event Listeners

- **addEventListener()** attaches an event listener to the specified element.

```
<button id="myBtn">Try it</button>
<p id="demo"></p>
```

```
document.getElementById("myBtn").
   addEventListener("click", displayDate);

function displayDate()  {
  document.getElementById("demo").
     innerHTML = Date();
}
```

- Multiple event listeners, even of the same type, can be attached to the same element.

# Event Listeners

- **addEventListener()** attaches an event listener to the specified element.

```
<button id="myBtn">Try it</button>
<p id="demo"></p>
```

```
document.getElementById("myBtn").
   addEventListener("click", displayDate);

function displayDate()  {
  document.getElementById("demo").
     innerHTML = Date();
}
```

- Multiple event listeners, even of the same type, can be attached to the same element.
- Event listeners is preferred over **onEvent** attributes since it separates JavaScript from HTML, thereby increasing cohesion.

# Passing Parameters to Event Listeners



- The following code illustrates how to pass parameters to event listeners.

```
<button id="myBtn">Try it</button>
<p id="demo"></p>
```

```
document.getElementById("myBtn").
   addEventListener("click", function() {
      showLabel(this);
   });

function showLabel(source) {
  document.getElementById("demo").innerHTML
     = source.innerHTML;
}
```

# Event Bubbling

- When an element fires an event, also the event handlers of its parents are invoked.

# Event Bubbling

- When an element fires an event, also the event handlers of its parents are invoked.
- An event first triggers the deepest possible element, then its parents in nesting order.

# Event Bubbling Example

- **HTML:**

```
<div onclick="show(1)">1
    <div onclick="show(2)">2
        <div onclick="show(3)">3
            <p id="event-log"></p>
        </div>
    </div>
</div>
```

**JavaScript:**

```
function show(sourceNo) {
   var curr = document.
      getElementById("event-log").innerHTML;
   document.getElementById("event-log").
      innerHTML = curr + " " + sourceNo;
}
```

# Event Bubbling Example

- **HTML:**

```
<div onclick="show(1)">1
    <div onclick="show(2)">2
        <div onclick="show(3)">3
            <p id="event-log"></p>
        </div>
    </div>
</div>
```

**JavaScript:**

```
function show(sourceNo) {
   var curr = document.
      getElementById("event-log").innerHTML;
   document.getElementById("event-log").
      innerHTML = curr + " " + sourceNo;
}
```

- When clicking the innermost div (number 3), the output is `3 2 1`

# Stopping the Bubbling

- Bubbling is prevented by calling **`stopPropagation()`** on the **`event`** object.

# Stopping the Bubbling

- Bubbling is prevented by calling `stopPropagation()` on the `event` object.

```
<div onclick="show(1)">1
    <div onclick="show(2)">2
        <div onclick='show(3, event)'>3
            <p id="log"></p>
        </div>
    </div>
</div>
```

```
function show(sourceNo, event) {
   var curr = document.
      getElementById("log").innerHTML;
   document.getElementById("log").
      innerHTML = curr + " " + sourceNo;
   event.stopPropagation();
}
```

# Stopping the Bubbling

- Bubbling is prevented by calling `stopPropagation()` on the `event` object.

```
<div onclick="show(1)">1
    <div onclick="show(2)">2
        <div onclick='show(3, event)'>3
            <p id="log"></p>
        </div>
    </div>
</div>
```

```
function show(sourceNo, event) {
   var curr = document.
      getElementById("log").innerHTML;
   document.getElementById("log").
      innerHTML = curr + " " + sourceNo;
   event.stopPropagation();
}
```

- When clicking the innermost div, output is now **3**

# Event Capturing

- Before bubbling, the event goes the other way, from outermost to innermost element. This is called capturing.

# Event Capturing

- Before bubbling, the event goes the other way, from outermost to innermost element. This is called capturing.
- The capturing phase is ignored by all **`onEvent`** attributes and event listeners, except listeners with the **`useCapture`** argument set to true:

```
document.getElementById("myId").
    addEventListener("click", handler, true);
```

# Navigating the DOM Tree

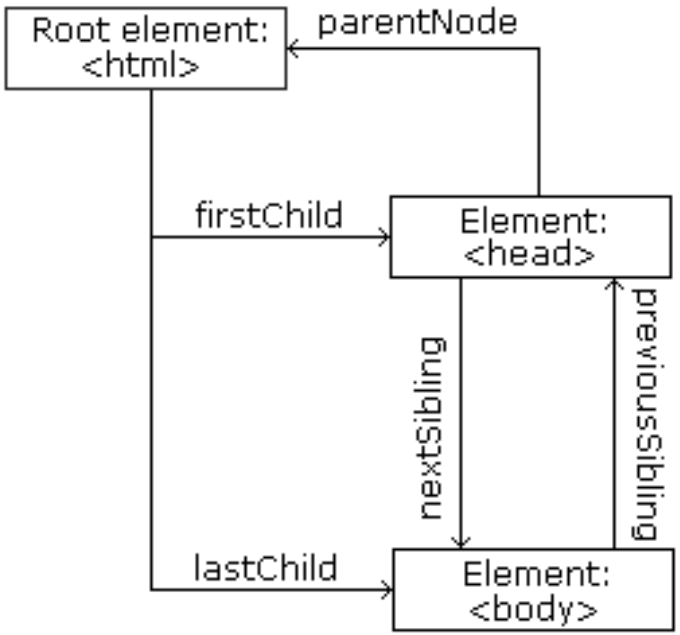



Image from
http://www.w3schools.com/js/js_htmldom_navigation.asp

- The image to the left illustrates parent, child and sibling relationships between nodes in the DOM tree.

# Navigating the DOM Tree

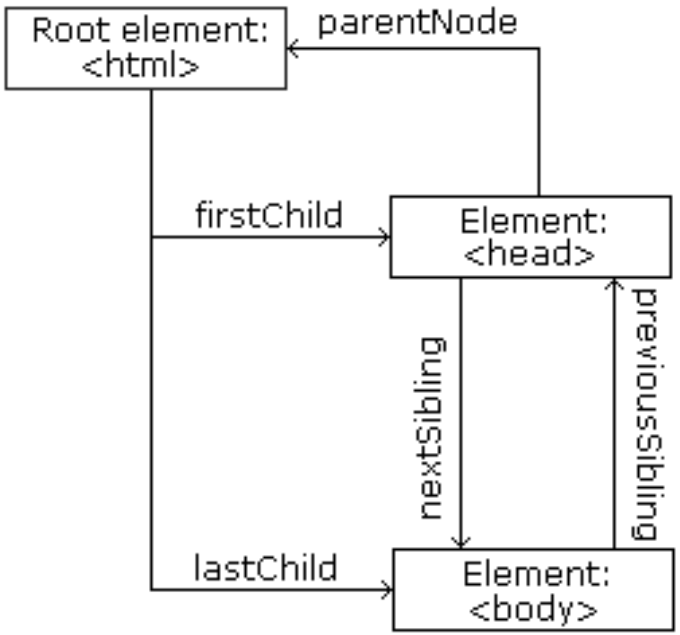



Image from
http://www.w3schools.com/js/js_htmldom_navigation.asp

- The image to the left illustrates parent, child and sibling relationships between nodes in the DOM tree.
- The DOM tree can be navigated with the node properties **`parentNode`**, **`childNodes`**, **`firstChild`**, **`lastChild`**, **`nextSibling`**, and **`previousSibling`**

# Navigating the DOM Tree (Cont'd)

- Note that, in the code below, the **<p>** node contains a child text node with the value **the text**

```
<p>the text</p>
```

# Navigating the DOM Tree (Cont'd)

- Note that, in the code below, the **<p>** node contains a child text node with the value **the text**

```
<p>the text</p>
```

- Text content can be accessed with the **innerHTML** and **nodeValue** properties.

```
<p id="demo">text content</p>
```

Using **innerHTML**:

```
var text = document.getElementById("demo").
    innerHTML;
```

Using **nodeValue**:

```
var text = document.getElementById("demo").
childNodes[0].nodeValue;
```

# Adding Elements

- To add a new element, first create it, then insert it in the DOM tree.

```
<div id="target"></div>
```

```
var elem = document.createElement("p");
var text =
   document.createTextNode("new text");
elem.appendChild(text);
document.getElementById("target").
   appendChild(elem);
```

# Removing Elements



- To remove an element, use the **`removeChild`** method.

```
<div id="parent">
  <p>To be removed</p>
</div>
```

```
var parent =
   document.getElementById("parent");
var child =
   parent.getElementsByTagName("p")[0];
parent.removeChild(child);
```

# Section

- The Document Object Model, DOM
- The Browser Object Model, BOM
- The jQuery JavaScript Library
- The Knockout JavaScript Framework

# The Browser Object Model, BOM

- While the DOM provides an API for accessing the current document, the Browser Object Model, BOM, provides an API that gives access to the browser.

# The Browser Object Model, BOM

- While the DOM provides an API for accessing the current document, the Browser Object Model, BOM, provides an API that gives access to the browser.
- The BOM is not standardized, but more or less the same methods are implemented in all modern browsers.

# The Browser Object Model, BOM

- While the DOM provides an API for accessing the current document, the Browser Object Model, BOM, provides an API that gives access to the browser.
- The BOM is not standardized, but more or less the same methods are implemented in all modern browsers.
- The following slides contain a short overview of major objects and methods, to give an idea of what can be done with the BOM.

# BOM Objects

- The **`window`** object has:
  - Properties for height and width of the browser window.
  - Methods to open, close, move and resize the browser window.
  - Methods to execute some code at specified time-intervals.

# BOM Objects

- The **`window`** object has:
  - Properties for height and width of the browser window.
  - Methods to open, close, move and resize the browser window.
  - Methods to execute some code at specified time-intervals.
- The **`location`** object has:
  - Properties that gives information about the current URL.
  - The **`assign`** method that loads a new document.

# BOM Objects (Cont'd)

- The **`navigator`** object can give information about browser type and browser features.

# BOM Objects (Cont'd)

- The **`navigator`** object can give information about browser type and browser features.
- The **`screen`** object has properties for height, width and pixel depth of the user's screen.

# BOM Objects (Cont'd)

- The **`navigator`** object can give information about browser type and browser features.
- The **`screen`** object has properties for height, width and pixel depth of the user's screen.
- The **`document`** object has the **`cookie`** property, which is used to get and set cookies.

# Section

- The Document Object Model, DOM
- The Browser Object Model, BOM
- The jQuery JavaScript Library
- The Knockout JavaScript Framework

# The jQuery Library

- jQuery provides an API that simplifies many common JavaScript tasks, like DOM manipulation, CSS manipulation, event handling, effects and animation.

# The jQuery Library

- jQuery provides an API that simplifies many common JavaScript tasks, like DOM manipulation, CSS manipulation, event handling, effects and animation.
- There are many jQuery plugins that provide more features.

# The jQuery Library

- jQuery provides an API that simplifies many common JavaScript tasks, like DOM manipulation, CSS manipulation, event handling, effects and animation.
- There are many jQuery plugins that provide more features.
- jQuery hides cross-browser issues, all jQuery code will work the same way in all browsers supporting jQuery.

# The jQuery Library

- jQuery provides an API that simplifies many common JavaScript tasks, like DOM manipulation, CSS manipulation, event handling, effects and animation.
- There are many jQuery plugins that provide more features.
- jQuery hides cross-browser issues, all jQuery code will work the same way in all browsers supporting jQuery.
- jQuery is very commonly used.

# Installing jQuery

- jQuery is a JavaScript file, to use it you just have to provide a link to that file.

# Installing jQuery

- jQuery is a JavaScript file, to use it you just have to provide a link to that file.
- The jQuery library file comes in two versions:
  - A development version, which is uncompressed and therefore readable.
  - A live website version, which has been minified and compressed and therefore is not readable. Instead it is shorter and thereby faster to download.

# Installing jQuery (Cont'd)

- Either you download it from **jquery.com** and place it on your server, or you provide a link to a Content Delivery Network, CDN, as follows:

```
<script src="https://cdnjs.cloudflare.com/
ajax/libs/jquery/1.10.2/jquery.min.js">
</script>
```

# Installing jQuery (Cont'd)

- Either you download it from **jquery.com** and place it on your server, or you provide a link to a Content Delivery Network, CDN, as follows:

```
<script src="https://cdnjs.cloudflare.com/
ajax/libs/jquery/1.10.2/jquery.min.js">
</script>
```

- Using a CDN is normally faster, since:

# Installing jQuery (Cont'd)

- Either you download it from **jquery.com** and place it on your server, or you provide a link to a Content Delivery Network, CDN, as follows:

```
<script src="https://cdnjs.cloudflare.com/
ajax/libs/jquery/1.10.2/jquery.min.js">
</script>
```

- Using a CDN is normally faster, since:
  - The file is delivered from the CDNs server closest to the user.

# Installing jQuery (Cont'd)

- Either you download it from **jquery.com** and place it on your server, or you provide a link to a Content Delivery Network, CDN, as follows:

```
<script src="https://cdnjs.cloudflare.com/
ajax/libs/jquery/1.10.2/jquery.min.js">
</script>
```

- Using a CDN is normally faster, since:
  - The file is delivered from the CDNs server closest to the user.
  - Many users already have downloaded jQuery from the CDN when visiting another site. As a result, it is loaded from browser cache.

# The jQuery Function

- Very central to jQuery is the jQuery function, which has two names, **`jQuery`** and the commonly used **`$`**.

# The jQuery Function

- Very central to jQuery is the jQuery function, which has two names, `jQuery` and the commonly used `$`.
- Remember that `$` is a perfectly legal JavaScript identifier, there is nothing magic about that name.

# The jQuery Function

- Very central to jQuery is the jQuery function, which has two names, `jQuery` and the commonly used `$`.
- Remember that `$` is a perfectly legal JavaScript identifier, there is nothing magic about that name.
- The jQuery function normally takes one parameter, which is either a CSS selector or a reference to an object in the document, and returns a jQuery object wrapping all HTML element(s) corresponding to the search criteria.

```
$(document)   //The document object.
$(this)       //The current element.
$("#someId") //All elements with id "someId"
$(div)        //All div elements.
```

# The jQuery Function

- Very central to jQuery is the jQuery function, which has two names, **`jQuery`** and the commonly used **`$`**.
- Remember that **`$`** is a perfectly legal JavaScript identifier, there is nothing magic about that name.
- The jQuery function normally takes one parameter, which is either a CSS selector or a reference to an object in the document, and returns a jQuery object wrapping all HTML element(s) corresponding to the search criteria.

```
$(document)   //The document object.
$(this)       //The current element.
$("#someId") //All elements with id "someId"
$(div)        //All div elements.
```

- Any CSS selector can be used as search criteria.

# The jQuery Object

- The jQuery object also has two names, **`jQuery`** and the commonly used **`$`**.

# The jQuery Object

- The jQuery object also has two names, **`jQuery`** and the commonly used **`$`**.
- The jQuery object contains many methods that operate on the wrapped HTML element. For example the **`html`** method that gets or sets the HTML content of the wrapped element:

```
/* Store the HTML of the element with
id "someId" in the variable content. */
var content = $("#someId").html();

/* Set the HTML of the element with
id "someId" to "content<br/>". */
$("#someId").html(content + "<br/>");
```

# The jQuery Object (Cont'd)

- The jQuery object supports array subscripting via brackets:

```
$("h1")[0]; //The first h1 element.
```

# The jQuery Object (Cont'd)



- The jQuery object supports array subscripting via brackets:

```
$("h1")[0]; //The first h1 element.
```

- The jQuery object also has utility methods that are not related to a HTML element:

```
// Returns the string "extra whitespace"
$.trim( "    extra whitespace    " );
```

# Event Handlers

- In jQuery, an event handling function is passed as argument to a method with the event name in the jQuery object wrapping the desired event source.

# Event Handlers

- In jQuery, an event handling function is passed as argument to a method with the event name in the jQuery object wrapping the desired event source.
- The following code adds an event handler to all **<p>** elements. The event handler will change the paragraph's text to **You clicked!** when the user clicks it.

```
$("p").click(function(){
  $(this).html("You clicked!");
});
```

# The Document Ready Event

- jQuery defines the document ready event, which is fired when the DOM has been constructed.

# The Document Ready Event

- jQuery defines the document ready event, which is fired when the DOM has been constructed.
- It is usually best to wait for this event before running JavaScript code, to avoid operating on elements that have not been defined.

# The Document Ready Event

- jQuery defines the document ready event, which is fired when the DOM has been constructed.
- It is usually best to wait for this event before running JavaScript code, to avoid operating on elements that have not been defined.
- It is normally not necessary to wait for the JavaScript load event, which fires when everything, including images, is loaded and rendered.

# The Document Ready Event

- jQuery defines the document ready event, which is fired when the DOM has been constructed.
- It is usually best to wait for this event before running JavaScript code, to avoid operating on elements that have not been defined.
- It is normally not necessary to wait for the JavaScript load event, which fires when everything, including images, is loaded and rendered.
- Therefore, unless otherwise needed, jQuery code is written like this:

```
$(document).ready(function(){
    // jQuery code here...
});
```

# Effects

- There are lots of effects provided by jQuery, here are some examples.

# Effects

- There are lots of effects provided by jQuery, here are some examples.
  - The **`hide`** and **`show`** methods can hide/show elements. It is also possible to specify the speed of the (dis)appearance.

# Effects

- There are lots of effects provided by jQuery, here are some examples.
  - The **`hide`** and **`show`** methods can hide/show elements. It is also possible to specify the speed of the (dis)appearance.
  - Various **`fade`** methods causes an element to fade in/out.

# Effects

- There are lots of effects provided by jQuery, here are some examples.
  - The **`hide`** and **`show`** methods can hide/show elements. It is also possible to specify the speed of the (dis)appearance.
  - Various **`fade`** methods causes an element to fade in/out.
  - Various **`slide`** methods causes an element to slide up/down.

# Effects

- There are lots of effects provided by jQuery, here are some examples.
  - The **`hide`** and **`show`** methods can hide/show elements. It is also possible to specify the speed of the (dis)appearance.
  - Various **`fade`** methods causes an element to fade in/out.
  - Various **`slide`** methods causes an element to slide up/down.
  - The **`animate`** method is used to create custom animations.

# Effects (Cont'd)



- Effects can have callback functions that are executed when the effect is done.

```
$("button").click(function(){
  $("p").hide("slow",function(){
    alert("The paragraph is now hidden");
  });
});
```

# jQuery Method Chaining

- Many of the element manipulation methods of the jQuery object return the jQuery object itself.

# jQuery Method Chaining



- Many of the element manipulation methods of the jQuery object return the jQuery object itself.
- This means it is possible to create chains of such methods.

```
$("button").click(function(){
  $("#p1").css("color","blue")
    .slideUp(3000)
    .slideDown(2000);
});
```

# Element Content and Element Attributes

- The **`text`** method is used to access text content of an HTML element, the **`html`** method is used for text content with HTML tags, the **`val`** method is used for form field values, and **`attr`** is used for an element's attributes.

# Element Content and Element Attributes

- The **`text`** method is used to access text content of an HTML element, the **`html`** method is used for text content with HTML tags, the **`val`** method is used for form field values, and **`attr`** is used for an element's attributes.
- If called without arguments, these methods return the current value. If called with arguments they set a new value.

```
$("#btn").click(function(){
  var current = $("#test").html());
});
```

```
$("#btn").click(function(){
  $("#textField").val("New value");
});
```

# Element Content and Element Attributes (Cont'd)

- The **`text`**, **`html`**, **`val`**, and **`attr`** methods can have callback functions.

# Element Content and Element Attributes (Cont'd)

- The **`text`**, **`html`**, **`val`**, and **`attr`** methods can have callback functions.
- The callback function takes two parameters, the index of the current element in the list of elements selected and the original value.

# Element Content and Element Attributes (Cont'd)



- The **`text`**, **`html`**, **`val`**, and **`attr`** methods can have callback functions.
- The callback function takes two parameters, the index of the current element in the list of elements selected and the original value.
- The return value of the callback function becomes the new text of the element.

```
$("#btn").click(function(){
  $("p").text(function(i,oldText){
    return "Old text: " + oldText +
           ", index: " + i;
  });
});
```

# To Add or Remove Elements

- The **append**, **prepend**, **before**, and **after** methods are used to add elements.

```
// Append a list item to the ordered list
// with id "someList".
$("#someList").
    append("<li>Appended item</li>");
```

# To Add or Remove Elements

- The **append**, **prepend**, **before**, and **after** methods are used to add elements.

```
// Append a list item to the ordered list
// with id "someList".
$("#someList").
    append("<li>Appended item</li>");
```

- The **remove** and **empty** methods are used to remove elements.

```
// Remove the element with id "#menu".
$("#menu").remove();
```

# CSS Rules

- When passed a CSS property name, the **`css`** method returns the value of that property.

```
$("body").css("background-color"));
```

# CSS Rules

- When passed a CSS property name, the **`css`** method returns the value of that property.

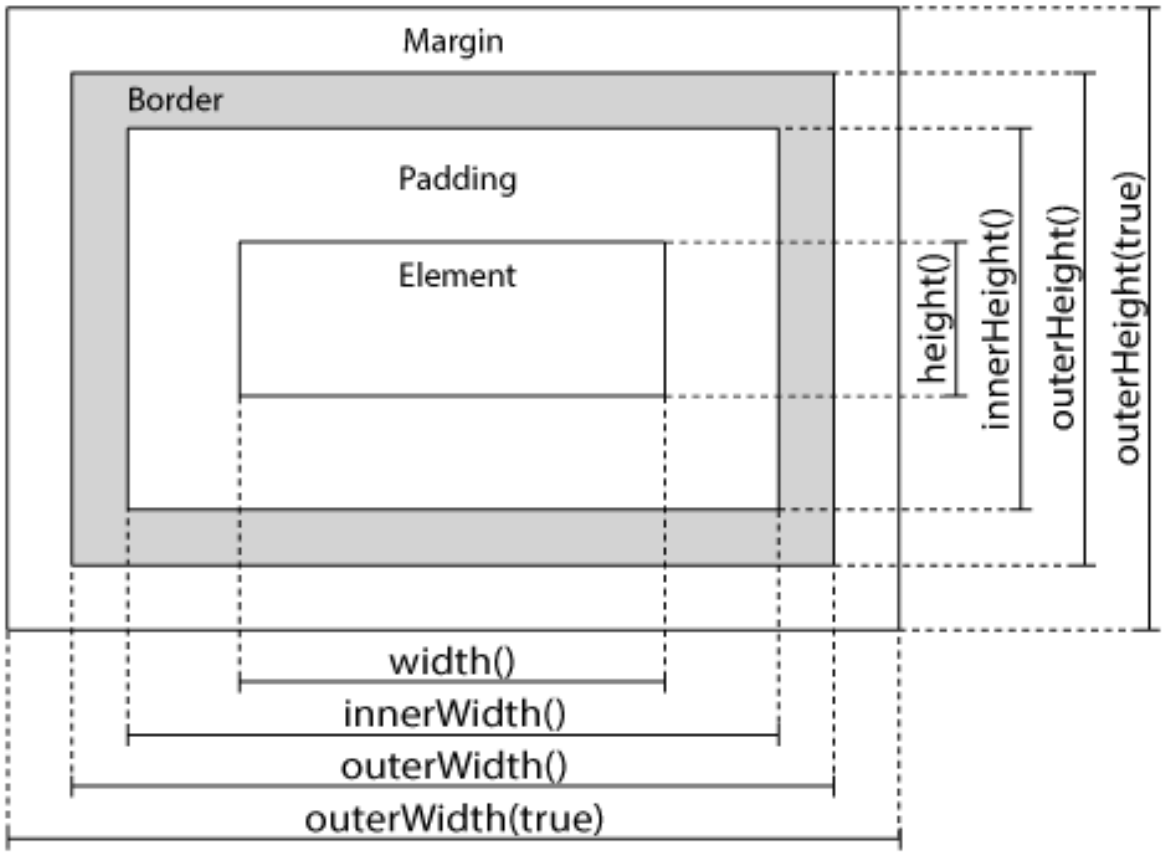
```
$("body").css("background-color"));
```

- When passed one or more property:value pairs, those rules are set for the specified element(s).

```
$("body").css("background-color","yellow");
```

# Dimensions

The following image, taken from **`http://www.w3schools.com/jquery/jquery_dimensions.asp`**, illustrates the methods used to set or get element dimensions.



# Traversing the DOM Tree

Here are samples of jQuery methods used to traverse the DOM tree.

# Traversing the DOM Tree

Here are samples of jQuery methods used to traverse the DOM tree.

**`parent`** Returns the parent on the nearest higher level.

# Traversing the DOM Tree

Here are samples of jQuery methods used to traverse the DOM tree.

`parent` Returns the parent on the nearest higher level.

`parents` Returns all parents all the way up to the html element.

# Traversing the DOM Tree

Here are samples of jQuery methods used to traverse the DOM tree.

`parent` Returns the parent on the nearest higher level.

`parents` Returns all parents all the way up to the html element.

`children` Returns all children on the nearest lower level.

# Traversing the DOM Tree

Here are samples of jQuery methods used to traverse the DOM tree.

| | |
|---|---|
| `parent` | Returns the parent on the nearest higher level. |
| `parents` | Returns all parents all the way up to the html element. |
| `children` | Returns all children on the nearest lower level. |
| `find` | Returns all descendants on all lower levels. |

# Traversing the DOM Tree

Here are samples of jQuery methods used to traverse the DOM tree.

| | |
|---|---|
| `parent` | Returns the parent on the nearest higher level. |
| `parents` | Returns all parents all the way up to the html element. |
| `children` | Returns all children on the nearest lower level. |
| `find` | Returns all descendants on all lower levels. |
| `siblings` | Returns all siblings. |

# Traversing the DOM Tree

Here are samples of jQuery methods used to traverse the DOM tree.

**`parent`** Returns the parent on the nearest higher level.

**`parents`** Returns all parents all the way up to the html element.

**`children`** Returns all children on the nearest lower level.

**`find`** Returns all descendants on all lower levels.

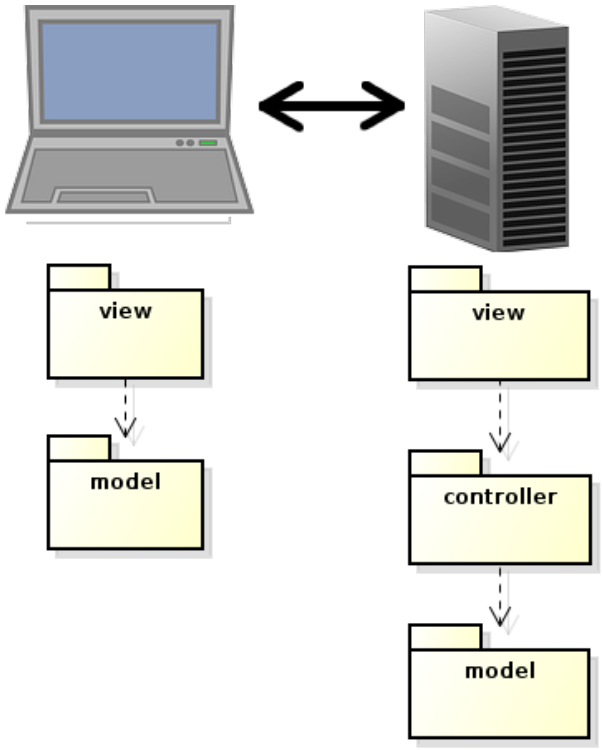
**`siblings`** Returns all siblings.

filtering The **`first`**, **`last`**, **`eq`**, and **`filter`** methods can be used to filter the search results of the methods above.

# Section

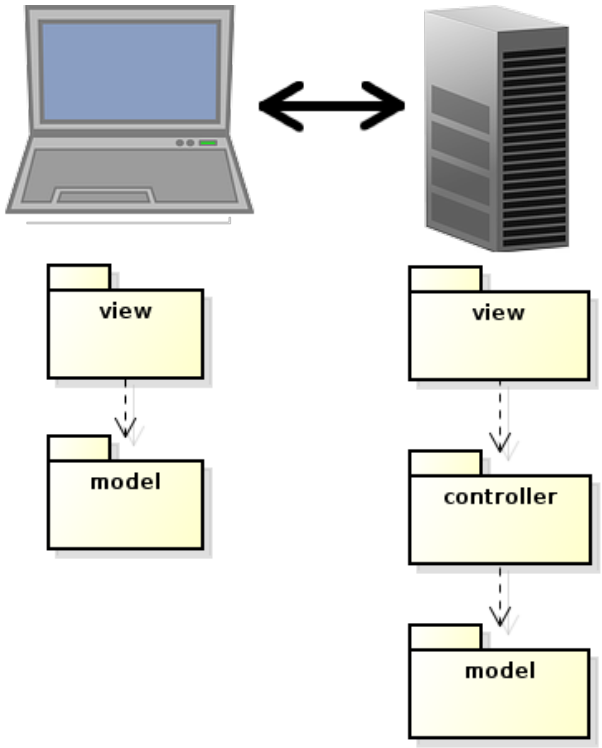
- The Document Object Model, DOM
- The Browser Object Model, BOM
- The jQuery JavaScript Library
- The Knockout JavaScript Framework

# Reminder: The MVVM Pattern



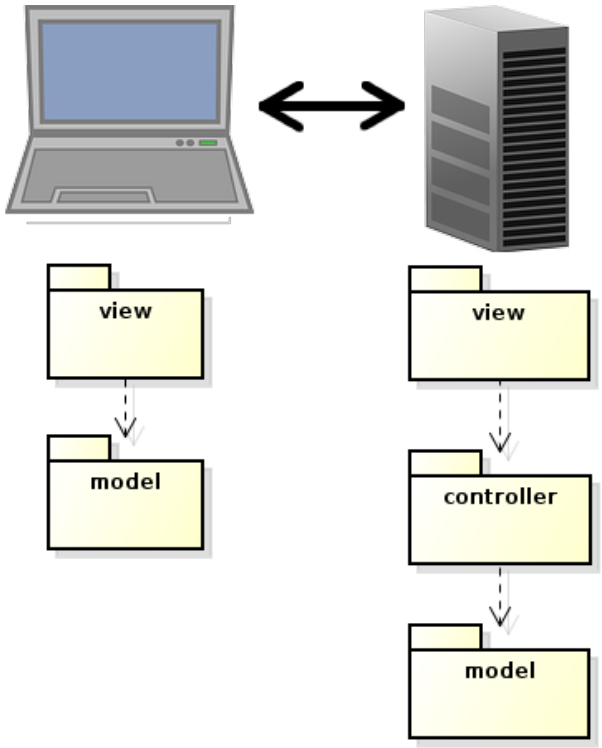

- The MVVM pattern intoduces a client-side model which reflects the server-side model and is responsible for notifying the view of updates.

# Reminder: The MVVM Pattern




- The MVVM pattern intoduces a client-side model which reflects the server-side model and is responsible for notifying the view of updates.
- The server-side view is relieved from creating HTML code.

# Reminder: The MVVM Pattern




- The MVVM pattern intoduces a client-side model which reflects the server-side model and is responsible for notifying the view of updates.
- The server-side view is relieved from creating HTML code.
- Also, network communication is reduced, since only model updates are fetched from the server. There is no need to reload the entire web page at each user action.

# MVVM JavaScript Frameworks

- The code for implementing the Observer pattern to have the view reflect changes in the viewmodel will be the same for more or less all applications.

# MVVM JavaScript Frameworks

- The code for implementing the Observer pattern to have the view reflect changes in the viewmodel will be the same for more or less all applications.
- Also the code for viewmodel-to-server communication will be quite similar in all applications.

# MVVM JavaScript Frameworks

- The code for implementing the Observer pattern to have the view reflect changes in the viewmodel will be the same for more or less all applications.
- Also the code for viewmodel-to-server communication will be quite similar in all applications.
- This calls for a client-side framework, since we do not want to rewrite the same code for each new application. This is exactly the purpose of Knockout.

# MVVM JavaScript Frameworks

- The code for implementing the Observer pattern to have the view reflect changes in the viewmodel will be the same for more or less all applications.
- Also the code for viewmodel-to-server communication will be quite similar in all applications.
- This calls for a client-side framework, since we do not want to rewrite the same code for each new application. This is exactly the purpose of Knockout.
- There are also many alternative frameworks, perhaps the most commonly used is Backbone. Backbone is more used and more powerful than Knockout, but too complicated for this course.

# The Knockout JavaScript Framework

- Like jQuery, Knockout is a JavaScript file the can be linked from a CDN, for example:

```
<script src="https://cdnjs.cloudflare.com/
  ajax/libs/knockout/3.1.0/knockout-min.js">
</script>
```

# The Knockout JavaScript Framework

- Like jQuery, Knockout is a JavaScript file the can be linked from a CDN, for example:

```
<script src="https://cdnjs.cloudflare.com/
  ajax/libs/knockout/3.1.0/knockout-min.js">
</script>
```

- Knockout should be used together with jQuery, which handles low-level DOM interaction.

# The Knockout JavaScript Framework

- Like jQuery, Knockout is a JavaScript file the can be linked from a CDN, for example:

```
<script src="https://cdnjs.cloudflare.com/
  ajax/libs/knockout/3.1.0/knockout-min.js">
</script>
```

- Knockout should be used together with jQuery, which handles low-level DOM interaction.
- Knockout implements the MVVC pattern, by managing View-to-Viewmodel bindings.

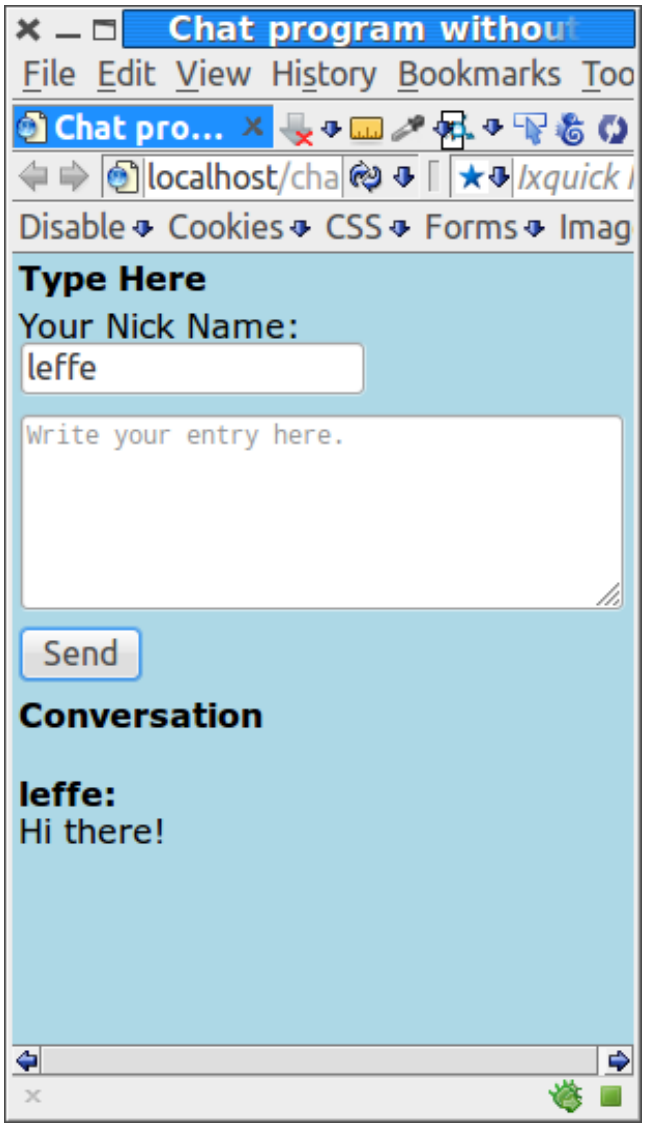
# The Knockout JavaScript Framework

- Like jQuery, Knockout is a JavaScript file the can be linked from a CDN, for example:

```
<script src="https://cdnjs.cloudflare.com/
  ajax/libs/knockout/3.1.0/knockout-min.js">
</script>
```

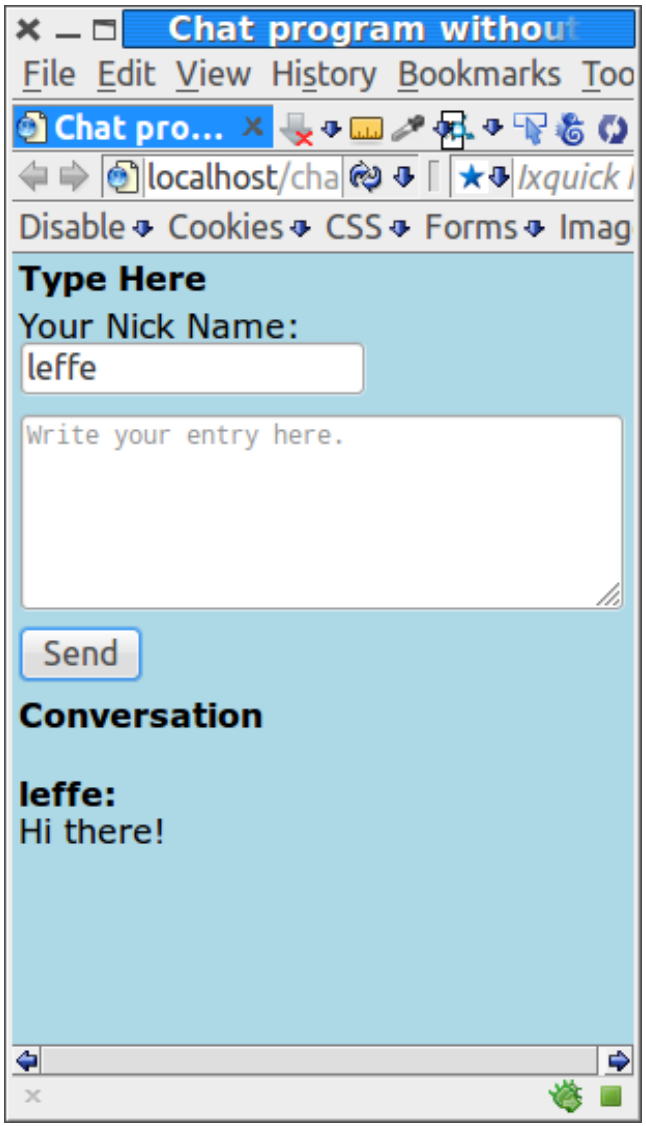
- Knockout should be used together with jQuery, which handles low-level DOM interaction.
- Knockout implements the MVVC pattern, by managing View-to-Viewmodel bindings.
- The following slides contain a brief introduction to Knockout. For a more extensive guide, see **`http://knockoutjs.com/documentation/introduction.html`**

# The Chat Application



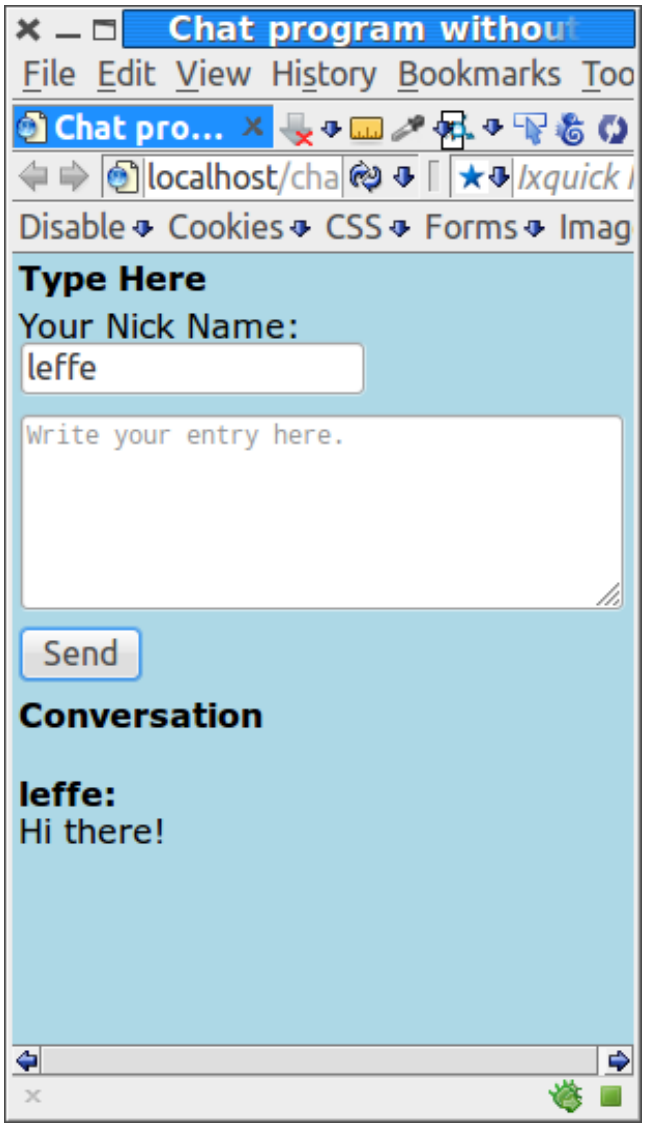

- To illustrate this Knockout tutorial, we will use the chat application.

# The Chat Application



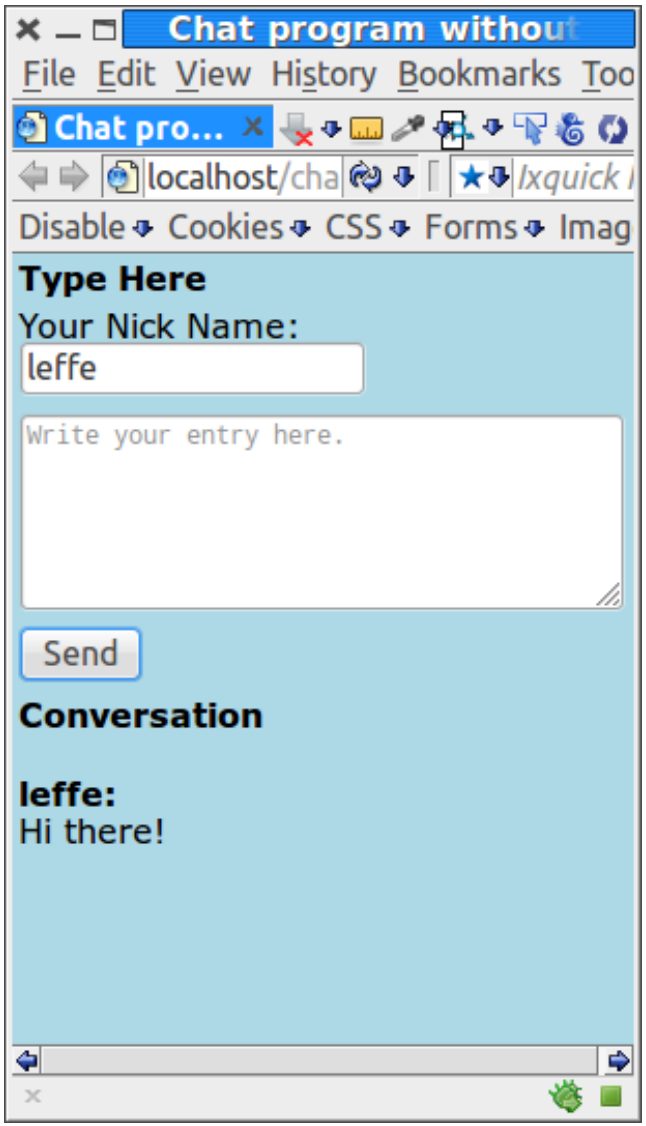

- To illustrate this Knockout tutorial, we will use the chat application.
- We will create a viewmodel that holds the current conversation.

# The Chat Application




- To illustrate this Knockout tutorial, we will use the chat application.
- We will create a viewmodel that holds the current conversation.
- The view shall be updated as soon as the viewmodel changes state.

# The Chat Application




- To illustrate this Knockout tutorial, we will use the chat application.
- We will create a viewmodel that holds the current conversation.
- The view shall be updated as soon as the viewmodel changes state.
- The viewmodel can change state either because the user wrote an entry or because another user wrote an entry and the server pushed that entry to this user's viewmodel.

# The Knockout Viewmodel

- The viewmodel is an ordinary JavaScript object, but to make use of Knockout's observer pattern viewmodel-to-view binding, the properties must be declared as observables.

```
function Person(name, age) {
  var self = this;
  self.name = ko.observable(name);
  self.age = ko.observable(age);
}
```

# The Knockout Viewmodel

- The viewmodel is an ordinary JavaScript object, but to make use of Knockout's observer pattern viewmodel-to-view binding, the properties must be declared as observables.

```
function Person(name, age) {
  var self = this;
  self.name = ko.observable(name);
  self.age = ko.observable(age);
}
```

- To read or write a property value, call the property as a function.

```
var olle = new Person("Olle", 35);
olle.name();  // Returns "Olle"
olle.age(36); // Sets the age to 36.
```

# The Chat Application's Viewmodel

- First we add two properties to the chat's viewmodel, **`author`** for the user's nick and **`newMsg`** for the author's last message. We also need a property for previous messages, that will soon be added.

```
function Conversation() {
  var self = this;
  self.author = ko.observable();
  self.newMsg = ko.observable("");
}
```

# The Chat Application's Viewmodel

- First we add two properties to the chat's viewmodel, **`author`** for the user's nick and **`newMsg`** for the author's last message. We also need a property for previous messages, that will soon be added.

```
function Conversation() {
  var self = this;
  self.author = ko.observable();
  self.newMsg = ko.observable("");
}
```

- The viewmodel must be registered with Knockout to enable notifying the observers in the view.

```
ko.applyBindings(new Conversation());
```

# Observable Arrays

- To observe a collection, use **`observableArray`**.

```
var myObservableArray = ko.observableArray();
```

# Observable Arrays

- To observe a collection, use **observableArray**.

```
var myObservableArray = ko.observableArray();
```

- Now we can give the chat viewmodel a property for the entire conversation. The **entries** property is an array of objects with properties **author** and **msg**

```
function Conversation() {
  var self = this;
  self.author = ko.observable();
  self.newMsg = ko.observable("");
  self.entries = ko.observableArray();
}
```

# Data Bindings

- A HTML element in the view is connected to a viewmodel property with a binding.

# Data Bindings

- A HTML element in the view is connected to a viewmodel property with a binding.
- A binding is declared by adding the **`data-bind`** attribute to the HTML element.

# Data Bindings

- A HTML element in the view is connected to a viewmodel property with a binding.
- A binding is declared by adding the **`data-bind`** attribute to the HTML element.
- There are many different types of bindings, like:

# Data Bindings

- A HTML element in the view is connected to a viewmodel property with a binding.
- A binding is declared by adding the **`data-bind`** attribute to the HTML element.
- There are many different types of bindings, like:

  text The property value is inserted to the HTML element.

```
The message is:
<span data-bind="text: msg"></span>
```

# Data Bindings

- A HTML element in the view is connected to a viewmodel property with a binding.
- A binding is declared by adding the **`data-bind`** attribute to the HTML element.
- There are many different types of bindings, like:

  **text** The property value is inserted to the HTML element.

```
The message is:
<span data-bind="text: msg"></span>
```

  **visible** Decides if the element is rendered.

```
<div data-bind=
   "visible: shouldShowMessage">
```

# Data Bindings

- A HTML element in the view is connected to a viewmodel property with a binding.
- A binding is declared by adding the **`data-bind`** attribute to the HTML element.
- There are many different types of bindings, like:

  **text** The property value is inserted to the HTML element.

```
The message is:
<span data-bind="text: msg"></span>
```

  **visible** Decides if the element is rendered.

```
<div data-bind=
   "visible: shouldShowMessage">
```

  **css** Adds or removes CSS classes. The following binding adds the class **`warning`** if the **`profit`** property is negative.

```
<div data-bind=
   "css: {warning: profit() < 0 }">
```

# Flow Control Data-Bindings

- There are also flow control data bindings, like **`if`** and **`foreach`**.

# Flow Control Data-Bindings

- There are also flow control data bindings, like **`if`** and **`foreach`**.
- **`foreach`** duplicates the containing element for each entry in an array, and binds each copy of the element to the corresponding array item.

# Flow Control Data-Bindings

- There are also flow control data bindings, like **`if`** and **`foreach`**.
- **`foreach`** duplicates the containing element for each entry in an array, and binds each copy of the element to the corresponding array item.
- Useful for rendering lists or tables.

# Flow Control Data-Bindings

- There are also flow control data bindings, like **`if`** and **`foreach`**.
- **`foreach`** duplicates the containing element for each entry in an array, and binds each copy of the element to the corresponding array item.
- Useful for rendering lists or tables.
- If the array is an observable array, whenever you later add, remove, or re-order array entries, the UI will be updated to reflect the new array contents.

# `foreach` Data-Binding (Cont'd)

Assuming **`people`** is a JavaScript array of objects with **`firstName`** and **`lastName`** properties, the following generates a table with one object per row.

```
<table>
  <thead>
      <tr><th>First name</th>
      <th>Last name</th></tr>
  </thead>
  <tbody data-bind="foreach: people">
      <tr>
          <td data-bind="text: firstName"></td>
          <td data-bind="text: lastName"></td>
      </tr>
  </tbody>
</table>
```

# `foreach` Data-Binding (Cont'd)

- The current array element is referred using **`$data`**

```
<ul data-bind="foreach: months">
  <li>
    <span data-bind="text: $data"></span>
  </li>
</ul>
```

# `foreach` Data-Binding (Cont'd)

- The current array element is referred using **`$data`**

```
<ul data-bind="foreach: months">
  <li>
    <span data-bind="text: $data"></span>
  </li>
</ul>
```

- **`as`** gives an alias to the current array element.

```
<ul data-bind=
     "foreach: { data: categories, as: 'category' }">
    <li>
        <ul data-bind=
             "foreach: { data: items, as: 'item' }">
            <li>
                <span data-bind="text: category.name">
                </span>:
                <span data-bind="text: item"></span>
            </li>
        </ul>
    </li>
</ul>
```

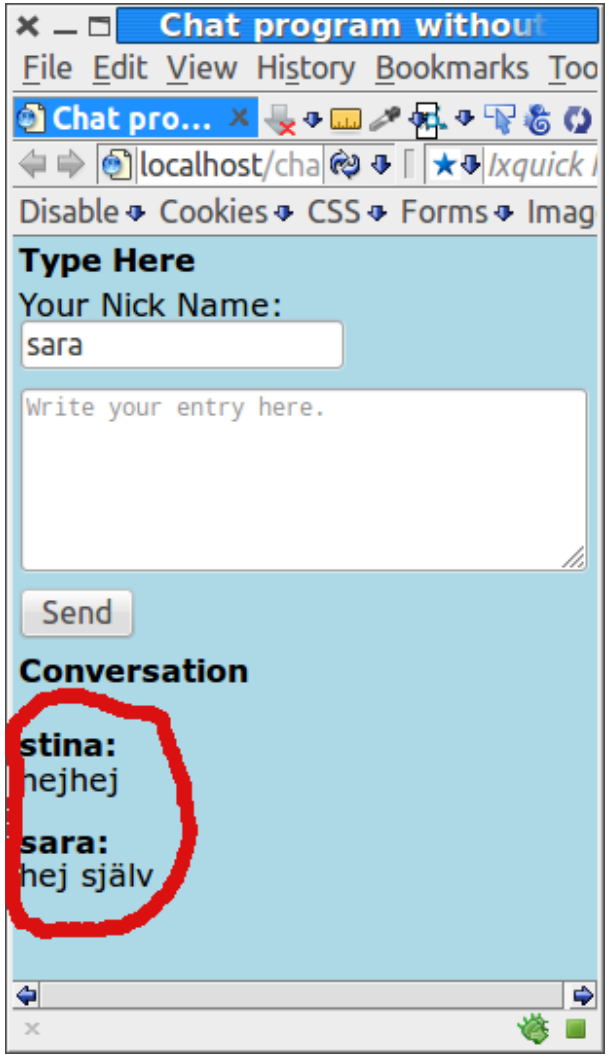
# Containerless Control Flow Syntax



- If there is no containing element for the data-binding, use the containerless syntax, based on HTML comment tags.

```
<!-- ko foreach: {data:items, as:'item'} -->
    <p>
        <span data-bind="text:item.price">
        </span>
    </p>
<!-- /ko -->
```

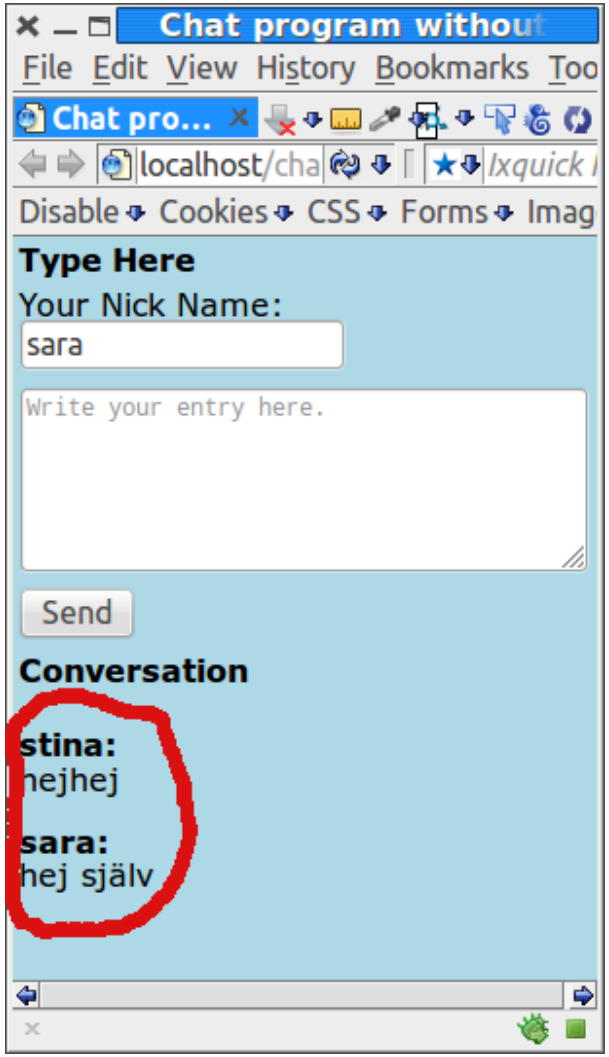
# The Conversation Data-Binding




- Now we can create the data-binding for the conversation part of the chat application.

```
<h2>Conversation</h2>
<div>
  <!-- ko foreach: {data: entries,
                    as: 'entry'} -->
    <p class='author'>
      <span data-bind=
        "text: entry.author"></span>:
    </p>
    <!-- ko foreach: entry.msg -->
      <p>
        <span data-bind=
          "text: $data"></span>
      </p>
    <!-- /ko -->
  <!-- /ko -->
</div>
```

# The Conversation Data-Binding




- Now we can create the data-binding for the conversation part of the chat application.

```
<h2>Conversation</h2>
<div>
  <!-- ko foreach: {data: entries,
                    as: 'entry'} -->
    <p class='author'>
      <span data-bind=
        "text: entry.author"></span>:
    </p>
    <!-- ko foreach: entry.msg -->
      <p>
        <span data-bind=
          "text: $data"></span>
      </p>
    <!-- /ko -->
  <!-- /ko -->
</div>
```

- The inner loop is needed for multi-line messages.

# Form Field Bindings

There are also bindings for form elements, such as:

**click** Specifies a method that is called when the element is clicked.

```
<button data-bind=
   "click: clickHandler">Click me</button>
```

# Form Field Bindings

There are also bindings for form elements, such as:

**click** Specifies a method that is called when the element is clicked.

```
<button data-bind=
   "click: clickHandler">Click me</button>
```

**event** Binds any type of event to a method.

```
<div data-bind="event: { mouseover: enableDetails,
   mouseout: disableDetails }">
```

# Form Field Bindings

There are also bindings for form elements, such as:

**click** Specifies a method that is called when the element is clicked.

```
<button data-bind=
   "click: clickHandler">Click me</button>
```

**event** Binds any type of event to a method.

```
<div data-bind="event: { mouseover: enableDetails,
   mouseout: disableDetails }">
```

**value** Links an element's value with a property.

```
<input type="password"
   data-bind="value: userPassword"/>
```

# Form Field Bindings

There are also bindings for form elements, such as:

**click** Specifies a method that is called when the element is clicked.

```
<button data-bind=
   "click: clickHandler">Click me</button>
```

**event** Binds any type of event to a method.

```
<div data-bind="event: { mouseover: enableDetails,
   mouseout: disableDetails }">
```

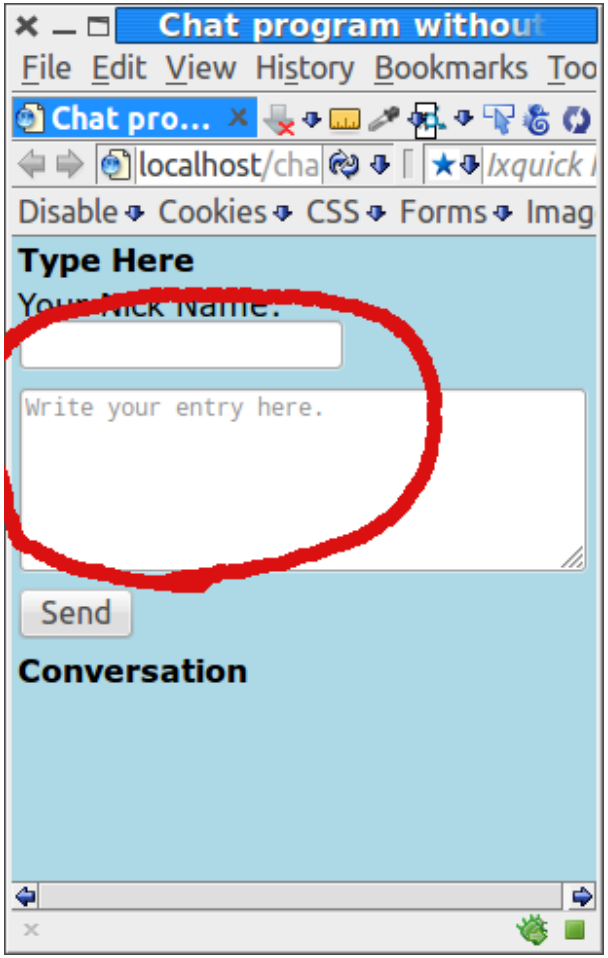
**value** Links an element's value with a property.

```
<input type="password"
   data-bind="value: userPassword"/>
```

**enable** The element is enabled only when the value is **`true`**

```
Your cellphone number:
<input type='text' data-bind=
   "value: cellphoneNumber, enable: hasCellphone"/>
```

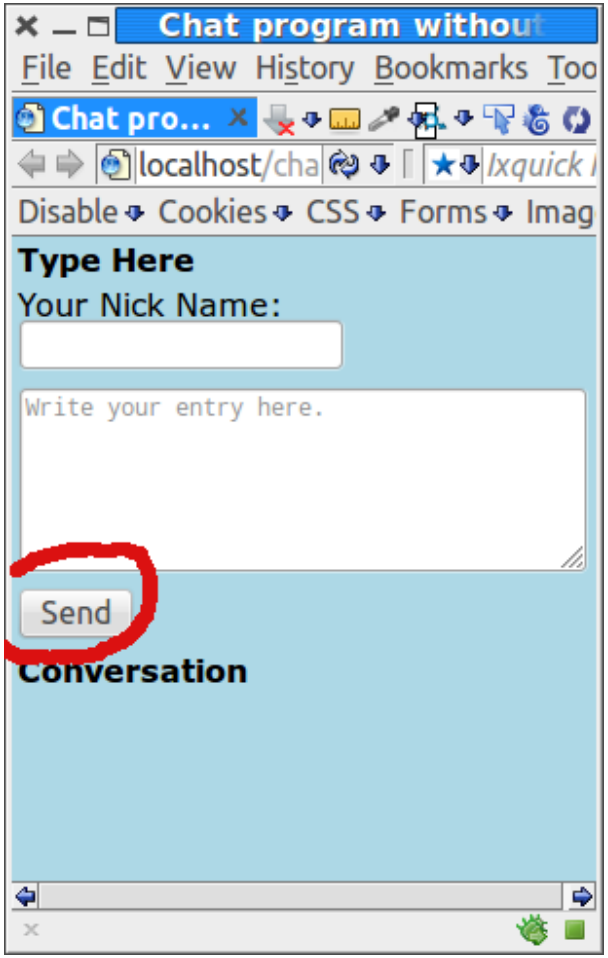
# Chat Value Data-Bindings




- Now let us create the data-bindings for the nick name text field and message text area.

```
<input id="nickName"
       class='text-author'
       data-bind="value: author"/>
<textarea id= "entry" rows = 5
          data-bind="value: newMsg"
          placeholder="Write your entry here.">
          </textarea>
```

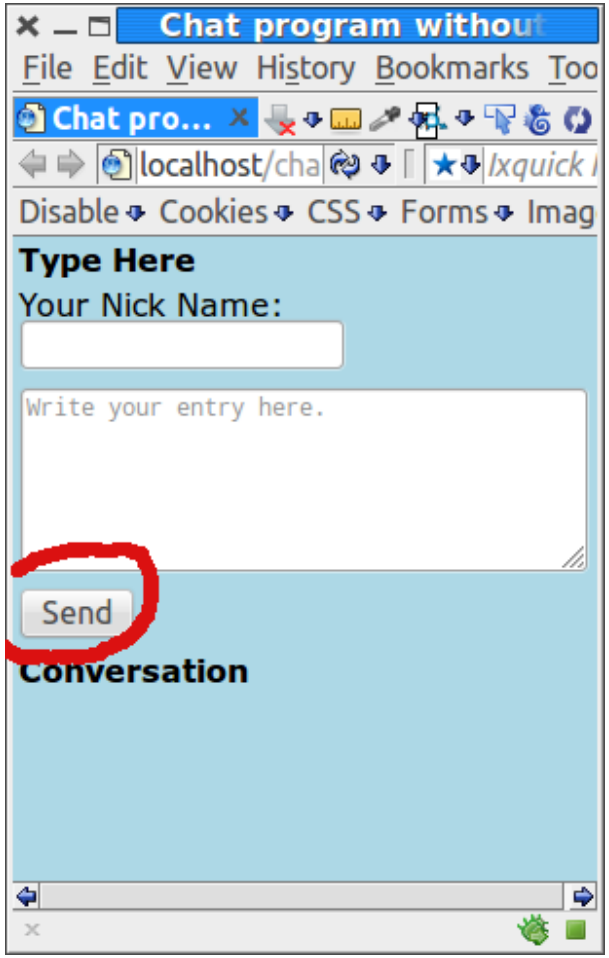
# Chat Click Data-Bindings




- Then let us create the data-binding for the Send button.

```
<button data-bind="click: addEntry">
   Send
</button>
```

# Chat Click Data-Bindings




- Then let us create the data-binding for the Send button.

```
<button data-bind="click: addEntry">
   Send
</button>
```

- Here we specified that the **`addEntry`** method is called when the user clicks the button. To create this method is the last thing that remains.

# The `addEntry` Method in the `conversation` Object

```
1  self.addEntry = function() {
2    if (self.newMsg() !== "") {
3      var msg =
4        new String(self.newMsg()).split("\n");
5      self.entries.push({author:self.author(),
6                         msg:msg});
7      self.newMsg("");
8    }
9  };
```

- Line 2 checks that the user had typed a message.

# The `addEntry` Method in the `conversation` Object

```
1  self.addEntry = function() {
2    if (self.newMsg() !== "") {
3      var msg =
4        new String(self.newMsg()).split("\n");
5      self.entries.push({author:self.author(),
6                         msg:msg});
7      self.newMsg("");
8    }
9  };
```

- Line 2 checks that the user had typed a message.
- Lines 3-4 splits the lines of a multi-line message.

# The `addEntry` Method in the `conversation` Object

```
1  self.addEntry = function() {
2    if (self.newMsg() !== "") {
3      var msg =
4        new String(self.newMsg()).split("\n");
5      self.entries.push({author:self.author(),
6                         msg:msg});
7      self.newMsg("");
8    }
9  };
```

- Line 2 checks that the user had typed a message.
- Lines 3-4 splits the lines of a multi-line message.
- Lines 5-6 adds a new entry to the observable array property **`entries`**.

# The `addEntry` Method in the `conversation` Object

```
1  self.addEntry = function() {
2    if (self.newMsg() !== "") {
3      var msg =
4        new String(self.newMsg()).split("\n");
5      self.entries.push({author:self.author(),
6                         msg:msg});
7      self.newMsg("");
8    }
9  };
```

- Line 2 checks that the user had typed a message.
- Lines 3-4 splits the lines of a multi-line message.
- Lines 5-6 adds a new entry to the observable array property `entries`.
- Line 7 clears the text area.

- That was all, now we have seen an overview of some features in Knockout, and also created the chat application.

- That was all, now we have seen an overview of some features in Knockout, and also created the chat application.
- There are many more features in Knockout.

- That was all, now we have seen an overview of some features in Knockout, and also created the chat application.
- There are many more features in Knockout.
- The complete chat application is available on the course's web page.

- That was all, now we have seen an overview of some features in Knockout, and also created the chat application.
- There are many more features in Knockout.
- The complete chat application is available on the course's web page.
- Of course, we are not done writing the chat, since there is no communication with the server. That is left for a coming lecture when server side programming is covered.