中山大學
SUN YAT-SEN UNIVERSITY

# Lecture 17
# DOM Events

**SE-805 Web 2.0 Programming (supported by Google)**

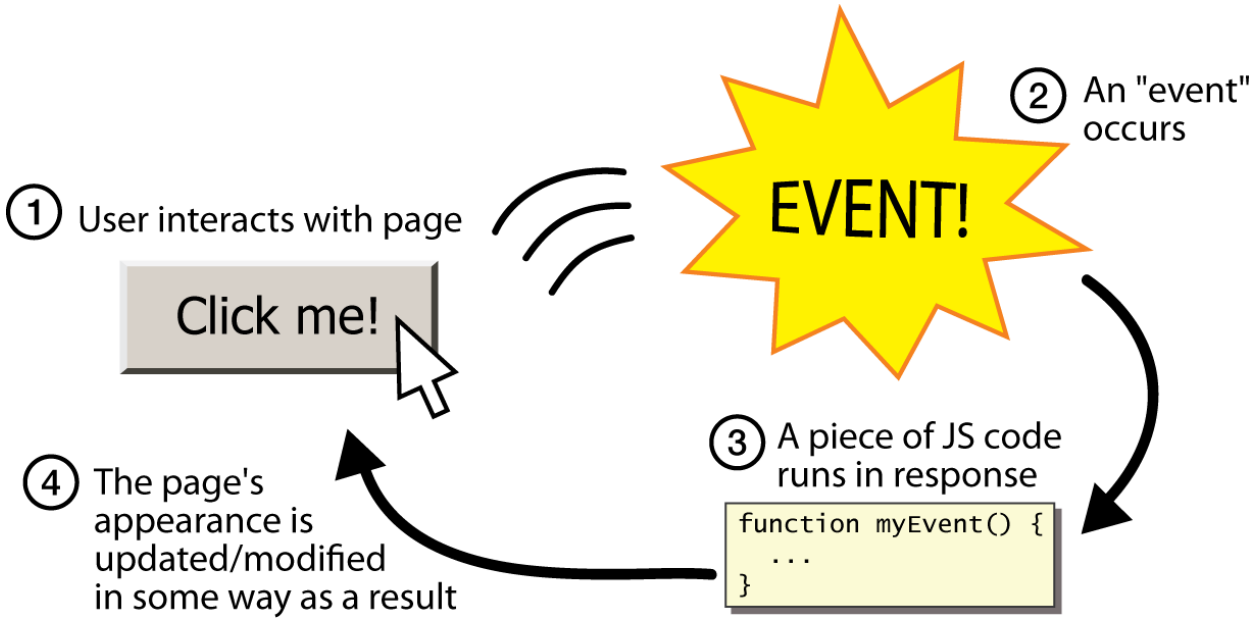
http://my.ss.sysu.edu.cn/courses/web2.0/

**School of Software, Sun Yat-sen University**

# Outline

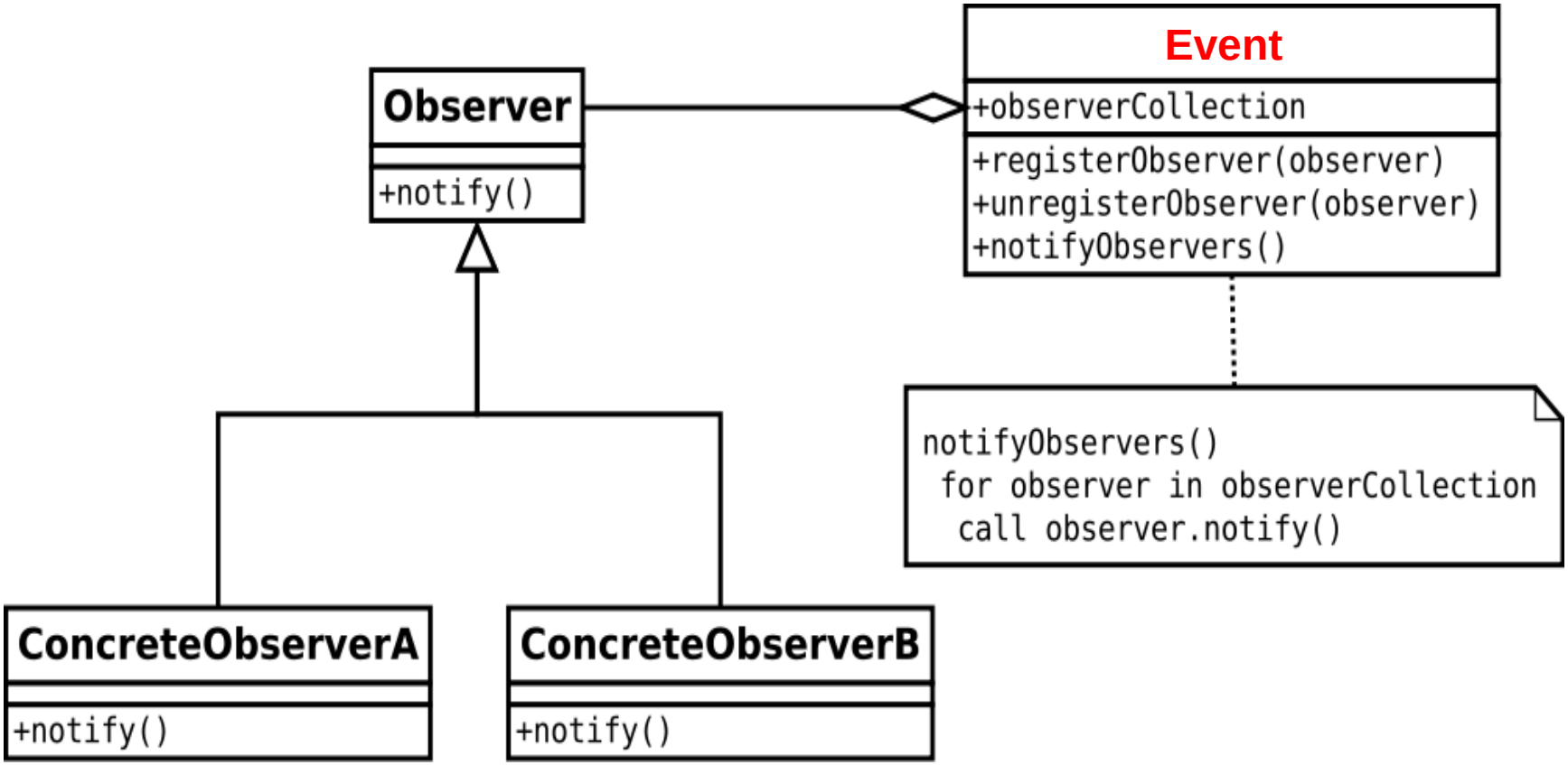
- **Observer pattern**
- DOM 2 event flow
- Event handling

# Event-driven Programming



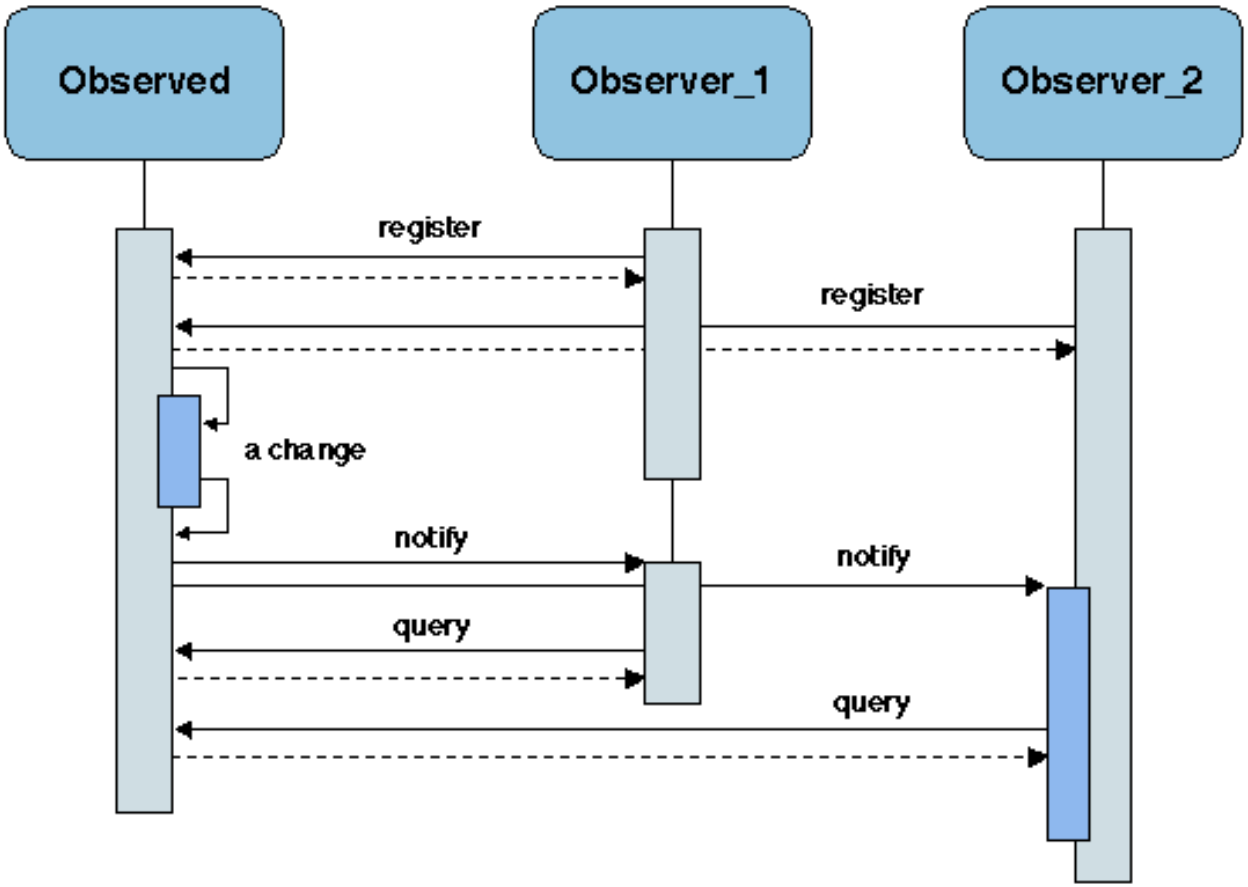

- **Event-driven programming**: writing programs driven by user events

# Observer Pattern




Events makes a subject may have **multiple** observer queues

# Observer Pattern



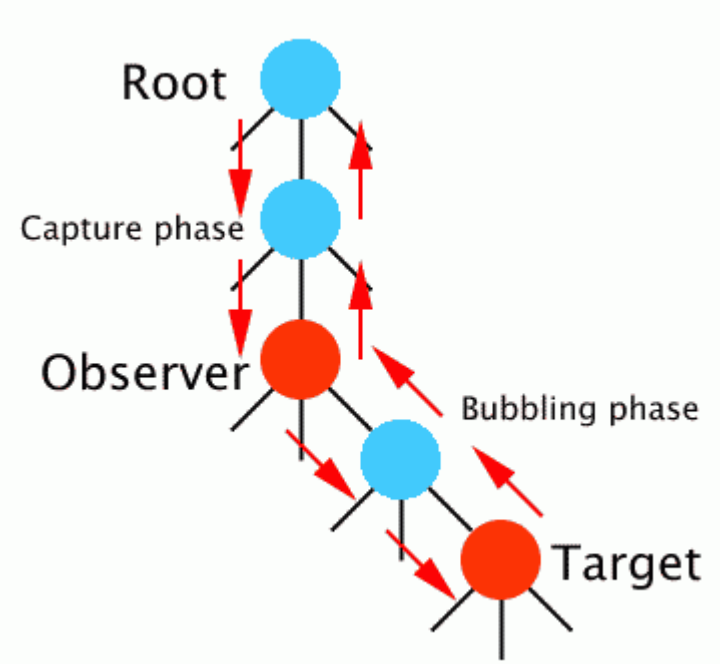

- Observer pattern is the base for event-driven programming
- How can we apply observer pattern on the complex DOM tree?

# Outline

- Observer pattern
- **DOM 2 event flow**
- Event handling

# Event Flow

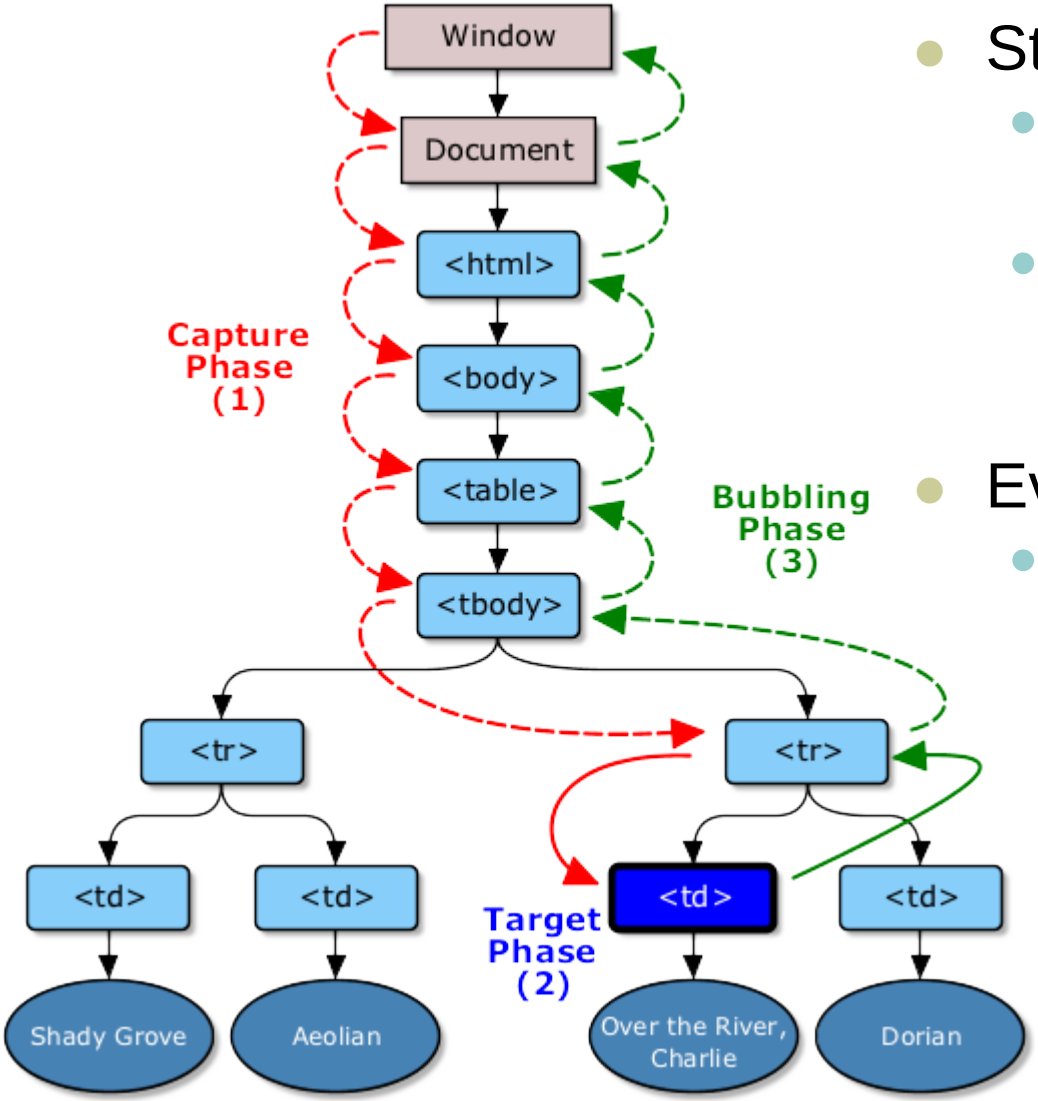
- Each event has a target, which can be accessed via event




```
element.onclick = handler(e);
function handler(e){
    if(!e) var e = window.event;
    // e refers to the event
    // see detail of event
    var original = e.eventTarget;
}
```

- Each event originates from the browser, and is passed to the DOM
- DOM propagates the event in 3 phases:
  - **Capture** phase, **target** phase, **bubbling** phase (some events have no bubbling phase, i.e. load event of document element.)
  - Register a capture phase handler: (IE can't do this)
    `element.addEventListener('click',handler,true);`

# Event Flow




- Stopping event propagation
  - By throwing any exception inside an event handler
  - By calling `event.stopProgagation();` inside a handler
- Event cancelation
  - Canceling default action (i.e. navigating to a new page when clicking on a hyperlink): `event.preventDefault();`

# Event Handler Registration

- Inline:
  - `<a href="somewhere.html" onClick="doSomething()">`
- Traditional:
  - `element.onclick = doSomething;`
- DOM 2:
  - `element.addEventListener('click', doSomething, false);`
- IE: (**evil enough!**)
  - `element.attachEvent('onclick', doSomething);`
- Prototype: (**prefered ☺**)
  - `Event.observe('target', 'click', doSomething);`
  - `document.observe('dom:loaded', doSomething);`
- More details

# Outline

- Observer pattern
- DOM 2 event flow
- **Event handling**

# The Keyword this

```
this.fieldName                    // access field
this.fieldName = value;           // modify field

this.methodName(parameters);      // call method
```
JS

- All JavaScript code actually runs inside of an object
- By default, code runs inside the global window object
  - All global variables and functions you declare become part of window
- The this keyword refers to the current object
- An event handler is executed in the scope of the element it registered on, therefore it can use this to access the DOM node of the element, that is to say,
  - Inside the handler, that element becomes this (rather than the window)

# DOM 2 Event Types

- UI event types:
  - DOMFocusIn, DOMFocusOut, DOMActivate
- Mouse event types:
  - click, mousedown, mouseup, mouseover, mousemove, mouseout
- Key event types: (not in DOM 2, but will in DOM 3)
- Mutation events:
  - DOMSubtreeModified, DOMNodeInserted, …
- HTML event types:
  - load, unload, abort, error, select, change, submit, reset, focus, blur, resize, scroll

- more details

# Useful Event Types

| | | | | | |
|---|---|---|---|---|---|
| abort | blur | change | click | dblclick | error |
| keydown | keypress | keyup | load | mousedown | mousemove |
| mouseover | mouseup | reset | resize | select | submit |
| focus | mouseout | unload | | | |

- **Problem**: events are tricky and have incompatibilities across browsers reasons: fuzzy W3C event specs; IE disobeying web standards; etc.
- **Solution**: Prototype includes many event-related features and fixes

# Attaching Event Handlers the Prototype Way

```JS
element.onevent = function;
element.observe("event", "function");
```

```JS
// call the playNewGame function when the Play button is clicked
$("play").observe("click", playNewGame);
```

- To use Prototype's event features, you **must** attach the handler using the DOM element object's **observe** method (added by Prototype)
- Pass the event of interest and the function to use as the handler
- Handlers *must* be attached this way for Prototype's event features to work
- observe substitutes for addEventListener and attachEvent (IE)

# Attaching Multiple Event Handlers with $$

```
// listen to clicks on all buttons with class "control" that
// are directly inside the section with ID "game"
window.onload = function() {
  var gameButtons = $$("#game > button.control");
  for (var i = 0; i < gameButtons.length; i++) {
    gameButtons[i].observe("click", gameButtonClick);
  }
};

function gameButtonClick() { ... }
```
JS

- You can use $$ and other DOM walking methods to unobtrusively attach event handlers to a group of related elements in your `window.onload` code

# The Event Object

```JS
function name(event) {
  // an event handler function ...
}
```

- Event handlers can accept an optional parameter to represent the event that is occurring. Event objects have the following properties / methods:

| Method / property name | Description |
|---|---|
| type | what kind of event, such as "click" or "mousedown" |
| element() * | the element on which the event occurred |
| stop() ** | cancels an event |
| stopObserving() | removes an event handler |

* Replaces non-standard `srcElement` and `which` properties

** Replaces non-standard `return false;`, `stopPropagation`

# Mouse Events

## Clicking

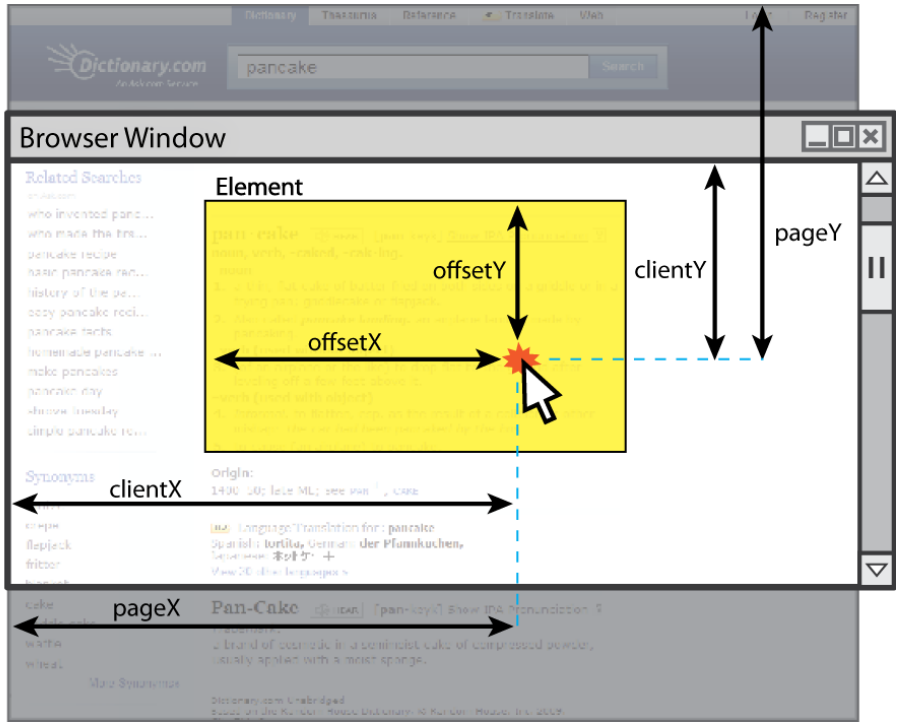
| | |
|---|---|
| click | user presses/releases mouse button on this element |
| dblclick | user presses/releases mouse button twice on this element |
| mousedown | user presses down mouse button on this element |
| mouseup | user releases mouse button on this element |

## Movement

| | |
|---|---|
| mouseover | mouse cursor enters this element's box |
| mouseout | mouse cursor exits this element's box |
| mousemove | mouse cursor moves around within this element's box |
| mouseover | mouse cursor enters this element's box |

# Mouse Event Objects

- The event parameter passed to a mouse event handler has the following properties:




| Property/ Method | Description |
|---|---|
| clientX, clientY | coordinates in browser window |
| screenX, screenY | coordinates in screen |
| offsetX, offsetY | coordinates in element |
| pointerX(), pointerY() * | coordinates in entire web page |
| isLeftClick() ** | true if left button was pressed |

* Replaces non-standard properties pageX and pageY
** Replaces non-standard properties button and which

# Mouse Event Example

```html
<pre id="target">Move the mouse over me!</pre>
```

HTML

```js
window.onload = function() {
  $("target").observe("mousemove", showCoords);
};

function showCoords(event) {
  this.innerHTML =
      "pointer: (" + event.pointerX() + ", " + event.pointerY() + ")\n"
    + "screen : (" + event.screenX + ", " + event.screenY + ")\n"
    + "client : (" + event.clientX + ", " + event.clientY + ")";
}
```

JS

```
pointer: (1333, 471)
screen : (-100, 734)
client : (1333, 471)
```

output

# Summary

- Observer pattern
  - Event-driven programming, observer pattern
- DOM 2 event flow
  - Event flow (capture, target, bubbling, stop, cancel)
  - Handler registration (inline, traditional, DOM 2, IE, Prototype)
- Event handling
  - this
  - Event types & useful event types
  - Handling events in prototype ways
  - Mouse events

# Exercises

- Write a simple to-do list application as a web page.
  - A  element wraps all html elements for this to-do application
  - A form with a textarea for specifying a new to-do item and a "add" button for adding it to the list
  - A list of current to-do items
  - Each item has a checkbox for select
  - Buttons "select all", "deselect all", "remove" (which removes all selected to-do items from the list)
  - When the "add" button is clicked the new to-do item will be inserted to the bottom of the list
  - Use the Unobtrusive JavaScript technique learned
    - Use the prototype.js functions of DOM events handling
    - All event handlers are registered on the "to-do" div element
    - Use the event.element() to figure out the source of an event

# Further Readings

- W3School DOM node reference
  http://www.w3school.com/dom/dom_node.asp/
- W3School DOM tutorial
  http://www.w3schools.com/htmldom/
- Quirksmode DOM tutorial
  http://www.quirksmode.org/dom/intro.html
- Prototype Learning Center
  http://www.prototypejs.org/learn
- How prototype extends the DOM
  http://www.prototypejs.org/learn/extensions

# Thank you!