Object-Oriented JavaScript Dynamic HTML

Prof. Cesare Pautasso

http://www.pautasso.info

cesare.pautasso@unisi.ch

Contents

- JavaScript
  - as an object-oriented programming language
  - inheritance styles
    - Pseudo-Classical
    - Prototypical
    - Parasitic
- Document Object Model (DOM)
  - Traversing the document tree
  - W3C DOM API
  - Access the HTML Parser
- Dynamic HTML

Università della Svizzera italiana

Facoltà di scienze informatiche

Object-Oriented JavaScript

Università della Svizzera italiana

Facoltà di scienze informatiche

Object-Oriented 101

1. Encapsulation

   “separate the interface from the implementation of the object”

   Although JavaScript does not have the usual “private”, “protected”, “public” keywords, *there are ways for hiding parts of the implementation of an object based on closure*

2. Inheritance

   “define more specialized versions of a super-class”

   JavaScript supports 3 inheritance styles
   (Pseudo-Classical, Prototypical, Parasitic)

3. Polymorphism

   “treat derived class members just like their parent class members”

   Thanks to dynamic typing
   in JavaScript you get this for free

Properties

- Object = container of unordered collection of named properties (and methods)
  - In Java: JSObject = Map<String, Object>
- Create an empty object:

```
var student = {};
```

- Populate its properties:

```
student.name = “Peggy”;
Student.university = “USI”;
student.date_of_birth = new Date(…);
```

Methods

- Declare a method for the object:

```
student.age = function() {
  return (new Date().getTime()) –
  this.date_of_birth.getTime();
}
```

- Use this to access the fields of the objects
- Call a method:

```
if (student.age() > 18) { … };
```

Università della Svizzera italiana | Facoltà di scienze informatiche

Methods and This

- Function Call `f();`
  - this = the global object
- Method Call `obj.m();`
  - this = the object **obj**
- Constructor `new C();`
  - this = the new object
- Event Handler `onclick=“m(this)”`
  - this = the DOM element on which the event occurs

Università della Svizzera italiana | Facoltà di scienze informatiche

Constructors

- Objects of the same “class” can be setup by a special function, the constructor
  - Any function called with new becomes a constructor
  - Constructors name typically begin with an uppercase letter
- The constructor initializes the properties

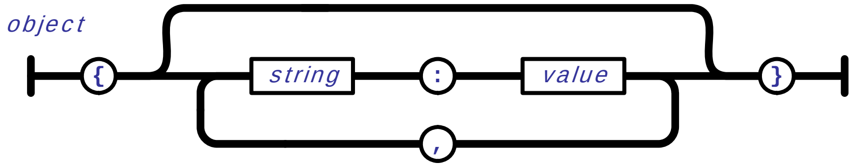
```
function Person(name)
{
  this.name = name;
  this.age = function() {…};
  return this; //not needed
}

me = new Person(“Peggy”);
me.age();
```

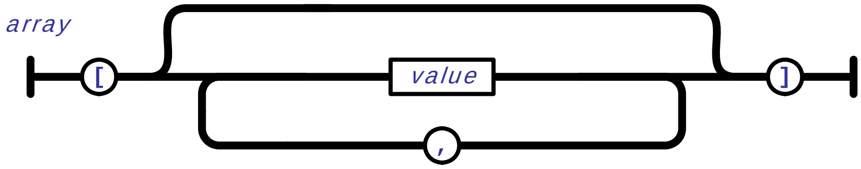
Object Literals

- Objects can also be created directly using object literals:

```
var person = {
  name: "Peggy",
  date_of_birth: new Date(1927,0,31),
  address: {
          street: 'Via Larga',
          number: 22
        }
  };
```

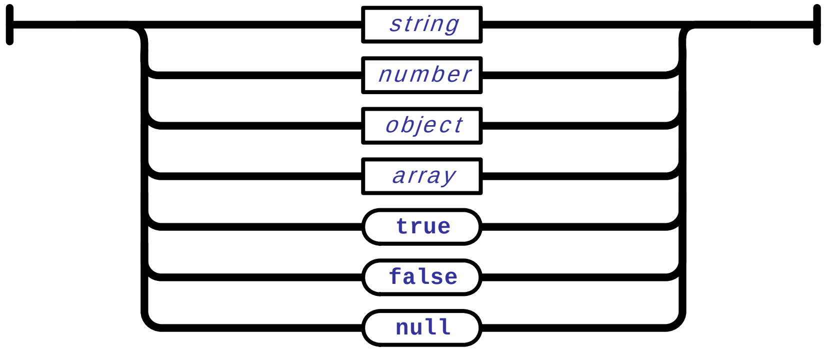
Object Literal Syntax





Value Syntax

value



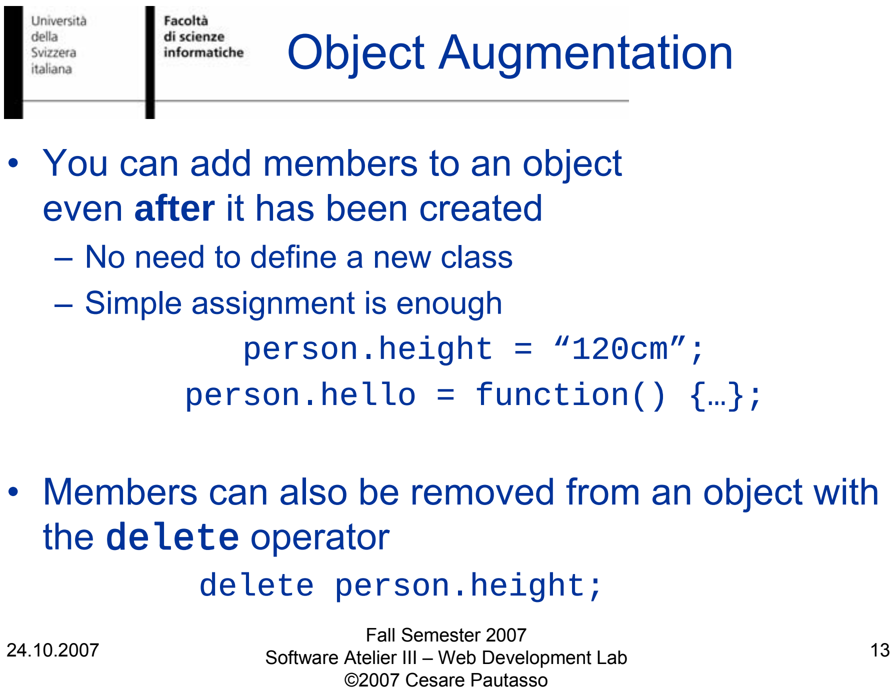
string
number
object
array
true
false
null

- In general, values inside object literals can be any JavaScript expression.

Constructor with Literals

```
function person(name, dob) {
 return {
     name: name,
     date_of_birth: dob,
     age: function() {
          return…
     }
   };
}

var p = person(“Peggy”, new Date(…));
```

Warning: **new** is not needed here!



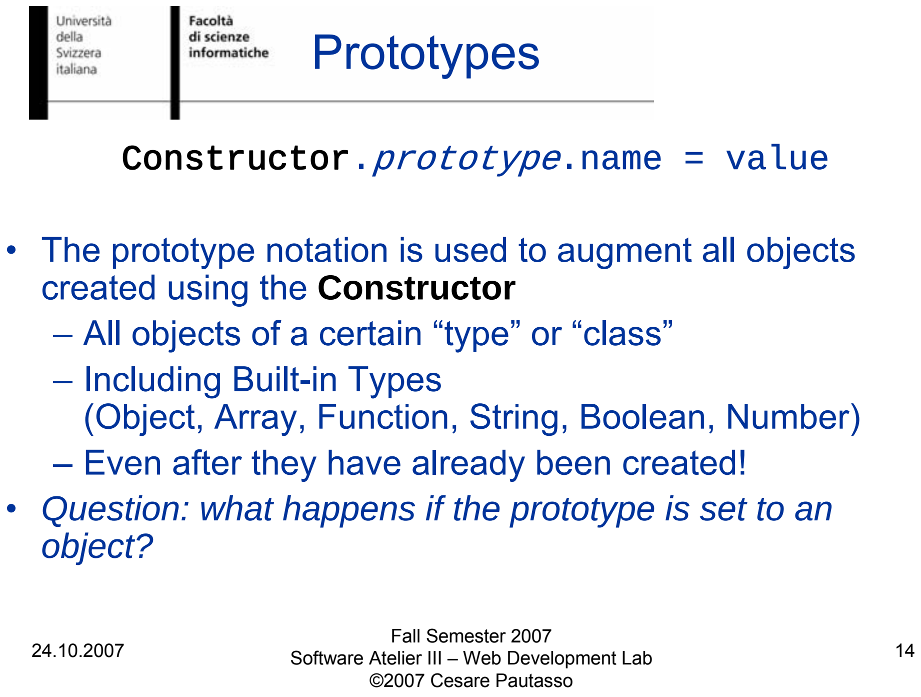
Object Augmentation

- You can add members to an object even **after** it has been created
  - – No need to define a new class
  - – Simple assignment is enough

```
        person.height = “120cm”;
    person.hello = function() {…};
```

- Members can also be removed from an object with the **`delete`** operator

```
    delete person.height;
```



Prototypes

```
Constructor.prototype.name = value
```

- The prototype notation is used to augment all objects created using the **Constructor**
  - – All objects of a certain “type” or “class”
  - – Including Built-in Types (Object, Array, Function, String, Boolean, Number)
  - – Even after they have already been created!
- *Question: what happens if the prototype is set to an object?*

String prototype Example

- This will add a method called `trim` to the built-in String class

```
String.prototype.trim = function () {
    return this.replace(
        /^\s*(\S*(\s+\S+)*)\s*$/, "$1");
};


"    hallo world! ".trim();
```

Prototypes and Constructors

```
function Person(name,dob)
{
 this.name = name;
 this.date_of_birth = dob;
}

Person.prototype.age = function()
{
     return new Date() - this.date_of_birth;
}

me = new Person(“Peggy”, new Date(1929,9,24));
me.age();
```

Note: this is just a matter of style!

Inheritance Styles

- **Object hierarchies are constructed by assigning an object as the prototype associated with a constructor function.**

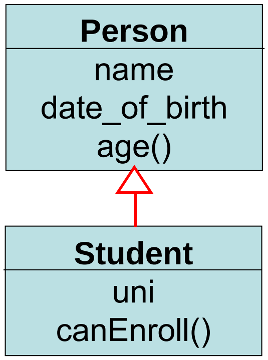
- The basic (and very simple) JavaScript syntax supports different styles of object inheritance.

1. **Pseudo-Classical**
   - For people that still think in terms of *classes* and inheritance between them (not recommended)
2. **Prototypical**
   - Create an object that inherits from another one (the two are linked using *prototypes*)
3. **Parasitic**
   - Augment objects using “power” constructors
     (also supports private/public members)

- In the first two styles, some “sugar” is needed to hide the machinery involved in establishing inheritance links between objects.

Pseudo-Classical Inheritance

```
function Person(name,dob) {
   this.name = name;
   this.date_of_birth = dob;
}
Person.prototype.age = function () {…}

function Student(name,dob,uni) {
   this.Person(name,dob);
   this.uni = uni;
}
Inherit(Student,Person);

Student.prototype.canEnroll = function () {…}

var s = new Student(“Peggy”,new Date(…),”USI”);
s.age();
if (s.canEnroll())…
```

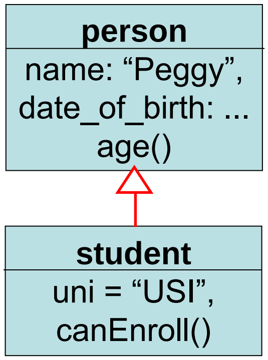


Pseudo-Classical Inheritance Machinery

```
function Inherit(descendant, parent) {
  var sConstructor = parent.toString();
  var aMatch =
  sConstructor.match( /\s*function (.*)\(/ );
  if ( aMatch != null ) {
  descendant.prototype[aMatch[1]] = parent;
  }
  for (var m in parent.prototype) {
   descendant.prototype[m] =
                          parent.prototype[m];
  }
};
```

Prototypical Inheritance

```
person = {};
person.name = “Peggy”;
person.date_of_birth = new Date(…);
person.age = function() {…};

student = object(person);
student.uni = “USI”;
student.canEnroll = function () {…};

student.age();
if (student.canEnroll())…
```



Questions:

1. What happens to **student.name** if you write **person.name=“Sue”**?
2. What happens to **person.name** if you write **student.name=“Sue”**?

Prototypical Inheritance **Machinery**

```
function object(o) {
  function F() {};
  F.prototype = o;
  return new F();
}
```

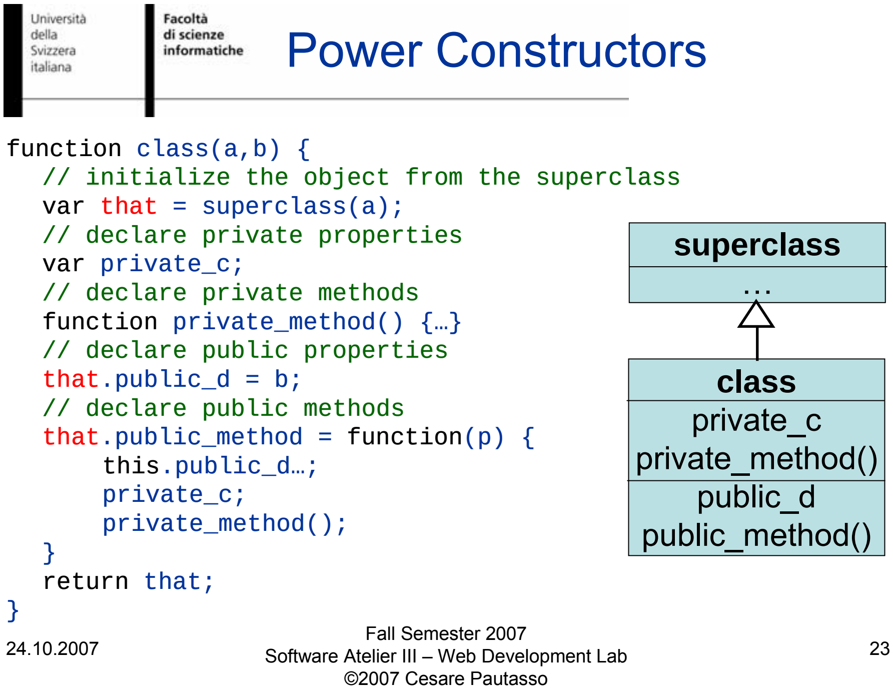
Parasitic Inheritance

```
function person(name, dob) {
   return {
     name: name,
     date_of_birth: dob,
     age: function(){…}
   };
}

function student(name, dob, uni) {
   var that = person(name, dob);
   that.uni = uni;
   that.canEnroll = function() {…};
   return that;
}

var s = student("Peggy", new Date(…), "USI");

s.age();
if (s.canEnroll())…
```

Note: Look, no special machinery!

Warning: These are not constructors!

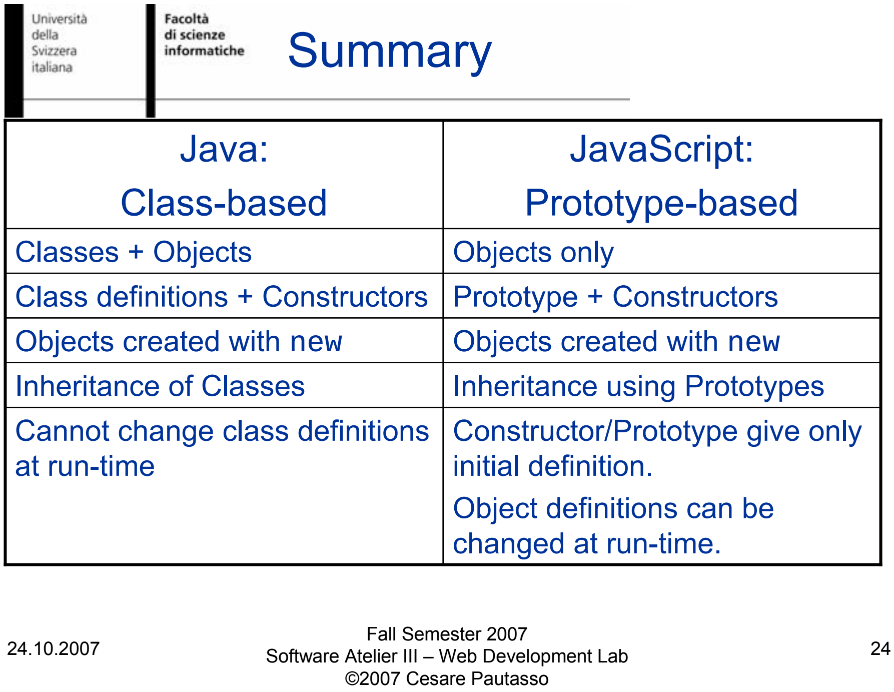
Note: `prototype`, `new` not used



Università della Svizzera italiana | Facoltà di scienze informatiche

Power Constructors

```
function class(a,b) {
   // initialize the object from the superclass
   var that = superclass(a);
   // declare private properties
   var private_c;
   // declare private methods
   function private_method() {…}
   // declare public properties
   that.public_d = b;
   // declare public methods
   that.public_method = function(p) {
        this.public_d…;
        private_c;
        private_method();
   }
   return that;
}
```



Università della Svizzera italiana | Facoltà di scienze informatiche

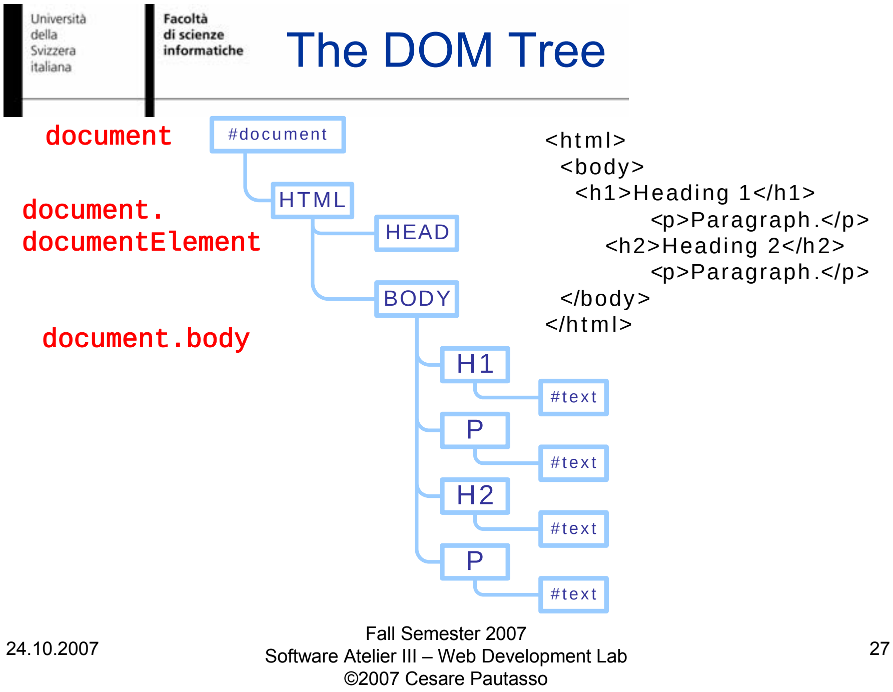
Summary

Java: Class-based	JavaScript: Prototype-based
Classes + Objects	Objects only
Class definitions + Constructors	Prototype + Constructors
Objects created with new	Objects created with new
Inheritance of Classes	Inheritance using Prototypes
Cannot change class definitions at run-time	Constructor/Prototype give only initial definition. Object definitions can be changed at run-time.

Dynamic HTML

What is Dynamic HTML?

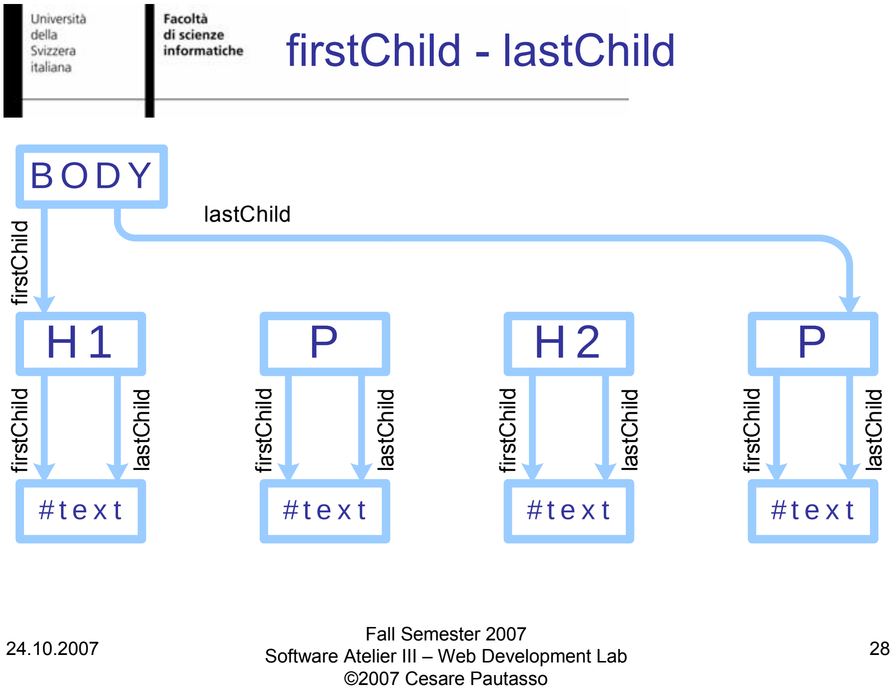
- Manipulate the DOM of an HTML page from the JavaScript code
  - Add new elements
  - Remove existing elements
  - Change the position of elements in the tree
  - Modify element content (`innerHTML`)
  - Control the element CSS style (formatting, visibility, position, layout)
  - Respond to user events



Università della Svizzera italiana | Facoltà di scienze informatiche

The DOM Tree

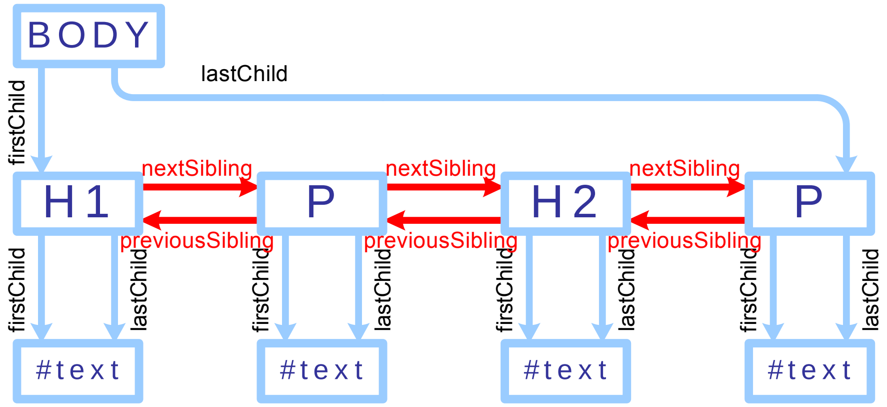
```
<html>
 <body>
  <h1>Heading 1</h1>
       <p>Paragraph.</p>
    <h2>Heading 2</h2>
       <p>Paragraph.</p>
 </body>
</html>
```



Università della Svizzera italiana | Facoltà di scienze informatiche

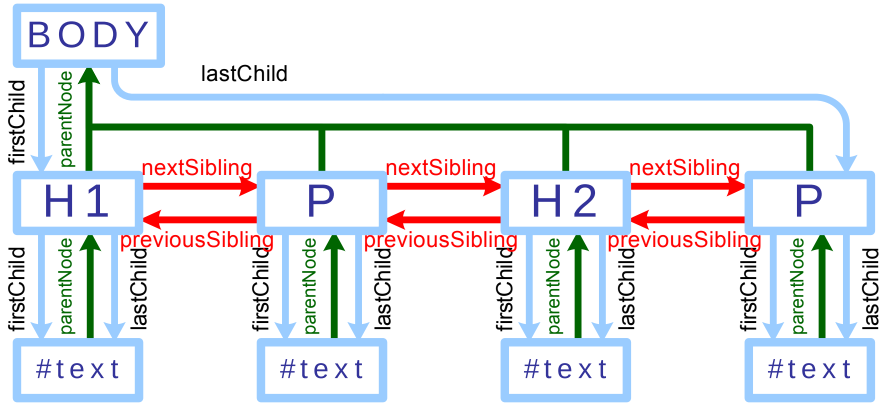
firstChild - lastChild

nextSibling, previousSibling



BODY
lastChild
firstChild
nextSibling nextSibling nextSibling
H1 P H2 P
previousSibling previousSibling previousSibling
firstChild lastChild firstChild lastChild firstChild lastChild firstChild lastChild
#text #text #text #text

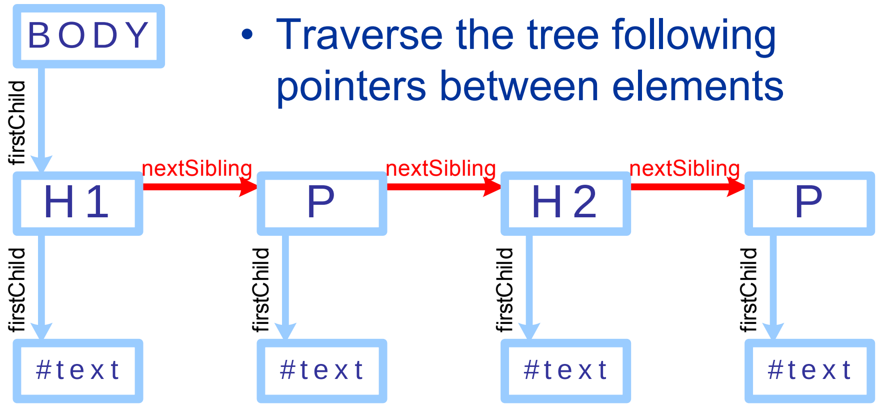
parentNode



BODY
lastChild
firstChild parentNode
nextSibling nextSibling nextSibling
H1 P H2 P
previousSibling previousSibling previousSibling
firstChild parentNode lastChild firstChild parentNode lastChild firstChild parentNode lastChild firstChild parentNode lastChild
#text #text #text #text

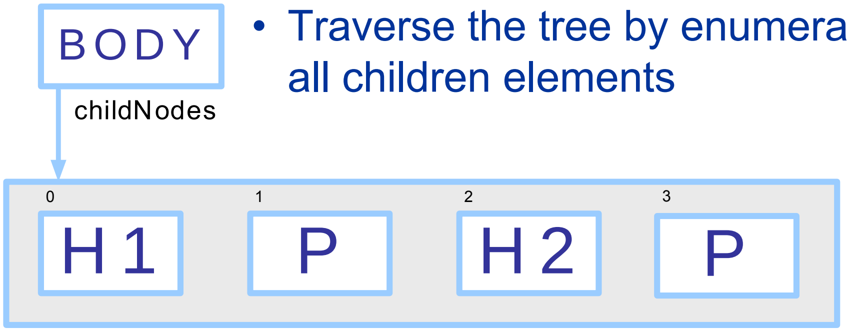
Summary

- Traverse the tree following pointers between elements



childNodes

- Traverse the tree by enumerating all children elements



Create DOM Elements

```
document.createElement(tagName)
```

```
document.createTextNode(text)
```

```
node.cloneNode()
```

– Clone an individual element.

```
node.cloneNode(true)
```

– Clone an element and all of its descendents.

- **Note:** The new nodes are not connected to the document.

Link Elements into the Tree

```
node.appendChild(new)
```

- Add new as the lastChild of node

```
node.insertBefore(new, sibling)
```

– Add to the children of node before sibling.

```
node.insertBefore(new, node.firstChild)
```

```
node.replaceChild(new, old)
```

– Swap the old child element with new.

```
old.parentNode.replaceChild(new, old)
```

Delete DOM Elements

node.removeChild(*old*)

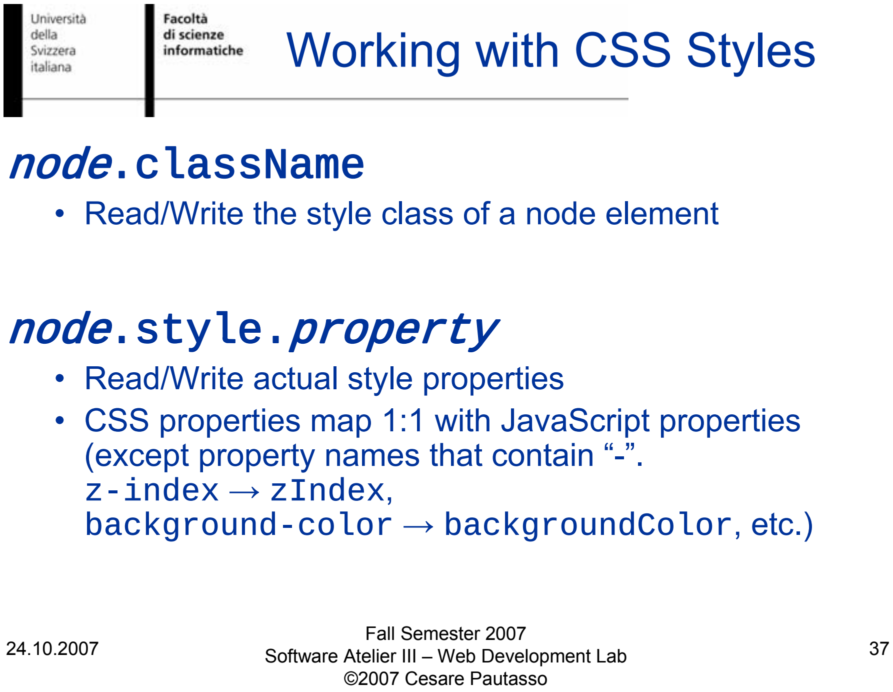
- – It returns the old node.
- – (Be sure to remove any event handlers to avoid memory leaks).

old.parentNode.removeChild(*old*)

innerHTML

- The W3C standard does not provide access to the HTML parser.
- All browsers implement Microsoft's `innerHTML` property.

- Two options available to create DOM sub-trees:
  - – Work with DOM methods (createElement, appendChild)
  - – Pass the raw HTML string to the parent node using innerHTML



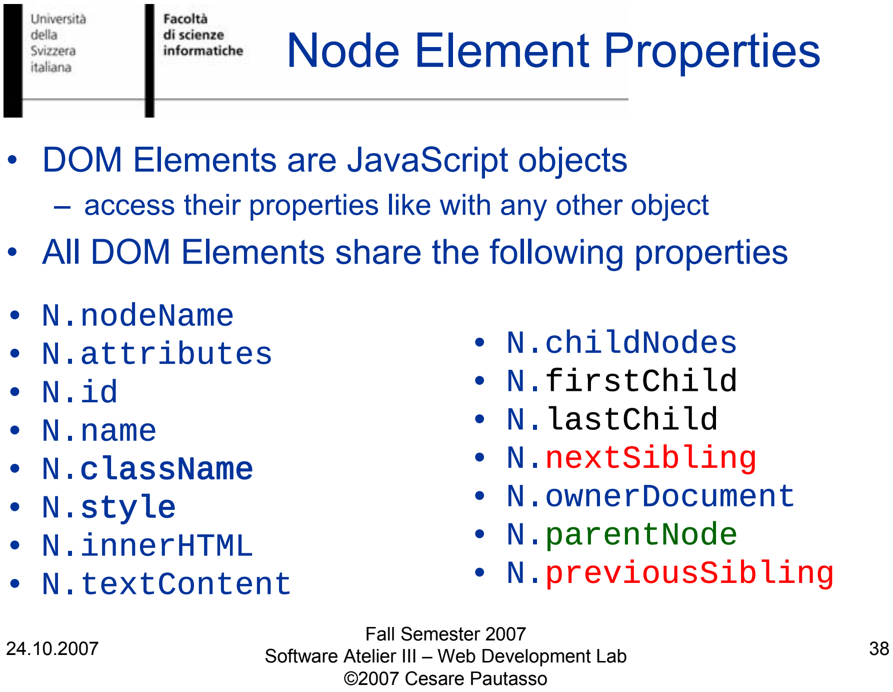
Working with CSS Styles

node.className

- Read/Write the style class of a node element

node.style.*property*

- Read/Write actual style properties
- CSS properties map 1:1 with JavaScript properties (except property names that contain “-”.
  `z-index` → `zIndex`,
  `background-color` → `backgroundColor`, etc.)



Node Element Properties

- DOM Elements are JavaScript objects
  - access their properties like with any other object
- All DOM Elements share the following properties

- `N.nodeName`
- `N.attributes`
- `N.id`
- `N.name`
- **`N.className`**
- **`N.style`**
- `N.innerHTML`
- `N.textContent`

- `N.childNodes`
- `N.firstChild`
- `N.lastChild`
- `N.nextSibling`
- `N.ownerDocument`
- `N.parentNode`
- `N.previousSibling`

Università della Svizzera italiana

Facoltà di scienze informatiche

References

- Follow the links on Moodle for an in-depth video tutorial on JavaScript and DOM
by Douglas Crockford, Yahoo
- Danny Goodman, Michael Morrison, **JavaScript Bible**, 6th Edition, Wiley, April 2007
- David Flanagan, **JavaScript: The Definitive Guide**, Fifth Edition, O'Reilly, August 2006
- Mark Pilgrim, **Greasemonkey Hacks**, O'Reilly