WYKŁAD 2
DOM
Document Object Model

MODEL DOM

Model DOM (z ang. Document Object Model) czyli obiektowy model dokumentu pozwala na odwoływanie się do elementów witryny oraz ich zmienianie, dodawanie i usuwanie.

HTML DOM definiuje:

- Elementy HTML jako obiekty;
- Właściwości wszystkich elementów HTML;
- Metody elementów HTML;
- Zdarzenia dla elementów HTML.

SCHEMAT DOM

DOM jest modelem hierarchicznym i udostępnia zestaw obiektów odzwierciedlających dokument HTML oraz elementy okna przeglądarki.

Jest to struktura drzewiasta, w której elementy niższego poziomu są węzłami elementów wyższego poziomu.

SCHEMAT DOM

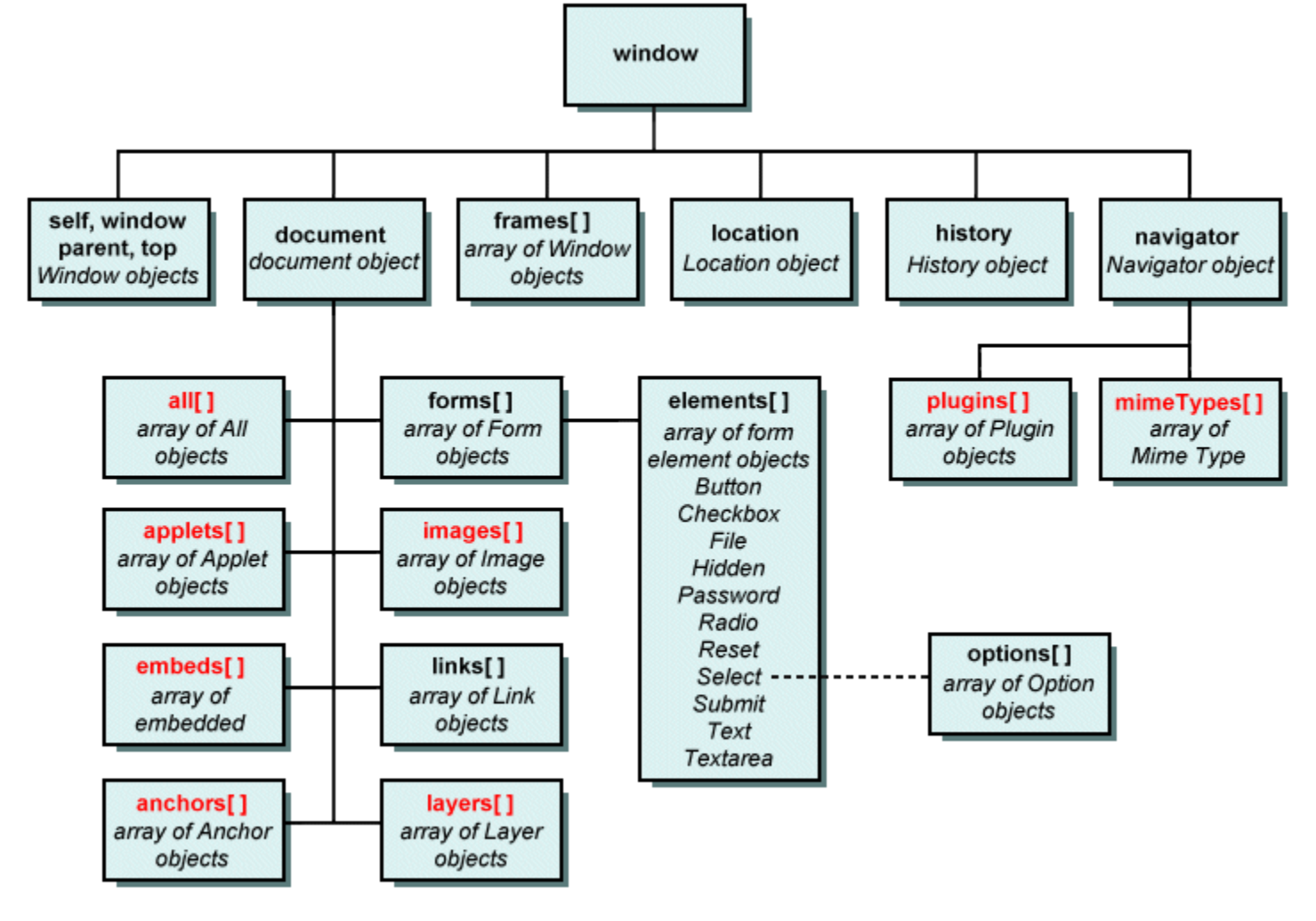
Głównym, globalnym obiektem DOM przeglądarki jest *window*. W tym obiekcie przechowywane są wszystkie globalne zmienne i funkcje.

W nim jest także obiekt *document*, który reprezentuje całą stronę www.

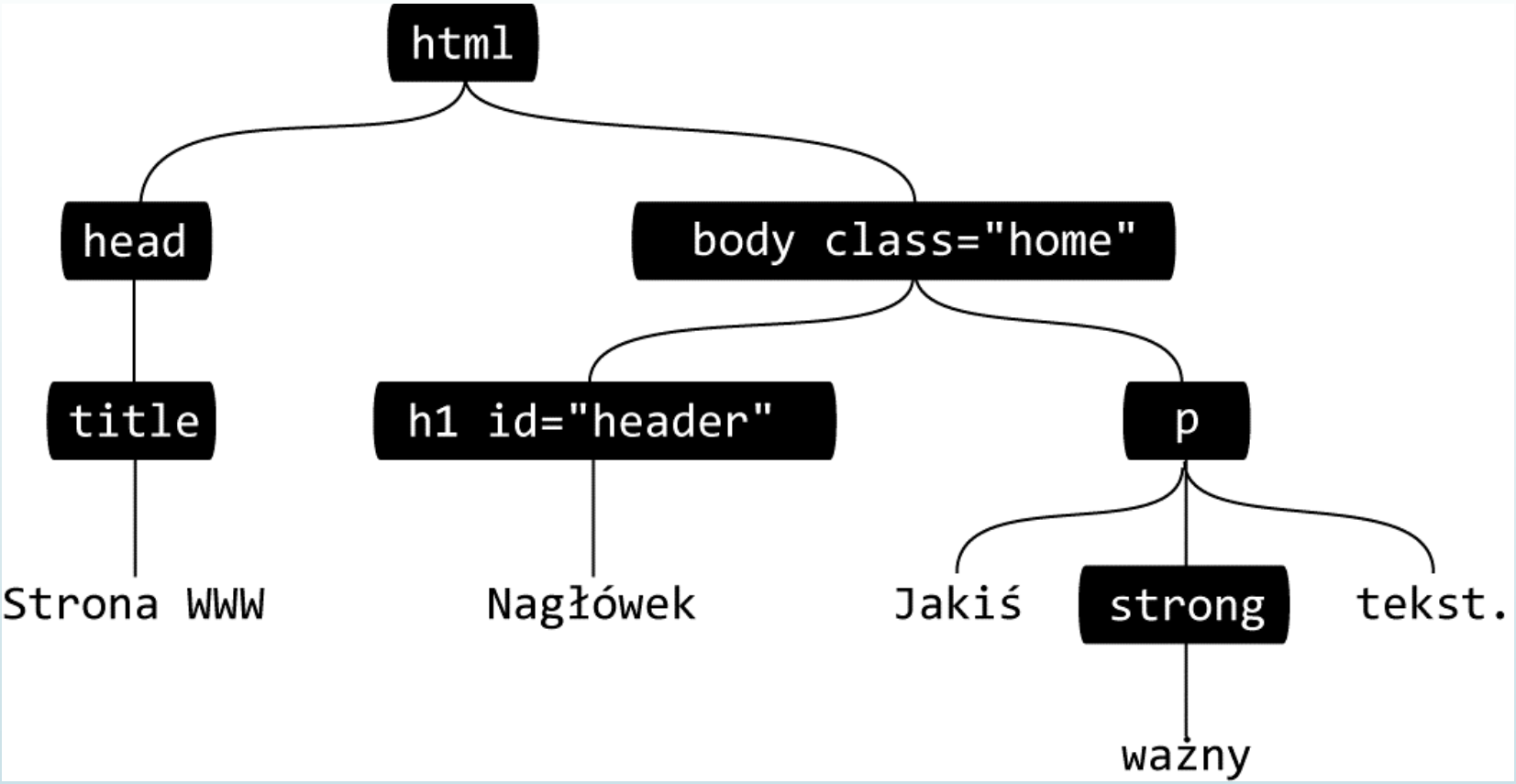
W oparciu o DOM JavaScript może:

- Dodawać, zmieniać i usuwać wszystkie elementy HTML i ich atrybuty na stronie;
- Zmieniać wszystkie style i klasy CSS na stronie;
- Dodawać i reagować na wszystkie zdarzenia HTML na stronie;

SCHEMAT DOM


window
self, window
parent, top
Window objects
document
document object
frames[ ]
array of Window objects
location
Location object
history
History object
navigator
Navigator object
all[ ]
array of All objects
forms[ ]
array of Form objects
elements[ ]
array of form element objects
Button
Checkbox
File
Hidden
Password
Radio
Reset
Select
Submit
Text
Textarea
plugins[ ]
array of Plugin objects
mimeTypes[ ]
array of Mime Type
applets[ ]
array of Applet objects
images[ ]
array of Image objects
embeds[ ]
array of embedded
links[ ]
array of Link objects
options[ ]
array of Option objects
anchors[ ]
array of Anchor objects
layers[ ]
array of Layer objects

SCHEMAT DOM


html
head
body class="home"
title
h1 id="header"
p
Strona WWW
Nagłówek
Jakiś
strong
tekst.
ważny

PORUSZANIE PO DOM HTML

Wszystkie elementy HTML na stronie są zawarte w obiekcie *document* drzewa DOM HTML tworzą tzw. kolekcje elementów.

Kolekcje elementów - tablice zawierające informacje o wszystkich elementach danego typu.

Np.: kolekcja ***forms*** zawiera w sobie wszystkie formularze i odwołujemy się do niej poprzez `document.forms`.

Ilość elementów w kolekcji odczytuje się za pomocą właściwości *lenght*

```
var formsCounter = document.forms.length;
```

PORUSZANIE PO DOM HTML

Kolekcja *document.forms* zawiera wszystkie formularze na stronie. Jeśli formularz ma atrybut *name*, to można go wybrać po nazwie w kolekcji *document.forms*.

```
<form name="form1"></form>
<form name="form2"></form>
```

Do pierwszego z formularzy można się odwołać poprzez instrukcje:

```
document.forms["form1"]
document.forms.form1
document.forms[0]
```

PORUSZANIE PO DOM HTML

Wewnątrz obiektu formularza można odwoływać się do jego kontrolek po nazwach za pośrednictwem kolekcji *elements* lub bezpośrednio po nazwie:

```
<input type=”text” name=”elName”>
```

```
document.forms[0].texts['elName']
```

```
document.forms[0].elName
```

Żeby nie powtarzać w kółko document.forms... można sobie przypisać obiekt formularza do zmiennej tymczasowej:

```
var f = document.forms[0];
```

```
f.elName.value=”Agnieszka”;
```

PODSTAWOWE METODY I WŁASNOŚCI

Metoda getElementById(id)

Jeśli element ma identyfikator, to można go znaleźć za pomocą *getElementById(id)*. Jest to najczęściej używany sposób dostępu do elementu.

Własność innerHTML

Najprostszy sposób na pobranie lub zastąpienie zawartości elementu.

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

Własność *innerHTML* może być użyta do każdego elementu HTML, także <html> i <body>.

ODNAJDYWANIE ELEMENTÓW HTML

- **document.getElementById(id)** – wykorzystuje element ID

PRZED:

```
<p id=”demo”>…</p>          // …
<p class=”full”>Akapit</p>  // Akapit
<script>
document.getElementById(”demo”).innerHTML=”Lepszy akapit”;
</script>
```

PO:

```
<p id=”demo”>…</p>          // Lepszy akapit
<p class=”full”>Akapit</p>  // Akapit
```

ODNAJDYWANIE ELEMENTÓW HTML

- **document.getElementsByTagName(name)** - wykorzystuje znacznik;

PRZED:

```
<p id=”demo”>…</p>          // Lepszy akapit
<p class=”full”>Akapit</p>  // Akapit
<script>
document.getElementByTagName(”p”).innerHTML=”Zwykły akapit”;
</script>
```

PO:

```
<p id=”demo”>…</p>          // Zwykły akapit
<p class=”full”>Akapit</p>  // Zwykły akapit
```

ODNAJDYWANIE ELEMENTÓW HTML

- **document.getElementsByClassName(name)** – wykorzystuje klasę;

PRZED:

```
<p id=”demo”>…</p>          // Lepszy akapit
<p class=”full”>Akapit</p>  // Akapit
<script>
document.getElementByClassName(”full”).innerHTML=”Jestem najlepszy!”;
</script>
```

PO:

```
<p id=”demo”>…</p>          // Zwykły akapit
<p class=”full”>Akapit</p>  // Jestem najlepszy!
```

ZMIANA ELEMENTÓW HTML

- element.innerHTML – zmienia zawartość elementu HTML;
- element.attribute - zmienia wartość atrybutu elementu HTML;

```
document.getElementById(”demo”).value=”Brukselka”;
document.getElementById(”demo”).className=”title”;
```

- element.setAttribute(attribute, value) - zmienia wartość atrybutu elementu HTML;
- element.style.property – zmienia styl elementu HTML;

```
document.getElementById(”demo”).style.width=”200px”;
document.getElementById(”demo”).style.borderColor =”#ff0000”;
```

ZMIANA ELEMENTÓW HTML

element.style.property oraz element.className

Dzięki możliwości modyfikowania stylu danego elementu możliwe jest uzyskanie ciekawych efektów:

- Ukrywanie elementów: `element.style.display = 'none'`
- Zmiana pozycji elementów: `element.style.left = x + 'px';`

Przy używaniu nazw styli wykorzystuje się również notację wielbłądzią (camel case).

```
background-color = backgroundColor
float = cssFloat
```

Zaleca się jednak wykorzystywanie własności className, w celu oddzielenia kodu JavaScript od wyglądu strony.

DZIĘKUJĘ ZA UWAGĘ