3.8. Obiekty DOM

DOM (ang. *Document Object Model*) to sposób reprezentacji złożonych dokumentów XML i HTML w postaci modelu obiektowego.

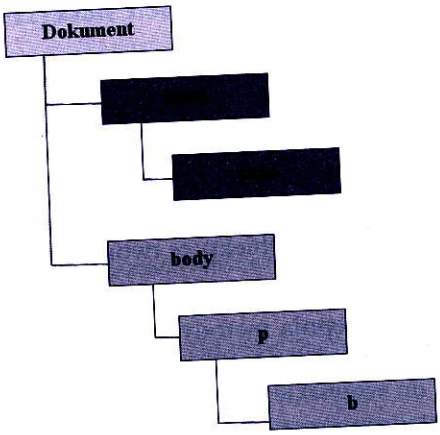
Gdy kod strony jest wczytywany do przeglądarki, przeglądarka zamienia ciąg znaków na stronę internetową. Informacje na temat interpretacji kodu HTML przeglądarka przechowuje w elementach będących obiektami (są to np. informacje o tym, które elementy przedstawić w postaci nagłówków, paragrafów itp.). Obiekty te tworzą obiektowy model dokumentu.

DOM opisuje hierarchię obiektów na stronie oraz udostępnia metody i właściwości, które umożliwiają manipulowanie nimi. W tej hierarchii na samej górze znajduje się okno przeglądarki, czyli obiekt `window`. Zawiera on wszystkie inne obiekty, funkcje i właściwości strony. W oknie znajduje się obiekt `document`, czyli otwarta strona internetowa. W obiekcie `document` znajdują się obiekty strony. W skryptach definiujemy różne działania związane z istniejącymi obiektami, czyli manipulujemy przez skrypty obiektami strony internetowej. Dzięki skryptom można wczytać nową stronę do przeglądarki, zmienić elementy dokumentu, otwierać okna lub modyfikować tekst na stronie. Dzięki DOM język JavaScript staje się narzędziem tworzenia dynamicznych stron internetowych.

Przykład 3.59

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Tytuł strony</title>
</head>
<body>
<p>Moja strona <b>internetowa</b></p>
</body>
</html>
```

Podany w przykładzie dokument można rozrysować w postaci drzewa (rysunek 3.5). Na samej górze jest dokument HTML, niżej znajdują się węzły (`nodes`).



Rysunek 3.5. Hierarchia obiektów przykładowej strony internetowej

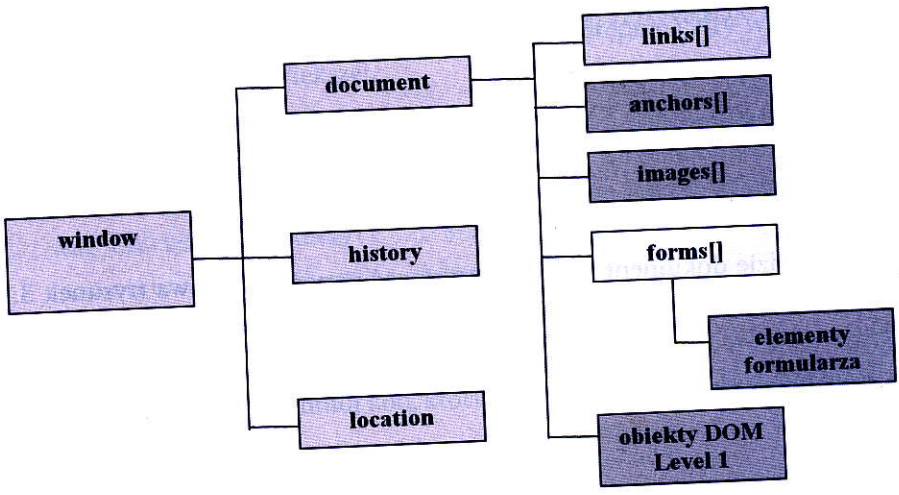
3.8.1. Hierarchia obiektów DOM

Odwołując się do obiektu, należy używać nazw obiektów nadrzędnych oddzielonych kropkami, po których następuje nazwa wybranego obiektu.

Przykład 3.60

```
window.document.link1
```

Wycinek hierarchii DOM z najważniejszymi obiektami strony internetowej zosta pokazany na rysunku 3.6.



Rysunek 3.6. Hierarchia obiektów DOM

3.8.2. Obiekty przeglądarki

Obiekt window

Obiektem nadrzędnym dla wszystkich obiektów jest obiekt `window`, który zawiera okno przeglądarki. W danej chwili może istnieć wiele obiektów `window`. Każdy z nich reprezentuje otwarte okno przeglądarki. Odwołanie do jego właściwości lub metod nie wymaga podania nazwy obiektu. Tworzony jest automatycznie podczas otwierania okna przeglądarki. Do otwierania nowego okna używamy metody `open()`. Jako parametry metody występują adres URL otwieranej strony oraz nazwa wewnętrzna okna (nie mylić z nazwą wyświetlaną przez przeglądarkę zdefiniowaną metatagiem `<title></title>`):

```
window.open('http://helion.pl','Wydawnictwo');
```

Do określania rozmiaru okna mogą być używane właściwości:

- `window.innerHeight` — wysokość okna przeglądarki,
- `window.innerWidth` — szerokość okna przeglądarki.

Do zamknięcia okna używana jest metoda `close()`.

Obiekt document

Obiekt `document` reprezentuje stronę internetową (dokument HTML). Jest on potomkiem obiektu `window`. Za pomocą polecenia `window.document` można odwołać się do bieżącego dokumentu. Można to zrobić również za pomocą polecenia `document`. Odwołanie nastąpi do bieżącego dokumentu w bieżącym oknie. Jeżeli zostało otwarte kilka okien, to aby określić, do którego dokumentu powinno nastąpić odwołanie, należy podać nazwę okna i nazwę dokumentu.

Informacje o bieżącym dokumencie otrzymamy, odwołując się do właściwości i metod obiektu `document`.

- `document.URL` — zwraca adres URL dokumentu jako ciąg tekstu,
- `document.title` — zwraca tytuł strony zdefiniowany w znaczniku `<title>`,
- `document.lastModified` — zwraca datę ostatniej modyfikacji strony,
- `document.bgColor` — określa kolor tła dokumentu ustawianego atrybutem `bgcolor` znacznika `<body>`,
- `document.fgColor` — określa kolor pierwszego planu dokumentu ustawianego atrybutem `text` znacznika `<body>`,
- `document.linkColor` — określa kolor łącza w dokumencie ustawianego atrybutem `link`,
- `document.alinkColor` — określa kolor łącza w dokumencie ustawianego atrybutem `alink`,
- `document.vlinkColor` — określa kolor łącza w dokumencie ustawianego atrybutem `vlink`,
- `document.cookie` — ustawia lub odczytuje `cookie` dla dokumentu.

Do odwołania się do elementu strony służą metody `getElementById()` oraz `getElementByTagName()`. Metoda `getElementById()` używana jest, gdy element, do którego się odwołujemy, posiada atrybut `id`, natomiast metodę `getElementByTagName()` wykorzystujemy do pobrania kolekcji zawierającej elementy danego typu.

Przykład 3.61

```
<p id="tekst1">Skrypty języka JavaScript</p>
<p>Dokument HTML</p>
<h2>Model dokumentu DOM</h2>
```

Odwołanie w skrypcie do tak zdefiniowanych elementów może mieć postać:

- `document.getElementById("tekst1")` — odwołanie do akapitu z atrybutem `id="tekst1"`,
- `document.getElementByTagName(p)` — odwołanie do kolekcji akapitów,
- `document.getElementByTagName(p)[0]` — odwołanie do pierwszego akapitu w kolekcji.

Używając metody `getElementById()`, należy pamiętać, że jest to metoda obiektu `document`, dlatego dostęp do niej jest możliwy tylko za pomocą tego obiektu. Odwołanie do elementu strony będzie możliwe tylko wtedy, gdy temu elementowi zostanie nadany atrybut `id`.

Przykład 3.62

```
<input type="button" id="klik" value="Kliknij!"/>
<script type="text/javascript">

var b = document.getElementById("klik");
alert(b.value);

</script>
```

Możliwość odwołania się do elementu strony jest wykorzystywana do zmiany jej zawartości. Zawartość elementu strony można odczytać i zmienić, używając właściwości `innerHTML`. Właściwość tę posiada każdy element strony internetowej. Określa ona wartość przypisaną elementowi. Właściwość `innerHTML` może być użyta tylko razem z metodą `getElementById()` i tylko dla elementów, dla których został zdefiniowany identyfikator (`id`).

Przykład 3.63

```
<html>
<head>
<title>Tekst</title>
```

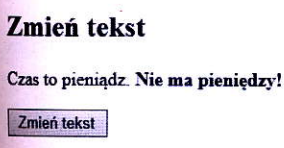
```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h2>Zmień tekst</h2>
<script type='text/javascript'>

function zmien_Tekst()
{
  document.getElementById('blok').innerHTML = 'Nie ma czasu!';
}
</script>

<p>Czas to pieniądz. <b id='blok'>Nie ma pieniędzy!</b></p>
<input type="button" onclick="zmien_Tekst()" value = 'Zmień tekst'/>

</body>
</html>
```

Po kliknięciu przycisku efektem wykonania kodu będzie zmiana wyświetlanego tekstu (rysunek 3.7).



Rysunek 3.7. Możliwość zmiany tekstu wyświetlanego na stronie

Przykład 3.64

W przykładzie 3.49 zostało zdefiniowane wyświetlanie daty i czasu. Zmiana czasu następowała po odświeżeniu strony. Zmodyfikujemy powstały kod tak, aby czas wyświetlany na stronie był zawsze aktualny.

```
<html>
<head>
<title> Mój zegar</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h2>Uciekający czas</h2>
```

```
<div id='zegar' style='width: 100px; border: solid 2px #1d330a;'></div>

<script type='text/javascript'>

var timerID = null;
var timerRunning = false;

function stopclock()
{
    if(timerRunning)
        clearTimeout(timerID)
    timerRunning = false;
}

function startclock()
{
    stopclock();
    showtime();
}

function showtime()
{
  var data_n = new Date();
  var godz = data_n.getHours();
  var min = data_n.getMinutes();
  var sek = data_n.getSeconds();

  var czas = "" + ( godz);
  czas  += ((min < 10) ? ":0" : ":") + min;
  czas  += ((sek < 10) ? ":0" : ":") + sek;

  document.getElementById('zegar').innerHTML = czas;
  timerID = setTimeout("showtime()",1000);
  timerRunning = true;
}
```

```
</script>

<script>startclock();</script>

 </script>
</body>
</html>
```

Metoda `getElementById()` odwołuje się do wcześniej zdefiniowanego identyfikatora `'zegar'`. W przykładzie za pomocą właściwości `innerHTML` identyfikatorowi `'zegar'` przypisana została wartość zmiennej `'czas'`, która zawiera bieżący czas. Użyty operator warunkowy *((min < 10) ? ":0" : ":") + min;* ma trzy argumenty: *test* po którym występuje znak `?` oraz *wyrażenie1* i *wyrażenie2* oddzielone znakiem `:`. Jeżeli wartość testu jest *true* to wynikiem jest *wyrażenie1*, w przeciwnym razie wynikiem jest *wyrażenie2*. Wynik interpretacji kodu został pokazany na rysunku 3.8.

Uciekający czas

21:37:01

Rysunek 3.8. Wyświetlanie na stronie aktualnego czasu

Metodą obiektu `document` jest również metoda `document.write()`, która wyświetla na stronie internetowej w oknie dokumentu podany tekst.

Zadanie 3.10

Wykorzystując arkusze CSS oraz skrypty z poprzednich przykładów, zdefiniuj kod HTML, który na stronie internetowej wyświetli bieżący czas w postaci zegara cyfrowego.

Zadanie 3.11

Utwórz i dodaj do dokumentu HTML skrypt, który wyświetli datę ostatniej modyfikacji tego dokumentu.

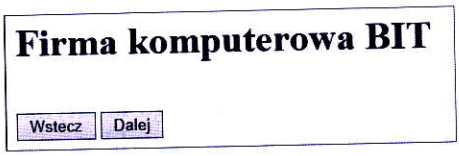
Obiekt history

Drugim obiektem potomnym w stosunku do obiektu `window` jest obiekt `history`. Zawiera on informację o odwiedzanych adresach URL. Posiada zdefiniowane metody, które pozwalają na przejście do wcześniej odwiedzanych stron.

- `history.go()` — otwiera określony adres URL z listy historii. W nawiasach należy podać liczbę dodatnią lub ujemną, określającą, o ile do przodu lub do tyłu należy przemieścić się, aby otworzyć określony adres, np. `history.go(3)`.
- `history.back()` — otwiera poprzedni adres URL z listy historii.
- `history.forward()` — otwiera następny adres URL z listy historii, jeżeli taki istnieje.

Obiekt `history` posiada jedną właściwość, `history.length`, która zawiera informację o długości listy historii.

Wykorzystując metody `back()` i `forward()`, można utworzyć skrypty, które wyświetlą na stronie przyciski *Wstecz* oraz *Dalej*, umożliwiające poruszanie się w przeglądarce po odwiedzanych stronach (rysunek 3.9).



Rysunek 3.9. Wyświetlenie za pomocą metody back() i forward() przycisków Wstecz i Dalej

Przykład 3.65

```
<html>
<head>
<title>Przyciski</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<h1>Firma komputerowa BIT</h1><br>
<input type="button" onclick="history.back()" value="Wstecz">
<input type="button" onclick="history.forward()" value="Dalej">
</body>
</html>
```

Zadanie 3.12

Dla dokumentu HTML utwórz skrypt, dzięki któremu po kliknięciu przycisku *Data modyfikacji* będzie możliwe wyświetlenie daty modyfikacji bieżącej strony internetowej.

Obiekt location

Trzecim obiektem potomnym w stosunku do obiektu `window` jest obiekt `location`. Zawiera on informację o bieżącym adresie dokumentu otwartego w oknie. Za pomocą właściwości tego obiektu można uzyskać pełną informację o adresie URL, można też uzyskać dostęp do jego fragmentów.

- `location.href` — zawiera cały adres URL,
- `location.protocol` — zawiera protokół,
- `location.hostname` — zawiera nazwę hosta,
- `location.port` — zawiera numer portu,

- `location.pathname` — zawiera nazwę pliku ze ścieżką,
- `location.search` — zawiera zapytanie, jeżeli znajduje się ono w adresie,
- `location.hash` — zawiera nazwę kotwicy, jeżeli kotwica występuje w adresie.

Przykład 3.66

```
window.location.href = "http://www.helion.pl"
```

UWAGA

Właściwość `location.href` zawiera ten sam adres co właściwość `document.URL`. Jednak właściwości `document.URL` nie można modyfikować. W celu otwarcia nowej strony należy posługiwać się właściwością `location.href`.

Obiekt `location` posiada dwie metody:

- `location.reload()` — odświeża (ponownie wczytuje) bieżący dokument. Jeżeli dodany zostanie parametr `true`, odświeżanie odbędzie się niezależnie od tego, czy dokument uległ zmianie, czy nie.
- `location.replace()` — zastępuje bieżący adres URL nowym.

Zadanie 3.13

Dla dokumentu HTML utwórz skrypt, który wyświetli nazwę pliku oraz ścieżkę dostępu do bieżącej strony internetowej.

Obiekt link

Obiektem potomnym w stosunku do obiektu `document` jest obiekt `link`. Zawiera on informację o łączu do określonego adresu. Obiekty `link` są zapisane w tablicy `links`. W dokumencie może wystąpić wiele obiektów `link`. Każdy z nich jest zapisany jako oddzielny element tablicy.

Właściwość tablicy `document.links.length` określa liczbę linków na stronie.

Każdy obiekt `link` zapisany w tablicy ma listę właściwości określających adres URL. Są to właściwości takie same jak dla obiektu `location`. Można się do nich odwoływać, podając numer w tablicy i nazwę właściwości.

Przykład 3.67

```
link1 = links[0].href;
```

Obiekt anchor

Kolejnym obiektem potomnym w stosunku do obiektu `document` jest obiekt `anchor`. Reprezentuje on kotwicę w bieżącym dokumencie.

UWAGA

Kotwica określa zdefiniowaną lokalizację w dokumencie HTML, do której można się przenieść.

Podobnie jak łącza, kotwice są zapisywane w tablicy o nazwie `anchors`. Każdy element jest obiektem `anchor`.

Właściwość tablicy `document.anchors.length` określa liczbę elementów kotwi na stronie.

Obiekt form

Obiektem potomnym w stosunku do obiektu `document` jest również obiekt `for` Zawiera on informacje dotyczące formularzy występujących w dokumencie HTM Obiekty form są zapisane w tablicy `forms`. Ponieważ w dokumencie może wystąp wiele formularzy, każdy z nich jest zapisany jako oddzielny element tablicy. Do wyb nego formularza można odwoływać się przez indeks lub przez nazwę, wpisując w kodz polecenie `document.forms[0]` lub `document.forms['Form1']`. Lepszą meto odwołania do formularza jest wykorzystanie metody `getElementById()`, np. `d cument.getElementById('form1')`.

Przykład 3.68

```
<body>
<form id="form1" name="form1" action="mailto:nauka@gmail.com" method="post
...
</form>
<script type="text/javascript">
document.forms['form1']
...
</script>
</body>
```

Jeżeli został zastosowany atrybut `name` (jak w przykładzie 3.69), to do formular można odwołać się również w ten sposób: `document.form1`.

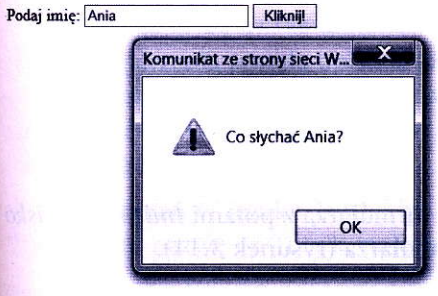
Każdy element formularza jest obiektem, więc posiada właściwości. Jedną z nich je właściwość `value`, która przechowuje bieżącą wartość elementu.

Przykład 3.69

```
<html>
<head>
<title>Co słychać</title>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<form id="form1" name="form1" action="mailto:nauka@gmail.com" method="post">
Podaj imię: <input type="text" name="imie"/>
<button onclick="Witaj()"> Kliknij!</button>
</form>
<script type="text/javascript">
function Witaj()
{
var imie = document.forms['form1'].imie.value;
alert('Co słychać ' + imie + '?');
}
</script>
</body>
</html>
```

Podany kod definiuje formularz z polem *Imie* i przyciskiem *Kliknij*. Gdy dla przycisku wystąpi zdarzenie `onclick` (kliknięcie przycisku), zostanie uruchomiona funkcja `Witaj()`, która pobierze wartość elementu `imie` (`var imie = document.forms['form1'].imie.value;`) i wyświetli ją w oknie z komunikatem (rysunek 3.10).



Rysunek 3.10. Pobrane z formularza imię zostało wyświetlone w oknie z komunikatem

Elementy formularza tworzą tablicę. Dostęp do nich jest możliwy przez odwołanie się do kolejnych elementów tej tablicy (`elements[i]`). Podobnie jak do całego formularza, do jego elementów można odwoływać się przez indeks lub przez nazwę:

```
document.forms['form1'].elements[0]
document.forms['form1'].elements['nazwa']
document.forms['form1'].nazwa
```

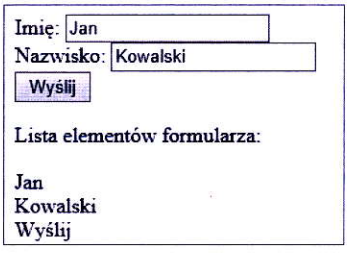
lub za pomocą metody `getElementById()`:

```
document.getElementById('Imie').value
```

Przykład 3.70

```
<html>
<head>
<title>Lista formularza</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<form id="form1" action="mailto:nauka@gmail.com" method="post">
Imię: <input type="text" name="imie" value="Jan"/><br />
Nazwisko: <input type="text" name="nazwisko" value="Kowalski" /><br />
<input type="submit" value="Wyślij" />
</form>
<p>Lista elementów formularza:</p>
<script type="text/javascript">
var x=document.getElementById("form1");
for (var i=0;i<x.length;i++){
document.write(x.elements[i].value);
document.write("<br />");
}
</script>
</body>
</html>
```

Wynikiem wykonania kodu będzie wyświetlenie formularza z polami *Imię* i *Nazwisko* oraz przycisku *Wyślij* wraz z listą elementów formularza (rysunek 3.11).



Rysunek 3.11. Wyświetlenie na stronie formularza wraz z listą jego elementów