Aplikacje internetowe i rozproszone

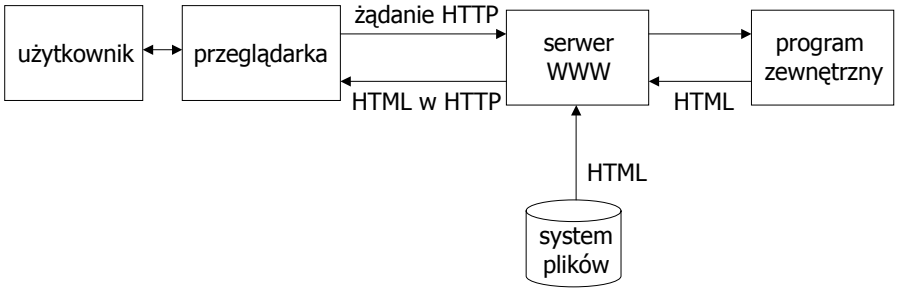
dr inż. Maciej Zakrzewicz
mzakrz@cs.put.poznan.pl
http://www.cs.put.poznan.pl/~mzakrz/

Program wykładów

- Architektury aplikacji intra- i internetowych (WWW, HTTP, HTTPS, WAP)
- Technologie prezentacji danych w sieci Internet (HTML, CSS, XML, XSL, VRML, WML, JavaScript)
- Przetwarzanie dokumentów XML w aplikacjach Java (parsery DOM, SAX)
- Technologie budowy aplikacji pracujących po stronie serwera WWW
  - CGI, Perl/DBI, SSI, ASP, PHP/MySQL
  - serwlety Java, sesje HTTP, zmienne Cookies
  - JavaServer Pages, biblioteki znaczników
  - dostęp aplikacji Java do baz danych: JDBC, SQLJ
- Zagrożenia bezpieczeństwa aplikacji internetowych
- Technologie budowy komponentowych aplikacji rozproszonych (CORBA, Enterprise JavaBeans, WebServices)
- Projektowanie aplikacji J2EE (architektura Model-View-Controller, Struts)
- Architektura aplikacji mobilnych J2ME
- Zaawansowane rozwiązania WWW (web caching, web searching, web mining)

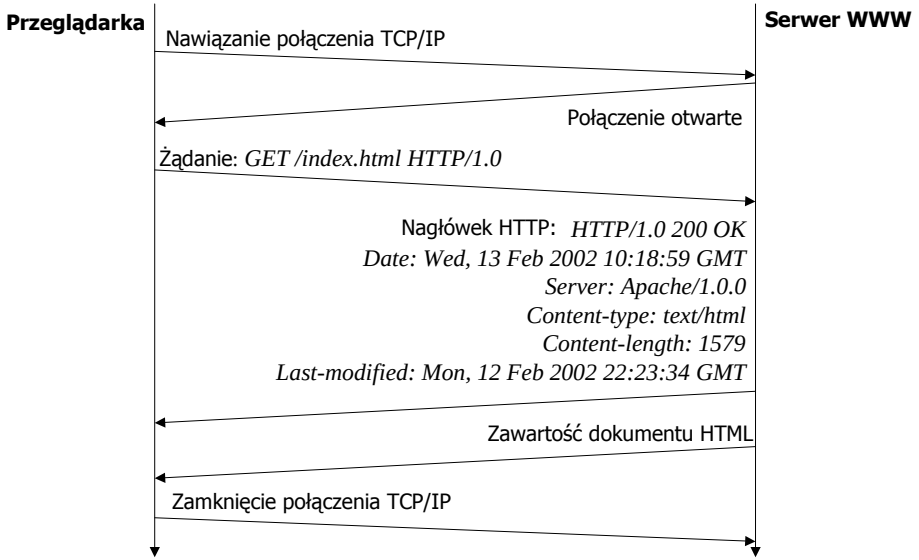
Architektury aplikacji intra- i internetowych

Podstawy usługi WWW



Użytkownik korzysta z programu przeglądarki (klienta HTTP) aby przyłączyć się do serwera WWW (serwera HTTP), podając URL opisujący żądany dokument HTML. Serwer WWW pobiera dokument i przesyła do użytkownika. Serwer WWW może też uruchomić zewnętrzny program, który wygeneruje dokument HTML dla użytkownika.

Protokół HTTP



Przesyłanie innych typów danych poprzez HTTP

- Protokół HTTP może być wykorzystany do przesyłania dokumentów HTML, jak i danych innych typów, np. obrazów GIF, dokumentów XML, plików wideo AVI, itp.
- W nagłówku HTTP znajduje się pole *Content-type*, informujące przeglądarkę o formacie przesyłanego obiektu (opis zgodnie ze specyfikacją MIME)

Content-type	**Format**
text/html	Dokument HTML
application/postscript	Plik PostScript
application/zip	Skompresowany plik ZIP
audio/x-wav	Plik dźwiękowy WAV
image/gif	Obraz graficzny GIF
image/jpeg	Obraz graficzny JPEG
text/plain	Plik tekstowy
video/x-msvideo	Plik wideo AVI

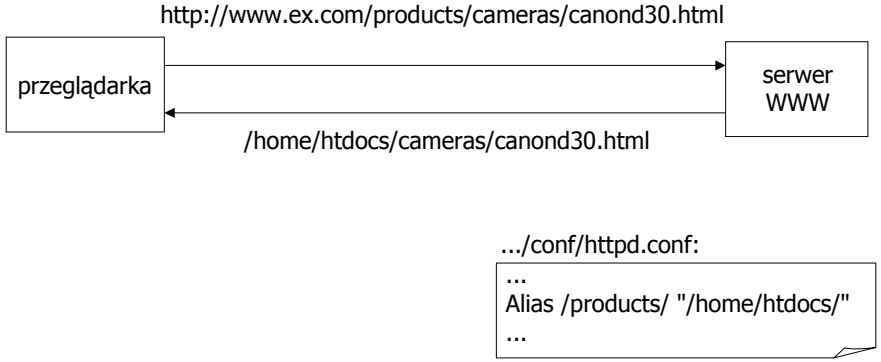
Adresy URL

- Żądanie wysyłane przez przeglądarkę zawiera adres URL
- Uniform Resource Locator (URL) jest wskaźnikiem do zasobu w sieci Internet, np.:
  - http://www.foo.org/info/welcome.txt
- URL można rozbić na następujące części:
  - <protocol>://<user>:<password>@<host>:<port>/<path>
    - <protocol> - wykorzystywany protokół (np. HTTP, HTTPS, itd.)
    - <user> - opcjonalna nazwa użytkownika (np. dla FTP)
    - <password> - opcjonalne hasło dla użytkownika
    - <host> - nazwa domenowa lub adres IP serwera
    - <port> - numer portu IP (jeśli nie zostanie podany, przyjęta będzie watość domyślna dla protokołu)
    - <path> - katalog/nazwa pliku

Katalogi logiczne a katalogi fizyczne

- Ścieżki dostępu umieszczane w adresach URL nazywane są katalogami logicznymi (wirtualnymi); ich nazewnictwo nie musi być powiązane z nazewnictwem stosowanym w systemie plików serwera WWW
- W celu znalezienia żądanego pliku, serwer WWW odwzorowuje otrzymany katalog logiczny w faktyczną ścieżkę dostępu w systemie plików (katalog fizyczny)
- Odwzorowanie katalogów odbywa się w oparciu o pliki konfiguracyjne serwera WWW

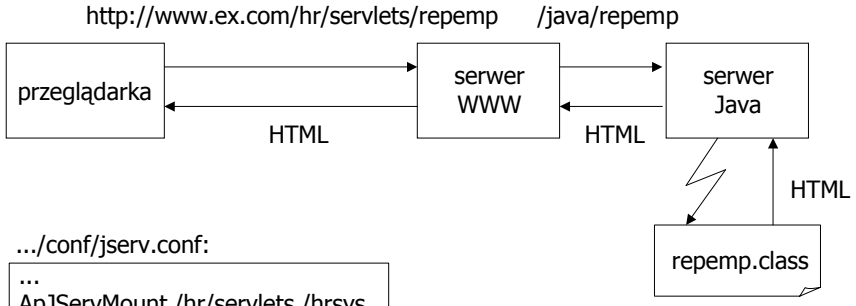
Katalogi logiczne a katalogi fizyczne
Przykład



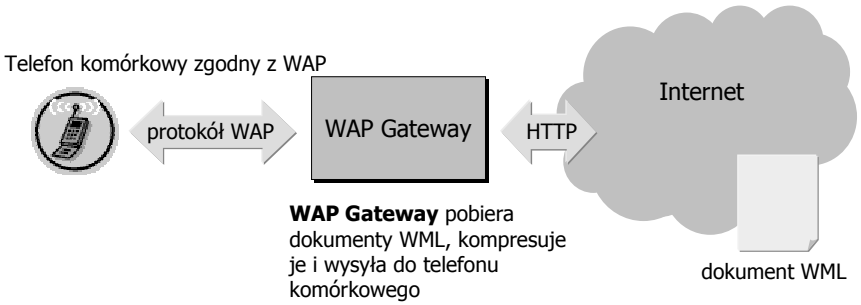
Aplikacje WWW pracujące po stronie serwera

- Dokumenty, które serwer WWW dostarcza przeglądarce, mogą być generowane programowo w odpowiedzi na żądanie użytkownika; przykłady:
  - zawartość dokumentu HTML jest uzależniona od parametrów przekazanych przez użytkownika
  - dane zawarte w dokumencie HTML ulegają częstym zmianom
  - dokument HTML zawiera dane pochodzące z innych systemów informatycznych (np. baz danych)
- Istnieje wiele technologii tworzenia aplikacji WWW pracujących po stronie serwera, np. CGI, PHP, ASP, serwlety Java, itd.

Przykład aplikacji WWW: serwlety Java



Architektura WAP



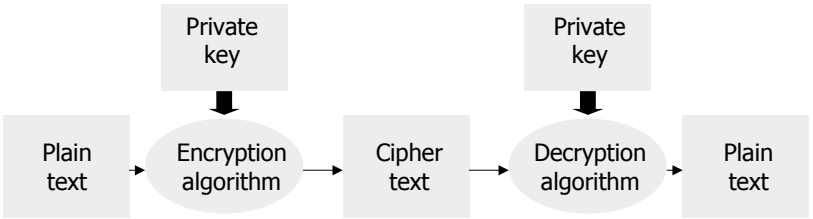
WAP (Wireless Application Protocol) to standard dostępu telefonów komórkowych do Internetu; WAP pozwala urządzeniom mobilnym na korzystanie z rozmaitych źródeł informacyjnych (głównie WWW)

Dokumenty przesyłane przez WAP są tworzone w języku **WML** (Wireless Markup Language); istnieje możliwość tworzenia funkcji interakcji w, podobnym do JavaScript, języku **WMLScript**.

Bezpieczeństwo transmisji

Szyfrowanie symetryczne

- Ten sam klucz musi być w posiadaniu nadawcy i odbiorcy wiadomości
- Rodzaje algorytmów szyfrujących:
  - **blokowe** - otrzymuje blok danych źródłowych, szyfruje go i zwraca jako wynik działania
  - **strumieniowe** - szyfruje pojedyncze bity danych źródłowych



Data Encryption Standard (DES)

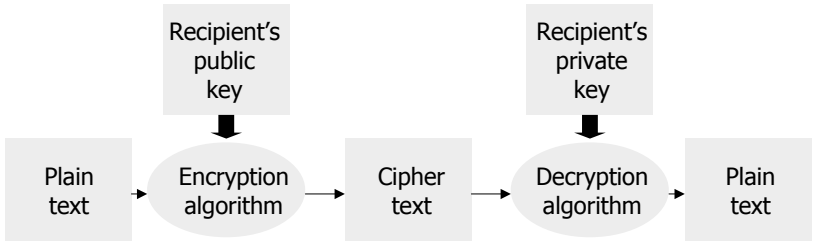
- Algorytm blokowy, operujący na 64-bitowych blokach danych
- Dwa tryby pracy:
  - **ECB** (Electronic Code Book): każdy blok 64-bitowy jest niezaleznie szyfrowany,
  - **CBC** (Cipher Block Chaining): każdy blok 64-bitowy jest poddawany operacji XOR z poprzednim blokiem, a następnie szyfrowany; gwarantuje to, że taki sam blok pojawiający się w wielu miejscach, będzie za każdym razem posiadał inną postać zaszyfrowaną
- **Triple-DES** - szyfruje dane przy pomocy DES, wynik szyfrowany jest ponownie przy użyciu innego klucza, a następnie jeszcze raz, przy pomocy trzeciego klucza; klucze są niezależne

Inne algorytmy szyfrowania symetrycznego

- **IDEA** (International Data Encryption Algorithm): algorytm blokowy; klucz 128-bitowy; standard europejski (ETH Zurich) z 1990; konkuruje z DES w zakresie szybkości i bezpieczeństwa
- **CAST**: algorytm blokowy; klucze 128-bitowe (w przeszłości Europa 40-bitowe); używany przez Northern Telecom
- **Skipjack**: algorytm blokowy rozwijany przez NSA (National Security Agency); klucze 80-bitowe; implementacja przy wykorzystaniu układu scalonego Capstone; szczegóły owiane tajemnicą
- **RC2**: algorytm blokowy, **RC4**: algorytm strumieniowy; klucze 40-bitowe; rozwijane przez RSA Data Securities; celem było osiągnięcie szybkości porównywalnej lub większej niż DES

Szyfrowanie asymetryczne

- Stosowane dwa typy kluczy:
  - **klucz publiczny** jest publicznie znany i wykorzystywany przez nadawców wiadomości adresowanych do właściciela klucza
  - **klucz prywatny** jest znany tylko właścicielowi i służy do odczytywania zaszyfrowanych wiadomości
- Oba klucze mogą być zamienione miejscami, co stanowi podstawę algorytmów podpisu cyfrowego



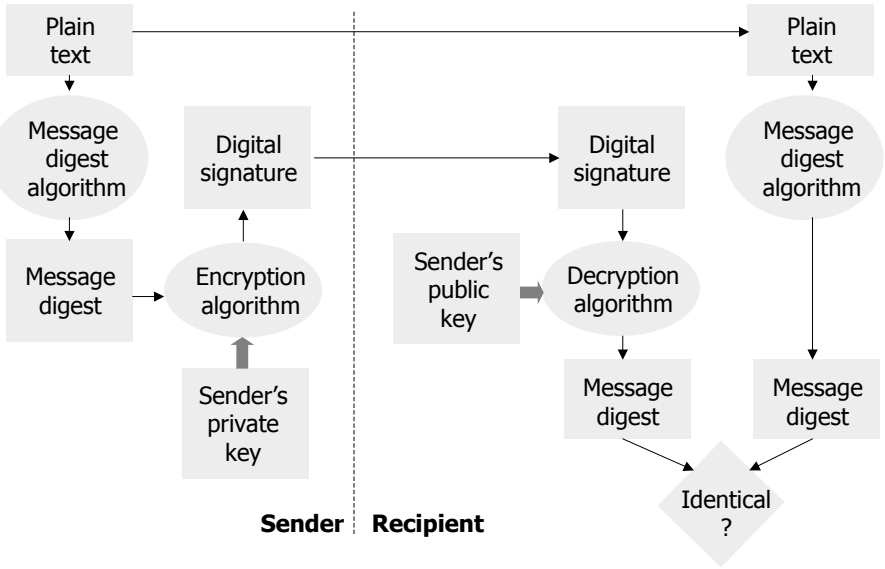
Algorytmy szyfrowania asymetrycznego

- **RSA** (R. Rivest, A. Shamir, L. Adleman - założyciele RSA Data Security): podstawą matematyczną jest założenie, że bardzo trudno jest znaleźć składniki dużej liczby powstałej w wyniku przemnożenia dwóch liczb pierwszych; klucze 1024-bitowe traktowane jako bardzo dobre
- **PKCS** (Public Key Cryptography Standards): zaproponowany przez RSA i konsorcjum firm: Microsoft, Apple, Digital, Sun Microsystems, Lotus itd.; rodzina rozmaitych algorytmów kryptograficznych
- **DSS** (Digital Signature Standard): algorytm zalecany przez rząd USA; długość kluczy od 512 do 1024 bitów; przeznaczony wyłącznie do generowania podpisów cyfrowych

Algorytmy kryptograficznych sum kontrolnych (Message Digest)

- Algorytmy kryptograficznych sum kontrolnych stanową, wraz z szyfrowaniem asymetrycznym, podstawę dla implementacji podpisów cyfrowych
- Algorytm kryptograficznych sum kontrolnych implementuje funkcję haszową, która dla danych wejściowych zwraca ciąg znakowy o stałej długości; użyta funkcja jest nieodwracalna
- Algorytmy **MD2**, **MD4** i **MD5** generują wartości 128-bitowe; MD2 jest najwolniejszy, MD4 - najszybszy
- **SHA** (Secure Hash Algorithm) generuje wartości 160-bitowe; przyjęty przez rząd USA; mniej bezpieczny niż MD4

Podpisy cyfrowe



Praktyczne zastsosowania algorytmów kryptograficznych

- Zwykle zarówno szyfrowanie symetryczne, jak i asymetryczne są wykorzystywane współbieżnie
- Szyfrowanie asymetryczne jest dużo wolniejsze, aniżeli symetryczne, w związku z czym w praktyce generuje się klucze symetryczne i wymienia się je przez bezpieczny kanał bazujący na szyfrowaniu asymetrycznym
- Standardy bezpiecznej komunikacji klient-serwer:
  - **SSL** (Secure Sockets Layer): wykorzystuje RSA, RC2, RC4, IDEA, DES, Triple-DES, MD5
  - **PCT** (Private Communications Technology): wykorzystuje RSA, DES, Triple-DES, RC2, RC4, DSA
- Protokół HTTPS (S-HTTP) to HTTP szyfrowany przez SSL

Techniki prezentacji danych w sieci Internet

Formaty dokumentów WWW

- Dokument, jaki otrzymuje przeglądarka od serwera WWW może być zapisany w różnych formatach
- Język **HTML** służy do definiowania *treści* i *układu graficznego* dokumentów udostępnianych przez serwery WWW
- Język **CSS** służy do definiowania *układu graficznego* dla dokumentów HTML
- Język **XML** służy do definiowania *treści* dokumentów udostępnianych przez serwery WWW w sposób umożliwiający ich przetwarzanie automatyczne
- Język **XSL** służy do definiowania *układu graficznego* dla dokumentów XML
- Język **VRML** służy do definiowania *trójwymiarowych scen graficznych*
- Język **WML** służy do definiowania treści i układu graficznego dokumentów udostępnianych przez protokół WAP

Format HTML

- Język HTML służy do definiowania treści i układu graficznego dokumentów udostępnianych przez serwery WWW
- Dokument HTML to dokument tekstowy, w którym umieszczone są rozkazy dla przeglądarki, dotyczące sposobu wyświetlenia danego fragmentu tekstu; rozkazy te nazywane są znacznikami (tags)
- Większość znaczników HTML występuje w parach; każda para posiada element otwierający i zamykający; znacznik otwierający jest otoczony "<" i ">", a zamykający - "</" i ">"; niektóre znaczniki są pojedyncze
- Znaczniki mogą posiadać dodatkowe atrybuty sterujące, umieszczane wewnątrz "<" i ">" w formacie *atrybut='wartość'*
- Komentarze umieszcza się w "<!--" i "-->"

Struktura dokumentu HTML

```
<HTML>
 <HEAD>
  <TITLE>
   <!-- tytuł dokumentu -->
  </TITLE>
  <!-- pozostałe informacje nagłówkowe --->
 </HEAD>
 <BODY>
  <!-- treść dokumentu -->
 </BODY>
</HTML>
```

Najważniejsze znaczniki HTML (1)

<Hy> *Tekst nagłówka* **</Hy>**	Tekst nagłówka; sześć poziomów, (y=1..6); 1 - największy; nagłówki wyświetlane są pogrubioną i powiększoną czcionką
<P> *Tekst paragrafu* **</P>**	Paragraf – blok tekstu zakończony przejściem do nowego wiersza; znaki ENTER są w HTML nieznaczące
** **	Wymusza przejście do nowego wiersza bez wprowadzania odstępu pionowego (<P> dodaje odstęp)
<HR>	Znacznik tworzy poziomą linię o szerokości okna przeglądarki; wykorzystywany do rozdzielania sekcji dokumentu
<B>	Tekst wytłuszczony
<I>	Tekst pochylony
<A HREF="*URL***">** *Tekst łącznika* **</A>**	Umieszczenie łącznika do innego dokumentu; *URL* do adres wskazywanego dokumentu, „*Tekst łącznika*" jest wyświetlany na ekranie
<IMG SRC="*URL***">**	Umieszczenie obrazu graficznego (XBM, GIF lub JPEG), znajdującego się pod podanym adresem URL

Najważniejsze znaczniki HTML (2)

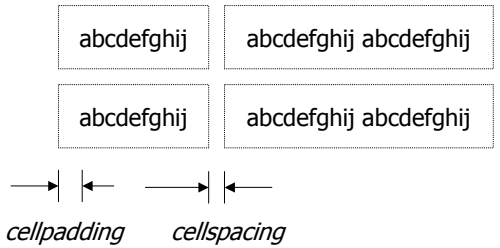
<TABLE></TABLE>	Definicja tabelki HTML; liczba kolumn i wierszy będzie wynikać z liczebności definicji wewnątrz <TABLE></TABLE>
<TR></TR>	Definicja jednego wiersza tabelki HTML; zagnieżdżone w <TABLE></TABLE>
<TD>*zawartość***</TD>**	Definicja jednej komórki w wierszu tabelki HTML; w komórce wyświetlona zostanie *„zawartość"*

Najczęściej stosowane atrybuty znaczników (1)

width np.<HR WIDTH='10%'>	Szerokość elementu w pikslach lub procentach szerokości elementu otaczającego (okna przeglądarki, komórki tabeli)
align np.<P ALIGN='CENTER'>	Wyrównanie poziome elementów wewnątrz znacznika; LEFT, RIGHT lub CENTER
valign np.<TD VALIGN='TOP'>	Wyrównanie pionowe elementów wewnątrz znacznika; TOP, BOTTOM lub CENTER
style np.<P STYLE='{color: red}'>	Definicja stylu graficznego w formacie CSS
colspan np.<TD COLSPAN='3'>	Stosowane w tabelach; komórka tabeli ma powstać ze złączenia kilku kolumn
rowspan np.<TR ROWSPAN='2'>	Stosowane w tabelach; komórka tabeli ma powstać ze złączenia kilku wierszy
border np.<TABLE BORDER='1'>	Określa grubość obramowania elementu; BORDER=0 oznacza brak obramowania
bgcolor np.<TABLE BGCOLOR='red'>	Kolor tła elementu

Najczęściej stosowane atrybuty znaczników (2)

class np.<P CLASS='NICE_PAR'>	Stosowane przy współpracy z CSS; określa nazwę klasy wizualnej, do której należy znacznik
cellspacing np.<TABLE CELLSPACING='4'>	Stosowane w tabelach; wyrażony w pikslach prześwit pomiędzy komórkami tabeli; wartość 0 oznacza brak prześwitu
cellpadding np.<TABLE CELLPADDING='10'>	Stosowane w tabelach; wyrażony w pikslach margines, o jaki komórka jest większa od swojej zawartości

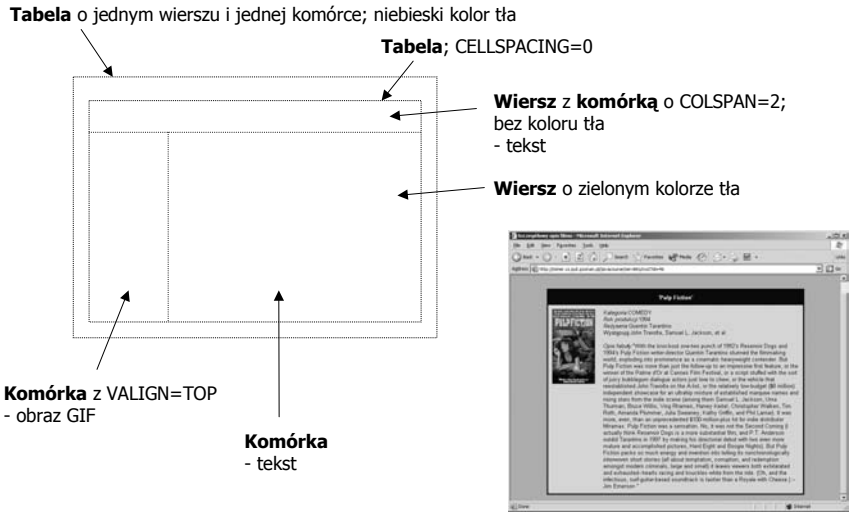


Przykład prostej tabelki

```
<table border=1>
 <tr>
  <td>Rok</td>
  <td>Kwota</td>
 <tr>
  <td>2001</td>
  <td>189,982.000</td>
 <tr>
  <td>2002</td>
  <td>250,321.000</td>
</table>
```

Rok	Kwota
2001	189,982.000
2002	250,321.000

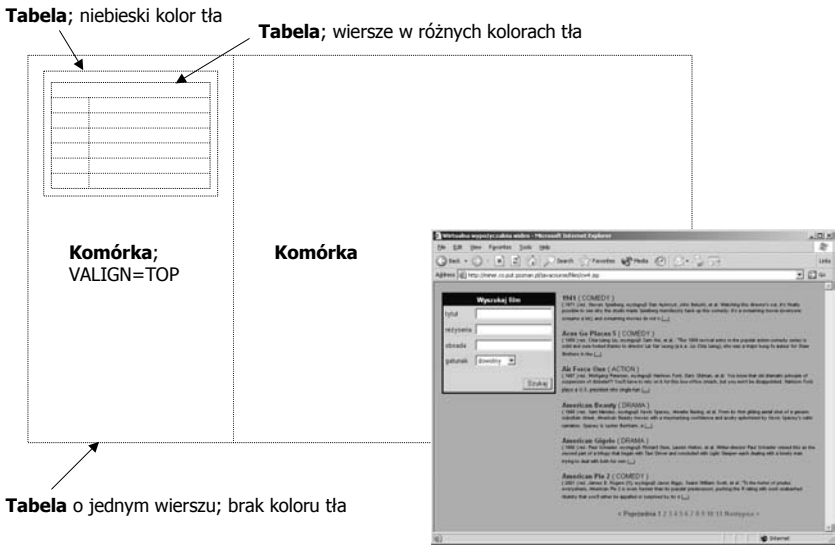
Inne zastosowania tabelek HTML (1)



Inne zastosowania tabelek HTML (1) kod źródłowy

```
<table class=table_border width=80%>
 <tr>
  <td>
   <table cellspacing=0 cellpadding=10 width=100%>
    <tr>
     <td colspan=2 align=center>'Pulp Fiction'
     </td>
    <tr class=row_odd>
     <td valign=top align=center>
      <img width=100 src='cw2img?id=46' border=1>
     </td>
     <td valign=top>
      <i>Kategoria:</i>...
     </td>
   </table>
  </td>
</table>
```

Inne zastosowania tabelek HTML (2)



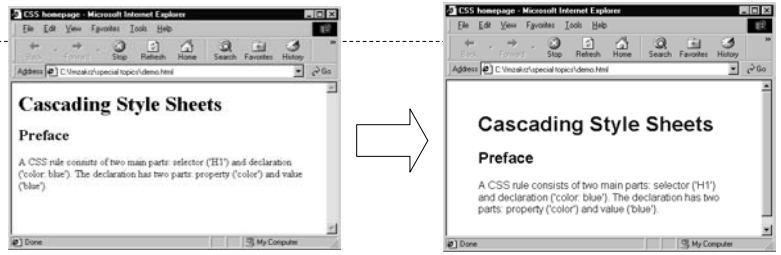
CSS

Arkusze reguł stylistycznych CSS

- Arkusz reguł stylistycznych stylu (ang. **style sheet**) jest kolekcją reguł formatujących, które mogą odnosić się do wielu dokumentów HTML; spełnia on funkcję wzorca, pozwalając na zapewnienie takiego samego wyglądu wszystkich wystąpień danego elementu
- Powszechnie zaakceptowanym i ustandaryzowanym mechanizmem specyfikacji stylów dla HTML jest CSS (standardy CSS1 i nowszy CSS2 - będący rozszerzeniem CSS1)
- CSS umożliwia znacznie dokładniejszą kontrolę sposobu wyświetlania elementów, aniżeli sam HTML
- Nie wszystkie mechanizmy CSS1/CSS2 są zaimplementowane w dostępnych obecnie przeglądarkach

CSS w HTML - Przykład

```
<HTML>
  <HEAD>
  <TITLE>CSS homepage</TITLE>
    <STYLE type="text/css">
      H1 { color: blue }
      BODY { font-family: "Gill Sans", sans-serif; font-size: 12pt;
             margin: 3em }
    </STYLE>
  </HEAD>
  <BODY>
    <H1>Cascading Style Sheets</H1>
    <H2>Preface</H2>
    <P> A CSS rule consists of two main parts: selector
    ('H1') and declaration ('color: blue'). The
    declaration has two parts: property ('color') and
    value ('blue').  </P>
  </BODY>
</HTML>
```



Dołączanie reguł do dokumentu HTML

- Reguły stylistyczne w oddzielnym pliku, wskazanym znacznikiem `<LINK REL="stylesheet">`
- Reguły stylistyczne zagnieżdżone w dokumencie za pomocą znacznika `<STYLE>` w sekcji HEAD
- Atrybut STYLE elementów w sekcji BODY, np. `<P STYLE='{color: blue}'>`

Reguły stylistyczne CSS

- Reguła CSS zawiera dwie części: **selektor** (np. 'H1') i **deklarację** (np. 'color: blue'); deklaracja ma dwie części: właściwość (np. 'color') i wartość (np. 'blue')
- **Selektor** składa się z jednego lub więcej prostych selektorów połączonych łącznikami; **łącznikami** są: odstępy, ">" i "+"
- Gdy wszystkie warunki selektora są spełnione dla danego elementu, mówimy, że element *spełnia* selektor

Formaty selektorów reguł CSS

`*`	spełniony przez każdy element
`E`	spełniony przez każdy element <E> (np. BODY)
`E F`	spełniony przez każdy element F zagnieżdżony wewnątrz elementu E (np. P I)
`E > F`	spełniony przez każdy element F zagnieżdżony bezpośrednio wewnątrz elementu E (np. P I)
`E + F`	spełniony przez każdy element F, który następuje bezpośrednio za elementem E
`A:link`	spełniony przez każdy link <A>, który nie został jeszcze odwiedzony
`A:visited`	spełniony przez każdy link <A>, który został już odwiedzony
`E:active`	spełniony przez każdy element E, który jest właśnie naciskany myszką
`E:hover`	spełniony przez każdy element E, nad którym właśnie przesuwa się wskaźnik myszki
`E[atr="val"]`	spełniony przez te elementy E, które posiadają atrybut *atr* o wartości *val*
`E.val`	spełniony przez te elementy E, które posiadają atrybut CLASS o wartości *val*

Przykład wykorzystania reguł CSS (1)


```
body           {background-color: orange}
*              {font-family: Sans-serif}
*.row_even     {background-color: lightblue}
*.row_odd      {background-color: lightgreen}
*.row_caption  {background-color: darkblue;
                color: white; font-weight: bold}
*.table_border {background-color: darkblue}
*.button_even  {background-color: lightblue; border-color: lightblue}
*.button_odd   {background-color: lightgreen; border-color: lightgreen}
*.result_title {color: darkblue; font-weight: bold}
*.result_story {font-size: smaller}
*.page_header  {font-family: "Tahoma",Sans-serif; font-weight:800;
                letter-spacing: 8; color: white}
a.active_link  {color: green; font-weight: bold; text-decoration: none}
a:hover        {color: red}
```

Przykład wykorzystania reguł CSS (2)

```
<table class=table_border width=100%>
 <tr><td><table cellspacing=0 cellpadding=4 width=100%>
   <tr class=row_caption>
    <td colspan=2 align=center>Wyszukaj film</td>
   <tr class=row_odd>
    <td>tytuł</td><td><input type='text' name=p_title></td>
   <tr class=row_even>
    <td>reżyseria</td><td><input type='text' name=p_dir></td>
   <tr class=row_odd>
    <td>obsada</td><td><input type='text' name=p_cast></td>
   <tr class=row_even>
    <td>gatunek</td><td><select name=p_category>
        <option>ACTION</option>
        <option SELECTED >COMEDY</option>
        <option>FAMILY</option>
        <option>SCI-FI</option>
        </select></td>
    <tr class=row_odd>
      <td colspan=2 align=right><br>
       <input type='submit' value='Szukaj' class=button_even></td>
  </table>
 </td>
</table>
```

Deskryptory CSS: marginesy

- Właściwości marginesów (obszaru pomiędzy obramowaniem elementu a elementem zewnętrznym): **margin-top, margin-right, margin-bottom, margin-left, margin**; np.:

```
BODY { margin: 2em }         /* all margins set to 2em */
BODY { margin: 1em 2em }     /* top & bottom = 1em, right & left = 2em */
BODY { margin: 1em 2em 3em } /* top=1em, right=2em, bottom=3em, left=2em */
```

ostatnia z powyższych reguł jest równoważna poniższej:

```
BODY {
  margin-top: 1em;
  margin-right: 2em;
  margin-bottom: 3em;
  margin-left: 2em;
}
```

Uwaga: '1em' = rozmiar używanej czcionki

Deskryptory CSS: wypełnienie

- Własności wypełnienia (obszaru pomiędzy treścią elementu a ewentualnym obramowaniem) - właściwości: **padding-top, padding-right, padding-bottom, padding-left,** i **padding**; np.:

```
H1 {
  background: white;
  padding: 1em 2em;
}
```

Powyższy przykład specyfikuje padding w pionie na '1em' (padding-top and padding-bottom) oraz padding w poziomie na '2em' (padding-right and padding-left)

Deskryptory CSS: obramowanie

- Szerokość krawędzi: **border-top-width, border-right-width, border-bottom-width, border-left-width** i **border-width**; wartości: **thin, medium, thick**, lub podane jawnie; np. `H1 {border-width: thin}`
- Kolor krawędzi: **border-top-color, border-right-color, border-bottom-color, border-left-color** i **border-color**; np. `H1 {border-color: red}`
- Styl krawędzi: **border-top-style, border-right-style, border-bottom-style, border-left-style**, and **border-style**; wartości: **none, dotted, dashed, solid, double, groove, ridge, inset, outset**; np. `H1 {border-style: solid dotted}`

Deskryptory CSS: specyfikacja czcionki

- Właściwość **font-family**; wartości: **serif, sans-serif, cursive, fantasy, monospace**, i inne
- Właściwość **font-style**; wartości : **normal, italic, oblique**
- Właściwość **font-variant**; wartości : **normal, small-caps**
- Właściwość **font-weight**; wartości : **normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900**
- Właściwość **font-stretch**; wartości :**ultra-condensed, extra-condensed, condensed, semi-condensed, normal, semi-expanded, expanded, extra-expanded, ultra-expanded**
- Właściwość **font-size**; wartości : **xx-small, x-small, small, medium, large, x-large, xx-large, larger, smaller,** bezwzględny rozmiar czcionki, względny rozmiar czcionki (wyrażony procentowo)

Deskryptory CSS: właściwości tekstu

- Wcięcie: **text-indent**; np. `P {text-indent: 3em}`
- Wyrównanie: **text-align**; wartości: **left, right, center, justify**; np. `P {text-align: center}`
- Dekoracja: **text-decoration**; wartości: **none, underline, overline, line-through, blink**; np. `P {text-decoration: line-through}`
- Cień: **text-shadow**; np. `H1 {text-shadow: 0.2em 0.2em}`
- Odstępy: **letter-spacing** i **word-spacing**; np. `H1 {letter-spacing: 0.1em; word-spacing: 1em}`
- Wielkość liter: **text-transform**; wartości: **capitalize, uppercase, lowercase, none**

Deskryptory CSS: kolory i tło

- Kolor pierwszego planu: **color**; np.:

```
EM { color: red }              /* predefined color name */
EM { color: rgb(255,0,0) }     /* RGB range 0-255   */
```

- Tło: **background-color, background-image, background-repeat, background-attachment, background-position, background**; np.:

```
H1 { background-color: #F00 }

BODY { background-image: url("marble.gif") }

BODY { background: white url("pendant.gif"); background-
  repeat: repeat-y; background-position: center;}

BODY {
  background: red url("pendant.gif");  background-repeat:
  repeat-y; background-attachment: fixed;}
```

XML

Format XML

- Formalnie, **XML** stanowi podzbiór języka Standard Generalized Markup Language (**SGML**) (ISO 8879:1986)
- XML jest formatem służącym do opisywania strukturalnych danych, przeznaczonych do upowszechniania w sieci Internet
- Dokumenty XML posługują się znacznikami, zapisywanymi podobnie jak w HTML; jednak wszystkie znaczniki XML są definiowane przez programistę
- Dokumenty XML są łatwe w przetwarzaniu maszynowym; istnieje wiele bibliotek tzw. parserów XML, z których mogą korzystać programiści (Java, JavaScript, C++, itd.)

Przykładowy prosty dokument XML

```
<company>
 <department>
  <dname>Marketing</dname>
  <employees>
   <employee>Jones</employee>
   <employee>Smith</employee>
  </employees>
 </department>
 <department>
  <dname>Production</dname>
  <employees>
   <employee>Mitchell</employee>
   <employee>Barker</employee>
  </employees>
 </department>
</company>
```

Formalna struktura dokumentu XML

- Dokument XML może się składać z trzech części (pierwsze dwie są opcjonalne):
  - **Processing Instruction**, określa wersję XML, metodę kodowania znaków narodowych, itd., np.:
    `<?xml version= "1.0" encoding="ISO-8859-1" standalone="yes"?>`
  - **Document Type Declaration** (DTD), opisuje użyte znaczniki oraz ich gramatykę; DTD może znajdować się wewnątrz dokumentu lub w oddzielnym pliku, np.:
    `<!DOCTYPE company SYSTEM "http://foo.org/dtds/company.dtd">`
  - **Document Instance**, ciało dokumentu; uwaga: każdy znacznik musi być jawnie zamknięty; znaczniki pojedyncze wolno otwierać i zamykać w jednym kroku stosując zapis jak w przykładzie:

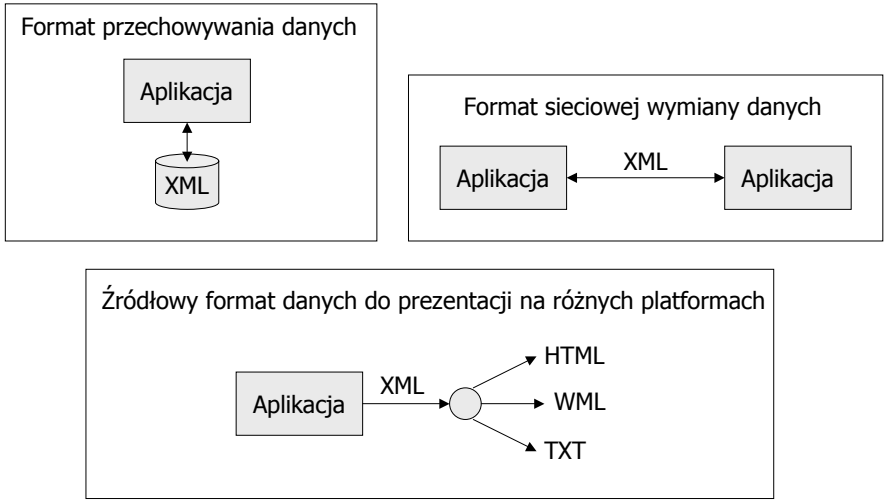
Przykładowy zapis DTD

```
<company>
<department>
<dname>Marketing</dname>
<employees>
<employee>Jones</employee>
<employee>Smith</employee>
</employees>
</department>
<department>
...
</department>
...
</company>
```

może wystąpić jeden lub wiele razy (**+**),
zero lub wiele (*****),
zero lub jeden (**?**)

```
<!DOCTYPE company [
<!ELEMENT company (department)+>
<!ELEMENT department (dname, employees)>
<!ELEMENT dname (#PCDATA)>
<!ELEMENT employees (employee)+>
<!ELEMENT employee (#PCDATA)> ]>
```

Motywacje: wybrane zastosowania XML

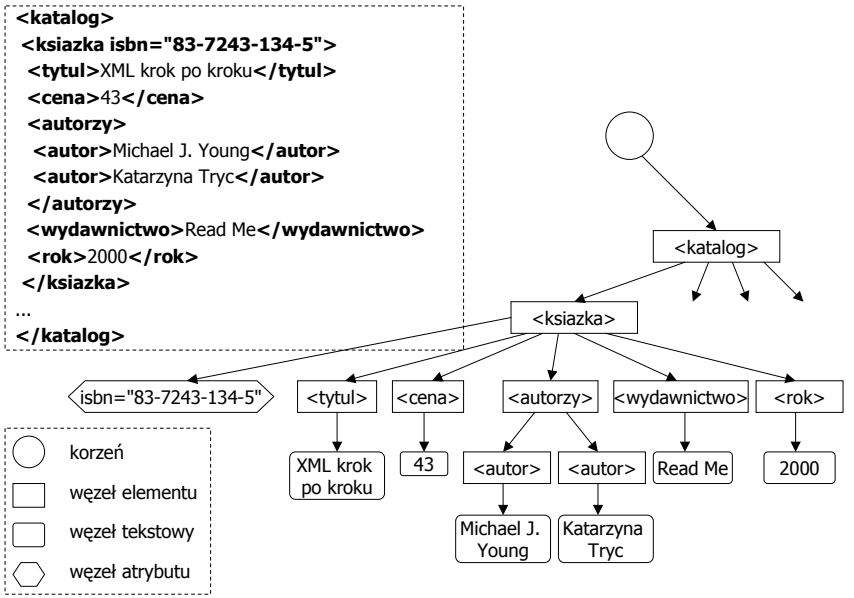


Drzewa DOM

Document Object Model (DOM)

- Document Object Model jest standardem modelowania dokumentów XML przy użyciu struktury drzewa – znaczniki XML i ich zawartość są modelowane przez węzły drzewa; zagnieżdżanie znaczników służy za podstawę do konstruowania hierarchii
- Document Object Model jest wykorzystywany jako forma reprezentacji dokumentów XML w pamięci komputera
- Transformacja dokumentu XML do postaci Document Object Model jest realizowana automatycznie przez parser DOM
- Implementacja, adresowanie i przeszukiwanie drzew Document Object Model mogą być realizowane przy użyciu biblioteki DOM API

Przykład struktury drzewa DOM



W3C DOM API: obiekt Node

- Obiekt klasy/typu **Node** reprezentuje węzeł w drzewie DOM (węzeł elementu, węzeł tekstowy, itd.)

Atrybuty (W3C)

attributes	tablica atrybutów węzła
childNodes	tablica węzłów potomnych
firstChild	pierwszy węzeł potomny
lastChild	ostatni węzeł potomny
nextSibling	prawy węzeł sąsiedni
nodeName	nazwa węzła
nodeType	identyfikator typu węzła
nodeValue	wartość węzła
parentNode	węzeł nadrzędny
previousSibling	lewy węzeł sąsiedni

Metody (W3C)

appendChild(n)	dołącza nowy węzeł jako ostatni węzeł potomny
cloneNode(b)	zwraca kopię węzła z/bez węzłami potomnymi
hasChildNodes()	zwraca prawdę, jeżeli węzeł zawiera węzły potomne
insertBefore(n,n)	dołącza nowy węzeł jako węzeł potomny przed wskazanym węzłem
removeChild(n)	usuwa wskazany węzeł potomny
replaceChild(n,n)	zamienia istniejący węzeł potomny z podanym węzłem

W3C DOM API: obiekt NodeList

- Obiekt klasy/typu **NodeList** reprezentuje zbiór obiektów typu Node

Atrybuty (W3C)

length	liczba elementów w zbiorze

Metody (W3C)

item(i)	zwraca element i-ty element zbioru

W3C DOM API: obiekt Document

- Obiekt klasy/typu **Document** modeluje całe drzewo DOM; wszystkie węzły drzewa są jego potomkami

Atrybuty (W3C)

documentElement	element najwyższego poziomu w dokumencie
doctype	DTD lub XML Schema dla dokumentu

Metody (W3C)

createAttribute(s)	tworzy nowy węzeł atrybutu
createComment(s)	tworzy nowy węzeł komentarza
createElement(s)	tworzy nowy element
createTextNode(s)	tworzy nowy węzeł tekstowy
getElementsByTagName(s)	zwraca zbiór węzłów o podanej nazwie

W3C DOM API: obiekt Element

- Obiekt klasy/typu **Element** modeluje węzeł reprezentujący znacznik XML

Atrybuty (W3C)

tagName	nazwa węzła

Metody (W3C)

getAttribute(s)	zwraca wartość podanego atrybutu
getAttributeNode(s)	zwraca węzeł podanego atrybutu
getElementsByTagName(s)	zwraca zbiór węzłów o podanej nazwie
removeAttribute(s)	usuwa wartość podanego atrybutu
removeAttributeNode(n)	usuwa podany węzeł atrybutu
setAttribute(s,s)	ustawia nową wartość atrybutu
setAttributeNode(n)	wstawia nowy węzeł atrybutu

W3C DOM API: obiekt Attr i Text

- Obiekt klasy/typu **Attr** reprezentuje atrybut znacznika XML w formie tzw. węzła atrybutu; obiekt Attr posiada ogólne atrybuty i metody klasy/typu Node plus poniższe:

Atrybuty (W3C)

name	nazwa atrybutu
specified	prawda oznacza, że wartość atrybutu jest ustawiona w dokumencie
value	wartość atrybutu

- Obiekt klasy/typu **Text** reprezentuje treść umieszczoną wewnątrz znacznika XML

Implementacja W3C DOM: Java

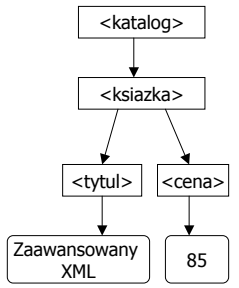
- Wszystkie typy DOM zostały zaimplementowane w języku Java jako interfejsy w pakiecie org.w3c.dom (posiadają nazwy jak w specyfikacji W3C) i jako klasy rzeczywiste w pakiecie oracle.xml.parser.v2 (posiadają nazwy z prefiksem XML)

Java: funkcje konstrukcji drzew DOM

- **createElement(String)** [interfejs Document] – tworzy nowy węzeł, reprezentujący znacznik o podanej nazwie; węzeł ten nie wchodzi jeszcze w skład drzewa dokumentu
- **createTextNode(String)** [interfejs Document] – tworzy nowy węzeł tekstowy; węzeł ten nie wchodzi jeszcze w skład drzewa dokumentu
- **appendChild(Node)** [interfejs Node] – dodaje nowy węzeł jako ostatni węzeł potomny
- **cloneNode(boolean)** [interfejs Node] – wykonuje kopię wskazanego węzła wraz z lub bez jego węzłów potomnych
- **removeChild(Node)** [interfejs Node] – odpina wskazany węzeł potomny od jego węzła nadrzędnego
- **replaceChild(Node, Node)** [interfejs Node] – odpina istniejący węzeł potomny i na jego miejscu umieszcza nowy węzeł potomny
- **setNodeValue(String)** [interfejs Node] – nadaje węzłowi wartość tekstową

Konstrukcja drzewa DOM w języku Java

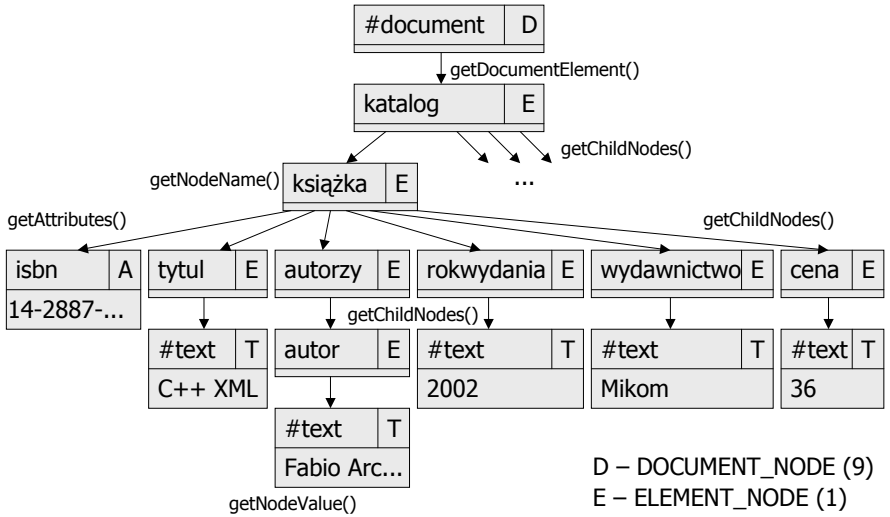
```
XMLDocument xmlDoc = new XMLDocument();

Node katalogNode = xmlDoc.createElement("katalog");
xmlDoc.appendChild(katalogNode);

Node ksiazkaNode = xmlDoc.createElement("ksiazka");
katalogNode.appendChild(ksiazkaNode);

Node tytulNode = xmlDoc.createElement("tytul");
ksiazkaNode.appendChild(tytulNode);

Node tytulText = xmlDoc.createTextNode("Zaawansowany XML");
tytulNode.appendChild(tytulText);

Node cenaText = xmlDoc.createTextNode("85");
tytulNode.appendChild(cenaText);
```



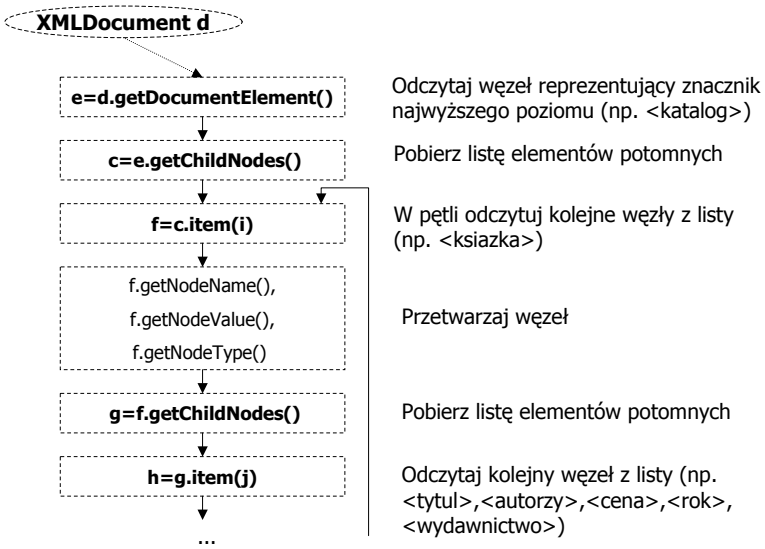
Java: funkcje nawigacyjne DOM API

- **getDocumentElement()** [interfejs Document] – zwraca obiekt węzła reprezentującego znacznik najwyższego poziomu
- **getElementsByTagName(String)** [interfejs Document] – zwraca tablicę obiektów węzłów reprezentujących podany znacznik XML
- **getChildNodes()** [interfejs Node] – zwraca tablicę obiektów węzłów potomnych (bez węzłow atrybutowych)
- **getAttributes()** [interfejs Node] – zwraca tablicę obiektów potomnych węzłów atrybutowych
- **getNodeName()** [interfejs Node] – zwraca nazwę znacznika dla węzła
- **getNodeType()** [interfejs Node] – zwraca numeryczny identyfikator typu węzła
- **getNodeValue()** [interfejs Node] – zwraca treść węzła (tylko dla węzłów tekstowych)
- **getFirstChild()** [interfejs Node] - zwraca obiekt pierwszego węzła potomnego (z pominięciem węzłów atrybutowych)
- **getLastChild()** [interfejs Node] - zwraca obiekt ostatniego węzła potomnego (z pominięciem węzłów atrybutowych)
- **getNextSibling()** [interfejs Node] – zwraca obiekt prawego sąsiada węzła (z pominięciem węzłów atrybutowych)
- **getPreviousSibling()** [interfejs Node] – zwraca obiekt lewego sąsiada węzła (z pominięciem węzłów atrybutowych)
- **getParentNode()** [interfejs Node] – zwraca obiekt węzła nadrzędnego

DOM API: funkcje nawigacyjne



Java: prosta nawigacja w drzewie DOM



Odczytaj węzeł reprezentujący znacznik najwyższego poziomu (np. <katalog>)

Pobierz listę elementów potomnych

W pętli odczytuj kolejne węzły z listy (np. <ksiazka>)

Przetwarzaj węzeł

Pobierz listę elementów potomnych

Odczytaj kolejny węzeł z listy (np. <tytul>,<autorzy>,<cena>,<rok>, <wydawnictwo>)

Java: nawigacja w drzewie DOM

`getChildNodes()`

Wyświetl tytuły wszystkich książek opisanych w dokumencie XML

```
XMLDocument xmlDoc;
...
Node docNode = null, bookNode = null, elementNode = null;
NodeList docNodeList = null, bookNodeList = null;
try {
   docNode = xmlDoc.getDocumentElement();
   docNodeList = docNode.getChildNodes();
   for (int i=0; i<docNodeList.getLength(); i++)  {
     bookNode = docNodeList.item(i);
     bookNodeList = bookNode.getChildNodes();
     for (int j=0; j<bookNodeList.getLength(); j++)  {
       elementNode = bookNodeList.item(j);
       if (elementNode.getNodeName().equals("tytul"))
          System.out.println(elementNode.getFirstChild().getNodeValue());
}}}  catch (Exception e) {System.out.println(e);}
```

```
Access 2002. Projektowanie baz danych. Księga eksperta
Access 2002/XP PL dla każdego
ASP.NET. Vademecum profesjonalisty
C++ XML
Dane w sieci WWW
Delphi 6. Praktyka programowania - tom 1,2
Delphi. Almanach
...
```

Java: nawigacja w drzewie DOM
`getAttributes()`

Wyświetl wartość pierwszego atrybutu każdego znacznika <ksiazka> w dokumencie XML

```
XMLDocument xmlDoc;
...
Node docNode = null, bookNode = null, urlNode = null;
NodeList docNodeList = null, bookNodeList = null;
try {
    docNode = xmlDoc.getDocumentElement();
    docNodeList = docNode.getChildNodes();
    for (int i=0; i<docNodeList.getLength(); i++) {
      bookNode = docNodeList.item(i);
      urlNode = bookNode.getAttributes().item(0);
      System.out.println(urlNode.getNodeValue());
     }
} catch (Exception e) {System.out.println(e);}
```

```
83-7197-669-0
83-7197-786-7
83-7197-691-7
83-7279-215-1
83-7279-149-X
83-7279-214-3
83-7197-469-8
83-7197-377-2
...
```

Java: nawigacja w drzewie DOM
`getElementsByTagName()`

Wyświetl tytuły wszystkich książek opisanych w dokumencie XML

```
XMLDocument xmlDoc;
...
Node titleNode = null;
NodeList titleNodeList = null;
try {
    titleNodeList = xmlDoc.getElementsByTagName("tytul");
    for (int i=0; i<titleNodeList.getLength(); i++) {
      titleNode = titleNodeList.item(i);
      System.out.println(titleNode.getFirstChild().getNodeValue());
     }
} catch (Exception e) {System.out.println(e);}
```

```
Access 2002. Projektowanie baz danych. Księga eksperta
Access 2002/XP PL dla każdego
ASP.NET. Vademecum profesjonalisty
C++ XML
Dane w sieci WWW
Delphi 6. Praktyka programowania - tom 1,2
Delphi. Almanach
...
```

Java: konwersja drzewa DOM do pliku XML

```
...
try {
   xmlDoc.print(new FileOutputStream("C:\\katalog.xml"));
} catch (Exception e) {System.out.println(e);}
...
```

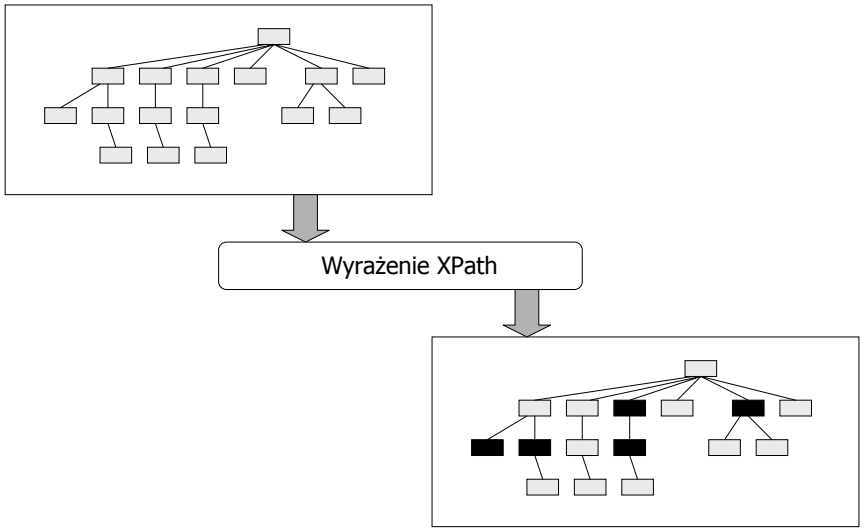
```
<?xml version = '1.0' encoding = 'WINDOWS-1250'?>
<katalog>
   <ksiazka isbn="83-7197-669-0">
      <tytul>C++ XML</tytul>
      <autorzy>
         <autor>Fabio Arciniegas</autor>
      </autorzy>
      <rokwydania>2002</rokwydania>
      <wydawnictwo>Mikom</wydawnictwo>
      <cena>36</cena>
   </ksiazka>
   <ksiazka isbn="83-7197-669-0">
...
```

Język XPath

Język XPath

- XPath to specyfikacja języka służącego do adresowania, odczytywania i przeszukiwania drzew DOM dokumentów XML
- XPath odgrywa podobną rolę w stosunku do drzew DOM, jak język SQL w stosunku do relacyjnych baz danych
- XPath stosuje notację przypominającą ścieżki dostępu w systemach plików
- Wynikiem ewaluacji wyrażenia XPath jest zbiór węzłów spełniających warunki selekcji
- XPath pozwala stosować dwa rodzaje zapisu wyrażeń: skrócony i pełny; rodzaje te mogą być mieszane

Przetwarzanie wyrażeń XPath



Skrócone wyrażenia XPath (1/3)

- Wybór węzłów w drzewie DOM:
  - wybierz autorów wszystkich książek
    /katalog/ksiazka/autorzy
  - wybierz wszystkie wydawnictwa dowolnie zagłębione w drzewie
    //wydawnictwo
  - wybierz wszystkie węzły potomne (dzieci) każdego węzła książka
    //ksiazka/*
- Wybór n-tego węzła danego rodzaju:
  - wybierz pierwszego autora każdej książki
    //autorzy/autor[1]
  - wybierz drugiego autora pierwszej książki
    //ksiazka[1]//autor[2]
  - wybierz ostatnią książke
    //ksiazka[last()]

Skrócone wyrażenia XPath (2/3)

- Wybór węzłów, które posiadają podany węzeł potomny:
  - wybierz książki, które posiadają autorów
    //ksiazka[autorzy]
  - wybierz książki, które napisał Serge Abiteboul
    //ksiazka[autorzy/autor="Serge Abiteboul"]
- Alternatywa ścieżek:
  - wybierz tytuły książek i wydawnictwa
    //tytul | //wydawnictwo
  - wybierz tytuły książek napisanych przez Serge'a Abiteboula lub Kurta Walla

//tytul[..//autor="Serge Abiteboul"] | //tytul[..//autor="Kurt Wall"]

Skrócone wyrażenia XPath (3/3)

- Wybór węzłów zawierających atrybuty:
  - wybierz książkę o numerze ISBN "83-7279-149-X"
    //ksiazka[@isbn="83-7279-149-X"]
  - wybierz wszystkie numery ISBN
    //@isbn
  - wybierz książki, które posiadają numer ISBN
    //ksiazka[@isbn]
- Odczyt treści węzła:
  - odczytaj treści wszystkich tytułów książek
    //ksiazka/tytul/text()

Pełne wyrażenia XPath

- Wyrażenie ścieżkowe XPath składa się z tzw. kroków rozdzielonych ukośnikami
- W pełnym zapisie, każdy z kroków może składać się z:
  - **specyfikatora współrzędnych** (axis), służącego do określenia miejsca w drzewie, począwszy od którego wyszukiwane będą węzły
  - **testu węzła** (node test), służącego do określenia, które węzły są wyszukiwane w obszarze drzewa określonym przez specyfikator współrzędnych
  - **predykatów**, dodatkowo zawężających test węzła, powodujących wybór tylko tych węzłów, które spełniają podany warunek
- Każdy krok pełnego wyrażenia XPath zapisywany jest przy użyciu następującej notacji:
  spec_współrzędnych::test_węzła[predykaty]

Specyfikatory współrzędnych

ancestor	obejmuje wszystkie węzły nadrzędne (ojciec, dziadek, itd.) bieżącego węzła
ancestor-or-self	obejmuje bieżący węzeł plus wszystkie węzły nadrzędne
attribute	obejmuje wszystkie atrybuty bieżącego węzła
child	obejmuje wszystkie węzły bezpośrednio podrzędne bieżącego węzła (dzieci)
descendant	obejmuje wszystkie węzły podrzędne (syn, wnuk, itd.) bieżącego węzła
descendant-or-self	obejmuje bieżący węzeł plus wszystkie węzły podrzędne
following	obejmuje wszystkie węzły, które w dokumencie następują za węzłem bieżącym
following-sibling	obejmuje wszystkie węzły sąsiednie, które w dokumencie następują za węzłem bieżącym
parent	obejmuje węzeł bezpośrednio nadrzędny bieżącego węzła (ojciec)
preceding	obejmuje wszystkie węzły, które w dokumencie następują przed węzłem bieżącym
preceding-sibling	obejmuje wszystkie węzły sąsiednie, które w dokumencie następują przed węzłem bieżącym
self	obejmuje bieżący węzeł

Testy węzłów, operatory i funkcje

node()	dowolny węzeł
text()	węzeł tekstowy
*	dowolny element
@*	dowolny atrybut
nazwa	węzeł o podanej nazwie

=, !=, <, >, <=, >=	porównania
or, and	operatory logiczne
+, -, *, div, mod	operatory arytmetyczne
count()	liczba węzłów wybieranych przez wyrażenie
last()	liczba porządkowa ostatniego węzła wybieranego przez wyrażenie
name()	nazwa węzła wybieranego przez wyrażenie
position()	liczba porządkowa węzła wybieranego przez wyrażenie
not()	negacja logiczna
true()	zwraca prawdę logiczną
false()	zwraca fałsz logiczny

Funkcje - ciąg dalszy

Funkcje operujące na tekstach

concat()	konkatenacja tekstów
contains()	test zawierania tekstów
starts-with()	sprawdzenie, czy tekst rozpoczyna się od podanego ciągu
string()	konwersja do tekstu
string-length()	długość tekstu
substring()	ekstrakcja podciągu znaków
substring-after()	zwraca ciąg znaków znajdujący się za wycinanym podciągiem
substring-before()	zwraca ciąg znaków znajdujący się przed wycinanym podciągiem
translate()	dokonuje zamiany wszystkich wystąpień podanego podciągu

Funkcje operujące na liczbach

ceiling()	górne domknięcie całkowite
floor()	dolne domknięcie całkowite
number()	konwersja do liczby
round()	zaokrąglenie
sum()	suma zbioru wartości liczbowych

Przykłady wyrażeń XPath (1/2)

- Wybierz książki, których cena nie przekracza 20 zł
  //ksiazka[cena<=20]
- Wybierz tytuły książek o cenach w przedziale 30-40 zł
  //ksiazka/tytul[../cena>30 and ../cena<40]
- Wybierz książki napisane przez więcej niż dwóch autorów
  //ksiazka[count(autorzy/autor)>2]
- Wybierz co drugą książkę
  //ksiazka[position() mod 2 = 1]
- Wybierz książki zawierające w tytule słowo XML
  //ksiazka[contains(tytul,"XML")]
- Wybierz autorów o imieniu "Stephen"
  //autor[starts-with(.,"Stephen")]
- Wybierz tytuły złożone z ponad 20 znaków
  //tytul[string-length(.)>20]

Przykłady wyrażeń XPath (2/2)

- Wybierz książki, których numer ISBN spełnia wzorzec ***7197******
  //ksiazka[substring(@isbn,4,4) = "7197"]
- Wybierz wszystkie węzły autorzy oraz wszystkie ich węzły potomne
  //autorzy/descendant-or-self::*
- Wybierz wszystkie książki, które w dokumencie znajdują się za książką o numerze ISBN "83-7197-786-7"
  //ksiazka[@isbn="83-7197-786-7"]/following::ksiazka
- Wybierz wszystkie atrybuty pierwszej książki
  //ksiazka[1]/attribute::*
- Wybierz cenę książki pt. "XML dla każdego"
  //ksiazka[tytul="XML dla każdego"]/child::cena

Transformacja wyrażeń skróconych do pełnych

skrócone	**pełne**	**przykład**
brak	child::	//ksiazka/cena => //ksiazka/child::cena
@	attribute::	//katalog/ksiazka[@isbn= "83-7197-786-7"] => //katalog/child::ksiazka[attribute::isbn= "83-7197-786-7"]
.	self::node()	//tytul[string-length(.)>10] => //tytul[string-length(self::node())>10]
..	parent::node()	//autor/.. => //autor/parent::node()
//	/descendant-or-self::node()	//tytul => /descendant-or-self::node()

Funkcje XPath w DOM API

Java

- **selectNodes(String)** [interfejs Node] – zwraca tablicę obiektów węzłów spełniających podaną ścieżkę XPath
- **selectSingleNode(String)** [interfejs Node] – zwraca pierwszy znaleziony obiekt węzła spełniającego podaną ścieżkę XPath
- **valueOf(String)** [interfejs Node] – zwraca treść pierwszego znalezionego obiektu węzła spełniającego podaną ścieżkę XPath

Java: zapytania XPath

Wyświetl tytuły wszystkich książek wydanych w roku 2002

```
XMLDocument xmlDoc;
...
Node titleNode = null;
NodeList queryNodeList = null;

try {
  queryNodeList = xmlDoc.selectNodes("//ksiazka[rokwydania='2002']/tytul");
  for (int i=0; i<queryNodeList.getLength(); i++) {
      titleNode = queryNodeList.item(i);
      System.out.println(titleNode.getFirstChild().getNodeValue());
  }
}  catch (Exception e) {System.out.println(e);}
```

```
C++ XML
Flash i XML. Techniki zaawansowane
HTML and XML dla początkujących
Programowanie Microsoft SQL Server 2000 z XML
Vademecum XML
XML Kompendium programisty
```

Java: zapytania XPath

Wyświetl tytuły wszystkich książek, których jeden z autorów ma imię "Fabio"

```
XMLDocument xmlDoc;
...
Node titleNode = null;
NodeList queryNodeList = null;

try {
  queryNodeList =
        xmlDoc.selectNodes("//tytul[../autorzy[contains(autor,'Fabio')]]");
  for (int i=0; i<queryNodeList.getLength(); i++) {
      titleNode = queryNodeList.item(i);
      System.out.println(titleNode.getFirstChild().getNodeValue());
  }
}  catch (Exception e) {System.out.println(e);}
```

```
C++ XML
XML Kompendium programisty
```

Java: zapytania XPath

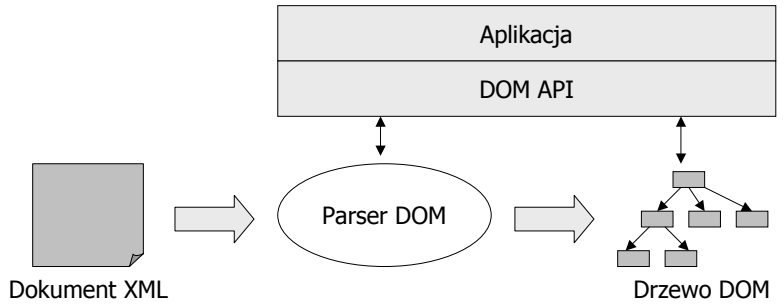
Wyświetl nazwisko pierwszego autora książki pt. "Java i XML"

```
try {
   System.out.println(xmlDoc.valueOf("//ksiazka[tytul='Java i XML']//autor"));
}  catch (Exception e) {System.out.println(e);}
```

```
Brett McLaughlin
```

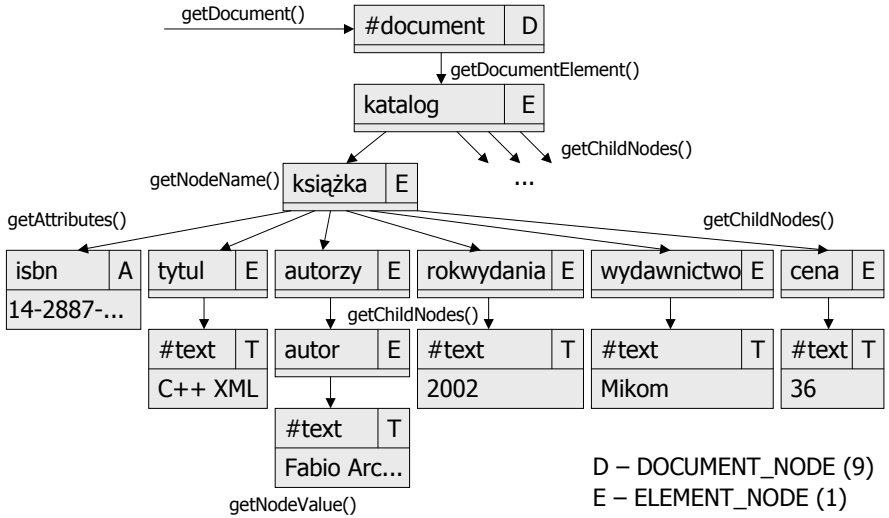
Wykorzystywanie parsera DOM w programach Java

Parser DOM



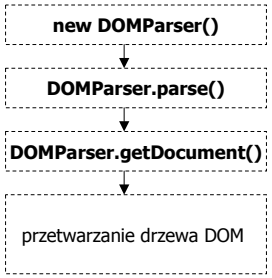
Parser DOM dokonuje przekształcenia dokumentu XML w drzewo DOM. Dostęp do funkcji parsera odbywa się w ramach interfejsu DOM API. Parser DOM jest dostępny w formie bibliotek programistycznych, m.in. Java (oracle.xml.parser.v2)

DOM API: struktura fizyczna drzewa DOM



D – DOCUMENT_NODE (9)
E – ELEMENT_NODE (1)
A – ATTRIBUTE_NODE (2)
T – TEXT_NODE (3)

Java: proces parsowania dokumentu XML



Java: konwersja pliku XML do drzewa DOM

```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
...
DOMParser dp = new DOMParser();
Node titleNode = null;
NodeList titleNodeList = null;
XMLDocument xmlDoc = null;

try {
   dp.parse("file://c:/katalog.xml");
} catch (Exception e) {System.out.println(e);}
try {
     xmlDoc = dp.getDocument();
     titleNodeList = xmlDoc.getElementsByTagName("tytul");
     for (int i=0; i<titleNodeList.getLength(); i++) {
       titleNode = titleNodeList.item(i);
       System.out.println(titleNode.getFirstChild().getNodeValue());
      }
} catch (Exception e) {System.out.println(e);}
```

C++ XML
Flash i XML. Techniki zaawansowane
HTML and XML dla początkujących
Java i XML
Nauka języka XML
Po prostu XML
Poznaj XML w 24 godziny
Programowanie Microsoft SQL Server 2000 z XML
Vademecum XML
XML Kompendium programisty
XML dla każdego

Java: parsowanie zmiennej String

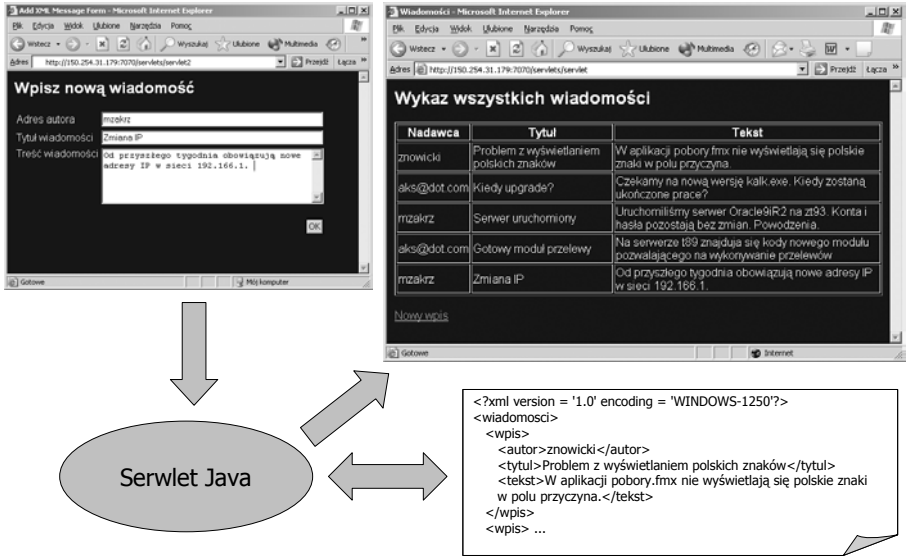
```
import org.w3c.dom.*;
import oracle.xml.parser.v2.*;
import java.io.*;
...
DOMParser dp = new DOMParser();
Node titleNode = null;
NodeList titleNodeList = null;
XMLDocument xmlDoc = null;

try {
  dp.parse(new StringReader("<katalog><ksiazka><tytul>Zaawansowany XML</tytul>"+
           "</ksiazka><ksiazka><tytul>SQL/XML</tytul></ksiazka></katalog>"));
} catch (Exception e) {System.out.println(e);}
try {
     xmlDoc = dp.getDocument();
     titleNodeList = xmlDoc.getElementsByTagName("tytul");
     for (int i=0; i<titleNodeList.getLength(); i++) {
       titleNode = titleNodeList.item(i);
       System.out.println(titleNode.getFirstChild().getNodeValue());
      }
} catch (Exception e) {System.out.println(e);}
```

Zaawansowany XML
SQL/XML

Przykładowa aplikacja

XML jako internetowa baza danych
(DOM API, XSLT)

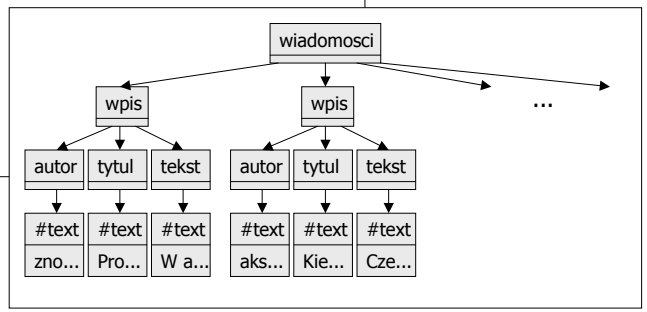
Przykładowa aplikacja
XML jako internetowa baza danych




```
<?xml version = '1.0' encoding = 'WINDOWS-1250'?>
<wiadomosci>
  <wpis>
    <autor>znowicki</autor>
    <tytul>Problem z wyświetlaniem polskich znaków</tytul>
    <tekst>W aplikacji pobory.fmx nie wyświetlają się polskie znaki
    w polu przyczyna.</tekst>
  </wpis>
  <wpis> ...
```

Przykładowa aplikacja
Struktura dokumentu XML i drzewo DOM

```
<?xml version = '1.0' encoding = 'WINDOWS-1250'?>
<wiadomosci>
  <wpis>
    <autor>znowicki</autor>
    <tytul>Problem z wyświetlaniem polskich znaków</tytul>
    <tekst>W aplikacji pobory.fmx nie wyświetlają się polskie znaki
     w polu przyczyna.</tekst>
  </wpis>
  <wpis>
  ...
  </wpis>
  ...
</wiadomosci>
```



Przykładowa aplikacja
Dodawanie nowych wpisów przez serwlet

```
String autor, tytul, tekst;
XMLDocument xmlDoc;
DOMParser dp = new DOMParser();

autor = request.getParameter("autor");
tytul = request.getParameter("tytul");
tekst = request.getParameter("tekst");
try {
   dp.parse("file://C:/messages.xml");
   xmlDoc = dp.getDocument();
   Node wiadomosci = xmlDoc.getDocumentElement();
   Node nowy_wpis = xmlDoc.createElement("wpis");
   Node nowy_autor = xmlDoc.createElement("autor");
   Node nowy_tytul = xmlDoc.createElement("tytul");
   Node nowy_tekst = xmlDoc.createElement("tekst");
   Node nowy_autor_txt = xmlDoc.createTextNode(autor);
   Node nowy_tytul_txt = xmlDoc.createTextNode(tytul);
   Node nowy_tekst_txt = xmlDoc.createTextNode(tekst);
   nowy_autor.appendChild(nowy_autor_txt);
   nowy_tytul.appendChild(nowy_tytul_txt);
   nowy_tekst.appendChild(nowy_tekst_txt);
   nowy_wpis.appendChild(nowy_autor);
   nowy_wpis.appendChild(nowy_tytul);
   nowy_wpis.appendChild(nowy_tekst);
   wiadomosci.appendChild(nowy_wpis);
   xmlDoc.print(new FileOutputStream("C:\\messages.xml"));
} catch (Exception e) {System.out.println(e);}
```

Przykładowa aplikacja
Zastosowanie XSLT do prezentacji zawartości dokumentu XML

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();

URL xmlUrl, xslUrl;
XMLDocument xmlDoc, xslDoc, result;
XSLStylesheet xsl;
DocumentFragment htmlDoc;
XSLProcessor processor = new XSLProcessor();
DOMParser dp = new DOMParser();

try {
   xmlUrl = new URL("file://C:/messages.xml");
   xslUrl = new URL("file://C:/messages.xsl");

   dp.parse(xmlUrl);
   xmlDoc = dp.getDocument();

   dp.parse(xslUrl);
   xslDoc = dp.getDocument();
   xsl = new XSLStylesheet(xslDoc, xslUrl);
   htmlDoc = processor.processXSL(xsl, xmlDoc);
   result = new XMLDocument();
   result.appendChild(htmlDoc);
   result.print(out);
} catch (Exception e) {out.println(e);}
```

Przykładowa aplikacja
Arkusz stylistyczny XSL

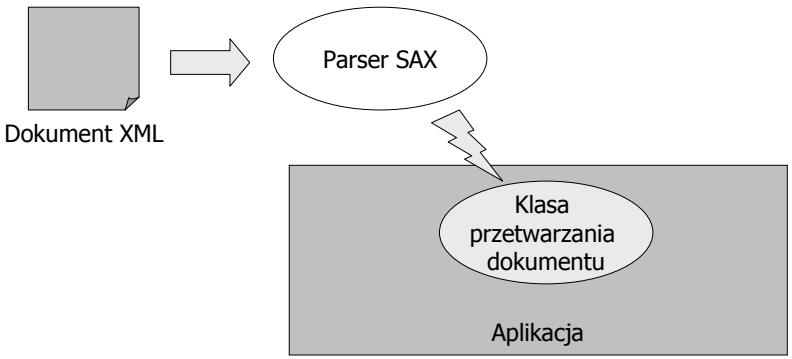
```
<?xml version="1.0" encoding="WINDOWS-1250"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="html"/>
<xsl:template match="/">
<html>
<head>
<meta http-equiv="Content-Type"
  content="text/html;charset=windows-1250"/>
<title>Wiadomości</title>
</head>
<body>
<h2>Wykaz wszystkich wiadomości</h2>
<table border="1">
<tr>
<th>Nadawca</th>
<th>Tytuł</th>
<th>Tekst</th>
</tr>
<xsl:for-each select="/wiadomosci/wpis">
<tr>
<td><xsl:value-of select="autor"/></td>
<td><xsl:value-of select="tytul"/></td>
<td><xsl:value-of select="tekst"/></td>
</tr>
</xsl:for-each>
</table>
<br/>
<a href="../xmlform.html">Nowy wpis</a>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Wykorzystywanie parsera SAX w programach Java

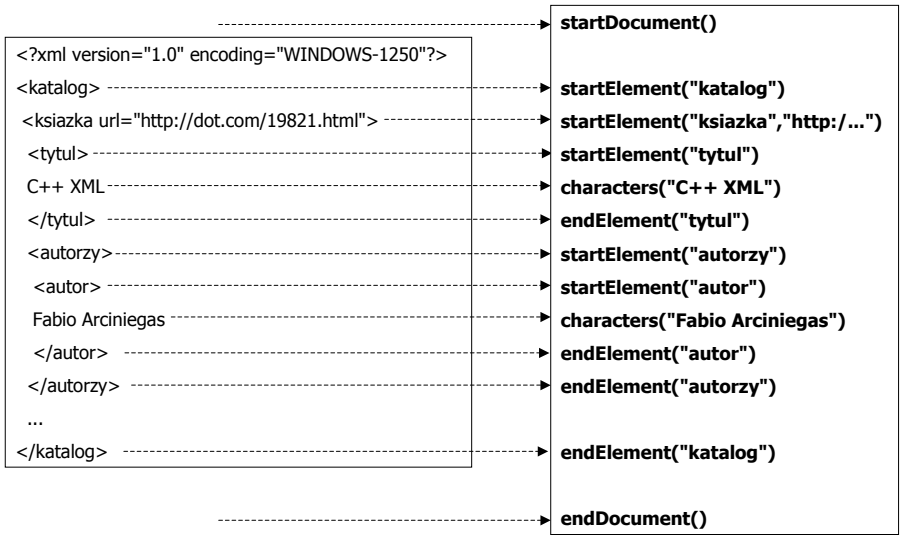
Parser SAX i SAX API

- SAX API to interfejs programistyczny wykorzystujący model zdarzeniowy do budowy parserów dokumentów XML
- Parser SAX odczytuje wskazany dokument XML i dla każdego zidentyfikowanego elementu składniowego wywołuje standardowe metody obiektu klasy przygotowanej przez programistę
- SAX API umożliwia konwersję dokumentu XML do dowolnej struktury danych, a nie wyłącznie do drzewa DOM
- Zadaniem programisty jest przygotowanie klasy dziedziczącej z DocumentHandler i oprogramowanie jej metod:
  - startDocument() / endDocument()
  - startElement() / endElement()
  - characters()

Parser SAX



Sekwencja wywoływania metod klasy DocumentHandler



Implementacja klasy DocumentHandler

```
import org.xml.sax.*;

public class MyHandler extends HandlerBase {
  public void startElement(String name, AttributeList atts) {
  // metoda wywoływana gdy parser SAX natrafi na znacznik otwierający
  // name – nazwa znacznika, atts – lista nazw i wartości atrybutów
  }
  public void endElement(String name) {
  // metoda wywoływana gdy parser SAX natrafi na znacznik zamykający
  // name – nazwa znacznika
  }
  public void characters(char ch[], int start, int length) {
   String value = new String(ch, start, length);
   // metoda wywoływana gdy parser SAX zakończy odczyt treści znacznika
   // value – odczytana treść
}}
```

Wykorzystanie klasy DocumentHandler

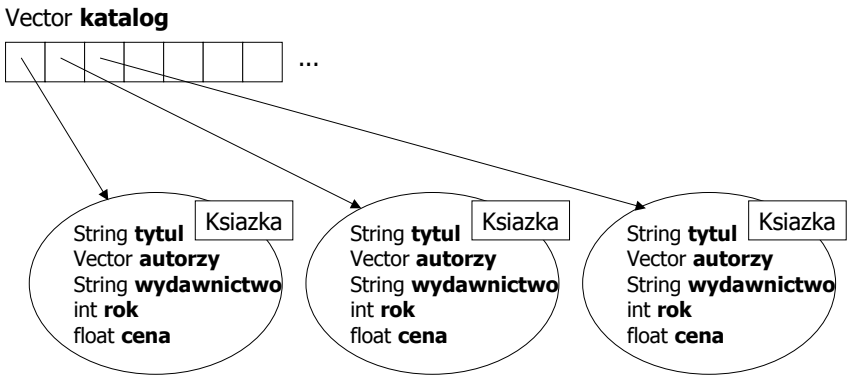
1. Utwórz obiekt własnej klasy przetwarzania dokumentu XML
2. Utwórz obiekt parsera SAX (klasa SAXParser)
3. Za pomocą metody "setDocumentHandler" przyłącz obiekt własnej klasy przetwarzania dokumentu XML do obiektu parsera XML
4. Wywołaj metodę "parse" obiektu parsera, przekazując jej adres URL przetwarzanego dokumentu XML

```
import org.xml.sax.*;
import oracle.xml.parser.v2.*;
...
MyHandler handler = new MyHandler();
Parser parser = new SAXParser();
parser.setDocumentHandler(handler);
parser.parse("file://C:/katalog.xml");
```

Przykładowa aplikacja
Dokument XML

```
<?xml version="1.0" encoding="WINDOWS-1250"?>
<katalog>
 <ksiazka isbn="83-7279-215-1">
  <tytul>C++ XML</tytul>
  <autorzy>
   <autor>Fabio Arciniegas</autor>
  </autorzy>
  <rok>2002</rok>
  <wydawnictwo>Mikom</wydawnictwo>
  <cena>36</cena>
 </ksiazka>
 <ksiazka isbn="83-4579-335-1">
  ...
  </ksiazka>
</katalog>
```

Przykładowa aplikacja
Model obiektowy



Przykładowa aplikacja
Implementacja modelu obiektowego

```
import java.util.*;
public class Ksiazka extends Object {

  String tytul;
  Vector autorzy = new Vector();
  String wydawnictwo;
  int rok;
  float cena;
```

```
 public void setTytul(String t) { tytul = t; }
 public void setWydawnictwo(String w) { wydawnictwo = w; }
 public void setRok(int r) { rok = r; }
 public void setCena(float c) { cena = c; }
 public void addAutor(String a) { autorzy.add(a); }
 public String getTytul() { return tytul; }
 public String getWydawnictwo() { return wydawnictwo; }
 public int getRok() { return rok; }
 public float getCena() { return cena; }
 public int getNumAutor() { return autorzy.size(); }
 public String getAutor(int i) { return (String)autorzy.get(i); }
}
```

Przykładowa aplikacja
Klasa przetwarzania dokumentu (1/2)

```
import org.xml.sax.*;
import java.util.*;

public class KatalogHandler extends HandlerBase {
 Vector katalog = new Vector();  Ksiazka tmp;  int curr_tag;
 static final int TYTUL_TAG = 1;
 static final int ROK_TAG = 2;
 static final int WYDAWNICTWO_TAG = 3;
 static final int CENA_TAG = 4;
 static final int AUTOR_TAG = 5;

 public Vector getKatalog() { return katalog; }

 public void startElement(String name, AttributeList atts) {
  if (name.equals("ksiazka")) { tmp = new Ksiazka(); }
  else if (name.equals("tytul")) { curr_tag = TYTUL_TAG;  }
  else if (name.equals("rok")) { curr_tag = ROK_TAG; }
  else if (name.equals("cena")) { curr_tag = CENA_TAG; }
```

Przykładowa aplikacja
Klasa przetwarzania dokumentu (2/2)

```
else if (name.equals("autor")) { curr_tag = AUTOR_TAG; }
   else if (name.equals("wydawnictwo")) { curr_tag = WYDAWNICTWO_TAG; }
 }

 public void endElement(String name) {
  if (name.equals("ksiazka")) { katalog.add(tmp);}
 }

 public void characters(char ch[], int start, int length) {
  String value = new String(ch, start, length);
  switch (curr_tag) {
    case TYTUL_TAG: tmp.setTytul(value); break;
    case AUTOR_TAG: tmp.addAutor(value); break;
    case WYDAWNICTWO_TAG: tmp.setWydawnictwo(value); break;
    case ROK_TAG: tmp.setRok(Integer.parseInt(value)); break;
    case CENA_TAG: tmp.setCena(Float.parseFloat(value)); break; }
}}
```

Przykładowa aplikacja
Wykorzystanie parsera SAX

```
import org.xml.sax.*;
import oracle.xml.parser.v2.*;
import java.util.*;

public class PrintKatalog {
 public static void main(String[] args) {

  try {
   KatalogHandler handler = new KatalogHandler();
   Parser parser = new SAXParser();
   parser.setDocumentHandler(handler);
   parser.parse("file://C:/katalog.xml");
   Vector katalog = handler.getKatalog();

   for (int i=0; i<katalog.size(); i++) {
    Ksiazka k = (Ksiazka)katalog.elementAt(i);
    System.out.println(k.getTytul()+", "+k.getWydawnictwo()+", "+k.getRok());
   }
  } catch (Exception e) {System.out.println(e);}}}
```

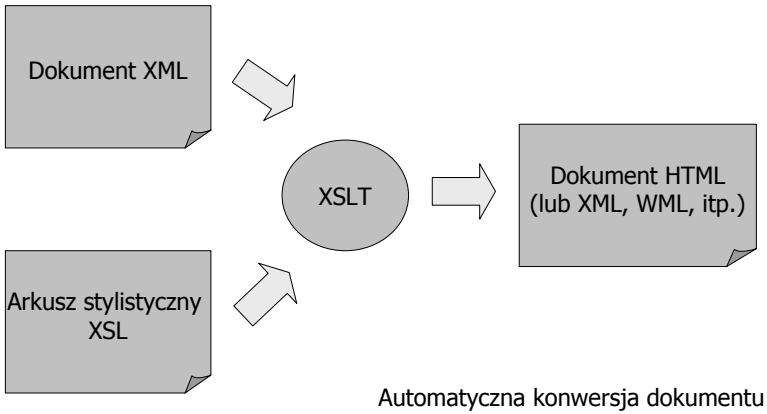
```
C++ XML, Mikom, 2002
Flash i XML. Techniki zaawansowane, Helion, 2002
HTML and XML dla początkujących, Promise, 2002
Java i XML, Helion, 2001
Nauka języka XML, Read Me, 2001
Po prostu XML, Helion, 2001
...
```

Transformacje dokumentów XML przy pomocy XSLT

Dlaczego transformować XML?

- Dokument XML zawiera wyłącznie dane i definicje ich struktur; nieobecne są instrukcje formatujące układ graficzny prezentacji tych danych
- Urządzenia wyświetlające posługują się bogatymi językami opisu prezentacji graficznej dokumentów: HTML, WML, itd.
- W celu zrozumiałej dla użytkownika prezentacji dokumentu XML na ekranie przeglądarki konieczne jest jego przekształcenie do innego formatu
- Przekształcenia XML mogą być realizowane automatycznie przy użyciu technologii XSLT

Transformacja XSLT

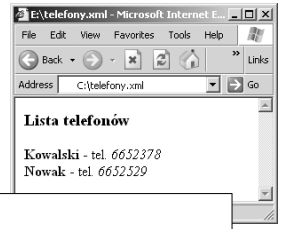


Automatyczna konwersja dokumentu XML do formatu HTML, WML, XML, itp.

Przykład transformacji XSLT

```
<?xml version="1.0" encoding="WINDOWS-1250"?>
<telefony>
  <osoba>
    <nazwisko>Kowalski</nazwisko>
    <numer>6652378</numer>
  </osoba>
  <osoba>
    <nazwisko>Nowak</nazwisko>
    <numer>6652529</numer>
  </osoba>
</telefony>
```

XSLT




```
<html>
 <body>
  <h3>Lista telefonów</h3>
  <b>Kowalski</b> -
  tel. <i>6652378</i><br>
  <b>Nowak</b> -
  tel. <i>6652529</i><br>
 </body>
</html>
```

Metody transformacji XSLT

- Programista przygotowuje arkusz stylistyczny XSL, opisujący sposób transformacji oryginalnego dokumentu XML
- Transformacja może być opisana w sposób **rekurencyjny**, **proceduralny** lub mieszany
- Transformacja dokumentu XML może odbyć się po stronie serwera WWW lub po stronie przeglądarki WWW; w obu przypadkach niezbędny jest moduł XSLT
- Moduł XSLT jest wbudowany w przeglądarkę Microsoft Internet Explorer oraz dostępny w formie zewnętrznej biblioteki Java

Transformacja rekurencyjna (1/2)

Ogólne reguły transformacji

Każde wystąpienie znacznika <telefony>*</telefony> zamień na:

<html><body><h3>Lista telefonów</h3>*</body></html>

Każde wystąpienie znacznika <osoba>*</osoba> zamień na:

***
**

Każde wystąpienie znacznika <nazwisko>*</nazwisko> zamień na:

*** -**

Każde wystąpienie znacznika <numer>*</numer> zamień na:

tel. *

Transformacja rekurencyjna (2/2)

Arkusz stylistyczny XSL

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="html" encoding="windows-1250"/>
 <xsl:template match="telefony">
  <html><body><h3>Lista telefonów</h3><xsl:apply-templates/></body></html>
 </xsl:template>
 <xsl:template match="osoba">
    <xsl:apply-templates/><br/>
 </xsl:template>
 <xsl:template match="nazwisko">
    <b><xsl:value-of select="text()"/></b> -
 </xsl:template>
 <xsl:template match="numer">
    tel. <i><xsl:value-of select="text()"/></i>
 </xsl:template>
</xsl:stylesheet>
```

Transformacja proceduralna (1/2)

Pseudokod transformacji

```
wyświetl "<html><body><h3>Lista telefonów</h3>";
dla każdego znacznika <osoba> {
  wyświetl "<b>";
  wyświetl zawartość znacznika <nazwisko>;
  wyświetl "</b> -";
  wyświetl "tel. <i>";
  wyświetl zawartość znacznika <numer>;
  wyświetl "</i><br>";
}
wyświetl "</body></html>";
```

Transformacja proceduralna (2/2)

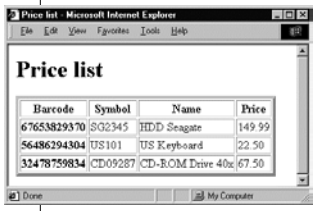
Arkusz stylistyczny XSL

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="html" encoding="windows-1250"/>
 <xsl:template match="telefony">
  <html><body><h3>Lista telefonów</h3>
    <xsl:for-each select="osoba">
     <b><xsl:value-of select="nazwisko"/></b> -
     tel. <i><xsl:value-of select="numer"/></i>
     <br/>
    </xsl:for-each>
  </body></html>
 </xsl:template>
</xsl:stylesheet>
```

Zaawansowany XSL: dostęp do atrybutów

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <HTML>
     <HEAD><TITLE>Price list</TITLE></HEAD>
     <BODY>
       <H1>Price list</H1>
       <TABLE>
       <xsl:attribute name="BORDER">4</xsl:attribute>
         <TR>
           <TH>Barcode</TH>
           <TH>Symbol</TH>
           <TH>Name</TH>
           <TH>Price</TH>
         </TR>
         <xsl:for-each select="pricelist/product">
         <TR>
           <TH><xsl:value-of select="@barcode"/></TH>
           <TD><xsl:value-of select="symbol"/></TD>
           <TD><xsl:value-of select="name"/></TD>
           <TD><xsl:value-of select="price"/></TD>
         </TR>
         </xsl:for-each>
       </TABLE>
     </BODY>
    </HTML>
  </xsl:template>
</xsl:stylesheet>
```

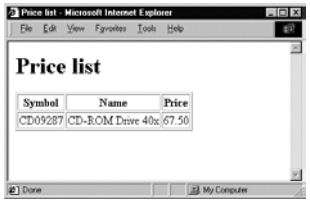
Dodaje do znacznika <TABLE> atrybut BORDER o wartości 4



Odczytuje wartość atrybutu *barcode* znacznika *product*

Zaawansowany XSL: transformacja warunkowa

```
<xsl:for-each select="pricelist/product">
  <xsl:if test=".[symbol='CD09287']">
    <TR>
      <TD><xsl:value-of select="symbol"/></TD>
      <TD><xsl:value-of select="name"/></TD>
      <TD><xsl:value-of select="price"/></TD>
    </TR>
  </xsl:if>
</xsl:for-each>
```




```
<xsl:for-each select="pricelist/product">
  <TR>
  <xsl:attribute name="BGCOLOR">
    <xsl:choose>
      <xsl:when test=".[symbol='US101']">green</xsl:when>
      <xsl:when test=".[symbol='SG2345']">yellow</xsl:when>
      <xsl:otherwise>red</xsl:otherwise>
    </xsl:choose>
  </xsl:attribute>
   <TD><xsl:value-of select="symbol"/></TD>
   <TD><xsl:value-of select="name"/></TD>
   <TD><xsl:value-of select="price"/></TD>
  </TR>
</xsl:for-each>
```

Transformacja XSLT po stronie przeglądarki

telefony.xml

```
<?xml version="1.0" encoding="WINDOWS-1250"?>
<?xml-stylesheet type="text/xsl" href="telefony.xsl"?>
<telefony>
 <osoba>
  <nazwisko>Kowalski</nazwisko>
  <numer>6652378</numer>
 </osoba>
 <osoba>
  <nazwisko>Nowak</nazwisko>
  <numer>6652529</numer>
 </osoba>
</telefony>
```

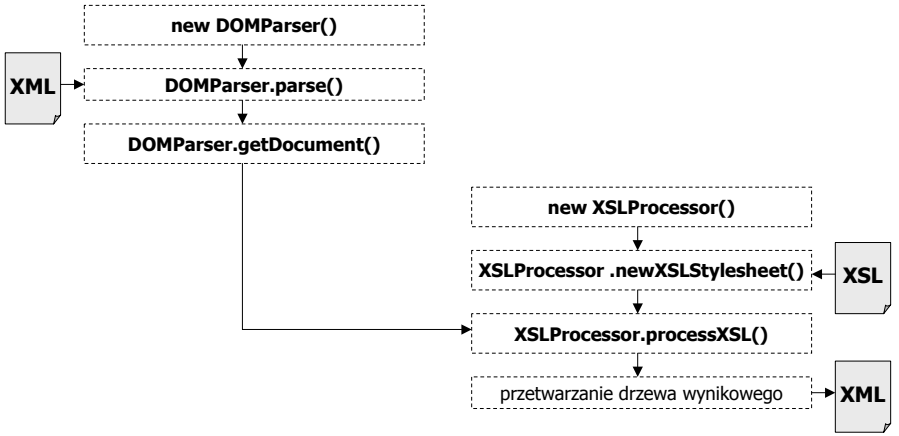
telefony.xsl

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
...="html" encoding="windows-1250"/>
...tch="telefony">
...efonów</h3>
...select="osoba">
...e-of select="nazwisko"/></b> -
...value-of select="numer"/></i>
...>
```

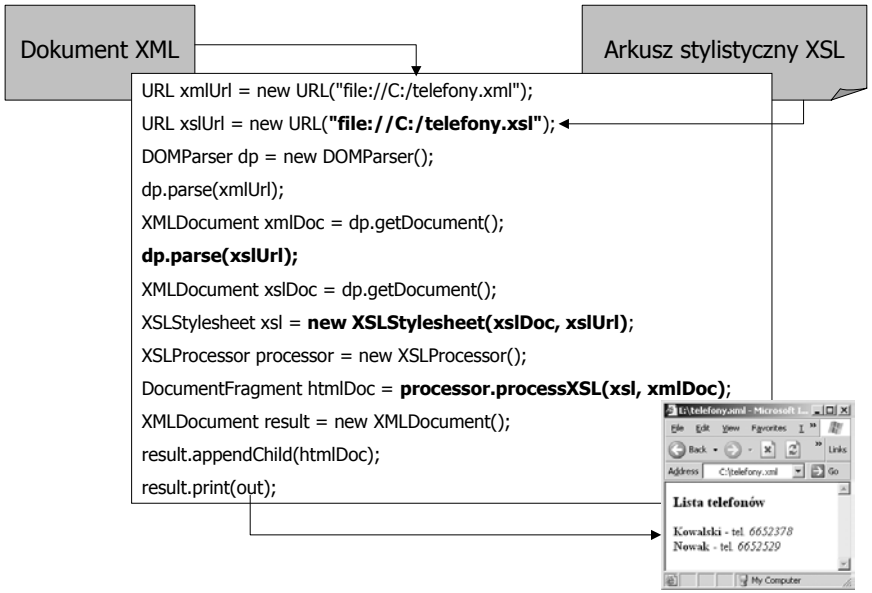
Transformacja XSLT po stronie serwera WWW

1. Przekształć źródłowy dokument XML w drzewo DOM za pomocą parsera DOM
2. Przekształć arkusz stylistyczny XSL w drzewo DOM za pomocą parsera DOM
3. Utwórz obiekt klasy XSLStylesheet, przekazując jej konstruktorowi drzewo DOM oraz adres URL arkusza stylistycznego XSL
4. Utwórz obiekt klasy XSLProcessor
5. Wywołaj dla tego obiektu metodę "processXSL", przekazując jej utworzony obiekt klasy XSLStylesheet oraz drzewo DOM przetwarzanego dokumentu XML – otrzymany wynik to przetransformowany dokument w postaci obiektu klasy DocumentFragment
6. Utwórz nowe drzewo DOM i dołącz do niego otrzymany powyżej przetransformowany dokument
7. Na podstawie utworzonego drzewa DOM wygeneruj i prześlij klientowi wynikowy dokument XML

Parser DOM: przekształcenia XSLT (Java)



Transformacja XSLT po stronie serwera WWW



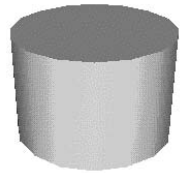
Język VRML

Virtual Reality Modeling Language (VRML 2.0)

- Język i format danych dla opisu multimedialnych, interaktywnych scen graficznych, udostępnianych w sieci Internet; wprowadzony w 1994; uznany przez większość producentów oprogramowania graficznego i przeglądarek WWW
- Pliki VRML zawierają:
  - Nagłowek pliku
  - Komentarze
  - Węzły - atomowe obiekty sceny
  - Pola - atrybuty węzłów
  - Wartości - wartości atrybutów
  - Nazwy węzłów - nazwy węzłów wielokrotnego użycia
  - itd.

Przykład VRML

```
#VRML V2.0 utf8
# A Cylinder
Shape {
   appearance Appearance {
      material Material { }
   }
   geometry Cylinder {
      height 2.0
      radius 1.5
   }
}
```



Shapes i Geometry

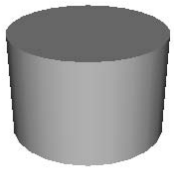
- **Shapes** to klocki, z których budowane są sceny VRML; predefiniowane bryły: ***Box***, ***Cone***, ***Cylinder***, ***Sphere***
- Do budowy brył służą węzły ***Shape***; każdy węzeł opisuje:
  - geometry - kształt bryły
  - appearance - kolor i teksturę bryły

```
Shape {
   geometry   . . .
   appearance . . .}
```

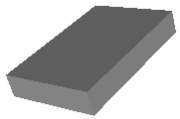
- Pole ***Geometry*** służy do definiowania rozmiarów

```
Shape{geometry Box {size 2.0 0.5 3.0}...}
Shape{geometry Cone {height 3.0 bottomRadius 0.75}...}
Shape{geometry Cylinder {height 2.0 radius 1.5}...}
Shape{geometry Sphere {radius 1.0}...}
```

Geometry - przykłady



```
geometry Cylinder {
      height 2.0
      radius 1.5
   }
```



```
geometry Box {
     size 2.0 0.5 3.0}
   }
```

```
geometry Cone {
     height 3.0 bottomRadius 0.75}
   }
```

```
geometry Sphere {
     radius 1.0}
   }
```


```
geometry Text {
     string "Text Example"  fontStyle FontStyle {
       size 1 }}
```

Grupy obiektów

- Węzły mogą być grupowane przy pomocy ***Group***:

  Group {children [. . .]}

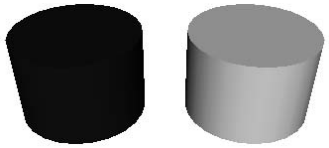
  Przykład:

```
#VRML V2.0 utf8
Group {children [
      Shape { . . . },
      Shape { . . . }, . . .]}
```

Transformacje 3D

- Domyślnie, wszystkie bryły powstają w początku układu współrzędnych; transformacje 3D pozwalają je przesuwać, obracać i skalować
- Węzeł ***Transform*** grupuje przekształecenia elementarne: translation, rotation, scale:

```
Transform { translation 2.0 0.0 0.0
            rotation 0.0 0.0 1.0  0.52
            scale 0.5 0.5 0.5
            children [ . . . ]}
```



```
Transform {
  translation 2 0 0
  children [
    Shape {
      geometry Cylinder {
        height 2.0 radius 1.5}
      appearance Appearance {
        material Material { diffuseColor 1 1 0 }
}}]}

Transform {
  translation -2 0 0
  children [
    Shape {
      geometry Cylinder {
        height 2.0  radius 1.5}
      appearance Appearance {
        material Material { diffuseColor 0 0 1 }
}}]}
```

Sterowanie wyglądem powierzchni - Appearance

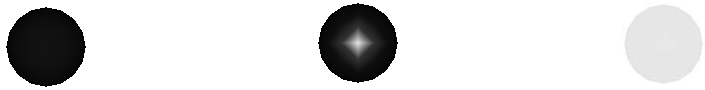
- Węzły ***Appearance*** opisują: **material** - kolor, przezroczystość, itd.:

```
Shape {
   appearance Appearance {
        material ...}
   geometry ...}
```

- Węzły ***Material*** opisują: **diffuse color** - barwa światła odbitego dyfuzyjnie, **emissive color** - światło emitowane, **transparency** - przezroczystość, **specularColor** - barwa światła odbijanego lustrzanie (Phong) etc.

```
Material {diffuseColor  ...
          emissiveColor ...
          transparency  ...}
```

Material - przykłady



```
Shape {
  appearance Appearance {
    material Material {
      diffuseColor 0 0 1
    }
  }
  geometry Sphere {
    radius 1
  }
}
```

```
Shape {
  appearance Appearance {
    material Material {
      diffuseColor 0 0 1
      specularColor 1 1 1
    }
  }
  geometry Sphere {
    radius 1
  }
}
```

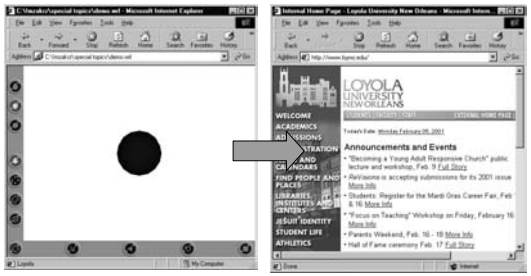
```
Shape {
  appearance Appearance {
    material Material {
      diffuseColor 0 0 1
      transparency 0.9
    }
  }
  geometry Sphere {
    radius 1
  }
}
```

Łączniki

- Każdy obiekt lub grupa obiektów mogą stać się łącznikiem (**anchor**); węzeł ***Anchor*** służy do realizacji funkcjonalności podobnej do łączników HTML:

```
Anchor {
    url "stairway.wrl"
    description "Floating Stairs"
    children [ . . . ]}
```

```
Anchor {
   url "http://www.loyno.edu"
   description "Loyola"
   children [
      Shape {
        appearance Appearance {
          material Material {diffuseColor 0 0 1}}
          geometry Sphere {radius 1}
        }
    ]
}
```



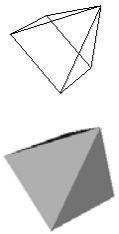
Punkty, odcinki, ściany

- Węzły ***PointSet***, ***IndexedLineSet*** i ***IndexedFaceSet*** służą do tworzenia kształtów określanych przez zbiór punktów:

```
PointSet {coord Coordinate {point [1 2 3.5, 2 8 1.1, ...]}}
```

```
IndexedLineSet {coord Coordinate {point [...]}
        coordIndex [1, 0, 3, -1, 2, 4, 6, -1, ...]}
```

```
IndexedFaceSet {coord Coordinate {point [...]}
        coordIndex [1, 0, 3, -1, ...]}
```



Tekstury

- Ustawienia ***Texture*** przesłaniają ustawienia ***Material*** ; tekstura może być zdefiniowana jako ***ImageTexture***, ***MovieTexture*** lub ***PixelTexture***:

```
Appearance {...
            texture ImageTexture {url "myimage.jpg"}}
```

```
Appearance {...
            texture MovieTexture {url "mymovie.mpg"
              speed 1
              loop FALSE}}
```

```
Appearance {...
            texture PixelTexture {image 2 4 3 0xFF0000 0x00FF00
              0 0 0 0 0xFFFFFF 0xFFFF00}}
```

Tekstury - przykład

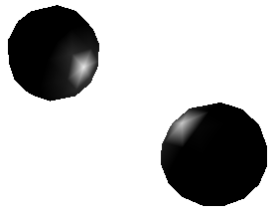
```
Shape {
 geometry Cylinder {
   height 2.0
   radius 1.5}
 appearance
Appearance {
   texture ImageTexture
{
      url "photo.jpg"
      }
   }
}
```

```
Shape {
 geometry Cylinder {
   height 2.0
   radius 1.5}
 appearance Appearance {
   texture PixelTexture {image 3 3 3 0
   255 0 255 0 255 0 255 0}}
}
```

Źródła światła PointLight

- Trzy rodzaje źródeł światła: ***PointLight***, ***DirectionalLight***, ***SpotLight***

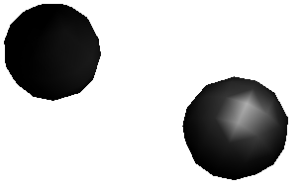
```
PointLight {
    intensity 1.0
    ambientIntensity 0.0
    color 1.0 1.0 1.0
    location 0.0 0.0 0.0
    radius 1.0
    attenuation 1.0 0.0 0.0}
```



DirectionalLight i SpotLight

```
DirectionalLight {
  ambientIntensity 0
  color 1 1 1
  direction 0 0 -1
  intensity 1
}
```

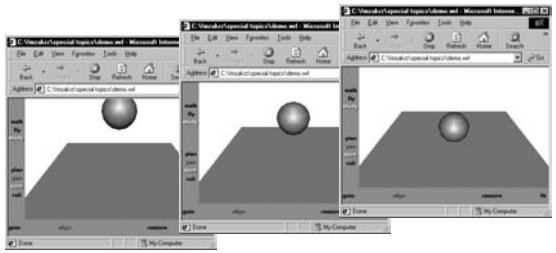
```
SpotLight {
   ambientIntensity 0
   attenuation 1 0 0
   beamWidth 1.570796
   color 1 1 1
   cutOffAngle 0.785398
   direction 0 0 -1
   intensity 1
   location 0 0 0
   radius 100 }
```



VRML i JavaScript

```
#VRML V2.0 utf8
Transform
 {translation 0 -1 0
  children [
  Shape{
   geometry Box {
      size 10.0 0.1 10.0}
   appearance Appearance {
     material Material {diffuseColor 0.8 1 0.8}}
 }]}

DEF Ball Transform {
    children [
        Shape {
            appearance Appearance {
                material Material {
                    diffuseColor 1.0 0.5 0.5
                    specularColor 0.7 0.7 0.7
                }

            }
            geometry Sphere { }
}]}

DEF Clock TimeSensor {
    cycleInterval 2.0
    startTime 1.0
    stopTime 0.0
    loop TRUE
}
```

```
DEF Bouncer Script {
    field    SFFloat bounceHeight 3.0
    eventIn  SFFloat set_fraction
    eventOut SFVec3f value_changed
    url "vrmlscript:
        function set_fraction( frac, tm ) {
            y = 4.0 * bounceHeight * frac * (1.0 - frac);
            value_changed[0] = 0.0;
            value_changed[1] = y;
            value_changed[2] = 0.0;
        }"
}
ROUTE Clock.fraction_changed TO Bouncer.set_fraction
ROUTE Bouncer.value_changed  TO Ball.set_translation
```



Język WML

Składnia dokumentów WML

- Wireless Markup Language (WML) jest oparty na XML
- Dokumenty WML składają się z tzw. kart (cards); w danej chwili tylko jedna karta może być wyświetlana na ekranie
- Każdy dokument WML rozpoczyna się nagłowkiem, złożonym z dwóch części: nagłówka XML i znacznika DTD:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
      "http://www.wapforum.org/DTD/wml_1.1.xml">
```

Card

- Do grupowania treści wewnątrz dokumentu WML służy znacznik <card>:

```
<card id="id" title="title">
<p>
content
</p>
</card>
```

id - nazwa karty, używana do nawigacji

title - tytuł karty, do wyświetlania na szczycie ekranu

content - treść tekstowa lub kombinacja innych znaczników WML (znaczniki paragrafów .. są obowiązkowe)

- Wszystkie karty muszą być zamknięte wewnątrz znacznków **<wml>..</wml>**

Znaczniki WML

- Bold:
- Underline: <u></u>
- Italic:
- Big: <big></big>
- Small: <small></small>
- Emphasis:
- Strong:
- Line break:
- Image:
- Tables: <table columns="value"></table>,<tr></tr>,<td></td>
- Outer links: Link text
- Links to cards: Link text

Przykład WML

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
    "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
 <card id="home" title="My WAP Home">
  <p>
   <img src="photo1.wbmp" alt="My photo"/>
   <b>This</b> is an example WML page.(<a href="#addr">contact</a>)
  </p>
 </card>
 <card id="addr" title="Contact">
  <p align="center">
   <big>My address:</big>
   Maciej Zakrzewicz<br/>
   Piotrowo 3a<br/>
   Poznan<br/>
   POLAND
  </p>
 </card>
</wml>
```

Język JavaScript

Wprowadzenie

- **JavaScript** przenaszalnym, zorientowanym obiektowo językiem skryptowym
- **JavaScript** nie jest samodzielnym, pełnowartościowym językiem programowania, został tak zaprojektowany, aby można było go łatwo zagnieżdżać w innych produktach i aplikacjach
- **JavaScript** został opracowany przez firmę Netscape
  - ECMAScript – ustandaryzowana wersja języka (JavaScript jest z nią zgodny, jednocześnie oferując dodatkową funkcjonalność)
  - Jscript – rozszerzona implementacja ECMAScript firmy Microsoft
- **Core JavaScript** obejmuje podstawowy zbiór obiektów, takich jak *Array*, *Date* i *Math*, oraz podstawowe elementy składni języka: operatory, struktury sterujące, instrukcje
- **Core JavaScript** może być rozszerzony o dodatkowe elementy, wymagane w określonych zastosowaniach

Podstawowe rozszerzenia JavaScript

- **Client-side JavaScript** rozszerza podstawę języka o obiekty reprezentujące środowisko przeglądarki i model dokumentu – tzw. Document Object Model (DOM), pozwalając m.in. na:
  - Dynamiczną generację dokumentów HTML
  - Obsługę formularzy HTML
  - Obsługę zdarzeń i komunikację z użytkownikiem
  - Nawigację między dokumentami
- **Server-side JavaScript** rozszerza podstawę języka o obiekty typowe dla środowiska serwera, umożliwiając m.in.:
  - Komunikację z bazami danych
  - Operacje na plikach

JavaScript – Przykład programu

```
<HTML>
<HEAD>
<TITLE>JavaScript Example</TITLE>
<SCRIPT LANGUAGE="JavaScript">
  document.writeln("<H1>Welcome!</H1>");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Wykonuje się zanim wyświetlane jest <BODY>

Średnik na końcu linii można pominąć, gdy koniec instrukcji wynika z kontekstu

Obiekt ***document*** reprezentuje dokument HTML aktualnie wyświetlany w przeglądarce

Metoda ***writeln*** wypisuje jedną linię tekstu w treści wyświetlanego dokumentu

Wyświetlanie komunikatów

```
<HTML>
<HEAD>
<TITLE>JavaScript Example</TITLE>
<SCRIPT LANGUAGE="JavaScript">
  window.alert("Welcome to \n JavaScript!");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Obiekt ***window*** reprezentuje okno przeglądarki

Metoda ***alert*** wyświetla okienko dialogowe z komunikatem

Wartości i typy danych

- JavaScript wykorzystuje następujące typy danych:
  - **liczby**, np. `42` lub `3.14159`
  - **wartości logiczne**, `true` oraz `false`
  - **łańcuchy znaków**, np. `"hello!", 'Wow!', "", "one \n two"`
  - **null**, wartość null (JavaScript rozróżnia wielkość liter: np. `NULL` nie jest traktowane jako `null`), w kontekście liczbowym zachowuje się jak `0`, w kontekście logicznym jak `false`
  - **undefined**, oznacza wartość niezdefiniowaną, w warunkach logicznych zachowuje się jak `false`
- JavaScript jest językiem „dynamicznie typowanym" – deklarując zmienną nie podaje się jej typu; konwersja typów przeprowadzana jest automatycznie w trakcie działania skryptu
  - `x = "The answer is " + 42 // zwraca "The answer is 42"`
  - `"37" - 7 // zwraca 30`
  - `"37" + 7 // zwraca "377"`

Zmienne

- Zmienne deklaruje się na dwa sposoby:
  - Przez **przypisanie** wartości; np. `x = 42`
  - Za pomocą słowa kluczowego **var**; np. `var x = 42`
- Zmienna lub element tablicy, któremu nie przypisano wartości ma wartość **undefined**; wynikiem jej ewaluacji jest:
  - Gdy zmienna deklarowana z **var** – **undefined** lub **NaN**
  - Gdy zmienna deklarowana bez **var** – generowany jest błąd
- Zmienne deklarowane poza funkcjami są zmiennymi **globalnymi**; zmienne deklarowane w obrębie funkcji są zmiennymi **lokalnymi** – dostępnymi tylko w danej funkcji
- Użycie słowa kluczowego **var** przy deklaracji zmiennej globalnej jest opcjonalne, ale jest wymagane przy deklaracji zmiennych lokalnych
- Stałe deklaruje się za pomocą **const**; np. `const g = 9.81`

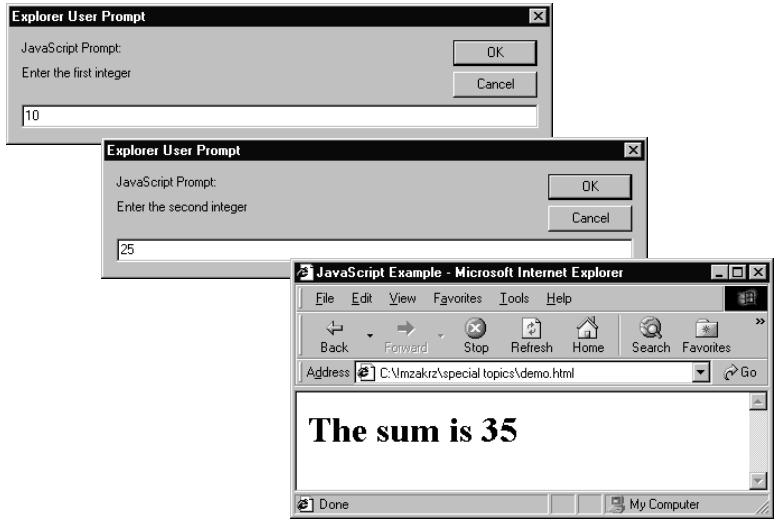
Korzystanie ze zmiennych – Przykład (1)

Metoda ***prompt*** wyświetla okienko do wprowadzania wartości

```
<HTML>
<HEAD>
<TITLE>JavaScript Example</TITLE>
<SCRIPT LANGUAGE="JavaScript">

 firstNumber = window.prompt("Enter the first integer","0");
 secondNumber = window.prompt("Enter the second integer","0");

 number1 = parseInt(firstNumber);
 number2 = parseInt(secondNumber);

 sum = number1 + number2;

 document.writeln("<H1>The sum is " + sum + "</H1>");
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```

Jawnie konwertuje łańcuch znaków na liczbę całkowitą

Korzystanie ze zmiennych – Przykład (2)



Wyrażenia i operatory

- Przypisania: `+=`, `-=`, `*=`, `/=`, `%=`, `<<=`, `>>=`
- Arytmetyczne: `+`, `-`, `*`, `/`, `%`, `++`, `--`
- Bitowe: <<, >> (zachowuje znak), >>>, &, ^, |
- Porównania: `==`, `!=`, `===` (równe wartości i ten sam typ), `!==`, `>`, `<`, `>=`, `<=`
- Logiczne: `&&`, `||`, `!`
- Inne: `condition ? val1:val2`, `op1, op2` (operator przecinka), `delete` (niszczy obiekt), `new` (tworzy obiekt), `this` (bieżący obiekt), `typeof` (zwraca typ operandu jako łańcuch znaków), `void`, `in`, `instanceof`

Instrukcje

```
if (condition) {
statements1
}
[else {
statements2
} ]
```

```
do {
statement
} while (condition)
```

```
while (condition) {
statements
}
```

```
switch (expression){
 case label :
   statement;
   break;
 case label :
   statement;
   break;
   ...
 default : statement;
}
```

```
with (object){
statements
}
```

```
throw exception
```

```
try {
}
catch (exception)
{}
```

```
for (variable in object) {
statements }
```

```
label
```

```
break
```

```
continue
```

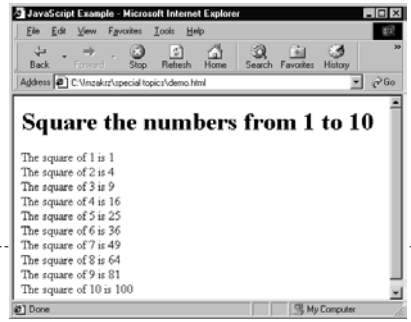
```
/* comments */
```

```
// comments
```

```
for ([initialExpression]; [condition]; [incrementExpression]) {
statements
```

Definiowanie funkcji

```
<HTML>
<HEAD>
<TITLE>JavaScript Example</TITLE>
<SCRIPT LANGUAGE="JavaScript">
 document.writeln("<H1>Square the numbers from 1 to 10</H1>");

 for (x = 1; x <= 10; x++)
  document.writeln("The square of "+x+" is "+square(x)+ "</BR>");

 function square(y)
  {
    return y*y;
  }
</SCRIPT>
</HEAD>
<BODY>
</BODY>
</HTML>
```



Obiekty (1)

Dostęp do atrybutów obiektu:

```
myCar.make = ”Ford”
myCar.model = ”Mustang”
myCar.year = 1969;
```

Tworzenie obiektu:

```
myNewCar = {make:”Honda”,model: ”Civic”, year: 1999}
```

Tworzenie obiektu za pomocą konstruktora:

```
function car(make, model, year) {
   this.make = make;
   this.model = model;
   this.year = year;
}

myNextCar = new car(”Volvo”, ”S70”, 1998);
```

Obiekty (2)

Definiowanie metody obiektu:

```
function displayCar() {
  var result = "A Beautiful " + this.year + " " +
               this.make + " " + this.model
  pretty_print(result)
}

function car(make, model, year) {
   this.make = make;
   this.model = model;
   this.year = year;
   this.displayCar = displayCar;
}
```

Nowe metody i pola można dodawać również do istniejących obiektów:

```
myCar = new car(”Volvo”, ”S70”, 1998);
myCar.speed = 240;
```

Obiekty (3)

Tworzenie hierarchii dziedziczenia za pomocą prototypów:

```
function car(make, model, year) {
  this.make = make; this.model = model;
  this.year = year || new Date().getFullYear();
}

function truck(make, model, capacity , year) {
  this.base = car;
  this.base(make, model, year);
  this.capacity = capacity;
}

truck.prototype = new car;
```

Konstruktor *car* uczyniony metodą *truck* (nazwa *base* - dowolna)

Prototypowy obiekt *car* staje się prototypem dla konstruktora *truck*

Predefiniowane obiekty

- **Array**, e.g.: `billingMethod = new Array(5)`
- **Boolean**, e.g.: `myBoleanObject = new Boolean(true)`
- **Date**, e.g.: `Xmas95 = new Date("December 25, 1995 13:30:00")`
  - metody obiektu Date: `set*`, `get*` (np. `getFullYear()`), ...
- **Function**, e.g.:`var setBGColor = new Function( "document.bgColor='antiquewhite'")`
- **Math**, e.g.: `Math.sin(1.56)`
  - metody obiektu Math: `abs, sin, cos, tan, acos, asin, atan, exp, log, ceil, floor, min, max, pow, random, round, sqrt`
- **String**, e.g.: `s1 = new String("foo")`
- **RegExp**
- **Number**

Tablice

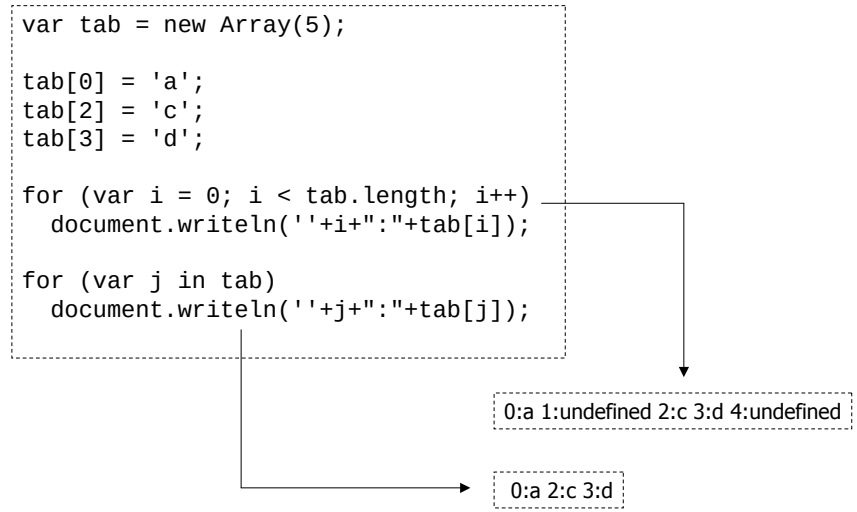
- Do obsługi tablic służy obiekt **Array**
- Przykłady tworzenia i inicjalizowania tablic:

```
billingMethod = new Array(5);
billingMethod[1] = "cash";
myArray = new Array("Wind","Rain","Fire");
coffees = ["French Roast", "Columbian", "Kona"]
```

- Metody obiektu Array: `concat, join, pop, push, reverse, shift, slice, splice, sort, unshift`; Pole: `length`
- Indeks liczbowy od `0`; indeksem tablicy mogą być również łańcuchy znaków (tablice asocjacyjne)

Nawigacja po elementach tablicy

```
var tab = new Array(5);

tab[0] = 'a';
tab[2] = 'c';
tab[3] = 'd';

for (var i = 0; i < tab.length; i++)
  document.writeln(''+i+":"+tab[i]);

for (var j in tab)
  document.writeln(''+j+":"+tab[j]);
```



Łańcuchy znaków

- Łańcuchy znaków występują jako obiekty **String** i jako literały (wywołanie metody obiektu String na rzecz literału powoduje automatyczną konwersję literału do tymczasowego obiektu String)
- Metody obiektu String:
  - zorientowane na HTML: `anchor, big, blink, fixed, italics, small, strike, sub, sup, link`
  - zorientowane na przetwarzanie treści: `charAt, charCodeAt, indexOf, lastIndexOf, concat, fromCharCode, split, slice, substring, substr, match, replace, search, toLowerCase, toUpperCase`
- Operatory konkatenacji: +, +=; Pole: `length`

```
document.writeln("First bold text".bold());
document.writeln("<B>Second bold text</B>");
```

JavaScript w dokumentach HTML

- JavaScript może być zagnieżdżony w dokumencie HTML:
  - jako instrukcje i funkcje w obrębie znacznika **<SCRIPT>...</SCRIPT>** w sekcji <HEAD>
  - poprzez wyspecyfikowanie pliku z kodem JavaScript, np. **<SCRIPT SRC="filename.js"></SCRIPT>**
  - poprzez podanie wyrażenia JavaScript jako wartości atrybutu znacznika HTML, np. **<HR WIDTH="&{barWidth};%" ALIGN="LEFT">**
  - jako procedury obsługi zdarzeń dla znaczników HTML, np. **<INPUT TYPE ="button" VALUE="Press Me" onClick="myfunc('astring')">**
- Kod JavaScript jest zazwyczaj umieszczany w komentarzu HTML w celu ukrycia go przed starszymi przeglądarkami
- Kod HTML zawarty w znaczniku **<NOSCRIPT>** jest wyświetlany w przeglądarkach nieobsługujących języka JavaScript

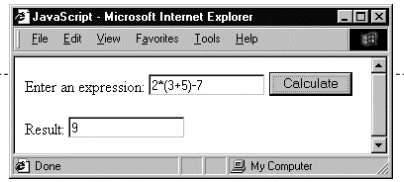
Obsługa zdarzeń

- **Nazwy** zdarzeń: `onAbort, onBlur, onChange, onClick, onDragDrop, onError, onFocus, onKeyDown, onKeyUp, onKeyPress, onLoad, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, onMove, onReset, onResize, onSelect, onSubmit, onUnload`
- Definiowanie **procedur obsługi zdarzeń**:
  - **`<TAG eventHandler="JavaScript Code">`**
  - przykład: `<INPUT TYPE="button" NAME="Button1" VALUE="Open Sesame!" onClick="window.open('mydoc.html', 'newWin')">`

Obsługa zdarzeń - Przykład (1)

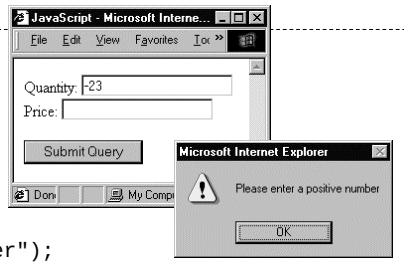
```
<HTML>
<HEAD>
<SCRIPT>
function compute() {
  myform.result.value = eval(myform.expr.value)
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="myform">
Enter an expression: <INPUT TYPE="text" NAME="expr" SIZE=15 >
<INPUT TYPE="button" VALUE="Calculate" onClick="compute()">
<BR><BR>
Result: <INPUT TYPE="text" NAME="result" SIZE=15 >
</FORM>
</BODY>
</HTML>
```

funkcja ***eval*** wartościuje podane wyrażenie



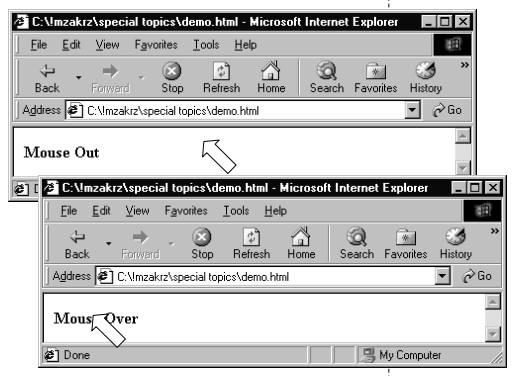
Obsługa zdarzeń – Przykład (2)

```
<HTML>
<HEAD>
<SCRIPT>
function isaPosNum(s) {
  return (parseInt(s) > 0)
}
function value_check(item) {
 if (!isaPosNum(item.value))
   alert("Please enter a positive number");
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
Quantity: <INPUT TYPE="text" NAME="q" onChange="value_check(this)"><BR>
Price: <INPUT TYPE="text" NAME="p" onChange="value_check(this)"><BR><BR>
<INPUT TYPE="submit">
</FORM>
</BODY>
</HTML>
```



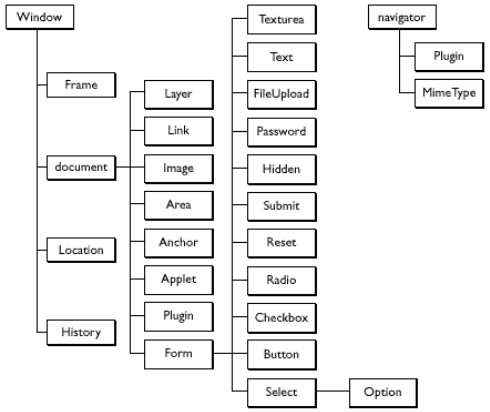
Obsługa zdarzeń – Przykład (3)

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function m1()
{
  i1.innerText="Mouse Over";
}

function m2()
{
  i1.innerText="Mouse Out";
}

</SCRIPT>
</HEAD>
<BODY>
<B id="i1" onMouseOver="m1()" onMouseOut="m2()">Mouse Out</B>
</BODY>
</HTML>
```



Hierarchia obiektów dokumentu HTML i przeglądarki

- Otwarcie dokumentu HTML w przeglądarce powoduje utworzenie wielu obiektów języka JavaScript
- Obiekty te tworzą hierarchię będącą odbiciem struktury strony HTML

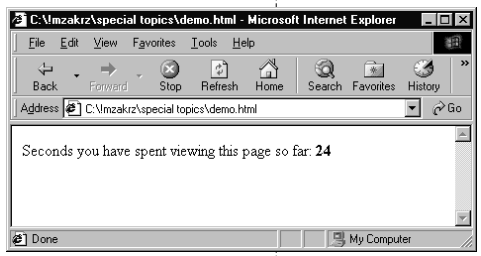


Obiekty window i frame

- Podstawowe metody:
  - `open` (otwiera okno przeglądarki)
  - `close` (zamyka okno przeglądarki)
  - `alert` (wyświetla okienko z komunikatem)
  - `confirm` (wyświetla okienko z przyciskami OK i Anuluj)
  - `prompt` (wyświetla okienko z polem tekstowym do wprowadzenia wartości)
  - `focus`, `blur` (aktywuje / deaktywuje element)
  - `scrollTo` (przewija zawartość okna do określonej współrzędnej)
  - `setInterval` (wartościuje wyrażenie lub wywołuje funkcję zgodnie z podanym interwałem czasowym)
  - `setTimeout` (wartościuje wyrażenie lub wywołuje funkcję po upłynięciu podanego czasu)
- Podstawowe pola:
  - `location` (pozwala przekierować klienta do innego adresu URL)

window.setInterval() - Przykład

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
var seconds = 0;

function startTimer(){
  window.setInterval("updateTime()", 1000);
}

function updateTime(){
  seconds++;
  soFar.innerText = seconds;
}

</SCRIPT>
</HEAD>
<BODY OnLoad="startTimer()">

<P>Seconds you have spent viewing this page so far:
<B ID="soFar">0</B></P>
</BODY>
</HTML>
```



Obiekty document i form

- Metody obiektu **document**:
  - `write, writeln, ...`
- Właściwości obiektu document:
  - `bgColor, fgColor, linkColor, alinkColor, vlinkColor, lastModified, referrer, cookie, ...`
- Każdy formularz reprezentowany jest przez obiekt **form**; dokument HTML może zawierać wiele formularzy – są one dostępne przez tablicę **forms** lub swoje nazwy, np.: `document.forms[0], document.myForm`
- Elementy formularza (pola tekstowe, przyciski radiowe, itp.) są dostępne poprzez tablicę **elements** lub swoje nazwy, np.: `document.forms[0].elements[0], document.forms[0].imie`

Obiekty location, history i navigator

- Pola obiektu **location** zawierają informacje o adresie URL aktualnie załadowanej strony; **location** ma dwie methody:
  - `reload` – ponownie ładuje bieżący dokument
  - `replace` – ładuje podany adres URL w bieżącej pozycji historii
- Obiekt **history** zawiera listę łańcuchów znaków reprezentujących odwiedzone adresy URL; właściwości: `current`, `next`, `previous`, metody: `go`; e.g. `history.go(-1)`
- Obiekt **navigator** zawiera informacje o wersji używanej przeglądarki; jedna z metod:
  - `javaEnabled` – informuje czy włączona jest Java

Typowe zastosowania JavaScript po stronie przeglądarki

- Weryfikacja poprawności danych wprowadzanych do formularzy HTML
- Dynamiczne modyfikacje list wyboru w formularzach
- Otwieranie nowych okien z możliwością określenia ich położenia i funkcjonalności
- Otwieranie dodatkowych okien przy otwarciu lub opuszczeniu dokumentu
- Nawigacja za pomocą przycisków
- Modyfikacja wyglądu elementów w związku z nawigacją myszą
- Budowa rozwijanych menu

Weryfikacja poprawności danych w formularzu - Przykład

```
<HTML>
<HEAD>
<SCRIPT>
function sprawdz() {
  var liczby = "0123456789";
  if (mojaForma.rok.value == "")
  { alert("Podaj wartość!"); return false; }
  for (var i=0; i<mojaForma.rok.value.length; i++)
    if (liczby.indexOf(mojaForma.rok.value.charAt(i)) == -1)
      { alert("Rok musi być liczbą!"); return false; }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM NAME="mojaForma">
Podaj rok: <INPUT TYPE="text"
NAME="rok" SIZE=5 >
<INPUT TYPE="button"
VALUE="Szukaj" onClick="sprawdz()">
</FORM>
</BODY>
</HTML>
```

Zwrócenie ***false*** wstrzymuje wysłanie danych z formularza