Programowanie współbieżne i rozproszone

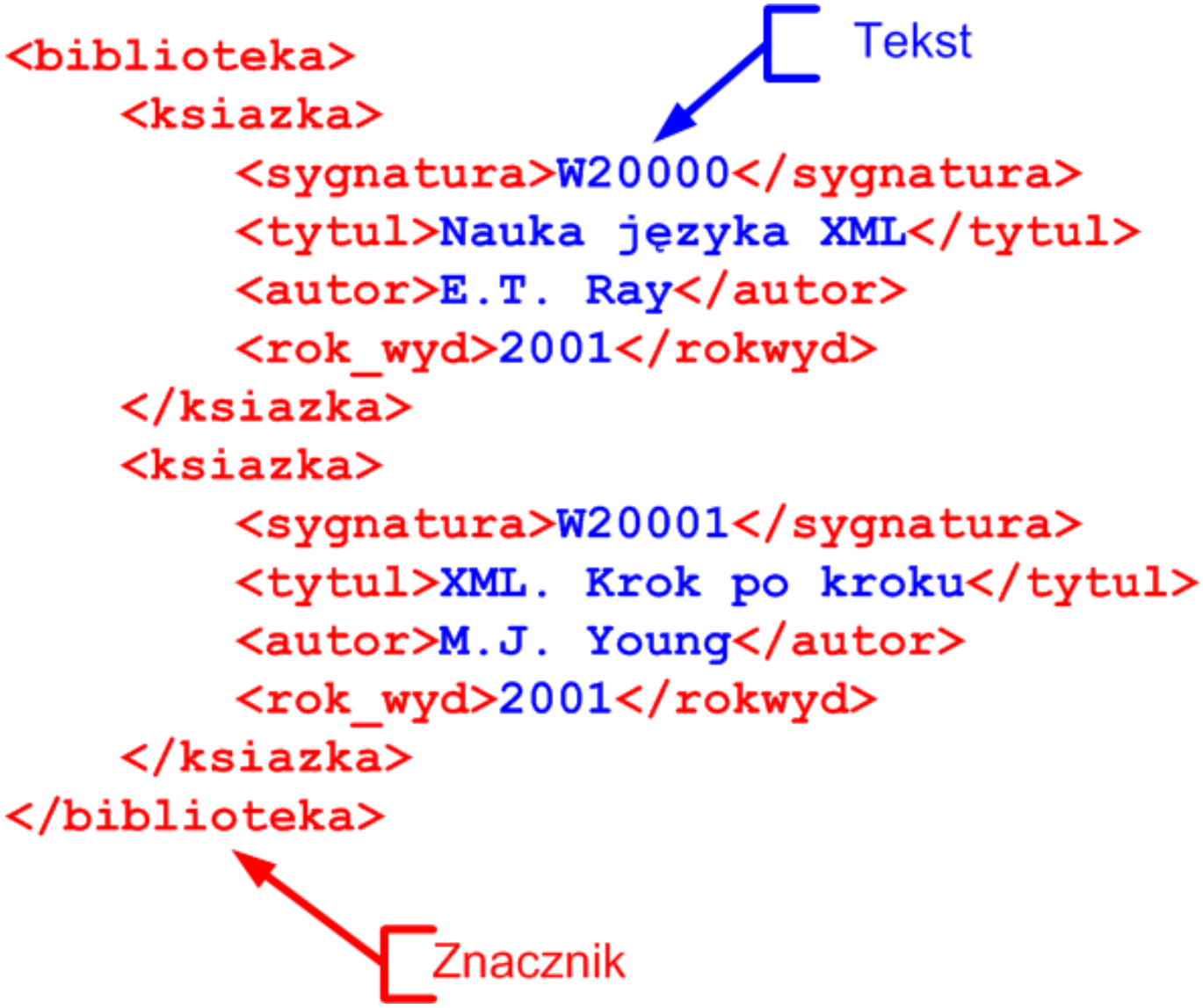
WYKŁAD 6

Krzysztof Pancerz

XML

- XML (ang. eXtensible Markup Language - rozszerzalny język znaczników) definiuje ogólną składnię, stosowaną przy oznaczaniu danych za pomocą znaczników.
- XML jest **metajęzykiem** i dostarcza możliwość definiowania własnych znaczników.
- XML oferuje standardowy format dokumentów.
- XML może służyć do opisu praktycznie wszystkich rodzajów danych.

Dokument XML



Znaczniki

- Znaczniki występujące w dokumencie XML opisują tylko strukturę oraz semantykę dokumentu.
- Znaczniki nie informują o sposobie prezentacji (wyświetlania) dokumentu.

Elementy

- Elementy dzielą dokument na części składowe.
- Elementy mogą być pojemnikami zawierającymi tekst lub inne elementy lub mogą być puste.
- Dla elementów pustych wartość informacyjną ma tylko ich położenie i atrybuty.

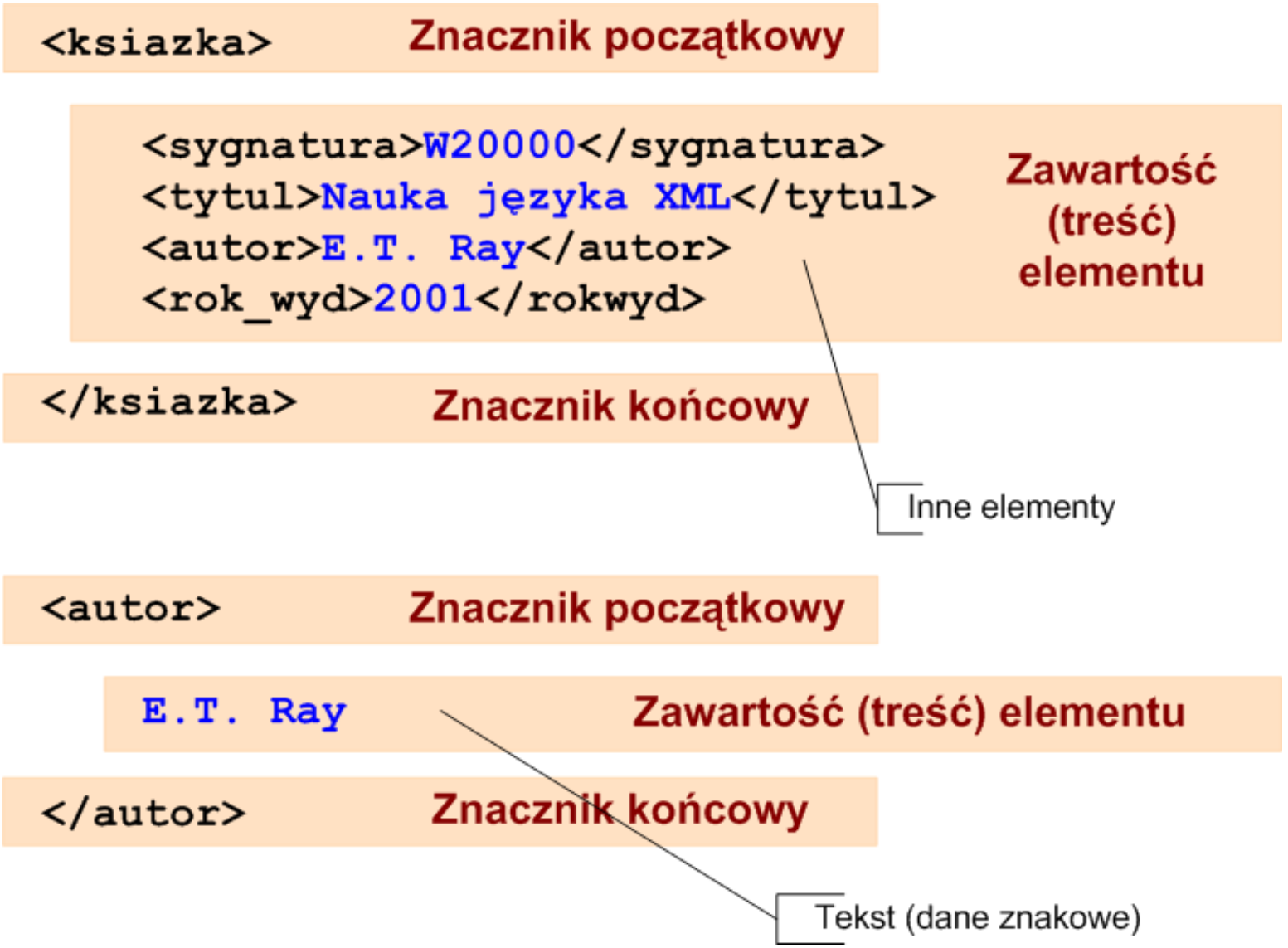
Elementy (cd.)


<ocena>
<punkty>
<kolokwium> 20 </kolokwium>
<obecnosc> 5 </obecnosc>
<aktywnosc/>
</punkty>
</ocena>

Element pojemnika

- Element pojemnika zawiera inne elementy lub tekst (dane znakowe). Zwartość może też być mieszana (inne elementy, dane znakowe).

Element pojemnika (cd.)



Element pusty

- Element pusty pozbawiony jest zawartości (treści).

```
<ksiazka/>
```

Znacznik

```
<ksiazka/>  ≡  <ksiazka></ksiazka>
```

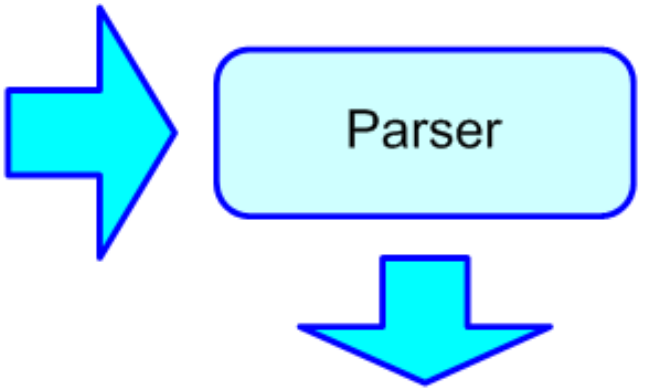
Struktura drzewa

- Dokumenty XML posiadają hierarchiczną strukturę drzewa, w której jedne elementy są całkowicie zagnieżdżone w innych.
- Każdy dokument XML zawiera dokładnie jeden element bazowy (element główny, element dokumentu) nie posiadający rodzica, który zawiera wszystkie pozostałe elementy.
- Element taki wyznacza granicę dokumentu.

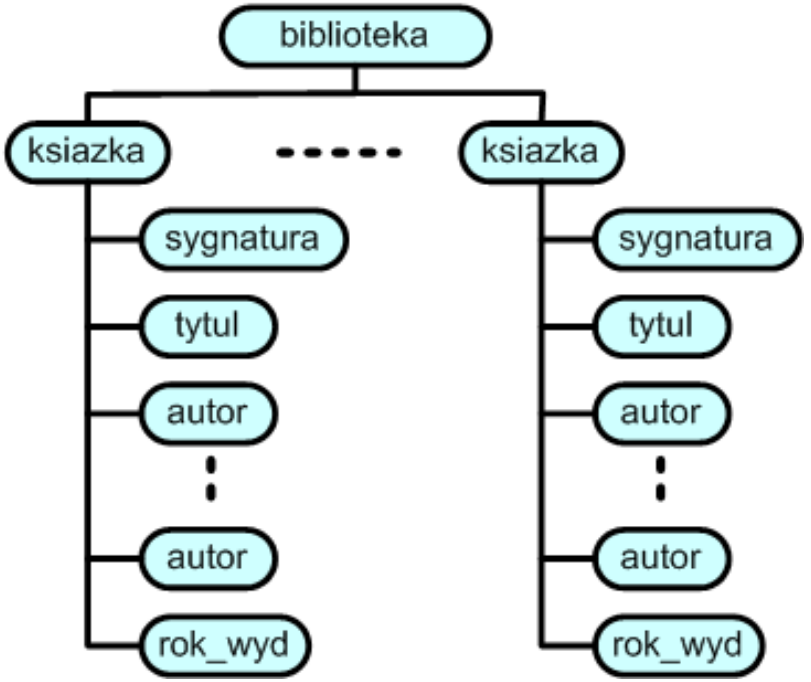
Przetwarzanie dokumentów XML

Dokument XML

```
<biblioteka>
<ksiazka>
<sygnatura>W20000</sygnatura>
<tytul>Nauka języka XML</tytul>
<autor>E.T. Ray</autor>
<rok_wyd>2001</rokwyd>
</ksiazka>
<ksiazka>
<sygnatura>W20001</sygnatura>
<tytul>XML. Krok po kroku</tytul>
<autor>M.J. Young</autor>
<rok_wyd>2001</rokwyd>
</ksiazka>
</biblioteka>
```



Reprezentacja wewnętrzna



Atrybuty

```
nazwa="wartość"
```

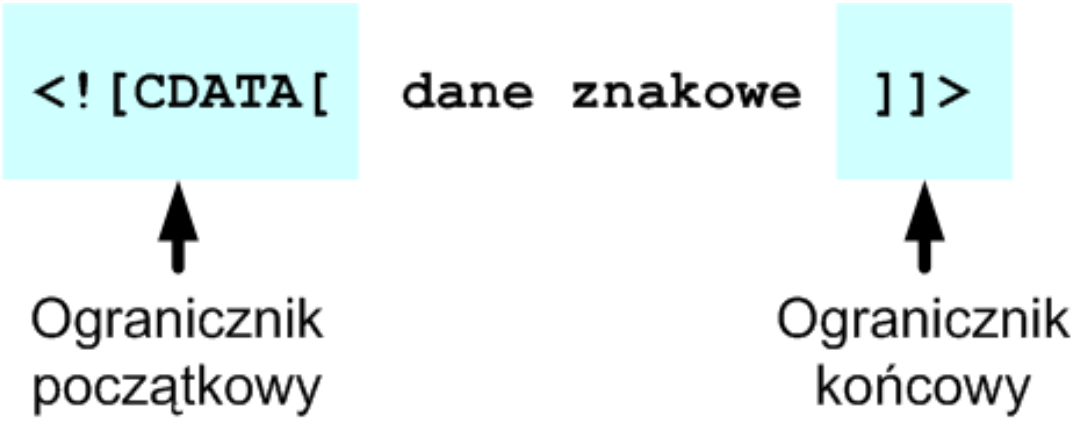
Atrybut

```
<ksiazka sygnatura="W20000">

    <tytul>Nauka języka XML</tytul>
    <autor>E.T. Ray</autor>
    <rok_wyd>2001</rokwyd>

</ksiazka>
```

Atrybuty

```
<autor imie="Eric" nazwisko="Ray"/>
```

Sekcje CDATA




```
<rownanie numer=”20">
    <![CDATA[
    Jeśli a>=b wtedy c=10
    ]]>
</rownanie>
```

Sekcja CDATA

Modele przetwarzania dokumentów XML w języku Java

- Model SAX
- Model DOM
- Model JDOM

Model SAX

- Model SAX (*Simple API for XML*) definiuje sposób przetwarzania dokumentów XML.
- SAX nie jest parserem (tzn. sam nie dokonuje przetwarzania dokumentów XML).
- SAX określa sposób odczytywania dokumentów XML i rozbijania danych na odpowiednie elementy za pomocą mechanizmu obsługi zdarzeń.
- SAX definiuje zdarzenia procesu przetwarzania dokumentu XML, które podlegają monitorowaniu i obsłudze.

Model SAX (cd.)

- Parser działający w oparciu o model SAX przetwarza dokument XML i za każdym razem, gdy napotyka jakiś znacznik, komentarz, dane tekstowe lub jakikolwiek inny element dokumentu sygnalizuje odpowiednie zdarzenie.
- W obsłudze sygnalizowanego przez parser zdarzenia mogą zostać wykonane odpowiednie działania.

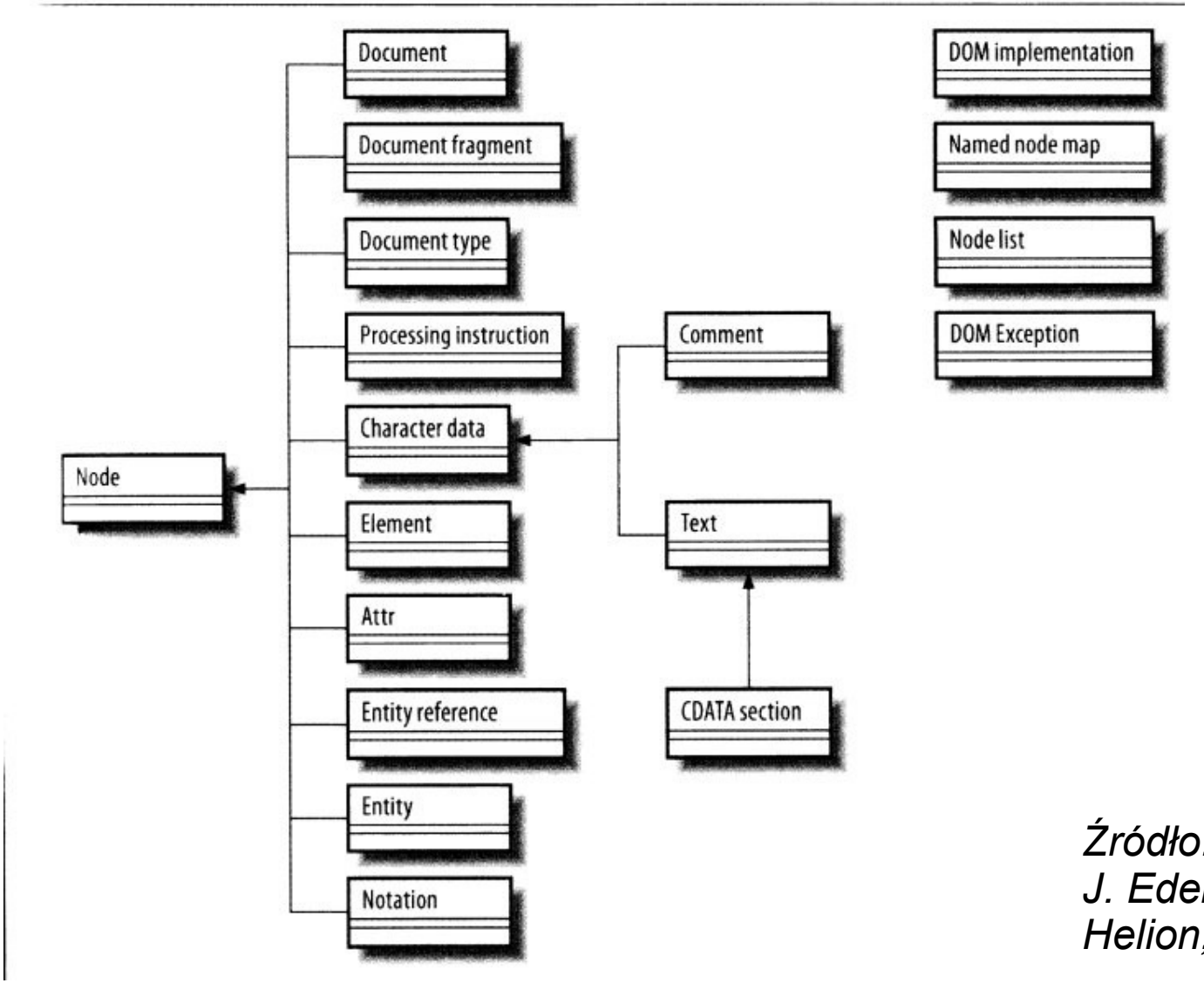
Model DOM

- Model DOM (*Document Object Model*) został zdefiniowany przez W3C (*World Wide Web Consortium*) DOM Working Group.
- DOM umożliwia dostęp do danych oraz manipulowanie danymi w dokumentach XML.
- Dokument XML w modelu DOM reprezentowany jest za pomocą struktury drzewa.
- Cały dokument XML wczytywany jest do pamięci, a wszystkie dane umieszczane są w węzłach drzewa.

Model DOM (cd.)

- DOM jest specyfikacją niezależną od platformy i języka programowania.
- Model DOM jest dostępny praktycznie we wszystkich językach programowania.

Hierarchia interfejsów modelu DOM w języku Java związanych z reprezentacją dokumentu XML



Źródło: B.D. McLaughlin, J. Edelson: Java i XML. Helion, Gliwice, 2007.

Podstawowe klasy i interfejsy modelu DOM

DocumentBuilderFactory

- klasa abstrakcyjna
- “fabryka” API umożliwiająca otrzymanie parsera tworzącego obiekt DOM z dokumentu XML
- wybrane metody:
  - **newInstance()** – tworzy instancję „fabryki”
  - **newDocumentBuilder()** – tworzy nową instancję DocumentBuilder

Podstawowe klasy i interfejsy modelu DOM (cd.)

DocumentBuilder

- klasa abstrakcyjna
- definicja API umożliwiającego otrzymanie obiektu DOM z dokumentu XML
- wybrana metoda:
  - **parse(File f)** – parsuje dokument XML z podanego pliku i zwraca obiekt DOM

Podstawowe klasy i interfejsy modelu DOM (cd.)

Document

- interfejs
- wewnętrzna reprezentacja dokumentu XML
- wybrane metody:
  - **createElement(String tagName)** – tworzy w dokumencie element o podanej nazwie
  - **getDocumentElement()** – zwraca element dokumentu

Podstawowe klasy i interfejsy modelu DOM (cd.)

Element

- interfejs
- element dokumentu XML
- wybrane metody:
  - **setAttribute(String name, String value)** – ustawia wartość podanego atrybutu
  - **getAttribute(String name)** – zwraca wartość podanego atrybutu
  - **getElementsByTagName(String name)** – zwraca listę węzłów potomnych o podanej nazwie

Podstawowe klasy i interfejsy modelu DOM (cd.)

Node

- interfejs
- węzeł dokumentu XML
- wybrane metody:
  - **appendChild(Node newChild)** – dodaje do węzła podany węzeł potomny
  - **getChildNodes()** - pobiera wszystkie węzły potomne danego węzła

Podstawowe klasy i interfejsy modelu DOM (cd.)

NodeList

- interfejs
- lista węzłów dokumentu XML
- wybrane metody:
  - **getLength()** – zwraca rozmiar listy

Porównanie modeli SAX i DOM

Pobieranie danych:

SAX: dane pobierane w obsłudze zdarzeń

DOM: dane pobierane ze struktury drzewa

Dostęp do danych:

SAX: dostęp sekwencyjny do danych

DOM: dostęp swobodny do danych

Porównanie modeli SAX i DOM (cd.)

Zużycie pamięci:

SAX: małe zużycie pamięci, możliwe przetwarzanie w pamięci części dokumentu XML

DOM: znaczne zużycie pamięci, w pamięci przetwarzany jest cały dokument XML

Przetwarzanie:

SAX: przetwarzanie jednokrotne dokumentu

DOM: przetwarzanie wielokrotne dokumentu

JDOM

- JDOM umożliwia dostęp do dokumentów XML z poziomu języka Java poprzez strukturę drzewa.
- JDOM został zaprojektowany specjalnie dla języka Java dzięki czemu jest bardziej intuicyjny od DOM.
- JDOM składa się z konkretnych klas umożliwiających bezpośrednie tworzenie obiektów.
- W JDOM używane są kolekcje języka Java, np. kolekcja *List* zamiast listy *NodeList* modelu DOM.

Pakiety dla języka Java

SAX

- org.xml.sax
- org.xml.sax.helpers, org.xml.sax.ext

DOM

- org.w3c.dom

JDOM

- org.jdom
- org.jdom.input, org.jdom.output, org.jdom.adapters, org.jdom.filter, org.jdom.transform

Protokół SOAP

- Na bazie XML zdefiniowano protokoły wykorzystywane w aplikacjach rozproszonych i sieciowych, np. protokół SOAP.
- SOAP - Simple Object Access Protocol (Prosty protokół dostępu do obiektów) - protokół wykorzystywany do wywoływania metod serwerów, serwisów, komponentów oraz obiektów).
- Dane w postaci XML przesyłane są zasadniczo za pomocą protokołu HTTP.

Protokół SOAP

- Specyfikacja SOAP określa rodzaje nagłówków HTTP oraz definiuje słownik znaczników XML wykorzystywanych do reprezentacji parametrów, zwracanych wartości oraz generowanych przez metody wyjątków.

Protokół SOAP

- Struktura komunikatu SOAP:
  - **Envelope** - opakowanie, element bazowy (korzeń dokumentu).
  - **Header** - nagłówek (opcjonalny). Bloki nagłówka (podelementy elementu Header) zawierają informacje precyzujące sposób traktowania wiadomości (np. kodowanie).
  - **Body** - ciało zawierające parametry wywołania / odpowiedź / informacje o błędach.

Protokół SOAP

- Cechy komunikatu SOAP:
  - musi być zapisany w formacie XML
  - musi zawierać element Envelope
  - może zawierać element Header
  - musi zawierać element Body
  - musi używać odpowiednich przestrzeni nazw
  - nie może zawierać odwołania do DTD
  - nie może zawierać instrukcji przetwarzania XML

Protokół SOAP

Przykład struktury komunikatu SOAP:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

Źródło: http://www.w3schools.com/

Protokół SOAP

Przykład komunikatu żądania SOAP:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
  <m:GetPrice xmlns:m="http://www.w3schools.com/prices">
    <m:Item>Apples</m:Item>
  </m:GetPrice>
</soap:Body>

</soap:Envelope>
```

Źródło: http://www.w3schools.com/

Protokół SOAP

Przykład komunikatu odpowiedzi SOAP:

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
  <m:GetPriceResponse xmlns:m="http://www.w3schools.com/prices">
    <m:Price>1.90</m:Price>
  </m:GetPriceResponse>
</soap:Body>

</soap:Envelope>
```

Źródło: http://www.w3schools.com/

JSON

- JSON (JavaScript Object Notation):
  - samoopisujący się standard wymiany danych w aplikacjach internetowych,
  - sposób przekształcania obiektów JavaScript w łańcuch tekstowy i odwrotnie.
- Format JSON jest formatem tekstowym niezależnym od języka programowania.
- JSON oparty jest jest o:
  - zbiory par "atrybut-wartość",
  - uporządkowane listy wartości.

JSON

- Zalety JSON w stosunku do XML:
  - JSON jest bardziej efektywny (używa mniejszej liczby znaków),
  - transformacje danych w formacie JSON na postać tekstową i odwrotnie jest szybsze niż przetwarzanie plików XML,
  - składnia JSON jest zgodna ze składnią języka JavaScript,
  - w JSON mogą być używane tablice.

JSON a XML

- Używanie XML jest wskazane w przypadkach złożonych obiektów oraz w sytuacjach gdy istnieją zdefiniowane transformacje XSLT.

JSON

- Wspólne cechy formatów JSON i XML:
  - są samoopisujące się,
  - są hierarchiczne,
  - mogą być parowane w programach napisanych w różnych językach,
  - mogą być przesyłane za pomocą XMLHttpRequest.

Składnia JSON

- Dane są parami "atrybut-wartość".
- Dane są odseparowane od siebie za pomocą przecinka.
- Nawiasy klamrowe { } obejmują obiekt.
- Nawiasy prostokątne [ ] obejmują tablice.

Składnia JSON

- Wartość JSON może być:
  - liczbą (całkowitą lub zmiennoprzecinkową),
  - łańcuchem znaków,
  - wartością logiczną (*true* albo *false*),
  - tablicą,
  - obiektem,
  - wartością pustą (*null*).

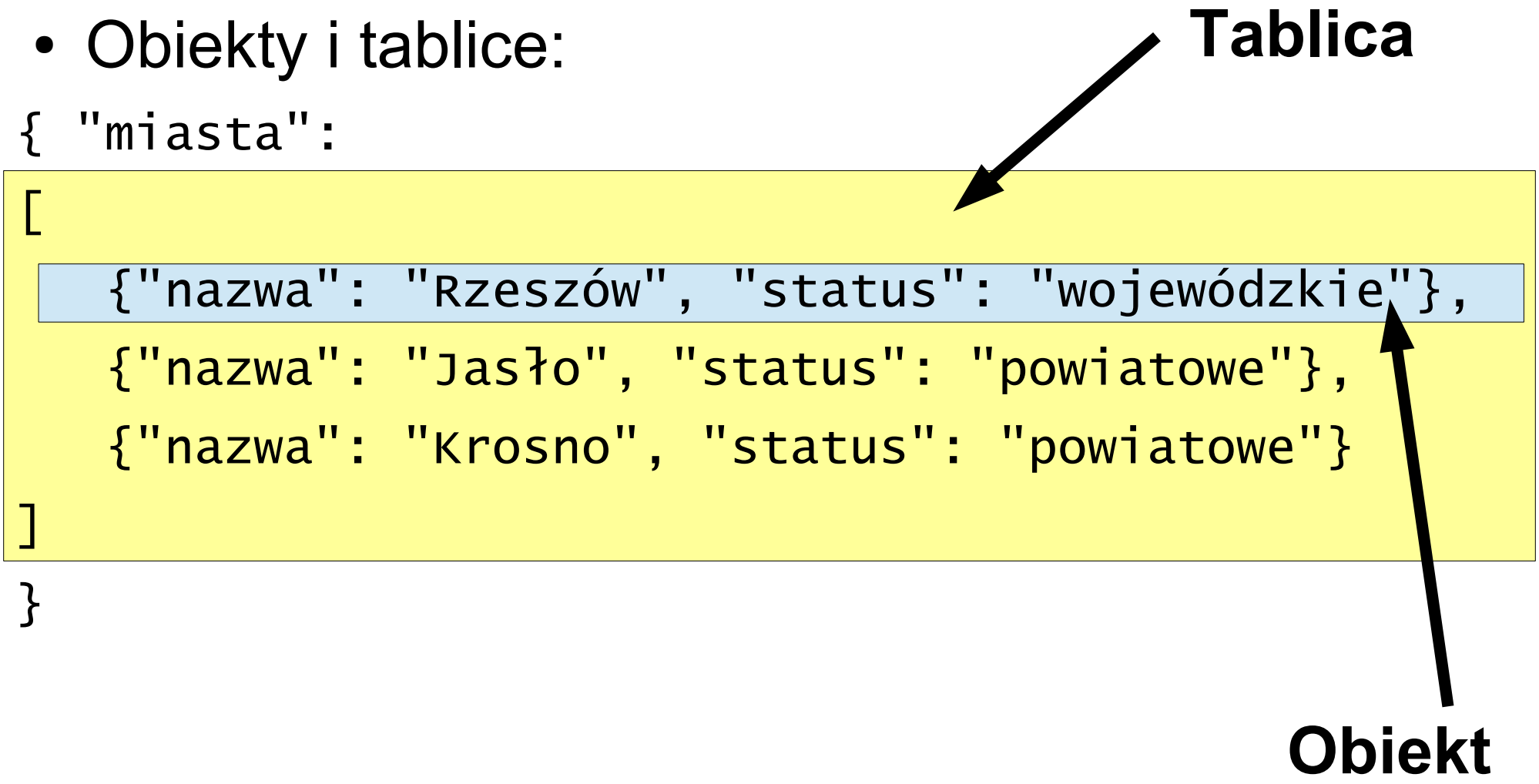
Dane w formacie JSON

- Przykład danych w formacie JSON:

```
{ "miasta":[
    {"nazwa": "Rzeszów", "status": "wojewódzkie"},
    {"nazwa": "Jasło", "status": "powiatowe"},
    {"nazwa": "Krosno", "status": "powiatowe"}
]}
```

Dane w formacie JSON

- Obiekty i tablice:



API w języku Java dla JSON

Pakiet javax.json:

https://docs.oracle.com/javaee/7/api/javax/json/package-summary.html

API w języku Java dla JSON

- Wybrane klasy i interfejsy:
  - **Json** - wykorzystanie: tworzenie obiektów przetwarzania JSON

```
StringReader reader = new StringReader(lancuch);
JsonParser parser = Json.createParser(reader);
```

API w języku Java dla JSON

- Metoda next() pozwala na parsowanie zdarzeń w wyspecyfikowanych lokalizacjach, np.:

```
{START_OBJECT

  "imie"KEY_NAME: "Jan"VALUE_STRING, "nazwisko"KEY_NAME: "Kowalski"VALUE_STRING,
"numer_indeksu"KEY_NAME: 99999VALUE_NUMBER,

  "indeks"KEY_NAME : [START_ARRAY

    {START_OBJECT "przedmiot"KEY_NAME: "matematyka"VALUE_STRING, "ocena"KEY_NAME:
"3.5"VALUE_STRING }END_OBJECT,

    {START_OBJECT "przedmiot"KEY_NAME: "fizyka"VALUE_STRING, "ocena"KEY_NAME:
"4.0"VALUE_STRING }END_OBJECT

  ]END_ARRAY

}END_OBJECT
```

```
Event zdarzenie = parser.next(); // START_OBJECT
zdarzenie = parser.next();       // KEY_NAME
zdarzenie = parser.next();       // VALUE_STRING
zdarzenie.getString();           // "Jan"
```

API w języku Java dla JSON

- Wybrane klasy i interfejsy:
  - **JsonReader** - wykorzystanie: odczyt JSON
  - **JsonArrayBuilder** - wykorzystanie: tworzenie tablicy JSON
  - **JsonWriter** – wykorzystanie: zapis JSON

API w języku Java dla JSON

- Wybrane klasy i interfejsy (c.d.):
  - **JsonValue** - wykorzystanie: reprezentacja wartości w JSON
  - **JsonObject** - wykorzystanie: reprezentacja obiektu w JSON
  - **JsonArray** - wykorzystanie: reprezentacja tablicy w JSON
  - **JsonString** - wykorzystanie: reprezentacja łańcucha znaków w JSON
  - **JsonNumber** - wykorzystanie: reprezentacja liczby w JSON

Wczytywanie danych JSON

- Przykład – dane JSON:

```
{ "daty":
[
  {"dzien": "11", "miesiac": "maj", "rok": "2017"},
  {"dzien": "8", "miesiac": "czerwiec", "rok": "2013"},
  {"dzien": "3", "miesiac": "lipiec", "rok": "2015"}
]
}
```

Wczytywanie danych JSON

- Przykład (cd.) - kod w języku Java:

```
InputStream is=new ByteArrayInputStream(json.getBytes());
JsonReader reader=Json.createReader(is);
JsonObject object=reader.readObject();
JsonArray results=object.getJsonArray("daty");
for (JsonObject result:results.getValuesAs(JsonObject.class))
{
  System.out.println(result.getString("dzien"));
  System.out.println(result.getString("miesiac"));
  System.out.println(result.getString("rok"));
}
```