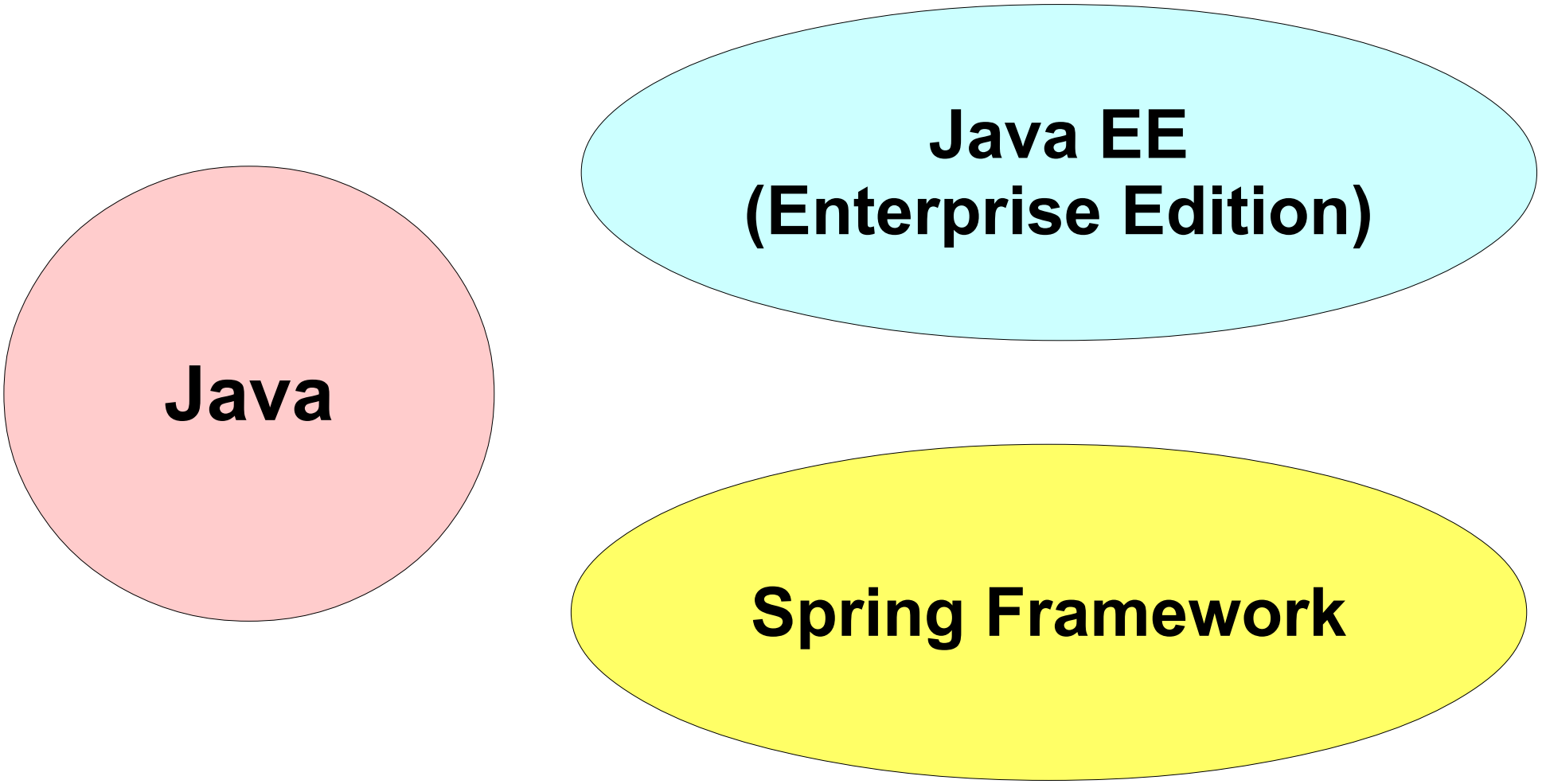
Programowanie portali biznesowych

WYKŁAD 1

Narzędzia i technologie



Modele przetwarzania dokumentów XML w języku Java

- Model SAX
- Model DOM

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

- 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.

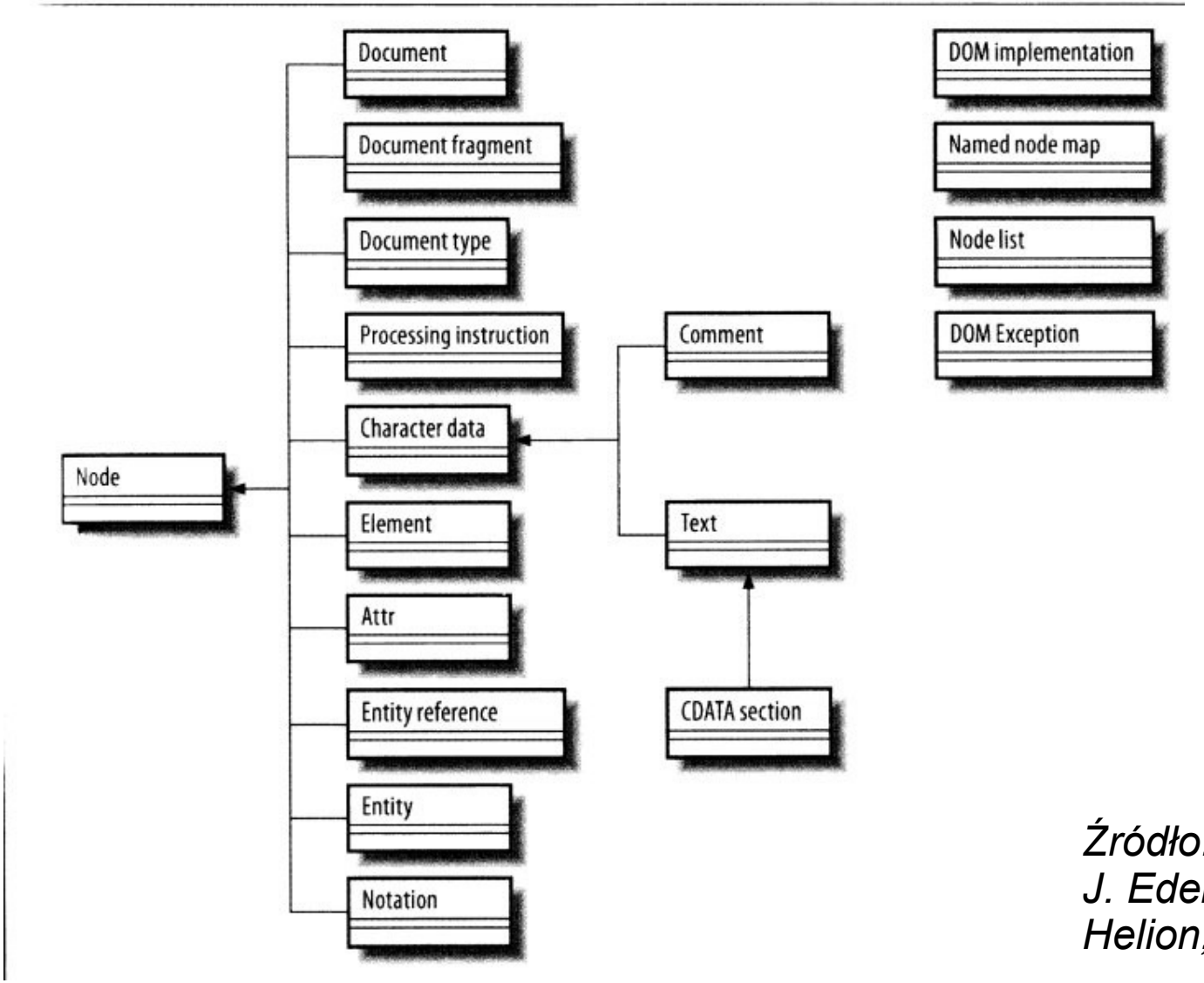
Model DOM

- 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

- 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

- **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

- **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

- **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

- **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

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

Model DOM

- Odczyt danych XML (przykład):

```
DocumentBuilderFactory
fabryka=DocumentBuilderFactory.newInstance();

DocumentBuilder parser=fabr.newDocumentBuilder();

Document dokument=parser.parse(plik);

Element root=dokument.getDocumentElement();

NodeList osoby=root.getElementsByTagName("osoba");

for(int i=0; i<osoby.getLength(); i++)

{

Object imie = ((Element)linie.item(i)).getAttribute("imie"));

Object nazwisko =
((Element)linie.item(i)).getAttribute("nazwisko"));

}
```

Model DOM

- Zapis danych XML (przykład):

```
DocumentBuilderFactory fabr=DocumentBuilderFactory.newInstance();
DocumentBuilder parser=fabr.newDocumentBuilder();
Document dokument=parser.newDocument();
Element root=dokument.createElement("osoby");
dokument.appendChild(root);
Element osoba=dok.createElement("osoba");
osoba.setAttribute("imie", "Jan");
osoba.setAttribute("nazwisko", "Kowalski");
root.appendChild(osoba);
Transformer
przekszt=TransformerFactory.newInstance().newTransformer();
Source wejscie=new DOMSource(dokument);
Result wyjscie=new StreamResult(plik);
przekszt.transform(wejscie, wyjscie);
```

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

Pakiety dla języka Java

- **SAX**
  - org.xml.sax
  - org.xml.sax.helpers, org.xml.sax.ext
- **DOM**
  - org.w3c.dom

Przetwarzanie danych w formacie JSON w języku Java

Specyfikacja JSR (Java Specification Request) 353:

https://jcp.org/en/jsr/detail?id=353

Przetwarzanie danych w formacie JSON w języku Java

- Rodzaje API:
  - API strumieniowe.
  - API obiektowe.

API strumieniowe

- W API strumieniowym parsowanie i generowanie danych JSON oparte jest o strumienie.
- API strumieniowe dostarcza parser zdarzeniowy działający na zasadzie „prośby” o kolejne zdarzenie.
- API strumieniowe jest API niskopoziomowym.
- API strumieniowe odpowiada modelowi SAX przetwarzania danych XML.

API strumieniowe

- Typy zdarzeń:
  - START_ARRAY
  - END_ARRAY
  - START_OBJECT
  - END_OBJECT
  - KEY_NAME
  - VALUE_STRING
  - VALUE_NUMBER
  - VALUE_TRUE
  - VALUE_FALSE
  - VALUE_NULL

API strumieniowe

- Wybrane klasy i interfejsy:
  - **Json** – klasa wykorzystywana do tworzenia obiektów przetwarzania JSON.
  - **JsonParser** – interfejs parsera strumieniowego JSON.
  - **JsonParserFactory** – interfejs fabryki parserów strumieniowych JSON.
  - **JsonGeneratorFactory** – interfejs fabryki generatorów strumieniowych JSON.

API strumieniowe

- Parsowanie danych JSON (przykład):

```
StringReader strumien = new StringReader(lancuch);
```

lub

```
FileInputStream strumien = new FileInputStream(plik);
```

```
JsonParser parser = Json.createParser(strumien);
```

lub

```
JsonParserFactory fabryka = Json.createParserFactory(null);
JsonParser parser = fabryka.createParser(strumien);
```

API strumieniowe

- 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 strumieniowe

- Generowanie danych JSON (przykład):

```
JsonGeneratorFactory fabryka = Json.createGeneratorFactory(null);
JsonGenerator generator = fabryka.createGenerator(strumien);

generator.writeStartObject()
.write("imie", "Jan")
.write("nazwisko", "Kowalski")
.writeEnd();
```

lub

```
generator.writeStartArray()
.writeStartObject().write("nazwisko", "Kowalski").writeEnd()
.writeStartObject().write("nazwisko", "Nowak").writeEnd()
.writeEnd();
```

API obiektowe

- API obiektowe wykorzystuje model obiektowy.
- API obiektowe jest API wysokopoziomowym.
- API obiektowe odpowiada modelowi DOM przetwarzania danych XML.

API obiektowe

- Wybrane interfejsy:
  - **JsonReader** – interfejs dla odczytu JSON
  - **JsonArrayBuilder** - wykorzystanie: tworzenie tablicy JSON
  - **JsonWriter** – interfejs dla zapisu JSON

API obiektowe

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

API obiektowe

- Odczyt danych JSON (przykład):

```
StringReader strumien = new StringReader(lancuch);
```

lub

```
FileInputStream strumien = new FileInputStream(plik);
```

```
JsonReader odczyt = Json.createReader(strumien);
```

lub

```
JsonReaderFactory fabryka = Json.createReaderFactory(null);
JsonReader odczyt = fabryka.createReader(strumien);
```

API obiektowe

- Odczyt danych JSON (przykład – cd.):

```
JsonObject obiekt = odczyt.readObject();
obiekt.getString("imie");
obiekt.getString("nazwisko");
```

lub

```
JsonArray tablica = odczyt.readArray();
JsonObject obiekt = tablica.getJsonObject(0);
// ...
```

API obiektowe

- Zapis danych JSON (przykład):

```
JsonBuilderFactory fabryka = Json.createBuilderFactory(null);
JsonArrayBuilder budowaTablic = fabryka.createArrayBuilder();
JsonObjectBuilder budowaObiektow = fabryka.createObjectBuilder();

JsonObject obiekt = budowaObiektow
.add("imie", "Jan").add("nazwisko", "Kowalski")
.build();
```

lub

```
JsonArray tablica = budowaTablic
.add(obiekt_1)
.add(obiekt_2)
//...
.build();
```

API obiektowe

- Zapis danych JSON (przykład – cd.):

```
JsonWriter zapis = Json.createWriter(strumien);
zapis.writeObject(obiekt);
```

lub

```
JsonWriterFactory fabryka = Json.createWriterFactory(null);
JsonWriter zapis = fabryka.createWriter(strumien);
zapis.writeObject(obiekt);
```