Programowanie urządzeń mobilnych

dr inż. Andrzej Grosser
na podstawie wykładu
dr inż. Juliusz Mikoda

Extensible Markup Language

- XML - Extensible Markup Language (Rozszerzalny Język Znaczników),
- Uniwersalny język formalny przeznaczony do reprezentowania różnych danych w strukturalny sposób,
- XML jest niezależny od platformy, co umożliwia łatwą wymianę dokumentów pomiędzy heterogenicznymi systemami,
- XML jest standardem rekomendowanym przez organizację W3C.

Poprawność dokumentu XML

- Mówimy o dokumencie, że jest poprawny **składniowo** (ang. well-formed), jeżeli jest zgodny z reguławami składni XML (np.: wszystkie znaczniki są domknięte).
- Mówimy o dokumencie, że jest poprawny **strukturalnie** (ang. valid), jeżeli jest zgodny z definicją dokumentu, tzn. dodatkowymi regułami określonymi przez użytkownika.
- Do precyzowania tych reguł służą specjalne języki (DTD, XML Schema oraz RELAX NG).

Poprawna składnia XML

- Na początku pliku musi znaleźć się deklaracja XML (może być poprzedzona komentarzem), musi posiadać wymagany atrybut **version** (1.0 lub 1.1), dodatkowe atrybuty to **encoding** oraz **standalone**.

- musi zawierać dokładnie jeden element główny (root).

- każdy element musi zaczynać się znacznikiem początku elementu np. **`<data>`** oraz kończyć identycznym znacznikiem końca elementu np. **`</data>`**, wyjątek stanowią elementy puste **`<element-pusty />`**, elementy te mogą zawierać atrybuty.

Poprawna składnia XML

- Nazwy elementów mogą zawierać znaki alfanumeryczne (litery a-z, A-Z oraz cyfry 0-9), znaki ideograficzne (ą, ó, ń, jednak należy unikać takich konstrukcji) oraz 3 znaki interpunkcyjne (podkreślenie _, łącznik -, kropka.).
- Znak dwukropka zarezerwowany jest dla identyfikacji przestrzeni nazw, której nazwa dopisywana jest przed nazwą elementu np. <przestrzeń1:element>.
- Nazwy elementów nie mogą zaczynać się od znaku łącznika -, kropki, czy cyfry. Dodatkowo nie mogą zaczynać się od xml, XML, xML itp. (wielkość liter bez znaczenia).

Poprawna składnia XML

- Elementy można zagnieżdżać w sobie i wtedy każdy element znajdujący się wewnątrz innego elementu jest nazywany "dzieckiem" tego elementu, a element wewnątrz którego znajdują się inne elementy zwany jest "rodzicem" tych elementów.
- Każdy element może zawierać atrybuty, które definiuje się w znaczniku początku elementu np. atrybutem elementu <news potw="yes"> jest atrybut o nazwie potw o wartości yes. Wartości atrybutów podaje się w cudzysłowach (”) albo apostrofach(').

Poprawna składnia XML

- Informacje, które zawiera element muszą być zapisane pomiędzy znacznikiem początku i końca elementu.

- W danych, atrybutach oraz nazwach elementów nie mogą pojawiać się niektóre znaki. Specyfikacja XML daje możliwość używania takich znaków z wykorzystaniem predefiniowanych odniesień jednostki: < = "<", > = ">", & ="&", ' = ', " = ".

- Wartości mogą zawierać znaki unicode, reprezentowane wartościami numerycznymi: Ū zostanie przesłany jako &#x016A lub &#362.

Poprawna składnia XML

- Bloki danych z predefiniowanymi znakami, które zawierają np. kod html lub xml mogą być zapisać w sekcji danych znakowych, która nie będzie przetwarzana przez analizator składni XML. <![CDATA[…]]>.

- W dokumencie XML można wykorzystywać komentarze, które zaczynają się znakami: <!--, a kończą: -->

- Specyfikacja XML zezwala na wstawianie instrukcji przetwarzania, które są wykorzystywane do przeniesienia informacji do aplikacji. Instrukcje przetwarzania rozpoczynają się znakami: <?, a kończą: ?>.

Przykładowy XML

```
<?xml version="1.0" encoding="UTF-8"?>
<ksiazka-telefoniczna
    kategoria="bohaterowie książek">
  <!-- Pan Kleks -->
  <osoba charakter="dobry">
    <imie>Ambroży</imie>
    <nazwisko>Kleks</nazwisko>
    <telefon>123-456-789</telefon>
  </osoba>
  <!-- Fryzjer -->
  <osoba charakter="zły">
    <imie>Alojzy</imie>
    <nazwisko>Bąbel</nazwisko>
    <telefon/>
  </osoba>
</ksiazka-telefoniczna>
```

Zapis XML

- XmlSerializer – interfejs służący do serializacji XML wersji 1.0.
  - startDocument(String encoding, Boolean standalone) – rozpoczyna dokument XML
  - endDocument() - kończy tworzenie dokumentu
  - comment(String text) – wstawia komentarz
  - startTag(String namespace, String name) – wstawia znacznik w wybranej przestrzeni nazw
  - endTag(String namespace, String name) – kończy wstawianie znacznika

Zapis XML

- text(String text) – wstawienie tekstu wewnątrz znacznika
- text(char[] buf, int start, int len) – jw.
- attribute(String namespace, String name, String value) wprowadzenie atrybutu wewnątrz znacznika otwierającego
- getDepth() – poziom znaczników
- getName() – nazwa obecnego znacznika
- setOutput(OutputStream os, String encoding) – strumień wyjściowy

Zapis XML

```
XmlSerializer serializer = Xml.newSerializer();
StringWriter writer = new StringWriter();
try {
      serializer.setOutput(writer); // ustawienie odbiorcy XML'a
      serializer.startDocument("UTF-8", false); // nagłówek XML
      serializer.startTag("", "strony"); // korzeń – root
      serializer.attribute("", "liczba", "1");
      serializer.comment("Startujemy"); // komentarz
      serializer.startTag("", "strona"); // pierwsze dziecko strony
      serializer.startTag("", "tytul");  // pierwsze dziecko strona
      serializer.text("Strona google");
      serializer.endTag("", "tytul");
      serializer.startTag("", "url");
      serializer.text("www.google.pl");
      serializer.endTag("", "url");
      serializer.endTag("", "strona");
      serializer.endTag("", "strony");
      serializer.endDocument();  // zakończenie przetwarzania dokumentu
      Log.i("Save XML", writer.toString());
} catch (Exception e) {
      throw new RuntimeException(e);
}
```

Parsowanie XML

- **Analiza składniowa** (parsowanie, ang. parsing) w informatyce i lingwistyce proces analizy tekstu, w celu ustalenia jego struktury gramatycznej i zgodności z gramatyką języka.

- **Parser** (inaczej analizator składniowy) w informatyce program dokonujący analizy danych wejściowych w celu określenia ich gramatycznej struktury w związku z formalną gramatyką.

- Dwie najważniejsze metody parsowania XML to:
  - **DOM**
  - **SAX**

DOM - **D**ocument **O**bject **M**odel

- DOM - obiektowy model dokumentu.
- Parser DOM dokonuje przekształcenia dokumentu XML do postaci modelu obiektowego - drzewa DOM.
- Dostęp do poszczególnych elementów dokumentu XML obywa się poprzez przechodzenie pomiędzy węzłami drzewa i odczytu ich wartości.
- Standard W3C DOM definiuje zespół klas i interfejsów, pozwalających na dostęp do struktury dokumentów oraz jej modyfikację poprzez tworzenie, usuwanie i modyfikację tzw. węzłów (ang. nodes).

SAX - **S**imple **A**PI for **X**ML

- SAX - Proste API dla XML
- Interfejs programistyczny do sekwencyjnego parsowania dokumentów XML. SAX jest popularną alternatywą dla DOM.
- Parser, który implementuje SAX, działa jako parser strumieniowy sterowany zdarzeniami. Użytkownik określa szereg metod, które obsługują zdarzenia pojawiające się podczas przetwarzania danych.
- Przetwarzanie z użyciem SAX jest jednokierunkowe - wcześniej przetworzone dane nie mogą być ponownie odczytane bez ponownego uruchomienia całej procedury.

DOM vs. SAX

DOM	SAX
reprezentacja w postaci drzewa	dokument czytany linia po linii
wolniejszy	szybszy
trudny w użyciu ale bardzo elastyczny	prosty w użyciu
kontrola dopiero po zakończeniu parsowania	proces parsowania kontrolowany na bieżąco
dobry do przetwarzania niesekwencyjnego	dobry do parsowania dużych dokumentów
możliwość odczytu i zapisu	dane tylko do odczytu

Klasy parsera DOM

- DocumentBuilderFactory - Definiuje fabrykę, API umożliwiające aplikacjom uzyskać parser.
  - DocumentBuilder newDocumentBuilder()
- DocumentBuilder - Definiuje interfejs do otrzymania obiektu Document DOM z pliku XML.
  - Document parse(InputStream is), String uri, File f

Klasy parsera DOM

- Document - Interfejs reprezentujący cały dokument XML. Zasadniczo, jest to korzeń drzewa dokumentu i zapewnia dostęp do podstawowych danych dokumentu.
  - Element getDocumentElement()

Klasy parsera DOM

- Node – interfejs ten jest podstawowym typem danych w strukturze drzewa DOM.
- Podtypy: Attr, Document, Element, Text
- Short getNodeType() : ATTRIBUTE_NODE, DOCUMENT_NODE, ELEMENT_NODE, TEXT_NODE i inne
- boolean   hasChildNodes()
- NodeList getChildNodes()
- Node getFirstChild()
- Node getNextSibling()
- Node getLastChild()

Klasy parsera DOM

- Node getParentNode() - rodzic węzła
- boolean hasAttributes()
- NamedNodeMap getAttributes()
- String getNodeValue() - wartość atrybutu
- Node insertBefore(Node newChild, Node refChild)
- Node removeChild(Node oldChild)
- void setNodeValue(String nodeValue)

Klasy parsera DOM

- CharacterData – rozszerzenie interfejsu Node o zbiór atrybutów i metod dostępu do danych znakowych w DOM
- int getLength()
- String getData()
- void insertData(int offset, String arg)
- void setData(String data)
- Text – interfejs używany do obsługi elementów zawierających tekst
- Obie klasy implementują: getNodeValue();

Klasy parsera DOM

- NodeList - interfejs zapewnia pozyskanie uporządkowanego zbiór węzłów, bez określenia kolejności elementów.
  - int getLength()
  - Node item(int index)
- NamedNodeMap – interfejs określający zbiór węzłów powiązanych z nazwami tych węzłów.
  - int getLength()
  - Node getNamedItem(String name)
  - Node item(int index)

Klasy parsera DOM

- Element – interfejs stanowiący element w dokumencie XML. Elementy mogą mieć atrybuty z nimi związane.
- boolean hasAttribute(String name)
- String getAttribute(String name)
- NodeList getElementsByTagName(String name)
- String getTagName() - getNodeName()

Przykład parsera DOM

```
public void parsingDOM(View view) {
  DocumentBuilderFactory factory =
        DocumentBuilderFactory.newInstance();
  try {
    DocumentBuilder builder = factory.newDocumentBuilder();
    // budowa struktury drzewa
    Document dom = builder
        .parse(getAssets().open("test.xml"));
    // główny element – root - korzeń
    Element root = dom.getDocumentElement();
    NodeList items = root.getElementsByTagName("osoba");
    // pętla po elementach o nazwie osoba
    for (int i = 0; i < items.getLength(); i++){
        // ...
    }

  } catch (Exception e) {
    throw new RuntimeException(e);
  }
}
```

Przykład parsera DOM

```
// odczytanie elementu - osoba
Node osoba = items.item(i);
// jeśli istnieją atrybuty
if (item.hasAttributes()) {
   // budowa struktury drzewa
   NamedNodeMap attributes = osoba.getAttributes();
   // pętla po atrybutach
   for (int k = 0; k < attributes.getLength(); k++){
      Node attribute = attributes.item(k);
      // nazwa oraz wartość atrybutu
      String name = attribute.getNodeName();
      String value = attribute.getNodeValue();

      Log.i("A: " + name, value);
   }
}
```

Przykład parsera DOM

```
// dzieci znacznika osoba
NodeList children = osoba.getChildNodes();
// pętla po elementach
for (int j=0;j<children.getLength();j++){
  Node child = children.item(j);
  // nazwa elementu
  String name = child.getNodeName();
  if (name.contentEquals("imie") ||
      name.contentEquals("nazwisko") ||
      name.contentEquals("telefon")) {
    // jeśli ma dzieci to ...
    if (child.hasChildNodes()) {
      String value =
          child.getFirstChild().getNodeValue();
      // wypisane jego wartości
      Log.i(name, value);
    } else Log.i(name, "Brak wartości");
  }
}
```

Przykład parsera DOM

- Wynik działania programu:

12-12 17:08:24.267: INFO/A: charakter: dobry
12-12 17:08:24.277: INFO/imie: Ambroży
12-12 17:08:24.287: INFO/nazwisko: Kleks
12-12 17:08:24.287: INFO/telefon: 123-456-789
12-12 17:08:24.297: INFO/A: charakter: zły
12-12 17:08:24.297: INFO/imie: Alojzy
12-12 17:08:24.319: INFO/nazwisko: Bąbel
12-12 17:08:24.319: INFO/telefon: Brak wartości

Budowa parsera SAX

- Parser strumieniowy sterowany zdarzeniami
- SAXParserFactory - Definiuje fabrykę API, która umożliwia aplikacjom skonfigurować i uzyskać parser SAX do analizy dokumentów XML.
- SAXParserFactory factory = SAXParserFactory.newInstance();
- SAXParser parser = factory.newSAXParser();
- parser.parse(xmlStream, saxHandler);

Budowa parsera SAX

- saxHandler – Obiekt klasy pochodnej po klasie DefaultHandler z zaimplementowanymi metodami do obsługi zdarzeń.
- Dostępne metody DefaultHandler do przedefiniowania
  - void startDocument() – Otrzymywanie powiadomienia o początku dokumentu.
  - void endDocument() – Otrzymywanie powiadomienia o końcu dokumentu.

Budowa parsera SAX

- void startElement(String uri, String localName, String qName, Attributes attributes) – Otrzymywanie powiadomienia o początku znacznika.
- void endElement(String uri, String localName, String qName) Otrzymywanie powiadomienia o końcu znacznika.

Budowa parsera SAX

- Dostępne metody DefaultHandler do przedefiniowania
  - void characters(char[] ch, int start, int length) – Powiadomienia o danych tekstowych wewnątrz elementu.
  - void warning(SAXParseException e) – Powiadomienia o ostrzeżeniach parsowania.
  - void error(SAXParseException e) – Powiadomienia o błędach parsowania.
  - void fatalError(SAXParseException e) – Powiadomienia o krytycznych błędach parsowania.

Budowa parsera SAX

- Dostępne metody DefaultHandler do przedefiniowania
  - void startPrefixMapping(String prefix, String uri) – Otrzymywanie powiadomienia o rozpoczęciu mapowania przestrzeni nazw.
  - void endPrefixMapping(String prefix) – Otrzymywanie powiadomienia o zakończeniu mapowania przestrzeni nazw.
  - void ignorableWhitespace(char[] ch, int start, int length) – Otrzymywanie powiadomień o zignorowaniu białych znaków w treści element.

Przykład parsera SAX

```
public void parsingSAX(View view)
{
    SAXParserFactory factory =
        SAXParserFactory.newInstance();

    try {
        SAXParser parser = factory.newSAXParser();
        SaxHandler handler = new SaxHandler();
         // Parsowanie dokumentu assest/test.xml
        parser.parse(getAssets().open("test.xml"),
                handler);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

Przykład parsera SAX

```
public class SaxHandler extends DefaultHandler {
  String tag;

  @Override
  public void startDocument() throws SAXException {
    super.startDocument();
    Log.i("startDocument", "");
  }

  @Override
  public void startElement(String uri, String localName,
        String name, Attributes attributes)
        throws SAXException {
    super.startElement(uri, localName, name, attributes);
    tag = localName;
    for (int i=0; i < attributes.getLength(); ++i) {
      Log.i("startElement", "Params: "
            + attributes.getLocalName(i) + "="
            + attributes.getValue(i));
    }
}
```

Przykład parsera SAX

```
public void characters(char[] ch, int start,
          int length) throws SAXException {
  super.characters(ch, start, length);
  int s = start, l = length;
  while (l>0 && (ch[s]==' ' || ch[s]=='\n'))
            { ++s; --l; }
  while (l>0 && (ch[s+l-1]==' ' || ch[s+l-1]=='\n'))
            { --l; }
  if (l > 0) {
    String str = new String(ch, start, length);
    Log.i("characters  ", "<" + tag + ">" + str);
  }
}

public void endElement(String uri, String localName,
          String name)  throws SAXException {
  super.endElement(uri, localName, name);
}
}
```

Budowa parsera XmlPullParser

- XmlPullParser jest interfejsem, który określa możliwości analizowania oferowane w API XMLPULL V1 - parser sterowany zdarzeniami.
- Do przesuwania się po zdarzenia służą dwie metody: next () oraz nextToken (). Ta druga jest operacją niższego poziomu pozwalającą na odczyt dodatkowych informacji jak: COMMENT, PROCESSING_INSTRUCTION, i inne.
- Odczyt danych:
  - void setInput(Reader in)
  - void setInput(InputStream inputStream, String inputEncoding)

Budowa parsera XmlPullParser

- String getInputEncoding() - Zwraca kodowanie danych wejściowych, jeśli wiadomo, null inaczej.
- int getColumnNumber() - Zwraca aktualny numer kolumny, zaczynając od 0.
- int getLineNumber() - Zwraca bieżący numer linii, począwszy od 1.
- int getDepth() - Zwraca aktualną głębokość elementu.
- int getEventType() - Zwraca typ bieżącego zdarzenia (START_TAG, END_TAG, TEXT, itp.)

Budowa parsera XmlPullParser

- boolean isEmptyElementTag() - Zwraca true, jeśli obecne zdarzenie jest START_TAG i tag jest pusty (np. <foobar/>).
- String getPrefix() - Zwraca prefiks bieżącego elementu.
- String getText() - Zwraca zawartość tekstu bieżącego zdarzenia jako String.
- boolean isWhitespace() - Sprawdza, czy bieżącego zdarzenia Tekst zawiera tylko białe znaki.

Budowa parsera XmlPullParser

- String getName() - Dla START_TAG lub END_TAG, zwraca nazwę bieżącego elementu.
- String getNamespace() - Zwraca URI przestrzeni nazw bieżącego elementu.
- int getAttributeCount() - Zwraca liczbę atrybutów obecnego start-tag lub -1, jeśli bieżącego typu zdarzenie nie jest START_TAG.

Budowa parsera XmlPullParser

- String getAttributeName(int index) - Zwraca lokalną nazwę atrybutu.
- String getAttributeNamespace(int index) - Zwraca przestrzeń nazw atrybutu (URI).
- String getAttributePrefix(int index) - Zwraca prefiks (przestrzeni nazw) określonego atrybutu lub null, jeśli ten element nie ma prefiksu

Budowa parsera XmlPullParser

- String getAttributeType(int index) - Zwraca typ atrybutu (CDATA).
- String getAttributeValue(int index) - Zwraca określoną wartość atrybutów.
- String getAttributeValue(String namespace, String name) - Zwraca wartość atrybutów określonych przez URI przestrzeni nazw i nazwę atrybutu.

Przykład parsera XmlPullParser

```
public void parsingPull(View view)
{
  XmlPullParserFactory factory=XmlPullParserFactory.newInstance();
  XmlPullParser xpp = factory.newPullParser();
  xpp.setInput( getAssets().open("test.xml"), null );
  int eventType = xpp.getEventType();
  while (eventType != XmlPullParser.END_DOCUMENT) {
    switch (eventType) {
      case XmlPullParser.START_DOCUMENT :
        System.out.println("Start document"); break;
      case XmlPullParser.END_DOCUMENT :
        System.out.println("End document"); break;
      case XmlPullParser.START_TAG :
        System.out.println("Start tag "+xpp.getName()); break;
      case XmlPullParser.END_TAG :
        System.out.println("End tag "+xpp.getName()); break;
      case XmlPullParser.TEXT :
        if (!xpp.isWhitespace()) System.out.println("Text "
              + xpp.getText());
    }
    eventType = xpp.next();
  }
}
```

Przykład parsera XmlPullParser

```
public void parsingPull(View view)
     throws XmlPullParserException, IOException {
   ArrayList<Osoba> tab = new ArrayList<Osoba>();
   XmlPullParserFactory factory
         = XmlPullParserFactory.newInstance();
   XmlPullParser xpp = factory.newPullParser();

   xpp.setInput( getAssets().open("test.xml"), null );
   int eventType = xpp.getEventType();
   while (eventType != XmlPullParser.END_DOCUMENT) {
     switch (eventType) {
        // ....................
     }
     eventType = xpp.next();
   }
}
```

Przykład parsera XmlPullParser

```
switch (eventType) {
  case XmlPullParser.START_TAG :
    if (xpp.getName()
            .contentEquals("ksiazka-telefoniczna")) {
      for (int i = 0; i < xpp.getAttributeCount(); ++i) {
        if (xpp.getAttributeName(i)
            .contentEquals("kategoria")) {
          System.out.println("Czytamy kategorię "
              +xpp.getAttributeValue(i));
          break;
        }
      }
    } else if (xpp.getName().contentEquals("osoba")) {
      Osoba o = new Osoba();
      if (o.parse(xpp)) tab.add(o);
    }
    break;
}
```

Przykład parsera XmlPullParser

```
public boolean parse(XmlPullParser xpp)
    throws XmlPullParserException, IOException {

  int intype = 0;
  int eventType = xpp.getEventType();
  while (eventType != XmlPullParser.END_DOCUMENT) {
    switch (eventType) {
      case XmlPullParser.START_TAG :
      // ....................
      case XmlPullParser.END_TAG :
      // ....................
      case XmlPullParser.TEXT :
      // ....................
    }
    eventType = xpp.next();
  }
  return false;
}
```

Przykład parsera XmlPullParser

```
switch (eventType) {

  case XmlPullParser.START_TAG :
    String name = xpp.getName();
    if (name.contentEquals("osoba")) readAtributes(xpp);
    else if (name.contentEquals("nazwisko")) intype = 1;
    else if (name.contentEquals("imie")) intype = 2;
    else if (name.contentEquals("telefon")) intype = 3;
    break;

case XmlPullParser.TEXT :
    switch (intype) {
      case 1 : nazwisko = xpp.getText(); break;
      case 2 : imie = xpp.getText(); break;
      case 3 : telefon = xpp.getText(); break;
    }
    break;
```

Przykład parsera XmlPullParser

```
switch (eventType) {
   // ...
  case XmlPullParser.END_TAG :
    if (xpp.getName().contentEquals("osoba")) return true;
    intype = 0;
    break;
}
```

```
private void readAtributes(XmlPullParser xpp) {
  //charakter = xpp.getAttributeValue("", "charakter");
  for (int i = 0; i < xpp.getAttributeCount(); ++i) {
    if (xpp.getAttributeName(i).contentEquals("charakter")){
      charakter = xpp.getAttributeValue(i);
      break;
    }
  }
}
```

JSON

- JSON – JavaScript Object Notation
- Podobnie jak XML jest formatem tekstowym pozwalającym na wymianę informacji.
- JSON jest zapisem wziętym z notacji obiektów języka JavaScript.
- Dane są wysyłane w postaci par atrybut – wartość.

JSON


```
{
  "imie": "Jan",
  "nazwisko": "Abacki",
  "wiek": 20,
  "adres": {
    "ulica": "21 2nd Street",
    "miejscowosc": "New York",
    "kodPocztowy": "10021-3100"
  },
  "telefony": [
    { "numer": "202 123-1234"},
    { "numer": "642 123-4567"}]
}
```

Typy podstawowe JSON

- Liczba (Number)
- Łańcuch znaków (String)
- Wartość logiczna (Boolean)
- Tablica (Array)
- Obiekt (Object)
- Wartość pusta null

JSON

```
private String wczytajPlik(InputStream inputStream) throws
IOException {
        BufferedReader reader = new BufferedReader(new
InputStreamReader(inputStream));
        String line;
        StringBuilder builder = new StringBuilder();
        while ((line = reader.readLine()) != null)
            builder.append(line);
        return builder.toString();
}
```

JSON

```
public void wczytaj(View view) {
        try {
            InputStream inputStream= getAssets().open("osoba.json");
            String data = wczytajPlik(inputStream);
            JSONObject json = new JSONObject(data);
            textView.setText(json.toString());

            Log.e("JSON", json.get("imie").toString());
            Log.e("JSON", json.get("nazwisko").toString());

            JSONArray array = (JSONArray) json.get("telefony");

            for(int i = 0; i < array.length(); ++i) {
                JSONObject elem = (JSONObject)array.get(i);
                Log.e("ARRAY", array.getString(i));
                Log.e("ARRAY", elem.get("numer").toString());
            }
        }
        catch (Exception exc) { exc.printStackTrace(); }
}
```

Usługi HTTP

- URLConnection – klasa abstrakcyjna służąca do pobierania i wysyłania danych z/do adresu URL.
- Klasy pochodne: HttpURLConnection, HttpsURLConnection, JarURLConnection.
- Po podaniu adresu tworzona jest odpowiednia klasa pochodna.
- Do utworzenia połączania służy klasa URL.
- Aby korzystać z połączenia internetowego należy nadać uprawnienia: **<uses-permission android:name="android.permission.INTERNET"> </uses-permission>**

Usługi HTTP – przykład

```
try {
    // Utworzenie obiektów połączenia
    URL url = new URL("http://www.wp.pl/");
    URLConnection urlConn = url.openConnection();
    // Pobranie strumienia danych - połączenie
    InputStream result = urlConn.getInputStream();
    if (result != null) {
      // Buforowany odczyt
       BufferedReader in = new BufferedReader(
          new InputStreamReader(result));
       while (in.ready()) { // wypisanie zawartości
         System.out.println(in.readLine());
       }
       in.close(); // zwolnienie buforów
    }
    result.close(); // zwolnienie strumienia
} catch (IOException ioe) {
    Log.e("getFromInternet", "Could not connect to server");
}
```

Usługi HTTP – przykład parsera

- Parsowanie danych pobranych ze strony internetowej:

```
try {
    URL url = new
        URL("http://adres.pl/pum/test.xml");
    URLConnection urlConn = url.openConnection();
    InputStream result = urlConn.getInputStream();
    parser(result);
    result.close();
} catch (IOException ioe) {
    Log.e("getFromInternet",
        "Could not connect to server");
} catch (Exception e) {
    e.printStackTrace();
}
```

Usługi HTTP – przykład parsera

```
protected void parser(InputStream result) throws Exception
{
    XmlPullParserFactory factory = XmlPullParserFactory.newInstance();
    XmlPullParser xpp = factory.newPullParser();
    xpp.setInput( result, null );
    int eventType = xpp.getEventType();
    String tag = "";
    boolean show = false;
    while (eventType != XmlPullParser.END_DOCUMENT) {
      switch (eventType) {
        case XmlPullParser.START_TAG :
          tag = xpp.getName();
          if (tag.contentEquals("nazwisko")) show = true;
          else if (tag.contentEquals("imie")) show = true;
          break;
        case XmlPullParser.END_TAG :  show = false; break;
        case XmlPullParser.TEXT :
          if (show) Log.i(tag, xpp.getText());
          break;
      }
      eventType = xpp.next();
    }
}
```

Wywołanie HTTP GET

- Służy do odbioru odpowiedzi z adresu URI, zawierającego parametry zapisane w sposób jawny (w adresie wywołania) .
- W wywołaniu można wysyłać dane w postaci Cookie.
- Klasa HttpClient służy do kontroli wysyłania zapytania.
- Klasa HttpGet służy do tworzenia zapytania typu GET.
- Klasa HttpResponse przechowuje odpowiedź uzyskaną ze wskazanego adresu.

Wywołanie HTTP GET

```
BufferedReader in = null;
try {
  DefaultHttpClient  client = new DefaultHttpClient();

  HttpGet request = new HttpGet();

  request.setURI(new URI("http://jakisadres.pl/pum/
          test.php ?send=1&aaa=bbb&eee=t"));

HttpResponse response = client.execute(request);
```

Wywołanie HTTP GET

```
in = new BufferedReader(
  new InputStreamReader(
    response.getEntity().getContent()));

StringBuffer sb = new StringBuffer("");
String line = "";
while ((line = in.readLine()) != null) sb.append(line);

  String page = sb.toString();
  System.out.println(page);

} finally {
  if (in != null) in.close();
}
```

Wywołanie HTTP POST

- Służy do odbioru odpowiedzi z adresu URI, zawierającego parametry zapisane w sposób niejawny (wysyłanych jako dane dodatkowe w wywołaniu) .
- Pozwala na przesłanie większej porcji danych (HTTP GET pozwala wysłać URL o długości maksymalnie 2048 znaków).
- Do obsługi takiego wywołania służy klasa HttpPost.
- Poza konstrukcją wywołania HttpPost pozostałe elementy wywołania pozostają bez zmian w porównaniu do wywołania HttpGet.

Wywołanie HTTP POST

```
BufferedReader in = null;
try {
  DefaultHttpClient httpClient = new DefaultHttpClient();
  HttpPost request =
   new HttpPost("http://jakisadres.pl/pum/test.php");

  List<NameValuePair> nameValuePairs =
        new ArrayList<NameValuePair>();
nameValuePairs.add(
      new BasicNameValuePair("send", "send"));
nameValuePairs.add(
      new BasicNameValuePair("id", "12345"));
nameValuePairs.add(
      new BasicNameValuePair("stringdata",
            "Android is Cool!"));
request.setEntity(
    new UrlEncodedFormEntity(nameValuePairs));

HttpResponse response = httpClient.execute(request);
```