

# Projekt TKOM - Dokumentacja

Konrad Miziński

20 stycznia 2013

## Spis treści

<b>1</b>	<b>Treść projektu</b>	<b>2</b>
<b>2</b>	<b>Cel projektu</b>	<b>2</b>
<b>3</b>	<b>Wymagania funkcjonalne</b>	<b>2</b>
<b>4</b>	<b>Wymagania нефункционалне</b>	<b>3</b>
<b>5</b>	<b>Przykładowy plik konfiguracyjny</b>	<b>3</b>
<b>6</b>	<b>Projekt realizacji</b>	<b>4</b>
6.1	Podział na moduły . . . . .	4
6.2	Moduł obsługi plików xml . . . . .	4
6.2.1	Analizator leksykalny(Lekser) . . . . .	4
6.2.2	Analizator składniowy(Parser) . . . . .	4
6.3	Moduł API użytkownika . . . . .	6

## 1 Treść projektu

Biblioteka java do obsługi plików konfiguracyjnych XML.

## 2 Cel projektu

Celem projektu jest dostarczenie zestawu klas języka Java pozwalających na:

- Odczyt z pliku konfiguracyjnego właściwości (reprezentowanych jako pary nazwa-wartość) pogrupowanych w sekcje.
- Dodawanie, usuwanie, modyfikacja właściwości. W szczególności tworzenie własnych obiektów zawierających zdefiniowane przez użytkownika właściwości.
- Serializacja i zapis do pliku zmodyfikowanych/utworzonych obiektów.

## 3 Wymagania funkcjonalne

- Pliki konfiguracyjne są formatu \*.xml o sztywno zdefiniowanej strukturze (opisanej w dalszej części dokumentacji).
- Właściwości zawarte w pliku konfiguracyjnym reprezentowane są przez pary nazwa-wartość.
- Właściwości są pogrupowane w sekcje.
- Sekcje reprezentowane są przez unikalne nazwy sekcji.
- Dopuszczalne są sekcje puste.
- Nazwy właściwości są unikalne w ramach sekcji.
- Wartości właściwości mogą być puste.
- Pliki konfiguracyjne reprezentowane są przez klasy implementujące interfejs PropertiesFile, sekcje przez klasy implementujące interfejs Section, a właściwości przez klasy implementujące interfejs Property; Kody źródłowe tych interfejsów zostały zamieszczone w dalszej części dokumentacji.
- Dostęp do klasy reprezentującej plik właściwości można uzyskać za pomocą obiektu serwisowego, udostępniającego metody odczytujące zapisujące do pliku \*.xml obiekty typu PropertiesFile.
- Za tworzenie obiektów klasy PropertiesFile odpowiada osobna klasa pełniąca rolę fabryki obiektów.
- Do biblioteki dołączona jest dokumentacja wygenerowana przez javadoc.

## 4 Wymagania niefunkcjonalne

- Biblioteka jest dostarczona w postaci archiwum typu JAR.
- Wraz z biblioteką jest dostarczony prosty program demonstrujący możliwości biblioteki (w postaci wykonywalnego pliku JAR). Program będzie pracował w trybie tekstowym.

## 5 Przykładowy plik konfiguracyjny

```
<?xml version="1.0" encoding="ASCII"?>
<properties>
  <section name="section1">
    <property name="name1">value1</property>
    <property name="name2">value2</property>
    <property name="name3"></property>
    <property name="name4"/>
  </section>
  <section name="section2">
    <property name="name1">value3</property>
    <property name="name5">value5</property>
  </section>
  <section name="section3"></section>
  <section name="section4"/>
</properties>
```

## 6 Projekt realizacji

### 6.1 Podział na moduły

Biblioteka składa się z 2 podstawowych modułów:

- Moduł obsługi plików xml - odpowiada za analizę leksykalną i składniową plików xml. Służy do tworzenia struktur danych odpowiadających poszczególnym składowym drzewa xml. Odpowiada za odczyt i zapis do pliku \*.xml. Jest niewidoczny dla użytkownika biblioteki.
- Moduł API użytkownika - dostarcza metod implementujących zadania biblioteki. Wykorzystuje obiekty z modułu pierwszego.

### 6.2 Moduł obsługi plików xml

#### 6.2.1 Analizator leksykalny(Lekser)

Lexer ma za zadanie rozbić wczytany z pliku text na leksemy. Na ich podstawie zostają rozpoznane tokeny. Rozróżnianie są następujące tokeny: `<?, ?>, <, </, >, />, =, ", WhiteCharacter, Letter`.

#### 6.2.2 Analizator składniowy(Parser)

Po analizie leksykalnej dane są poddawane analizie składniowej. Parser stara się utworzyć drzewo składni na podstawie następującej gramatyki (uproszczonej gramatyki drzewa xml):

```
Document = Prolog, [S], Element;
Prolog = '<?', Text, S, Attributes, '?>';
Element = '<', Text, Attributes, [S], ('>', Content, EndTag) | '/>';
Attributes = {S, Attribute}
Attribute = Text, Equals, ", ExtendedText, ";
Equals = [S], '=', [S];
EndTag = '</', Text, [S], '>';
Content = [S], ( {Element, [S]} | ExtendedText ), [S] ;
Text = Letter, {Letter};
ExtendedText = [ ExtendedLetter, {ExtendedLetter | S} ], ExtendedLetter;
ExtendedLetter = Letter | '&'; | '<'; | '>'; | '"'; | '''; ;
S = WhiteCharacter, {WhiteCharacter};
Letter = ? znak ascii z przedziału 0x21-0x7e bez znaków: <, >, ", ', & ?;
WhiteCharacter = ' ' | '\t' | '\n' | '\r';
```

Elementy drzewa xml są opakowane w klasy o następujących interfejsach:

```
package pl.waw.mizinski.xmlproperties.xml;
```

```
import java.util.List;
```

```

import pl.waw.mizinski.xmlproperties.exceptions.CanNotUpdateElementException;
import pl.waw.mizinski.xmlproperties.exceptions.MissingObjectException;

public interface XMLElement
{
    String getName();

    void setName(String name);

    boolean isComplexElement();

    boolean isEmpty();

    List<XMLAttribute> getAttributes();

    void setAttribute (XMLAttribute attribute);

    void removeAttribute (XMLAttribute attribute) throws MissingObjectException;

    List<XMLElement> getChildElements();

    void addChildElement(XMLElement element) throws CanNotUpdateElementException;

    void removeChildElement(XMLElement element) throws MissingObjectException;

    String getValue();

    void setValue(String value) throws CanNotUpdateElementException;
}

package pl.waw.mizinski.xmlproperties.xml;

public interface XMLAttribute
{
    String getName();

    void setName(String name);

    String getValue();

    void setValue(String value);
}

```

### 6.3 Moduł API użytkownika

Moduł ten ma za zadanie dostarczyć użytkownikowi klas pozwalających dodawać/usuwać/edytować właściwości zawarte w pliku konfiguracyjnym. Buduje on swoje klasy w oparciu o dane odczytne z plików i zapisane w klasach modułu pierwszego. Podczas zapisu do pliku dane najpierw są zamieniane na klasy plików xml(klasy modułu pierwszego), a następnie zapisywane na dysk. Fakt korzystania z formatu xml jakie prawie całkowicie ukryty przed użytkownikiem. Klasy wchodzące w skład tego modułu implementują następujące interfejsy:

```
package pl.waw.mizinski.xmlproperties.properties;

import java.util.List;

import pl.waw.mizinski.xmlproperties.exceptions.CanNotUpdateElementException;
import pl.waw.mizinski.xmlproperties.exceptions.MissingObjectException;

public interface PropertiesFile
{
    List<Section> getSections();

    Section getSectionByName(String name);

    void addSection (Section section) throws CanNotUpdateElementException;

    void removeSection(Section section) throws MissingObjectException;
}
```

```

package pl.waw.mizinski.xmlproperties.properties;

import java.util.List;

import pl.waw.mizinski.xmlproperties.exceptions.MissingObjectException;

public interface Section
{
    String getName();

    void setName(String name);

    boolean isEmpty();

    List<Property> getProperties();

    Property getPropertyByName(String name);

    String getPropertyValue(String name);

    void setProperty(Property property);

    void setProperty(String name, String value);

    void removeProperty(Property property) throws MissingObjectException;

    void removeProperty(String propertyName) throws MissingObjectException;
}

package pl.waw.mizinski.xmlproperties.properties;

public interface Property
{
    String getName();

    void setName(String name);

    String getValue();

    void setValue(String value);
}

```