1. Dop. Post. Programování DU2

Vypracovali: Štěpán Balcar, Filip Řepka

Zadání

Cílem práce bylo vytvořit knihovnu pro práci s konfiguračním souborem.

Pro vytvoření knihovny jsme museli překonat několik závažných problémů, které nám značně kompikovaly vývoj. Díky snaze obecně překonat následující logické problémy, které vyplývají ze zadání jsme následně vytvořili návrh systému.

Diskutované problémy:

Nejednoznačnost souboru

Tato knihovna musi umět načíst konfigurační soubor bez toho aby měla předem zadaný jeho formát. Bez znalosti formátu naní načtení souboru vůbec jednoznačné. Bez znalosti oddělovačů nemůžeme určit jak přesně napársovat hodnoty elementů.

Řešení:

Z této skutečnosti jsme dospěli k závěru, že jediná možná cesta vede přes rozdělení problému na dvě části, na Instanci a Formát. Instance bude obsahovat pouze předpársovaná data elementů a Formát na základě dalčích informací bude následně moci ověřit jejich validitu.

Interface Instance

Uživatel který bude knihovnu používat, by určině ocenil možnost pohodlného vytváření a následného výpisu konfiguračního souboru. Snažili jsem se vytvořit objektové a maximálně robustné API.

Interface Formátu

Pro vyspecifikování formátu konfiguračního souboru jsme se snažili vytvořit mnohem jednodušší způsob zadávání informací. Předpokládali jsme, že toto API bude sloužit pouze k testování správného formátu souboru.

Defaultní hodnoty elementů

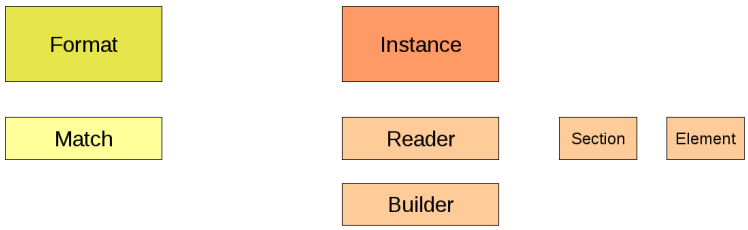
Po vytvoření formátu konfiguračního souboru by měla knihovna umět vytvořit a zapsat soubor s defaulními hodnotami. Musíme se ovšem spolehnout na to, že uživstel tyto hodnoty zadá, protože jinak bychom něměli co vypast. Tuto skutečnost jsme zohlednili ve vytváření API pro formát souboru.

Načítání Instance ze souboru

Pro načítání Instance ze souboru je potřeba aspoň částečně rozpársovat vstupní data. Nemůžeme se ovšem spolehnout na to, co se bude vyskytovat na konkrétním řádku. Nevíme jestli tam bude deklarace sekce, elementu nebo komentář, případně prázdná řádka. Všechny tyto možnosti musíme otestovat a vyzkoušet.

Návrh systému:

Systém byl rozdělen do dvou velkých částí, Formát a Instance.



Instance

Instance slouží k vytváření nebo načítání konfiguračního souboru. Obasahuje bohaté GUI pro jednoduchou objektovou manipulaci se Sekcemi a Elementy ale také provádí načítání této instance ze soubroru. Instance provede načtení a částečné rozpárosování souboru, ale ne všechny informace umí zpracovat. Na načtené hodnoty elementů se vždy dívá jako na řetězce. Instance ani nerozlisuje jestli je element pole a nebo jen promenná. Bez zadaného oddělovače nejsme tuto otázku rozhodnout.

Sekce a Element

Instance využívá dvou tříd Sekce a Element jako struktur pro vytváření konfiguračního souboru. Tyto třídy obsahují, identifikátory, komentáře, bílé znaky mezi každými dvěmi řetězci, prázdné řádky a celořádkové komnetáře. Načtením souboru do instance nepřijdeme o jedinou informaci a po vypsání Instance do souboru získáme úplně totožný soubor s původním. Tuto schopnost využíváme při ověření správnéhpo postavení instance na základě dat v souboru.

Reader a Builder

Reader je třída kterou využívá Instance k načítání a pársování. Třída operuje s regulárnímu výrazy a ve spolupráci s třídou Builder postaví celou instanci. Builder volá funkce na třídě Instance a tím vytváří aktuální Instanci konfuguračního souboru. Je obsluhován Rederem a vytváří jakousi podporu pro komfortní volání funkcí Instance ze třídy Reader.

Format

Formát specifikuje vlastnosti konfiguračního souboru a dokáže odkontorolovat jestli vytvořená nebo načtená instance odpovádá zadanému formátu. Třída Formát poskytuje rozhraní pri jednoduchou specifikaci formátu. Dále řídí třídu Matcher zajišťující fokečné rozpársování.

Matcher

Matcher provádí konečné rozpárosování hodnot elementů a dekódování odkazů. Kontoroluje jaké dimenze data jsou a jestli odpovídají správnému formátu. Hodnoty elementů můžou obsahovat odkazy na ostatní elementy. Třída také detekuje zda-li nenastal v odkazech kruh.

Implementace

Parsování

Načti soubor:

- načítej dokud nenarazíš na Sekci
- načítej Sekce dokud není konec souboru

Načti Sekci

- načti hlavičku Sekce
- načti tělo Sekce

Načti Element

- načti hlavičku Elementu
- načti tělo Elementu

Načti hlavičku Sekce

- načti bílé znaky před identifikátorem
- načti identifikátor
- načti bílé znaky za identifikátorem
- načti zbytek řádky (bílé znaky + komentář)

Načti tělo Sekce

- načítej dokud nenarazíš na Sekci nebo Element
- načítek Elementy dokud nenarazíš na Sekci

Načti hlavičku Elementu

- načti bílé znaky před identifikátorem
- načti identifikátor
- načti bílé znaky přče rovná se
- načti rovná se
- načti bíle znaky za rovná se
- načti hodnotu
- načti zbytek řádky (bílé znaky + komentář)

Načti tělo Elementu

- načítej dokud nenarazíš na Sekci nebo Element

Načítej dokud nenarazíš na Sekci nebo Element

- -je další řádka hlavička Sekce?
- zpět
- je další řádka hlavička Elementu?
- zpět
- je další řádka bílá řádka nebo komentář?
- ulož a pokračuj

2. Dokumentace:

Knihovna je napsána v programovacím jazyku Java ve vývojářském prostředí Netbeans.
Software je tvořen dvěmi nejdůležitějšími třídami.

Třídy

- Instance
- Format

Instance

- public void addCommentLinesBeforeFirstSection(String string)
- public void addSection(Section section) throws InstanceException
- public void addSection(int index, Section section) throws InstanceException
- public List¡Section¿ getSections()
- public Section getSection(String name)
- public Section getLastSection()
- public int getNumOfSections()
- public int getNumOfElements()
- public void ReadInstance() throws FileNotFoundException, IOException, InstanceException
- public List¡String¿ toStringLines()
- public void myPrint()

Tato třída zajišťuje vkládání komentářů na začátek souboru a vkádání sekcí. Pomocí metod get, jdou sekce opět získat, nebo lze získat celkový počet elementů v souboru. Metoda ReadInstance() zajišťuje načítání konfiguračního souboru.

Format

- public void setDelimiter(String delimiter) throws FormatException
- public String getDelimiter()
- public void addSection(String sectionName, boolean isRequired) throws FormatException
- public SectionFormat getSectionFormats(String sectionName)
- public List¡SectionFormat¿ getSectionFormat()

- public void addElement(String element, String sectionName, ElementTypes type, boolean isArrayType, String defaultValue, boolean isRequired) throws FormatException
- public boolean matchToFormat(Instance instance) throws FormatException
- public void writeDefaultConfiguration(File outputFile) throws IOException

Třída Formát definuje formát konfiguračního souboru. Nastavuje se zde oddšlovač a vkládají Sekce a Elementy. Argumenty funcí addSection nebo addElement umožňují zadefinovat jestli bude sekce nebo element povinná nebo jaké defaultní hodnoty bude element nabývat. Definuje se zde dimenze a typ Elemenentu. Funkce writeDefaultConfiguration umožňuje vypsat defaultní konfiguraci do souboru.

Zbylé metody jsou popsány v dokumetaci JavaDoc.

Příklad1

Zde můžeme vidět načtení konfiguračního souboru a vytvoření Instance.

```
File file = new File("tests/input1");

Instance instance = new Instance(file);

try {
    instance.ReadInstance();

} catch (InstanceException ex) {
    System.out.println("CHYBA InstanceException");
} catch (FileNotFoundException ex) {
    System.out.println("CHYBA FileNotFoundException");
} catch (IOException ex) {
    System.out.println("CHYBA IOException");
}

instance.myPrint();
```

Příklad2

V tomto příkladu načtenou konfiguraci porovnáváme s vytvořeným formátem. Soubor intut2 je soucasti knihovny.

```
File file = new File("tests/input2");

Instance instance = new Instance(file);

try {
    instance.ReadInstance();
```

```
     instance.myPrint();

 } catch (InstanceException ex) {
     System.out.println("CHYBA InstanceException");
 } catch (FileNotFoundException ex) {
     System.out.println("CHYBA FileNotFoundException");
 } catch (IOException ex) {
     System.out.println("CHYBA IOException");
 }

Format format = null;
try {
format = new Format();
format.setDelimiter(",");
format.addSection("Sekce 1", Format.IS_NOT_REQUIRED);
format.addElement("Option 1", "Sekce 1", ElementTypes.STRING,
        Format.IS_NOT_ARRAY, "1", Format.IS_REQUIRED);
format.addElement("oPtion 1", "Sekce 1", ElementTypes.STRING,
        Format.IS_NOT_ARRAY, "1", Format.IS_REQUIRED);

List<String> hodnoty = new ArrayList();
hodnoty.add("tyhle");
hodnoty.add("tamty");
ElementTypes enu = ElementTypes.ENUM;
enu.setCoorectValues(hodnoty);

format.addElement("frantisek 1", "Sekce 1", enu,
        Format.IS_ARRAY, "tyhle", Format.IS_NOT_REQUIRED);
format.addSection("$Sekce::podsekce", Format.IS_NOT_REQUIRED);
format.addElement("Option 2", "$Sekce::podsekce",
        ElementTypes.STRING, Format.IS_ARRAY, "1", Format.IS_REQUIRED);
format.addElement("Option 3", "$Sekce::podsekce",
        ElementTypes.STRING, Format.IS_ARRAY, "1", Format.IS_REQUIRED);
format.addElement("Option 4", "$Sekce::podsekce",
        ElementTypes.STRING, Format.IS_ARRAY, "1", Format.IS_REQUIRED);
format.addElement("Option 5", "$Sekce::podsekce",
        ElementTypes.STRING, Format.IS_ARRAY, "1", Format.IS_REQUIRED);
format.addSection("Cisla", Format.IS_NOT_REQUIRED);
format.addElement("cele", "Cisla", ElementTypes.SIGNED,
        Format.IS_NOT_ARRAY, "1", Format.IS_REQUIRED);
format.addElement("cele_bin", "Cisla", ElementTypes.UNSIGNED,
        Format.IS_NOT_ARRAY, "1", Format.IS_REQUIRED);
format.addElement("cele_hex", "Cisla", ElementTypes.UNSIGNED,
        Format.IS_ARRAY, "1", Format.IS_REQUIRED);
format.addElement("cele_oct", "Cisla", ElementTypes.UNSIGNED,
        Format.IS_NOT_ARRAY, "1", Format.IS_REQUIRED);
format.addElement("float1", "Cisla", ElementTypes.FLOAT,
        Format.IS_NOT_ARRAY, "1", Format.IS_REQUIRED);
```

```
format.addElement("float2", "Cisla", ElementTypes.FLOAT,
        Format.IS_NOT_ARRAY, "1", Format.IS_REQUIRED);
format.addElement("float3", "Cisla", ElementTypes.FLOAT,
        Format.IS_NOT_ARRAY, "1", Format.IS_REQUIRED);
format.addElement("float4", "Cisla", ElementTypes.FLOAT,
        Format.IS_NOT_ARRAY, "1", Format.IS_REQUIRED);
format.addSection("Other", Format.IS_NOT_REQUIRED);
format.addElement("bool1", "Other", ElementTypes.BOOLEAN,
        Format.IS_NOT_ARRAY, "1", Format.IS_REQUIRED);
format.addElement("bool2", "Other", ElementTypes.BOOLEAN,
        Format.IS_NOT_ARRAY, "1", Format.IS_REQUIRED);
format.addElement("bool3", "Other", ElementTypes.BOOLEAN,
        Format.IS_NOT_ARRAY, "1", Format.IS_REQUIRED);

File fileOut = new File("tests/output2");
try {
     format.writeDefaultConfiguration(fileOut);
} catch (IOException ex) {
        Logger.getLogger(
                Test2.class.getName()).log(
                Level.SEVERE, null, ex);
}

} catch (FormatException ex) {
    System.out.println(ex.getMessage());
}

try {
   format.matchToFormat(instance);
} catch (FormatException e) {
    System.out.println(e.getMessage());
    System.out.println("FormatException");
   return;
}

System.out.println("Instance odpovida Formatu");
```

Příklad3

Zde je vidět jak může být vytvořena vlastní Instance konfiguračního souboru. Kařdý oběkt obsahuje velké množství setů i getů.

```
Instance instance = new Instance();

Section section1 = new Section("Section1");
Element element1 = new Element("Element1");
section1.addElement(element1);
instance.addSection(section1);
```

```
Section section2 = new Section("Section2");
Element element2 = new Element("Element2");
section2.addElement(element2);
instance.addSection(section2);
```