X-definice 3.1

Úvod do programování v Javě

Author: **Václav Trojan**

Version: **3.1**
Date: **2017-12-03**

Obsah

1 Úvod

Účelem tohoto textu je úvod do programování X-definic. Součástí je ukázka doporučených postupů, jak programovat v jazyce Java typické úlohy. Detailní popis X-definic čtenář najde v dokumentu na internetové adrese: http://xdef.syntea.cz/tutorial/cs/userdoc/xdef3.1_ces.pdf.

Java dokumentace je na internetové adrese http://xdef.syntea.cz/tutorial/en/userdoc/xdef.

Tutoriál, s jehož pomocí si čtenář může osvojit základy X-definic je na adrese http://xdef.syntea.cz/tutorial/cs.

Knihovna pro práci s X-definicemi v Javě je v JAR souboru syntea_xdef3.1.jar. Stáhnout si jej můžete z adresy: http://www.xdefinice.cz/download/.

Úvod do konstručního režímu je na adrese:

http://xdef.syntea.cz/tutorial/cs/userdoc/xdef3.1_construction_mode_ces.pdf

Popis práce s X-komponentami je na adrese:

http://xdef.syntea.cz/tutorial/cs/userdoc/xdef3.1_X-component_ces.pdf

Na několika příkladech, odvozených ze zjednodušené úlohy zpracování objednávky v hypotetickém system, ukážeme způsob použití aparátu X-definic v jazyce Java. Předpokládáme, že čtenář ovládá základy programování v jazyce Java a je seznámen s jazykem XML. Rovněž předpokládáme, že se čtenář seznámil se základy technologie X-definic.

Dotazy, poznámky a hlášení chyb poslílejte na adresu: *xdef@syntea.cz*.

Aktuální verzi X-definic můžete stáhnout na adrese: *http://www.xdef.cz*.

2 Spuštění X-definic

X-definice je možné spustit ve dvou různých režimech. Buď v režimu, který provádí validaci a zpracuje vstupní XML data podle zadané X-definice (určitého modelu), nebo naopak použít zadaný model v X-definici a výsledný XML dokument je podle tohoto modelu zkonstruován. V každém případě musíme nejdříve přeložit zdrojový tvar X-definic a vytvořit objekt XDPool. Tento objekt je reentrantní (všechny hodnoty jsou po vytvoření konstantní) a proto je vhodné tento objekt uložit do statické proměnné nebo "singleton" objektu. Pro spuštění X-definic je třeba z objektu XDPool vytvořit objekt **XDDocument**.

Výsledkem spuštění X-definic je buď XML element, nebo instance Java třídy, v níž jsou přístupné hodnoty zpracovaného XML dokumentu (tzv. **X-komponenta**). Hlášení o chybách a stavech v průběhu zpracování se zapisuje do tzv. „reportéru". Reportér může záznamy ukládat buď do pracovní paměti (třída cz.syntea.xdef.ArrayReporter), nebo do souboru (třída cz.syntea.xdef.FileReporterWriter). V metodách pro spuštění X-definic můžeme zadat reportér jako parametr. Pokud reportér nezadáme (uvedeme „null"), pak když v průběhu zpracování jsou hlášeny chyby, program skončí výjimkou, ve které jsou chyby uvedeny. Byl-li reportér zadán, můžeme otestovat přítomnost chyb, resp. varovných hášení, metodami reporteru „errors()" resp. „errorWarnings()".

2.1 Režim validace (a zpracování) vstupního dokumentu

Režim validace a zpracování se spustí metodou „xparse“ instance třídy XDDocument. Vstupní data jsou metodě předána prvním parametrem. Je vhodné uvést jako druhý parametr reportér pro hlášení chyb.

Program nalezne v X-definici příslušný model (ten musí být deklarován atributem „root“ v záhlaví X-definice). Další zpracování probíhá podle tohoto modelu. Při zpracování se kontroluje výskyt jednotlivých prvků podle kvantifikátorů a textové hodnoty se kontrolují pomocí odpovídající validační metody. Při zpracování se v různých situacích volají příslušné příkazy deklarované ve skriptu X-definic.

Je-li požadován určitý prvek, ale není ve vstupních datech přítomen, oblásí se chyba a případně se zavolá příkaz ze sekce „onAbsence“. A obdobně, je-li naopak určitý prvek ve vstupních datech navíc, opět se ohlásí chyba do

reportéru a případně se zavolá příkaz sekce „onExcess“. Pokud je hodnota dat shledána validační metodou jako správná a je uvedena sekce „onTrue“, zavolá se příkaz této sekce. Jestliže je hodnota dat nevalidní, ohlásí se chyba a případně se zavolá příkaz sekce „onFalse“.

Po přečtení všech atributů se vyvolá příkaz sekce „onStartElement“ (je-li uvedena). Na konci zpracování elementu se zavolá příkaz sekce „finally“ (je-li uvedena).

Zvláštní význam má sekce „forget“. Tato sekce neobsahuje žádný příkaz, ale jejím zápisem se způsobí, že zpracovaný element se uvolní z paměti počítače. To umožňuje zpracovat značně rozsáhlá data, která by se během zpracování nevešla do pracovní paměti procesoru.

2.2 Režim konstrukce dokumentu

Režim konstrukce dokumentu („create mód“) nezpracovává vstupní XML dokument, ale podle zadaného modelu se vytváří výsledný XML dokument. Konstrukční režim se spustí metodou „**xcreate**“ instance třídy XDDocument. V parametru této metody musí být uveden model, podle něhož bude výsledný XML document vytvořen. Opět je vhodné zadat parametr s reportérem.

Jednotlivé části vytvářeného dokumentu se konstruhují podle zadaného modelu na základě tzv. kontextu. Kontext jsou data, která jsou v daném okamžiku tvorby použita pro tvorbu cílového XML objetu. Při tvorbě každé části vytvářeného dokumentu je (anebo není) k dispozici nějaký aktuální kontext. Pokud je ve Skriptu modelu XML objektu terý je konstruován zapsán příkaz sekce „create“, pak se pro tvorbu tohoto XML objektu stane kontextem hodnota, která je výsledkem příkazu sekce „create“ (tato hodnota musí být použitelná jako kontext). Tato hodnota se pak stane výchozí pro všechny vytvářené potomky. Požadovaný typ této hodnoty se liší podle druhu vytvářeného prvku. Často používanou metodou create sekce je XPath výraz, který pracuje s XML dokumentem, který byl přiřazen jako výchozí kontext pro tvorbu výsledku (tímto způsobem lze provést „transformaci“ XML dokumentu z kontextu do výsledné podoby).

Např. pro textové hodnoty a nebo pro atributy se hodnota kontextu převede na string. Tato hodnota je pak dosazena jako hodnota vytvářeného prvku. Pokud je hodnota kontext „null“ nebo prázdný string, atribut nebo textový uzel se nevytvoří.

Pro elementy mohou být jako kontext použity hodnoty následujících typů:

- Element …… výsledek je tvořen z obsahu tohoto elementu. Je-li hodnota „null“, pak se element nevytváří.
- celé číslo …… vytvoří se počet elementů odpovídající minimu z tohoto čísla a maxima v kvantifikátoru.
- String …….. jako kontext se vytvoří element s textovým uzlem vytvořeným z daného (tj. je-li požadovaným potomkem elementu textový uzel, vytvoří se z tohoto stringu)
- Container …….. je-li počet elementů v konteineru je 0, nic se nevytváří. Jinak se vyberou elementy a z nich se vytvoří počet elementů, odpovídající minimum z počtu elementů a maxima v sekci occurs.
- boolean …….. element se vytvoří jen pokud je hodnota true.
- null ..... element se nevytvoří

Příklad s explicitní specifikací sekce „create“:

```
<A xd:script="create 1"  x="string; create 'abc'">
  optional string; create 'def';
</A>
```

Výsledek bude:

```
<A x = "abc">def</A>
```

Podrobný popis použití konstrukřního režimu je na internetové adrese:
http://xdef.syntea.cz/tutorial/cs/userdoc/xdef3.1_construction_mode_ces.pdf

3 Nástroje pro programování X-definic v jazyce Java

3.1 Kompilace X-definic (vytvoření objektu XDPool)

Pro práci s X-definicemi budeme potřebovat objekt XDPool (popsaný rozhraním **cz.syntea.xdef.XDPool**). XDPool je binární objekt, který vznikne překladem X-definic ze zdrojového tvaru. Objekt XDPool lze vytvořit např. pomocí

statické metody **cz.syntea.xdef.XDFactory.compileXD(...)**. První parametr této metody je hodnota typu „java.util.Properties“, kde je možné nastavit některé parametry pro zpracování (viz...XXXX). Pokud je tento parametr „null“, použijí se hodnoty nastavené sytémem (metodou System.getProperties()). Další parametry pak specifikují X-definice, které se přeloží.

Objekt XDPool je reentrantní (všechna data v něm uložená jsou konstanty). Z toho vyplývá, že je možné jednu instanci tohoto objektu použít mnohonásobně, případně je možné ji sdílet ve více procesech. Vytvořený XDPool můžeme také uložit do souboru a z tohoto souboru zas vytvořit. Toho lze využít pro efektivní programování: objekt XDPool je možné připravit samostatně ze zdrojového tvaru a uložit jej do souboru. Nadále pak můžeme vytvářet XDPool z tohoto souboru (tj. nepotřebujeme zdrojový tvar X-definic a tvorba XDPool je pak mnohem rychlejší). Další možnost je vygenerovat z objektu XDPool pomocí metody „XDFactory.genXDPoolClass“ zdrojový tvar Java třídy, ze ketré je možné objekt XDPool získat. Tato třída pak může být součástí projektu distribuovaného ve zdrojovém tvaru.

*Pozn. V některých případech je třeba vytvořit objekt XDPool z různorodých zdrojů (například kombinace zdrojových souborů uložených v databázi, z Internetu a lokálního souborového systému apod.). V takovém případě musíme pomocí třídy XDFactory vytvořit nejdříve objekt popsaný rozhraním **cz.syntea.xdef.XDBuilder** a pomocí něho postupně inkrementálně připravit překlad pomocí metod **setSource**. Nakonec objekt XDPool vytvoříme metodou **genXDPool** volanou z XDBuilderu. Tento postup však ve většině případů není nutný a většinou vystačíme se statickou metodou „compileXD“ ze třídy XDFactory. Viz Java dokumentace třídy XDBuilder.*

3.2 Vytvoření instance objektu XDDocument

Pro práci s X-definicemi je třeba pomocí objektu XDPool vytvořit instanci objektu XDDocument, který je použit pro práci s požadovaným XML dokumentem. Tento objekt lze vytořit z objektu XDPool vytvořit pomocí metody „**createXDDocument**“. Parametr této metody určuje konkrétní výchozí X-definici, které je použita pro validační nebo konstrukční režim. Objekt XDDocument již není reentrantní a jeho instance se vztahuje k práci s konkrétními daty. Výsledkem procesu z objektu XDDocument je XML dokument (případně X-komponenta). Při zpracování X-definice vznikne také protokol (informace o chybách, varování atd). V objektu XDDocument jsou uloženy hodnoty proměnných deklarovaných v X-definicích. Kromě výsledného XML dokumentu je pak možné případně získat z objektu XDDocument i hodnoty proměnných (pomocí metody **getVariable**). Protokol o chybách, varování atd. je uložen v tzv. **reportéru**, který je předán procesoru X-definic pomocí parametru při spuštění.

Před zahájením zpracování X-definice je možné také nastavit do objektu XDDocument některé hodnoty. Hodnoty proměnných deklarovaných v X-definicích mohou být předány pomocí metody **setVariable.** Pomocí metody **setXDContext** můžeme předat hodnotu kontextu, který bude použit v režimu konstrukce. Metodou **setUserObject** můžeme nastavit před spuštěním uživatelský objekt, který může být použit v externích uživatelských metodách k propojení programu s externími uživatelskými daty. Vlastní zahájení zpracování se vyvolá metodou **xparse** (validace a zpracování XML dokumentů) nebo **xcreate** (konstrukce XML objektů).

3.3 Ostatní Java objekty pro práci s X-definicemi

Při práci X-definice vnitřně vytváří dočasné řídící objekty, které se vztahují k právě zpracovávané části XML objektu. Všechny tyto objekty vycházejí ze společného rozhraní **cz.syntea.xdef.proc.XXNode** (to platí i o XDDocument). Jedná se o objekt **cz.syntea.xdef.proc.XXElement**, který je vytvořen při zpracování XML elementu a objekt **cz.syntea.xdef.proc.XXData**, který se vytvoří při zpracování textových hodnot XML objektů (tj. atributů nebo textových uzlů). Tyto objekty vznikají dynamicky až v průběhu zpracování, po zpracování zanikají a mohou být přístupné v externích „uživatelských“ metodách v jazyce Java volaných ze Skriptu. V těchto metodách je často třeba pracovat s detailní informací o stavu zpracování a zpracovávaných objektech nebo je ovlivnit. Proto může být externí metodě předán jako parametr pracovní objekt XXNode (případně XXElement resp. XXData), který je vytvořen při zpracování příslušného XML objektu. Pokud to externí metoda vyžaduje, může být tento objekt externí metodě předán (vždy se zapisuje jako první parametr, ale ve Skriptu se tento parametr se neuvádí). Musí být uveden v deklaraci příslušné externí metody a X-definice předají automaticky tuto hodnotu podle toho, je-li v externí metodě požadována).

Hodnoty parametrů a proměnných, deklarovaných v X-definicích, jsou reprezentovány rozhraním **cz.syntea.xdef.XDValue**. Pro běžné hodnoty v jazyce Java (čísla, String apod.) mohou být parametry externích procedur uvedeny jako seznam parametrů běžným způsobem a překladač automaticky zajistí jejich konverzi z vnitřního tvaru do objektu Java. Druhý způsob předání parametrů externím metodám je možnost uvést parametr

procedury jako pole hodnot XDValue. V takovém případě se předá metodě seznam objektů s požadovanými hodnotami na základě seznamu parametrů, uvedených v konkrétním volání externí metody ve Skriptu. Jedna externí metoda tak může odpovídat volání této metody ze Skriptu s různým počtem parametrů, jejichž hodnoty se dosadí do pole hodnot, které je pak metodě předáno.

V některých speciálních případech programátor potřebuje znát i hodnoty obsažené v modelech v X-definicích (například limity výskytu atd.). Tyto objekty jsou přístupné pomocí rozhraní **cz.syntea.xdef.model.XMDefinition**, **cz.syntea.xdef.model.XMElement** a **cz.syntea.xdef.model.XMData**. Jsou to vlastně objekty odpovídající modelům deklarovaným v X-definicích, nebo jejich částem. Přístup k těmto objektům je umožněn z XXNode pomocí metod **getXMDefinition**, **getXMElement**, **getXMData**. XDPool je možné získat metodou **getXDPool**.

Některé parametry pro zpracování (např. jazyková mutace, výchozí kódová stránka pro soubory, tabulky reportů atd.) je možné před spuštěním procesoru X-definic nastavit pomocí třídy **cz.syntea.xdef.reporter.SMagager** a ovlivnit pomocí systémových **Properties**. Např. je-li nastaveno „xdef.doctype“ na hodnotu „false“, procesor X-definic nedovolí při zpracování XML dokumentu výskyt „DOCTYPE“, což je důležité z bezpečnostního hlediska, nebo nastavením „xdef.warnings“ na hodnotu „false“ jsou varovná hlášení ignorována.

Chybová a informační hlášení jsou procesorem X-definic předávána pomocí **reportéru**, který umožňuje zápis protokolu o zpracování ve formě objektů **cz.syntea.xdef.reporter.Report** do souborů či do pracovní paměti počítače.

Hodnoty datumu a času jsou v procesoru X-definic implementovány třídami **cz.syntea.xdef.sys.SDatetime**.
Pro časové intervaly je použita třída **cz.syntea.xdef.sys.SDuration**.

Pro práci s textovými hodnotami popsanými pomocí rozšířené gramatiky BNF slouží třída **cz.syntea.xdef.sys.BNFGrammar**.

Pro účely testování je k dipsozici třída **cz.syntea.xdef.sys.AbstractTester**, která umožňuje efektivní tvorbu testovacích programů.

4 Příklady

V této kapitole rozebereme několik příkladů postavených na konkrétním zpracování hypotetického seznamu objednávek. Naše úloha zpracuje vstupní soubor objednávek a zkontroluje jejich správnost. V příkladech uvádíme jak zdrojový text Java programu, tak i X-definice a vstupní data a jsou ve funkční podobě dostupné v souborech, které jsou dodávány spolu s touto učebnicí na http://www.xdefinice.cz/cz/download.

4.1 Úloha 1

Nejdříve zkontrolujeme, zda je struktura objednávek formálně správná. Vstupní data jsou v adresáři „task1/input/“. Není-li ve vstupních datech nalezena chyba, bude zpracovaná (zkontrolovaná) objednávka uložena do adresáře „task1/output“, v opačném případě, bude do adresáře „task1/errors/“ uložen soubor s popisem chyb.

X-definice a zdrojový kód Java použité v jednotlivých variantách tohoto příkladu jsou uloženy v adresáři „src/task1/“. Na jednotlivých variantách řešení našeho příkladu ukážeme postupně různé možnosti použití X-definic i podpůrných prostředků jejich implementace.

4.1.1 Varianta 1

Začněme nejjednodušší variantou. Pomocí X-definice zkontrolujeme správnost vstupních dat a není-li nalezena chyba, zapíšeme data do adresáře „output“, jinak zapíšeme protokol o chybách vytvořený překladem X-definic (v textové podobě) do adresáře „task1/errors/Order_err.txt“.

Vstupní soubor formálně správné objednávky „task1/input/Order.xml“:

```
<Order Number="123" CustomerCode="ALFA">
   <DeliveryPlace>
      <Address Street="Oldroad" House="5" City="NewTown" ZIP="32321" />
   </DeliveryPlace>
   <Item ProductCode="0002" Quantity="2" />
   <Item ProductCode="0003" Quantity="1" />
</Order>
```

X-definice popisující vstupní data („src/task1/Order1.xdef"):

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1" name="Order" root="Order">
<Order Number="int" CustomerCode="string(1,20)">
   <DeliveryPlace>
      <Address Street="string(2,100)"
         House="int(1,9999)"
         City="string(2,100)"
         ZIP="num(5)"
      />
   </DeliveryPlace>
   <Item xd:script="1..10" ProductCode="num(4)" Quantity="int(1,1000)"/>
</Order>
</xd:def>
```

Při tvorbě programu musíme nejdříve získat objekt XDPool přeložený ze zdrojového tvaru X-definic. Protože tento objekt obsahuje jen konstaty, můžeme jej uložit do statické finální proměnné. Z něho pak vytvoříme objekt XDDocument, který budeme potřebovat pro další práci. Pro protokol o chybách připravíme „reportér" pro zápis protokolu o chybách (ArrayReporter ukládá protokol do pracovní paměti počítače). Metodou „xparse" spustíme zpracování vstupních dat podle X-definice. Otestujeme, zda byly hlášeny nějaké chyby (zda vstupní data odpovídají X-definici) a podle výsledku testu provedeme zápis protokolu o chybách nebo zpracovaného dokumentu (pomocí utility „KXmlUtils.writeXml") do příslušných adresářů.

Program v jazyce Java (soubor „src /task1/Order1.java"):

```
package task1;

import cz.syntea.xdef.reporter.ArrayReporter;
import cz.syntea.xdef.xml.KXmlUtils;
import cz.syntea.xdef.XDDocument;
import cz.syntea.xdef.XDFactory;
import cz.syntea.xdef.XDPool;
import java.io.PrintStream;
import org.w3c.dom.Element;

public class Order1 {
        // Compile X-definitions to XDPool
        static final XDPool xpool = XDFactory.compileXD(null, "src/task1/Order1.xdef");

        public static void main(String[] args) throws Exception {
                // Create the instance of XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Order");

                // Create reporter
                ArrayReporter reporter = new ArrayReporter();

                // Run validation mode (you can also try task1/input/Order_err.xml)
                Element result = xdoc.xparse("task1/input/Order.xml", reporter);

                // Check if an error was reported
                if (reporter.errors()) {
                        // Print errors to the file
                        PrintStream ps = new PrintStream("task1/errors/Order_err.txt ");
                        reporter.printReports(ps);
                        ps.close();
                        System.err.println("Incorrect input data");
                } else {
                        // No errors, write the processed document to the file
                        KXmlUtils.writeXml("task1/output/Order.xml", result);
                        System.out.println("OK");
                }
        }
}
```

Výstupní soubor formálně správné objednávky „task1/output/Order.xml" :

```
<?xml version="1.0" encoding="UTF-8"?>
<Order CustomerCode="ALFA" Number="123">
   <DeliveryPlace>
      <Address City="NewTown" House="5" Street="Oldroad" ZIP="32321"/>
   </DeliveryPlace><Item ProductCode="0002" Quantity="2"/>
   <Item ProductCode="0003" Quantity="1"/>
</Order>
```

Pokud by vstupní data obsahovala chybu (v našem případě jsme změnili atribut „Quantity" v prvním elementu „Item" vstupního dokumentu tak, že není číselný:

```
...
   <Item CommodityCode="0002" Quantity="xx"/>
...
```

pak bude chyba zapsána do souboru: *„task1/errors/Order_err.txt“:*

```
E XDEF809: Incorrect value of 'int'; line=6; column=38;
source="file:/D:/cvs/DEV/java/examples/task1/input/Order_err.xml"; xpath=/Order/Item[1]/@Quantity; X-
position=Order#Order/Item/@Quantity
```

4.1.2 Varianta 2: uložení přeložených X-definic do binárního souboru

Uveďme ještě ukázku jak postupovat, pokud bychom chtěli uložit objekt XDPool vytvořený ze zdrojového tvaru X-definic do binárního souboru a pak jej použít. V jednom běhu programu samostatně přeložíme X-definice a vytvořený objekt XDPool uložíme do souboru „src/task1/Order1a.xp“.

Program „src/task1/Order1a_gen.java“ pro generování binárního souboru „src/task1/Order1a.xp“ ze zdrojového tvaru X-definice:

```
package task1;

import cz.syntea.xdef.XDFactory;
import cz.syntea.xdef.XDPool;
import java.io.FileOutputStream;
import java.io.OutputStream;
import java.util.Properties;

public class Order1a_gen {

        public static void main(String[] args) throws Exception {
                // Compile XDPool from the X-definition
                Properties props = new Properties();
                XDPool xpool = XDFactory.compileXD(props, "src/task1/Order1.xdef");
                // Write XDPool to the file
                OutputStream outstr= new FileOutputStream("src/task1/Order1a.xp");
                xpool.writeXDPool(outstr);
                outstr.close();
        }
}
```

V dalších bězích programu pak vytvoříme XDPool z tohoto souboru. Samozřejmě bude čas na vytvoření instance objektu XDPool výrazně kratší, než při jeho generaci ze zdrojového tvaru X-definic. Program se bude lišit jen v příkazu získání objektu XDPool (příkazem „readXDPool“).

Uvědomme si, že s každou novou verzí X-definic je vhodné znova vytvořit binární soubor z XDPool přeloženého ze zdrojového tvaru, protože se implementačně mohou v jednotlivých verzích lišit.

Program „src/task1/Order1a.java“ používá pro tvorbu objektu XDPool binární soubor „src/task1/Order.xp“, který byl vytvořen spuštěním programu „Order1a_gen“. Jinak je program identický s předchozí variantou:

```
package task1;

import cz.syntea.xdef.reporter.ArrayReporter;
import cz.syntea.xdef.xml.KXmlUtils;
import cz.syntea.xdef.XDDocument;
import cz.syntea.xdef.XDFactory;
import cz.syntea.xdef.XDPool;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.PrintStream;
import org.w3c.dom.Element;

public class Order1a {
        public static void main(String[] args) throws Exception {
                // Get XDPool object from the file
                InputStream in = new FileInputStream("src/task1/Order1a.xp");
                XDPool xpool = XDFactory.readXDPool(in);
                in.close();

                // Create instance of XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Order");

                // Create reporter
                ArrayReporter reporter = new ArrayReporter();
```

```
                // Run validation mode (you can also try task1/input/Order_err.xml)
                Element result = xdoc.xparse("task1/input/Order.xml", reporter);

                // Check if an error was reported
                if (reporter.errorWarnings()) {
                        // Print errors to the file
                        PrintStream ps = new PrintStream("task1/errors/Order_err.txt");
                        reporter.printReports(ps);
                        ps.close();
                        System.err.println("Incorrect input data");
                } else {
                        // No errors, write the processed document to the file
                        KXmlUtils.writeXml("task1/output/Order.xml", result);
                        System.out.println("OK");
                }
        }
}
```

4.1.3 Varianta 3: vytvoření Java třídy z XDPool

Z přeložených X-definic můžeme dokonce vygenerovat také zdrojový tvar Java třídy obsahující XDPool. Ta pak může být součástí tříd projektu.

Program „src/task1/Order1b_gen.java“ pro generování Java třídy „src/task1/Order1bXD.java“ obsahující přeložený XDPool ze zdrojového tvaru X-definice:

```
package task1;

import cz.syntea.xdef.XDFactory;
import cz.syntea.xdef.XDPool;

public class Order1b_gen {

        public static void main(String[] args) throws Exception {
                // Compile X-definitions to XDPool
                XDPool xpool = XDFactory.compileXD(null, "src/task1/Order1.xdef");
                // Generate Java source class with the compiled XDPool
                XDFactory.genXDPoolClass(xpool, "src", "task1.Order1bXP", null);
        }
}
```

Z vygenerované třídy Order1bXD pak můžeme objekt XDPool získat statickou metodou getXDPool():

Program „src/task1/Orderb.java“ používá pro získání objektu XDPool vygenerovanou Java třídu „src/task1/Order1bXD.java“ (tu je po vygenerování nutné přeložit). Jinak je program identický s předchozí variantou:

```
package task1;

import cz.syntea.xdef.reporter.ArrayReporter;
import cz.syntea.xdef.xml.KXmlUtils;
import cz.syntea.xdef.XDDocument;
import cz.syntea.xdef.XDPool;
import java.io.PrintStream;
import org.w3c.dom.Element;

public class Order1b {
        public static void main(String[] args) throws Exception {
                // Get XDPool from the class Order1bXD (generated by Order1b_gen)
                XDPool xpool = Order1bXP.getXDPool();

                // Create instance of XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Order");

                // Create reporter
                ArrayReporter reporter = new ArrayReporter();

                // Run validation mode (you can also try task1/input/Order_err.xml)
                Element result = xdoc.xparse("task1/input/Order.xml", reporter);

                // Check if an error was reported
                if (reporter.errorWarnings()) {
                        // Print errors to the file
                        PrintStream ps = new PrintStream("task1/errors/Order_err.txt");
                        reporter.printReports(ps);
                        ps.close();
```

```
                        System.err.println("Incorrect input data");
                } else {
                        // No errors, write the processed document to the file
                        KXmlUtils.writeXml("task1/output/Order.xml", result);
                        System.out.println("OK");
                }
        }
}
```

4.1.4 Varianta 4: generování XML dokumentu a chyb ve dvou krocích

V dalších variantě našeho příkladu budeme požadovat protokol o chybách ve tvaru XML. Požadovaný výstup bude v XML dokumentu „Errors“, který bude mít tvar popsaný v následující ukázce:

Formát výstupního souboru pro chybnou objednávku:

```
<Errors>
  <Error ErrorCode="nnn" Customer="ALFA" Line="n" Column="m" />
...
</Errors>
```

Do X-definice tedy přidáme model popisující požadovaný tvar chybového dokumentu. Ve skriptu nadefinujeme v deklarační části potřebné pomocné proměnné a procedury. Volání procedur pro generaci chyb zapíšeme jako akce, volané v případě výskytu chybových událostí při zpracování vstupních dat (sekce „onFalse“ se zavolá při chybě formátu dat a sekce „onAbsence“ se zavolá když pozadovaný objekt chybí). Data o chybách budeme průběžně zapisovat do proměnné „errors“ typu Container.

V této ukázce předvedeme rozdělení zpracování vstupních dat a tvorbu dokumentu s chybami do dvou kroků. V prvním kroku provedeme zpracování vstupních dat a uložení informací o chybách do proměnné „errors“ typu „Container“. V Java programu zjistíme počet záznamů o chybách. Pokud nenastaly chyby, zapíšeme výsledný dokument zpracovaný příkazem „xparse“. Pokud nenastaly chyby, výsledek zapíšeme do výstupního souboru. Pokud chyby nastaly, spustíme procesor X-definic znova v režimu vytváření modelu „Errors“ příkazem „xcreate“ přičemž jako kontext pro vytváření výsledku je použita proměnná „errors“, jejíž hodnota byla nastavena v předchozím zpracování. Procesor X-definic automaticky použije pojmenované položky pro tvorbu atributů a sekvenci položek pro tvorbu elementů. Výsledek bude XML element s chybami, který zapíšeme do chybového souboru.

X-definice je umístěna v souboru „src/task1/Order2.xdef“:

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1" name="Order" root="Order">

<xd:declaration>
   Container errors = []; /* The Container with error items. */
   void err(int code) {
     Container c = []; /* Create new empty Container. */
     /* Setting named Values. */
     c.setNamedItem("Customer", xpath("/Order/@CustomerCode"));
     c.setNamedItem("ErrorCode", code);
     c.setNamedItem("Line", getSourceLine());
     c.setNamedItem("Column", getSourceColumn());
     /* Add the item to the sequential part of the container "errors". */
     errors.addItem(c); /* Add item */
   }
</xd:declaration>

<Order Number="int; onFalse err(1); onAbsence err(2);"
   CustomerCode="string; onAbsence err(3);">

   <DeliveryPlace xd:script="onAbsence err(11);">
      <Address Street="string(2,100); onFalse err(12); onAbsence err(13);"
         House="int(1,9999); onFalse err(14); onAbsence err(15);"
         City="string(2,100); onFalse err(16); onAbsence err(17);"
         ZIP="num(5); onFalse err(18); onAbsence err(19);" />
   </DeliveryPlace>

   <Item xd:script="occurs 1..10; onAbsence err(21); onExcess err(22)"
      ProductCode="num(4); onFalse err(23); onAbsence err(24);"
      Quantity="int(1,1000); onFalse err(25); onAbsence err(26);" />
</Order>

<Errors>
   <Error xd:script="occurs +; create errors"
      Customer="?string"
```

```
      ErrorCode="int"
      Line="int"
      Column="?int" />
</Errors>

</xd:def>
```

V programu získáme z objektu XDDocument po spuštění validace vstupních dat hodnotu proměnné „errors“. Ta je typu Container a do ní byly ukládány informace o chybách. Pokud byly nalezeny chyby (Container je neprázdný), pomocí konstrukčního režimu vytvoříme XML dokument s chybami. Jinak jen uložíme výsledek zpracování.

Program „src/task1/Order2.java“ :

```
package task1;

import cz.syntea.xdef.reporter.ArrayReporter;
import cz.syntea.xdef.xml.KXmlUtils;
import cz.syntea.xdef.XDContainer;
import cz.syntea.xdef.XDDocument;
import cz.syntea.xdef.XDFactory;
import cz.syntea.xdef.XDPool;
import org.w3c.dom.Element;

public class Order2 {
        public static void main(String[] args) throws Exception {
                // Compile X-definitions to XDPool
                XDPool xpool = XDFactory.compileXD(null, "src/task1/Order2.xdef");

                // Create instance of XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Order");

                // Create reporter
                ArrayReporter reporter = new ArrayReporter();

                // Run validation mode (you can also try task1/input/Order_err.xml)
                Element result = xdoc.xparse("task1/input/Order.xml", reporter);

                // Get Container with errors from the variable "errors" in X-definition.
                XDContainer errors = (XDContainer) xdoc.getVariable("errors");

                // Check if an error was reported
                if (errors.getXDItemsNumber() > 0) { // errors reported
                        // Run the construction mode to create document with errors.
                        // The context is in the variable "errors"
                        result = xdoc.xcreate("Errors", null);
                        // write errors to the file
                        KXmlUtils.writeXml("task1/errors/Order_err.xml", result);
                        System.err.println("Incorrect input data");
                } else {
                        // No errors, write the processed document to the file
                        KXmlUtils.writeXml("task1/output/Order.xml", result);
                        System.out.println("OK");
                }
        }
}
```

4.1.5 Varianta 5: generování XML dokumentu a chyb v jednom kroku

V této variantě ve skriptu otestujeme, zda na konci zpracování (viz událost „finally“ popsaná ve skriptu kořenového elementu) došlo k nějaké chybě. Pokud ano, vyvoláme konstrukci dokumentu s chybami podle modelu „Errors“ metodou „xcreate“ přímo ve skriptu. Výsledek konstrukce bude uložen do proměnné „errs“. X-definici poněkud dále upravíme, abychom si ukázali možnosti Skriptu. Předně použijeme konstruktory pro hodnoty typu „Container“. Navíc zavedeme proměnnou „customer“ do které uložíme hodnotu atributu „CustommerCode“ v sekci „onStartElement“. Diky tomu, že kód zakazník jsme uložili na začátku zpracování objednávky nemusíme získat tuto hodnotu pomocí metody „xpath“. Připomeňme si, že příkazy této sekce se provedou až po zpracování všech atributů právě zpracovávaného elementu. Aby tento atribut nebyl prázdný v případě že chybí, zavoláme v sekci „onAbsence“ atributu „CustomerCode“ kromě ohlášení chyby i metodu „setText(‘-1’)“, pomocí níž je hodnota atribufu nastavena.

X-definice popisující dané požadavky (je umístěna v souboru „src/task1/Order2a.xdef“):

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1" name="Order" root="Order">

<xd:declaration>
```

```
  /* Here will be XML document with error messages. */
  Element errs = null;
  /* Container, to which we store the information about errors. */
  Container errors = [];
  /*Saved customer code from the input data - see "onStartElement". */
  String customer;

  void err(int code) {
    /* Create container. */
    Container c = [ %Customer = customer,
      %ErrorCode = code,
      %Line = getSourceLine(),
      %Column = getSourceColumn()];
          /* Add the item to the Container errors. */
          errors.addItem(c);
  }

  void createErrors() {
   /* Check if errors were generated. */
   if (errors() GT 0 || errors.getLength() GT 0) {
            /* Create element with errors to the variable "errs". */
           errs = xcreate("Errors");
          }
  }
 </xd:declaration>

<Order
   xd:script="onStartElement customer = (String) @CustomerCode; finally createErrors();"
   Number="int; onFalse err(1); onAbsence err(2);"
   CustomerCode="string; onAbsence {err(3); setText('error');}">

   <DeliveryPlace xd:script="onAbsence err(11);">
      <Address Street="string(2,100); onFalse err(12); onAbsence err(13);"
         House="int(1,9999); onFalse err(14); onAbsence err(15);"
         City="string(2,100); onFalse err(16); onAbsence err(17);"
         ZIP="num(5); onFalse err(18); onAbsence err(19);"/>
   </DeliveryPlace>

   <Item xd:script="occurs 1..10; onAbsence err(21); onExcess err(22)"
      ProductCode="num(4); onFalse err(23); onAbsence err(24)"
      Quantity="int(1,1000); onFalse err(25); onAbsence err(26)"/>
</Order>

<Errors>
   <Error xd:script="occurs +; create errors"
      ErrorCode="int"
      Customer="string"
      Line="int"
      Column="int"/>
</Errors>

</xd:def>
```

Proměnnou „errs“ převezmeme a ostestujeme programem Java podobným způsobem, jako v předešlé variantě. Pokud byly nalezeny chyby, převezmeme vytvořený element z proměnné „errs“ a zapíšeme je.

Program „src/task1/Order2a.java“, pro zpracování vstupních dat a obhospodaření chyb :

```
package task1;

import cz.syntea.xdef.reporter.ArrayReporter;
import cz.syntea.xdef.xml.KXmlUtils;
import cz.syntea.xdef.XDDocument;
import cz.syntea.xdef.XDFactory;
import cz.syntea.xdef.XDPool;
import cz.syntea.xdef.XDValue;
import org.w3c.dom.Element;

public class Order2a {
        public static void main(String[] args) throws Exception {
                // Compile X-definitions to XDPool
                XDPool xpool = XDFactory.compileXD(null, "src/task1/Order2a.xdef");

                // Create instance of XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Order");

                // Create reporter
                ArrayReporter reporter = new ArrayReporter();

                // Run validation mode (you can also try task1/input/Order_err.xml)
```

```
                Element result = xdoc.xparse("task1/input/Order.xml", reporter);

                // Get variable "errs".
                XDValue errs = xdoc.getVariable("errs");

                // Check if an error was reported
                if (errs != null && !errs.isNull()) {
                        // Write errors to the file
                        KXmlUtils.writeXml("task1/errors/Order_err.xml", errs.getElement());
                        System.err.println("Incorrect input data");
                } else {
                        // No errors, write parsed document to the file
                        KXmlUtils.writeXml("task1/output/Order.xml", result);
                        System.out.println("OK");
                }
        }
}
```

4.1.6 Varianta 6: vytvoření chybového souboru pomocí externí metody

Zkusme nyní upravit náš příklad tak, že budeme chyby obhospodařovat uživatelskými externími metodami v jazyce Java a nikoliv metodami ve skriptu. Tyto externí metody budou volány opět ze skriptu, ale výkonný kód bude v jazyce Java. Protože dokument s chybami je generován externí metodou, není třeba v X-definici uvádět model elementu s chybami, chybová data budou tvořena přímo v příslušné metodě. Cena, kterou za tento postup zaplatíme je, že struktura chybového dokumentu není popsána v X-definici a je závislá na implementaci externí metody v jazyce Java (a tudíž není X-definicí kontrolována).

Všimněte si, že jsme do X-definice přidali deklaraci externích metod použitých ve Skriptu. Metody, které jsou volány ze skriptu musí být statické a jejich parametry musí odpovídat typům, které jsou uvedeny v popisu. Popis vlastností externích Java metod je uveden v příloze spolu s tabulkou povolených typů.

Vzhledem k tomu, že externí metody musí být statické, je třeba umožnit v externích metodách pracovat s instancemi objektů, které externí metody využívají pomocí metody „getUserObject“. Tento objekt musí být jediný a proto je třeba do něj uložit před spuštěním zpracování potřebná data. Objekt se připojí k objektu XDDocument metodou „setUserObject“ (v naší ukázce slouží k vytvoření XML dokumentu pro ukládání chyb).

Externí Java metody volané ze skriptu musí mít určité vlastnosti. Předně musí být deklarovány jako statické a „public“.

První parametr externí metody deklarované v Java může být typu XXNode a tento parametr předá procesoru X-definic automaticky (nenií deklarován ve volání metody ve skriptu, ale je deklarován v externí metodě). V našem příkladu jej potřebujeme k získání potřebných údajů o stavu zpracování, např o aktuální pozici zpracovávaného vstupního zdrojového tvaru ale také k získání připojeného uživatelského objektu metodou „getUserObject“. Tento objekt musí být uložien do objektu XDDocument před spuštěním X-definice Java metodou „setUserObject“ (v naší ukázce slouží k vytvoření XML dokumentu pro ukládání chyb).

Typy dalších parametrů odpovídají typům parametrů specifikovaných ve volání externí metody ze skriptu.Druhý parametr v našem příkladě odpovídá číslu kódu chyby. Všimněme si, že v Java programu musí být deklarován jako „long“ (viz tabulka v příloze), ale ve skriptu jsou všechna celá čisla typu „int“. Naše externí metoda z těchto údajů vytvoří element s informacemi o chybách a připojí jej ke kořenovému elementu pro chyby.

Program v jazyce Java (soubor „src/task1/Order3.java“):

```
package task1;

import cz.syntea.xdef.reporter.ArrayReporter;
import cz.syntea.xdef.sys.SPosition;
import cz.syntea.xdef.xml.KXmlUtils;
import cz.syntea.xdef.XDDocument;
import cz.syntea.xdef.XDFactory;
import cz.syntea.xdef.XDPool;
import cz.syntea.xdef.proc.XXNode;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class Order3 {

        public static void main(String[] args) throws Exception {
                // Create instance of XDDocument object (from XDPool)
                // (external method "err" called from the Script see below)
                XDPool xpool = XDFactory.compileXD(null, "src/task1/Order3.xdef");
```

```
                // Create instance of XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Order");

                // Create reporter
                ArrayReporter reporter = new ArrayReporter();

                // Prepare XML element for recording of errors
                Element errors =
                        KXmlUtils.newDocument(null, "Errors", null).getDocumentElement();
                xdoc.setUserObject(errors);

                // Run validation mode (you can also try task1/input/Order_err.xml)
                xdoc.xparse("task1/input/Order.xml", reporter);

                // Check errors
                if (errors.getChildNodes().getLength() > 0) {
                        // Write error information to the file
                        KXmlUtils.writeXml("task1/errors/Order_err.xml", errors);
                        System.err.println("Incorrect input data");
                } else {
                        // No errors, write the processed document to the file
                        KXmlUtils.writeXml("task1/output/Order.xml", xdoc.getElement());
                        System.out.println("OK");
                }
        }

        // External method called from the Script of X-definition
        public static void err(XXNode xnode, long code) {
                Document doc = ((Element) xnode.getUserObject()).getOwnerDocument();
                Element newElem = doc.createElement("Error");
                newElem.setAttribute("ErrorCode", String.valueOf(code));
                Element root = xnode.getElement().getOwnerDocument().getDocumentElement();
                newElem.setAttribute("Customer", root.getAttribute("CustomerCode"));
                SPosition pos = xnode.getSPosition();
                newElem.setAttribute("Line", String.valueOf(pos.getLineNumber()));
                newElem.setAttribute("Column", String.valueOf(pos.getColumnNumber()));
                doc.getDocumentElement().appendChild(newElem);
        }
}
```

X-definice (soubor „src/task1/Order3.xdef“):

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1" name="Order" root="Order">

<xd:declaration>
   external method void task1.Order3a_ext.err (XXNode, XDValue[]);
</xd:declaration>

<Order
   Cislo="int; onFalse err(1); onAbsence err(2);"
   CustomerCode="string; onAbsence err(3);">

   <DeliveryPlace xd:script="onAbsence err(11);">
      <Address Street="string(2,100); onFalse err(12); onAbsence err(13);"
         House="int(1,9999); onFalse err(14); onAbsence err(15);"
         City="string(2,100); onFalse err(16); onAbsence err(17);"
         ZIP="num(5); onFalse err(18); onAbsence err(19);"/>
   </DeliveryPlace>

   <Product xd:script="occurs 1..10; onAbsence err(21); onExcess err(22)"
      CommodityCode="num(4); onFalse err(23); onAbsence err(24)"
      Quantity="int(1,1000); onFalse err(25); onAbsence err(26)"/>

</Order>

</xd:def>
```

Java třída s externí metodou "err" (soubor „src/task1/Order3_ext.java“):

```
package task1;

import cz.syntea.xdef.sys.SPosition;
import cz.syntea.xdef.XDValue;
import cz.syntea.xdef.proc.XXNode;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class Order3_ext {
```

```
        /** Add error item. */
        public static void err(XXNode xnode, XDValue[] params) {
                Document doc = ((Element) xnode.getUserObject()).getOwnerDocument();
                Element newElem = doc.createElement("Error");
                newElem.setAttribute("ErrorCode", params[0].toString());
                Element root = xnode.getElement().getOwnerDocument().getDocumentElement();
                newElem.setAttribute("Customer", root.getAttribute("CustomerCode"));
                SPosition pos = xnode.getSPosition();
                newElem.setAttribute("Line", String.valueOf(pos.getLineNumber()));
                newElem.setAttribute("Column", String.valueOf(pos.getColumnNumber()));
                doc.getDocumentElement().appendChild(newElem);
                xnode.clearTemporaryReporter(); // remove the error
        }
}
```

4.2 Úloha 2

Tato úloha je modifikací předešlé, ale zpracováváme v něm ne jednu objednávku, nýbrž soubor více objednávek. Jejich počet není omezen a dokonce se nemusí vejít do paměti počítače.

Vstupní soubor s objednávkami „task2/input/Orders..xml“:

```
<Orders id = "123456789">

<Order Number="123" CustomerCode="ALFA">
   <DeliveryPlace>
      <Address Street="LIBERECKA" House="5" City="LIBEREC" ZIP="32321"/>
   </DeliveryPlace>
   <Item ProductCode="0002" Quantity="2"/>
   <Item ProductCode="0003" Quantity="1"/>
</Order>

<Order Number="456" CustomerCode="BETA">
   <DeliveryPlace>
      <Address Street="NA SLUPI" House="9" City="PRAHA" ZIP="11000"/>
   </DeliveryPlace>
   <Item ProductCode="0019" Quantity="50"/>
</Order>

</Orders>
```

4.2.1 Varaianta 1: použití příkazu „forget“

Při řešení této úlohy využijeme příkaz skriptu „forget“. Tento příkaz může být uveden u elementu a znamená, že po zpracování elementu X-definicí jej uvolní z paměti. V naší úloze však musíme zajistit, aby element před uvolněním byl zapsán do výstupního souboru. Procesor X-definic umí takovou situaci řešit – k objektu XDDocument je možné připojit datový kanál, do kterého se zapisují všechny zpracované objekty (ještě předtím, než se uvolní z paměti) pomocí metody „setStreamWriter“. V první variantě opět ponecháme hlášení chyb na aparátu X-definic.

X-definice popisující vstupní data (“src/task2/Orders1.xdef”):

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1" name="Orders" root="Orders">

<Orders id="num(9)">
   <Order xd:script="occurs +; forget"
      Number="int"
      CustomerCode="string(1,20)">
      <DeliveryPlace>
         <Address Street="string(2,100)" House="int(1,9999)" City="string(2,100)" ZIP="num(5)"/>
      </DeliveryPlace>
      <Item xd:script="occurs 1..10" ProductCode="num(4)" Quantity="int(1,1000)"/>
   </Order>
</Orders>

</xd:def>
```

Průběžný zápis zpracovávaného vstupního dokumentu do výstupního souboru zajistíme nastavením pomocí procedury „setStreamWriter“ pro objekt XDCodument. Procesor X-definic v tomto případě průběžně zapisuje zpracovávané elementy na výstupní kanál. Příkazem „forget“ v X-definici sdělíme procesoru, že na konci zpracování elementu má tento element uvolnit z paměti. Protokol o chybách tentokrát zapisujeme přímo do

souboru (použijeme proto třídu FileReportWriter namísto třídy ArrayReporter kterou jsme použili v předchozích příkladech).

Program „src/task2/Orders1.java“:

```
package task2;

import cz.syntea.xdef.sys.FileReportWriter;
import cz.syntea.xdef.XDDocument;
import cz.syntea.xdef.XDFactory;
import cz.syntea.xdef.XDPool;
import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStream;

public class Orders1 {
        public static void main(String[] args) throws Exception {
                // Compile X-definitions to XDPool
                XDPool xpool = XDFactory.compileXD(null, "src/task2/Orders1.xdef");

                // Create the instance of XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Orders");

                // The file where to rite result
                File vystup = new File("task2/output/Orders.xml");
                OutputStream out = new FileOutputStream(vystup);
                xdoc.setStreamWriter(out, "UTF-8", true);

                // The file with errors
                File chyby = new File("task2/errors/Orders_err.txt");

                // Prepasre the error reporter (write error directly to the file)
                FileReportWriter reporter = new FileReportWriter(chyby);

                // Run validation mode (you can also try task2/input/Order_err.xml)
                xdoc.xparse("task2/input/Orders.xml", reporter);

                // close the output stream.
                out.close();
                reporter.close();

                // Check if errors reported
                if (reporter.errorWarnings()) {
                        System.err.println("Incorrect input data");
                } else {
                        System.out.println("OK");
                }
        }
}
```

4.2.2 Varianta 2: zápis správných objednávek a chybných objednávek do souborů

V této variantě si ukážeme možnost rozdělit výstup do dvou souborů, do jednoho budeme zapisovat pouze správné objednávky a do druhého zapíšeme ty chybné. Ukážeme si zároveň složitější použití externích metod a připojeného uživatelského objektu.

V X-definici přibylo zavolani metody „writeOrder“ ve skriptu pro model elementu „Order“ v sekci „finally“. Namísto nastavení automatického zápisu zpracovávaného vstupního dokumentu do výstupního souboru budeme provádět zápis zpracovaných elementů v pomocí metody „writeOrder“. Chyby budeme zapisovat metodou „err“. Nakonec musíme ukončit zápis metodou „closeAll“, která ukončí zápis do výstupních souborů pokud je příslušný soubor neprázdný.

V deklační sekci nadeklarujeme tři čítače, do kterách dáme počáteční hodnotu „0“ a do kterých budem ukládat počet chyb a počet správných objednávek. Čítač „count“ počítá správné objednávky, čítač „errCount“ počítá chyby a do čítače „errCountOld“ je uložen počet chyb po zpracování chybné objednávky, abychom mohli zjistit, zda ve zpracované objednávce byly nalezeny chyby (abychom je nezapsali do výstupního souboru).

Výstupní a chybový soubor zapisujeme pomocí objektů „XmlOutStream“, které mají implementovány prostředky pro průběžný zápis XML objektů. Element lze zapsat buď jako celek metodou „writeElement“. Druhá možnost je rozdělit zápis na zápis záhlaví elementu metodou „writeElementStart“, dále zapisovat potomky elementu metodami „writeElement“ a případně „writeText“ a nakonec ukončit zápis elementu, jehož zápis jsme zahájili

metodou „writeElementEnd“. Vlastností objektu „XmlOutStream“ je, že se výstupní soubor vytvoří teprve při prvním zápisu.

Abychom mohli zápis provést, musíme tedy mít před prvním zápisem elementy pro zápis root elementu metodou „writeElementStart“ a abychom mohli zápis správně ukončit metodou „writeElementEnd“. Jednotlivé objednávky resp. chybové záznamy zapisujeme do příslušných datových kanálů metodou „writeElement“.

Všimněte si, že jsme deklarační část zapsali do CDATA sekce, abychom mohli volně používat znaky „<“, „&“ a abychom nemuseli používat alternativní zápis operátorů.

Jména souborů jsou procesoru X-definice předána programem pomocí externích proměnných „outFile“ a „errFile“ typu String.

X-definice popisující vstupní data (“src/task2/Orders2.xdef”):

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1" name="Orders" root="Orders">

<xd:declaration>
<![CDATA[
  /* File names of files with constomers and commodity oodes. */
  external String outFile, errFile;
  int errCount = 0, errCountOld = 0, count = 0;
  XmlOutStream outStream, errStream;

  /* write file with errors */
  void err(int code) {
    if (errCount++ == 0) {
      errStream = new XmlOutStream(errFile);
           /* Document header and start of the root element */
      errStream.writeElementStart(new Element("Errors"));
    }
    Element e = new Element('Error'); /* Pripravime element. */
    /* Set attributes. */
    e.setAttr("Customer", xpath("/Orders/Order/@CustomerCode").toString());
    e.setAttr("ErrorCode", toString(code));
    e.setAttr("Line", toString(getSourceLine()));
    e.setAttr("Column", toString(getSourceColumn()));
    errStream.writeElement(e); /* Write the element with error information */
  }

  /* Write result */
  void writeResult() {
    if (errCount != errCountOld) { /* Error in order */
      errCountOld = errCount; /* Save number of errors, to know in new error occurred */
      return; /* write nothing */
    }
    if (count++ == 0) {
      /* this is the first item*/
      outStream = new XmlOutStream(outFile, "windows-1250"); /* Prepare data stream */
      outStream.writeElementStart(getRootElement()); /* Write document header and root element */
    }
    outStream.writeElement(getElement()); /* write the processed order */
  }

  /* Close the output stream and the file with errors */
  void closeAll() {
    if (errCount > 0) { /* check if errors were reported */
      errStream.writeElementEnd(); /* write root element end tag */
      errStream.close(); /* close the data stream */
    }
    if (count > 0) { /* Check if a correct order exists */
      outStream.writeElementEnd(); /* write root element end tag */
      outStream.close(); /* close the data stream */
    }
  }
]]>
</xd:declaration>

<Orders id="num(9); onFalse err(98)"
   xd:script="finally closeAll();">
   <Order xd:script="occurs +; onAbsence err(99); finally writeResult();"
      Number="int; onFalse err(1); onAbsence err(2);"
      CustomerCode="string; onAbsence err(3);">
      <DeliveryPlace xd:script="onAbsence err(11);">
         <Address
            Street="string(2,100); onFalse err(12); onAbsence err(13);"
            House="int(1,9999); onFalse err(14); onAbsence err(15);"
            City="string(2,100); onFalse err(16); onAbsence err(17);"
```

```
            ZIP="num(5); onFalse err(18); onAbsence err(19);"/>
      </DeliveryPlace>
      <Item xd:script="occurs 1..10; onAbsence err(21); onExcess err(22);"
         ProductCode="num(4); onFalse err(23); onAbsence err(24);"
         Quantity="int(1,1000); onFalse err(25); onAbsence err(26);"/>
   </Order>
</Orders>

</xd:def>
```

JavaProgram „src/task2/Orders2.java“:

```
package task2;

import cz.syntea.xdef.reporter.ArrayReporter;
import cz.syntea.xdef.XDDocument;
import cz.syntea.xdef.XDFactory;
import cz.syntea.xdef.XDPool;

public class Orders2 {
        public static void main(String[] args) throws Exception {
                // Compile X-definition to XDPool
                XDPool xpool = XDFactory.compileXD(null, "src/task2/Orders2.xdef");

                // Create the instance of XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Orders");

                // Set external variables
                xdoc.setVariable("outFile", "task2/output/Orders.xml");
                xdoc.setVariable("errFile", "task2/errors/Orders_err.xml");

                // Prepare reporter
                ArrayReporter reporter = new ArrayReporter();

                // Run validation mode (you can also try task2/input/Order_err.xml)
                xdoc.xparse("task2/input/Orders.xml", reporter);

                // Throw an exception if unexpected errors detected
                reporter.checkAndThrowErrors();

                // checck reported errors
                if (xdoc.getVariable("errCount").intValue() != 0) {
                        System.err.println("Incorrect input data");
                } else {
                        System.out.println("OK");
                }
        }
}
```

4.2.3 Varianta 3: obdoba předchozí varianty s externími metodami

V této variantě si ukážeme opět možnost rozdělit výstup do dvou souborů, do jednoho budeme zapisovat pouze správné objednávky a do druhého zapíšeme ty chybné. Ukážeme si zároveň složitější použití externích metod a připojeného uživatelského objektu, ve kterém jsme naprogramovali metody z předešlé ukázky pomocí externích metod v Javě.

X-definice opět provádí zápis výstupního objektu a hlásí chyby pomocí metod, které jsou naprogramovány v externí třídě.

X-definice popisující vstupní data (“src/task2/Orders3.xdef”):

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1" name="Orders" root="Orders">

<xd:declaration>
   external method void task2.Orders3ext.writeOrder(XXNode);
   external method void task2.Orders3ext.err(XXNode, long);
</xd:declaration>

<Orders id="num(9); onFalse err(0)">
   <Order xd:script="occurs +; onAbsence err(0); finally writeOrder(); forget"
      Number="int; onFalse err(1); onAbsence err(2);"
      CustomerCode="string; onAbsence err(3);">
      <DeliveryPlace xd:script="onAbsence err(11);">
         <Address
            Street="string(2,100); onFalse err(12); onAbsence err(13);"
            House="int(1,9999); onFalse err(14); onAbsence err(15);"
            City="string(2,100); onFalse err(16); onAbsence err(17);"
```

```
            ZIP="num(5); onFalse err(18); onAbsence err(19);"/>
      </DeliveryPlace>
      <Item xd:script="occurs 1..10; onAbsence err(21); onExcess err(22);"
         ProductCode="num(4); onFalse err(23); onAbsence err(24);"
         Quantity="int(1,1000); onFalse err(25); onAbsence err(26);"/>
   </Order>
</Orders>

</xd:def>
```

Namísto nastavení průběžného zápisu zpracovávaného vstupního dokumentu do výstupního souboru budeme provádět zápis zpracovaných elementů v pomocí metody „writeOrder“ v externí třídě „Oorders2ext“. Instanci externí třídy uložíme do proměnné „writer“ a připojíme ji pomocí metody „setUserObject“ k objektu XDDocument. Nakonec ukončíme zápis metodou „closeAll“, která ukončí zápis do výstupního souboru i do souboru s chybami. Jako reporter připojíme instanci třídy ArrayReporter – chyby by měly být hlášeny pomocí metod deklarovaných ve skriptu. Kdyby se přesto vyskytly, bude program ukoncen vyjimkou (viz příkay „reporter.checkAndThrowErrors()“). Dále uzavřeme soubory (příkazem „writer.closeAll();“), které jsme vytvořili a nakonec získáme z uživatelského objektu počet zapsaných chyb (příkazem „writer.errNum()“).

Program „src/task2/Orders3.java“:

```
package task2;

import cz.syntea.xdef.XDDocument;
import cz.syntea.xdef.XDFactory;
import cz.syntea.xdef.XDPool;

public class Orders3 {
        public static void main(String[] args) throws Exception {
                // Compile X-definition to XDPool
                XDPool xpool = XDFactory.compileXD(null, "src/task2/Orders3.xdef");

                // Create the instance of XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Orders");

                // create instasnce of Orders3ext
                Orders3ext writer = new Orders3ext("task2/output/Orders.xml",
                        "task2/errors/Orders_err.xml");
                xdoc.setUserObject(writer);

                // Run validation mode (you can also try task2/input/Order_err.xml)
                xdoc.xparse("task2/input/Orders.xml", null);

                // close all streams
                writer.closeAll();

                if (writer.errNum() != 0) {
                        System.err.println("Incorrect input data");
                } else {
                        System.out.println("OK");
                }
        }
}
```

Externí třída pro průběžný zápis dat zajišťuje zápis bezchybé objednávky do zapisovače v proměnné „_outputWriter“ a chybových informací v proměnné „_errorWriter“. Konstruktor zapisovače se vytvoří až při zápisu první objednávky nebo chybové informace. Při založení zapisovače je nutné zapsat záhlaví XML souboru a začátek kořenového elementu (jmeno elementu a jeho atributy) pomocí metody „writeElementStart“. Jednotlivé vnitřní elementy zapisujeme metodou „writeNode“. Na konci musíme ještě zapsat ukončení kořenového elementu metodou „writeElementEnd“ (viz metodu „closeAll()“ v přiloženém Java zdrojovém programu.

Java třída s externími metodami („src/task2/Orders3ext.java“):

```
package task2;

import cz.syntea.xdef.sys.SPosition;
import cz.syntea.xdef.xml.KXmlUtils;
import cz.syntea.xdef.xml.KXpathExpr;
import  cz.syntea.xdef.XDFactory;
import  cz.syntea.xdef.XDXmlOutStream;
import  cz.syntea.xdef.proc.XXNode;
import java.io.IOException;
import javax.xml.xpath.XPathConstants;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
```

```
public class Orders3ext {

	private final String _outputFile;		// Output file
	private final String _errorFile;		// Error file
	private XDXmlOutStream _outputWriter;	// writer for output
	private XDXmlOutStream _errorWriter;	// writer for errors
	private Document _errorDoc;		// XML document with errors
	private Document _objDoc;		// XML document with orders
	private final KXpathExpr _xpath;	// prepared xpath expression
	private int _errCount;			// error counter
	private int _errCountOld;		// previous value of the error counter
	private int _count;			// counter of correct orders

	// Create instance of this class.
	public Orders3ext(String outputFile, String errorFile) throws IOException {
		_outputFile = outputFile; // output file name.
		_errorFile = errorFile; // error file name.
		// writers will be created when an item to bew written occurs
		_errorWriter = _outputWriter = null;
		// Prepare XPath expression to get customer code from an order
		// Because of the command "forget" it will be in the processed document
		// only one (the actually processed) order.
		// This XPath expression will be executed when an error item is generated.
		_xpath = new KXpathExpr("/Orders/Order[1]/@KodZakaznika");
		// Clear counters
		_errCount = _errCountOld = _count = 0;
	}

	// Write the order (only if no error was reported
	public static void writeOrder(XXNode xnode) {
		// Get "User object" (i.e. the instance of this class).
		Orders3ext x = (Orders3ext) xnode.getUserObject();
		if (x._errCount != x._errCountOld) { // an error was reported?
			// set old error counter (i.e. no errors reported for next item)
			x._errCountOld = x._errCount;
		} else {
			// No error reported, so writ the order the result.
			Element el = xnode.getElement();
			if (x._count == 0) { // check if nothig was written yeat
				// Create writer a write the XML header and the root element
				try {
					x._outputWriter =
						XDFactory.createXDXmlOutStream(x._outputFile,
						"windows-1250", true);
				} catch (Exception ex) {
					throw new RuntimeException(ex.getMessage());
				}
				x._objDoc = el.getOwnerDocument();
				x._outputWriter.writeElementStart(x._objDoc.getDocumentElement());
			}
			x._count++; // increase the counter of correct orders.
			// write the processed order
			x._outputWriter.writeNode(el);
		}
	}

	// Create the writeOrder and set the variable "_error".
	public static void err(XXNode xnode, long code) {
		// Get "User object" (i.e. the instance of this class).
		Orders3ext x = (Orders3ext) xnode.getUserObject();
		// Create the XML writer for errors (if it was not created yeat)
		if (x._errCount == 0) {
			try {
				x._errorWriter = XDFactory.createXDXmlOutStream(x._errorFile,
					"windows-1250",true);
			} catch (Exception ex) {
				throw new RuntimeException(ex.getMessage());
			}
			x._errorDoc = KXmlUtils.newDocument(null,"Errors",null);
			// write XML header and the root element.
			x._errorWriter.writeElementStart(x._errorDoc.getDocumentElement());
		}
		x._errCount++; // increase error counter
		// Create the element to be written.
		Element el = x._errorDoc.createElement("Error");
		el.setAttribute("ErrorCode", String.valueOf(code));
		String customer =
			(String) x._xpath.evaluate(xnode.getElement(), XPathConstants.STRING);
		el.setAttribute("Customer", customer);
		SPosition pos = xnode.getSPosition();
```

```
                el.setAttribute("Line", String.valueOf(pos.getLineNumber()));
                el.setAttribute("Column", String.valueOf(pos.getColumnNumber()));
                x._errorWriter.writeNode(el);
        }

        // Get number of errors
        public int errNum() {return _errCount;}

        // Close created files
        public void closeAll() {
                // close result output stream (if something was written)
                if (_outputWriter != null) {
                        _outputWriter.closeStream(); // write root end tag and close the stream
                }
                // close error output stream (if something was written)
                if (_errorWriter != null) {
                        _errorWriter.closeStream(); // write root end tag and close the stream
                }
        }
}
```

4.3 Úloha 3

Nyní zkontrolujme ve vstupních datech správnost kódu zákazníka a čísla zboží podle dat uložených v XML souborech. Vstupní data jsou stejná jako v prvním příkladě.

Vstupní soubor správné objednávky „task3/input/Order.xml“:

```
<Order Number="123" CustomerCode="ALFA">
   <DeliveryPlace>
      <Address Street="LIBERECKÁ" House="5" City="LIBEREC" ZIP="32321"/>
   </DeliveryPlace>
   <Item ProductCode="0002" Quantity="2"/>
   <Item ProductCode="0003" Quantity="1"/>
</Order>
```

Soubor s kódy zákazníků „task3/input/Customers..xml“:

```
<Customers>
   <Customer CustomerCode="ALFA">
      <Company Name="ALFA S.R.O." ID="1234578"/>
      <Address Street="KLADENSKÁ" House="5" City="KLADNO" ZIP="54321"/>
   </Customer>
   <Customer CustomerCode="BETA">
      <Company Name="BETA A.S" ID="1234580"/>
      <Address Street="MÌLNICKÁ" House="5" City="MÌLNÍK" ZIP="45321"/>
   </Customer>
</Customers>
```

Soubor s čísly zboží „task3/input/Products..xml“:

```
<Products>
   <Product Code="0001" Name="bicycle" Price="150.50"/>
   <Product Code="0002" Name="motorcycle" Price="2340.00"/>
   <Product Code="0003" Name="car" Price="7777.00"/>
</Products>
```

4.3.1 Varianta 1: automatická generace chybového kódu

V první variantě jako obvykle ukážeme nejjednodušší řešení, které generuje chybový kód automaticky. XML elementy s kódy zákazníků a s čísly zboží předáme pomocí externích proměnných. Kód zákazníka a kód produktu ve Skriptu získáme pomocí metody „xpath“. Uvědomme si, že výsledkem této metody je Container. Nad objekty typu Container je ve skriptu implementována metoda „getLength“, která vrátí počet sekvenčních prvků v tomto objektu. Očekáváme, že je-li hodnota správná, je pomocí xpath nalezen příslušný prvek v příslušném souboru. Hodnotu, kterou pomocí xpath výrazu hledáme, získáme pomocí metody „getText“, která vrátí string s hodnotou právě zpracovávaného atributu, nebo textového uzlu. Typ hodnoty zkontrolujeme výrazem typu boolean. Je-li jeho výsledek „true“, je hodnota správná, je-li „false“, je chybná.

Soubor s X-definicí „src/task3/Order1.xdef“:

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1" name="Order" root="Order">

<xd:declaration>
```

```
   /* Information about the customers and products. */
   external Element customers, products;
</xd:declaration>

<Order Number="int"
   CustomerCode="xpath('Customer[@CustomerCode=&quot;' + getText() + '&quot;]', customers).getLength()!=0">

   <DeliveryPlace>
      <Address
         Street="string(2,100)"
         House="int(1,9999)"
         City="string(2,100)"
         ZIP="num(5)"/>
   </DeliveryPlace>

   <Item xd:script="occurs 1..10"
      ProductCode="xpath('Product[@Code=&quot;' + getText() + '&quot;]', products).getLength() != 0"
      Quantity="int(1,1000)"/>
</Order>

</xd:def>
```

Program v jazyce Java je prakticky stejný jako v prvním příkladě, navíc však musíme do objektu XDDocument nastavit externí proměnné „items“ a „customers“. Tyto proměnné jsou typu Element. Metoda „setVariable“ v Java kódu automaticky provede konverzi souboru do typu Element.

Java program „src/task3/Order1.java“:

```
package task3;

import cz.syntea.xdef.reporter.ArrayReporter;
import cz.syntea.xdef.xml.KXmlUtils;
import cz.syntea.xdef.XDDocument;
import cz.syntea.xdef.XDFactory;
import cz.syntea.xdef.XDPool;
import java.io.PrintStream;

public class Order1 {
        public static void main(String[] args) throws Exception {
                // Compile X-definitions to XDPool
                XDPool xpool = XDFactory.compileXD(null, "src/task3/Order1.xdef");

                // Create the instance of XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Order");

                // set variables "products" and "customers"
                xdoc.setVariable("products", "task3/input/Ptoducts.xml");
                xdoc.setVariable("customers", "task3/input/Customers.xml");

                // Prepare error reporter
                ArrayReporter reporter = new ArrayReporter();

                // Run validation mode (you can also try task3/input/Order_err.xml)
                xdoc.xparse("task3/input/Order.xml", reporter);

                // Check if an error was reported
                if (reporter.errorWarnings()) {
                        // Print errors to the file
                        PrintStream ps = new PrintStream("task3/errors/Order_err.txt");
                        reporter.printReports(ps);
                        ps.close();
                        System.err.println("Incorrect input data");
                } else {
                        // write porocessed document
                        KXmlUtils.writeXml("task3/output/Order_123.xml", xdoc.getElement());
                        System.out.println("OK");
                }
        }
}
```

Výsledek programu je podobný jako v prvním příkladě.

4.3.2 Varianta 2: ukázka deklarace validačních metod

V této variantě provedeme změnu malou změnu: kontrolu typu provedeme pomocí deklarace vlastních typů. Deklarovaný typ je vlastně metoda, jejímž výsledkem je boolean hodnota. Ukážeme si, že chybové hlášení

můžeme provést pomocí metodÿ „error“, která zapíše chybové hlášení do pracovního protokolu a vrátí hodnotu „false“. Program Java bude stejný jako v předešlé variantě, uveďme jen X-definici:

Soubor s X-definicí „src/task3/Order2.xdef“:

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1" name="Order" root="Order">

<xd:declaration>
  /* Information about the customers and products. */
  external Element customers, items;

  boolean customer() {
    String s = getText();
    Container c = xpath('Customer[@CustomerCode="' + s + '"]', customers);
    return c.getLength()==0 ? error("Incorrect customer code: " + s) : true;
  }

  boolean item() {
    String s = getText();
    Container c = xpath('Product[@Code="' + s + '"]', products);
    return c.getLength()==0 ? error("Incorrect item code: " + s) : true;
  }
</xd:declaration>

<Order Number="int" CustomerCode= "customer()">

  <DeliveryPlace>
    <Address Street="string(2,100)"
      House="int(1,9999)"
      City="string(2,100)"
      ZIP="num(5)"/>
  </DeliveryPlace>

  <Item xd:script="occurs 1..10"
    ProductCode="item()"
    Quantity="int(1,1000)"/>
</Order>

</xd:def>
```

V případě chyby bude v protokolu chybového hlášení informace s textem uvedeným v metodě „error“:

Výstupní soubor chybné objednávky „task3/chyby/Order_err.txt“:

```
Incorrect customer code: AGFA; line=2; column=35;
source='file:/D:/cvs/DEV/java/examples/task3/input/Order_err.xml'; pos=82; xpath=/Order/@CustomerCode;
X-position=Order#Order
E XDEF515: Value error; line=2; column=35; source="file:/D:/cvs/DEV/java/examples/task3/input/Order_err.xml";
xpath=/Order/@CustomerCode; X-position=Order#Order/@CustomerCode
```

4.3.3 Varianta 3: použití souboru s texty chybových hlášení v jazykových mutacích

Tato varianta je shodná s předchozí, ale ukážeme možnost připojit texty chybových hlášení. Výhodou je, že tyto texty a jejich jazykové mutace můžeme provádět odděleně od programu. Metoda skriptu „error“ používá reportér procesoru X-definic a modely reportů je možné k programu připojit jako soubor s properies (viz program Java). V metodě „error“ musíme v tomto případě použít jako první parametr identifikátor chyby, jako druhý parametr text hlášení a jako třetí parametr modifikační text (pokud modifikaci požadujeme). Je-li nalezen identifiátor zprávy, je obsah druhého parametru nahrazen textem z nalezeného report, jinak je pro sestavení hlášení použije tento parametr. Pokud víme. že príslušná zpráva existuje, může být druhý parametr prázný string nebo null.

Soubor s X-definicí „src/task3/Orders2a.xdef“:

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1" name="Order" root="Order">

<xd:declaration>
<![CDATA[
  /* Information about the customers and products. */
  external Element customers, items;

  /* Validation of customer code. */
  boolean customer() {
    String s = getText();/* get value of the attribute. */
    /* find customer description and save it to Container. */
    Container c = xpath('Customer[@CustomerCode="' + s + '"]', customers);
    /* Check if the customer was found. If yes, return true; otherwise
```

```
     * call the method error, which writes the error report and returns false. */
    return c.getLength()==0 ?
      error("POBJ001", "Customer code: &{0}", "&{0}" + s) : true;
  }

  /* Validate product code from Item. */
  boolean item() {
    String s = getText(); /* get attribute value. */
    /* Find product description and save it to Container. */
    Container c = xpath('Product[@Code="' + s + '"]', products);
    /* Check if the code was found. If yes, return true; otherwise
     * call the method error, which writes the error report and returns false. */
    return c.getLength()==0 ?
      error("POBJ002", "Product code: &{0}", "&{0}" + s) : true;
  }
]]>
</xd:declaration>

<Order Number="int" CustomerCode= "customer()">

  <DeliveryPlace>
    <Address Street="string(2,100)"
      House="int(1,9999)"
      City="string(2,100)"
      ZIP="num(5)"/>
  </DeliveryPlace>

  <Item xd:script="occurs 1..10"
    ProductCode="item()"
    Quantity="int(1,1000)"/>
</Order>

</xd:def>
```

Všimněte si, že v textech zpráv je přidán odkaz „&{#SYS000}“, který umožňuje vkládat řádky, sloupec, název zdrojového souboru a polohu XPath do aktuálně zpracovávaného objektu přes parametry „&{line}“, „&{column}“, „&{source}“ atd.). Položky property „_prefix“ a „_language“ jsou povinné a spefifikují prefix popsaných zpráv a ISO jméno popisu zpráv. Soubor musí být zapsán ve znakovém kódu UTF-8.

Tabulka reportů v čestině „src/task3/Order_ces.properties“:

```
# Prefix of messages.
_prefix=POBJ

# ISO name of language.
_language=ces

# ISO name of default language.
_defaultLanguage=eng

# *************** Messages: ***************
POBJ001=Chybný kód zákazníka: "&{0}"&{#SYS000}
POBJ002=Chybné číslo položky: "&{0}"&{#SYS000}
POBJ003=Chyba ve vstupních datech
POBJ004=Vstupní data jsou zapsána
```

Tabulka reportů ve slovenštině „src/task3/Order_slk.properties“:

```
# Prefix of messages.
_prefix=POBJ

# ISO name of language.
_language=slk

# ISO name of default language.
_defaultLanguage=eng

# ISO name of default language.
_defaultLanguage=eng

# *************** Messages: ***************
POBJ001=Chybný kód zákazníka: "&{0}"&{#SYS000}
POBJ002=Chybné čislo položky: "&{0}"&{#SYS000}
POBJ003=Chyba vo vstupných dátech
POBJ004=Vstpuná dáta zapísaná
```

Tabulka reportů v angličtině „src/task3/Order_eng.properties“:

```
# Prefix of messages.
```

```
_prefix=POBJ

# ISO name of language.
_language=eng

# ISO name of default language.
_defaultLanguage=eng

# ************** Messages: ***************
POBJ001=Incorrect customer code: "&{0}"&{#SYS000}
POBJ002=Invalid item identifier: "&{0}"&{#SYS000}
POBJ003=Input data error
POBJ004=Input data saved
```

V Java programu nejdříve nastavíme pomocí methody „setProperty“ přístup k souborům s jazykovými mutacemi zpráv. Můžeme také nastavit jazyk, ve kterém budou hlášení zobrazována (pokud jazyk nenastavíme, použije se systémové nastavení, není-li v tabulkách tento jazyk nalezen, použije se defaultní jazyk (tj. angličtina) a pokud ani takové hlášení není nalezeno, je použit text z druhého parametru metody „error“). Ukážeme si, že i v hlášeních v Java programu můžeme využít aparátu reportů (viz hlášení o chybách nebo o správném průběhu – metoda „Report.error“ a „Report.info“).

Program „src/task3/Order2a.java“ :

```
package task3;

import cz.syntea.xdef.reporter.ArrayReporter;
import cz.syntea.xdef.reporter.Report;
import cz.syntea.xdef.reporter.SManager;
import cz.syntea.xdef.xml.KXmlUtils;
import cz.syntea.xdef.XDDocument;
import cz.syntea.xdef.XDFactory;
import cz.syntea.xdef.XDPool;
import java.io.PrintStream;

public class Otder2a {
         public static void main(String[] args) throws Exception {
                  // Compile X-definitions to XDPool
                  XDPool xpool = XDFactory.compileXD(null, "src/task3/Order2a.xdef");

                  // Create the instance of XDDocument object (from XDPool)
                  XDDocument xdoc = xpool.createXDDocument("Order");

                  // Set files with reports
                  xdoc.setProperty(SManager.REPORTTABLE_FILE + "POBJ", "src/task3/*.properties");

                  // Set the actual language for reporter  (you can also try to set "ces")
                  xdoc.setProperty(SManager.REPORT_LANGUAGE, "eng"); // English

                  // Set external variables "products" and "customers"
                  xdoc.setVariable("products", "task3/input/Products.xml");
                  xdoc.setVariable("customers", "task3/input/Customers.xml");

                  // Prepare error reporter
                  ArrayReporter reporter = new ArrayReporter();

                  // Run the validation mode (you can also try task3/input/Order_err.xml)
                  xdoc.xparse("task3/input/Order.xml", reporter);

                  // Check errors
                  if (reporter.errorWarnings()) {
                           // print errors to the file
                           PrintStream ps = new PrintStream("task3/errors/Order_err.txt");
                           reporter.printReports(ps); //vytisteni chyb
                           ps.close();
                           // print the message to the system consloe
                           Report rep = Report.error("POBJ003", null);
                           System.err.println(rep.toString());
                  } else {
                           // Write processed document to the file
                           KXmlUtils.writeXml("task3/output/Order.xml", xdoc.getElement());
                           // print the message to the system consloe
                           Report rep = Report.info("POBJ004", null);
                           System.out.println(rep.toString());
                  }
         }
}
```

V případě chyby bude v protokolu chybového hlášení informace s textem uvedeným v metodě „error“ (jazyk je nastaven na češtinu):

Výstupní soubor chybné objednávky „task3/chyby/Order_err.txt“ v češtině:

```
E POBJ001: Incorrect customer code: 'AGFA'; line=2; column=35;
source="file:/D:/cvs/DEV/java/examples/task3/input/Order_err.xml"; xpath=/Order/@CustomerCode; X-
position=Order#Order
```

Řešení generování chybového souboru pomocí metod nechť čtenář navrhne sám jako cvičení.

4.4 Úloha 4

V této úloze provedeme opět validaci dat, ale doplníe některé informace do výsledného dokumentu. Opět budeme také pracovat se vstupním souborem, jejhož velikost je neomezená. K řešení přidáme kontrolu správnosti dat jako v předešlém přikladu, ale úlohu poněkud zkomplikujeme. Element s adresou zákazníka tentokrát není součástí vstupního souboru, ale my jej doplníme z dat, uložených v souboru s informacemi o zákaznících. Navíc do výstupního souboru doplníme informaci o položky, kterou vypočítáme jako násobek počtu položek a ceny za položku, kterou získáme z dat s informacemi o položce. V elementech s položkami tedy přibývá atribut „Price“. Do výstupního souboru a do souboru s chybami zapisujeme opět pouze správné objednávky. Pro jednoduchost budeme předpokládat formální správnost vstupních dat (všechny povinné údaje jsou ve správném formátu). Soubor s chybami pro jednoduchost ponecháme ve formátu reportéru jako string. Vstupní data tentokrát budou jednodušší (odpadá element „DeliveryPlace“).

Vstupní soubor s objednávkami „task4/input/Orders.xml“:

```
<Orders id="123456789">
  <Order Number="123" CustomerCode="ALFA">
    <Item ProductCode="0002" Quantity="2"/>
    <Item ProductCode="0003" Quantity="1"/>
  </Order>
  <Order Number="124" CustomerCode="BETA">
    <Item ProductCode="0001" Quantity="5"/>
  </Order>
</Orders>
```

4.4.1 Varianta 1: bez externích Java metod

V X-definici zajistíme doplnění chybějících prvků: element „DeliveryPlace“ a atributu „Price“ do objednávek. Popíšeme model vstupních dat „Order“ a vzhledem k tomu, že výstupní tvar objednávek je odlišný, popíšeme výstupní objednávku jako samostatný model, který použijeme pro konstrukci výsledku. XML data se seznamem validních zákazníků a položek předáme jako do příslušných externích proměnných „products“ a „customers“ programem. Programem také předáme výstupní kanál pro zápis výsledku do proměnné „output“.

Soubor s X-definicí „src/task4/Orders1.xdef“:

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1" name="Orders" root="Orders">

<xd:declaration>
  external Element products, /* information about products. */
         customers; /* information about customers. */
  external XmlOutStream output; /* Output stream (set by the external program). */

  Container c; /* information about customer created by the method "customer". */
  int errors = 0, errorsOld = 0, errorsOld1 = 0, count = 0; /* counters */

  /* Check customer code */
  boolean customer() {
    String s = getText(); /* get attribute value. */
    c = xpath('Customer[@CustomerCode="' + s + '"]', customers); /* Find the customer. */
        /* Check if the customer found */
    if (c.getLength()==0) {
      /* Customer not found, increase error counter and report an error. */
      errors++;
      return error("Incorrect customer code: " + s); /* sets error message and returns false. */
    }
    return true; /* Customer found, OK */
  }

  /* Check the item code */
  boolean item() {
```

```
      String s = getText(); /* get attribute value. */
      /* Find the description of the item. */
      Container c = xpath('Product[@Code="' + s + '"]', products);
      if (c.getLength()==0) { /* Item found? */
        /* Item not found, increase the error counter and report error. */
        errors++;
        return error("Incorrect item number: " + s);
      }
      return true; /* Item was found, OK */
  }

  /* Write order. */
  void writeObj() {
    if (errors != errorsOld || errorsOld1 != errors()) {
      /* An error occured */
      errorsOld = errors; /* save the counter to "errorsOld"; write nothing. */
      errorsOld1 = errors();
    } else {
        /* Is it the first record? */
        if (count++ == 0) {
                /* nothing was written yet, write XML header and root element. */
                output.setIndenting(true); /* set output indentation. */
                output.writeElementStart(getRootElement());
        }
        /* Create object from the context with the actual order and write it. */
        output.writeElement(xcreate('Order', getElement()));
    }
  }

  /* Close output. */
  void closeAll() {
      /* Something was written?. */
    if (count != 0) {
      /* yes, close output. */
      output.close();
    }
  }

  /* Calculate the total cost of the item of a order. */
  float price() {
    /* Get quantity */
    int number = parseInt(xpath("@Quantity")); /* Get quantity from the input data. */
        String code = from("@ProductCode"); /* Get commodity code from the input data. */
        /* Find the product (we already know that it exists - see customer()). */
        Element el1 = xpath('Product[@Code="' + code + '"]', products).getElement(0);
    float price = parseFloat(el1.getAttr("Price")); /* get price of one product. */
    return price * number; /* Compute total cost and return it. */
  }
  uniqueSet checkObjId int();
</xd:declaration>

<Orders xd:script="finally output.close();" id="string(9)">
  <Order xd:script="occurs +; finally writeObj(); forget"
      Number="checkObjId.ID()"
      CustomerCode="customer()">
    <Item xd:script="occurs 1..100;" ProductCode="item()" Quantity="int"/>
  </Order>
</Orders>

<Order Number="string" CustomerCode="string">
  <DeliveryPlace xd:script="create c" >
    <Address Street="string(2,100)"
      House="int(1,9999)"
      City="string(2,100)"
      ZIP="num(5)"/>
  </DeliveryPlace>
  <Item xd:script="occurs 1..100;"
    ProductCode="string();"
    Quantity="int();"
    Price="float; create price();" />
</Order>

</xd:def>
```

Program pro spuštění úlohy je podobný, jako v ostatních případech.

Program „src/task4/Orders1.java“:

```
package task4;
```

```
import cz.syntea.xdef.reporter.ArrayReporter;
import cz.syntea.xdef.XDDocument;
import cz.syntea.xdef.XDFactory;
import cz.syntea.xdef.XDPool;
import java.io.PrintStream;
import java.util.Properties;

public class Orders1 {
         public static void main(String[] args) throws Exception {
                  // compile XDPool from the X-definition
                  Properties props = new Properties();
                  XDPool xpool = XDFactory.compileXD(props, "src/task4/Orders1.xdef");

                  // Create the instance of XDDocument object (from XDPool)
                  XDDocument xdoc = xpool.createXDDocument("Orders");

                  // set variables "products", "customers" and "output"
                  xdoc.setVariable("products", "task4/input/Products.xml");
                  xdoc.setVariable("customers", "task4/input/Customers.xml");
                  xdoc.setVariable("output",
                           XDFactory.createXDXmlOutStream("task4/output/Orders.xml", "UTF-8", true));

                  // prepare error reporter
                  ArrayReporter reporter = new ArrayReporter();

                  // run validation mode (you can also try task4/input/Order_err.xml)
                  xdoc.setProperties(props);
                  xdoc.xparse("task4/input/Orders.xml", reporter);

                  // check errors
                  if (reporter.errorWarnings()) {
                           // write log file with errors
                           PrintStream ps = new PrintStream("task4/errors/Orders_err.txt");
                           reporter.printReports(ps); //vytisteni chyb
                           ps.close();
                           System.err.println("Incorrect input data");
                  } else {
                           System.out.println("OK");
                  }
         }
}
```

Výstup programu s doplněnými udaji s adresou a cenou bude uložen v adresáři „task4/vystup/Objednavky.xml“:

```
<Orders id="123456789">
  <Order CustomerCode="ALFA"
    Number="123">
    <DeliveryPlace>
      <Address City="KLADNO"
        House="5"
        Street="KLADENSKÁ"
        ZIP="54321"/>
    </DeliveryPlace>
    <Item Price="4680.0"
      ProductCode="0002"
      Quantity="2"/>
    <Item Price="3455.0"
      ProductCode="0003"
      Quantity="1"/>
  </Order>
  <Order CustomerCode="BETA"
    Number="124">
    <DeliveryPlace>
      <Address City="MĚLNÍK"
        House="5"
        Street="MĚLNICKÁ"
        ZIP="45321"/>
    </DeliveryPlace>
    <Item Price="602.5"
      ProductCode="0001"
      Quantity="5"/>
  </Order>
</Orders>
```

4.4.2 Varianta 2: s externími Java metodami

X-definice „src/task4/Orders2.xdef“:

```
<xd:def xmlns:xd="http://www.syntea.cz/xdef/3.1" name="Orders" root="Orders">
```

```
<xd:declaration>
   external method {
      boolean task4.Orders2ext.customer(XXData);
      boolean task4.Orders2ext.item(XXData);
      String task4.Orders2ext.price(XXNode);
      void task4.Orders2ext.closeAll(XXNode);
      void task4.Orders2ext.writeObj(XXNode);
   }
   Element address;
   uniqueSet checkObjId int();
</xd:declaration>

<Orders xd:script="finally closeAll()" id="num(9)">
   <Order xd:script="occurs +; finally writeObj(); forget"
      Number="checkObjId.ID()"
      CustomerCode="customer()">
      <Item xd:script="occurs 1..100;"
         ProductCode="item()"
         Quantity="int"/>
   </Order>
</Orders>

<Order Number="int" CustomerCode="string">
   <DeliveryPlace>
      <Address xd:script="create address"
         Street="string(2,100)"
         House="int(1,9999)"
         City="string(2,100)"
         ZIP="num(5)"/>
   </DeliveryPlace>
   <Item xd:script="occurs 1..100;"
      ProductCode="string;"
      Price="float; create price()"
      Quantity="int"/>
</Order>

</xd:def>
```

Java program „src/task4/Orders2.java“:

```
package task4;

import cz.syntea.xdef.reporter.ArrayReporter;
import  cz.syntea.xdef.XDDocument;
import  cz.syntea.xdef.XDFactory;
import  cz.syntea.xdef.XDPool;
import java.io.PrintStream;

public class Orders2 {
        public static void main(String[] args) throws Exception {
                // compile XDPool from the X-definition
                XDPool xpool = XDFactory.compileXD(null, "src/task4/Orders2.xdef");

                // Create the instance of XDDocument object (from XDPool)
                XDDocument xdoc = xpool.createXDDocument("Orders");

                // set the instance of Orders2ext as user object
                xdoc.setUserObject(new Orders2ext(xpool, // the instance of Orders2ext
                  "task4/input/Products.xml", // file with the infromation obout commodity items
                  "task4/input/Customers.xml", // file with the information abou customers
                  "task4/output/Orders.xml")); // output file.

                // Prepare error reporter
                ArrayReporter reporter = new ArrayReporter();

                // run validation mode (you can also try task4/input/Order_err.xml)
                xdoc.xparse("task4/input/Orders.xml", reporter);

                // Check errors
                if (reporter.errorWarnings()) {
                        // write log file with errors
                        PrintStream ps = new PrintStream("task4/errors/Orders_err.txt");
                        reporter.printReports(ps);
                        ps.close();
                        System.err.println("Incorrect input data");
                } else {
                        System.out.println("OK");
                }
        }
```

```
}
```

Java source s externími methodami „src/task4/Orders2ext.java“:

```
package task4;

import cz.syntea.xdef.reporter.ArrayReporter;
import cz.syntea.xdef.xml.KXmlOutStream;
import cz.syntea.xdef.xml.KXmlUtils;
import cz.syntea.xdef.XDDocument;
import cz.syntea.xdef.XDPool;
import cz.syntea.xdef.proc.XXData;
import cz.syntea.xdef.proc.XXNode;
import java.io.IOException;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpression;
import javax.xml.xpath.XPathFactory;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

/** External methods called from Order2.xdef */
public class Orders2ext {

        Element _items, _customers;
        int _errors, _errorsOld, _count;
        KXmlOutStream _output;
        XDPool _xpool;

        private final XPath _xp = XPathFactory.newInstance().newXPath();

        public Orders2ext(XDPool xpool,
                String items,
                String customers,
                String output) throws IOException {
                _items = KXmlUtils.parseXml(items).getDocumentElement();
                _customers = KXmlUtils.parseXml(customers).getDocumentElement();
                _output = new KXmlOutStream(output, "UTF-8", true);
                _xpool = xpool;
                _errorsOld = _errors = _count = 0;
        }

        public static boolean customer(XXData xdata) {
                /* Get the instance of this class. */
                Orders2ext u = (Orders2ext) xdata.getUserObject();
                String s = xdata.getTextValue(); /* get attribute value. */
                try {
                        /* Find the customer. */
                        XPathExpression xExpr = u._xp.compile("Customer[@CustomerCode='" + s + "']");
                        NodeList nl =
                                (NodeList) xExpr.evaluate(u._customers, XPathConstants.NODESET);
                        if (nl == null || nl.getLength() == 0) {
                                /*customer not found, increase erro counter and report an error.*/
                                u._errors++;
                                xdata.error("Incorrect customer code: " + s, null);
                                return false; /* returns false -> incorrect. */
                        }
                        return true; /* Customer found, OK */
                } catch (Exception ex) {
                        u._errors++;
                        xdata.error("Unexpected exception: " + ex, null);
                        return false;
                }
        }

        public static boolean item(XXData xdata) {
                /* Get the instance of this class. */
                Orders2ext u = (Orders2ext) xdata.getUserObject();
                String s = xdata.getTextValue(); /* get attribute value. */
                try {
                        /* Find description of the Item according to code. */
                        XPathExpression xExpr = u._xp.compile("Product[@Code='" + s + "']");
                        NodeList nl = (NodeList) xExpr.evaluate(u._items, XPathConstants.NODESET);
                        if (nl == null || nl.getLength() == 0) {
                                /* Item not found, increase the error counter and report an error. */
                                u._errors++;
                                xdata.error("Incorrect item number: " + s, null);
                                return false;
                        }
                        return true; /* Item was found, OK */
```

```
                } catch (Exception ex) {
                        u._errors++;
                        xdata.error("Unexpected exception: " + ex, null);
                        return false;
                }
        }

        public static void writeObj(XXNode xnode) {
                /* Get the instance of this class. */
                Orders2ext u = (Orders2ext) xnode.getUserObject();
                if (u._errors != u._errorsOld) {
                        /* an new error occured, do not write recore*/
                        u._errorsOld = u._errors; /* save error counter to "errorsOld". */
                } else {
                        /* Check if this the first record. */
                        if (u._count++ == 0) {
                                /* first time, so write the root element. */
                                u._output.setIndenting(true); /* nastaveni indentace na vystupu. */
                                u._output.writeElementStart(
                                xnode.getElement().getOwnerDocument().getDocumentElement());
                        }
                        /* Prepare XDDocument for the construction according model "Order". */
                        XDDocument xdoc = u._xpool.createXDDocument("Orders");
                        xdoc.setUserObject(u);
                        Element el = xnode.getElement();
                        xdoc.setXDContext(el);
                        try {
                                String s = el.getAttribute("CustomerCode");
                                XPathExpression xExpr = u._xp.compile(
                                        "Customer[@CustomerCode='" + s + "']/Address");
                                NodeList nl = (NodeList) xExpr.evaluate(
                                        u._customers, XPathConstants.NODESET);
                                Element adresa = (Element) nl.item(0);
                                xdoc.setVariable("address", adresa);
                                /* Create the object and write it. */
                                ArrayReporter reporter = new ArrayReporter();
                                u._output.writeNode(xdoc.xcreate("Order", reporter));
                        } catch (Exception ex) {
                                // do nothing, an error was already reported when parsed
                        }
                }
        }

        public static void closeAll(XXNode xnode) {
                /* Get the instance of this class. */
                Orders2ext u = (Orders2ext) xnode.getUserObject();
                /* Check if a record was written. */
                if (u._count != 0) {
                        /* Yes, close the stream. */
                        u._output.writeElementEnd();
                        u._output.closeStream();
                }
        }

        public static String price(XXNode xnode) {
                try {
                        /* Get the instance of this class. */
                        Orders2ext u = (Orders2ext) xnode.getUserObject();
                        Element el = xnode.getXDContext().getElement(); /* Actual context. */
                        String s = el.getAttribute("ProductCode"); /* get commodity code. */
                        /* Find the item description. */
                        XPathExpression xExpr = u._xp.compile("Product[@Code='" + s + "']");
                        /* We already know that it exists. */
                        NodeList nl = (NodeList) xExpr.evaluate(u._items, XPathConstants.NODESET);
                        Element el1 = (Element) nl.item(0);
                        /* get qouantity as a number */
                        int pocet = Integer.parseInt(el.getAttribute("Quantity"));
                        /* compute price */
                        float cena = Float.parseFloat(el1.getAttribute("Price")) * pocet;
                        /* Return it as a string */
                        return String.valueOf(cena);
                } catch (Exception ex) {return "-1"; /* this never should happen! */}
        }
}
```