Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

# Bakalářská práce

# Porovnání verzí aplikací na platformě Oracle Developer a Oracle DB

Plzeň, 2012 Ondřej Šimice

# PROHLÁŠENÍ

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 7. 8. 2012

Ondřej Šimice

PODĚKOVÁNÍ

Tímto bych chtěl poděkovat vedoucím bakalářské práce Ing. Luďkovi Kratochvílovi a Ing. Petrovi Přibylovi ze společnosti CCA a Ing. Martinovi Zímovi, Ph.D. z FAV ZČU za trpělivost, cenné rady a přívětivé vedení po celou dobu bakalářské práce.

# ABSTRACT

**Comparison of applications' versions on platform of Oracle Developer and Oracle DB**

The subject of my bachelor thesis was to design mechanism for comparing two instances of an application created by CCA. Application is created on platform of Oracle Developer and Oracle DB. The mechanism must be able to compare database layer of an application and application directory structure. Company CCA already has a tool for creating fingerprint database layer. The result of this thesis is a tool for creating fingerprint application directory structure and tool for comparing database layer of the application and application directory structure. The tool gets as an input two fingerprints of database layer or two fingerprints of directory structure, compares it, saves results of comparison to list of results in memory and shows the results on standard output.

# ABSTRACT

**Porovnání verzí aplikací na platformě Oracle Developer a Oracle DB**

Předmětem mé bakalářské práce bylo navrhnout mechanismus pro porovnání dvou instancí aplikace vytvořené firmou CCA. Aplikace je vytvořena na platformě Oracle Developer a Oracle DB. Mechanismus musí být schopen porovnat databázovou vrstvu aplikace a adresářovou strukturu aplikace. Společnost CCA již má nástroj pro vytvoření otisku databázové vrstvy. Výsledkem této práce je nástroj pro tvorbu otisků adresářové struktury a nástroj pro porovnání databázové vrstvy aplikace a adresářové struktury aplikace. Nástroj dostane jako vstup dva otisky databázové vrstvy nebo dva otisky adresářové struktury, porovná je, uloží výsledky porovnání do seznamu výsledků v paměti a zobrazí výsledky na standardní výstup.

# OBSAH

# 1. ÚVOD

V aplikacích firmy CCA se používá jádro pro společnou správu aplikací. Jádro pracuje v databázi Oracle, používá triggery a uložené procedury v jazyce PL/SQL. Součástí jádra je uživatelské rozhraní, napsané v prostředí Oracle Developer. Jádro firma CCA používá ve všech aplikacích na bázi technologie Oracle Developer. Čas od času se stane, že se jádro „rozjede“ v různých aplikacích a pracovníci firmy musí pracně dohledávat změny.

Cílem práce bylo vytvořit sadu nástrojů, které by byly schopné porovnat verze jádra v různých aplikacích, a to jak databázovou vrstvu, tak vrstvu uživatelského rozhraní. V databázi je jádro uloženo v samostatném schématu, pro každou aplikaci v jiné databázi. Nástroj musí umět porovnávat tato schémata a objekty v nich obsažené. Uživatelské rozhraní tvoří sada souborů a složek, nástroj porovná stromovou strukturu složek a souborů a vlastnosti souborů.

Nyní se podíváme, co nás čeká v následujících kapitolách. Druhá kapitola je věnována aplikaci ISSPOL. Dozvíte se něco o struktuře aplikace a systémovém katalogu databáze Oracle. Poté budou popsány databázové a souborové objekty aplikace. Třetí kapitola popisuje stávající řešení problému, nástroje při něm využívané a jejich použití. Čtvrtá kapitola se zabývá návrhem řešení zadání bakalářské práce a v páté kapitole popisuji implementaci tohoto řešení. A v šesté kapitole se můžete dočíst, jak byla aplikace otestována přímo v prostředí firmy CCA.

# 2. APLIKACE ISSPOL

## 2.1. Strukturou aplikace

Aplikace ISSPOL (Informační Systém pro SPOLečnou část) je tvořena databázovou vrstvou a aplikační vrstvou. Databázová vrstva je tvořena objekty databáze Oracle (například tabulka, PL/SQL kód, pohled, trigger, apod.). Databázové vrstvě se blíže věnuje bod 2.2 a bod 2.3. Aplikační vrstva je vytvořena v prostředí Oracle Developer a skládá se hlavně z objektů Oracle Forms, Oracle Reports nebo podpůrných Java utilit. Aplikační vrstvě se blíže věnuje bod 2.4.

## 2.2. Systémový katalog Oracle

Systémový katalog se skládá z množin pohledů. Tyto pohledy mohou být tří typů odlišených prefixem názvu pohledu:

- USER – všechny objekty, které vlastní aktuálně přihlášený uživatel
- ALL – všechny objekty, ke kterým má tento uživatel přístup
- DBA – všechny objekty (pohledy pro DBA – database administrator)

Sloupce v každé množině jsou identické až na několik výjimek:

- Pohledy s prefixem user většinou nemají sloupce OWNER, protože se předpokládá, že jejich vlastníkem je uživatel, jenž je spustil.
- Některé DBA pohledy mají navíc další sloupce s informacemi, jenž jsou užitečné pro DBA.

Pro pohledy s prefixem DBA nejsou vytvářeny synonyma, protože by tyto pohledy měl sledovat pouze DBA. Jiný uživatel musí pro přístup k pohledům použít jméno majitele (SYS.název_pohledu), má-li oprávnění k přístupu nebo si musí vytvořit vlastní synonymum.

Jeden z možných přístupů do schématu SYS (a tedy i do systémového katalogu) ovlivňuje inicializační parametr O7_DICTIONARY_ACCESSIBILITY. Pokud je nastaven na true a uživatel dostane nějaké právo s parametrem ANY („cokoliv“), má přístup i do schématu SYS, pokud je nastaveno na false, uživatel nemá přístupné objekty ze schématu SYS, přestože má nastaven parametr ANY. Jedná se o bezpečnostní opatření Oracle. Tento parametr můžou podle svého uvážení měnit pouze administrátoři databáze. Infomace čerpány z [1], [3] a [5].

## 2.3. Databázové objekty aplikace

- TABLE – tabulka
    - TABLE_COLUMN – sloupec tabulky (není samostatný databázový objekt)
    - CONSTRAINTS – omezení (např.: integritní, primární klíč, cizí klíč)
    - INDEX
    - TRIGGER
- PL/SQL kód – program uložený v jazyce PL/SQL („jazyk SQL + podmínky a smyčky“)
    - FUNCTION - funkce
    - PROCEDURE - procedura
    - PACKAGE - balík
- VIEW – databázový pohled
- SEQUENCE - sekvence
- ROLE - role
- SYNONYM – synonymum

V mé bakalářské práci se budu zabývat pouze tabulkou, PL/SQL kódem a pohledem. Objekty sekvencí, rolí a synonyma se v aplikaci ISSPOL také vyskytují, ale jejich porovnání není součástí zadání bakalářské práce. Toto porovnání může být implementováno jako součást rozšíření v jiné bakalářské nebo diplomové práci.

## 2.4. Souborové objekty aplikace

V tomto bodě popíšu adresářovou strukturu aplikace ISSPOL, které se pro různé verze může trochu lišit, ale její kostra bude z velké části stejná. Pro porovnávání adresářové struktury použité v mé aplikaci však tato adresářová struktura není důležitá, protože moje aplikace umí porovnat libovolnou adresářovou strukturu.

Kořen adresářové struktury aplikace ISSPOL začíná zpravidla **\CCAPL** a obsahuje obvykle tyto složky:

- com - obsahuje utility pro doplnění Oracle Forms, mohou to být například nějaké .dll knihovny
- fmx - obsahuje kompiláty formulářů Oracle v rozlišení 800 x 600px
- fmx1024 – obsahuje kompiláty formulářů Oracle v rozlišení 1024 x 768px
- icons - obsahuje ikony a obrázky použité ve formulářích Oracle

- java
    - jre – Java Runtime Environment (prostředí pro běh Javy) verze 1.4
    - jre5 – Java Runtime Environment verze 1.5
    - jre6 – Java Runtime Environment verze 1.6
    - lib – java aplikace, různé knihovny a utility v javě pro podporu aplikace ISSPOL
    - web – Java Enterprise aplikace – různé utility pro web
- lib – obsahuje knihovny Oracle Developeru
- mmx – obsahuje kompiláty menu Oracle Developeru (menu je uloženo jako zvláštní modul)
- org – historická složka, nepoužívá se
- oraterm – nastavení pro klienta Oracle Developeru. Oracle Forms jsou multiplatformní – mohou běžet například i na terminálech. Z tohoto vyplývá, že na různých platformách může být různé ovládání. A právě zde je uloženo mapování ovládání a vzhledu pro různá prostředí (konfigurace klávesových zkratek, fontů, apod.). Například v jednom prostředí se bude provádět commit (příkaz jazyka SQL, který ukončí databázovou transakci s uložením výsledků modifikací datových objektů během transakce zviditelněním změn pro ostatní uživatele databáze) klávesou F10 a v jiném to může být přímo klávesa s nápisem commit.
- printers – obsahuje definiční soubory tiskáren pro Oracle Developer (nastavení pro pro různé typy tiskáren)
- příručky – uživatelská dokumentace
    - ISSPOL – zpravidla .htm soubory s uživatelskou dokumentací
    - images – soubory s obrázky používané v uživatelské dokumentaci
- protokol – obsahuje informace o tom, co bylo upraveno v této verzi aplikace, co bylo přidáno nebo změněno
- rep – kompilátory pro Oracle Reports (tiskové sestavy)
- vzory – šablony pro tisk ve dvou formátech .doc a rtf
    - doc – šablony pro tisk ve formátu .doc
    - rtf – šablony pro tisk ve formátu .rtf
- w-script – distribuční skript log pro databázi (instalace + patche)

# 3. STÁVAJÍCÍ ŘEŠENÍ

Srovnání databází bylo již dříve řešeno jako diplomová práce a modifikováno jako jiná diplomová práce a dále upravováno pracovníky společnosti CCA. Možnosti srovnání jsou však nedostačující. Proto je snaha vytvořit buď řešení vycházející ze stávajícího, nebo řešení nové. Měl jsem za úkol seznámit se s dosavadním řešením.

Firma vyvíjí dva typy aplikací: První typ je pro soukromé firmy a druhý pro státní správu. Je zde snaha o to, aby každá aplikace obsahovala stejné jádro. Dostal jsem k prostudování předchozí řešení, které obsahovalo soubory pro export struktury databáze a skripty pro kontrolu. Nyní je zde vypíšu a popíšu jejich funkčnost:

- CCAS_KONTROLA.sql je package pro export struktury databáze.
- CCAS_SAA_VAZEBNI_PROMENNE_PKG.sql obsahuje podpůrné objekty pro export výsledků z package CCA_KONTROLA do souboru na disku. Je nutné dodat, že export do souboru na disku z PL/SQL je sice možný přes package se jménem DBMS_OUTPUT, ale je zde omezení maximální velikostí bufferu (což je nevyhovující).
- CCAS_KONTROLA ukládá výsledky do pomocné tabulky, která se exportuje do souboru SQL dotazem.
- CCAS_SAA_VAZEBNI_PROMENNE_PKG.sql obsahuje univerzální modul pro práci s výše zmíněnou pomocnou tabulkou.

Tyto databázové objekty jsou součástí jádra. (To znamená, že jsou v databázovém schématu ISSPOL). Mohou být založeny i pod jiným vlastníkem. Stačí, když tento vlastník bude mít práva na pohledy ze systémového katalogu s admin option. Dále tu máme skript export.sql, který spouští samotný export struktury databáze. Export je prováděn do souboru export.lis – tzn. export schématu ISSPOL, teoreticky je možné změnou parametrů ve volání procedur v CCAS_KONTROLA exportovat libovolné schéma.

Pracovníci společnosti CCA nazývají výstupní soubory s otiskem databáze jako „listingy“, proto si vysloužily příponu .lis. Od této chvíle budu v textu taktéž tyto soubory označovat jako listingy nebo také otisky databázové vrstvy.

## 3.1. Formát listingu

Každý objekt začíná “/<název objektu>/“. Za druhým lomítkem bývají zpravidla další vlastnosti objektu, které jsou vždy odděleny lomítkem, některá vlastnost se může skládat z více informací a je pak oddělena ještě tečkou. Vlastnosti objektu můžou být

nepovinné – poznáme je podle uzavření do hranatých závorek v popisu objektu (nikoli však v listingu!).

**Blok tabulky, pohledy**

```
/table/<owner>/<name>[/<komentář>]
/view/<owner>/<name>/<SQL>[/<komentář>]
<sloupec>/<typ>/<délka>[.<des.míst>]/<notnull:N|Y>/<koment-
ář>[/<default>]
```

Řádek začínající /table/ označuje tabulku. Na stejném řádku jsou ještě lomítkem uvedené další vlastnosti tabulky. Owner označuje vlastníka tabulky, neboli schéma do kterého tabulka v databázi patří. Name představuje jméno tabulky a komentář je nepovinný atribut, který je ve stávající době nevyužit a je vždy nastaven na 610000. Po této řádce může následovat sloupec tabulky, který je v popisu výše na třetím řádku. Dále pak nějaký jiný objekt patřící pod tabulku (constraint, index, trigger), nebo jiný objekt, který už však logicky nepřísluší tabulce. Sloupec patří logicky pod tabulku a sám o sobě nemá význam. Popíši-li jeho složení, tak na pozici <sloupec> se nachází jméno sloupce, za lomítkem pak typ položek ve sloupci (např.: VC znamená datový typ varchar), za dalším lomítkem je rozsah datového typu, který může mít u různých datových typů různý význam. Za dalším lomítkem následuje položka nullable. Může mít obsah N nebo Y. Pokud obsahuje Y, znamená to, že sloupeček nesmí být null – tedy, že tato hodnota musí být v databázi vždy vyplněna. N znamená opak – hodnota může zůstat prázdná. Poslední povinný atribut je komentář, který je nevyužit a je opět nastaven na 610000. Za komentářem může následovat nepovinný atribut default, který je nevyužit.

Řádek začínající /view/ uvozuje pohled. Vlastnosti <owner> a <name> značí vlastníka a jméno pohledu. Obě mají podobný význam jako u tabulky. Za čtvrtým lomítkem na tomto řádku je zakódován SQL kód pohledu. Za dalším lomítkem následuje opět nepovinný a nevyužitý komentář. V listingu je jeho hodnota vždy 610000. Po této řádce může následovat sloupec pohledu, jehož struktura je stejná jako u tabulky, nebo nějaký jiný objekt, který nemá už s pohledem nic společného.

**Blok omezení**

```
/constraint/<schema>/<tabulka>
<jméno>/<typ>[/<další údaje>]
<jméno>/C/<check>
<jméno>/R/[<owner>.]<constraint>/<on delete cascade: C|N>
```

Řádek začínající /constraint/ uvozuje blok omezení, který sám o sobě nemá žádný význam, protože logicky patří pod tabulku, musí tedy následovat po bloku tabulka. Vlastnost <schema> určuje, do jakého schématu omezení patří a <tabulka> znamená, ke které tabulce přísluší. Po této řádce následují jednotlivá omezení. První je vždy jméno omezení, po lomítku následuje typ omezení: Pokud jde o typy R nebo C, tak můžou být za lomítkem další údaje, jinak nenásledují žádné další údaje. Omezení typu C má navíc za typem kontrolní součet z podmínky omezení typu check. Typ R má po lomítku za položkou typ jméno omezení, které představuje unikátní, nebo cizí klíč v jiné tabulce. Za dalším lomítkem je přepínač on delete cascade (C – nastaveno on delete cascade, N - nenastaveno).

**Blok indexu**

```
/index/<schema>/<tabulka>
[<schéma>.]<index>/<unique U|N>/<počet sloupců>/<zabezpeče-
ní>
```

Řádek začínající /index/ uvozuje blok indexů. Význam vlastností <schema> a <tabulka> je stejný, jako u constraintu. Další řádky popisují jednotlivé indexy. Zaměříme – li se na řádek popisující konkrétní index, tak můžeme vidět, že začíná jménem indexu, před kterým může být ještě tečkou oddělený název jiného schématu. Po názvu indexu následuje za lomítkem vlastnost unique, která nám říká, jestli je index unikátní (U) nebo není (N). Za dalším lomítkem je počet sloupců a zabezpečení je opět jen kontrolní součet.

**Blok triggeru**

```
/trigger/<schema>/<tabulka>
[<schéma>.]<trigger>/<Typ>/<Událost>/<Tělo>[/<Podmínka>]
```

Formát je opět podobný jako u constraintu a indexu. Vlastnost událost je zakódovaná událost a tělo je zakódovaný obsah těla, podmínka je nepovinný a pravděpodobně nepoužívaný atribut.

**Blok programu**

```
/source/<schema>/<typ>
<jméno>/<počet řádek kódu>/<zabezpečeni>
u balíků (PACKAGES) údaje typu: <head>.<body>
```

V tomto bloku můžeme najít PL/SQL program. Může být tří typů (uvedeno v <typ>): FUNCTION, PACKAGE, PROCEDURE. U PACKAGE je počet řádek kódu, rozdělen na počet řádek kódu hlavičky a těla (jsou odděleny tečkou). Hodnoty jsou opět

zakódovány pomocí algoritmu, který je pro účely této práce nepodstatný. Údaj zabezpečení je kontrolní součet z těla kódu a u PACKAGE je to kontrolní součet z hlavičky (jsou odděleny tečkou).

Následující bloky se v listingu také objevují. Jejich porovnání však není cílem této práce, proto jen uvedu jejich strukturu. Pro jejich přidání by stačilo postupovat podle pokynů v kapitole 5.4.

**Blok synonym**

```
/synonym/<schema>[/public]
<name>/[<schema>.]<table>
```

**Blok rolí**

```
/role
 user|role/<tabulky><sloupce><role>
```

**Blok katalogu**

```
/katalog/<schema>
<table>/<zabezpečení>
```

Zde uvádím fragment skutečného listingu, který je ve skutečnosti mnohonásobně delší:

```
/--BEGIN--/UNITEST (UNITEST.CCA.CZ) 16.12.2003 14:46
/source/ISSPOL/FUNCTION
CCAS_DEFAULT_VALUE/32/380F13
CCAS_IIF_B/13/FC04D4
CCAS_IIF_C/13/FC04D4
CCAS_IIF_D/13/FC04D4
CCAS_IIF_N/13/FC04D4
CCAS_POCET_SPUSTENI_APLIKACE/4/653233
CCAS_SSK_VRAT_TITUL/11/CB9A9C
CCAS_VERZE_DATABAZE_APLIKACE/10/949A6B
FG_GET_PARAM/16/5E6837
F_SAJ_ID_OSOBY/16/5E6837
/source/ISSPOL/PACKAGE
CCAA_PRISTUPY_K_OBJEKTUM_PKG/11.8E/CB9A9C.3157C6
CCAE_EVIDENCE_DISTRIBUCE/2A.84/010509.4756B9
CCAS_ADMIN_PKG/4F.202/323806.F9DC0E
CCAS_ALERT_PKG/30.57/D4D6DE.3B474A
```

```
CCAS_ANALYZE_PKG/C.1A/FE0005.F9CFCF
CCAS_APPLICATION_INFO/6F.74/42416D.373671
CCAS_BTE_CVS_PKG/B2.1B0/15E02E.42DC3F
CCAS_BTE_DXC_PKG/2D.3E/696C72.E1B4B4
CCAS_BTE_STGS_PKG/97.1DB/EA2DA1.FEC10F
CCAS_BTE_STYLY_PKG/2B.79/353809.2C2D7A
CCAS_CISLO_TEXT_PKG/9.D8/9C9FA2.6CB8F3
CCAS_CTI_PARAMETR/71.3D6/A4A3D2.81CF0A
```

### 3.1.1. Popis kontroly aplikací

Dostal jsem k dispozici aplikaci pro grafické porovnání a dvou listingů struktury databáze, abych si dokázal udělat představu, jak stávající řešení funguje. Intuitivní postup můžete vidět z obrázků A1 až A3, které jsou umístěny v příloze A, nebo si jej přečíst v následujícím odstavci.

Po spuštění aplikace na porovnání listingů uvidíte stejné okno jako je na obrázku A1. Tlačítky přidat a odebrat lze přidat a odebrat listingy. Až budeme mít na každé straně jeden listing (obrázek A2), můžeme je porovnat stisknutím tlačítka zobrazit rozdíly. Dostaneme porovnání kontrolované aplikace vůči referenční. Příklad výsledku porovnání je vidět na obrázku A3. Defaultně se zobrazují jen ty prvky, ve kterých se jádra liší. Kliknutím na „zobrazit vše“ se zobrazí i ty prvky, které jsou v obou aplikacích stejné. Vlevo je kořenová struktura, která obsahuje: Kořen – to je naše nultá úroveň. Na úrovni pod ním jsou schémata – zde vidíme schémata ISSPOL a ISYZ. Každé schéma může obsahovat nějaké položky jako například funkce, balíky, veřejná synonyma, tabulky, pohledy. Některé z nich však obsahovat nemusí. A naopak tyto položky můžou mít navíc vnořeny ještě další položky a ty další, … Maximální úroveň zanoření mi není známa, ale závisí na typu objektu a na zvolení atributů, které budou porovnávány – to je však pevně zakotveno v aplikaci a nemůžeme to změnit. Ledaže bychom změnili zdrojový kód aplikace. V pravém panelu pak vidíme detaily odlišností na jednotlivých úrovních vnoření. To co uvidíme vpravo, závisí přímo na odlišnostech verzí jader a i na tom na jakou položku a do jaké úrovně jsme se ve stromové struktuře v levé části „proklikali“. Nevýhodou této aplikace, je že neumí zobrazit rozdíl dvou adresářových struktur aplikace ISSPOL.

## 3.2. Postup instalace stávajícího řešení

Z domovských webových stránek Oracle (v [1]) jsem si musel stáhnout instalaci databáze Oracle 11g XE (protože ta je zadarmo a mohu si ji tak nainstalovat i doma). Po instalaci databáze lze pustit SQLPlus a připojit a použít podobný postup. Může se lišit maximálně přístupové heslo a login k databázi.

**Toto je postup, jak je možné vytvořit strukturu databáze na základě skriptů, které jsem dostal:**

```
Connect system/manager@xe
Create user isspol identified by isspol;
Grant connect, resource to isspol;
Grant create view to isspol;
Grant select any table to isspol with admin option;
Connect isspol/isspol@xe
@Ccas_saa_vazebni_promenne_pkg.sql
@ccas_kontrola.sql
```

Příkazy nastavují prostředí databáze, aby byla možná instalace schématu `ISSPOL` a následně instalují toto schéma. Nebo lze balíky spustit pomocí SQL Developeru, který lze stáhnout také na stránkách Oracle (v `[1]`).

# 4. NÁVRH VLASTNÍHO ŘEŠENÍ

Mechanismus pro porovnání dvou nezávisle nainstalovaných instalací aplikace ISSPOL musí umět porovnat jak databázovou vrstvu, tak vrstvu uživatelského rozhraní (adresářovou strukturu aplikace). Společnost CCA preferuje offline řešení porovnání – vytvoření souborů s otiskem stavu aplikace z obou instancí aplikace a jejich pozdější porovnání. Termín otisk a listing má v kontextu tohoto dokumentu stejný význam.

Pro získání otisku databázové vrstvy bylo použito stávající řešení tak, jak bylo poskytnuto od společnosti CCA. Tento bod nebyl cílem této bakalářské práce.

**Mechanismus porovnání se sestává z těchto částí:**

- Získání otisku databázové vrstvy.
- Získání otisku adresářové struktury aplikace – popsáno v kapitole 4.1.
- Vytvoření otisku referenční aplikace a otisku porovnávané aplikace.
- Zadání cest k referenčnímu a porovnávanému souboru.
- Parsování souboru s otiskem a uložení načtených informací do objektů (do paměti)
- Porovnání referenčního souboru s otiskem s porovnávaným.
- Uložení výsledků do objektů pro uložení výsledků porovnání (do paměti).
- Vypsání rozdílů mezi referenční a porovnávanou aplikací z objektů s výsledky porovnání.

Porovnáním databázové vrstvy se zabývá bod 5.1. Porovnáním vrstvy uživatelského rozhraní se zabývá bod 5.2.

## 4.1. Získání otisku adresářové struktury aplikace

V kapitole 2.4 byla popsána adresářová struktura aplikace ISSPOL. Při pohledu na tuto strukturu je vidět, že jde o běžnou adresářovou strukturu s nepříliš velkou hloubkou zanoření.

Strukturu databáze lze tedy jednoduše zjistit rekurzivním prohledáváním od kořenové složky, kterou zadá uživatel.

**Mechanismus získání otisku:**

- Uživatel zadá cestu ke kořenové složce aplikace `ISSPOL` (např.: `C:/CCAPL/`) a cestu kam bude uložen otisk (např.: C:/OtiskAdresaroveStruktury.lis).
- Program rekurzivně projde adresářovou strukturu, pokud se jedná o složku, zanoří se dál, pokud se jedná o soubor, získá informace o objektu a uloží informace o něm do objektu reprezentující soubor.
- Po skončení prohledávání projedu všechny objekty podobným postupem a vytvořím z nich otisk

**Struktura otisku adresářové struktury:**

```
Složka:
/dir/<cesta ke složce včetně názvu>

Soubor:
<jméno souboru>|<typ objektu>|<velikost souboru>
```

**Příklad výstupu adresářové struktury:**

```
/dir/.
/dir/ccainstall
setup.log|f|108568
/dir/com
!INFO.TXT|f|975
aresvzr.txt|f|1910
ARIALCCA.TTF|f|49440
AWK.EXE|f|167936
BarcodeReader.dll|f|208996
bindings.xml|f|1675
/dir/java\jre\lib\i386
jvm.cfg|f|671
/dir/java\jre\lib\im
indicim.jar|f|10441
```

Ještě je třeba popsat některé specifičnosti. Jak bylo řečeno "`/dir/`" uvozuje složku, následující řádky až do dalšího elementu začínajícího "`/`" paří do této složky. Po "`/dir/`" následuje cesta ke složce. Cesta označená jako tečka, reprezentuje kořenovou složku, aby byly kam umístit soubory v kořenovém adresáři. Dále by někoho mohlo zmást, že v cestě jsou lomítka na opačnou stranu – je to pro to, že Java ukládá cestu s lomítky na tuto stranu.

## 4.2. Mechanismus pro porovnání databázové vrstvy a mechanismus porovnání vrstvy uživatelského rozhraní (adresářové struktury)

Tomuto kroku předchází pořízení otisků databázové vrstvy referenční a porovnávané aplikace, nebo pořízení těchto otisků adresářové struktury. Musí být tedy vytvořeny dva soubory.

Vytvoření otisků proběhne zvlášť, jelikož tato práce má za úkol jen pořízení otisku adresářové vrstvy. Vytvoření otisku databázové vrstvy bude provedeno již jedním nástrojem. Protože porovnání je navrženo pro obecný objekt, jsou si z pohledu porovnání například složka, pohled a tabulka ekvivalentní. Toto je docíleno tím, že jednotlivé části mechanismu implementují společná rozhraní - jako jsou: rozhraní parseru, rozhraní porovnávaného objektu, rozhraní komparátoru. Dále tu budou rozhraní pro výběr zdroje a výstupu. Jednotlivé objekty implementují tyto rozhraní a správný objekt je vybírán pomocí továrních tříd: `ParserFactory` (pro výběr parseru), `ComparatorFactory`.

Nyní může přijít na řadu vlastní porovnání. To začíná zadáním cest k referenčnímu otisku a porovnávanému otisku. Nejprve bude parsován referenční soubor řádku po řádce.

Hlavní myšlenkou parsování je to, že každý nový element je uvozován značkou "/<název objektu>/". Některé objekty jsou tzv. kořenové a některé objekty jsou zanořené do kořenových. Kořenové objekty jsou objekty, které mohou existovat samy o sobě. Mezi tyto objekty patří například: table, dir (directory), view, source (program PL/SQL – tj. funkce, procedura nebo package). Zanořené objekty jsou objekty logicky (případně i fyzicky) náležející do nějakého kořenového objektu. Např. do složky (dir) přísluší soubor (file), do tabulky náleží sloupeček tabulky (table column), constraint, index a trigger, do pohledu pak sloupeček pohledu.

Parsované objekty budou uloženy do objektů odvozených od obecného objektu a uloženy do seznamu. Poté může být stejným způsobem načten seznam porovnávaných objektů.

Každý objekt seznamu bude porovnán s každým objektem v porovnávaném seznamu, pokud je stejného typu. Pokud bude referenční objekt nalezen v seznamu porovnávaných a bude mít stejné vlastnosti, nebude se nic vypisovat. Pokud objekt chybí, uloží se do seznamu výsledků, že objekt chybí a reference na kořenový objekt, do nějž patří případně objekt sám, pokud se jedná o kořenový objekt. Pokud se objekt liší, postup je stejný, ale přidají se informace o tom, v čem se objekt liší.

Kořenové (nadřazené) objekty budou obsahovat seznam podřazených objektů, jejich porovnání bude provedeno podobně jako u kořenových v rámci porovnání nadřazeného

objektu. Seznam výsledků bude obsahovat informace o kořenových objektech, které se liší (přímo tyto objekty) a seznam zpráv o odlišnostech.

Nakonec bude zpracován seznam objektů s výsledky porovnání a bude na něm aplikován mechanismus výpisu na příslušný výstup – nyní pro jednoduchost byl zvolen jednoduchý textový výstup s kontextem objektu na obrazovku.

# 5. IMPLEMENTACE

Při implementaci návrhových vzorů, jako je např.: tovární třída, jsem nahlížel do [4]. Informace ohledně dědičnosti a rozhraní jsou také v [2].

## 5.1. Implementace mechanismu pro porovnání dvou nezávisle nainstalovaných instancí aplikace ISSPOL

Mechanismus porovnání se skládá ze tří nástrojů:

- Nástroj pro vytvoření otisku databázové vrstvy – Dodalo CCA, nebyl implementován.
- Nástroj pro vytvoření otisku adresářové struktury – byl implementován v rámci BP. Implementace je popsána v kapitole 5.2.
- Nástroj pro porovnání dvou otisků aplikace – byl implementován v rámci BP.

První nástroj je napsán v programovacím jazyce PL/SQL. Mnou vytvořené nástroje jsou napsány v programovacím jazyce Java.

## 5.2. Implementace nástroje pro vytvoření otisku adresářové struktury aplikace ISSPOL

Nástroj na vytvoření otisku adresářové struktury lze nalézt v package pojmenovaném `adres`. Tento package obsahuje hlavní třídu nástroje nazvanou `VylitiStrukturySouboruManager` a další čtyři vnořené balíky: `builder`, `colector`, `objects` a `output`.

Vytvoření otisku funguje následujícím způsobem:

Jako argumenty se programu předá cesta ke kořenové složce adresářové struktury a jméno souboru i s cestou k souboru, do něhož se má uložit otisk adresářové struktury.

Jelikož byl nástroj navrhován tak, aby jej bylo v budoucnu možné rozšířit na vytvoření otisku obecné struktury, musí být vytvořena instance třídy `DataCollectorFactory`, která má na starosti vrácení správného typu DataCollectoru (sběrače dat) při zavoní metody `createDataCollector("<typ objektu>")`. V našem případě je za typ objektu dosazeno "file", což říká, že se jedná o adresářovou strukturu a je vrácena instance `DirItemDataCollectoru` (sběrače dat pro adresářovou strukturu).

Pak už se začnou pomocí data collectoru sbírat data. Poté, co jsou data sesbírána, se projde seznam objektů a pomocí metody `computeDigest`, jenž patří do třídy `DirItemStructureBuilder`, se z atributů objeků sestaví otisk adresářové struktury.

## 5.3. Implementace nástroje pro porovnání dvou otisků aplikace ISSPOL

Nástroj pro porovnání otisku aplikace ISSPOL umí porovnat jak otisk databázové vrstvy, tak otisk adresářové struktury aplikace. Porovnání probíhá tak, že nástroj načte dva soubory s otiskem – referenčí a porovnávaný, uloží objekty do dvou kolekcí `List<IComparedObject>` z nichž jedna obsahuje objekty z referenčního souboru a druhá z porovnávaného souboru. Nad těmito dvěma kolekcemi proběhne mechanismus porovnání, při čemž se rozdíly ukládají do dalšího seznamu s výsledky porovnání. Nakonec proběhne zobrazení rozdílů výsledků tak, že se projde seznam objektů s výsledky porovnání a z těchto objektů se vypíší na konzoli výsledky porovnání – tedy rozdíly mezi referenční a porovnávanou aplikací.

### 5.3.1. Struktura zdrojových kódů nástroje

Nástroj se skládá celkem ze 49 zdrojových souborů, které obsahují třídy nebo rozhraní, a je uložen v balíku nazvaném `appInspector`. Ten obsahuje hlavní třídu `PorovnaniSouboruManager` a další tři balíky `input`, `comparator` a `output`. V balíku `input` jsou třídy a rozhraní, které mají na starosti načtení vstupních dat, jejich zpanování a uložení do objektů. V balíku `comparator` jsou třídy a rozhraní s objekty, do nichž jsou ukládaná data a komparátory pro tyto objekty. V balíku `output` jsou třídy a rozhraní pro uložení výsledků porovnání a pro jejich zobrazení. Nástroj je navržen tak, že většina tříd implementuje nějaké rozhraní, díky tomuto faktu, je návrh nástroje dost obecný a poměrně snadno rozšiřitelný.

### 5.3.2. Načtení vstupních dat

Jako vstup nástroj očekává absolutní cestu ke dvěma souborům s otiskem. Pro správné načtení, nesmí cesta obsahovat mezery. První soubor by měl obsahovat otisk referenční aplikace a druhý otisk porovnávané aplikace. Musí se jednat buď o dva otisky databázové vrstvy, nebo o dva otisky adresářové struktury aplikace. Nyní popíšu rozhraní obsažená v package `input` a v jeho vnořených package. Jedná se o rozhraní `IDigestSource` , `IDigestReader` a `IParser`.

`IDigestSource` reprezentuje získání cesty ke zdroji vstupních dat, obsahuje metodu `getFileName()` pro získání cesty, ke zdroji dat. Třída `FileDigestSource` implementuje toto rozhraní a představuje tak cestu ke zdroji dat typu soubor na disku.

Napsáním jiné třídy implementující toto rozhraní by se dal použít i jiný zdroj dat. IDigestReader je rozhraní zajišťující načtení otisku ze zdroje dat. Jeho implementací pro čtení ze souboru je třída `FileDigestReader`. `DigestReaderFactory` vrací instanci na `FileDigestReader` pomocí metody `createReader`. Čtení ze souboru je ve třídě `FileDigestReader` zajištěno v metodě `read`. Čtení probíhá v pěti krocích: získání cesty k souboru, inicializace souboru (otevření souboru a inicializace bufferů pro čtení), parsování souboru řádku po řádce až do konce souboru (metoda `parseFile()`), zavření souboru, vrácení seznamu načtených objektů.

Nyní popíšu na zdrojovém kódu algoritmus parsování souboru s otiskem:

```
private void parseFile() {
            parserFactory = ParserFactory.newInstance(); //vytvoří
//novou instanci továrny na parsery, vytvoří sem tam parsery pro
//jednotlivé typy directory, file, ...
            IParser parser = null; // reference na používaný parser
            IComparedObject comparedObject = null; //reference na
//porovnávaný objekt
            if(readNextLine() == null) return;
            //pokud není konec souboru
            do {
                  ctiDal = false;
                  if(currentLine.startsWith("/dir/")) {
                        parser = parserFactory.getParser("directory");
//vyber directory parser
                        comparedObject = parser.parse(this);//parsuj
//a získej porovnávaný objekt
                        if(comparedObject != null)
addToListOfComparedObjects(comparedObject); //pokuď není výsledek
//parsování null, přidej do seznamu objektů
                  } else if(currentLine.startsWith("/source/")) {
                        parser = parserFactory.getParser("source");
                        comparedObject = parser.parse(this);
                        if(comparedObject != null)
addToListOfComparedObjects(comparedObject);
                  } else if(currentLine.startsWith("/table/")) {
                        parser = parserFactory.getParser("table");
                        comparedObject = parser.parse(this);
                        if(comparedObject != null)
addToListOfComparedObjects(comparedObject);
                  } else if(currentLine.startsWith("/view/")) {
                        parser = parserFactory.getParser("view");
                        comparedObject = parser.parse(this);
                        if(comparedObject != null)
addToListOfComparedObjects(comparedObject);
                  } else {
                        ctiDal = true; //neznámý objekt nebo komentář,
//nedělej nic a bude se číst další řádka
                  }

                  if(currentLine == null) return; //pozor tohle není
//hloupost viz DirectoryParser, pokud je to null ukončí čtení, toto
//značí konec souboru
                  if(ctiDal) {
```

```
                        if(readNextLine() == null) return; // poku se
//má číst dál přečtu další řádku, pokud je null končím, pokud ne
//pokračuje další iterace smyčky
                    }
                }while(true);
        }
```

Třída `FileDigestReader` obsahuje tři důležité metody: `readNextLine()`, `getCurrentLine()` a `addToListOfComparedObjects(IComparedObject objekt)` a důležitou proměnnou třídy currentLine. První umí načíst novou řádku ze souboru a uloží jí do proměnné `currentLine`, druhá umí vrátit hodnotu aktuální načtené řádky a třetí umí vložit zpanovaný objekt do seznamu objektů, který bude vrácen metodou `read`. Bohužel formát otisku databázové vrstvy byl dost nešikovně zvolen – objekty nemají ukončovací značky, proto je pasování trochu komplikované. Nejdříve se připraví proměnné, které se budou používat, pak se pokusím načíst první řádku souboru. Jestliže je výsledek `null`, jedná se o konec souboru. Pokud ne, následuje zpracování načtené řádky. Pointa je v tom, že hledám objekty nejvyšší úrovně, které začínají /<objekt>/. Nachází-li se na řádku začátek složky, tabulky, pohledu nebo programu (source), řádka mi zajímá. Vyberu pro ni příslušný parser pomocí klíče z `ParserFactory` a pustím parsování. Parametr `this` v metodě parse, je tam kvůli referenci na aktuální třídu, v níž je proměnná currentLine obsahující aktuální načtenou řádku. A také proto, abychom mohli uvnitř parseru ovládat čtení další řádky. Výsledkem parsování je objekt typu IComparedObject (jakýkoliv implementující toto rozhranní). Pokud není objekt null, je vložen do seznamu načtených objektů. Nachází li se na řádku něco jiného než nějaký z objektů nejvyšší úrovně, řádka je přeskočena a testuje se další. Vše běží v cyklu do konce souboru. Nekonečný cyklus zde byl v tomto případě zvolen záměrně, pro lepší čitelnost ukončovacích podmínek uvnitř cyklu. Logická proměnná `ctiDal` je zavedena z důvodu již zmiňované absence koncových značek objektů v souboru s listingem (pokud uvnitř nějakého parseru narazím na objekt nejvyšší úrovně, ukončí se parsování objektu a vrátím se do tohoto cyklu, ale nechci načíst další řádku, nýbrž zvolit vhodný parser pro její zpracování). Při parsování je využita, znalost struktury parsovaných objektů. Oddělovačem vlastností bývá zpravidla lomítko. Oddělovač tedy najdu pomocí metody `indexOf("/")` a zpracovávám, vždy aktuální řádku, kterou mám uloženou v pomocné proměnné. Z této proměnné najdu vždy první vlastnost zleva za pomocí pozice lomítka a pak vezmu zbytek za lomítkem a přiřadím ho jako novou hodnotu pomocné proměnné.

Příklad parseru uvedu na parsování složky a souboru, ostatní parsery používají podobné techniky, parser pro tabulku je jen trochu komplikovanější tím, že má více vnořených objektů. Parser složky má na starosti třída `DirectoryComparator`. Ta v metodě `parse` zjistí jaká je aktuální řádka a uloží si jí do své lokální proměnné. Ještě jednou je zkontrolováno, jestli řádka začíná opravdu "`/dir/`", pokud ne vrátí se null a parsování této řádky končí, v hlavním cyklu třídy `FileDigestReader` se pak zpracuje další

řádka. Jestliže řádka však "`/dir/`" začíná, zpracuje se tato řádka. Po oddělení "`/dir/`" nám zůstane cesta ke složce a tu uložíme do objektu pro složku (`DirectoryComparedObject`). Po řádku složky by měly následovat soubory, proto v cyklu načteme a získáme další řádku, tu pak předáme parseru pro soubor a ten vrátí objekt souboru s jeho vlastnostmi nebo null. Pokud objekt není null přidám jej ještě do seznamu objektů složky. Tento cyklus běží, dokud nenarazí na řádku null, nebo na řádku začínající „/", v tom případě se ukončí cyklus a vrátí se objekt složky, který obsahuje vlastnosti složky (její cestu) a seznam jejich souborů (seznam objektů typu `FileComparedObject`). Princip parsování zbylých objektů je obdobný.

### 5.3.3. Porovnávané objekty a komparátory

Třídy a rozhraní pro uložení porovnávaných objektů a pro jejich porovnání jsou uloženy v balíku `appInspector.comparator`. Je tu rozhraní `IComparedObject`, které musí implementováno všemi porovnávanými objekty, a `IComparator`, jehož implementace musí být vytvořena pro každý porovnávaný objekt. Nakonec je tu tovární třída `ComparatorFactory`, která má na starosti vybrat podle typu objektu správný komparátor a pak už jen výše zmiňované implementace porovnávaných objektů a jejich komparátorů. Ve třídě `ComparatorFactory` je HashMapa, do níž, jsou v konstruktoru přidány využívané komparátory. Jako klíč je volen název třídy získaný z objektu a jako hodnota je volena nová instance komparátoru. Dále pak obsahuje metodu `getComparator(IComparedObject comparedObject)`, která podle typu objektu vybere správný komparátor tak, že získá z porovnávaného objektu jméno třídy a použije jej jako klíč do HashMapy.Porovnání popíšu opět na složce a souboru, pro další objekty je to analogické.

Komparátor pro složku má jako vstup referenční objekt typu `IComparedObject` a seznam porovnávaných objektů také typu `IComparedObject`. Porovnání probíhá tak, že si nejdříve přetypuji porovnávaný objekt na jeho typ (v tomto případě `DirectoryComparedObject`), aby nad ním šly volat jeho getry. Do pomocné (pomocných) proměnné (proměnných) si uložim informaci (informace), které ho jednoznačně identifikují. Zde je to cesta ke složce, která obsahuje i její jméno. Pak procházím v cyklu jeden po druhém celý seznam porovnávaných objektů a nejprve se ptám, je-li objekt stejného typu (pomocí operátoru `instanceof`), pokud není, přeskočím na další iteraci (na další objekt) a test provádím znovu. Pokud je stejného typu, objekt si přetypuji a uložím také do pomocné proměnné. Nyní se ptám jestli je objekt stejný jako porovnávaný objekt (porovnám na shodu atributy, které ho jednoznačně definují, tady je to již zmiňovaný atribut cesta), výsledek si uložím do pomocné proměnné (pro lepší čitelnost) podmínka – může nabývat buď true nebo false (ano nebo ne). Pak testuji tuto podmínku, pokud byla splněna (objekt byl nalezen) porovnám jestě jeho další atributy (pokud nějaké má – složka už nemá) a porovám také jeho vnořené objekty (opět, každý vnořený objekt referenčního objektu porovnám

v cyklu s celým seznamem vnořených objektů porovnávaného objektu) – tady to jsou soubory. Porovnání těchto vnitřních objektů vrátí výsledek porovnání typu `ComparisonResult`, který obsahuje referenci na nejvyšší objekt a seznam zpráv o rozdílech (typu `Message`). Objekt nejvyšší úrovně má také svůj `ComparisonResult`, ten vytvořím tak, že do jeho atributu vložím referenci na objekt nejvyšší úrovně (zde je to složka), a pokud nějaká složka chybí, vložím zprávu o tom, že složka chybí. Pokud chybí nějaký soubor složky nebo se liší, překopíruji všechny zprávy z výsledku porovnání nižšího objektu do nadřazeného objektu. Nakonec vrátím výsledek porovnání nejvyššího objektu (zde je to složka).

### 5.3.4. Uložení výsledků a výstup

Tato podkapitola se zabývá uložením výsledků porovnání a jejich zobrazením. V předchozí podkapitole byl již zmíněn objekt `ComparisonResult` a objekt `Message`. Druhý zmiňovaný má atributy typu `IComparedObject` (objekt, ve kterém došlo k nějakému rozdílu) a atribut typu `String` se zprávou popisující tento rozdíl. `ComparisonReusult` obsahuje rovněž atribut `IComparedObject` (sem se ukládá objekt nejvyšší úrovně) a seznam všech zpráv o rozdílech v tomto objektu a jeho podřízených objektech (seznam objektů typu `Message`). Nakonec je zde třída `ResultRenderer`, která je zodpovědná za interpretaci výsledků porovnání (tedy interpretaci seznamu objektů typu `ComparisonResult`). Důvodem uložení informací o porovnání do objektů je vize rozšíření tohoto nástroje (viz rozšíření v závěru). Takto totiž můžeme z výsledku porovnání získat jakoukoliv uloženou informaci. Rendering výsledků porovnání je prováděn na standardní výstup a je řešen tak, že se projde v cyklu celý seznam výsledků, podle typu objektu nejvyšší úrovně se přidá na začátek řádky jméno objektu, jeho kontext a dvojtečka a mezera (toto tvoří prefix) a za prefix se pak přidá zpráva o porovnání. Pro složku je to například: "`Složka <cesta>: <zpráva o porovnání>`".

### 5.3.5. Hlavní proces řídící porovnání

Porovnávání databázové vrstvy a adresářové struktury probíhá zvlášť – protože máme dva a dva soubory. Teoreticky by bylo možné porovnání sjednotit.

Třída `PorovnáníSouboruManager` je hlavní třídou tohoto nástroje. Obsahuje metodu main a řídí činnost celého porovnání.

Z argumentů programu se načtou cesty k otiskům referenčního a porovnávaného souboru. Vytvoří se referenční a porovnávaný zdroj dat – což je jen instance s cestou k souboru. Dále se vytvoří instance továrny na čtení otisků souborů z tovární třídy `DigestReaderFactory`. Pak musíme vytvořit instanci readeru pro referenční i porovnávaný soubor. Čteme ze souboru, proto instance `FileDigestReaderu`. Její

reference se načte do typu `IDigestReader`, což je rozhraní. Tento přístup je použit, aby bylo možné číst v budoucnu data i z jiných zdrojů než ze souboru. Změna by spočívala pouze v napsání jedné třídy pro jiný reader a jeho zvolení. Vytvoří se dva seznamy typu `IComparedObject` – jeden pro objekty zparsované z referenčního a druhý z porovnávaného souboru. Rozhraní `IComparedObject` musí implementovat každý porovnávaný objekt. Díky tomuto mohou být všechny objekty v jednom seznamu. Začneme číst z příslušných zdrojů do těchto seznamů (z referenčního do referenčního apod.). Když jsou objekty načtené, začíná fáze porovnávání. Nejprve však musíme vytvořit seznam pro výsledky porovnání. Ukládáme jen výsledky, kdy se objekty liší. Porovnání probíhá tak, že prochází postupně každý objekt z referenčního seznamu a je porovnán příslušným komparátorem, který mu přidělí tovární metoda `ComparatorFactory` podle typu porovnávaného objektu. Po pořízení výsledků porovnání se projde seznam s výsledky a pomocí rendereru se vytvoří výstup, který se zobrazí na standardním výstupu.

## 5.4. Popis implementace porovnání nového typu objektu

Předpokládá se, že nový objekt bude již obsažen v souboru s otiskem, ať už to bude otisk čehokoliv – databázové vrstvy, adresářové struktury či nějaké jiné vrstvy, kterou by mohla aplikace společnosti CCA v budoucnosti mít a obsahovala by objekty, které by bylo třeba porovnat.

Pro implementaci porovnání nového typu objektu se musejí provést tyto kroky:

1) Vytvořit nový balík v balíku `appInspector.comparator`. Nazvu jej například `novyObjekt`. Cesta k tomuto balíku bude tedy `appInspector.comparator.novyObjekt`. Tento krok není nezbytně nutný, ale je vhodný pro dodržení stávající struktury uložení souborů nástroje pro porovnání.

2) Vytvořit nový objekt implementující rozhraní `IComparedObject` z balíku `appInspector.comparator`. Označím ho `NovyObjektComparedObject` a umístím ho do balíku `appInspector.comparator.novyObjekt` vytvořeného v předchozím bodě.

3) Vytvořit parser pro objekt implementující rozhranní `IParser` z balíku `appInspector.input.parser`. Tento parser označím `NovyObjektParser`. Pravděpodobně se bude jednat o další objekt databázové vrstvy – v tom případě bude vhodné parser uložit do balíku `appInspector.parser.database`. Pokud by se jednalo o jiný objekt, může být v balíku `appInspector.parser` vytvořen nový balík a do něj by se parser uložil.

4) Upravit konstruktor třídy `ParserFactory` přidáním jednoho řádku, který by měl při označení objektů uvedeném v bodech výše takovýto tvar:

```
parser.put("novyObjekt", new novyParser());
```

Tento řádek přidá instanci nového parseru do HashMapy ve třídě `ParserFactory`, která má na starosti vybrání správného parseru podle klíče. Klíč pro vybrání nového parseru je označen"`novyObjekt`".

5) Upravit metodu `parseFile()` ve třídě `FileDigestReader` z balíku `appInspector.input` přidáním další podmínky na test začátku nové řádky – kód by mohl vypadat asi takto:

```
else if (currentLine.startsWith("<jak má začínat nová
řádka>")) {
parser = parserFactory.getParser("novyObjekt");
//"novyObjekt" je klíč do HashMapy ve třídě
ParserFactory (vis předchozí bod).
comparedObject = parser.parse(this);
if(comparedObject != null)
addToListOfComparedObjects(comparedObject);
}
```

6) Vytvořit komparátor pro nový objekt implementující rozhraní IComparator z balíku appInspector.comparator. Tento komparátor označím novyObjektComparator a umístím ho do balíku appInspector.novyObjekt. Tělo komparátoru bude pravděpodobně hodně podobné již existujícím komparátorům.

7) Upravit konstruktor třídy ComparatorFactory přidáním jednoho řádku:

```
comparators.put(novyObjektComparedObject.class.getName(
), new novyObjektComparator());
```

8) Ve třídě `ResultRenderer` upravit metodu `render` pro správné vypsání kontextu objektu přidáním:

```
else if(comparedObject instanceof
novyObjektComparedObject) {
NovyObjektComparedObject novyObjektComparedObject =
(NovyObjectComparedObject) comparedObject;
prefix = "<Název nového objektu> " +
novyComparedObject.getName() + ": ";
//metoda getName() reprezentuje vypsání názvu objektu
pro správný kontext
}
```

# 6. OVĚŘENÍ FUNKČNOSTI NÁSTROJŮ NA APLIKACI V PROSTŘEDÍ CCA

Byly vyvinuty dva nástroje, které bylo třeba otestovat:

- Nástroj pro vytvoření otisku adresářové struktury databázové aplikace společnosti CCA.
- Nástroj pro porovnání, který umí porovnat jak databázovou vrstvu aplikace, tak adresářovou strukturu aplikace.

Ověření funkčnosti nástroje pro vytvoření otisku adresářové struktury bylo provedeno následujícím způsobem:

- Společnost CCA připravila adresářovou strukturu aplikace. Nad touto strukturou, byl puštěn nástroj pro vytvoření otisku adresářové struktury aplikace.
- Tento nástroj byl také otestován na jiné jednodušší adresářové struktuře, která měla kořenovou složku a v ní rozumný počet složek a souborů tak, aby se dalo pouhým pohledem oka porovnat, zda nástroj funguje správně.
- Tento nástroj byl uznán jako funkční.

Ověření funkčnosti nástroje pro porovnání bylo provedeno ve dvou fázích. Fáze jedna měla otestovat porovnání adresářové struktury aplikace. Fáze dvě měla otestovat porovnání databázové vrstvy aplikace.

Fáze jedna probíhala takto:

- Byl vytvořen výše zmiňovaný otisk adresářové struktury – označíme jej `A`.
- Byla vytvořena kopie otisku `A` – označíme ji `AA`.
- Byla vytvořena kopie otisku `A` a ručně v ní provedeny změny - označíme ji `B`
- Vstupem nástroje je otisk referenční struktury a otisk porovnávané struktury
- Nejprve byly pro vstup použity soubory `A` a `AA` – porovnání nevypsalo žádné změny, což je správně, protože se jedná o shodné soubory a shodné soubory znamenají shodné adresářové struktury.
- Pak byl porovnán soubor `A` proti souboru `B` a soubor `B` proti souboru `A`. Výstupy byly porovnány s očekávanými výstupy a shodovaly se s nimi.
- Nástroj byl ověřen a uznán jako funkční.

Fáze dvě probíhala takto:

- Pomocí programu na vytvoření databázové struktury, který společnost již vlastnila, byl vytvořen otisk databázové vrstvy – označíme jej opět `A`
- Podobně jako ve fázi jedna byly vytvořeny kopie tohoto otisku `AA` a `B`. `AA` byl shodný jako `A` a v `B` byly provedeny oproti `A` nějaké změny.

- Opět byl porovnán soubor `A` proti souboru `AA` a soubor `A` proti `B` a `B` proti `A`. Výstupy se opět shodovaly s očekávanými výstupy.
- Nástroj byl ověřen a uznán jako funkční.

# 7. ZÁVĚR

V rámci řešení bakalářské práce jsem se seznámil se strukturou aplikace ISSPOL, prozkoumal katalog databáze Oracle, zjistil, jak jsou v katalogu uložené jednotlivé databázové objekty, seznámil se s uložením souborů aplikace ISSPOL a své poznatky sepsal v teoretické části.

Navrhl jsem a implementoval mechanismus pro porovnání dvou nezávisle nainstalovaných instancí aplikace ISSPOL. Mechanismus umí porovnat jak databázovou vrstvu, tak adresářovou strukturu aplikace. Dle požadavků společnosti CCA bylo zvoleno offline řešení porovnání. Tedy nejdříve vytvoření souboru s otiskem stavu aplikace z obou instancí aplikace a jejich pozdější porovnání. Vyústěním tohoto přístupu je nástroj pro vytvoření otisku databázové vrstvy – ten poskytlo CCA již hotový a nebyl mnou modifikován, dále pak dva nástroje, které jsou již mojí prací – nástroj pro vytvoření otisku adresářové struktury a nástroj, který načte dva otisky různých instancí, ale stejné vrstvy aplikace (buď databázové vrstvy, nebo adresářové struktury) a na základě jejich obsahu porovná objekty v těchto aplikacích, uloží informace o výsledcích porovnání do objektů (ukládají se jen rozdíly), projde seznam výsledků a vypíše jeho obsah na `system.out` (standardní výstup). Porovnání nástrojů bylo otestováno v prostředí CCA a uznáno jako fungující a splňující body zadání.

V textu bakalářské práce jsem uvedl návod na implementaci porovnání nového typu objektu. Těžištěm mého řešení byl hlavně návrh mechanismu porovnání, na kterém se bude moci v budoucnu stavět například v rámci nějaké bakalářské či diplomové práce. Nyní uvedu některá další možná rozšíření:

- Nástroj pro vytváření otisku adresářová struktury by mohl ukládat více informací o souborech, například by mohl obsahovat navíc otisk těla souboru, který by šel pořídit pomocí třídy `MessageDigest`. Tato třída používá MD5 algoritmus.
- Nástroj pro sofistikovanější zobrazení výsledků porovnání – pravděpodobně bude zadán jako nějaká diplomová práce.
- Jiný výstup porovnání – například ve formě .html – bylo by nutno upravit renderer výsledků.
- Nový nástroj pro vytvoření otisku databázové vrstvy napsaný místo v PL/SQL v Javě za použití rozhraní JDBC, které umí pracovat s více databázemi, takže by teoreticky bylo možné porovnávat obsahy i databázové vrstvy jiné databáze než Oracle – například Postgres
- Nástroj, který by na základě výsledků porovnání, které jsou uložené v objektech výsledků, převedl tyto informace o rozdílných objektech na skript nebo SQL dotaz, který by rozdíly vhodně opravil nebo navrhl změny, které by se daly provést k docílení shody.

# LITERATURA

[1] ORACLE. Oracle [online]. 2012 [cit. 2012-01-30]. Dostupné z: http://www.oracle.com/index.html

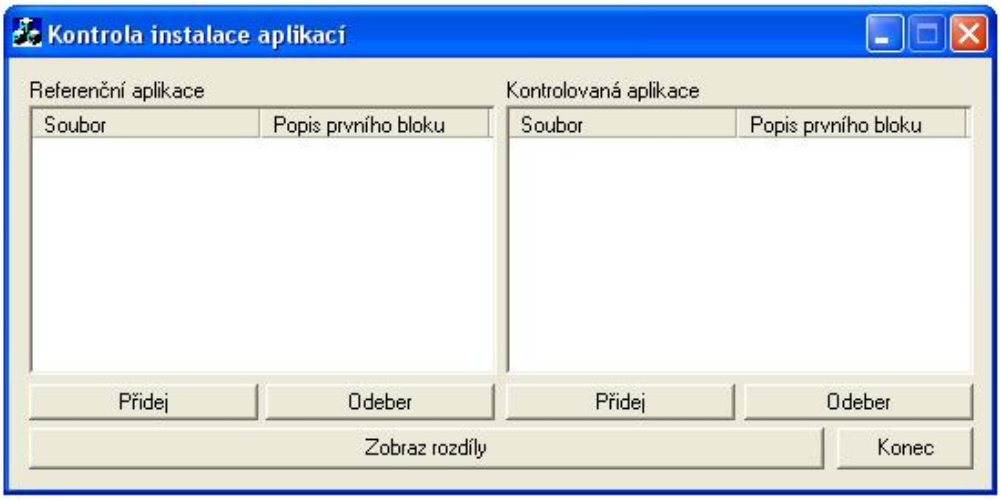
[2] HEROUT, Pavel. Učebnice jazyka Java. 1. vyd. České Budějovice: Kopp, 2001, 349 s. ISBN 80-723-2115-3.

[3] Cvičení KIV/DB2. ZÍMA, Martin. *http://www.kiv.zcu.cz/~zima/vyuka/db2/* [online]. 2011 [cit. 2012-08-07]. Dostupné z: http://www.kiv.zcu.cz/~zima/vyuka/db2/

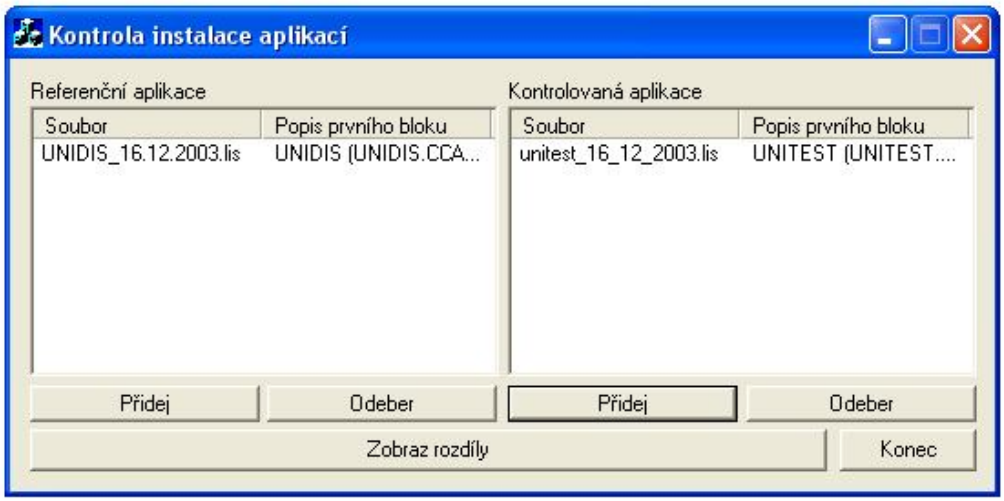
[4] PECINOVSKÝ, Rudolf. *Myslíme objektově v jazyku Java 5.0*. Vyd. 1. Praha: Grada, 2004, 601 s. ISBN 80-247-0941-4.

[5] RYDVAL, Slávek. Systémový katalog databáze Oracle. In: *NaWEBka* [online]. 2006 [cit. 2012-08-07]. Dostupné z: http://www.rydval.cz/phprs/view.php?cisloclanku=2005112502

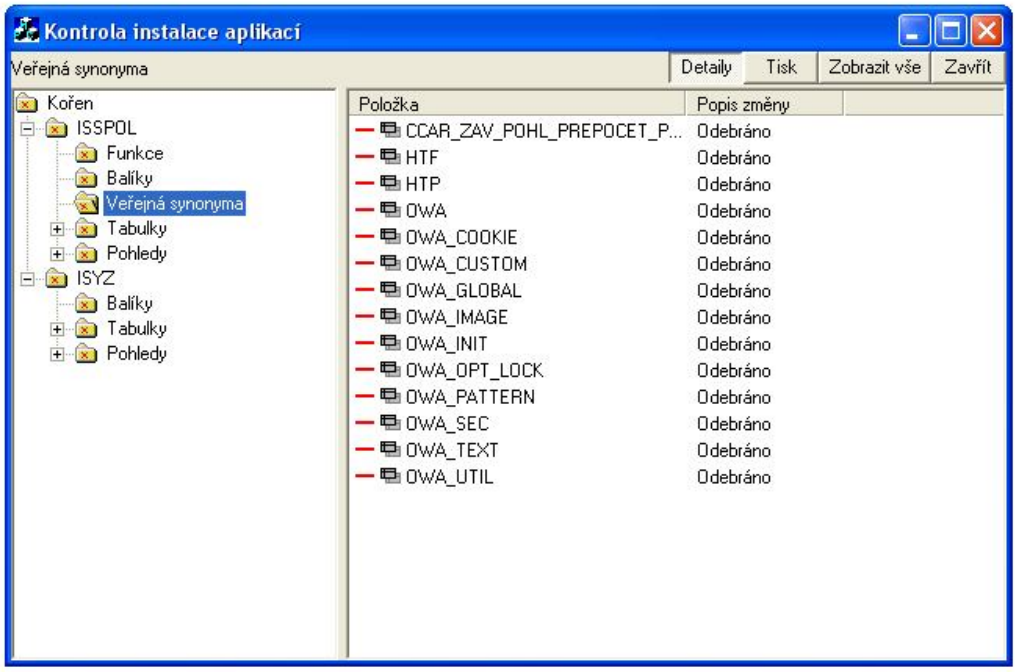
# PŘÍLOHA A




Obrázek A1 Kontrola instalace aplikací po spuštění




Obrázek A2 Přidány dva listingy




Obrázek A3 První pohled na výsledek porovnání

# PŘÍLOHA B – UŽIVATELSKÁ PŘÍRUČKA

## 1. Uživatelská příručka pro nástroj vytvoření otisku adresářové struktury

**Postup získání otisku:**

- Program se spustí pomocí příkazu:

```
java –jar vylitiAdresaroveStruktury.jar <cesta ke kořenové složce>
<cesta k listingu>
```

- Uživatel zadá cestu ke kořenové složce aplikace `ISSPOL` (např.: `C:/CCAPL/`) a cestu kam bude uložen otisk (např.: C:/OtiskAdresaroveStruktury.lis).
- Program rekurzivně projde adresářovou strukturu, a uloží její otisk.

**Struktura otisku adresářové struktury:**

```
Složka:
/dir/<cesta ke složce včetně názvu>

Soubor:
<jméno souboru>|<typ objektu>|<velikost souboru>
```

**Příklad výstupu adresářové struktury:**

```
/dir/.
/dir/ccainstall
setup.log|f|108568
/dir/com
!INFO.TXT|f|975
aresvzr.txt|f|1910
ARIALCCA.TTF|f|49440
AWK.EXE|f|167936
BarcodeReader.dll|f|208996
bindings.xml|f|1675
/dir/java\jre\lib\i386
jvm.cfg|f|671
/dir/java\jre\lib\im
indicim.jar|f|10441
```

Ještě je třeba popsat některé specifičnosti. "`/dir/`" uvozuje složku, následující řádky až do dalšího elementu začínajícího "`/`" paří do této složky. Po "`/dir/`" následuje

cesta ke složce. Cesta označená jako tečka, reprezentuje kořenovou složku, aby byly kam umístit soubory v kořenovém adresáři. Dále by někoho mohlo zmást, že v cestě jsou lomítka na opačnou stranu – je to pro to, že Java ukládá cestu s lomítky na tuto stranu.

# 2. Uživatelská příručka pro nástroj pro porovnání databázové vrstvy nebo adresářové vrstvy

**Postup pro porovnání:**

- Uživatel musí mít dva otisky struktury. Jeden otisk referenční aplikace a jeden otisk porovnávané aplikace.
- Pokud uživatel zadá dva otisky databázové vrstvy, porovná se databázová vrstva aplikace, pokud zadá dva otisky adresářové struktury, porovná se adresářová struktura aplikace.
- Program se spustí pomocí příkazu:

```
java –jar appInspector.jar <cesta k referenčnímu otisku> <cesta k porovnávanému otisku>
```

- Příklad zadání:

```
java –jar appInspector.jar C:/referencni.lis C:/porovnavany.lis
```

- Výsledek porovnání bude zobrazen na standardní výstup.