UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Zásilkový prodej v Javě

Václav Hataš

Bakalářská práce
2014

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2013/2014**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Václav Hataš**
Osobní číslo:	**I11057**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Zásilkový prodej v Javě**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem bakalářské práce je vytvořit program pro firmu zabývající se zásilkovým prodejem, s využitím programovacího jazyka Java a databázového systému MySQL.
Teoretická část:
Bude obsahovat popis programovacího jazyka Java a výčet použitých knihoven, popis databázového systému MySQL. Dále pak rozbor formátu csv použitého k exportu dat pro přepravní službu.
Implementační část:
Tato část bude popisovat tvorbu programu v jazyce Java, včetně grafického uživatelského prostředí, sloužícího k editaci, tvorbě a celkové správě objednávek a jejich exportu. Databáze bude tvořena v MySQL.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

SPELL, Brett. Java: programujeme profesionálně. Vyd. 1. Praha: Computer Press, 2002, 1022 s. ISBN 80-722-6667-5.
HEROUT, Pavel. Java: grafické uživatelské prostředí a čeština. 2. vyd. Překlad Jan Pokorný. České Budějovice: Kopp, 2007, 316 s. Encyklopedie Zoner Press. ISBN 978-80-7232-328-9.
GILMORE, W. Velká kniha PHP 5 a MySQL: kompendium znalostí pro začátečníky i profesionály. Nové, 3. vyd. Překlad Jan Pokorný. Brno: Zoner Press, 2011, 736 s. Encyklopedie Zoner Press. ISBN 978-80-7413-163-9.

Vedoucí bakalářské práce: **Ing. Zdeněk Šilar**
Katedra informačních technologií

Datum zadání bakalářské práce: **20. prosince 2013**
Termín odevzdání bakalářské práce: **9. května 2014**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D
vedoucí katedry

V Pardubicích dne 31. března 2014

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 5. 5. 2014 Václav Hataš

Poděkování

Tímto bych rád poděkoval svému vedoucímu bakalářské práce panu Ing. Zdeňku Šilarovi za to, že mi umožnil pracovat na mnou zvoleném tématu a dále za vedení, rady a podnětné připomínky při tvorbě této práce.

Chtěl bych také poděkovat své rodině za podporu při studiu a při zpracování bakalářské práce.

Anotace

Cílem této práce bylo vytvořit kancelářský program pro firmy zabývající se zásilkovým obchodem. Hlavní výstup programu je export zpracovaných objednávek do formátu, který akceptují přepravní služby. Program je vytvořen v programovacím jazyce Java. Veškerá databáze je vytvořena v MySQL a ke komunikaci mezi Javou a MySQL slouží ovladač JDBC.

Klíčová slova

Java, MySQL, JDBC, export, zásilkový obchod

Title

Mail order sales

Annotation

The aim of this work was to develop an office program for companies engaged in the mail order business. The main output of the program is export the processed orders into a format, that transportation services accept. The program is developed in the programming language Java. All database is created in MySQL and for communication between Java and MySQL is used JDBC driver.

Keywords

Java, MySQL, JDBC, export, mail order shop

Obsah

Seznam zkratek

API	Application Programming Interface
JDBC	Java Database Connectivity
JVM	Java Virtual Machine
JRE	Java Runtime Environment
JDK	Java Development Kit
OOP	Objektově orientované programování
AWT	Abstract Window Toolkit
XML	Extensible Markup Language
SAX	Simple API for XML
DOM	Domain Object Model
PDF	Portable Document Format
CSV	Comma Separated Values
GUI	Graphic User Interface
GPL	General Public Licence

Seznam obrázků

Seznam tabulek

Úvod

Cílem této práce je vytvoření kancelářského programu sloužícího převážně k práci s objednávkami a hlavně k jejich následnému exportu a odeslání přepravní firmě. Většina přepravců, jako Česká Pošta nebo PPL, k importu do jejich systémů používají formát CSV a právě proto je použit i v této práci. Program je vytvořen v programovacím jazyce Java a jedná se tedy o desktopovou a multiplatformní aplikaci.

Většina firem používá tuto funkcionalitu v rámci svých internetových stránek tedy přes webové rozhraní. Výhodou tohoto řešení je možnost přístupu prakticky odkudkoli a jednotná databáze objednávek, kde po nákupu zboží v eshopu může být automaticky zpracována. Nevýhodou může ale být odlišné fungování a zobrazení různými webovými prohlížeči. Pro firmy, které však eshop nemají, nebo většina přijatých objednávek není v elektronické podobě, je desktopové řešení výhodnější.

U každé kancelářské aplikace je třeba, aby někde uchovávala data. Jednou z možností je ukládání dat do speciálních binárních souborů nebo souborů obsahujících prostý text. Tato řešení však nejsou příliš efektivní a je tedy vhodnější k tomu využít databázi, jako je například Oracle nebo MySQL. Tyto databázové servery jsou nejpoužívanější. Z tohoto důvodu a hlavně bezplatné GPL licence byl vybrán server MySQL. Ten může být jak uložen na lokálním zařízení, tak na vzdáleném serveru, kam se aplikace připojí.

K propojení aplikace vytvořené v programovacím jazyce Java s vybraným databázovým serverem je použit ovladač JDBC. V této práci jsou vysvětleny různé typy těchto ovladačů a postup k jeho zprovoznění.

1 Programovací jazyk Java

Java [1] je objektově orientovaný programovací jazyk vytvořený firmou Sun Microsystems, nyní dceřinou společností Oracle Corporation, v roce 1991. Jazyk se původně jmenoval „Oak“ a název byl změněn na „Java“ v roce 1995. Java vychází z programovacích jazyků C a C++, tím pádem si je syntaxe těchto jazyků velice podobná. Díky své přenositelnosti se jedná o velmi oblíbený programovací jazyk.

Přenositelnost zajišťuje JVM. Na každém zařízení, kde je třeba spustit program psaný v jazyce Java, je tedy JVM nezbytnou součástí, jelikož překládá tzv. „bajtkód“ na kód strojový. Pokud je žádoucí aplikace pouze spouštět, nainstaluje se nástroj JRE. Pokud je třeba aplikace i vyvíjet, instaluje se nástroj JDK, který obsahuje všechny knihovny, které může programátor využít. Aktuální verze obou nástrojů byla v době psaní práce 8u5 (1).

1.1 Rozhraní

V jazyce Java existuje speciální typ třídy označené klíčovým slovem *Interface* neboli rozhraní. To slouží jako jakási šablona předepisující hlavičky všech metod, které je třeba doprogramovat třídou, která dané rozhraní implementuje. Pokud je v aplikaci použito rozhraní, vytváří se jeho objekt, který se nainicializuje třídou, která jej implementuje. `IBanka banka = new Banka ();` je příklad použití, kde *IBanka* je rozhraní a *Banka* je třída, která toto rozhraní implementuje.

Programování přes rozhraní je v jazyce Java velmi používaná programovací technika. Slouží ke skrývání implementace, čímž zvyšuje bezpečnost. Dále dovoluje, aby stejnou metodu implementovalo i více tříd různým způsobem, což je vhodné např. pokud je třeba, aby dva různé objekty měly metodu se stejným názvem, ale každá obsahovala jinou funkcionalitu. Další výhodou je možnost např. při aktualizacích změnit třídu implementující dané rozhraní za jinou bez nutnosti úpravy jiných částí kódu.

1.2 Knihovna Swing

Knihovna Swing [2], [8] je velmi rozsáhlá, obsahuje všechny grafické komponenty a stará se o jejich zobrazování. V minulosti nahradil Swing knihovnu AWT, která byla zastaralá a byla závislá na platformě, čímž porušovala jeden ze základních principů, na kterých je jazyk Java postaven. Některé třídy z knihovny Swing jsou však odvozené od AWT, takže AWT nezaniklo úplně.

Každá komponenta, která je použita při tvorbě programu s grafickým uživatelským prostředím, je třídou knihovny Swing. Další důležitou věcí, kterou všechny komponenty obsahují, jsou posluchači (listenery). Ty slouží k zachycení různých událostí např. pohyb myši nebo stisknutí klávesy. Díky těmto speciálním instancím může program dynamicky reagovat na aktivitu uživatele. Nyní budou zmíněny následující komponenty, které jsou nejčastěji použity v této práci.

1.2.1 JLable

`JLable` je základní komponentou pro zobrazování textu, umožňuje ale i zobrazit obrázek nebo obojí. Dále lze nastavit zarovnání textu v rámci jeho oblasti nebo typ písma. Základní metody pro získání a nastavení obsahu jsou: `String getText()` a `void setText(String text)`.

1.2.2 JTextField

Objekty třídy `JTextField` jsou základní komponenty pro zadávání nebo editování jednořádkového textu. Základní metody pro získání a nastavení obsahu mají stejnou syntaxi jako v případě `JLable` a to: `String getText()` a `void setText(String text)`. U této komponenty se často používají posluchače, např. pro vyhledávání dle zadaného textu je nutné, aby systém reagoval po každém stisknutí klávesy. Pro tento případ lze využít listener `keyReleased`. Speciálním případem `JTextField` je `JPasswordField`, který se používá tam, kde je třeba skrýt znaky např. hesla a nahradit je znaky hvězdičky.

1.2.3 JButton

`JButton` je další z často používaných prvků v grafickém uživatelském prostředí. Jedná se o standartní tlačítko, které po stisknutí může vyvolat akci, pokud je použit příslušný posluchač s názvem *actionPerformed*. Kromě prostého textu lze vybrat i ikonu, kterou bude používat při zobrazení. Lze tak tedy jednoduše vytvořit vlastní vzhled aplikace.

1.2.4 JTable

Komponenta `JTable` slouží k zobrazování a editaci tabulkových dat. Při jejím vložení do grafického návrhu se automaticky přidává do kontejneru `JscrollPane`, který zajistí posunutí oblasti, pokud se množství řádků nebo sloupců nevejde do nastavené velikosti. Pro práci s jednotlivými buňkami se používá rozhraní `TableModel`. Toto rozhraní implementují dvě třídy `AbstractTableModel` a `DefaultTableModel`.

V této práci byl použit `DefaultTableModel` z důvodu, že není třeba doimplementovat další metody jako je to v případě `AbstractTableModel`. Metody nutné k implementaci jsou:

```
public int getRowCount();
public int getColumnCount();
public Object getValueAt(int row, int column);
```

Vzhledem k použití tabulek v této práci pouze k statickému zobrazení dat, není přítěží struktura `DefaultTableModel`, která používá vektor vektorů k ukládání dat. U `AbstractTableModel` lze vybrat, v jaké datové struktuře má data ukládat, což může být výhodou.

2 Databázové systémy

V dnešním světě jsou databáze [3] hojně používaným nástrojem. Databáze pracují na pozadí všemožných úkonů, které během dne provádíme. Může to být od nákupu v obchodě, kde je zboží vedeno v databázi a při placení je automaticky rozpoznána cena, až po přihlášení k emailu, kde se porovnají zadané údaje s údaji právě v databázi. Databázi si tedy lze představit jako soubor dat sloužící k ukládání informací o reálném světě.

2.1 Jazyk SQL

SQL je zkratka anglických slov Structured Query Language. Český překlad je strukturovaný dotazovací jazyk. Tento jazyk vznikl již v 70. letech minulého století, tehdy ještě pod názvem *SEQEL*. Od té doby však ještě zaznamenal mnoho úprav. Tou poslední zásadní byla úprava z roku 1992, kdy byla vytvořena verze pod názvem SQL92, která se stala standardem a přetrvává dodnes.

Jazyk SQL není typickým programovacím jazykem. Je zástupcem tzv. deklarativních programovacích jazyků. To znamená, že námi psaný kód není přímo použit pro předpis chodu programu, ale je vkládán do jiného programovacího jazyka, který je již procedurální. Příkazy SQL se tedy nepodobají známým programovacím jazykům. Spíše připomínají anglicky psanou větu, což byl jeho cíl. Díky tomu je použití pro koncové uživatele velice přívětivé.

Tento jazyk se skládá z několika částí určených pro různé druhy uživatelů, jako jsou administrátoři, programátoři, návrháři databázových systémů nebo koncoví uživatelé. Tyto části jsou následující:

- **Data Definition Language** (DDL) – Tato oblast se zabývá správou a tvorbou databázových objektů, jako jsou např. tabulky. Základní příkazy používané touto částí jsou např. CREATE, ALTER nebo DROP.
- **Storage Definition Language** (SDL) – Tato část definuje způsob, jakým se mají data fyzicky ukládat.
- **View Definition Language** (VDL) – VDL slouží k vytváření pohledů. Pohled může být v podstatě virtuální tabulka složená z různých jiných nebo naopak pouze z části jedné tabulky. Pohledy se používají, pokud není žádoucí přiřadit uživateli přístup k celé tabulce, ale přesto je nutné, aby některá data mohl přečíst. Základní příkaz pro vytvoření pohledu je `CREATE VIEW názevPohledu AS SELECT …`
- **Data Manipulation Language** (DML) – Tato oblast jazyka SQL slouží k manipulaci s daty. Obsahuje nejznámější příkazy, jako jsou SELECT, INSERT, UDATE nebo DELETE. DML příkazy jsou nejpoužívanější koncovými uživateli a právě programátory aplikací využívajících databáze.

2.2 Typy databází

Pokud je řeč o databázích [7], většina lidí si představí typické zástupce jako MySQL, Oracle, PostgreSQL nebo MS SQL. Všechny tyto databázové systémy jsou však zástupci relačních databází. V následujících kapitolách bych tedy rád rozebral i další druhy databází.

2.2.1 Relační databáze

Základním pojmem spojovaným s těmito databázemi je relace. Relaci si lze zjednodušeně představit jako vztah mezi tabulkami nebo mezi entitami v jedné tabulce. Dalším základním prvkem tohoto typu databáze je *tabulka*. Ta je složená z řádků a sloupců a má vždy pevně stanovený název. Sloupce slouží jako obraz jednotlivých atributů dané entity a mají předem daný datový typ. Řádky poté představují fyzické hodnoty těchto atributů.

Pro efektivní práci s tabulkou je nutné, aby každý záznam (neboli řádek) obsahoval nějaký unikátní identifikátor, dle kterého lze jednoznačně rozlišit dva záznamy. Takovýto identifikátor se nazývá *primární klíč*. Primární klíč nemusí obsahovat pouze jeden zvolený atribut. Lze jej tudíž vytvořit i z více atributů zároveň. Výhoda je v možnosti opakování se části z nich, jelikož unikátní musejí být pouze jako celek. Posledním pojmem, který v souvislosti s relačními databázemi uvedu, je *cizí klíč*. Cizí klíč je hodnota ve sloupci odkazující na primární klíč jiné tabulky. Tímto způsobem lze realizovat vazby mezi jednotlivými tabulkami.

Relační databáze jsou dostačující pro kancelářské aplikace, vzhledem k možnosti ukládání velkého množství dat ve formě tabulek. Tento typ databází však není vhodný pro ukládání objektů, jelikož je nutné rozložit je na dílčí části, které se ukládají zvlášť. Většina dnes používaných relačních databází používá jazyk SQL. V současné době se však jedná o nejpoužívanější řešení.

2.2.2 Objektově orientované databáze

Pro použití v objektovaně orientovaných jazycích, vznikly i objektově orientované databáze. Základem takové databáze již není tabulka ale rovnou celý objekt. Všechny objekty mají svoje vlastnosti, které lze považovat za sloupce u relačních databází. Dále poté obsahují metody, které slouží k manipulaci právě s hodnotami.

Jelikož jsou data ukládána jako objekty, nabízí toto řešení možnost použití dědičnosti, zapouzdření i polymorfismu. V jistých ohledech může být toto řešení mnohem efektivnější než je tomu u relačních databází a proto je dobré myslet i na tuto alternativu. Objektově orientovanými databázemi jsou například CA-Ingres, ODB II, UniSQL a další.

2.2.3 Temporální databáze

Tyto databáze jsou v podstatě rozšířením předchozích netemporálních databází o aspekt času. V databázi lze použít například systému časových razítek, sloužících k rozlišení událostí současných a minulých. Tyto databáze zjednodušeně řečeno lépe pracují s verzováním, což je žádoucí například pokud chceme zjistit plat zaměstnance v různých

časových obdobích. K těmto databázím se používá i speciální strukturovaný jazyk vycházející z SQL, přinášející ovšem speciální vlastnosti jako například klauzuli *WHEN*. Příkladem takového jazyka může být TSQL. Využití mají tyto databáze především v bankovnictví nebo v rezervačních systémech.

2.3 Databázový systém MySQL

MySQL je relační databázový systém vyvinutý švédskou firmou MySQL AB, kterou koupila firma Sun Microsystems nyní již dceřiná společnost firmy Oracle. MySQL je k dispozici jak zdarma s licencí GPL, tak jako placený produkt. Právě díky bezplatnosti tohoto produktu je tento systém tak populární a rozšířený. Pro webové servery je často používán v kombinaci s jazykem PHP a webovým serverem Apach. Tento systém lze provozovat na různých operačních systémech jako například MS Windows nebo Linux.

MySQL je velice rychlý systém. Jako daň tomu byla v minulosti podstoupena nepřítomnost různých funkcí jako například triggery nebo uložené procedury. Od verze 5.0 však již tyto vlastnosti plně podporuje.

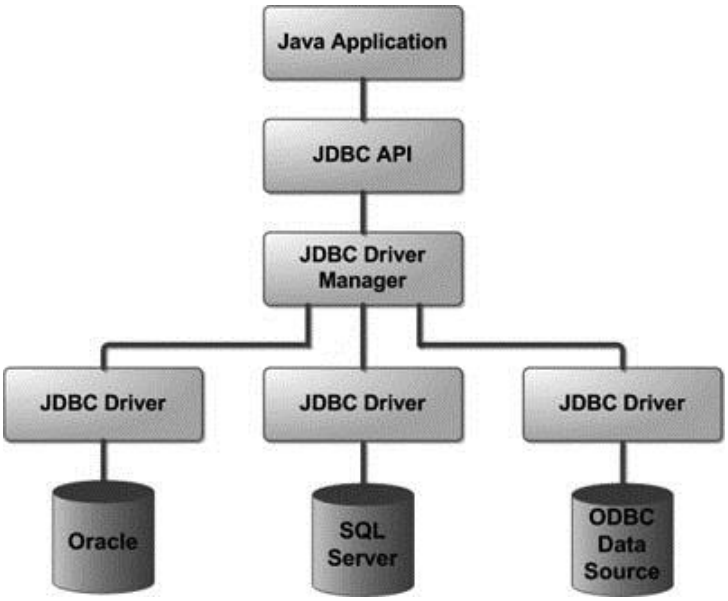
3 Rozhraní JDBC

JDBC [4], [6] neboli Java Database Connectivity je technologie určená pro práci s relačními databázemi prostřednictvím programovacího jazyka Java. Dovoluje tedy aplikaci provádět různé databázové operace. K tomu slouží API přímo integrované do Javy již od verze JDK 1.1 vytvořené v roce 1997. Základem této technologie je využití JDBC ovladače určeným pro přístup do různých databázových serverů jako například MySQL nebo PostgreSQL.

3.1 Architektura JDBC

JDBC je tedy něco jako prostředník mezi jazykem Java a databázovým serverem. Ze strany Javy je důležitou součástí JDBC jeho API, které poskytuje všechny potřebné metody pro práci s databázovými objekty a zároveň používá správce ovladačů (**driver manager**). Na straně druhé je to ovladač (**driver**) starající se o připojení k databázi.

Správce ovladačů tedy zajišťuje, že se použije vždy správný ovladač k přístupu do různých databázových systémů. Zároveň je schopný používat více ovladačů připojených k více databázím, čímž dovolí jediné aplikaci pracovat paralelně s více databázemi. Na Obrázek 1 je celá architektura znázorněna.



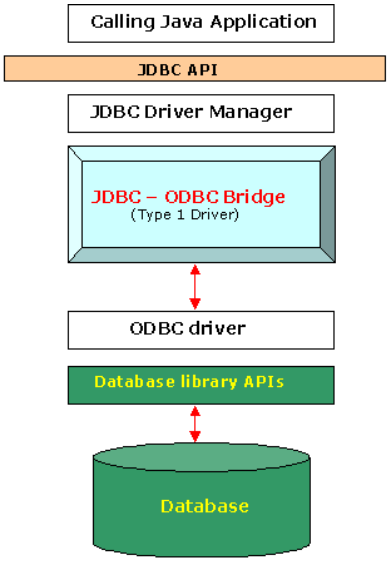
Obrázek 1 – Architektura JDBC [6]

3.1.1 Typy ovladačů

Každý z ovladačů musí implementovat rozhraní Driver, obsažené v balíčku `java.sql`. Nejdůležitější metodou, kterou musí třída ovladače obsahovat je metoda `Connection connect(String url, Properties info)`. Ta vrací na základě adresy `url` instanci třídy `Connection`, která zastupuje spojení s danou databází.

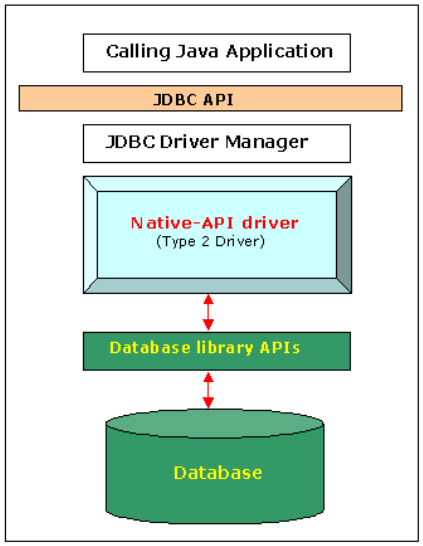
Ovladače se dělí do čtyř různých kategorií dle způsobu přístupu k databázi.

- **Typ 1** – Tento typ využívá ODBC (Open Database Conectivity) ovladač, k němuž přistupuje pomocí „JDBC-ODBC bridge“. Používá se v případě, kde není k dispozici JDBC ovladač ale ODBC ano. Jeho nevýhodami jsou ovšem nutnost instalace patřičného ODBC ovladače pro každou databázi, což může být složité vzhledem k specifickým postupům pro každou databázi, a nižší výkon vzhledem k ostatním typům. Schéma je zobrazeno na Obrázek 2.



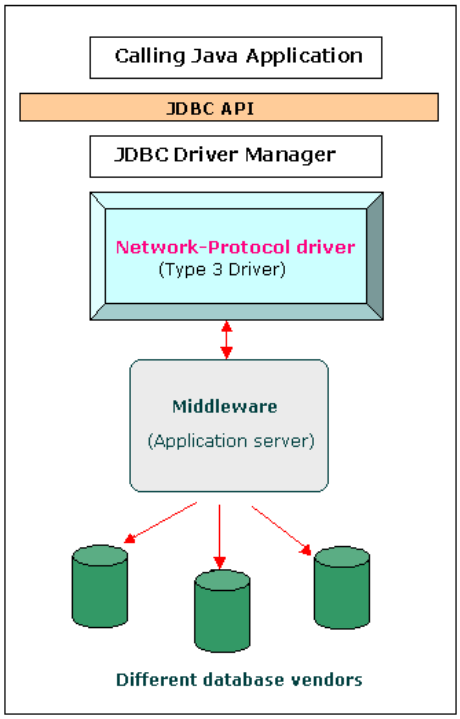
Obrázek 2 – Ovladač JDBC typ 1 [5]

- **Typ 2** – Označován jako „Native-API driver“ funguje podobně jako předchozí typ. Místo „JDBC-ODBC bridge“ používá specifický ovladač, který komunikuje přímo s API databázové knihovny. Je ovšem závislý na klientské knihovně databázového serveru, která nemusí být vždy k dispozici. Výhodou je o něco vyšší rychlost než u předchozího typu. Schéma je zobrazeno na Obrázek 3.



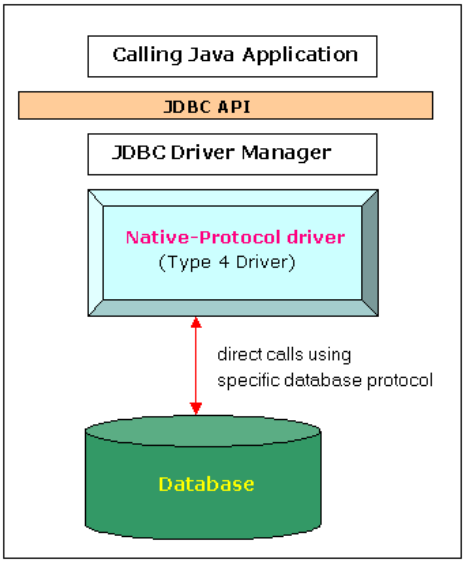
Obrázek 3 – Ovladač JDBC typ 2 [5]

- **Typ 3** – Zde již není třeba žádný nativní kód. Ovladač JDBC komunikuje s centrálním serverem pomocí síťového protokolu a server sám zajišťuje komunikaci s připojenými databázemi tím, že převede klientský protokol na databázový protokol, kterému již daná databáze rozumí. Výhodou tohoto řešení je možnost existence více heterogenních databází, jelikož s nimi komunikuje pouze centrální server. Díky tomu programátor pracuje s jednou databází, což je velice efektivní. Další výhodou je možnost komunikace i jiných klientů než pouze z jazyka Java díky použití nezávislého síťového protokolu. Toto řešení se používá převážně u rozsáhlých systémů. Schéma je zobrazeno na Obrázek 4.



Obrázek 4 – Ovladač JDBC typ 3 [5]

- **Typ 4** – Tento typ ovladače je kompletně napsán v jazyce Java. Ke komunikaci s databázovým systémem používá takový protokol, kterému daný systém přímo rozumí. Díky tomu, že je psán v jazyce Java, je platformě nezávislý a pro svou funkci již nepotřebuje žádné speciální nástroje. Nevýhodou je ovšem závislost řadiče na dané databázi. Pro každou jinou je tedy třeba její speciální ovladač. Schéma je zobrazeno na Obrázek 5.



Obrázek 5 – Ovladač JDBC typ 4 [5]

3.2 Připojení k databázi

Tato kapitola se bude věnovat samotnému připojení Java aplikace k databázovému serveru. Základem je JDBC ovladač, v tomto případě typu 4, který je většinou poskytován výrobcem databáze. Před jakýmkoli dalším použitím je třeba v aplikaci ovladač zaregistrovat příkazem `Class.forName("com.mysql.jdbc.Driver")`[1]. Volání této metody může doprovázet vyvolání výjimky `ClassNotFoundException`, a proto je ho třeba umístit do bloku `try{}`.

Jako další přichází na řadu třída `DriverManager` a její metoda `getConnection(String URL)`, která vrací vytvořené spojení rozhraním `Connection`. S tímto rozhraním může již programátor provádět veškeré operace nad databází. URL a název ovladače jsou závislé na databázi, k níž je třeba se připojit. V Tabulka 1 jsou zobrazeny jejich příklady pro vybrané databáze. Zvýrazněné části v URL formátu jsou statické a mění se tedy pouze zbývající části.

Tabulka 1 – Příklad URL adres a názvů ovladače pro připojení k databázi [6]

RDBMS	JDBC název ovladače	URL formát
MySQL	com.mysql.jdbc.Driver	**jdbc:mysql**://hostname/ databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	**jdbc:oracle:thin:@**hostname:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	**jdbc:db2**:hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	**jdbc:sybase:Tds**:hostname:port Number/databaseName

[1] V příkladu je uveden ovladač databáze MySQL.

3.3 Metody pro práci s JDBC

Po získání objektu typu `Connection` [8] nic nebrání práci s databázovými objekty. Rozhraní `Connection` obsahuje velké množství metod, já ovšem zmíním jen ty nejdůležitější a věnovat se budu hlavně jejim návratovým typům. Zmíněné metody jsou následující:

- `Statement createStatement()`,
- `CallableStatement prepareCall(String sql)`,
- `PreparedStatement prepareStatement(String sql)`,
- `DatabaseMetaData getMetaData()`.

3.3.1 Rozhraní Statement

Rozhraní `Statement` vrací metoda, volaná prostřednictvím rozhraní `Connection`, `createStatement()`. Jedná se o základní rozhraní, s kterým lze jednoduše provádět jak příkazy typu select, tak DML nebo DDL příkazy. Slouží především k provádění statických SQL příkazů za chodu programu.

Pro provedení příkazu typu select je nejvhodnější použít metodu této třídy `executeQuery(String sql)`. Jako parametr `sql` se používá text obsahující SQL příkaz. Výsledek příkazu je vrácen v objektu typu `ResultSet`, jemuž se budu věnovat později.

Co se příkazů typu insert, update nebo delete týče, lze pro jejich provedení použít metodu `executeUpdate(String sql)`. Tato metoda vrací číslo typu integer oznamující, kolik řádků bylo provedeným příkazem ovlivněno. Metoda dokáže provést i DDL příkazy typu create table atd. Po jejich provedení vrací metoda nulu. Příklad použití vypadá následovně:

```
Statement stmt = conn.createStatement();
ResultSet rset = stmt.executeQuery("select * from objednavky")
int pocetZaznamu = stmt.executeUpdate("delete from objednavky")
```

3.3.2 Rozhraní PreparedStatement

Toto rozhraní je rozšířením standartního rozhraní `Statement`. Slouží k efektivnějšímu zpracování podobných příkazů lišícími se pouze použitými parametry. V příkazu se místo hodnot použije zástupný znak „**?**“, za který se posléze dosadí hodnoty. V prvním kroku proběhne kontrola příkazu, zda je validní a při dalších použitích ke kontrole již nedochází, čímž výrazně urychluje zpracování. Proto je vhodné jej používat například při častém vkládání do jedné tabulky.

Pro nahrazení zástupného znaku požadovanou hodnotou slouží metody typu `setXXX()`. Jsou připraveny prakticky pro každý základní datový typ, který Java obsahuje. Po doplnění

všech požadovaných hodnot těmito metodami může dojít k potvrzení transakce metodou `executeUpdate()`. Následuje příklad použití tohoto rozhraní:

```
PreparedStatement stmt = conn.prepareStatement("delete from
     zakaznik where id = ? and jmeno = ?");
stmt.setInt(1, 137);
stmt.setString(1, Petr);
stmt.executeUpdate();
```

3.3.3 Rozhraní CallableStatement

Tato třída dále rozšiřuje rozhraní `PreparedStatement`. Slouží k volání procedur uložených v databázi. Funguje na podobném principu, jako tomu bylo u předchozího rozhraní. Liší se syntaxí příkazu a kromě vstupních parametrů (IN) dokáže pracovat i s parametry výstupními (OUT) nebo vstupně-výstupními (INOUT).

Příkaz se zadává ve formátu `{call názevProcedury (?,?)}`. Vstupní parametry se nastavují stejným způsobem jako je to v případě použití rozhraní `PreparedStatement` a to použitím metod `setXXX()`. Pro propojení výstupního parametru s lokální proměnou lze využít metod typu `getXXX()`. Výstupní parametr je však nutné nejprve zaregistrovat metodou `registerOutParameter(int parameterIndex, int sqlType)`, aby aplikace poznala, jaký datový typ má na daném indexu očekávat. Příklad může vypadat následovně:

```
CallableStatement cstmt = conn.prepareCall ({call
     mojeProcedura (?,?)});
cstmt.setInt(1, 10);
cstmt.registerOutParameter(2, java.sql.Types.VARCHAR);
cstmt.executeUpdate();
String vystupniHodnota = cstmt.getString(2);
```

3.3.4 Rozhraní ResultSet

Rozhraní `ResultSet` slouží k navrácení výsledků příkazu select již zmíněnou metodou `executeQuery()`. V datech se lze pohybovat pomocí kurzoru. K tomu slouží metoda `next()` vracející hodnotu typu `boolean`. Metoda vrací `true` v případě, že `ResultSet` obsahuje další data a `false` při přesunutí kurzoru na úplný konec.

Pro načtení dat z aktuální pozice kurzoru slouží opět metody typu `getXXX()`, podporující všechny základní datové typy. Pokud si uživatel není jistý datovým typem získávané hodnoty, lze použít univerzální metodu `getObject()`, jelikož každá třída v jazyce Java je potomkem právě třídy `Object`.

Všechny zmíněné metody typu `getXXX()` jsou ovšem přetížené. Jako parametr přijímají buď objekt typu `String`, nebo celé číslo. Parametr typu `String` lze použít v případě, kde si uživatel není jistý, jaké je pořadí kýženého sloupce a je si naopak jistý jeho názvem. Méně náročné je ovšem použití celočíselného parametru, který říká, jako kolikátý v pořadí se vybraný sloupec nachází. Názorná ukázka použití rozhraní `ResultSet` přímo z této práce je zobrazena v příloze B. Ukázka konverze datových typů mezi SQL a Javou je znázorněna tabulkou v příloze C.

3.3.5 Metadata

Metadata slouží k získávání informací o databázových objektech prostřednictvím JDBC. Pro práci s nimi existují dvě rozhraní a to `DatabaseMetaData` a `ResultSetMetaData`. `DatabaseMetaData` je přístupné skrze rozhraní `Connection`. Slouží především k získávání dílčích informací o připojené databázi. Pokud daná metody vrací více než jeden záznam, je jako návratový typ zvoleno rozhraní `ResultSet`. Nejčastěji používané metody, seřazeny abecedně, jsou následující:

- `getDatabaseProductName()` – metoda vracející název připojené databáze v objektu typu `String`.
- `getUserName()` – metoda sloužící k obdržení uživatelského jména přihlášeného uživatele opět jako datový typ `String`.
- `getURL()` – tato metoda vrátí URL adresu databáze.
- `getSchemas()` – tato metoda je již poněkud zajímavější. Vrací názvy všech použitých databázových schémat jako `ResultSet`.
- `getTables()` – poslední metoda se používá k získání názvů všech tabulek obsažených ve vybrané databázi.

Druhým typem metadat jsou metadata získaná prostřednictvím rozhraní `ResultSet`. Taková metadata obsahují převážně informace týkající se typů a vlastností sloupců obsažených v objektu `ResultSet`. Objekt typu `ResultSetMetaData` lze získat zavoláním metody nad objektem `ResultSet getMetaData()`. Opět zmíním několik základních metod pro práci s těmito metadaty. Příklad použití metadat pro tvorbu hlavičky tabulky, do níž jsou následně vložena data, je zobrazen v příloze B.

- `getColumnCount()` – jak již název napovídá, metoda vrátí celkový počet sloupců obsažených v objektu `ResultSet`. Vhodné využití této metody může být například jako limitující parametr v cyklu, pokud je třeba postupně projít všechny sloupce výsledku.
- `getColumnName()` – tato metoda vrátí název sloupce na základě jeho pořadí. Tato metoda ovšem není vhodná, pokud je třeba získat název přiřazený danému sloupci jako jeho alias. Pro tento případ je vhodnější metoda `getColumnLabel()`.
- `getColumnType()` – poslední metoda vrací datový typ vybraného sloupce jako číslo typu `int`. Název datového typu lze poté zjistit z třídy `java.sql.Types`.

4 Export do souboru

Pro export do souboru lze v dnešní době použít mnoho formátů. U výběru vhodného formátu závisí především na způsobu dalšího zpracování (zda strojově nebo ručně) a na jeho dostupnosti a otevřenosti.

Dále budou zmíněny následující formáty:

- XML,
- PDF,
- CSV,
- binární soubory.

4.1 XML

XML [9] je obecný značkovací jazyk vytvořený konsorciem W3C a vychází z programovacího jazyka SGML stejně jako například HTML. Jeho výhodou je nezávislost na platformě. Lze jej tedy použít na všech operačních systémech. Pro výměnu dat je to velmi používaný formát z důvodu jeho univerzálnosti a podporou mnoha programovacími jazyky (v jazyce Java například parsery SAX a DOM). Další možnosti použití tohoto jazyka je možnost publikování dokumentů, kde slouží k přenášení obsahu. O vzhled se poté mohou starat například kaskádové styly.

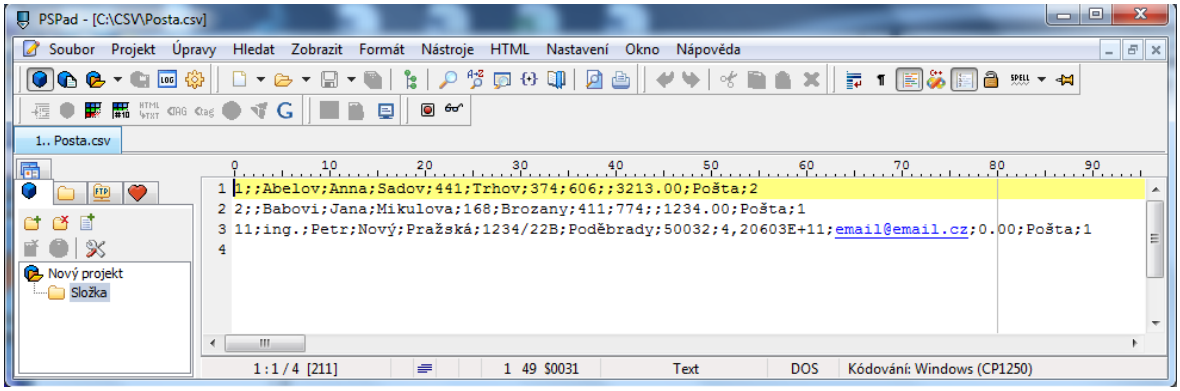
4.2 PDF

PDF [10] je formát vytvořený firmou Adobe v roce 1993 a slouží k ukládání především dokumentů. Výhodou oproti XML je zobrazení formátovaného textu s možností obsahu obrázků s garancí stejného zobrazení na všech zařízeních, nevýhodou poté složitější strojové zpracovávání. Další malou nevýhodou je nutnost instalace programu pro prohlížení PDF dokumentu. Většina je jich ovšem zdarma včetně jednoho z nejpoužívanějších, který se jmenuje Adobe Reader.

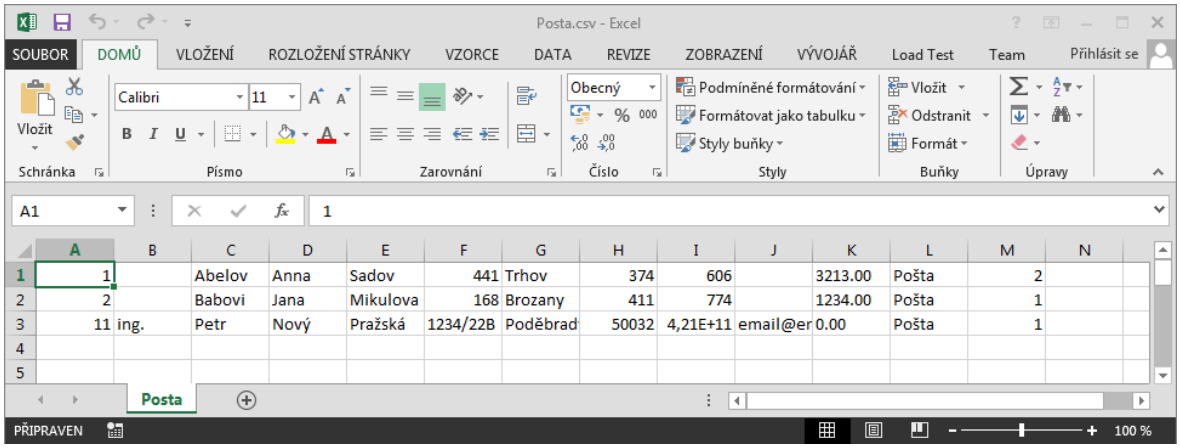
4.3 CSV

CSV [11] je formát určený především pro export tabulkových dat. Soubor je složen z řádků, v nichž je každý záznam uveden v uvozovkách („"“) a oddělen čárkou („**,**“), středníkem („**;**“), nebo tabulátorem. Formát je používán především díky své jednoduchosti a čitelnosti i bez speciálního programu a mnohem rychlejšímu zpracování dat ve srovnání s XML.

Pro tuto práci jsem zvolil právě tento formát z výše uvedených důvodů a dále z důvodu přímé podpory ze strany přepravních služeb, které pro import dat využívají výhradně tento formát. Jako oddělovač záznamů byl zvolen středník pro případ, že by v datech byla obsažena čárka (například jako oddělovač desetinných míst) a dokument tak byl přehlednější. Na Obrázek 6 je vidět jak CSV soubor interpretuje textový editor PSPad a na Obrázek 7 poté, jak jej zobrazuje program Microsoft Excel.



Obrázek 6 – Zobrazení CSV souboru programem PSPad



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	1		Abelov	Anna	Sadov	441	Trhov	374	606		3213.00	Pošta	2	
2	2		Babovi	Jana	Mikulova	168	Brozany	411	774		1234.00	Pošta	1	
3	11	ing.	Petr	Nový	Pražská	1234/22B	Poděbrad	50032	4,21E+11	email@er	0.00	Pošta	1	
4														
5														

Obrázek 7 – Zobrazení CSV souboru programem Microsoft Ecel

4.4 Binární soubory

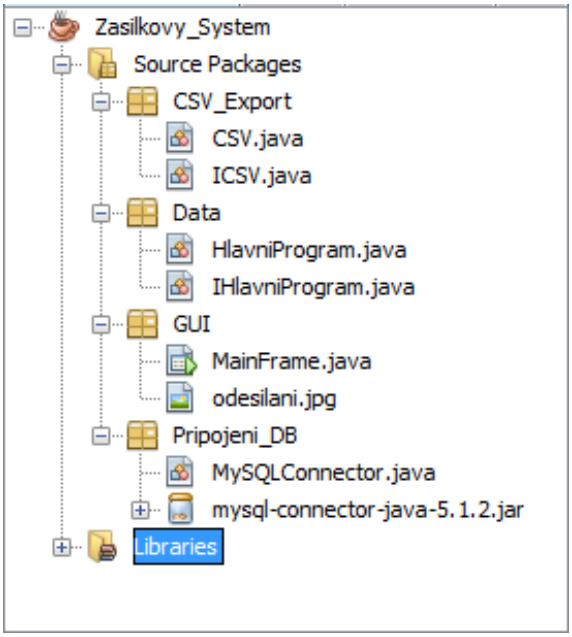
Binární soubory můžeme pro export využít pouze v případě, kdy exportujeme data pouze v rámci naší aplikace. Je totiž nutné znát přesnou strukturu souboru, aby bylo možné jeho korektní přečtení. Díky tomu je to velice bezpečný způsob exportování. Pokud je třeba přenášet data z programu mezi různými počítači (a není to řešeno jinak), lze využít této možnosti.

5 Implementace objednávkového systému

Tato aplikace by měla být určená především firmám, které se zabývají zásilkovým obchodem. Zároveň je vhodná pro takové společnosti, které pracují s objednávkami, které nejsou v digitální podobě a to díky možnosti tyto objednávky zadávat do databáze přímo v tomto programu a přehledným způsobem. Konkurencí tohoto programu by mohla být webová řešení obsažená přímo v e-shopu. Pokud je však třeba vykonávat tuto činnost na lokálním zařízení, žádná taková komerční řešení jsem neobjevil a je tedy vhodná tato aplikace.

5.1 Návrh tříd

Aplikace je implementována pomocí čtyř tříd, dvou rozhraní a obsahuje jeden obrázek jako hlavní logo aplikace. V balíčku CSV_Export se nachází rozhraní, které zajišťuje veškerou práci s tvorbou a používáním CSV soborů a dále třída, která jej implementuje. V dalším balíčku Data se nachází rozhraní `IHlavniProgram`, v němž jsou předpisy pro všechny stěžejní metody celého programu a implementující třída `HlavniProgram`. Následuje balíček `GUI` obsahující třídu s grafickým rozhraním a použité obrázky. Poslední balíček `Pripojeni_DB` obsahuje statickou třídu pro práci s databází a JDBC knihovnu. Stromová struktura souborů projektu je zobrazena na Obrázek 8.

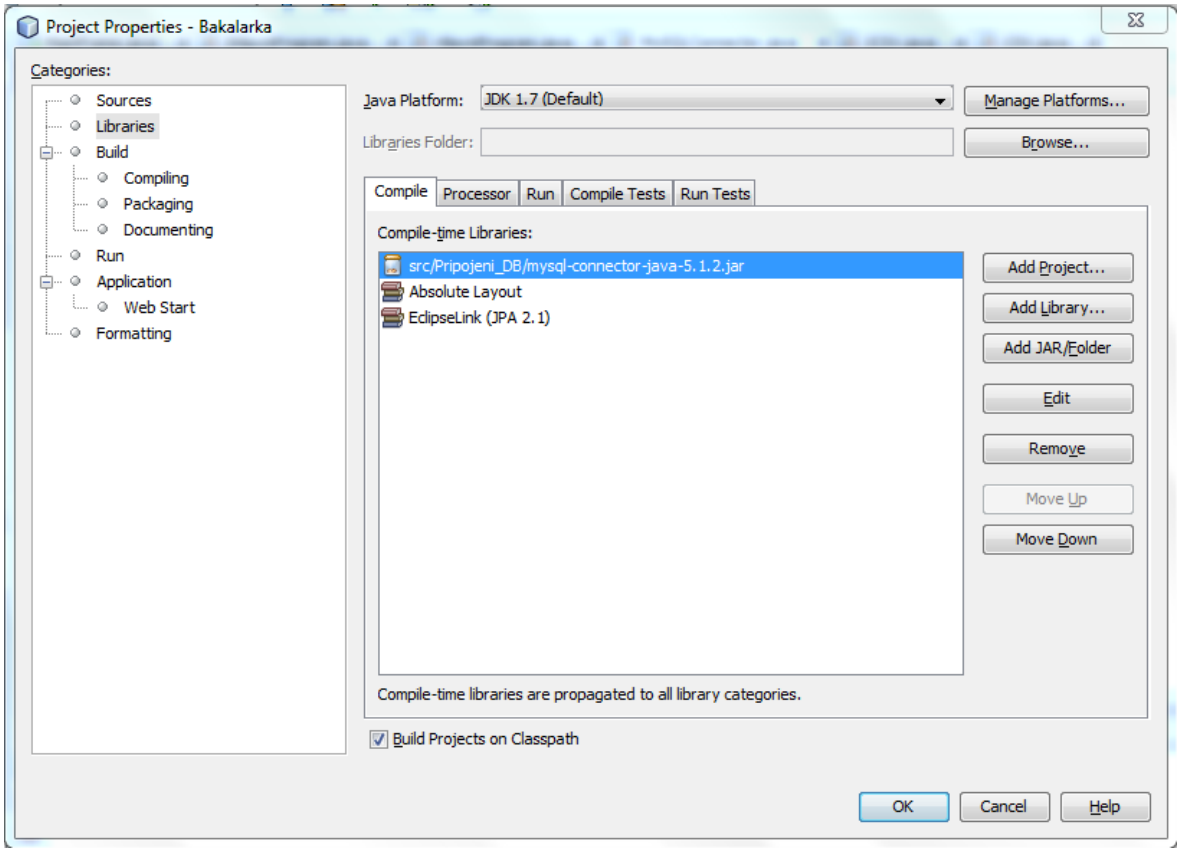


Obrázek 8 – Stromová struktura projektu

5.2 Použité nástroje

K tvorbě tohoto programu bylo použito vývojové prostředí NetBeans IDE ve verzi 7.3.1. Toto prostředí podporuje vše, co bylo pro tuto práci potřebné. Je to jak snadná tvorba GUI díky podpoře drag&drop při manipulaci s komponentami, tak výborná typová kontrola, která okamžitě reaguje na chyby a dokonce nabídne několik možných řešení problému.

Pro komunikaci s databázovým serverem MySQL je třeba řádně připojit knihovnu JDBC. V NetBeans je to snadná záležitost. Po stažení souboru s názvem *mysql-connector-java-5.1.2.jar*[2] se soubor zkopíruje do složky s projektem. Dalším krokem je již připojení knihovny ve vlastnostech projektu, v záložce *Libraries*, kde zvolíme *Add JAR/Folder* a vybereme cestu k souboru. Příklad takto připojené knihovny je zobrazen na Obrázek 9.



Obrázek 9 – Příklad připojené knihovny JDBC v NetBeans

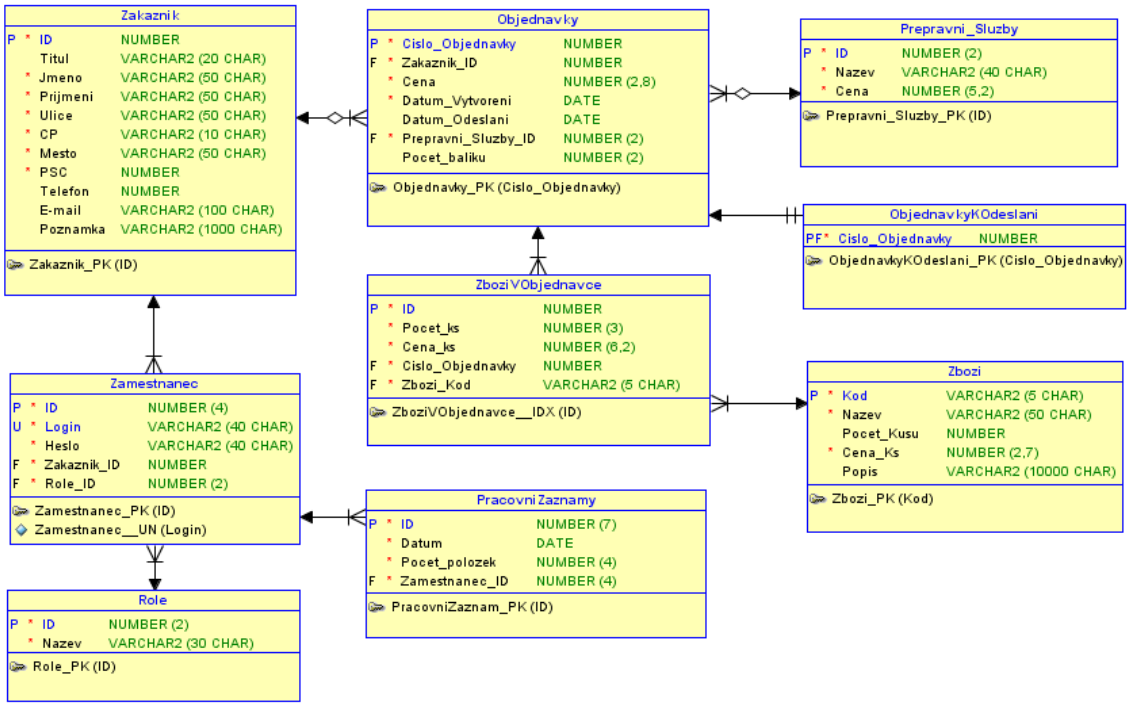
Další nástroj, který byl pro tuto práci využit je Data Modeler od společnosti Oracle, který sloužil k návrhu databáze a k tvorbě ER diagramu. Z něj exportovaná data sloužila k tvorbě databázových objektů v nástroji MySQL Workbench též od společnosti Oracle, který je určený pro správu databázového serveru MySQL. V tomto programu lze pracovat s databází jak s použitím SQL příkazů, tak pomocí grafického průvodce, který na pozadí vše převede do SQL a výsledek zobrazí ke schválení.

5.3 Struktura databáze

Databáze nezbytná pro tuto aplikaci je složena z devíti tabulek. U každé, kromě tabulky Zboží, kde je možné vytvořit vlastní alfa-numerický unikátní kód, je jako primární klíč zvolen číselný kód, který se po vložení nového záznamu automaticky přidá díky vlastnosti *AUTO_INCREMENT* sloupce ID. U každého sloupce tabulky, bez kterého vkládaný záznam

[2] Koncové číslo se mění v závislosti na pořadovém čísle vydání.

nemá smysl, je použita vlastnost *NOT NULL*. Seznam všech tabulek a relací je zobrazen na Obrázek 10.



Obrázek 10 – ER diagram

5.4 Export do CSV

Při výběru vhodného formátu pro export dat byl zvolen právě formát CSV. Bylo tomu tak jak z důvodu jednoduchosti implementace, tak z důvodu podpory tohoto formátu přepravními společnostmi.

Pro účely exportu do tohoto formátu bylo vytvořeno rozhraní obsahující potřebné metody a třída, která toto rozhraní implementuje. Nachází se zde metoda k otevření souboru, zápisu jedné položky do soboru, zápisu konce řádku, zavření souboru a popřípadě jeho smazání. Celý proces exportu závisí na vytvoření instance třídy `OutputStreamWriter` při otevření souboru a následně zápisu jednotlivých dat do takto vytvořeného výstupního proudu metodou `append(CharSequence csq)` a přidání oddělovače záznamů neboli znaku „**;**“ a pro ukončení řádku vložení zalamovacího znaku „**\n**“. Následuje příklad jednoduchého exportu dat do souboru export.csv.

```
soubor = new File ("C:\\export.csv");
OutputStreamWriter out= new OutputStreamWriter(new
      FileOutputStream(soubor, true), "CP1250");
out.append("Data která je třeba exportovat"+";");
out.append("\n");
out.close();
```

5.5 Funkce aplikace

Po spuštění aplikace se zobrazí jednoduchý dialog, který vyzve uživatele k přihlášení jeho uživatelským jménem a heslem. Pokud je zadáno existující přihlašovací jméno a k němu příslušné heslo, je uživatel vpuštěn do hlavní části programu. Pokud uživatel zadá chybné údaje, bude na to upozorněn zprávou vyvolanou statickou metodou třídy `JoptionPane`:

```
public static void showMessageDialog(Component parentComponent,
      Object message, String title,int messageType)
```

Tato metoda je často používaná k různým oznámením během chodu programu. Metodě se předává obsah zprávy argumentem `message`, titulek zobrazený v záhlaví okna argumentem `title` a nakonec typ zprávy položkou `messageType`. Jako typ zprávy lze zvolit informační, varovnou nebo chybovou zprávu doprovázenou příslušnou ikonou. Za `parentComponent` je vhodné dosadit tu komponentu, jejíž obsah vyvolal tuto metodu (nejčastěji hlavní okno) a tím je docíleno zobrazení okna zprávy vystředěné vzhledem k zadané komponentě.

Hlavní okno aplikace je tvořeno převážně komponentou `JTabbedPane`, která v současnosti obsahuje osm záložek. V každé z nich se nachází obsah relevantní k jedné činnosti, kterou je třeba ve spojení s danou záložkou provádět. Jsou zde tedy jednoduše odděleny pracovní úkony. Případné rozšíření aplikace by bylo tedy velice jednoduché a spočívalo by v přidání další záložky, která by obsahovala novou funkcionalitu.

5.5.1 Odesílání objednávek

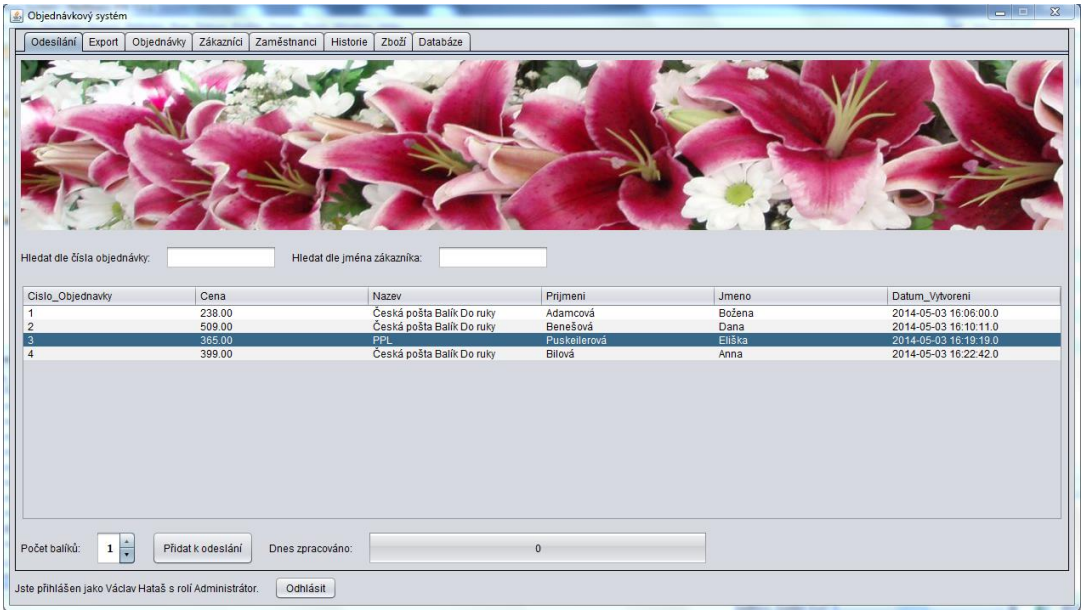
První ze záložek se jmenuje *Odesílání*. Je to jedna z nejhlavnějších částí aplikace, která předchází samotnému exportu. První z činností, která se provede, je vyhledání objednávky, která je připravena k odeslání. Vyhledávání lze ovlivnit jak zadáním části čísla objednávky, tak částí jména zákazníka do patřičných komponent `JTextArea`.

Výsledky vyhledávání jsou poté zobrazeny do tabulky, která svojí velikostí dominuje celému oknu. Je to z důvodu přehlednosti, kde dovoluje zobrazit všechny důležité údaje objednávky a zároveň díky tomu není nutné vypisovat celé číslo objednávky, jelikož lze v tabulce vybrat z více možných výsledků. V tabulce se ovšem nezobrazují všechny objednávky, které by vyhovovaly zadaným kritériím. Opět z důvodu přehlednosti jsou z výsledků vyhledávání odebrány ty objednávky, které již byly odeslány, nebo právě čekají na export. Díky tomu zde lze najít pouze takové objednávky, s kterými je třeba dále pracovat.

Ve spodní části okna se nachází `JSpinner`, který slouží k nastavení počtu balíků, do kterých byla objednávka rozdělena. Pokud se tedy objednávka fyzicky nevejde do jednoho balíku, nastaví se správný počet a díky tomu bude systém přepravce vědět, že danému zákazníkovi má doručit ne jeden ale více balíků. Další komponentou je `JProgressBar`, který slouží

jako informativní ukazatel počtu zpracovaných objednávek daným zaměstnancem, což může sloužit např. sledování dodržení denní kvóty stanovené zaměstnavatelem.

Poslední komponentou je tlačítko s popisem *Přidat k odeslání*, jehož stisknutím se zanese vybraný záznam z tabulky s vyhledanými objednávkami do pomocné databázové tabulky `ObjednávkyKOdeslani` a změní datum odeslání na aktuální, čímž danou objednávku vyřadí z vyhledávání v rámci této záložky. Tímto krokem je objednávka připravena k exportu a lze postup opakovat s dalšími objednávkami nebo se přesunout na záložku *Export*. Příklad otevřené záložky *Odesílání* je zobrazen na Obrázek 11.



Obrázek 11 – Ukázka záložky Odesílání

5.5.2 Export objednávek

Tato část programu slouží výhradně k exportu dat uložených v databázové tabulce `ObjednávkyKOdeslani` do souborů ve formátu CSV. Po kliknutí na tlačítko *Export do CSV* aplikace automaticky rozpozná, o jakou se jedná přepravní službu a pokud soubor s názvem dané přepravní služby existuje, přidá všechny zobrazené položky na konec souboru. Pokud soubor neexistuje, automaticky jej vytvoří.

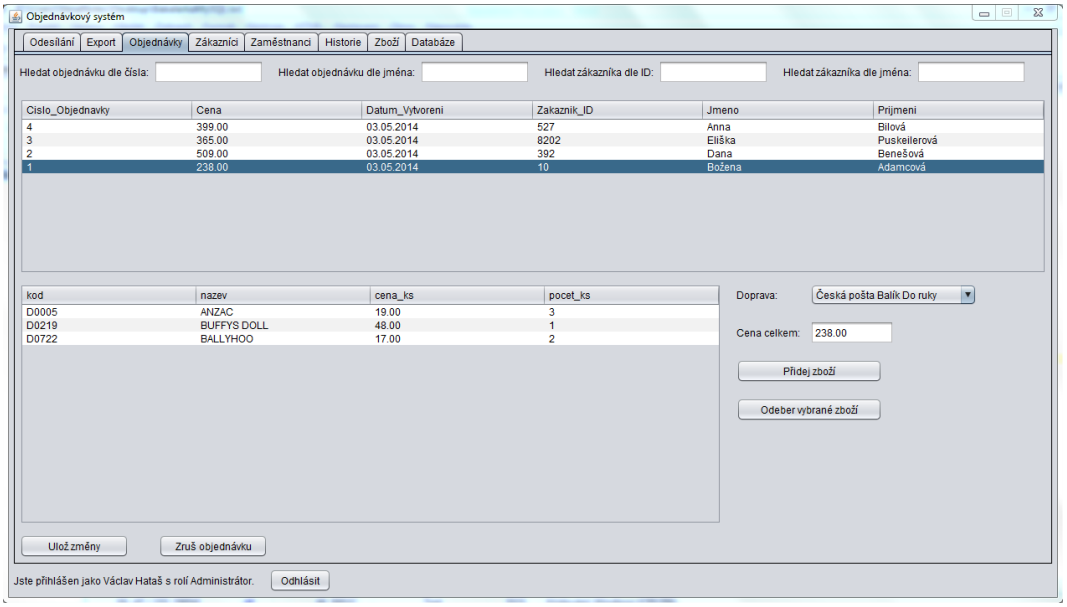
Poté následuje činnost nezávislá na tomto programu. Je nutné takto vytvořený soubor odeslat přepravní společnosti, která po jeho zpracování poskytne soubor ve formátu PDF, v němž se nacházejí etikety, které se po vytisknutí nalepí na balík. Následně je nutné smazat obsah CSV souboru, který byl právě odeslán, aby byl připraven k dalšímu použití. Smazání probíhá výběrem přepravní společnosti, jejíž soubor je třeba vymazat, z rozbalovacího seznamu a stisknout tlačítko *Smaž CSV*.

Další možností, kterou toto okno nabízí je odebrání objednávky z tohoto seznamu. Po výběru dané objednávky stačí kliknout na tlačítko *Odeber vybrané* a objednávka se vrátí do seznamu

objednávek k odeslání. Této možnosti lze využít například, pokud se zde nedopatřením objeví objednávka, která zatím není připravena.

5.5.3 Editace objednávek

Program je schopen pracovat i s objednávkami. K tomu je přizpůsoben obsah záložky *Objednávky*. Pokud tedy chce uživatel vytvořit novou objednávku nebo stávající upravit či smazat, použije právě tuto záložku. Příklad zobrazení této záložky je ukázán na Obrázek 12.



Obrázek 12 – Ukázka záložky Objednávky

Po otevření této záložky se zobrazí okno, kde dominují dvě velké tabulky. Tato velikost byla zvolena opět z důvodu přehlednosti. V první tabulce se zobrazují výsledky vyhledávání. Pokud je třeba vyhledat objednávky za účelem úpravy nebo smazání používají se k tomu textová pole pro hledání objednávky dle čísla nebo dle jména zákazníka. Pokud ovšem uživatel chce objednávku vytvořit, musí nejprve vyhledat zákazníka, pro nějž má být objednávka určená. Vyhledání provede postupným psaním do textových polí pro hledání zákazníka dle id nebo dle jména.

Po vybrání zvolené objednávky se vyplní její další obsah do příslušných komponent. Především je to obsažené zboží, které se zobrazí v druhé, již zmiňované tabulce. Pokud se však vytváří nová objednávka, po vybrání zákazníka se nevyplní nic a je tedy nutné všechny informace doplnit uživatelem. Mezi další položky, které lze ve všech případech upravovat patří rozbalovací seznam obsahující volbu přepravní služby a textové pole, kde se zobrazuje celková cena za objednávku. Cena se dopočítává automaticky z ceny zboží a vybrané dopravy. Pokud je ovšem z jistého důvodu třeba zadat jinou částku, je to možné přímým zapsáním do pole s cenou. Je ovšem nutné objednávku okamžitě uložit, respektive vytvořit, aby nedošlo k přepsání ceny např. změnou způsobu dopravy.

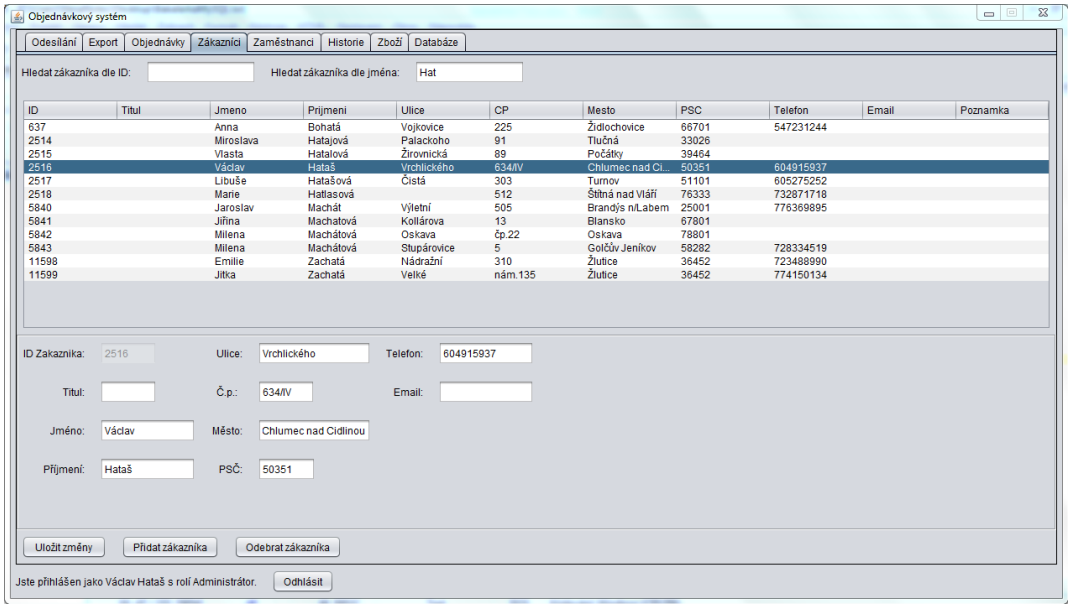
Pokud se vytváří objednávka nebo pokud se doplňuje stávající o novou položku zboží, stiskne se tlačítko *Přidat zboží*, čímž se otevře nové okno sloužící právě k přidávání nových

položek do objednávky. V tomto okně se zboží vyhledá opět postupným psaním do textových polí hledat dle kódu a dle názvu, nastaví se počet kusů a tlačítkem *Přidat* se vloží do objednávky. Pokud se objednávka teprve vytváří, zboží se přidá pouze do lokální tabulky a vše se musí potvrdit vytvořením objednávky. Pokud však uživatel zboží přidává při editaci objednávky, je zboží vloženo přímo do databáze a již není nutné nic ukládat.

5.5.4 Editace zákazníků

Pokud uživatel zjistí, že například při tvorbě objednávky nenajde hledaného zákazníka, nebo informace o něm nejsou relevantní, musí přejít na záložku *Zákazníci*. Zde opět může vyhledat stávajícího zákazníka dle id nebo dle jména a výsledek se zobrazí v přehledné tabulce.

V další části okna se nachází několik textových polí umožňujících zobrazit popřípadě doplnit údaje o zákazníkovi jako například jméno a adresa. Po doplnění nebo vyplnění údajů má uživatel možnost změny u zákazníka uložit, vytvořit nového nebo vybraného zákazníka odstranit z databáze. Náhled okna se záložkou Zákazníci je zobrazen na Obrázek 13.



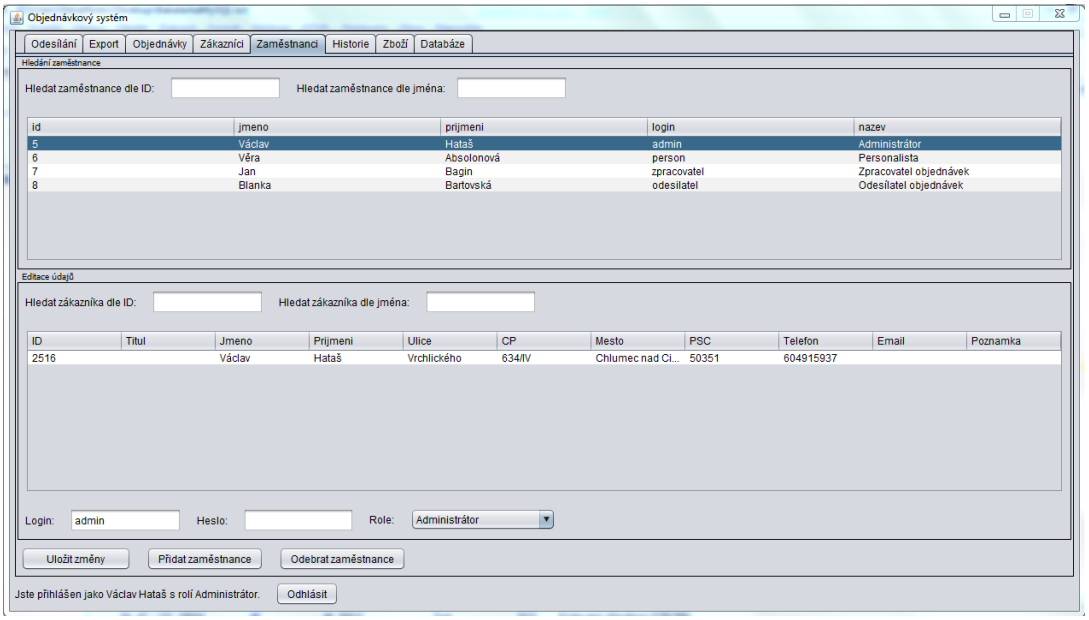
Obrázek 13 – Ukázka záložky Zákazníci

5.5.5 Editace zaměstnanců

Pokud potřebuje zaměstnavatel vytvořit nového zaměstnance nebo stávajícímu upravit údaje, přepne se do záložky *Zaměstnanci*. Zaměstnanec je v tomto programu speciálním případem zákazníka. To znamená, že pokud chceme vytvořit nového zaměstnance, je zapotřebí, aby daný člověk existoval v databázi jako zákazník. Tabulka zákazník totiž obsahuje veškeré nutné informace o člověku, jako je jméno a adresa, čímž dochází k šetření prostoru. Tabulka zaměstnanec poté již přidá pouze přihlašovací jméno a heslo do aplikace a přiřadí konkrétní roli v aplikaci.

Logika všech oken zůstává stejná. Nejprve lze vyhledat zaměstnance dle id a jména a výsledky se zobrazí v první tabulce. Po vybrání zaměstnance se vyplní jeho údaje týkající se zákazníka (jméno a adresa) do druhé tabulky a zobrazí se přihlašovací jméno a role uživatele v systému. Pokud je třeba zaměstnance vytvořit, vyhledá se zákazník, z něhož vznikne zaměstnanec, pomocí textových polí pro zadávání id zákazníka a jeho jména.

Na závěr lze vytvořit nového zaměstnance, odstranit stávajícího nebo uložit všechny provedené změny zaměstnance kromě změny jeho osoby. Nedává totiž smysl, aby se měnila osoba a zůstávalo stejné přihlašovací jméno a heslo. Z toho důvodu je nutné zaměstnance nejprve smazat a poté vytvořit nového. Dále pak není možné přiřadit nějakému zaměstnanci již existující přihlašovací jméno, aby nemohlo při přihlašování dojít k záměně. Ukázka záložky *Zaměstnanci* je zobrazena na Obrázek 14.

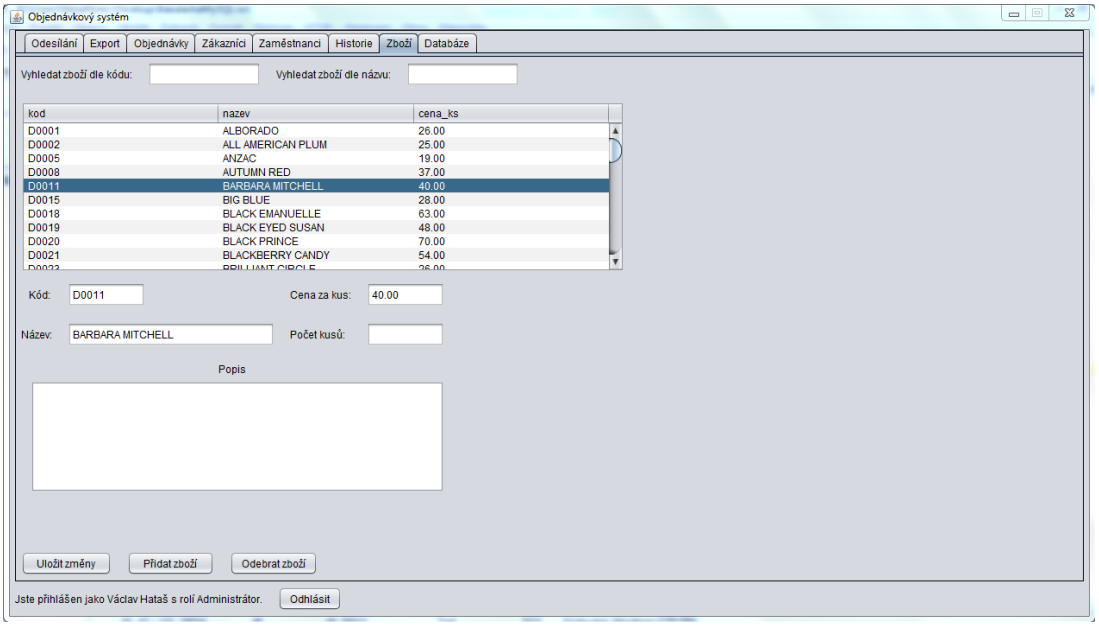


Obrázek 14 – Ukázka záložky Zaměstnanci

5.5.6 Editace zboží

Poslední záložkou, která slouží k nějakým úpravám nebo přidávání je záložka *Zboží*. Jak její název napovídá, půjde o editaci sortimentu zboží. Funkčně se jedná o velice jednoduchou záložku, kde lze opět vyhledat zboží dle kódu nebo dle názvu a výsledek vyhledávání se zobrazí do tabulky. Po vybrání nějakého zboží se vyplní textová pole obsahující položky kód, název, cena za kus, počet kusů a popis. Následně lze jednotlivá pole upravovat a výsledek uložit, nebo vytvořit zboží nové. Poslední funkcí je samozřejmě i smazání vybrané položky.

Tento program není určen jako náhrada za skladové účetnictví. Proto jsou použity pouze základní funkce. Neřeší se zde například DPH, oznámení při docházejícím zboží nebo hromadné naskladnění. Pokud by ovšem bylo třeba nějakou funkci nebo vlastnost přidat, je na to v této záložce prostor. Názorná ukázka této záložky je vyobrazena na Obrázek 15.



Obrázek 15 – Ukázka záložky Zboží

5.5.7 Pracovní záznamy

Tato záložka, v aplikaci označená jako *Historie*, je určená především pro zaměstnavatele. Dovoluje zde uživateli prohlížet, kolik, jaký den a který uživatel zpracoval objednávek k exportu. Tímto nástrojem lze kontrolovat například dodržování norem nebo posuzovat velikost osobního ohodnocení v závislosti na výkonu.

Všechny záznamy jsou ve výchozím stavu řazeny dle data. Řazení však lze změnit a to na základě jména nebo výkonnosti. Samozřejmostí je možnost řazení jak vzestupně, tak sestupně. Další věcí, které zpřehlední a zjednoduší tuto práci je možnost seskupení. Jako výchozí není nastavené žádné seskupení, lze si ovšem vybrat ze seskupení dle data nebo jména. Seskupením dle data lze dosáhnout součtu výkonností všech zaměstnanců za jeden den, naopak dle jména poté součtu všech objednávek, které daný zaměstnanec kdy zpracoval.

5.5.8 Přímý přístup do databáze

Poslední záložka, pojmenovaná jako *Databáze*, slouží jako okno k přímému přístupu do databáze. Umožňuje uživateli zadávat SQL příkazy. Dokáže nejen provést DDL nebo DML příkazy, ale zvládne i vrátit do připravené tabulky obsah příkazu select. V jisté míře může nahradit i specializované aplikace pro správu databází jako je například MySQL Workbench, ale proti nim má samozřejmě značné nevýhody. Jednou z největších nevýhod je nutnost znalosti všech databázových objektů pro práci s nimi, z důvodu absence jeho seznamu. Výhodou je však rychlost, danou nepotřebou spouštění další aplikace, se kterou je schopen uživatel během jedné vteřiny psát příkazy.

5.6 Uživatelské role

Tento program používá pro rozlišení pravomocí jednotlivých uživatelů role. Každá z nich říká, jaké části (přesněji záložky) programu mají být uživateli přístupné. Pro každou roli jsou v programu zobrazeny pouze takové záložky, ke kterým má daná role přístup. Příklad zobrazených záložek personalisty je zobrazen na Obrázek 16. V současnosti aplikace obsahuje následující čtyři role:

- odesílatel objednávek,
- zpracovatel objednávek,
- personalista,
- administrátor.

Všechny role jsou uloženy v databázové tabulce *Role* obsahující název role a její id. Každý řádek tabulky *Zamestnanec* musí mít přiřazenu právě jednu z nich. Pokud by byl vytvořen uživatel s neexistujícím id role, nebude mít přístup do žádné části programu a bude schopen se pouze odhlásit. V případě nutnosti vytvořit další roli, stačí přidat záznam do tabulky *Role* a ve zdrojovém kódu aplikace přidat jeden řádek s polem typu *boolean* určující možnost přístupu nové role do jednotlivých záložek programu.

5.6.1 Odesílatel objednávek

Uživatel s touto rolí má za úkol přidávat objednávky k exportu a následně všechny exportovat. Jedná se o základní roli a je proto vhodné jí používat. Tato role garantuje přístup do prvních dvou záložek programu s názvem *Odesílání* a *Export*. Ostatní záložky jsou vždy skryty.

Zaměstnanec s touto rolí by tedy měl přijmout seznam připravených objednávek a dle něho v rámci záložky Odesílání přidat k exportu. Po přidání všech a kontrole správnosti by měl přejít k exportu objednávek. Po dokončení procedur ze strany dopravce nakonec smaže dané CSV soubory a předá vytisknuté etikety k nalepení.

5.6.2 Zpracovatel objednávek

Zpracovatel objednávek je člověk, který má na starosti všechny úkony týkající se objednávek od editace a rušení až po vytváření. Pokud chce vytvořit objednávku, musí být schopen i vytvořit zákazníka. Má tedy přístup jak do záložky *Objednávky*, tak do záložky *Zákazníci*.

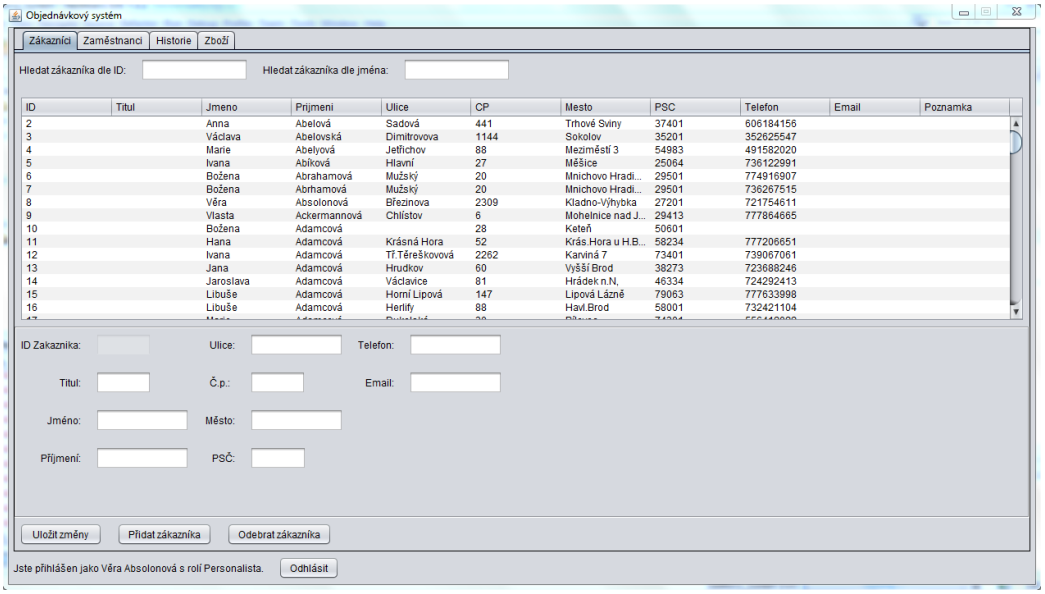
Pracovní náplní tohoto uživatele by bylo například vyřizování telefonických nebo písemných objednávek, které není možno zpracovat strojově. Dále by pak řešil nesrovnalosti v objednávkách a popřípadě je upravoval nebo rovnou rušil.

5.6.3 Personalista

Tato role obsahuje správu zaměstnaneckých účtů, popřípadě kontrolu pracovních výkonů jednotlivých zaměstnanců. Aby mohla pracovat se zaměstnanci, musí mít možnost pracovat i se zákazníky. Pro jednoduchost byla této roli přiřazena i možnost editace sortimentu zboží.

Má tedy přístup do záložek *Zákazníci*, *Zaměstnanci*, *Historie* a *Zboží*. Příklad zobrazení okna, když je přihlášen personalista, je zobrazen na Obrázek 16.

Tento člověk by tedy přijímal nové zaměstnance a přidělovat jim přihlašovací údaje. Dále by mohl kontrolovat výkonnost jednotlivých odesílatelů, zda plní normy. Jako poslední věc by se staral o přidávání nového zboží popřípadě editace stávajícího zboží.



Obrázek 16 – Příklad přístupných záložek personalistovi

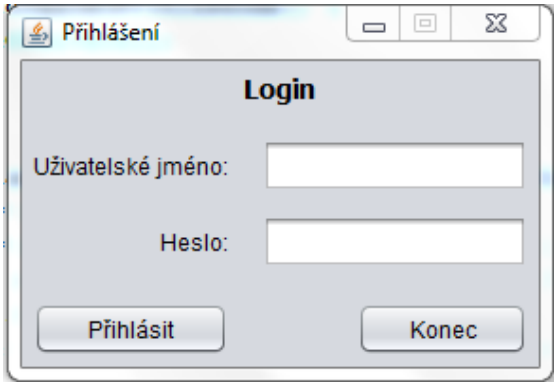
5.6.4 Administrátor

Administrátor je takzvaný „superuser“ a má tedy přístup do všech částí programu jako ostatní role. Navíc může vstoupit do záložky *Databáze*, kde může zadávat SQL příkazy přímo do databáze. Administrátor je nezbytná role, kterou ovšem nelze vytvořit přímo v programu. Pro přihlášení musí být uživatel již vytvořen například nástrojem MySQL Workbench. Příkaz vypadá takto:

```
INSERT INTO `zamestnanec`(`Login`,`Heslo`,`Zakaznik_ID`,`Role_ID`)
VALUES(md5("admin"),md5("admin"), 2516, 1);
```

5.7 Zabezpečení aplikace

Aplikace má, co se týče bezpečnosti, jeden kritický bod. Je jím úvodní přihlašovací obrazovka a její dvě textová pole, která dovolují zadávat jakýkoli text. Do dalších částí programu se nepřihlášený uživatel dostat nemůže. Veškerá zabezpečení se proto musela zaměřit právě na okno zobrazené na obrázku Obrázek 17.



Obrázek 17 – Přihlašovací okno

Jelikož je uživatelské jméno a heslo používáno jako podmínka příkazu select pro ověření přihlašovacích údajů, hrozí v tomto místě útok zvaný SQL Injection. Útok je založen na napsání takového příkazu, který ukončí původní výraz a místo toho provede vlastní, typicky uvedený za uvozovkami.

V prvních verzích návrhu jsem tento problém neřešil, a proto jsem se pokusil o přihlášení bez toho, abych zadal platné uživatelské jméno a heslo. Po vyzkoušení různých někdy velice obsáhlých příkazů se to podařilo. Do následujícího příkazu stačilo jako login a heslo napsat pouhé tři znaky `"+"`.

```
Select Jmeno, Prijmeni, Role_ID, Nazev, zamestnanec.ID from
zamestnanec join zakaznik on Zakaznik_ID = zakaznik.ID
join role on role_ID = role.ID
where Login = "login" and Heslo = "heslo"
```

Nejen že se aplikace po potvrzení přihlásila, přihlášen byl rovnou administrátorský účet s oknem pro přímý přístup do databáze. Tím byla objevena opravdu značná mezera v zabezpečení. Za prvé jsem přemýšlel a zjišťoval, jak je možné, že MySQL vyhodnotí zmíněnou podmínku jako pravdivou. Došel jsem k zajímavému zjištění, že MySQL nepoužívá znak „+“ ke spojování řetězců, nýbrž pouze ke sčítání. Tím převede prázdný text na nulu a při porovnávání implicitně převede login také na číslo nula. Nula se rovná nule. Podmínka splněna, příkaz vrátí všechna přihlašovací jména a hesla a aplikace vybere pro přihlášení první záznam, což je právě administrátor.

Tento problém má velice jednoduché řešení. Je jím zašifrování přihlašovacího jména a hesla například metodou `md5()`, která je podporovaná přímo databázovým serverem MySQL. Cokoli nyní útočník zadá, bude nejprve zašifrováno a až poté porovnáno se záznamy v tabulce zaměstnanců. Opravená kluzule *WHERE* vypadá tedy následovně:

```
where Login = md5("login") and Heslo = md5("heslo")
```

Závěr

Cílem této práce bylo vytvořit aplikaci, která může sloužit malým firmám zabývajícími se zásilkovým obchodem. Aplikace tedy měla primárně vytvářet soubory obsahující informace o odesílaných objednávkách v takovém formátu, aby jej mohli přepravní společnosti importovat do svého systému. To se mi podařilo s využitím formátu CSV a doufám, že i přehledným a inuitivním způsobem. Dále bylo nutné, aby aplikace uměla objednávky vytvářet i upravovat, což přehledným způsobem zajišťuje k tomu vytvořená záložka *Objednávky*. Do práce jsem navíc přidal i další funkce nad rámec zadání, jako například okno pro přímé zadávání SQL příkazů do databáze vhodnou pro administrátory, nebo kontrolu pracovních výkonů zaměstnanců. Aplikace navíc obsahuje uživatelské role pro omezení přístupu do různých částí programu a po startu aplikace je tedy nutné se přihlásit.

Aplikace je do budoucna snadno rozšiřitelná a to jak možností vytvoření nové přepravní společnosti pouhým přidáním záznamu do patřičné tabulky databáze, tak i možností vytvořit novou uživatelskou roli opět přidáním záznamu do databáze a přidělením povolení vstupu do určitých záložek v kódu programu. Možné by bylo i rozšíření funkcionality přidáním dalšího okna do záložkového panelu při doplnění patřičných metod. Reálně by mohla být aplikace doplněna například o tvorbu faktur.

Při tvorbě této aplikace jsem prohloubil svoje znalosti a zkušenosti s programovacím jazykem Java a s tvorbou grafického uživatelského prostředí aplikací. Dále pak s návrhem a tvorbou databáze a hlavně s jejím propojením s aplikací psanou v jazyku Java pomocí JDBC. Právě to byla stěžejní činnost, kterou bylo třeba nejvíce studovat. To jsem zvládl a aplikace je tedy plně funkčí. Práce na vývoji tohoto programu mě velice bavila a jsem rád, že jsem jeho tvorbu absolvoval.

Literatura

[1] SPELL, Brett. *Java: programujeme profesionálně*. Vyd. 1. Praha: Computer Press, 2002, 1022 s. ISBN 80-722-6667-5.

[2] HEROUT, Pavel. *Java: grafické uživatelské prostředí a čeština*. 2. vyd. České Budějovice: Kopp, 2007, 316 s. ISBN 978-80-7232-328-9.

[3] GILMORE, W. *Velká kniha PHP 5 a MySQL: kompendium znalostí pro začátečníky i profesionály*. Nové, 3. vyd. Překlad Jan Pokorný. Brno: Zoner Press, 2011, 736 s. Encyklopedie Zoner Press. ISBN 978-80-7413-163-9.

[4] ŠEDA, Jan. *Úvod do JDBC* [online]. Interval.cz, 2003-03-04 [cit. 2014-05-04]. Dostupný z WWW: <http://interval.cz/clanky/uvod-do-jdbc/>. ISSN 1212-8651.

[5] *JDBC driver* [online]. Wikipedie – otevřená encyklopedie, 2014-04-24 [cit. 2014-05-04]. Dostupné z: <http://en.wikipedia.org/wiki/JDBC_driver>.

[6] *JDBC Tutorial* [online]. TUTORIALSPOINT, 2014 [cit. 2014-05-04]. Dostupné z: <http://www.tutorialspoint.com/jdbc/index.htm>.

[7] SKŘIVAN, Jaromír. *Databáze nejsou jen MySQL* [online]. Interval.cz, 2002-01-05 [cit. 2014-05-04]. Dostupný z WWW: < http://interval.cz/clanky/databaze-nejsou-jen-mysql/>. ISSN 1212-8651.

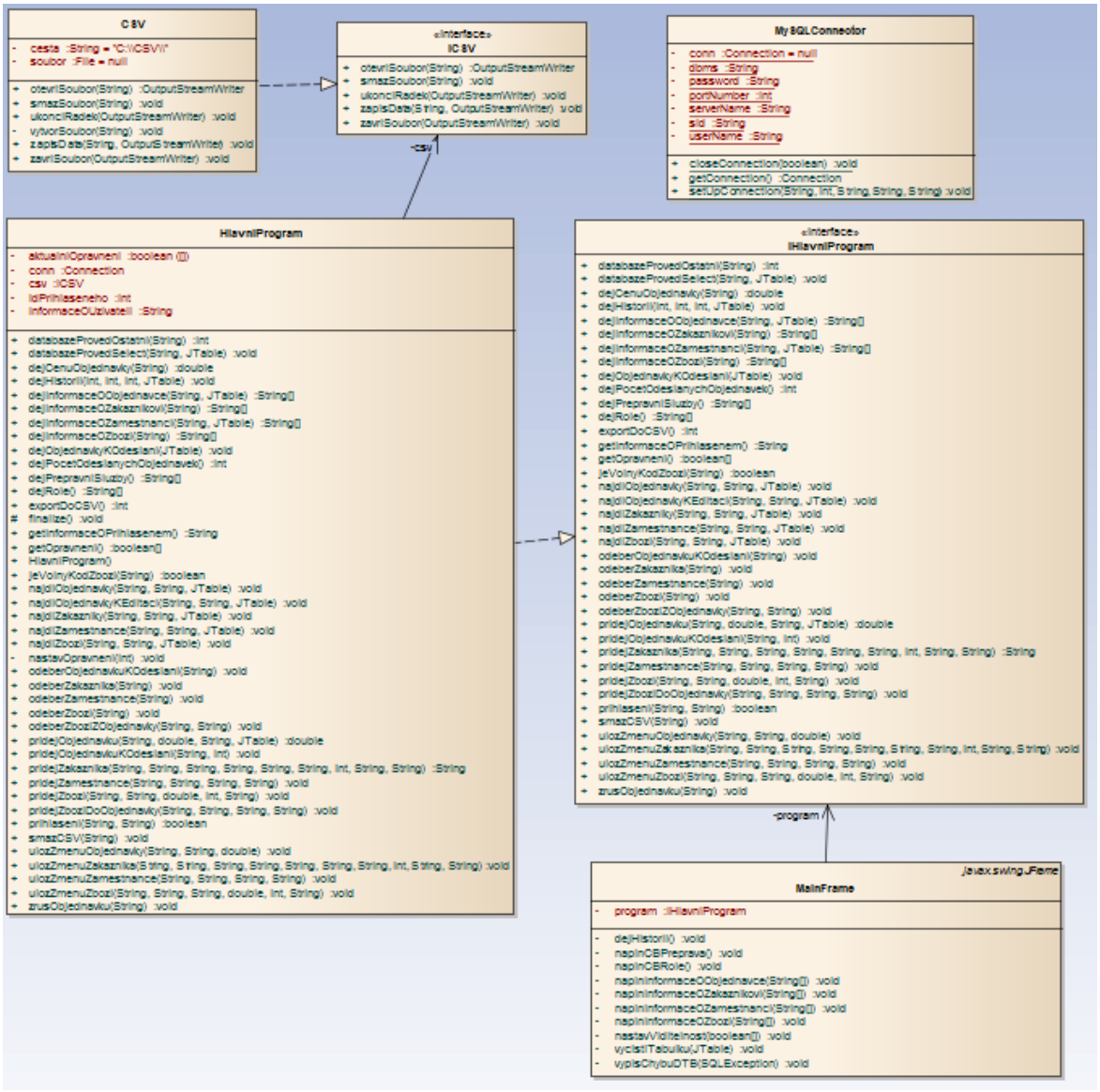
[8] *Java™ Platform, Standard Edition 7 API Specification* [online]. Oracle 2014 [cit. 2014-05-04]. Dostupné z: <http://docs.oracle.com/javase/7/docs/api/>.

[9] *Extensible Markup Language* [online]. Wikipedie – otevřená encyklopedie, 2013-11-18 [cit. 2014-05-04]. Dostupné z: <http://cs.wikipedia.org/wiki/Xml>.

[10] *Portable Document Format* [online]. Wikipedie – otevřená encyklopedie, 2014-02-10 [cit. 2014-05-04]. Dostupné z: <http://cs.wikipedia.org/wiki/Pdf>.

[11] *CSV* [online]. Wikipedie – otevřená encyklopedie, 2014-04-18 [cit. 2014-05-04]. Dostupné z: <http://cs.wikipedia.org/wiki/CSV>.

Příloha A – Diagram tříd



Příloha B – Ukázka zdrojového kódu použití metadat

```
public void databazeProvedSelect(String prikaz, JTable tabulka) throws
SQLException {

        Statement stmt = conn.createStatement();
        ResultSet rset = stmt.executeQuery(prikaz);
        ResultSetMetaData rsmd = rset.getMetaData();

        Vector<String> sloupce = new Vector<>();
        for (int i = 1; i <= rsmd.getColumnCount(); i++) {
            sloupce.add(rsmd.getColumnLabel(i));
        }

        DefaultTableModel model = new DefaultTableModel(sloupce, 0);
        tabulka.setModel(model);

        while (rset.next()) {
            Vector<String> radek = new Vector<>();
            for (int i = 1; i <= rsmd.getColumnCount(); i++) {
                radek.add(rset.getString(i));
            }
            model.addRow(radek);
        }
        stmt.close();

tabulka.setSelectionMode(DefaultListSelectionModel.SINGLE_SELECTION);
        tabulka.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_COLUMNS);
    }
```

Příloha C – Tabulka převodu datových typů mezi SQL a Javou

Tabulka 2 – Převod datových typů mezi SQL a Javou [6]

SQL	JDBC/Java	setXXX	getXXX
VARCHAR	java.lang.String	setString	getString
CHAR	java.lang.String	setString	getString
LONGVARCHAR	java.lang.String	setString	getString
BIT	boolean	setBoolean	getBoolean
NUMERIC	java.math.BigDecimal	setBigDecimal	getBigDecimal
TINYINT	byte	setByte	getByte
SMALLINT	short	setShort	getShort
INTEGER	int	setInt	getInt
BIGINT	long	setLong	getLong
REAL	float	setFloat	getFloat
FLOAT	float	setFloat	getFloat
DOUBLE	double	setDouble	getDouble
VARBINARY	byte[ ]	setBytes	getBytes
BINARY	byte[ ]	setBytes	getBytes
DATE	java.sql.Date	setDate	getDate
TIME	java.sql.Time	setTime	getTime
TIMESTAMP	java.sql.Timestamp	setTimestamp	getTimestamp
CLOB	java.sql.Clob	setClob	getClob
BLOB	java.sql.Blob	setBlob	getBlob
ARRAY	java.sql.Array	setARRAY	getARRAY
REF	java.sql.Ref	SetRef	getRef
STRUCT	java.sql.Struct	SetStruct	getStruct