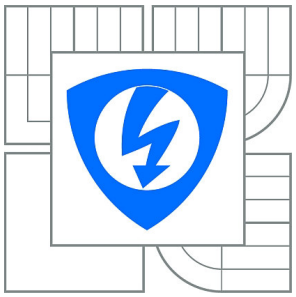
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

NÁVRH AUDIO-VIZUÁLNÍ DATABÁZE PRO EXTRAKCI SEKUNDÁRNÍCH BIOMETRICKÝCH A NE-BIOMETRICKÝCH ZNAKŮ

AUDIO-VISUAL DATABASE FOR SOFT BIOMETRIC AND NON-BIOMETRIC TRAITS EXTRACTION

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

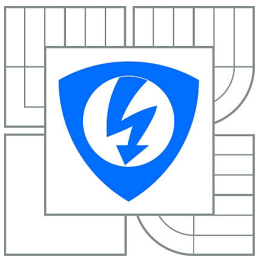
AUTOR PRÁCE — Bc. DAVID HOLEKSA

AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ PŘINOSIL, Ph.D.

SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. David Holeksa	***ID:***	126772
Ročník:	2	***Akademický rok:***	2014/2015

NÁZEV TÉMATU:

Návrh audio-vizuální databáze pro extrakci sekundárních biometrických a ne-biometrických znaků

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte možností extrakce sekundárních biometrických a ne-biometrických znaků. Na základě získaných poznatků vyberte vhodné sekundární biometrické a ne-biometrické znaky pro účely identifikace osob. Následně proveďte návrh koncepce databáze obrazů obsahujících vybrané sekundární biometrické a ne-biometrické znaky a tuto databázi naplňte vhodnými obrazovými daty. Nad těmito daty poté realizujte segmentaci za účelem definice pozice jednotlivých znaků v obraze. Na závěr navrhněte sadu algoritmů, které pro jednotlivé znaky jejich provede automatickou (polo-automatickou) anotaci.

DOPORUČENÁ LITERATURA:

[1] Jain, A. K., Flynn, P., Ross, A.: Handbook of Biometrics, Springer, 2008, ISBN 978-0-387-71041-9.

[2] Jain, A. K., Dass, S. C., Nandakumar, K.:Soft Biometric Traits for Personal Recognition Systems, Lecture Notes in Computer Science, Volume 3072, s. 731-738, 2004.

Termín zadání:	9.2.2015	***Termín odevzdání:***	26.5.2015

Vedoucí práce: Ing. Jiří Přinosil, Ph.D.
Konzultanti diplomové práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Abstrakt

Cílem této diplomové práce je navrhnout vhodnou databázovou strukturu pro audio-visuální data, části extrahované z těchto dat odpovídající sekundárním biometrickým a ne-biometrickým znakům a popis těchto znaků. Databáze obsahuje algoritmy, které provádí automatickou anotaci sekundárních znaků z předem připravených obrazových dat. Výsledná databáze bude sloužit pro výzkum identifikace audio-visuálního obsahu multimediálních dat.

Klíčová slova

Biometrika, Biometrické metody, biometrické znaky, ne-biometrické znaky, Databáze, Objektově orientovaný návrh, UML, Java,

Abstract

The aim of this semester project is to design a suitable database structure for the audio-visual data, parts extracted from the data corresponding to secondary biometric and non-biometric characteristics and description of these characters. The database contains algorithms that perform automatic annotation secondary characters from the image data prepared in advance. The resulting database will be used for research into the identification of audio-visual content of multimedia data.

Key words

Biometry, Biometrics methods, biometric characters, non-biometric characters, databases, object-oriented design, UML, Java

Bibliografická citace

HOLEKSA, D. *Návrh audio-vizuální databáze pro extrakci sekundárních biometrických a ne-biometrických znaků.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2015. 52 s. Vedoucí diplomové práce Ing. Jiří Přinosil, Ph.D.

Prohlášení

Prohlašuji, že svou diplomovou práci na téma „Návrh audio-vizuální databáze pro extrakci sekundárních biometrických a ne-biometrických znaků“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních nebo majetkových a jsem si plně vědom následků porušení ustanovení §11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonu (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne …… ……

(podpis autora)

Poděkování

Děkuji vedoucímu diplomové práce Ing. Jiřímu Přinosilovi, Ph.D. za velmi užitečnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce. Dále bych chtěl poděkovat mé rodině a přítelkyni za podporu a trpělivost při psaní této práce.

V Brně dne ……. …….

(podpis autora)

SIX research centre
sensor, information and communication systems

Faculty of Electrical Engineering
and Communication

Brno University of Technology
Technicka 12, CZ-61600 Brno, Czechia

http://www.six.feec.vutbr.cz

Výzkum popsaný v této diplomové práci byl realizovaný v laboratořích podpořených projektem Centrum senzorických, informačních a komunikačních systémů (SIX); registrační číslo CZ.1.05/2.1.00/03.0072, operačního programu Výzkum a vývoj pro inovace.

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj pro inovace

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM VÝPISŮ KÓDŮ

ÚVOD

Cílem této diplomové práce je navrhnout vhodnou databázovou strukturu pro audio-visuální data, části extrahované z těchto dat odpovídající sekundárním biometrickým a ne-biometrickým znakům a popis těchto znaků. Výsledná databáze bude sloužit jak pro výzkum de-identifikace audio-visuálního obsahu multimediálních dat, tak pro jejich ověření z hlediska spolehlivosti a robustnosti. Dále je třeba shromáždit dostatečné množství multimediálních dat a získat z nich oblasti odpovídající vhodně vybraným sekundárním znaků.

Nejprve se v práci budeme zabývat biometrií obecně a následně se pak zaměříme na problematiku identifikace člověka na základě jeho sekundárních znaků. Jelikož jako zdrojová data pro identifikaci budou použity multimediální data, řeší se v práci výběr vhodných sekundárních biometrických a ne-biometrických znaků pro tento případ.

V další části zde bude navržena databázová struktura na základě objektově orientovaného návrhu. Takto navržena struktura bude zobrazena pomocí UML diagramu. Databáze bude umět načítat nová data, mazat stávající a vyhledávat v těchto datech, proto zde budou uvedeny způsoby řešení těchto funkcí. Dalším požadavkem na databázi je aby prováděla automatickou anotaci nad vstupními obrazovými daty. Budou tedy zde uvedeny algoritmy provádějící definici pozice znaků v obraze a algoritmy určující barvu daného znaku. Kód realizující navrženou databázi bude napsán programovacím jazykem JAVA ve vývojovém prostředí Eclipse SDK 4.4.2.

Posledním úkolem je připravit vstupní data a naplnit s nimi navrženou databázi. Bude provedena segmentace obrazových dat pro jednotlivé znaky v grafickém nástroji Adobe Photoshop CS5.

1 ÚVOD DO BIOMETRIE

1.1 VZNIK BIOMETRIE

Termín biometrie byl používán již v první polovině 19. století. Její význam vzrostl až koncem 19. Století s rozvojem statistiky a biologie. Za zakladatele biometrie je považován Karl Pearson, podle něhož je biometrie užívání metod matematické statistiky při studiu proměnlivosti živých organismů.

Dalším Významným představitelem biometrie je Adolphe Lambert Quételet, od něhož pochází myšlenka vytvoření aritmetického průměru z většího počtu měření různých jedinců. V roce 1835 dokonce sestavil tzv. „průměrného člověka“ (homme moyen), což byl ideál sestavený z průměru jednotlivých znaků. Tento člověk byl pro něj biologický prototyp dokonalosti. Quételet je autorem tzv. Quételetova indexu QI, který stanovuje hranici obezity na základě poměru tělesné výšky a váhy. Dnes je tento index znám jako BMI.

Mezi představitele biometrie se řadí také Francis Galton, jehož jméno je spíše spojeno s počátky daktyloskopie, která je založena na principech biometrie. Galton byl jedním ze spoluzakladatelů časopisu Biometrika.[1]

1.2 BIOMETRIE

Biometrie je vědní obor zabývající se studií a zkoumáním živých organismů, především člověka, a měřením jeho anatomických a fyziologických vlastností a také jeho chováním. Pojem biometrika je odvozený z řeckých slov "bios" a "metron". První znamená "život", druhé pak "měřit". V přeneseném významu jde o měření a rozpoznávání určitých charakteristik člověka. Biometrika se věnuje studiu metod vedoucích k rozpoznávání člověka na základě jeho unikátních proporcí nebo vlastností. V zahraniční je pojem biometric přímo vykládán jako proces automatizované metody rozpoznávání jedince založený na měřitelnosti biologických a behaviorálních vlastností (dle NSTC – Nation Science and Technology Council – Národní rada pro vědu a technologii USA, Výboru pro vnitrostátní a národní bezpečnost). [2]

1.3 BEHAVIOMETRIKA

Jak bylo zmíněno výše, biometrie se zabývá i chováním člověka. Tomuto se věnuje speciální podkapitola behaviometrika. Je to mladší odvětví biometrie sledující vlastnosti, jako může být například styl psaní na klávesnici, styl chůze, gest apod. Oproti biometrii, která je zpravidla jednorázovým úkonem, povolení nebo zamítnutí přístupu, může behaviometrika probíhat stále což znamená, že neustále sleduje, jestli se systémem pracuje oprávněná osoba. [3]

1.4 ZÁKLADNÍ POJMY

V problematice biometrie je nutné správně rozumět základním pojmům, které mají původ v anglickém jazyce a do češtiny bývají občas nesprávně překládány.

- Recognition (rozpoznávání) je druhový termín, který nutně nemusí znamenat identifikaci ani verifikaci. Jedná se o rozpoznávání člověka použitím vhodné tělesné vlastnosti.
- Verification (ověření nebo verifikace) označuje proces, při kterém se biometrický systém pokouší potvrdit totožnost jedince, který se s ní prokazuje, srovnáním sejmutého vzorku s již dříve zapsaným (tzv. šablonou neboli template). Jedná se o tzv. princip one-to-one.
- Identification (identifikace) je proces, kdy se biometrické systém pokouší určit totožnost neznámého jedince. Biometrická informace je sejmuta a porovnávána se všemi uloženými vzorky (šablonami). Princip je znám jako one-to-many.
- Matching (srovnání) je srovnání biometrických vzorků a určuje stupeň shodnosti. Výsledkem je takzvané skóre.
- Skóre je hodnota, která určuje stupeň shody dvou porovnávaných biometrických předloh. Shodnost vzorků nikdy nebude stoprocentní.
- Mez je hodnota, která je předem dána administrátorem. Vzorek, který má skóre vyšší, než je stanovená mez, je vyhodnocen jako vyhovující a zbytek jako nevyhovující.
- Authentication (autentifikace, autentizace nebo legalizace) je pojem, který lze sloučit s termínem rozpoznávání. Ovšem na konci procesu v tomto případě získá uživatel určitý status, např. oprávněný/neoprávněný.[2], [4]

1.5 BIOMETRICKÉ INFORMACE

Biometrická informace je taková informace, která nese údaj o skenovaném subjektu. Aby byla biometrická informace použitelná, měla by vyhovovat kritériím pro výběr biologické nebo behaviorální vlastnosti člověka určené pro jeho identifikaci. Tyto kritéria jsou:

- jedinečnost: vlastnost musí být, co možná nejvíc výjimečná tzn., že se shodná vlastnost nesmí objevit u dvou lidí zároveň
- univerzálnost: vlastnost musí být měřitelná u co možná největší množiny lidí
- trvalost: vlastnost se nesmí měnit v čase
- měřitelnost: vlastnosti musí být měřitelné shodnými technickými zařízeními
- uživatelská přijatelnost: vlastnost musí být snadno a pohodlně měřitelná

Nejlépe prozkoumané a nejvíce rozšířené biometrické vlastnosti používané pro identifikační účely jsou uvedeny níže[2]:

- otisk prstu (struktura papilárních linií a jejich detailů)
- dynamika podpisu (rozdíly v tlaku a rychlosti psaní)
- geometrie tváře (vzdálenosti specifických částí – oči, nos, ústa…)
- duhovka oka (obrazový vzorec duhovky)
- sítnice oka (struktura žil na očním pozadí)
- geometrie ruky (rozměry dlaně a prstů)
- struktura žil na zápěstí (struktura žil)
- tvar ucha (rozměry viditelné části ucha)
- hlas (tón a zabarvení hlasu)
- DNA (řetězec deoxyribonukleové kyseliny)
- pach (chemické složení)
- psaní na klávesnici

Existují v podstatě tři základní způsoby vzniku biometrické vlastnosti člověka[2]:

- genetický vývoj: uplatňuje se vliv dědičnosti (DNA) – genotypické
- náhodné varianty vzniku v časném stádiu vývoje embrya – randotypické
- učení a výchovu: chování jedince – behaviorální

1.6 BIOMETRICKÉ SYSTÉMY

Biometrické systémy jsou založeny na automatickém snímání biometrických znaků a jejich následném porovnávání s údaji z předem vytvořené databáze. Důvod vzniku těchto systému je, že sledované znaky subjektu se nedají oklamat nebo odcizit jako například PIN kód. Následuje stručný popis nejrozšířenějších biometrických systémů založených na fyziologických charakteristikách.

- Verifikace obličeje: vychází se z předpokladu, že obličej obsahuje asi 80 typických rysů a pro zjištění totožnosti stačí rozpoznat 14 až 20. Nejdůležitější je tvar obličeje, poloha očí, obočí, nosu a úst. Na identifikaci nemá vliv paruka, vousy, brýle nebo změna účesu. Naopak vliv má stárnutí nebo plastická operace. Tento systém je pro snímaný subjekt přijatelný, protože zde nedochází k žádnému fyzickému kontaktu, ale je méně přesný než snímání otisku prstů nebo oční duhovky.
- Verifikace otisku prstu: je nejpoužívanější biometrickou metodou. Je známo zhruba 60 různých forem otisků prstů, které jsou různě kombinovány na jednom otisku. Tato metoda má však určité nevýhody a to, že je nutný fyzický kontakt snímaného subjektu, drobné poranění prstu může způsobit systémové odmítnutí a existuje zde možnost napodobení např. pomocí silikonu. Výhoda je však v tom, že otisky prstů jsou pro každého člověka jedinečné a to platí i u jednovaječných dvojčat.
- Verifikace duhovky: je metoda dosahující velmi vysoké přesnosti, protože duhovka je pro každého člověka naprosto jedinečná. U rozpoznávání duhovky známe, oproti otiskům prstu, více než 400 různých forem oční duhovky. Každý subjekt má dokonce obě duhovky naprosto jedinečné a to i dvojčata na rozdíl od DNA. Důležité je, že duhovka se během života téměř nemění.
- Verifikace sítnice: je starší metoda než identifikace podle duhovky. Jedná se o velmi přesnou a spolehlivou metodu, ale je nutné, aby se subjekt díval na přesně stanovené místo v daném bodě. Nevýhoda však nastává pro lidi nosící brýle.
- Geometrie ruky: vychází z toho, že od určitého věku se tvar ruky nemění. Původně se měřily jednotlivé prsty, ale v současné době se také měří délka a šířka dlaně a boční profil ruky.[1]

2 SEKUNDÁRNÍ ZNAKY

V předchozí části textu se píše o způsobech identifikace člověka na základě jeho biometrických znaků. Podle zmíněných biometrických znaků v kapitole 2.5, se dá určit identita člověka téměř jednoznačně, takové znaky se nazývají primární. Zadání tohoto projektu ale žádá návrh databáze a její následné naplnění sekundárními znaky a to jak biometrickými, tak ne-biometrickými. Za sekundární biometrické znaky jsou považovány takové znaky, pomocí kterých se nedá určit identita člověka jednoznačně, ale zúží se tak okruh lidí, kteří by mohli hledanému profilu odpovídat. U ne-biometrických znaků je to stejné, až na to s tím rozdílem, že tyto znaky se netýkají samotného lidského těla posuzované osoby ale okolí a věcí, kterými se tato osoba obklopuje a používá.

2.1 ZÍSKÁVÁNÍ A VYUŽITÍ SEKUNDÁRNÍCH ZNAKŮ

Nejvíce sekundárních znaků pozorované osoby se lze získat z přímého pohledu na tuto osobu, což ale není vždy možné. V takovém případě, kdy není pozorovaná osoba přítomna, jedná se například o pohřešovanou osobu, využívá se k získání informaci o této osobě vnějšího popisu této osoby poskytnutého třetí osobou, nebo fotografie. U informací získaných z fotografie dochází pouze k minimálnímu zkreslení této informace, zatím co u informace získané z popisu, závisí přesnost a množství těchto informací na popisující osobě. Podle způsobu, jakým se popisování osoby uskutečňuje, se rozlišují dva druhy popisu osoby:

- úřední popis – jde o kvalifikovaný popis osoby, který zhotovuje k tomuto určená, odborně způsobilá osoba (speciálně vyškolený kriminalistický technik)
- laický popis – jde o nekvalifikovaný popis osoby, který poskytne osoba, která zná nebo viděla popisovanou osobu (svědek, poškozený, známý, příbuzný) [1]

Sekundární znaky neurčují identitu osoby jednoznačně, ale na rozdíl od primárních, se získávají mnohem snadněji a pozorovaná osoba ani nemusí být osobně přítomná při získávání těchto znaků. Když se shromáždí dostatečné množství sekundárních znaků, zúží se tak počet potencionálně odpovídajících osob, který může v dané oblasti lidí určit hledanou osobu. Tohoto se využívá zejména v kriminalistice a to pro účely pátraní, vyšetřování a identifikování osob, které spáchaly trestný čin nebo jsou pohřešovány.

2.2 VÝBĚR SEKUNDÁRNÍCH ZNAKŮ

Aby bylo možné co nejvíce zúžit počet odpovídajících osob danému profilu na základě fotografie, je třeba určit vhodné sekundární znaky, které budou ze zdrojové fotografie extrahovány. Samozřejmě čím více znaků z fotografie získáme, tím efektivnější bude určování identity osoby. Na určení identity má vliv nejen množství získaných informací, ale i kvalita této informace nazývána identifikační hodnota, což znamená, že čím vzácnější je výskyt daného znaku, tím je vyšší jeho identifikační hodnota. U každé fotografie tak získáme kvantitativní a kvalitativní souhrn znaků, jejichž základní rozdělení je na biometrické znaky (vlastní) a ne-biometrické znaky (doprovodné).

Pro výběr vhodných znaků jsem se inspiroval na stránkách Policie České republiky, kde je k dispozici databáze hledaných a pohřešovaných osob, která využívá právě těchto sekundárních znaků, jako jsou: pohlaví, jméno a příjmení, bydliště, státní občanství, věk, výška, barva očí, barva vlasů a zvláštní znamení. Je samozřejmé, že z fotografie nejsme schopni určit všechny znaky, jako například jméno, příjmení, bydliště a tomu podobné. Protože zdrojové fotografie obsahují celou postavu, doplnil jsem seznam znaků ještě o horní část oblečení (bunda, kabát, svetr, triko,…), dolní část oblečení (kalhoty, kraťasy, sukně) a módní doplňky. Po konzultaci s vedoucím práce jsem doplnil seznam ještě o rasu pozorované osoby a zároveň vyřadil výšku postavy z důvodu nesnadného určení, a tím i velké pravděpodobnosti vzniku chybného posouzení.

2.2.1 Seznam vybraných znaků

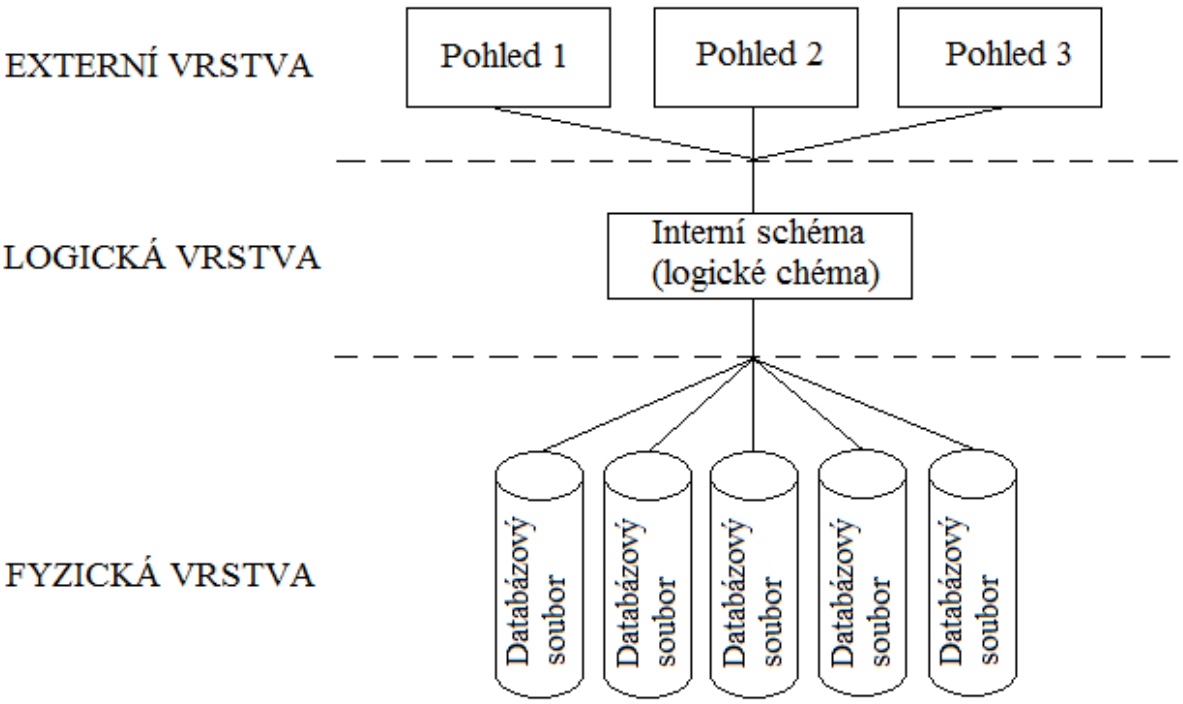
- Pohlaví: - nuž, žena
- Rasa: - europoidní, mongoloidní, negroidní, australoidní
- Stáří - dítě, dospělý, důchodce
- Vlasy - barva
  - délka (krátké, středně dlouhé, dlouhé)
- Oči - barva
- Horní část oblečení - typ (triko, mikina, svetr, bunda, kabát,bikini)
  - barva
- Dolní část oblečení - typ (kalhoty, kraťasy, sukně)
  - barva
- Módní doplňky - druh (klobouk, brýle, hodinky, kabelka, batoh)
- Speciální znaky - druh (jizva, tetování, piercing)

3 DATABÁZE

Databáze je kolekce vzájemně souvisejících dat, s nimiž pracujeme jako s ucelenou jednotkou [5]. Jinými slovy si lze databázi představit jako místo, kde ukládáme potřebná data. Příkladem může být knihovna nebo kartotéka, kde jsou data uložena podle určitého systému. Databáze a její databázový systém neboli Database Management Systems (DBMS), v české literatuře označovaný jako Systém řízení báze dat (SŘBD), slouží k definici ukládaných dat, určení způsobu přístupu k těmto datům a jaké operace je možné s těmito daty provádět. Abychom byli schopni navrhnout databázi, je nutné znát, na jakém principu databáze fungují a zásady nezbytné při postupu jejího návrhu.

3.1 VRSTVY DATOVÉ ABSTKRAKCE

V databázovém systému můžeme každému jednotlivému uživateli, který má přístup k databázi, nabídnout různé pohledy na stejná data, přičemž tyto uživatelské pohledy mohou být vytvořeny podle potřeb každého z nich, ale stále pracují nad jednou společně uloženou kopií dat. V pohledech nejsou však uložena žádná skutečná data, pouze se v nich odrážejí veškeré změny provedené v podkladových databázových objektech a o je možné díky vrstvám abstrakce zobrazeným na obrázku 3.1.



Obr.3.1: Vrstvy abstrakce v databázi [5]

3.1.1 Fyzická vrstva

Fyzická vrstva obsahuje datové soubory, do nichž se ukládají veškerá data příslušné databáze. Podle konkrétního databázového systému může být přitom jedna databáze uložena v několika datových souborech, rozmístěných často na různých fyzických diskových jednotkách. To umožnuje ,že diskové jednotky mohou pracovat souběžně a databáze tak dosáhne vyššího výkonu. Uživatel databáze vůbec nemusí vědět, jakým způsobem jsou data v těchto souborech uložena a nemusí také vědět, ve kterém fyzickém souboru se požadované datové položky nacházejí. [5]

3.1.2 Logická vrstva

Logická vrstva neboli logický model představuje první ze dvou vrstev abstrakce v databázi. Fyzická vrstva totiž skutečně existuje a je realizovaná v konkrétních souborech operačního systému. Na rozdíl od toho je logická vrstva pouze součásti abstraktních datových struktur, které se podle potřeby skládají z objektů fyzické vrstvy. Logická vrstva bývá někdy označována jako pojmem schéma. Podle konkrétního databázového systému muže být schéma tvořeno množinou dvourozměrných tabulek, hierarchickou nebo jinou strukturou. [5]

3.1.3 Externí vrstva

Externí vrstva neboli externím model je druhou z vrstev abstrakce v databázi. Tuto vrstvu tvoří již zmíněné uživatelské pohledy, kterým se souhrnně říká subschéma. V této vrstvě se k databázi připojují uživatelé a aplikační programy, které s ní dále pracují a zadávají v ní dotazy. V externí vrstvě se vytvářejí uživatelské pohledy, tedy souhrn dotazů vykonávajících se současně a ty mohou být předem definovány a uloženy do databáze nebo zde mohou být vytvořeny dočasné, jednorázové dotazy, které se po použití odstraní. [5]

3.2 MODELY DATABÁZÍ

Databázový model neboli schéma je v podstatě architektura, podle které databázový systém objekty do databáze a podle které je vzájemně provazuje. V následující části textu budou vyjmenovány a stručně popsány nejrozšířenější modely databází v pořadí jejich časového vývoje.

3.2.1 Otevřené soubory

Otevřené soubory jsou soubory, v nichž obsažené záznamy sebou nenesou žádné informace, které by do cílové aplikace vnášely strukturu nebo jakékoli vztahy mezi záznamy. Veškeré informace o struktuře dat a jejich významu musí být začleněny do příslušné aplikace, která se soubory pracuje, nebo je musí znát člověk jako cílový uživatel. Otevřené soubory nejsou v podstatě databází jako takovou. Do těchto souboru se často ukládají informace z databází, tzn. metadata, což jsou informace, které popisují, jaká data jsou v databázi uložena. Právě z těchto otevřených souborů vznikly první databázové systémy.[5]

3.2.2 Hierarchický model

První databáze byly postaveny právě na tomto modelu a záznamy v něm jsou uspořádány do určité hierarchie. Každý ze souborů, definovaných v systému otevřených souborů, je zde nahrazen typem záznamu, v hierarchickém názvosloví uzlem. Jednotlivé záznamy jsou zde propojeny pomocí ukazatelů, které obsahují adresu příslušného svázaného záznamu. To znamená, že ukazatel definuje vztah nadřízený-podřízený záznam, jinak řečeno rodič potomek. Jeden rodič může mít více potomků, ale každý potomek může mít pouze jednoho rodiče. [5]

3.2.3 Síťový model

Síťový model vznikl zhruba ve stejné době jako hierarchický model a stejně jako tento model, síťový model propojuje příbuzné záznamy pomocí ukazatelů s fyzickými adresami. Zde jsou tyto vztahy záznamů nazývané vlastník-člen nebo množiny. Je zde však rozdíl oproti hierarchickému modelu a to, že vztahy neboli relace mezi daty jsou zde pojmenovány. Pojmenování relace umožňuje programátorovi nařídit přechod z jednoho záznamu na jiný prostřednictvím právě této konkrétní relace. Síťový model je flexibilnější než hierarchický, ale však za cenu vyšší složitosti. [5]

3.2.4 Relační model

Síťový a hierarchický model jsou nejen složité, ale navíc mají jednu společnou nevýhodu, nejsou příliš přizpůsobivé a to protože efektivní zpracování dat je možné pouze při jejich průchodu po předem definované cestě. V historii vývoje databázi se označuje za velice významný bod zveřejnění výzkumné práce Dr. E. F. Codda, na jejímž základě se zrodil relační model. [5]

V relačním modelu jsou data reprezentována pomocí dvourozměrných tabulek, podobně jako list tabulkového procesoru, ale u relačního modelu nemusí být data nutně

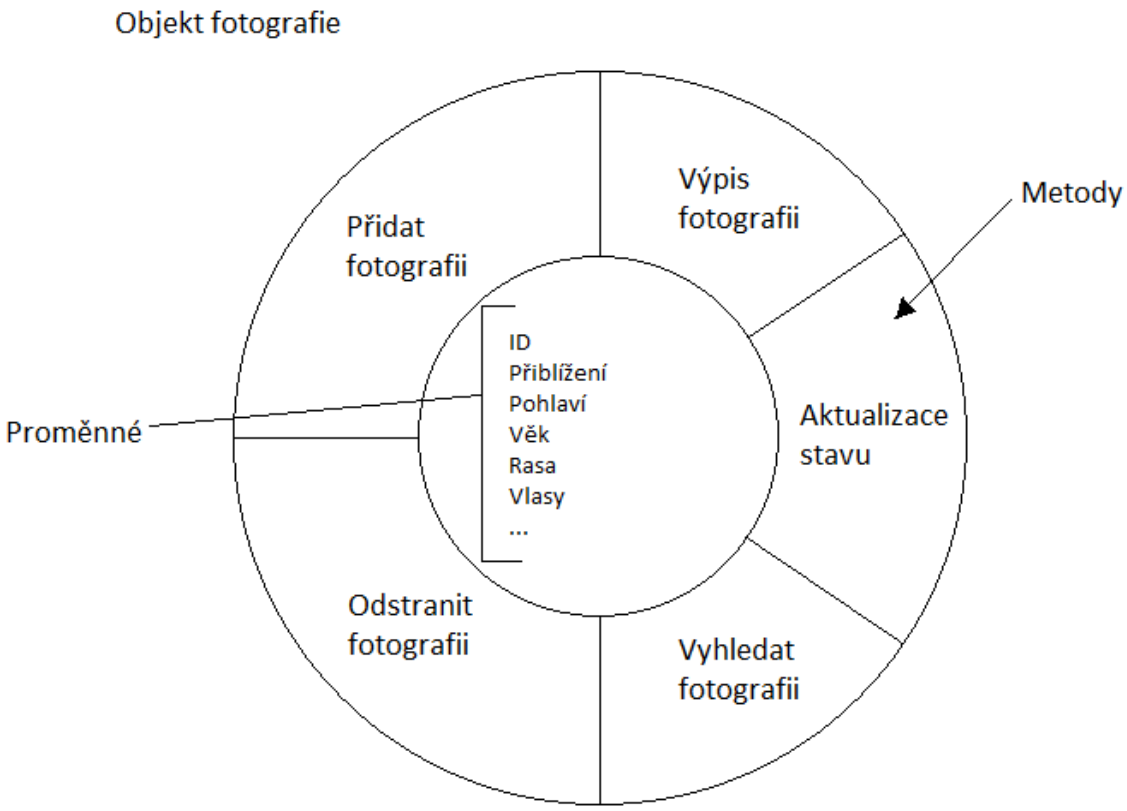
uložena v tabulkové podobě. Model také povoluje slučování neboli spojování tabulek do pohledů, které mají rovněž podobu dvourozměrných tabulek. Tento model obsahuje i dostatečně velké množství fyzické i logické datové nezávislosti. Místo propojování příbuzných záznamů pomocí ukazatelů, jako tomu bylo u hierarchického a síťového modelu se u tohoto modelu zapisuje do každé z tabulek určitá společná datová položka nazývaná primární klíč. Primární klíč je sloupec neboli atribut, který jednoznačně identifikuje každý řádek tabulky. [5]

3.2.5 Objektově orientovaný model

Objektově orientovaný (OO) model se dočkal komerčního využití v devadesátých letech a to z důvodu, kdy tehdejší relační databázové systémy nebyly schopné pracovat se složitějšími datovými typy, jako jsou obrázky, složité výkresy a zvukové či videosoubory. [5]

Pod pojmem objekt se rozumí logické seskupení příbuzných dat a programové logiky, které společně reprezentují nějakou věc nebo osobu z reálného světa. Jednotlivé datové položky, jako je identifikátor nebo jméno, se v tomto modelu nazývají proměnné neboli atributy a jsou uloženy u každého objektu. Nejdůležitější rozdíl mezi objektově orientovaným modelem a ostatními modely je, že se k těmto proměnným může přistupovat pouze pomocí tzv. metod. Metoda je část aplikační programové logiky, která pracuje nad určitým objektem a provádí nad ním danou funkci. [5] Tato vlastnost, která umožňuje přístup k proměnné pouze pomocí metody, se nazývá zapouzdření objektu a je znázorněná kruhovým uspořádáním metod kolem jádra s proměnnými na obrázku 3.2.

Může se zdát, že OO model je neefektivní z důvodu redundantního ukládání metody a definice proměnných zvlášť u každé instance tedy kopii objektu. Ve skutečnosti tomu tak ale není. Objekty jsou uspořádány do hierarchie tříd a společné metody i definice proměnných stačí definovat na jednom místě, odkud je zdědí ostatní členové stejné třídy. [5]



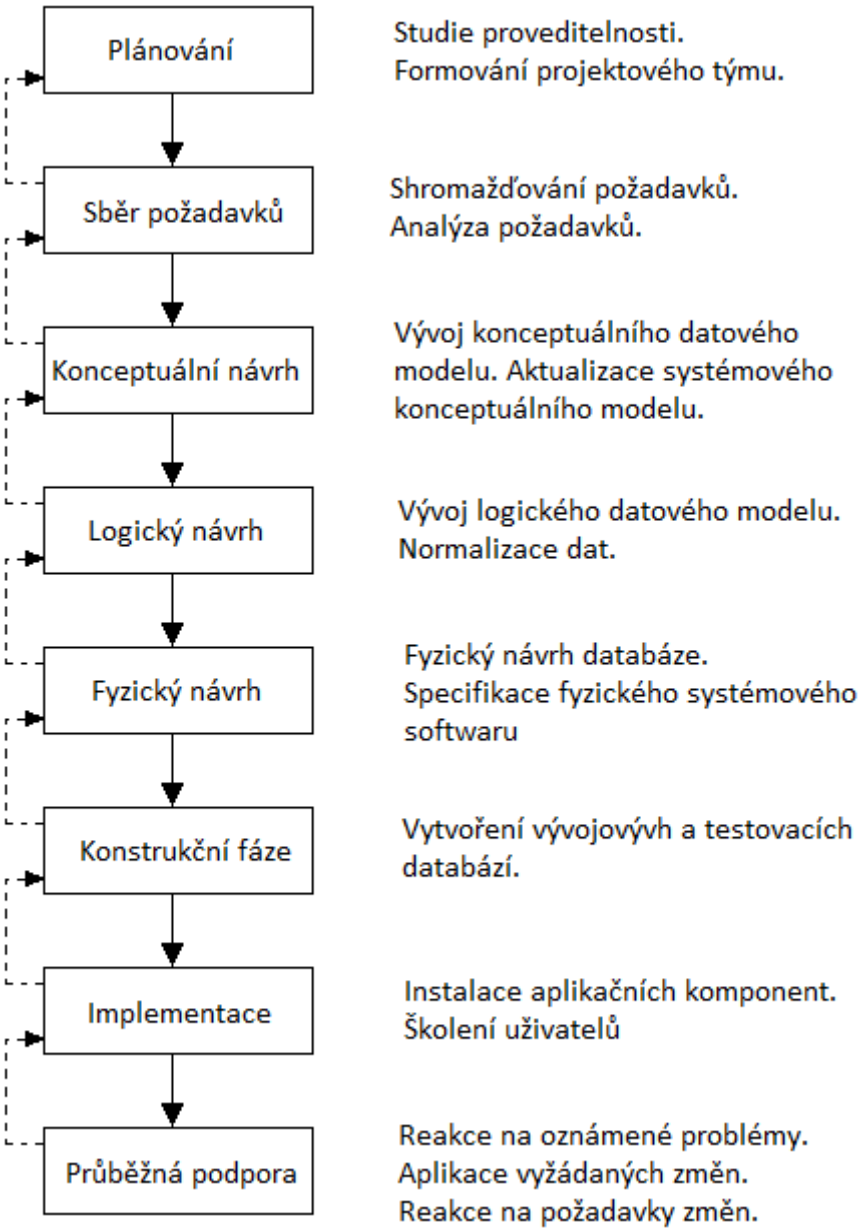
Obr. 3.2: Struktura objektu v objektově orientovaném modelu

3.2.6 Objektově relační model

Objektově orientovaný model má určité výhody a minimalizuje dopady modifikací v systému, ale chybí v něm možnost vytvářet jednorázové tzv. ad hoc dotazy. Někteří výrobci relačních databází však zaznamenali výhody OO modelu a doplnili právě tyto objektové funkce do běžného relačního modelu. Kombinace výhod OO a relačního modelu vedla ke vzniku objektově relačního modelu. [5]

3.3 ŽVOTNÍ CYKLUS SYSTÉMU

Životní cyklus systému, v našem případě databáze, zahrnuje a popisuje veškeré události, které nastanou od okamžiku uvědomění si potřeby databáze, přes celý její vývoj až po nasazení do provozu. Obecný přehled těchto událostí rozdělených do etap se stručným popisem jsou zobrazený na obrázku 3.3



Obr. 3.3: Klasický životní cyklus vývoje systému.[5]

Z obrázku je jasné, že proces vývoje není vždy pouze jednosměrný, tento chod znázorňují čárkované čáry, a to protože například během návrhu zjistíme, že některé informace chybí, takže se musíme vrátit do jiné etapy a vzniklé problémy dořešit.

4 NÁSTORJE REALIZACE

Protože databáze bude pracovat s fotografiemi, což jsou složitější datové typy a výsledná aplikace bude sloužit jako nástroj pro výzkum, je velice pravděpodobné, že s postupem času a získanými novými poznatky, bude zapotřebí tuto databázi upravovat. Z těchto důvodů byl zvolen objektově orientovaný databázový model. Pro grafické znázornění objektově orientovaného řešení použijeme UML diagram. Fyzická realizace bude pak napsána v programovacím jazyce JAVA.

4.1 UML DIAGRAMEM

UML je zkratkou pro Unified Modeling Language, což je grafický jazyk pro vizualizaci, specifikaci, návrh a dokumentaci programových systémů. [6] použití UML jazyka přináší do procesu návrhu systému následující výhody:

- Jednoznačnost
- Jednoduchost
- Srozumitelnost Snadná čitelnost pro člověka
- Je známý po celém světě

Diagramy jsou nejznámější a nejpoužívanější částí standardu. Následuje přehled diagramů v UML včetně jejich rozčlenění do skupin: [6]

4.1.1 Strukturní diagramy:

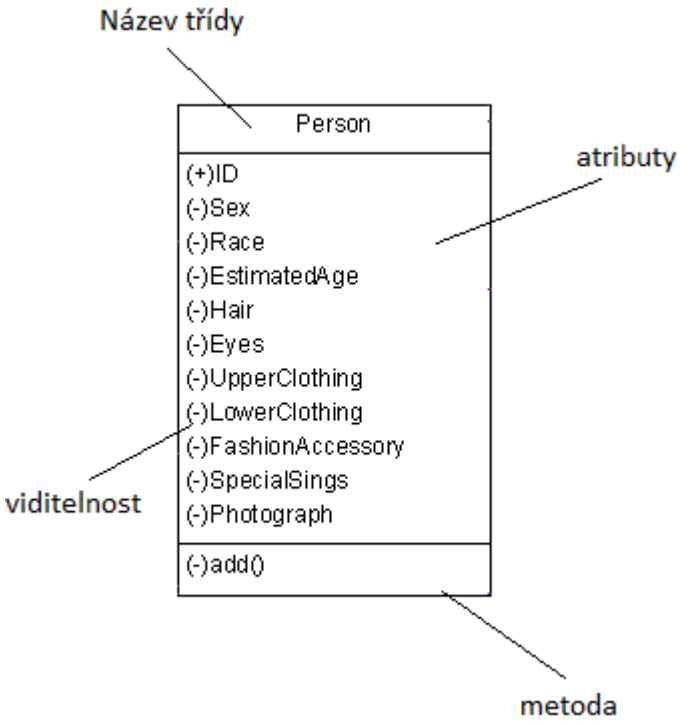
- diagram tříd (class diagram)
- diagram komponent (component diagram)
- composite structure diagram
- diagram nasazení (deployment diagram)
- diagram balíčků (package diagram)
- diagram objektů (object diagram), též se nazývá diagram instancí

4.1.2 Diagramy chování:

- diagram aktivit (activity diagram)
- diagram užití (use case diagram)
- stavový diagram (state machine diagram)
- diagramy interakce:
  - sekvenční diagram (sequence diagram)
  - diagram komunikace (communication diagram, collaboration diagram)
  - interaction overview diagram
  - diagram časování (timing diagram)

4.1.3 UML diagram tříd

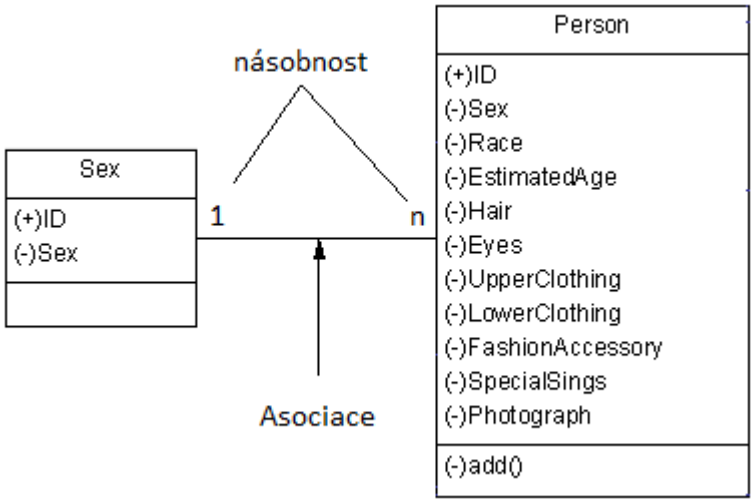
V tomto projektu bude použit pro znázornění daného návrhu pomocí UML diagramu tříd, kde základním stavebním kamenem diagramu tříd je třída samotná. Tato třída je zobrazena na obrázku 4.1, který obsahuje název třídy, atributy této třídy a jejich viditelnost. Dále pak metody a jejich viditelnost.



Obr. 4.1: Třída.

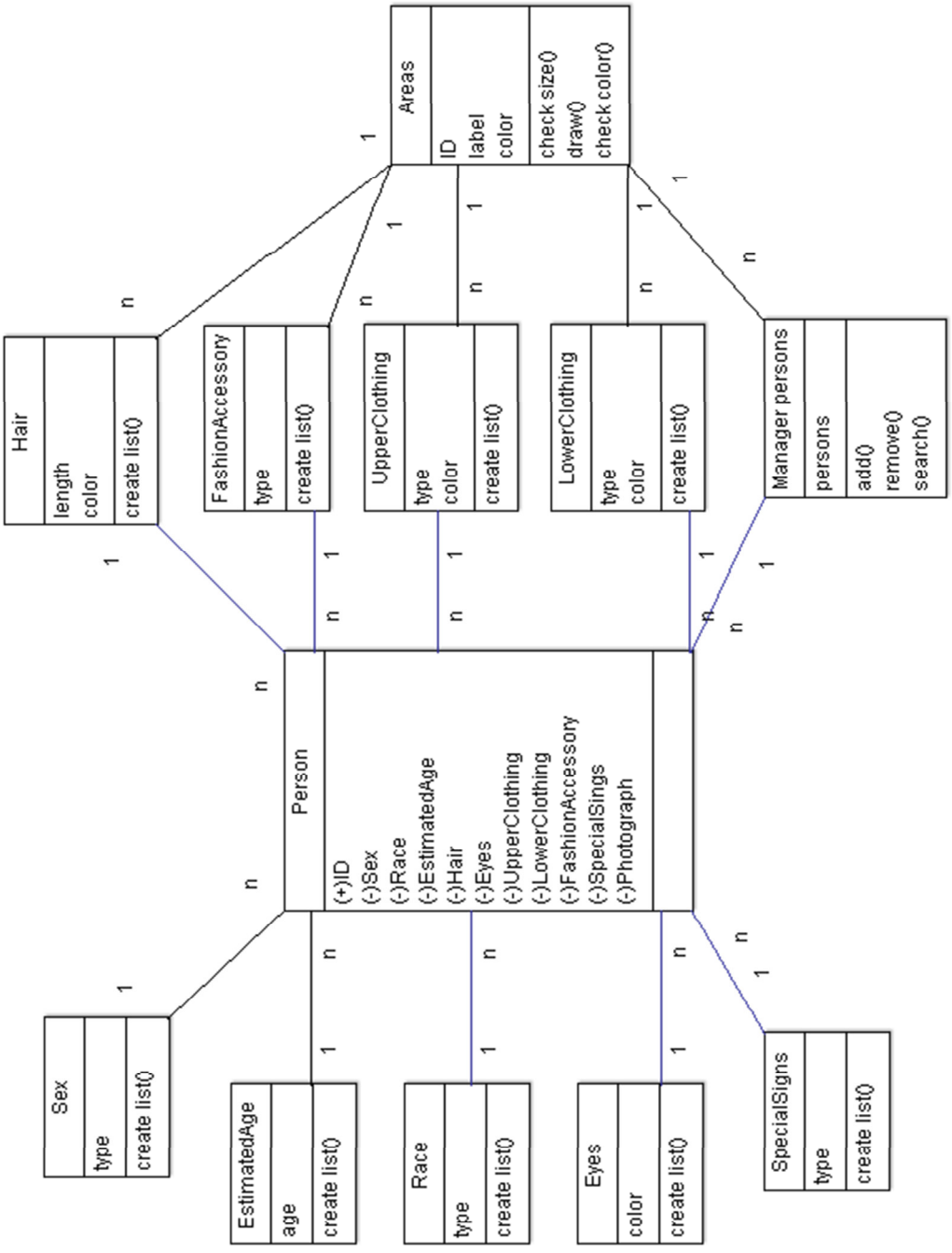
Důležitou součástí UML diagramu tříd je vztah mezi těmito třídami, který se nazývá asociace. Asociace může být orientovaná nebo neorientovaná a také k ní může být přidaná násobnost, jako např.: 1,0..1, 1..n, n. Příklad asociace je zachycen na obrázku 4.2, kde je

obousměrně orientovaná, tedy neorientovaná asociace s násobnosti 1 : n mezi třídou Pohlaví a třídou Osoba. Všimněte si, že informace o násobnosti platí do kříže – tedy, že pro jeden typ pohlaví může odpovídat n osobám, zatímco k fotografii je asociován vždy pouze jeden konkrétní typ pohlaví.



Obr. 4.2: Asociace

Na následujícím obrázku 4.3 je vyobrazeno navržené řešení zadání tohoto projektu. Toto schéma bude tvořit základ pro samotnou fyzickou realizaci.



Obr. 4.3: UML diagram tříd.

4.2 JAVA

Jak již bylo zmíněno na začátku této kapitoly, problém je řešen na základě objektově orientovaného modelu, proto je vhodné použít pro jeho realizaci i objektově orientovaný programovací jazyk. Je použit programovací jazyk JAVA, který je objektově orientovaný a oproti ostatním je nezávislý na operačním systému, což je jeho výhodou.

V této části kapitoly budou situace z kapitoly 4.1, zobrazeny pomocí UML diagramu tříd, realizovány v jazyce JAVA.

4.2.1 Objekt

V reálném světě si pod pojmem objekt můžeme představit například auto, psa, člověka, telefon, kolo apod. Každý objekt může být pospán svým stavem a chováním. Pod pojmem stav u auta si můžeme představit například zařazený rychlostní stupeň, aktuální rychlost a chování je například změna rychlosti, použití brzd apod. Pro každý takový objekt si můžeme položit následující otázky:

1. V jakém stavu může objekt být?
2. Jak se může daný objekt chovat?

Nutné je také poznamenat, že některé objekty mohou obsahovat jiné objekty. Softwarové objekty vychází z reálných objektů. Příklad objektu byl znázorněn na obr. 3.2 v kapitole 3. Softwarový objekt ukládá svůj stav v atributech (proměnných) a jeho chování popisují metody (funkce) [7].

4.2.2 Metoda

Metoda definuje chování objektu, tedy způsob přístupu a nastavení atributu (vlastnosti). Metody v Javě existují dvojího druhu [7]:

- metody třídy (statické metody),
- metody instance,

Každý program aplikace začíná voláním metody main a končí jejím opuštěním. Metoda je deklarována následovně:

```
public static typ_navratove_hodnoty nazev_metody(vstupni_parametry){
      Telo metody;
}
```

Metoda je následně volána pomocí názvu metody a jejich vstupních parametrů.

Existují metody bez návratové hodnoty i metody bez vstupních proměnných. V případě metod bez návratové hodnoty se místo položky `typ_navratove_hodnoty` uvádí klíčové slovo `void`.

Metody lze také přetížit. Přetížené metody mají stejný název s různou hlavičkou. Přetížená metoda se musí lišit ve svých vstupních parametrech a to konkrétně počtem, pořadím nebo typem [7].

Statické metody

Rozdíl oproti klasickým metodám:

- Statická metoda může přistupovat pouze ke statickým proměnným.
- Statickou metodu lze volat i v případě, že neexistuje žádná instance dané třídy.

Konstruktor

Konstruktor je speciální typ metody, který je vždy volán při vytvoření instance třídy (objektu). Má stejný název jako třída a obsahuje inicializaci objektu. Konstruktorů může být v rámci třídy více, musí se však lišit svými parametry [7].

4.2.3 Třída

Existuje velké množství různých objektů, ale spousta objektů má společné vlastnosti a a chovají se stejně. Uvedeme si opět příklad s automobilem, kterých existuje mnoho typů. Každé auto je jiné, obsahuje však stejné typy komponentů. V objektově orientovaném myšlení řekneme, že jedno auto je instance třídy Auto nebo objekt třídy Auto. Třída je tedy základní kámen, ze kterého jsou vytvářeny objekty. Objektů jedné třídy může existovat libovolné množství. Třída je uvozena klíčovým slovem „class“, její identifikátor začíná vždy velkým písmenem a ostatní písmena jsou malá. V případě víceslovných názvů tříd začíná každé slovo velkým písmenem. Tělo třídy je uvozeno složenými závorkami. Příklad implementace třídy je následující [7]:

Třída v jazyce JAVA

Zde je uveden kód programu s popisem odpovídající třídě Osoba z obrázku 4.1

```
public class Person {   //název třídy

      public int ID;          //veřejný atribut
      private Sex sex;        //privátní atribut,
                              //reference na objekt třídy pohlaví
      private Age age;
      private Hair hair;
      private Eyes eyes;
      private Race race;
      private UpperClothing upperClothing;
      private LowerClothing lowerClothing;
      private FashionAccessory fashionAccessory;
      private SpecialSings specialSings;
      private Photograph Photograph;

      private void add(){     //privátní metoda add
      }
```

Asociace v jazyce JAVA

Dále byla zobrazena na obrázku 4.2 neorientovaná asociace s násobností 1:n mezi třídami Pohlaví a Osoba. Pro realizaci asociace v jazyce JAVA existuje možnost využití třídy Vector, která je již její součástí. Jelikož je asociace neorientovaná, platí, že každá strana asociace, tedy obě třídy, bude disponovat referencemi. Reference na objekt třídy pohlaví již byla znázorněna v předcházející ukázce. Následuje ukázka třídy Pohlaví a reference na třídu Osoba.

```
import java.util.Vector;            //import třídy vector

public class Sex {
      private Vector<Person> partPerson = new Vector<Person>();
                                    //reference na objekty třídy osoba
      private String type;

      public static void main(String[] args) {

         Sex s1 = new Sex();  //vytvoření objektu třídy pohlaví
         s1.type = "Man";

         Sex s2 = new Sex();
         s2.type = "Woman";
      }
}
```

Vector je obdobou dynamického pole. Jediným rozdílem je, že je synchronizovaný. Synchronizovanost znamená, že lze tuto kolekci mohou využívat najednou z více vláken. Vector je zastaralá kolekce, proto bychom ji neměli v nových programech vůbec používat. Jelikož výsledná aplikace bude využívat pouze jedno vlákno a synchronizace tedy není potřebná, nahradíme pole typu Vector polem ArrayList. Jedná se o seznam postavený nad polem (dynamické pole). Objekt této třídy představuje nejpoužívanější typ kolekce, na kterou se přechází, přestávají-li nám stačit klasická pole [8].

4.3 JAVAFX

JavaFX je platforma pro vývoj tzv. bohatých internetových aplikací (Rich Internet Application – RIA). S využitím této technologie je možné vytvářet a jednoduše nasazovat interaktivní webové aplikace, kde je důraz kladen na přístupnost a jednoduchou použitelnost těchto aplikací z hlediska uživatele (tzv. UX – User eXperience). JavaFX poskytuje velkou škálu grafických a multimediálních rozhraní (API). Součástí JavaFX je vývojové i běhové prostředí, které umožňuje vývoj jak jednoduchých, tak rozsáhlých (tzv. enterprise) aplikací. JavaFX zaručuje, že aplikace vyvinuté na této platformě se budou chovat stejně na všech dnes běžně dostupných operačních systémech [7].

Ve verzích nižších než 2.0 bylo nutné využívat speciální jazyk (JavaFX skript) pro psaní JavaFX kódu. S verzí 2.0 bylo celé API importováno přímo do Javy, to znamená, že je možné je volat přímo z jazyka Java (k aplikaci se připojí v podobě knihovny). Mezi klíčové vlastnosti JavaFX ve verzi 2 patří [7]:

- Java API pro JavaFX umožňuje využívat všechny vlastnosti jazyka Java (anotace, generické programování apod.) při tvorbě JavaFX aplikací,
- Java API pro JavaFX je navrženo tak, aby jej bylo možné využívat i s jinými jazyky využívající virtuální stroj jazyka Java (JRuby, Scala apod.),
- Popdora moderních grafických jednotek (GPU – graphics processing unit),
- Podpora hardwarově akcelerovaného vykreslování grafiky,
- Jazyk FXML založený na XML (eXtensible Markup Language), který slouží pro definici grafického uživatelského rozhraní,
- Podpora přehrávání multimediálního obsahu na webu i v běžných aplikacích. Multimediální engine je založen na frameworku GStreamer,
- Velké množství ovládacích prvků jako Menu, Tabulky, Grafy.

5 VLATNÍ NÁVRH

Pro návrh databáze je dobré vědět, jak bude výsledná databáze používaná a co by měla umět. Při diskuzi s vedoucím projektu jsem byl seznámen s tím, že tato databáze bude uchovávat data ve formě fotografií, extrahované oblasti z fotografie odpovídající vybraným sekundárním znaků a popis těchto znaků. Všechny tyto informace mají být svázány se zdrojovou fotografií a zároveň podle těchto informací bude existovat možnost vyhledávat a zobrazovat odpovídající fotografie. Dále tato databáze bude obsahovat algoritmy sloužící k automatické definici pozice sekundárního znaku v obraze a automatickému určení barvy s možností ruční korekce, tohoto znaků.

5.1 GRAFICKÉ UŽIVATELSÉ ROZHRANÍ (GUI)

Termín GUI je zkratkou pro grafické uživatelské rozhraní (graphical user interface). Prakticky každý souborový program používá pro interakci s uživatelem grafiku. GUI je totiž intuitivní a efektivní způsob, jak shromažďovat informace od uživatele nebo mu je zobrazovat ke čtení.

Za vznikem GUI stojí vývojáři institutu Palo Alto Research Center firmy Xerox. Jejich dílo bylo později vylepšeno firmou Apple Computer a nakonec i společností Microsoft. Srdcem GUI jsou standardní grafické prvky, které dohromady vytvářejí uživatelské rozhraní. Těmto prvkům se obvykle říká okna, nabídky, příkazová tlačítka, popisky, textová okna, přepínače a další podobné objekty rozhraní GUI, s nimiž se setkáváme téměř ve všech současných programech. Programovací jazyk Java nám pro tyto účely nabízí dvě standardní grafická prostředí [9]:

- AWT (obsažené v balíčku java.awt), které se objevilo jako první z javovských prostředí a v současné době již není ve větší míře využíváno
- Swing (balík javax.swing), který představuje nástupce dnes již zastaralého AWT. Toto prostředí značně rozšiřuje možnosti komponent AWT a navíc přidává velkou porci speciálních komponent, které nebyly v AWT obsaženy

Swing se dá také považovat za starší grafické prostředí, poté co společnost Oracle oznámila, že ho má v budoucnu nahradit již zmíněná JavaFX. Proto má nové aplikace smysl programovat již jen v moderním vývojové prostředí JavaFX a tak to i bude v této práci.[10]

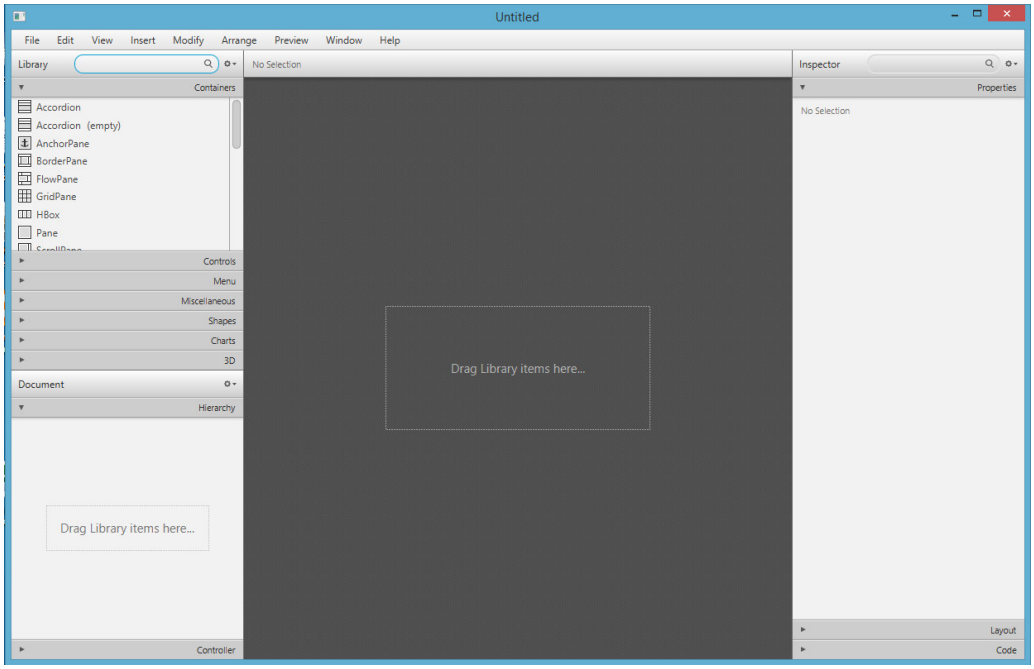
5.1.1 FXML

V JavaFX je možné vyvíjet podobně jako ve starším Swingu, tedy tak, že tvoříte instance jednotlivých formulářových prvků (tlačítko, textové pole). Ty poté vkládáte do tzv. layoutů, což jsou kontejnery na formulářové komponenty.

Druhým způsobem, který budeme pro vytvoření hlavního okna používat, je FXML. FXML je jazyk pro návrh formulářů. Asi vás podle názvu nepřekvapí, že je to další jazyk odvozený z XML. Použití XML pro návrh prezentační vrstvy aplikace (GUI) není nic nového, naopak se jedná o osvědčený princip z webových aplikací.[10]

5.1.2 Scene Builder

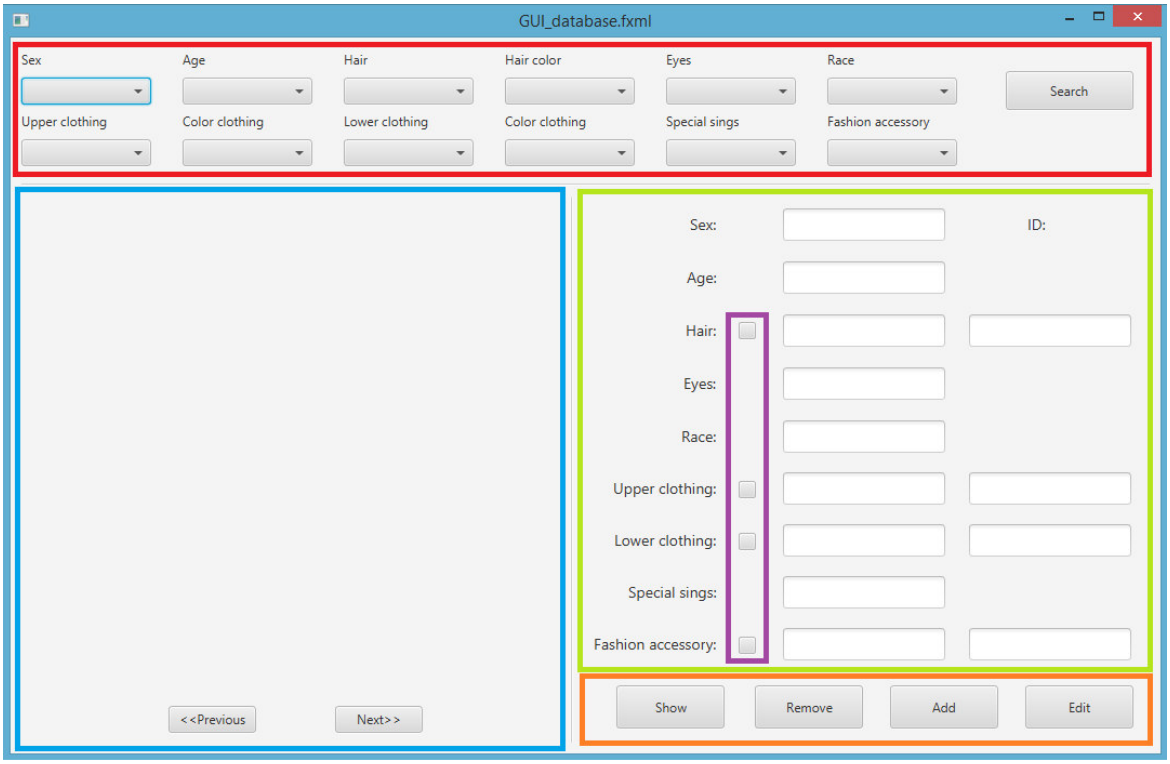
Jedná se o vizuální nástroj podporující technologii drag&drop pro vytváření grafického uživatelského rozhraní v JavaFX aplikacích bez potřeby psaní kódu. Uživatel pouze „přetahuje“ komponenty na pracovní plochu, upravuje jejich vlastnosti a aplikuje různé styly, zatímco se v pozadí generuje výsledný FXML kód, který lze samozřejmě upravovat. Manuální změny jsou ihned viditelné v návrhovém zobrazení. Scene Builder je napsána jako JavaFX aplikace a ukazuje tak sílu a eleganci právě této nové platformy, která je od vydání verze Java 8 součástí standardní Java knihovny. Nástroj je určen pouze pro vytváření aplikací postavených na JavaFX platformě, není tedy možné vyvíjet grafické rozhraní pro jiné jazyky[10]



Obr. 5.1: Scene Builder 2.0

5.1.3 Hlavní okno aplikace

Na obrázku 5.2 je vyobrazeno hlavní okno aplikace vytvořené pomocí nástroje Scene builder 2.0. V obrázku jsou barevně zvýrazněny jednotlivé oblastí provádějící určité úkony.



Obr. 5.2: Hlavní okno aplikace.

- Červená oblast: obsahuje komponenty (Combobox), které budou sloužit k zadávání parametrů a komponentu (Button) pro vyhledávání osob odpovídajících těmto parametrům.
- Modrá oblast: obsahuje komponentu (ImageView), ve které se zobrazí fotografie odpovídající osoby, dvě tlačítka (Button) s funkcí přepínání mezi jednotlivými fotografiemi (osobami) vpřed a vzad.
- Zelená oblast: slouží pro zobrazení informací o jednotlivých sekundárních znacích dané osoby. Komponenty (Label) na levé straně této oblasti obsahují názvy znaků a do zbývajících komponent (TextField) se vypisují informace.
- Fialová oblast: obsahuje čtyři komponenty (CheckBox), jejichž výběrem řekneme aplikaci, jaké znaky se ve fotografii zvýrazní.
- Oranžová oblast: obsahuje tři komponenty (Button) pro přidání a odebrání osoby z databáze a zobrazení fotografie v plné velikosti se zvýrazněnými znaky. Poslední komponenta v této oblasti (ToggleButton) umožní uživateli editaci barvy a následné uložení provedených změn.

5.2 OBSLUŽNÁ LOGIKA

Samotné grafické rozhraní tvoří pouze prezentační vrstvu a zatím nic neumí, protože jednotlivým komponentám ještě nebyla přiřazena žádná logika. To se provádí v, k tomu vytvořené, třídě MyController, kde jsou vytvořeny komponenty s anotací @FXML a metody, které se provedou při aktivaci dané komponenty. Propojení této třídy a FXML dokumentu se provádí v nástroji Scene Builder.

5.2.1 Přidávání a odebírání

Výsledná aplikace bude schopna načítat nové osoby a zapisovat je do souboru tak, aby při opuštění aplikace a jejím opětovném spuštění již načtená data zůstala uložena a nemusela se načítat znova.

Načtení nové osoby do aplikace se provede stisknutím tlačítka „Add“. Dojde k vytvoření nového dialogového okna, které slouží právě pro načtení a editaci nové osoby. Po stisku tlačítka „OK“ v dialogovém okně, se navolená osoba přidá do seznamu osob a zapíše do souboru k tomu určenému.

Pro tyto účely byla vytvořena třída ManagerPerson, kde seznam osob je jejím atributem. Tato třída obsahuje také metody pro přidávání a odebírání osoby do tohoto seznamu.

Výpis kódu 5.1: Třída ManagerPerson

```
public class ManagerPersons {
//atribut seznam osob
  private ObservableList<Person> persons =
                              FXCollections.observableArrayList();
//metoda pro přístup k seznamu osob
  public ObservableList<Person> getPersons() {
      return persons;   }
//metoda se vstupním argumentem osoba, která se přidá do seznamu
  public void addPerson(Person person){
       persons.add(person);   }
//metoda se vstupním argumentem osoba, která se odebere ze seznamu
  public void removePerson(Person person){
       persons.remove(person)
}
```

Vytvoření nového dialogového oka má na starosti třída DialogNewPerson, která dědí ze třídy Stage a obsahuje atribut osoba typu Person. Protože toto dialogové okno je mnohem jednodušší než hlavní okno aplikace, není vytvořeno pomocí FXML a Scene Builderu, ale pomocí instancí objektů jednotlivých komponent. Třída DialogNewPerson tedy obsahuje

metody pro vytvoření nové scény, vyvolání průzkumníka souborů (filechooser), konstruktor se vstupním argumentem typu Window a metodu pro automatické určení (nalezení) nevyužité hodnoty ID. Výpis kódu poslední zmíněné metody je uveden níže. Při stisku tlačítka „OK“ dojde k uložení navolené osoby do atributu třídy a uzavření dialogového okna. Po uzavření dialogového okna se předa atribut osoba metodě addPerson() a ta tuto osobu uloží do seznamu.

Výpis kódu 5.2: Zjištění nevyužitého ID

```
public int addID(){
      int ID = 0;
//vytvoření seznamu obsahující ID všech osob v seznamu
      ArrayList<Integer> ar = new ArrayList<>();
      for (int i = 0; i < ManagerPersons.getPersons().size(); i++) {
            ar.add(ManagerPersons.getPersons().get(i).getId());
            }
//přirozené seřazení seznamu s ID
      ar.sort(null);
//cyklus prohledávající pole s ID
      for (int i = 0; i < ar.size(); i++) {
            if (ar.get(i)!=i){ //pokud se během cyklu najde nevyužité ID
                  ID = i;      //uloží toto ID
                  break;       //ukončí cyklus
                  }
//pokud po ukončení cyklu nebylo nalezeno volné ID
//nastaví hodnotu ID na nejbližší volné
            ID = ar.size();
            }
            return ID;         //vrátí ID
      }
```

Součástí metody vytvoření scény je tlačítko Browse, které slouží pro načtení fotografie a upravení a uložení cesty k této fotografii. Úprava cesty se provádí proto, aby aplikace byla schopna načíst fotografie uložené v adresáři aplikace i na jiných zařízeních než pouze v zařízení kde byly fotografie do aplikace nahrány. Fotografie budou uloženy v adresáři aplikace a každá pak ve své složce, proto je potřeba upravit cestu k souboru tak, aby obsahovala pouze název souboru, rodičovský adresář a název balíčku aplikace.

Výpis kódu 5.3: Načtení a uložení fotografie z externího zdroje.

```
//vytvoření tlačítka, nastavení jeho parametrů a přiřazeni posluchače
Button browsePhoto = new Button("Browse");
browsePhoto.setMinWidth(120);
browsePhoto.setOnAction(new EventHandler<ActionEvent>(){
```

```
//obslužná metoda tlačítka
    @Override
   public void handle(ActionEvent e) {
//vyvolání průzkumníka a uložení vybraného souboru do objektu file
   File file = createChooser().showOpenDialog(cmb1.getScene().getWindow());
//pokud byl vybrán soubor, provede upravení a uložení cesty k souboru
        if (file != null){
//zjištění cesty k rodičovskému adresáři
            String level1 =file.getParent();
//zjištění cesty k rodičovskému adresáři předchozího adresáře
            String level2 = new File(level1).getParent();

//nahrazení části cesty k souboru řetězcem /application a zaměnění znaku
//„\“ za znak „/“
      String path =file.getPath().replace(level2,
                                          "/application").replace("\\","/");
//nastavení výsledného řetězce do komponenty photoLabel
            photoLabel.setText(path);
//vytvoření obrázku a zobrazení v dialogu New Person
            Image obr = new Image(getClass().getResourceAsStream(path));
             image.setImage(obr);
            }
        }
   });
```

Odebrání osoby se provede stisknutím tlačítka „Remove“. Provede se zjištění pomocí ID osoby, jaká osoba je zrovna zobrazena v aplikaci a respektive jaká osoba má být odebrána ze seznamu osob. Tato osoba je pak předaná metodě removePerson(), která tuto osobu ze seznamu odebere.

5.2.2 Zápis a čtení souboru

K tomu, abychom mohli do souboru něco **zapsat**, musíme si nejprve vytvořit proud pro výstup do souboru. Souborový výstupní proud otevírá soubor nebo vytváří nový, pokud soubor neexistuje. Jakmile je souborový výstupní proud otevřen, lze data do souboru zapisovat za použití třídy nazvané PrintWriter.

Soubor otevřeme pomocí konstruktoru třídy FileOutputStream, do kterého předáme název souboru, jenž se má otevřít. Pokud se soubor nenachází v aktuálním adresáři, musíme k němu specifikovat plnou cestu. Konstruktor vrátí odkaz na souborový výstupní proud. Dále je potřeba vytvořit instanci třídy PrintWriter. K tomu použijeme její konstruktor, do kterého předáme jako argument odkaz na výstupní proud. Konstruktor vrátí odkaz na instanci třídy PrintWriter. Odkaz budeme používat pro volání metod, které zapisují do souboru. Jakmile se zápisem do souboru skončíme, musíme zavolat metodu close(), kterou soubor uzavřeme.

Požadavek na aplikaci je aby zapisovala osoby a k ní přiřazené informace. Jedná se tedy o zápis objektu do souboru. Abychom mohli zapisovat celé objekty, tedy i s jejími atributy, musíme do třídy daných objektů implementovat rozhraní Serializable, které dovoluje převádět instance třídy na proud bajtů, který lze posléze zapisovat na disk nebo přenášet po síti. Soubor se pak otevírá stejně, jak již bylo zmíněno výše, za pomoci konstruktoru třídy FileOutputStream, kterému předáváme jméno souboru. Tento odkaz se však nepředává do konstruktoru třídy PrintWriter, ale protože chceme ukládat objekty, musíme předat odkaz na výstupní proud do konstruktoru třídy ObjectOutputStream, která definuje metody používané při zápisu objektu do souboru. Jakmile máme instanci třídy ObjectOutputStream, jsme připraveni zapsat objekt do souboru. K tomu je potřeba zavolat metodu writeObject() a předat do ní odkaz na instanci objektu, kterou chceme ukládat do souboru. Statické proměnné objektu se neukládají. Nakonec opět zavoláme metodu Close() a soubor uzavřeme.[9]

Výpis kódu 5.4: Metoda write() provádějící zápis jednotlivých prvků ze seznamu osob, tedy objektů třídy Person do souboru Person.dat.

```
public void write(){
   try {
//otevření souborového výstupního proudu do souboru Person.dat
      ObjectOutputStream vystupPerson =
            new ObjectOutputStream(new FileOutputStream("Person.dat"));
//cyklus postupně zapisuje objekty ze seznamu osob persons
      for (int y = 0; y < ManagerPersons.getPersons().size(); y ++){
      vystupPerson.writeObject(ManagerPersons.getPersons().get(y));
                  }
//uzavření výstupního proudu
            vystupPerson.close();
            }
//zachycení výjimky a její vypsaní do konzole
    catch (IOException v) {
            System.out.println(v);
            }
      }
```

Všechny příkazy související s otevřením souborového výstupního proudu a se zápisem do souboru máme umístěny v testovacím bloku try. Pokud by se při těchto operacích vyvolala nějaká výjimka, jako třeba v případě nedostatečného místa na disku, zachytíme ji v odchytávacím bloku catch.

Čtení objektu ze souboru se provádí stejně jako jeho zápis. Místo tříd FileOutputStream a ObjectOutputStream však používáme třídy FileIiputStream a ObjectIiputStream. Objekt načteme voláním metody readObject(), kterou definuje třída ObjectIiputStream. Metoda readObject() vrací instanci třídy Object.. Tuto třídu pak konvertujeme na specifickou třídu uloženého objektu.

Výpis kódu 5.5: Metoda fill() provádějící čtení objektu ze souboru Počet.dat a ze souboru Person.dat a přidání tohoto objektu do seznamu osob.

```
public void fill(){
   try {
//otevření souborového vstupního proudu ze souboru Pocet.dat
      ObjectInputStream vstupPocet =
                  new ObjectInputStream(new FileInputStream("Pocet.dat"));
//uložení přečtených dat do proměnné pocet
      int pocet = vstupPocet.readInt();
//uzavření vstupního proudu
      vstupPocet.close();
//otevření souborového vstupního proudu ze souboru Person.dat
      ObjectInputStream vstupPerson =
                  new ObjectInputStream(new FileInputStream("Person.dat"));
//cyklus postupně čte objekty ze souboru, konvertuje je na objekty třídy
//Person a přidává do seznamu osob
      for (int i = 0; i < pocet; i++) {
            ManagerPersons.addPerson((Person) vstupPerson.readObject());
                  }
//uzavření vstupního proudu
      vstupPerson.close();
            }
//zachycení výjimky a její vypsání do konzole
  catch (Exception v){
      System.out.println(v);
      }
}
```

5.2.3 Vyhledávání

Zadáním práce je návrh databáze a každá databáze musí umět uchovávat data a taky v těchto datech vyhledávat. Naše aplikace provádí vyhledávání za pomoci atributu ID objektu třídy Person, který má každá osoba jiný, tedy je jejím jedinečným identifikátorem.

Při stisku tlačítka „Search“ se vytvoří seznamy pro jednotlivé znaky obsahující informace (daný atribut) o každé uložené osobě. Poté se vyvolá metoda searchPerson(), kde jsou vstupními argumenty právě vytvořený odpovídající seznam jednoho atributu a hodnota vybraná pomocí comboboxu sloužícímu právě k definici požadavku k vyhledávání. Metoda searchPerson() vrací seznam obsahující již pouze ID osob odpovídajících zadané informaci.

Pokud je zadáno více informací o vyhledávané osobě, provádí se proces vyhledávání pro každý zadaný znak zvlášť a výsledné seznamy s odpovídajícími ID se mezi sebou porovnají a do konečného seznamu se uchovají pouze hodnoty obsažené ve všech seznamech.

Výpis kódu 5.6: Metoda searchPeron() porovnávající vstupní argumenty, tedy seznam a informace, a vytváří seznam odpovídajících ID.

```
public ArrayList<Integer> searchPerson(String combo, ArrayList<String>
                                                              atributList){
//vytvoření návratového seznamu ar a pomocného seznamu arAll
      ArrayList<Integer> ar = new ArrayList<>();
      ArrayList<Integer> arAll = new ArrayList<>();
//naplnění pomocného seznamu arAll atributem ID všech osob v seznamu
      for (int i = 0; i < ManagerPersons.getPersons().size(); i++) {
                  arAll.add(ManagerPersons.getPersons().get(i).getId());
                        }
//zjištění zda byla v comboboxu vybrána hodnota
      if (combo!=null && combo!=""){
//cyklus porovnávající hodnoty ze vstupního seznamu a comboboxu
            for (int j = 0; j < ManagerPersons.getPersons().size(); j++) {
//zjištění zda prvek ve vstupním seznamu obsahuje informaci
                  if(atributList.get(j)==null){
                                    }
//pokud ano provede se porovnání s hodnotou z comboboxu, při shodě se do
//do seznamu ar uloží hodnota ID osoby vlastnící porovnávaný atribut
                  else if (atributList.get(j).matches(combo)){
//při shodě se do seznamu ar uloží hodnota ID osoby vlastnící porovnávaný
//atribut
      ar.add(ManagerPersons.getPersons().get(j).getId());
                        }
                  }
            }
//pokud v comboboxu nebyla vybrána hodnota uloží se do seznamu ar celý
//seznam arAll pro další správné fungování procesu porovnávání s ostatními
//seznamy
      else  ar.addAll(arAll);
//vrací seznam ar
return ar;
      }
```

5.3 DEFINICE POZICE V OBRAZE

Pro účely automatického určení pozice a následné zvýraznění znaku v obraze, byla provedena příprava vstupních dat, tedy fotografií. Pro jednotlivé znaky byly extrahovány, z původní fotografie (a), obrazová data obsahující pouze daný znak (b) v prostředí Adobe Photoshop CS5. Obrazová data pro znak: Horní část oblečení, jsou uvedena na obrázku 5.3.

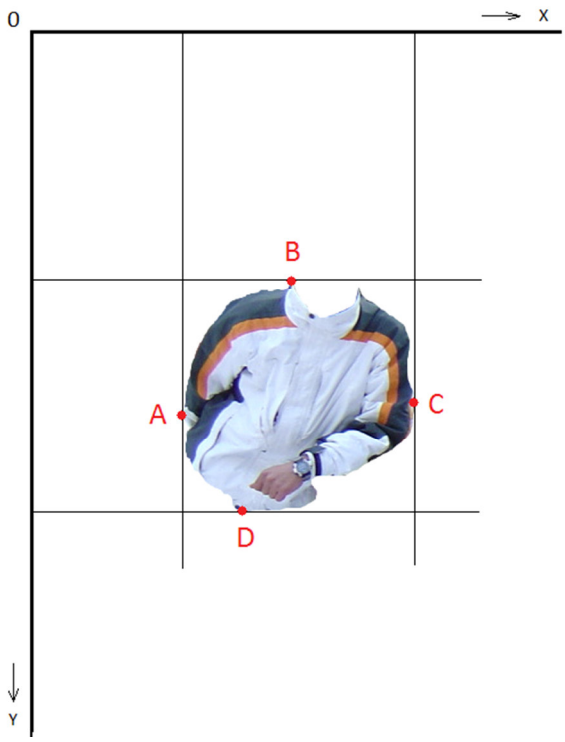
Obr. 5.3. a) Původní fotografie

Obr. 5.3. b) Horní část oblečení

Aplikace provádí definici pozice znaku v obraze ve dvou krocích. Prvním krokem je zjištění pozice extrahovaného znaku v původní fotografii a druhý krok provede vykreslení barevného ohraničení kolem tohoto znaku v původní fotografii.

5.3.1 Zjištění pozice znaku

Zjištění pozice znaku v obraze provádí algoritmus, který obsahuje metoda check_size(). Algoritmus prochází pixely fotografie prvně sloupec po sloupci a když narazí na první barevný pixel, uloží souřadnici sloupce do proměnné A. Algoritmus pokračuje v procházení sloupců, ale tentokrát kontroluje za je celý sloupec prázdný. Souřadnice prvního prázdného sloupce se uloží do proměnné C. Po zjištění těchto dvou hodnot, algoritmus zná šířku a pozici obrazu na ose x. Zjištění hodnot výšky a pozice obrazu na ose y se provádí analogicky s tím rozdílem, že se neprocházejí sloupce ale řádky. Výstupem algoritmu jsou tedy hodnoty uložené A a C, které odpovídají krajním hodnotám fotografie na ose x a hodnoty B a D odpovídající krajním hodnotám na ose y. Pozice těchto bodů jsou znázorněny na obrázku 5.4.



Obr. 5.4: Pozice bodů A, B, C, D v obraze

Výpis kódu algoritmu zjišťujícího hodnoty A a C potřebné pro výpočet šířky a pozice znaku v obraze na ose x, je uveden níže. Hodnoty B a D se určují analogicky, jak bylo popsáno výše.

Výpis kódu 5.7: Algoritmus zjišťující pozici v obraze.

```
//načtení obrazových dat daného znaku do obrázku im
Image im = new Image(getClass().getResourceAsStream(url1));
//zjištění barvy pixelu v bodě [0, 0]
      Color pr = im.getPixelReader().getColor(0, 0);

//cyklus procházející jednotlivé sloupce obrazu
//jedná se o pixel bez bary
      for (int x = 0; x < im.getWidth(); x++) {
//cyklus kontrolující jednotlivé pixely ve sloupci
            for (int y = 0; y < im.getHeight(); y++) {
//zjištění bary daného pixelu
                  Color c =im.getPixelReader().getColor(x, y);
//zjištění zda se jedná o barevný pixel
                  if (c.equals(pr)==false && A == 0){
//pokud je pixel barevný a zároveň se jedná o první barevný pixel
//uloží se hodnota sloupce do proměnné A a vnitřní cyklus se ukončí
                        A = x;
                        break;}
//zjištění zda je pixel barevný a zároveň jestli se nejedná o první barevný
//pixel
                  else if(c.equals(pr)==false && A !=0){
```

```
//pokud je podmínka splněna ukončí se vnitřní cyklus
                        break;
                    }
//při dokončení vnitřního cyklu se do proměnné konec uloží pozice řádku
                    konec = y;
                    }
//zjištění zda již byla identifikovaná pozice A a zároveň jestli
//prohledávání sloupne došlo až do konce, neobsahoval tedy žádný barevný
//pixel
            if(A!=0 && konec==im.getHeight()-1){
//pokud je podmínka splněna uloží se hodnota sloupce do proměnné C
// a ukončí se vnější cyklus
                    C = x;
                    break;
            }
      }
```

5.3.2 Vykreslení ohraničení

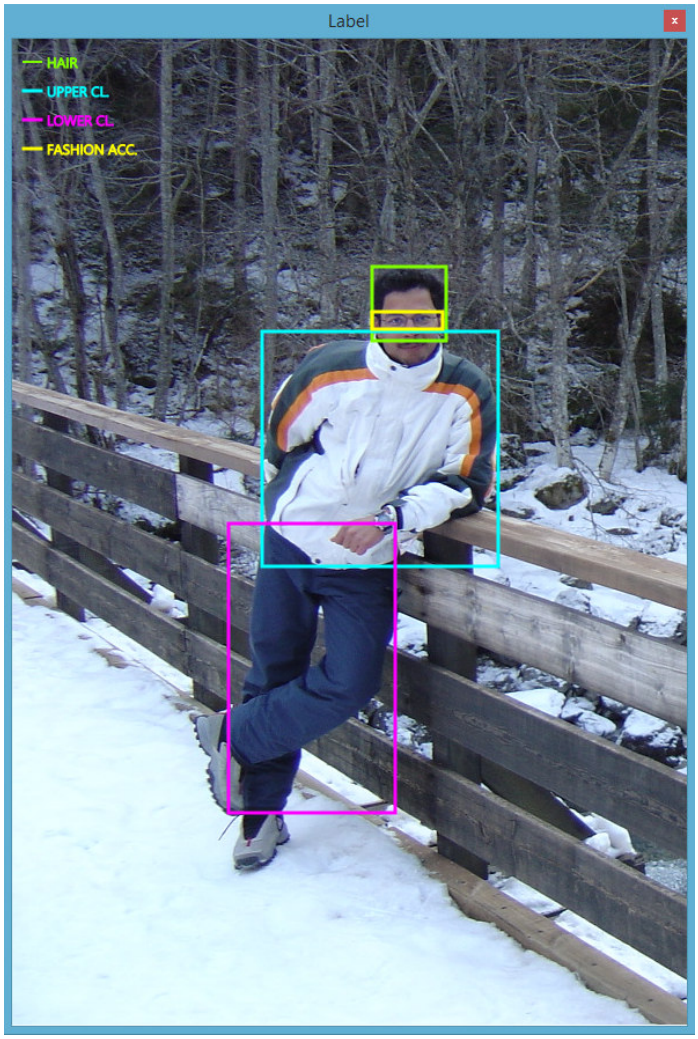
Ohraničení vybraného znaku se provádí vykreslením barevného obdélníku do původní fotografie. Protože pozice v obraze obrazových dat jednotlivých znaků je stejná jako v původní fotografii, pro určení pozice a velikosti vykreslovaného obdélníku se využívá hodnot získaných metodou check_size().

Pro kreslení využíváme komponentu Canvas. Jedná se o plátno, na které můžeme kreslit pomocí grafického kontextu obsahujícího metody pro vykreslení velkého množství tvaru a metody pro nastavení parametrů kreslení jako je tloušťka a barva čáry, font vykresleného textu a další.[11]

Výpis kódu 5.8: Vykreslení barevného ohraničení vybraného znaku.

```
//načtení původní fotografie
Image im1 = new Image(getClass().getResourceAsStream(url));
//vytvoření komponenty canvas a nastavení jeho rozměrů
Canvas canvas = new Canvas(im1.getWidth(), im1.getHeight());
//vytvoření grafického kontextu a jeho přiřazení komponentě canvas
GraphicsContext gc = canvas.getGraphicsContext2D();
//vykreslení původní fotografie
gc.drawImage(im1, 0, 0, im1.getWidth(), im1.getHeight());
gc.setLineWidth(1);                 //nastavení tloušťky čáry
gc.setStroke(Color.CHARTREUSE);     //nastavení barvy čáry
gc.strokeText("HAIR", 30, 25);      //výpis textu na daných souřadnicích
gc.setLineWidth(3);                 //nastavení tloušťky čáry
gc.strokeLine(10, 20, 25, 20);      //kreslení čáry na daných souřadnicích
//vykreslení obdélníku s využitím hodnot zjištěných metodou check_size()
gc.strokeRect(A, B, C-A, D-B);}
```

Po stisku tlačítka „Show“ se provede zjištění, jaké znaky se mají zvýraznit a to kontrolou vybraných checkboxu. Provede se zjištění velikosti a pozice znaku v obraze a pomocí těchto hodnot se v komponentě canvas vykresli do původní fotografie obdélník tvořící barevné ohraničení tohoto znaku. Celý obsah se pak zobrazí v novém okně s velikostí odpovídající původní fotografii. Výsledek tohoto procesu s vybranými všemi znaky je zobrazen na obrázku 5.5. Pokud není vybraný žádný znak ke zvýraznění, zobrazí se pouze fotografie v původní velikosti.



Obr. 5.5: Výsledek procesu definice pozice v obraze

5.4 URČENÍ BARVY

Jakoukoli informaci lze v počítači vyjádřit jen pomocí jedniček a nul. Ne jinak je tomu i v případě barvy. Každá barva je v počítači vyjádřena binárním kódem, který udává zastoupení červené, zelené a modré složky. Každá z těchto tří základních barevných složek, ze kterých vznikají ostatní barvy, je reprezentována 8 bity. Tedy máme k dispozici 256 odstínů červené, 256 odstínů zelené a 256 odstínů modré. Chce-li uživatel v počítači definovat nějakou barvu, většinou nezadává podíl červené, zelené a modré složky v binárním kódu, ale pro jednoduchost určí každou složku číslem v celočíselném rozsahu 0 až 255. Nula reprezentuje nulové zastoupení určité barevné složky v rámci výsledné barvy, číslo 255 reprezentuje její maximální zastoupení. Vybrané barvy a hodnoty jednotlivých barevných složek reprezentujících tuto barvu jsou uvedeny v následující tabulce. Hodnoty složek jsou uvedeny v desítkové soustavě.[12]

Tab. 5.1: Přehled vybraných barev s hodnotami jednotlivých barevných složek [12]

Barva		Barevné složky		
		R	G	B
Černá		0	0	0
Tmavě modrá		0	0	128
Modrá		0	0	255
Zelená		0	128	0
Modrošedá		0	128	128
Světle zelená		0	255	0
Světle modrá		0	255	255
Tmavě červená		128	0	0
Fialová		128	0	128
Olivová		128	128	0
Šedá		128	128	128
Červená		255	0	0
Růžová		255	0	255
Oranžová		255	165	0
Žlutá		255	255	0
Bílá		255	255	255

5.4.1 Automatické určení barvy

K určení barvy se opět používají extrahovaná obrazová data pro jednotlivé znaky. Algoritmus prochází obrazová data stejně jako algoritmus zjišťující pozici, tedy pixel po pixelu za pomocí dvou cyklu. V tomto případě nejde jenom o to, jestli je pixel barevný ale o to jakou barevnou informaci nese. Pro zjištění této informace, použijeme metodu getRGB, která vrací číslo typu int. Chceme-li získat hodnoty jednotlivých barevných složek, musíme tuto hodnotu upravit. Abychom tuto metodu mohli použít, musíme obrazová data nahrát jako instanci třídy BufferedImage pomocí metody read třídy ImageIO. Když známe hodnoty jednotlivých barevných složek, porovnáme je s hodnotami vybraných barev. Výsledkem algoritmu je informace poskytující četnost výskytu jednotlivých barev v obraze. Jako výsledná barva je vybrána ta s největším počtem výskytu.[13]

Pokud by algoritmus porovnával získané barevné složky s hodnotami uvedenými v tabulce 5.1, nejspíš by nebyla nalezena žádná shoda, protože různými kombinacemi hodnot složek R, G, B získáme 16 777 216 různých barev a my jsme jich definovali pouze 16. Zvýšením rozsahu barevných složek pro jednotlivé barvy docílíme zvýšení pravděpodobnosti přiřazení daného pixelu k nějaké definované barvě. Jelikož obrazová data, stejně jako všechna data, obsahují šum, který ovlivňuje různé barvy jinak, mají světlé barvy, při porovnávání barvy, nastaven větší rozsah než tmavší barvy. Šedá barva má nastaven nejužší rozsah. Přehled použitých rozsahu barevných složek pro jednotlivé barvy je uveden v tabulce 5.2.

Tab. 5.2: Přehled rozsahů jednotlivých barevných složek

Barva	Barevné složky		
	R	G	B
Černá	0 - 43	0 - 43	0 - 43
Tmavě modrá	0 - 43	0 - 43	44 - 169
Modrá	0 - 43	0 - 43	170 - 255
Zelená	0 - 43	44 - 169	0 - 43
Modrošedá	0 - 43	44 - 169	44 - 169
Světle zelená	0 - 43	170 - 255	0 - 43
Světle modrá	0 - 43	170 - 255	170 - 255
Tmavě červená	44 - 169	0 - 43	0 - 43
Fialová	44 - 169	0 - 43	44 - 169
Olivová	44 - 169	44 - 169	0 - 43
Šedá	112 - 144	112 - 144	112 - 144
Červená	170 -255	0 - 43	0 - 43
Růžová	170 - 255	0 - 43	170 - 255
Oranžová	170 - 255	44 - 169	0 - 43
Žlutá	170 - 255	170 - 255	0 - 43
Bílá	170 - 255	170 - 255	170 - 255

Výpis kódu 5.9: Algoritmu provádějícího zjištění barvy vstupního obrazu.

```
//načtení obrazových dat daného znaku do img
Image img = new Image(getClass().getResourceAsStream(url));
//zjištění barvy pixelu v bodě [0, 0] (bez barvy)
Color pr = img.getPixelReader().getColor(0, 0);
//vytvoření vstupního proudu dat
InputStream inputStream = getClass().getResourceAsStream(url);
//vytvoření instance třídy BufferedImage a přiřazení hodnoty
BufferedImage im = null;
try {
//načtení vstupního proudu do BufferedImage
      im = ImageIO.read(inputStream);
      }
//zachycení vyjímky
catch (IOException e) {
      e.printStackTrace();
      }
//cykly pro procházení jednotlivých pixelů
for (int x = 0; x < img.getWidth(); x++) {
      for (int y = 0; y < img.getHeight(); y++) {
//zjištění barvy pixelu v daném bodě
            Color c =img.getPixelReader().getColor(x, y);
//zjištění barevné hodnoty
            pix=im.getRGB(x, y);
//získání hodnot jednotlivých barevných složek z předešlé hodnoty
            R = (pix & 0x00ff0000) >> 16;
            G = (pix & 0x0000ff00) >> 8;
            B = pix & 0x000000ff;
//kontrola zda daný pixel nese informaci o barvě
            if (c!=pr){
//porovnání jednotlivých složek pixelu s předem definovaným rozsahem
//jednotlivých barev
                  if(0<R && R<43 && 0<G && G<43 && 0<B && B<43){
//pokud daný pixel zapadá do rozsahu dané bary, zvýší se počet výskytu o
//jedna
                        black++;
                  }
                  if(0<R && R<43 && 0<G && G<43 && 44<B && B<169){
                        darkblue++;
                        }
                              .
                              .
                              .
```

ZÁVĚR

Po dohodě s vedoucím práce bylo zadání práce pozměněno a to tak, že z návrhu byla vypuštěna zvuková část. V první části práce jsou vysvětleny pojmy týkající se biometrie a identifikace člověka. Byly zde zmíněny metody identifikace na základě biometrických znaků, které však k identifikaci využívají znaků takzvaně primárních. V další části textu jsou řešeny sekundární znaky jak biometrické, tak ne-biometrické a jejich výhody a využití v praxi. Důležitým cílem bylo vybrat vhodné sekundární znaky, pomocí kterých můžeme identifikovat osobu podle fotografie. Seznam vybraných znaků se nachází v kapitole 3.2.1.

Dalším z cílů bylo navrhnout databázovou strukturu, byly zde stručně vysvětlený a popsány databázové modely, z kterých byl vybrán model objektově orientovaný. Pro grafické znázornění navržené struktury byl použit UML diagram tříd, který byl vytvořen v programovém prostředí ArgoUML.

Pro možnost komunikace uživatele s databází bylo vytvořeno grafické uživatelské rozhraní pomocí nástroje JavaFX Scene Builder 2.0, jehož výstupem je soubor v jazyce FXML, což je jazyk pro návrh formulářů. Dále jsou zde uvedeny algoritmy, které plní základní funkce databáze a to načítání, mazání a vyhledávání dat. Jedním z hlavních úkolů této práce bylo navrhnout sadu algoritmů pro automatickou anotaci jednotlivých znaků. Je zde řešeno určení pozice znaku v obraze vyhledáním okrajových hranic znaku, vykreslení ohraničení kolem tohoto znaku s využitím již známých hranic a určení jeho barvy porovnáváním a přiřazováním pixelů k předem vybraným barvám. Zdrojový kód pak byl napsán pomocí objektově orientovaného programovacího jazyka JAVAFX v IDE Eclipse SDK 4.4.2, který byl nastaven pro možnost vývoje JavaFX aplikací podle návodu dostupného ze zdroje [15].

Fotografie použité v této práci jsou z volně šiřitelné databáze osob INRIA Person [14]. Z těchto fotografií byly extrahovány oblasti odpovídající příslušným sekundárním znakům pomocí Adobe Photoshop CS5.

Celý zdrojový kód je uveden v příloze. Příloha, fotografie a z nich extrahované obrazové části jsou nahrány na přiloženém CD, které je součástí této diplomové práce. Pro správnou funkci databáze je potřeba připravené fotografie vložit do adresáře src/application/ vytvořeného projektu.

LITERATURA

[1] BEZDĚKOVÁ, Monika. *Identifikace osob podle vnějších znaků*. Praha, 2006. Diplomová práce. Univezita Karlova v Praze. Vedoucí práce RNDr. Petr Štourač.

[2] ŠČUREK, Radomír. *Biometrické metody identifikace osob v bezpečnostní praxi*. Ostrava, 2008. Studijní text. VŠB TU Ostrava.

[3] PŘIBYL, Tomáš. Behaviometrika - Biometrika na druhou. 2014. Dostupné z: http://www.ictsecurity.cz/odborne-clanky/behaviometrika-biometrika-na-druhou.html

[4] Biometriky. *Biometrické čtečky* [online]. [cit. 2014-12-15]. Dostupné z: http://www.biometricke-ctecky.cz/biometriky/

[5] OPPEL, Andy. *Databáze bez předchozích znalostí*. 2004. vyd. nám. 28. dubna 48, 635 00 Brno: Computer Press, a.s., 2006. ISBN 80-251-1199-7.

[6] BURGET, Radim. *Úvod do UML*. Ústav telekominikací. Studijní text. VUT Brno.

[7] ČIKA, Petr a Martin ZUKAL. *Multimediální služby: počítačová cvičení*. Ústav telekomunikací, 2013. ISBN 978-80-214-4721-9. Studijní text. VUT Brno.

[8] HEROUT, Pavel. *Java - bohatství knihoven*. 3. vyd. České Budějovice: Kopp, 2008, 251 s. ISBN 978-80-7232-368-5.

[9] KOEGH, James. *Java bez předchozích znalostí: průvodce pro samouky*. Vyd. 1. Brno: Computer Press, 2005, 274 s. ISBN 80-251-0839-2.

[10] ČÁPKA, David. *1. díl - Úvod do JavaFX* [online]. [cit. 2015-05-24]. Dostupné z: http://www.itnetwork.cz/java-tutorial-uvod-do-javafx

[11] Oracle. HOMMEL, Scott. *Working with Canvas* [online]. 2013 [cit. 2015-05-24]. Dostupné z: https://docs.oracle.com/javafx/2/canvas/jfxpub-canvas.htm

[12] Tabulka barev. *Wikipedia: the free encyclopedia* [online]. San Francisco (CA), 2014 [cit. 2015-05-24]. Dostupné z: http://cs.wikipedia.org/wiki/Wikipedie:Tabulka_barev

[13] Oracle. *Working with Images: Writing/Saving an Image* [online]. [cit. 2015-05-24]. Dostupné z: https://docs.oracle.com/javase/tutorial/2d/images/saveimage.html

[14] INRIA. *INRIA Person Dataset* [online]. 2005 [cit. 2015-05-24]. Dostupné z: http://pascal.inrialpes.fr/data/human/

[15] *E(fx)clipse: Download & Install* [online]. 2014 [cit. 2015-05-25]. Dostupné z: https://www.eclipse.org/efxclipse/install.html

SEZNAM POUŽITÝCH ZKRATEK

API	Application Programming Interface (rozhraní pro programování aplikací)
BMI	Body Mass Index (Index tělesné hmotnosti)
DBMS	Database Management Systems, v české literatuře označovaný jako SŘBD Systém řízení báze dat
DNA	DeoxyriboNucleic Acid (Deoxyribonukleová kyselina)
FXML	Verze XML (eXtensible Markup Language) pro návrh formulářů
GUI	Graphical User Interface (Grafické uživatelské rozhraní)
ID	unikátní identifikátor
IDE	Integrated Development Environment (Vývojové prostředí)
NSTC	Nation Science and Technology Council (Národní rada pro vědu a technologii
OO	Objektově orientovaný
QI	Quételetův Indexu
RGB	Barevný model Red-Blue-Green
RIA	Rich Internet Application (bohaté internetové aplikace)
SDK	Software Development Kit (Systémový vývojový nástroj)
UML	Unified Modeling Language (Grafický jazyk pro vizualizaci programových systému)

OBSAH PŘILOŽENÉHO CD

- Elektronická verze diplomové práce ve formátu pdf.
- Zdrojový kód v jazyce JavaFX
- Fotografie sekundárních znaků pro 350 osob.