Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Katedra softwarových technologií

Objektově orientované programování - IOOP

Zadání semestrální práce

Evidence automobilů

Autor zadání:
Karel Šimerda

Garant předmětu:

Verze 1.1
7.11.2017

1 Úvod

Cíl semestrální práce Seznámit studenty se základy objektově orientovaného programování na malém projektu.

Účel dokumentu Tento dokument obsahuje zadání a postup prací na semestrální práci „Evidence automobilů“.

Zadání semestrální práce navazuje na úlohy, které byly řešeny v rámci předmětu IPALP. Většina studentů je tedy seznámena s problémovou doménou evidence automobilu a nyní se můžou více soustředit na pracovní postupy objektového přístupu a na implementaci samotného zdrojového kódu.

Úprava původní verze evidence automobilu z předmětu IPALP bude spočívat v tom, že se nejdříve vytvoří třída se spojovým seznamem, která bude nezávislá na typu vkládaných datových prvků. Tato třída muže procházet postupně změnami, na kterých bude ukázán objektový přístup a ověření jednotkovým testem. Při realizaci grafického rozhraní a jeho propojení se seznamem automobilů budou představeny další možnosti syntaxe jazyka, jako je například genericita nebo rozšířená syntaxe v interfejsu ve smyslu Java 8.

Teprve po dokončení vývoje obecného rozhraní a třídy spojového seznamu, bude přikročeno k jejich vytvoření tříd různých typů automobilů.

Při vývoji budeme používat princip výstavby programu: Ze zdola nahoru. To znamená, že nejdříve naprogramujeme třídy nejnižší úrovně, které nepoužívají jiné třídy. Teprve později přikročíme k naprogramování tříd vyšší úrovně a až naposled vytvořené třídy začleníme do uživatelského rozhraní.

2 Vize

Program „**Evidence automobilů**“ bude umožňovat vést seznam různých typů automobilů, jako jsou osobní automobily, dodávky a nákladní automobily. Tento omezený počet typů automobilů, naprosto stačí pro naše výukové potřeby. Jde o princip, jak pracovat s několika různými typy datových entit. V reálné aplikaci by samozřejmě bylo typů daleko víc a jednotlivé typy by obsahovaly více informací.
Řízení programu bude zajišťovat několik ovládacích prvků v hlavním okně aplikace. Tyto prvky budou umožňovat spouštění funkcí zobrazování, setřídění, vyhledávání nebo filtrování uložených informací o automobilech.
Z hlavního okna se budou vyvolávat jednotlivá dialogová okna, v kterých se budou zadávat nebo měnit parametry automobilů.
Bude požadováno, aby stav programu bylo možné trvale uložit do souboru a zpětně obnovit.
V programu bude vyžadováno co možná nejširší použití vlastností Java 8. Rozsah využití syntaxe Java 8 bude mít vliv na závěrečné hodnocení studenta.

3 Požadavky

3.1 Funkční požadavky

FR_01 Typy automobilů: Požaduje se, aby aplikace vedla nejméně tři typy automobilů a to

- osobní automobil
- dodávka
- nákladní automobil

FR_02 Charakteristika typů automobilů: Požaduje se, aby každý typ byl popsán alespoň jedním společným údajem pro všechny typy a dvěma jedinečnými údaji. Alespoň u jednoho typu bude použit vlastní výčet barev.

FR_03 Rozdílné charakteristiky typů automobilů: Každý student si vytvoří vlastní jedinečnou charakteristiku u každého typu v jejich počtech, názvech a významech. Shoda v charakteristice typů u dvou nebo více projektů, bude považována za nesplnění zadání semestrální práce.

FR_04 Zobrazování seznamu: Program bude v hlavním okně zobrazovat seznam automobilů včetně charakteristik každého typu.

FR_05 Testovací seznam: Stisknutím tlačítka **Test** se nahraje do seznamu zkušební seznam automobilů a seznam se zobrazí.

FR_06 Nový automobil: Tlačítkem **Nový** se otevře dialogové okno se zadáním nového automobilu. Po zvolení typu automobilu se zobrazí příslušné parametry automobilu, aby bylo možné zadat hodnoty. Po vložení nové nebo změněné položky se obnoví zobrazení seznamu. Každý student si zvolí vhodné parametry pro každý typ automobilu podle svého uvážení. Požaduje se nejméně 3 parametry, přičemž alespoň jeden musí být společný pro všechny typy.

FR_07 Výběr položky: V poli se zobrazeným seznamem automobilů bude možné vybrat jednu položku k dalším operacím.

FR_08 Zrušení automobilu v seznamu: Tlačítkem **Zruš** se zruší vybraný automobil ze seznamu a seznam se obnoví.

FR_09 Editace parametru automobilu: Tlačítkem **Změň** se otevře dialogové okno vybraného automobilu podle jeho typu. V dialogovém okně se zobrazí příslušné parametry automobilu k změně hodnot. Po ukončení změn v dialogovém okně se seznam automobilů automaticky obnoví.

FR_10 Filtrace podle typu automobilů: Po výběru typu automobilu se v hlavním okně zobrazí pouze automobily požadovaného typu.

FR_11 Zrušení filtrace: Stisknutím tlačítka **Zruš filtraci** dojde ke zrušení filtrace.

FR_12 Uložení zálohy: Stisknutím tlačítka **Ulož** dojde k uložení všech automobilů v seznamu do záložního binárního souboru. Do souboru se budou ukládat pouze serializované objekty automobilů. Jméno souboru bude pevné a to **zaloha.bin**.

FR_13 Obnovení seznamu: Stisknutím tlačítka **Obnov** dojde k nahrazení aktuálního seznamu seznamem automobilů ze souboru **zaloha.bin**.

3.2 Nefunkční požadavky

3.2.1 Požadavky na implementaci třídy Seznam

NRS_01 Obecný seznam: Požaduje se implementovat třídu `Seznam` jako jednosměrný spojový seznam, který bude nezávislý na typu vkládaných dat. Pro tento seznam nesmí být použita žádná třída z knihovny Java. Tato třída musí být zcela plně nově naprogramována.

NRS_02 Omezení velikosti seznamu: Požaduje se, aby třída `Seznam` měla nastavitelnou maximální velikost (kapacitu) seznamu vkládaných datových objektů.

NRS_03 Rozhraní seznam: Požaduje se, aby třída `Seznam` implementovala rozhraní `ISeznam`, které je součástí tohoto zadání. Je zakázáno toto rozhraní jakkoliv měnit.

NRS_04 Omezení veřejných metod implemetační třídy: Třída `Seznam` nesmí mít další veřejné metody než ty, které jsou dány rozhraním `ISeznam`.

NRS_05 Metody přímého ovládaní seznamu: Implementace výchozích (default) metod přímého ovládání seznamu v rozhraní `ISeznam` není povinná. Je jen doporučená. Implementace podobných metod bude povinná až na datových strukturách. Proto je, ale implementace doporučena.

NRS_06 Serializace objektů: Požaduje se, aby ukládání obsahu seznamu pomocí serializace do souborů a jejich opětovné obnovení bylo řešeno mimo třídu Seznam.

NRS_07 Procházení seznamu: Požaduje se, aby prohlídka seznamu byla zajištěna pomocí vnitřní nebo anonymní třídy, která bude implementovat rozhraní `Iterator` z knihovny Java.

3.2.2 Požadavky na implementaci uživatelského rozhraní

NFRGUI_01 Rozhraní JavaFX: Požaduje se, aby grafické uživatelské rozhraní (GUI) bylo realizovano v JavaFX.

NFRGUI_02 Tvorba grafického rozhraní: Požaduje se, aby sestavení scén (scene) na jevišti (stage) byly vytvářeny pouze prostředky jazyka Java. Není tedy povoleno použití programu SceneBuilder pro tvorbu scén.

3.2.3 Požadavky na architekturu aplikace

NRAPP_01 Ošetření chyb: Požaduje se, aby chyby ve vykonávání metod tříd byly hlášeny pouze pomocí výjimek a obslouženy třídou `Alert`.

NRAPP_02 Použití datových proudů Stream: Požaduje se, aby aplikace využívala k výběru obsahu seznamu přednostně datových proudů (datovodů).

NRAPP_03 Použití lambda-výrazů: Požaduje se, aby pro filtraci položek při výběru ze seznamu byly použity lambda výrazy.

NRAPP_04 Používání výčtů: Požaduje se, aby hodnoty výčtů byly zobrazovány přirozeným textem. Přičemž tento text hodnot nesmí být součástí GUI.

NRAPP_05 Oddělení výkonného kódu od GUI: Požaduje se, důsledné oddělení grafického uživatelského rozhraní od výkonného kódu aplikace. Doporučuje se použít nějaký vhodný návrhový vzor, jako jsou například adaptér nebo fasáda, Taky lze použít Model-View-Controller (MVC).

3.2.4 Požadavky na vedení projektu

NRP_01 Ukládání projektu Požaduje se projekt ukládat do verzovacího systému Subversion (SVN) na adrese https://fei-svn.upceucebny.cz do složky `ioop` studenta.

NRP_02 Omezení souborů v úložišti SVN Požaduje se ukládat do úložiště SVN pouze zdrojové a projektové soubory. Je zakázáno ukládat soubory, které byly získány překladem nebo generátorem Javadoc.

NRP_03 Verzování projektu Je povoleno mít na úložišti ve slozce `ioop` jen jeden projekt aplikace, který bude postupně verzován. V případě potřeby si může student vytvořit vývojovou větev projektu, ale mimo složku `ioop`. Vývojová větev projektu se musí vytvořit v rámci SVN a ne tím, že se naimportuje jako nový projekt!

4 Postup prací

1. Založit projekt v Netbeans s názvem `EvidenceAut`, jehož součásti bude příjmení studenta.
2. Přednastavit tyto balíčky

   kolekce pro všechny třídy, a rozhraní, které souvisí s třídou `Seznam`

   automobily pro třídy, které popisují typy automobilů

   gui pro třídy grafického rozhraní
3. Do balíčku `kolekce` vložit rozhraní `ISeznam`, třídu `Seznam` a další potřebné soubory.
4. K třídě `Seznam` vytvořte testovací třídu v JUnit.
5. Teprve po vývoji a otestování třídy `Seznam` pokračovat ve vývoji ostatních tříd, rozhraní a výčtů v ostatních balíčcích.
6. Pro ostatní třídy, než je třída `Seznam`, se nepožaduje otestování. Tyto třídy budou ověřeny pouze přes uživatelské rozhraní aplikace.

Poznámka: Po vytvoření projektu a vložení rozhraní ISeznam, lze vygenerovat dokumentaci pomocí Javadoc v menu Run v NetBeans. Po vygenerování dokumentace lze kontrakt a signaturu rozhraní a i dalších tříd nebo výčtů projektu prohlížet internetovém prohlížeči. Vygenerované soubory html neukládat do úložiště SVN.
Upozornění: Případné nesrovnalosti, které se můžou vyskytnout v tomto zadání, budou upřesněny učitelem na cvičení. V případě závažných připomínek bude vydána další verze tohoto dokumentu.

5 Rozhraní ISeznam

```java
package evidenceautomobilu.kolekce;

import java.util.function.Function;
import java.util.function.Supplier;
import java.util.stream.Stream;
import java.util.stream.StreamSupport;

/**
 * Rozhrani predepisuje jednoduche rozhrani pro ruzne implementace seznamu
 * objektu.
 *
 * Rozhrani rozsiruje rozhrani Iterable, ktere ma jednu abstraktni metodu
 * {@code Iterator<T> iterator()} a dalsi dve metody, a to
 * {@code void forEach(Consumer<? super T> action)} a
 * {@code Spliterator<T> spliterator()}, ktere jsou oznaceny jako
 * {@code default}. Tyto defaultni metody zajistuji implicitni chovani vsem
 * implementacnim tridam pro praci s prvky seznamu.
 *
 * <p>
 * Protoze se jedna o univerzalni rozhrani, jsou nektere metody navrzeny jako
 * {@code default}, aby je nektere tridy nemusely implementovat.
 *
 * <p>
 * Metody tohoto rozhrani byly zvoleny tak, aby se procvicila latka z prednasek
 * a cviceni. Dalsim duvodem je to, ze implementace podobnych rozhrani bude
 * vyzadovana v pristim semestru v predmetu datove struktury.
 *
 * @author karel@simerda.cz
 */
public interface ISeznam<E> extends Iterable<E> {

    /* ========================================================================
       Zjistovaci metody
     */
    /**
     * Metoda vrátí maximální velikost seznamu (kapacitu). Pokud to bude mít
     * smysl.
     *
     * @return maximalni pocet mist v seznamu nebo -1, kdyz to nebude mozne
     */
    default public int getVelikost() {
        return -1;
    }

    /**
     * Metoda vrati aktualni pocet vlozenych objektu.
     *
     * @return pocet objektu v seznamu
     */
    int getPocet();
```

```
51
52      /**
53       * Metoda zjisti, zda seznam obsahuje prvky.
54       *
55       * @return vraci {@code true}, kdy je seznam neprazdny, jinal {@code false}
56       */
57      boolean jePrazdny();
58
59      /**
60       * Metoda zjisti, zda je seznam plny.
61       *
62       * <p>
63       * Plnost seznamu zavisi na tom, jake jsou pozadavky na implementacni tridy.
64       * Toto rozhrani to neresí. Omezeni muze byt dano velikosti maximalnim
65       * poctem v parametru konstruktoru, pricemz to nemusi zalezet na tom, zda
66       * implementace je na poli nebo zda je realizovana spojovym seznamem.
67       *
68       * @return vraci {@code true}, kdyz je seznam plny, jinak {@code false}
69       */
70      default boolean jePlny() {
71          return false;
72      }
73
74      /* ========================================================================
75         Metody pridavani a odebirani datovych prvku ze seznamu.
76
77       */
78      /**
79       * Vlozi objekt s daty do seznamu na prvni volne misto.
80       *
81       * <p>
82       * U spojoveho seznamu to bude vzdy na konec seznamu. U implementace pomoci
83       * pole to bude zalezet na tom, zda jsou prvky po odebrani prvku presunuty
84       * nebo ne.
85       *
86       * @param data vkladany objekt do seznamu
87       *
88       * @throws KolekceException pokud se vyskytla chyba pri vkladani
89       */
90      void pridej(E data) throws KolekceException;
91
92      /**
93       * Metoda vlozi objekty v parametrech do seznamu
94       *
95       * <p>
96       * Tato metoda slouzi k usnadneni vkladani datovych objektu do seznamu. Tim,
97       * ze touto {@code default} metodou budou "vybaveny" vsechny implemetnacni
98       * tridy tohoto rozhrani. Samozrejme, ze bude zaviset na tom, zda bude
99       * funkcni implemetace metody {@code pridej(E data)}
100      *
101      * @param data vkladane objekty
102      *
```

```
103     * @throws KolekceException pokud se vyskytla chyba pri vkladani
104     */
105    default void pridej(E... data) throws KolekceException {
106        for (E prvek : data) {
107            pridej(prvek);
108        }
109    }
110
111    /**
112     * Metoda vrati datovy objekt ze seznamu podle shody s objektem (klicem) v
113     * parametru metody.
114     *
115     * Upozorneni: Klic v parametru nemusí obsahovat vsechny atributy ve shode s
116     * hledanym objektem v seznamu. To, ktere atributy se budou porovnavat urci
117     * metoda equals datoveho objektu.
118     *
119     * @param klic identifikace objektu, podle ktereho se vyhledá datovy objekt
120     * v seznamu
121     *
122     * @return instance nalezeneho objektu nebo pokud je seznam prazdny, tak
123     * vrati null.
124     */
125    E najdi(E klic);
126
127    /**
128     * Metoda odebere objekt ze seznamu podle shody s objektem v parametru
129     * metody. Porovnani se provede podle obsahu dvou objektu prekrytou metodu
130     * equals.
131     *
132     * @param klic identifikace objektu, podle ktereho se má objekt odebrat ze
133     * seznamu
134     *
135     * @return instance odebiraneho objektu
136     */
137    E odeber(E klic);
138
139    /* ========================================================================
140       Metody primeho ovladani seznamu - Jsou nepovinne!!!
141
142        Upozorneni: Tyto metody, jsou pozadovany z duvodu, ze jsou vyzadovany
143                    velmi casto v zadanich z datovych struktur. Metody slouzi k
144                    procviceni primeho ovladani seznamu. Tim se mysli to, ze lze
145                    pomoci techto metod menit strukturu seznamu. Prakticky lze
146                    jakykoliv prvek ulozit na libovolne misto a nebo ho z tohoto
147                    mista odebrat.
148
149     */
150    /**
151     * Metoda nastavi aktualni vnitrni ukazatel na prvni prvek seznamu.
152     *
153     * @throws KolekceException vyjimka se vystavi, kdyz je seznam prazdny nebo
154     * kdyz metoda neni implemenovana ve tride
```

```
155     */
156    default void nastavPrvni() throws KolekceException {
157        throw new KolekceException("Metoda neni implementovana");
158    }
159
160    /**
161     * Metoda presune ukazatel na aktualni prvek na dalsi v seznamu.
162     *
163     * Pred volanim teto metody musi byt vzdy nastaven ukazatel na aktualni
164     * prvek seznamu.
165     *
166     * @throws KolekceException vyjimka se vystavi, kdyz je seznam prazdny, neni
167     * nastaven aktualni prvek nebo kdyz metoda neni implemenovana ve tride
168     */
169    default void prejdiNaDalsi() throws KolekceException {
170        throw new KolekceException("Metoda neni implementovana");
171    }
172
173    /**
174     * Metoda vraci referenci na datovy objekt, na jehoz prvek ukazuje vnitrni
175     * aktualni ukazatel.
176     *
177     * @return datovy prvek seznamu z aktualni pozice seznamu
178     *
179     * @throws KolekceException vyjimka se vystavi, kdyz je seznam prazdny, neni
180     * nastaven aktualni prvek nebo kdyz metoda neni implemenovana ve tride
181     */
182    default E zpristupni() throws KolekceException {
183        throw new KolekceException("Metoda neni implementovana");
184    }
185
186    /**
187     * Metoda vraci informaci, zda je k dispozici dalsi aktualni prvek seznamu.
188     *
189     * @return true, kdyz aktualni prvek je napojen na dalsi prvek v seznamu
190     *
191     * @throws KolekceException vyjimka se vystavi, kdyz je seznam prazdny, neni
192     * nastaven aktualni prvek nebo kdyz metoda neni implemenovana ve tride
193     */
194    default boolean jeDalsi() throws KolekceException {
195        throw new KolekceException("Metoda neni implementovana");
196    }
197
198    /**
199     * Metoda vlozi datovy objekt jako novy za aktualni prvek
200     *
201     * @param data vkladany datovy objekt do seznamu
202     *
203     * @throws KolekceException vystavi vyjimku, kdyz neni nastaven aktualni
204     * prvek nebo kdyz neni metoda implementovana
205     */
206    default void vlozZa(E data) throws KolekceException {
```

```
207         throw new KolekceException("Metoda neni implementovana");
208     }
209 
210     /**
211      * Metoda vlozi datovy objekt jako novy pred aktualni prvek
212      *
213      *
214      * @param data vkladany datovy objekt do seznamu
215      *
216      * @throws KolekceException vystavi vyjimku, kdyz neni nastaven aktualni
217      * prvek nebo kdyz neni metoda implementovana
218      */
219     default void vlozPred(E data) throws KolekceException {
220         throw new KolekceException("Metoda neni implementovana");
221     }
222 
223     /**
224      * Metoda odebere aktualni prvek ze seznamu.
225      *
226      * Po odebrani prvku je seznam stale spojity. Kdyz se odebere prvni, tak
227      * nasledujici se automaticky stane prvnim. Kdyz se odebere prvek uprostred
228      * seznamu, tak dojde ke spojeni predchoziho s nasledujicim prvkem. Ukaztel
229      * aktualniho prvku je po odebrani nedefinovan. Musi se znovu nastavit
230      * ukazatel na prvni polozku a prejit na nove pozadovane misto.
231      *
232      * @return pokud je prvek nalezen vraci se reference na odebirany datovy
233      * prvek, jinak se vraci null
234      *
235      * @throws KolekceException vyjimka se vystavi, kdyz je seznam prazdny, neni
236      * nastaven aktualni prvek nebo kdyz metoda neni implemenovana ve tride.
237      */
238     default E odeber() throws KolekceException {
239         throw new KolekceException("Metoda neni implementovana");
240     }
241 
242     /**
243      * Metoda zrusi obsah seznamu
244      */
245     void zrus();
246 
247 
248     /* ========================================================================
249        Metody prevodu obsahu seznamu na pole objektu s typem prvku Object nebo
250        s typem, kterey je dan typovym parametrem tridy Seznam.
251 
252      */
253     /**
254      * Metoda vrati pole s kopiemi datovych prvku seznamu o delce presne
255      * odpovidajici poctu vlozenych objektu. Typ prvku pole bude Object, protoze
256      * typ prvku pole nelze zmenit.
257      *
258      * @return pole objektu
```

```
259      */
260     E[] toArray();
261 
262     /**
263      * Metoda vrati pole s kopiemi datovych prvku seznamu o delce presne
264      * odpovidajici poctu vlozenych objektu. Pole bude mit prvky stejneho typu
265      * jako pole v parametru.
266      *
267      * @param array vzorove pole s ocekavanym typem prvku pole
268      *
269      * @return pole objektu
270      *
271      * @throws IllegalArgumentException vystavi vyjimku, kdyz pole je mensi nez
272      * pocet prvku v seznamu
273      */
274     E[] toArray(E[] array) throws IllegalArgumentException;
275 
276     /**
277      * Metoda vrati pole s kopiemi datovych prvku seznamu o delce presne
278      * odpovidajici poctu vlozenych objektu. Pole bude mit prvky typu E
279      *
280      * @param createFunction funkce na vytvoreni pole skytecnym typem prvku
281      *
282      * @return pole s kopiemi datovych prvku seznamu
283      */
284     E[] toArray(Function<Integer, E[]> createFunction);
285 
286     /* ========================================================================
287        Metody prevodu obsahu seznamu na datovy proud objektu
288 
289      */
290     /**
291      * Metoda prevede obsah seznamu na datovy proud, ktery preda pri navratu.
292      *
293      * @return datovy proud
294      */
295     default Stream<E> stream() {
296         return StreamSupport.stream(spliterator(), false);
297     }
298 
299 }
```