**Jihočeská univerzita v Českých Budějovicích**
**Přírodovědecká fakulta**


# Soubor projektů pro potřeby výuky objektově orientovaného programování

Bakalářská práce

**Břetislav Roháček**

Školitel: RNDr. Jaroslav Icha

České Budějovice 2012

**Bibliografické údaje**

Roháček B., 2012: Soubor projektů pro potřeby výuky objektově orientovaného programování. [A set of projects for teaching object-oriented programming. Bc. Thesis, in Czech.] – 41 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

**Anotace**

Cílem této bakalářské práce je vytvořit univerzální šablonu pro řešení problémů malého rozsahu v rámci objektově orientovaného programování. K realizaci jednotlivých fází budou použity volně dostupné nástroje. Součástí bakalářské práce bude soubor projektů, které budou z této univerzální šablony vycházet. Výsledky práce bude možné využít při výuce objektově orientovaného programování.

**Abstract**

The goal of this thesis is to create an universal template for solving small problems in small-scale within object-oriented programming. Freely available tools will be used for the implementation of each phase. Part of the thesis will be a set of projects that will be based on this general template. The results of the thesis can be used for teaching object-oriented programming.

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích, 25. 4. 2012

.....

Podpis

**Poděkování**
Tímto bych rád poděkoval svému školiteli RNDr. Jaroslavu Ichovi za jeho podporu, trpělivost, spolupráci, cenné rady a čas, který mi věnoval při vypracování této bakalářské práce.

# Obsah

# Úvod

Úkolem práce je navrhnout jednotnou šablonu, podle které bude možné vytvářet programy menšího rozsahu. Šablona bude rozdělena do jednotlivých fází. Dále je zapotřebí zvolit sadu projektů k realizaci, které budou navrženy podle univerzální šablony. Všechny fáze projektu budou detailně dokumentovány a mohou obsahovat alternativní řešení.

Výsledky této práce mohou posloužit jako výukový materiál při výuce objektově orientovaného programování či jako studijní materiál pro studenty. Hlavním důvodem ke zpracování tohoto tématu je fakt, že aplikace tohoto přístupu v praxi zcela chybí, a že studenti takovéto projekty a obecné postupy a návrhy postrádají.

Při výběru projektů a návrhu univerzální šablony jsem se nechal inspirovat několika hotovými řešeními. Jde především o projekty Automated Teller Machine (ATM) a Address Book. Zaměřil jsem se především na obsah jednotlivých fází – které techniky byly využity. [1, 2]

K úplnému pochopení všech souvislostí tohoto díla jsou vyžadovány základní znalosti programovacího jazyka Java a pojmů objektově orientovaného programování. Dále je vyžadována znalost techniky CRC karet, diagramů UML a návrhového vzoru Model-View-Controller (MVC).

Pojem rozhraní v této práci, představuje konstrukci v jazyce Java pro označení interface. Bude-li hovořeno o grafickém rozhraní, bude tak uvedeno v plném znění jako „grafické rozhraní“, případně „grafické uživatelské rozhraní“.

# Cíle práce

Cílem práce je vytvoření souboru projektů, které bude možné využít při výuce objektově orientovaného programování. Řešení všech projektů bude náležitě dokumentováno a zpracování bude vycházet z jednotné šablony využitelné pro řešení problémů malého rozsahu s důrazem na objektovou analýzu a návrh. Při zpracování tématu je nutné dosáhnout těchto dílčích cílů:

1. Navrhnout jednotnou šablonu použitelnou pro řešení problémů malého rozsahu v rámci kurzu objektově orientovaného programování, která bude zahrnovat tyto fáze vývoje:
   a. Stanovení cílů
   b. Analýza
   c. Objektový návrh
   d. Podrobný návrh a implementace
   e. Testování
   f. Výsledek
   g. Udržování
2. Pro jednotlivé fáze vývoje budou použity techniky a nástroje dostupné pro vzdělávací potřeby zdarma. Bude se jednat zejména o produkty NetBeans a Visual Paradigm for UML. Pro návrh tříd bude využita technika CRC cards.
3. Pro implementaci bude aplikován programovací jazyk Java. Jako vývojové prostředí bude použito NetBeans (www.netbeans.org) a BlueJ (www.bluej.org).

# Metodika

1. Prostudovat doporučenou literaturu a materiály obdržené od školitele.
2. Stáhnout, nainstalovat a otestovat potřebné nástroje k vývoji.
   - JDK, Netbeans
   - Visual Paradigm for UML [3]
   - Altova UModel [4]
3. Připravit jednotnou šablonu, dle které budou projekty tvořeny.
4. Navrhnout sadu projektů, které budou odpovídat úrovni v rámci předmětů Java I a II.
5. Realizace všech projektů včetně kompletní dokumentace a testování.

# Současný stav problematiky

V současné době je možné se setkat s mnoha modely vývoje programů, například vodopádový (sekvenční) nebo iterativní model [5]. K samotné tvorbě návrhu a celkovému vývoji aplikace slouží nástroje a techniky k tomu určené.

K běžně užívaným technikám patří metoda UML. Při řešení problémů menšího rozsahu, se využívá techniky CRC karet (viz str. 6). The Unified Modeling Language, zkráceně UML, je všeobecně uznávaným jazykem pro vizualizaci návrhů programových systémů ve formě diagramů [5]. Pro účely úvodních kurzů programování budou zapotřebí tyto diagramy: Diagram tříd, diagram balíčků, diagram případů užití.

Dále se můžeme setkat s návrhovým vzorem Model-View-Controller, ve zkratce MVC, který je rozdělen na 3 nezávislé komponenty, aby při jejich úpravě došlo jen k minimálním vlivům na ostatní části. Patří mezi ně datový model (Model) grafické uživatelské rozhraní (View) a řídící jednotka (Controller):

- Model pracuje s informacemi a procesy v aplikaci.
- View, neboli pohled, převádí uživateli data do grafické podoby.
- Controller reaguje na události a zajišťuje změny v datovém modelu nebo pohledu.

K vývoji aplikace je možné využít některé z volně dostupných nástrojů. Pro tvorbu UML diagramů a CRC karet je k dispozici program Visual Paradigm for UML, který je velice bohatým nástrojem a výrazně jej doporučuji, nebo například Altova UModel. Mezi vývojová prostředí patří například NetBeans, Eclipse a BlueJ, který je vhodným nástrojem během výuky.

# Univerzální šablona

**Úvod**

Jde o vytvoření seznamu kroků a vývoje aplikace od jejího počátku do finální podoby. Tato univerzální šablona zahrnuje techniku CRC karet, UML diagramy a návrhový vzor MVC. Šablona se skládá z těchto dílčích částí:

- Stanovení cílů
- Analýza
  - Diagram případů užití (Use Cases), analýza tříd
- Objektový návrh
  - CRC karty, diagram tříd, diagram balíčků
- Podrobný návrh a implementace
  - Detailní popis tříd, zdrojové kódy
- Testování
- Výsledek
- Udržování

**Rozdělení balíčků**

Rozdělení balíčků vychází z myšlenky návrhového vzoru MVC, tedy oddělení komponent model, view a controller.

Mějme 4 balíčky: model, view, controller a mvc. Balíček mvc bude obsahovat všechna potřebná rozhraní pro aplikaci MVC vzoru. Balíčky model, view a controller budou obsahovat jednotlivé třídy určené každé komponentě. Otázkou zní, které třídy a rozhraní budeme pro danou komponentu potřebovat, případně bychom mohli potřebovat?

Komponenta Controller bude mít na starosti pouze předání události/akce, případně hodnot, tedy nebudou zapotřebí vlastní třídy. Aktuální rozdělení balíčků by bylo:

- model, view a mvc.

Komponenta Model bude mít na starosti práci s daty a bude přijímat události z Controlleru, tedy bude toto rozhraní implementovat a nebude potřebovat vlastní rozhraní. Proto dojde ke změně balíčku „mvc“ na „vc“, respektive viewcontroller.

Zde jsou znázorněny jednotlivé kroky:

- model → model → **model**
- view → view → **view**
- controller → ~~controller~~
- mvc → mvc → **viewcontroller**

# 1. Stanovení cílů

V první řadě je třeba představit projekt – čeho se daná problematika týká. Následuje formulace požadavků – co od aplikace očekáváme, jaké by měla mít funkce a parametry, jak by se měla v konkrétních situacích chovat, případně jak by aplikace měla vypadat.

# 2. Analýza

Analýza se skládá z diagramu případů užití a z analýzy jednotlivých tříd.

## 2.1. Diagram případů užití (Use Cases)

Diagram obsahuje seznam účastníků a procesů. Procesy můžeme rozdělit na ty, ke kterým mají účastnící přímý přístup a na ty, které mohou být vyvolány jiným procesem. Každý účastník může mít k dispozici jiné procesy anebo mohou mít některé procesy společné. Jednotlivé procesy se mezi sebou mohou navzájem ovlivňovat.

Tato fáze obsahuje nejen samotný diagram, ale také seznam popsaných procesů.

## 2.2. Analýza tříd

Tato analýza obsahuje seznam tříd a rozhraní. Dále může obsahovat grafický návrh aplikace. Všechny třídy a rozhraní budou stručně popsány, k čemu slouží a co daná třída reprezentuje.

Spouštěcí metoda main bude vždy zahrnuta ve třídě Main. Zde je uveden seznam tříd a společných rozhraní, u kterých se nepředpokládá změna zdrojového kódu:

**Seznam společných tříd**

- Main – zajistí zobrazení grafického uživatelského rozhraní.
- Format – třída se statickými metodami pro hromadnou úpravu grafických komponent.
- JFrameUtils – slouží k nastavení objektů třídy JFrame.
- OAplikaci – obsahuje informace o zadané aplikaci – jméno autora, název aplikace a její verze.

**Společná rozhraní**

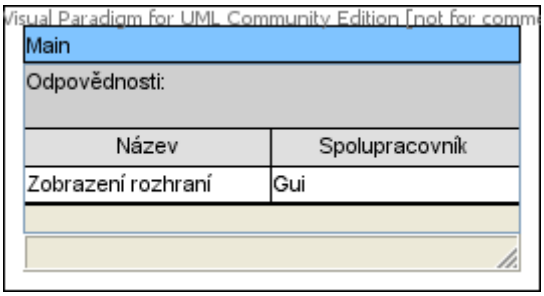
- View – zajistí aktualizaci pohledu při změně modelu pomocí rozhraní Controller. Obsahuje seznam typů jednotlivých aktualizací.
- Controller – při interakci s uživatelem, vznikají události, které jsou předávány modelu. Řídí změny nejen v modelu, ale také v pohledu při aktualizaci dat. Obsahuje seznam typů akcí, které mají být provedeny.
- ControllerExtended – stejně jako rozhraní Controller předává události při interakci s uživatelem, navíc zajistí předání hodnot a vykreslení grafických komponent.

# 3. Objektový návrh

## 3.1. CRC karty

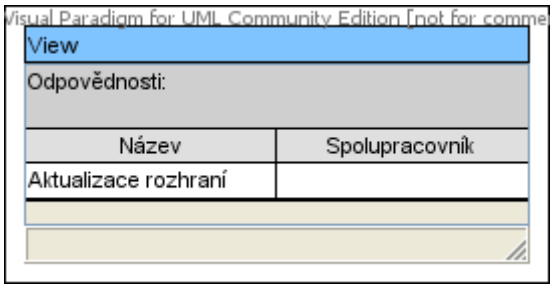
Technika CRC (Class-Responsibility-Collaboration, tedy třída-odpovědnost-spolupráce) karet obsahuje sadu karet, přičemž každé třídě je věnována právě jedna. Každá tato karta obsahuje seznam odpovědností, které třída má a se kterými třídami na těchto odpovědnostech spolupracuje. Karta může dále obsahovat seznam nadtříd a podtříd. [6]

### Karta základní třídy

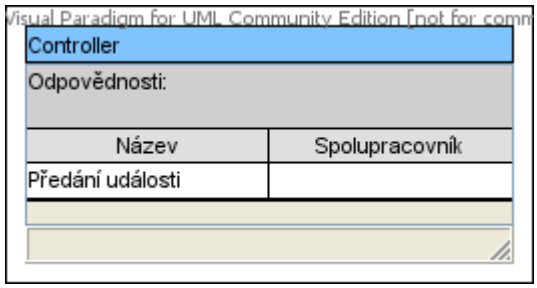



**Obr. 1: Šablona – CRC karta – Main**

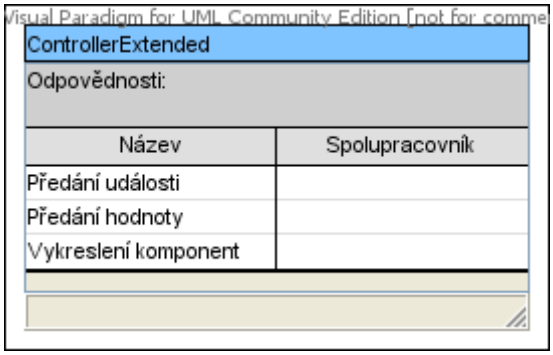
### Karty MVC tříd




**Obr. 2: Šablona – CRC karta – View**

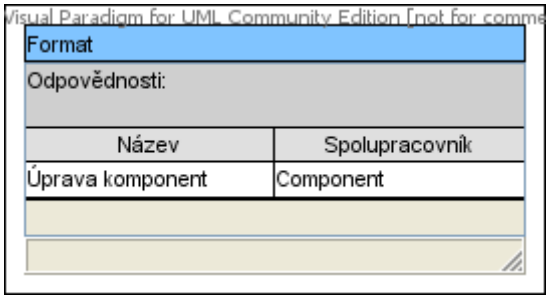



**Obr. 3: Šablona – CRC karta – Controller**

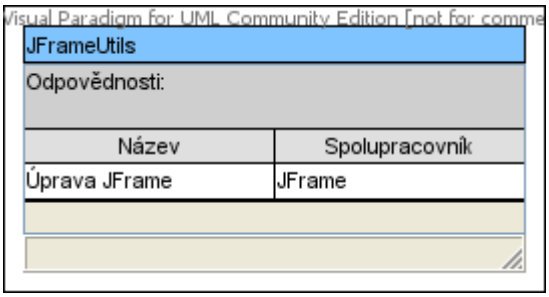



**Obr. 4: Šablona – CRC karta – ControllerExtended**

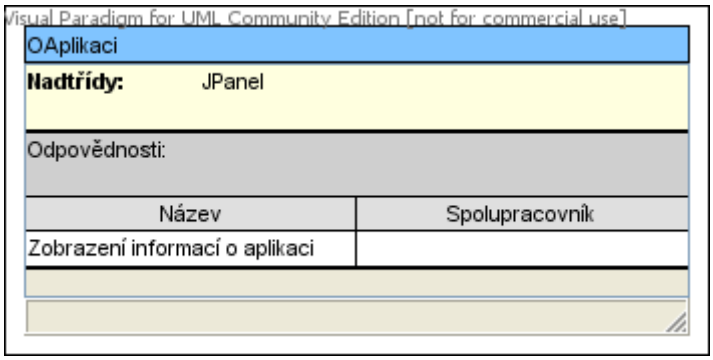
## Karty pomocných tříd




**Obr. 5: Šablona – CRC karta – Format**




**Obr. 6: Šablona – CRC karta – JFrameUtils**

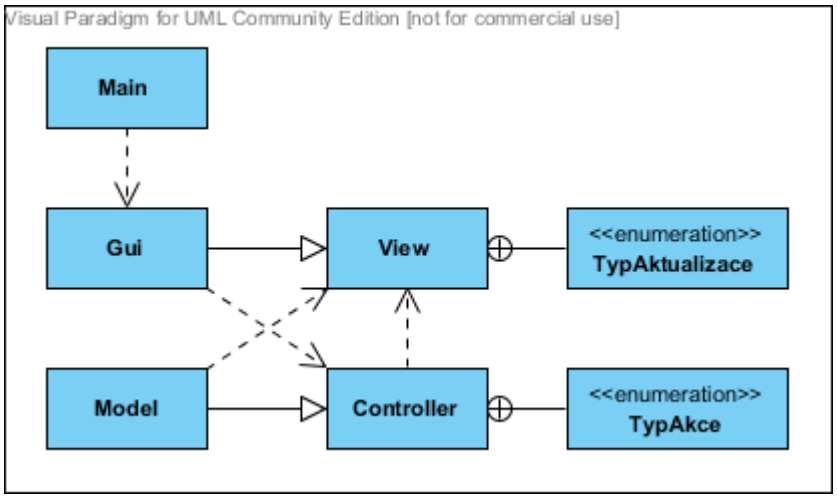



**Obr. 7: Šablona – CRC karta – OAplikaci**

## 3.2. Diagram tříd

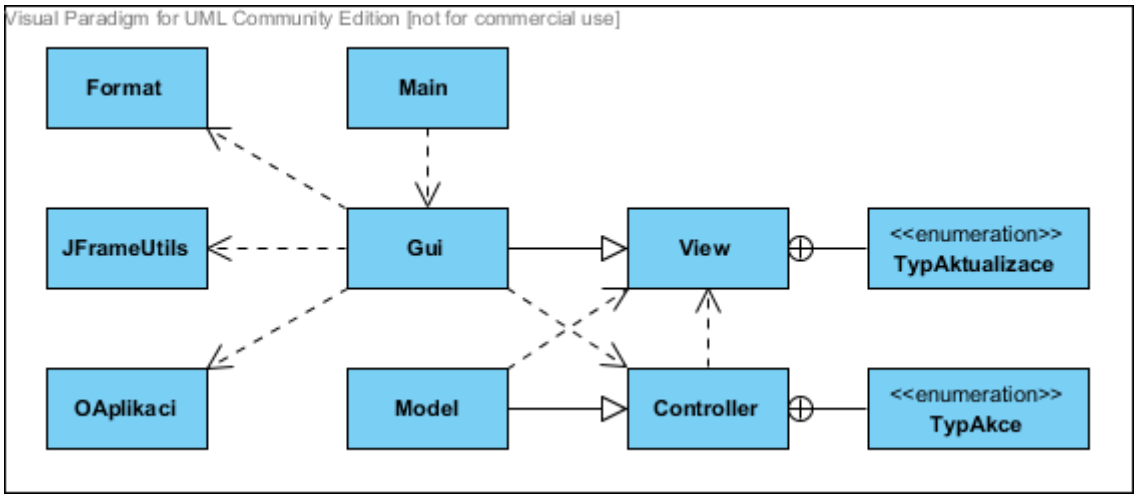
Diagram obsahuje seznam tříd a jejich vazby mezi sebou. Znázorňuje přehledný a uspořádaný systém mezi třídami. Diagram obsahuje základní značení, které Visual Paradigm for UML podporuje.

### Diagram společných tříd




**Obr. 8: Šablona – Diagram tříd 1**

### Diagram společných tříd včetně pomocných

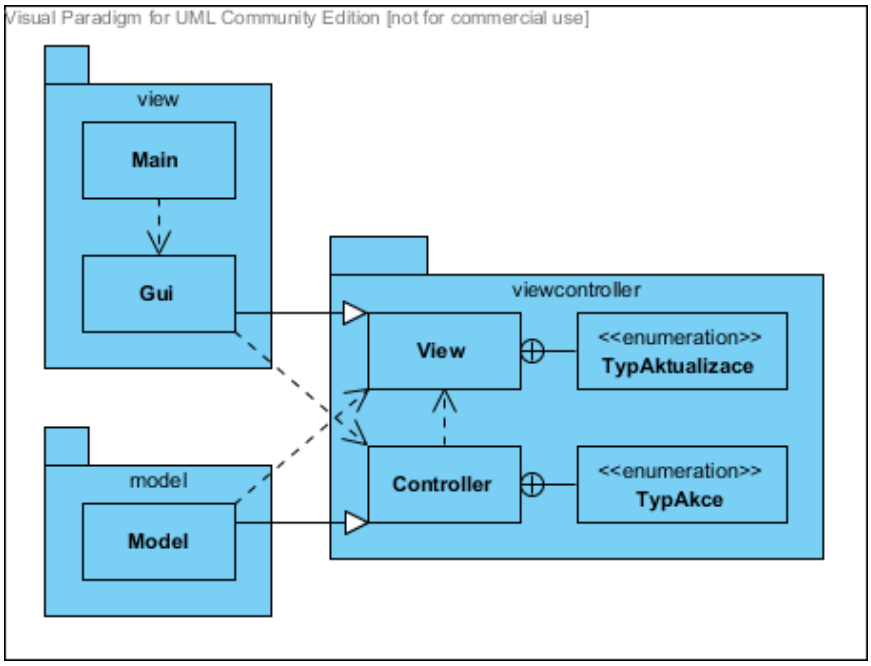



**Obr. 9: Šablona – Diagram tříd 2**

## 3.3. Diagram balíčků

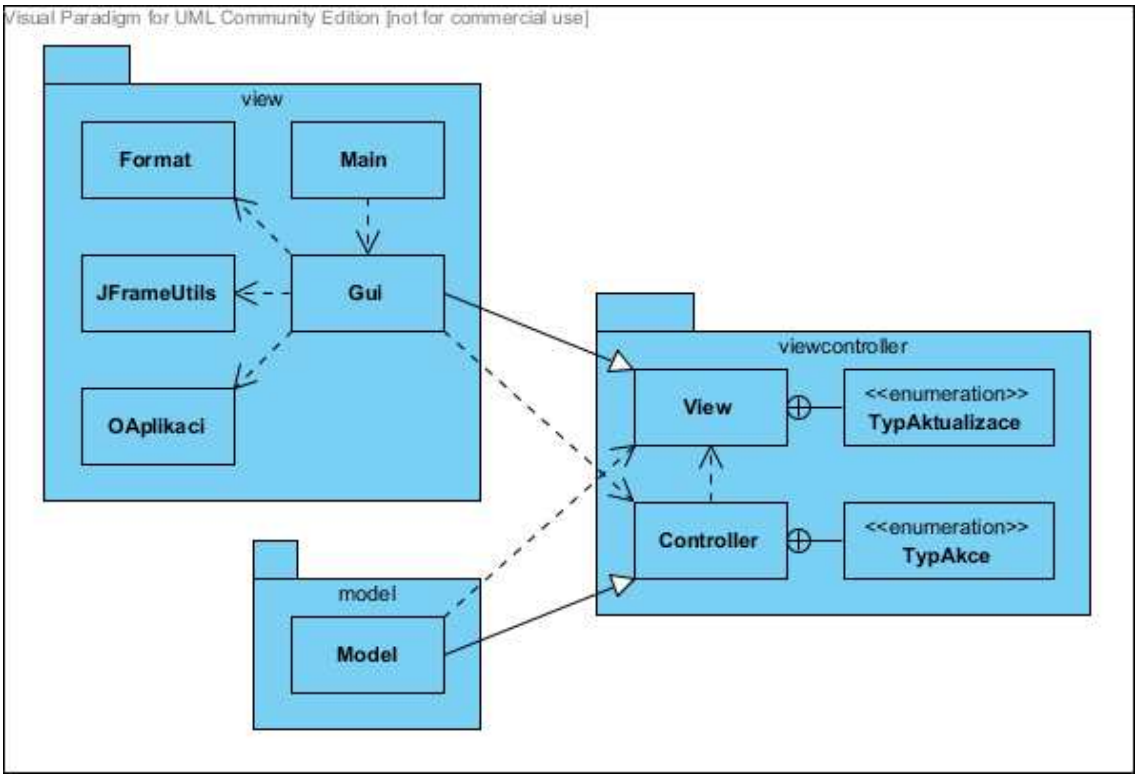
Diagram balíčků graficky vyjadřuje zapouzdření a závislost mezi samotnými balíčky. Tímto způsobem lze třídy rozdělit do určitých kategorií.

### Diagram společných balíčků




**Obr. 10: Šablona – Diagram balíčků 1**

### Diagram společných balíčků včetně pomocných tříd




**Obr. 11: Šablona – Diagram balíčků 2**

# 4. Podrobný návrh a implementace

## 4.1. Detailní popis tříd

Tento detailní popis je rozdělen podle jednotlivých balíčků a tříd. Třídy obsahují základní proměnné, konstruktory, metody a výčtové typy. U všech těchto prvků je uvedena přístupnost: + pro public, – pro private.

**Společné balíčky view a viewcontroller:**

**Balíček: viewcontroller**

Rozhraní: View

- Vnitřní výčtové typy
  + enum TypAktualizace
- Metody
  + void update(TypAktualizace typ, int index)
  + void update(TypAktualizace typ, int index, Object object)

Rozhraní: Controller

- Vnitřní výčtové typy
  + enum TypAkce
- Metody
  + void setView(View view)
  + void action(TypAkce typ, int index)

Rozhraní: ControllerExtended

- Vnitřní výčtové typy
  + enum TypAkce
- Metody
  + void setView(View view)
  + void action(TypAkce typ, int index)
  + void action(TypAkce typ, int index, Object object)
  + Object getObject(int index)
  + void paint(Graphics g)
  + boolean dispatchKeyEvent(KeyEvent e)

**Balíček: view**

Třída: Format

- Metody
  + static void setMenuFont(JFrame jFrame, Font font)
  + static void setFont(Component[] components, Font font)
  + static void setBackground(Component[] components, Color background)
  + static void setForeground(Component[] components, Color foreground)
  + static void setEditable(Component[] components, boolean editable)
  + static void setEnable(Component[] components, boolean enable)
  + static void setSelected(Component[] components, boolean selected)
  + static void setMargin(Component[] components, Insets insets)

Třída: JFrameUtils

- Proměnné
  + static final int TOP_LEFT
  + static final int TOP_RIGHT
  + static final int BOTTOM_LEFT
  + static final int BOTTOM_RIGHT
- Metody
  + static void maximalizovat(JFrame jFrame)
  + static void vycentrovat(JFrame jFrame)
  + static void setLocation(JFrame jFrame, int location)

Třída: OAplikaci

- Konstruktory
  + OAplikaci(String nazevAplikace, String verze, String autor)

## 4.2. Zdrojové kódy

Zdrojové kódy jsou uvedeny v příloze.

# 5. Testování

Testování aplikací bývá ze strany studentů často zanedbáváno, někdy dokonce zcela vynecháno, ačkoli je třeba této fázi vývoje věnovat zvláštní pozornost. Již během tvorby jednotlivých tříd se zavádí testovací třídy a metody. V našem případě implementace Javy budou využity jednotkové testy JUnit. Pokud tyto testy nebude možné provést algoritmicky, budou provedeny manuálně.

Testování zahrnuje seznam testovaných částí včetně popisu, jak byl test proveden. Zdrojové kódy testovacích tříd jsou uvedeny v příloze.

# 6. Výsledek

Aplikace je hotova a zbývá předložit výsledek. V této fázi je obsažen screen grafické podoby aplikace za jejího běhu.

# 7. Udržování

Program lze nyní dále ladit, upravovat a optimalizovat, případně doplnit a rozšířit o další možnosti. Během těchto úprav se mohou objevit důvody, pro které bude nutné provést případný refaktoring nebo celkově předělat návrh a vytvořit aplikaci znovu. Vracíme se tak zpět na počátek, kde si můžeme stanovit nové cíle, nové požadavky a opět projít celým procesem od začátku.

# Projekty

# 1. Automat na jízdenky

## 1.1. Stanovení cílů

### Automat

Automat bude představovat model reálného stroje, využívaný k výdeji jízdenek pro MHD v Českých Budějovicích. Automat umožní vybrat si jednu či více druhů jízdenek. Poté bude možné vhodit do automatu mince v hodnotách 50 Kč, 20 Kč, 10 Kč, 5 Kč, 2 Kč nebo 1 Kč. Během vhazování mincí bude umožněno přiobjednat jízdenky kteréhokoliv druhu. V průběhu celé objednávky bude možné objednávku stornovat, případně budou vráceny vhozené mince. Ve chvíli, kdy bude do automatu vloženo dostatečné množství peněz k vydání jízdenek, automat začne sám automaticky tisknout jízdenky a případně vrátí odpovídající přeplatek. Tisk jízdenek bude simulován v grafickém uživatelském rozhraní. Vracený obnos bude vrácen v mincích od nejvyšší platné hodnoty. V případě, že automat nebude mít na vrácení, bude objednávka stornována. Po dokončení objednávky, případně po stornování objednávky, bude automat připraven pro nové použití. Automat bude obsahovat displej, který bude zobrazovat průběžné změny během objednávky.

### Jízdenky

Každá jízdenka bude nést informaci o svém druhu, ceně a času platnosti. Budeme rozlišovat 2 druhy jízdného: základní, zlevněné. Časy platnosti: 20 minut, 40 minut, 60 minut a 24 hodin. Ceny se odvíjí podle daného druhu jízdného a času platnosti:

**Tabulka 1: Tarif jízdného MHD**

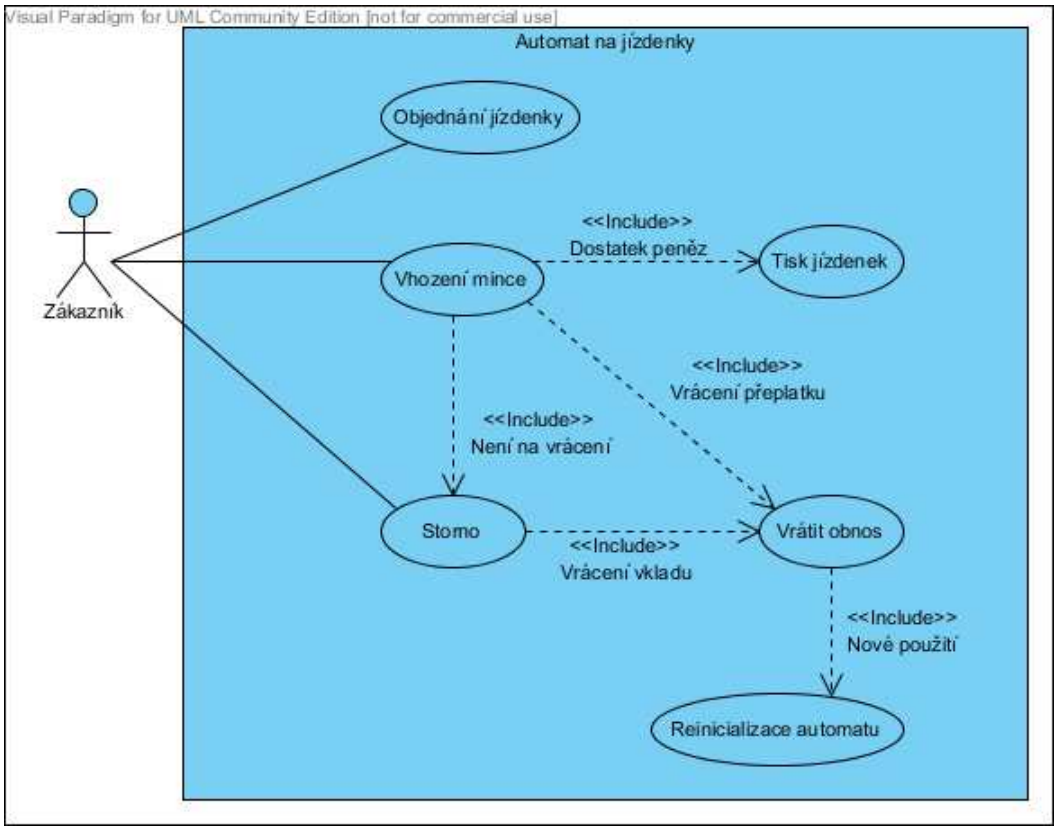
| | 20 minut | 40 minut | 60 minut | 24 hodin |
|---|---|---|---|---|
| Základní | 12 Kč | 14 Kč | 16 Kč | 40 Kč |
| Zlevněné | 6 Kč | 7 Kč | 8 Kč | 20 Kč |

### Displej

- Před použitím automatu se zobrazí uvítací text.
- Při každém objednání jízdenky se zobrazí částka, kterou bude ještě zapotřebí vložit do automatu, pro vydání všech objednaných jízdenek.
- Při každém vložení mince bude zobrazena částka jako u předešlého případu.
- Zobrazení zprávy při těchto činnostech:
  - Stornování objednávky.
  - Automat nemá na vrácení.
- Po řádném dokončení objednávky se zobrazí informace o tisku jízdenek s poděkováním za použití automatu.

## 1.2. Analýza

### 1.2.1. Diagram případů užití (Use Cases)




**Obr. 12: Automat – Diagram případů užití**

**Objednání jízdenky**

Zařadí zvolenou jízdenku do fronty v automatu a umožní vklad mincí.

**Vhození mince**

Operaci je možné provést pouze v případě, že je vybrána alespoň jedna jízdenka. Provede se vložení mince do zásobníku v automatu. Pokud je v automatu dostatek peněz a automat má na vrácení příslušného přeplatku, provede se tisk jízdenek a vrácení přeplatku. V opačném případě dojde ke stornování objednávky.

**Tisk jízdenek**

Provede simulaci tisku do grafického uživatelského rozhraní.

**Storno**

Stornuje aktuální objednávku a vrátí celkový vklad.

**Vrátit obnos**

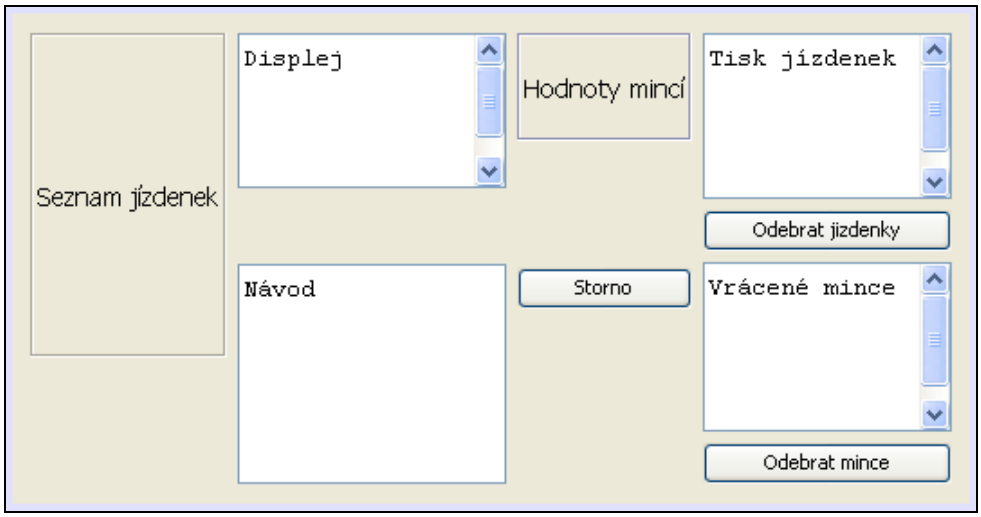
Vrátí požadovaný obnos v mincích od nejvyšší hodnoty. Buď půjde o vrácení přeplatku, nebo celého vkladu. Po vrácení peněz se automat připraví k novému použití (reinicializace automatu).

**Reinicializace automatu**

Automat se připraví k novému použití.

### 1.2.2. Analýza tříd

**Grafický návrh aplikace**




**Obr. 13: Automat – Návrh**

**Seznam tříd**

- Main – zajistí zobrazení grafického uživatelského rozhraní.
- Gui – grafické uživatelské rozhraní, obsahuje menu a panel s komponentami.
- MainPanel – hlavní panel s komponentami, který je součástí Gui a zajistí interakci uživatele s automatem.
- Automat – představuje model reálného automatu, který uživateli zajistí jeho obsluhu. Umožní objednání jízdenek, vhození mincí, případně stornování objednávky.
- Zasobnik – obsahuje platné hodnoty mincí, uchovává seřazené typy mincí a zajišťuje správné vrácení peněz.
- Mince – reprezentuje mince jedné hodnoty o určitém počtu, obsahuje informaci o své hodnotě, měně a počtu. Bylo by možné zvolit alternativní (typičtější) přístup, kdy by třída Mince reprezentovala jedinou minci dané hodnoty. V takovém případě by ovšem bylo zapotřebí uchovat v zásobníku informaci o počtech mincí jednotlivých hodnot.
- Jizdenka – reprezentuje jednu jízdenku daného typu, obsahuje informaci o svém druhu, ceně a času platnosti.

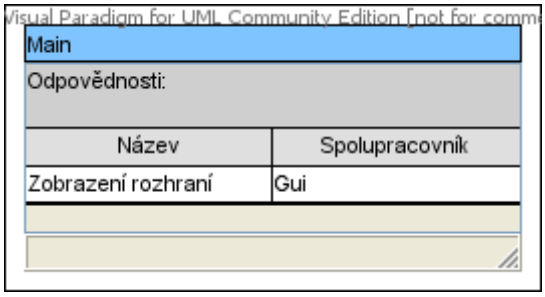
**Rozhraní**

- View, Controller – viz str. 5

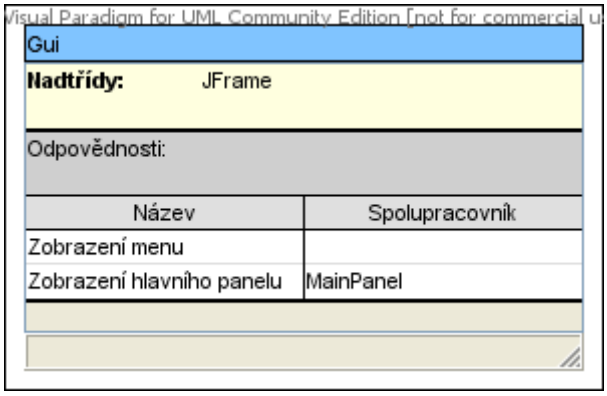
## 1.3. Objektový návrh

### 1.3.1. CRC karty

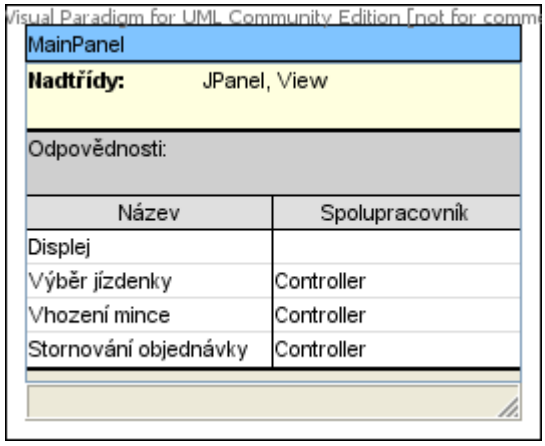
Karty: Format, JFrameUtils, OAplikaci – viz str. 7



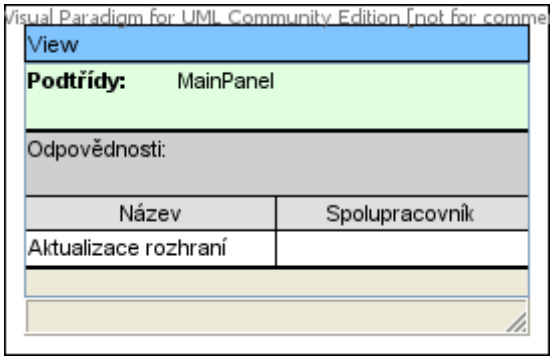

**Obr. 14: Automat – CRC karta – Main**



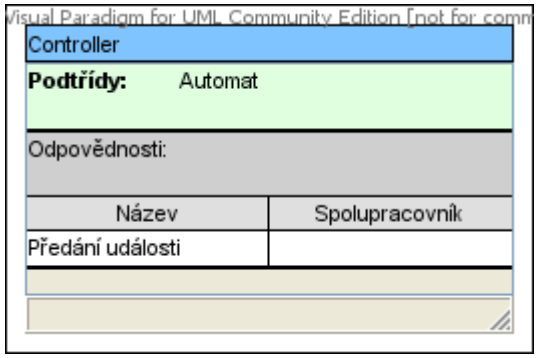

**Obr. 15: Automat – CRC karta – Gui**



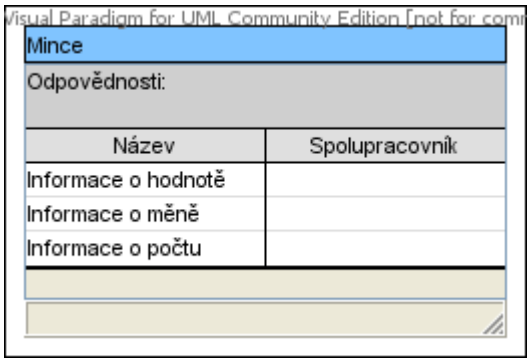

**Obr. 16: Automat – CRC karta – MainPanel**



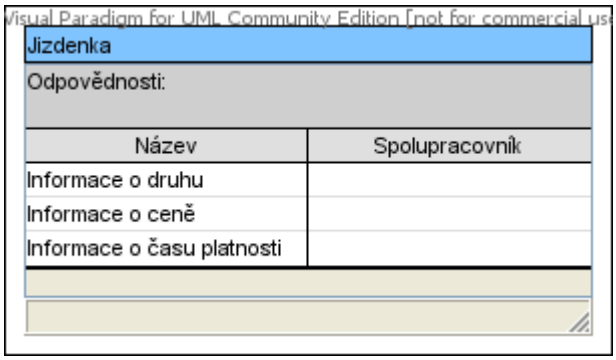

**Obr. 17: Automat – CRC karta – View**



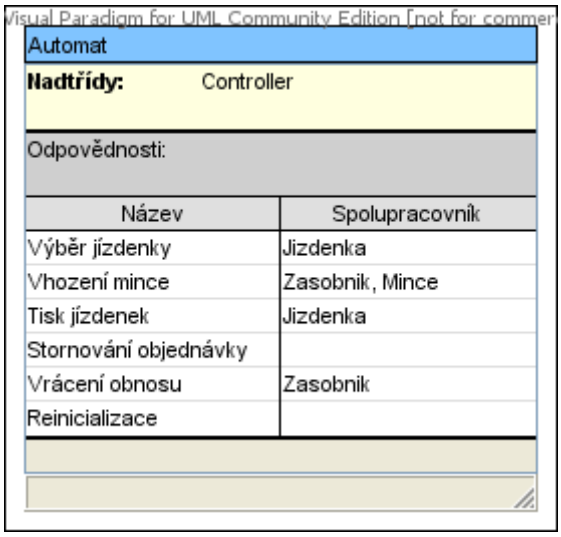

**Obr. 18: Automat – CRC karta – Controller**



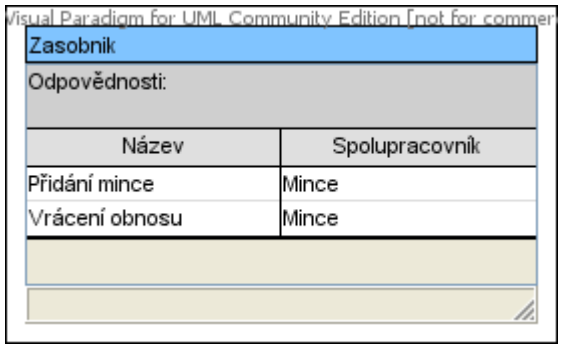

**Obr. 19: Automat – CRC karta – Mince**




**Obr. 20: Automat – CRC karta – Jizdenka**

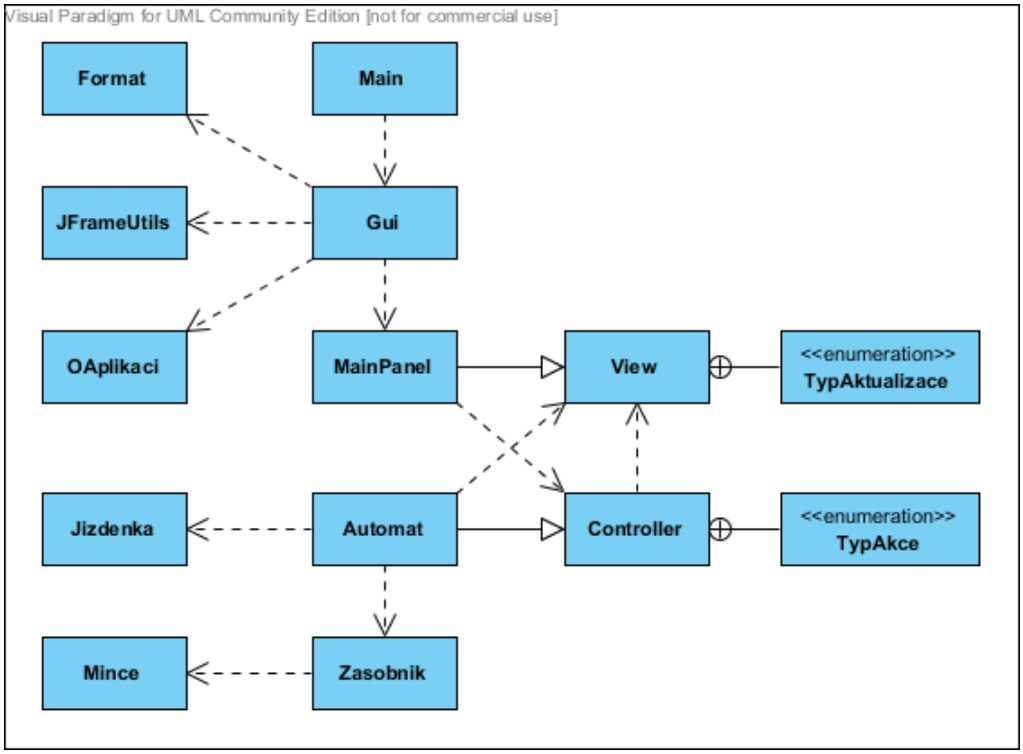



**Obr. 21: Automat – CRC karta – Automat**



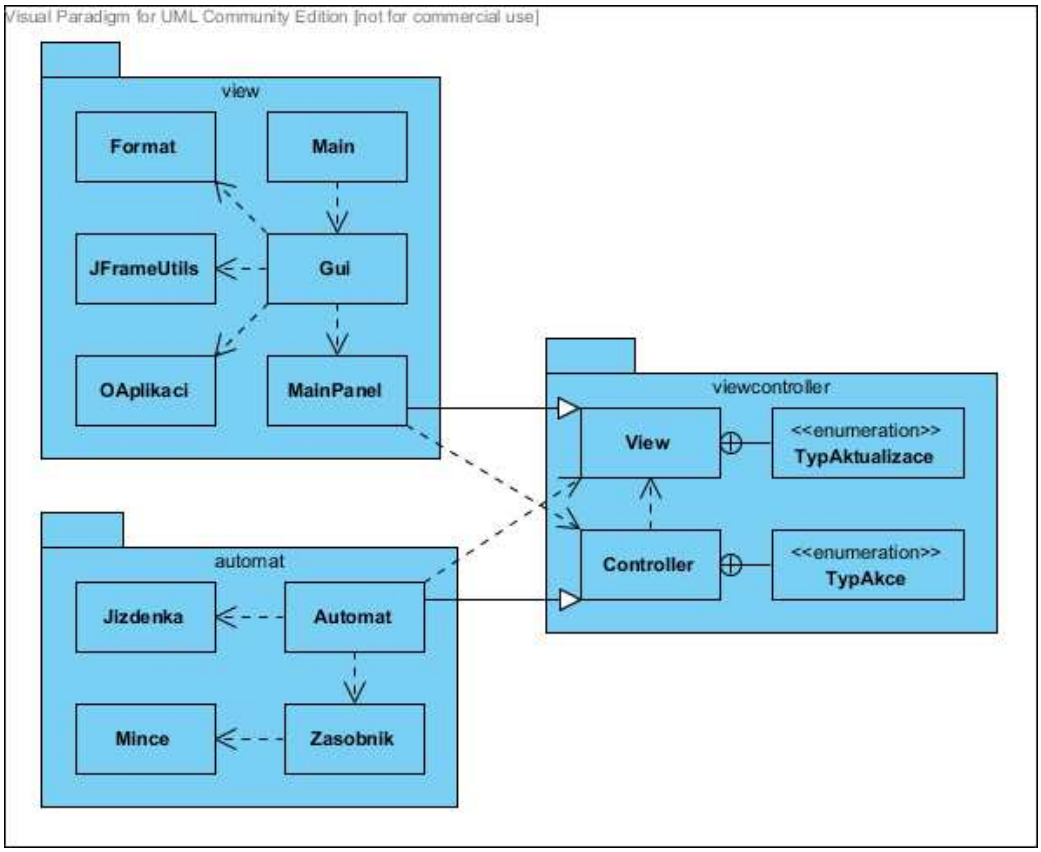

**Obr. 22: Automat – CRC karta – Zasobnik**

### 1.3.2. Diagram tříd




**Obr. 23: Automat – Diagram tříd**

### 1.3.3. Diagram balíčků




**Obr. 24: Automat – Diagram balíčků**

## 1.4. Podrobný návrh a implementace

### 1.4.1. Detailní popis tříd

**Balíček: automat**

Třída: Automat

- Proměnné
  - – View view
  - – ArrayList<Jizdenka> objednavka
  - – int vklad
  - – Zasobnik zasobnik
  - – Jizdenka[] jizdenky
- Metody
  - + void objednatJizdenku(Jizdenka jizdenka)
  - + void vhoditMinci(int hodnota)
  - + void storno()

Třída: Zasobnik

- Proměnné
  - + static final int[] HODNOTY
  - – ArrayList<Mince> mince
- Metody
  - + void pridatMinci(int hodnota)
  - + ArrayList<Mince> vratitObnos(int obnos)

Třída: Mince

- Proměnné
  - + static final String MENA
  - – int hodnota
  - – int pocet
- Metody
  - + int getHodnota()
  - + int getPocet()

Třída: Jizdenka

- Proměnné
  - + static final String[] DRUHY
  - + static final String[] PLATNOSTI
  - + static final int[][] CENY
  - – String druh
  - – int cena
  - – String platnost
- Metody
  - + String toString()

Jednotlivé konstanty – hodnoty mincí v zásobníku, měna mincí a jednotlivé typy jízdenek – by bylo možné nahradit výčtovým typem.

**Balíček: viewcontroller**

Rozhraní: View, Controller

- Viz str. 10

**Balíček: view**

Třída: Main

- Metody
  + static void main(String[] args)

Třída: Gui

- Proměnné
  – static final String NAZEV
  – static final String VERZE
  – static final String AUTOR
- Metody
  + void init()

Třída: MainPanel

- Proměnné
  – Controller controller
- Metody
  + void init()
  + void update(Typ typ, int index)
  + void update(Typ typ, int index, Object object)

Třídy: Format, JFrameUtils, OAplikaci

- Viz str. 11

### 1.4.2. Zdrojové kódy

- Zdrojové kódy jsou uvedeny v příloze.

## 1.5. Testování

Bylo provedeno testování grafického uživatelského rozhraní a třídy Zasobnik.

### 1.5.1. Třída: Zasobnik

Zdrojový kód je uveden v příloze.

**Metoda: pridatMinci**

- Vložení mincí v **platných** hodnotách.
- Vložení mincí v **neplatných** hodnotách.

**Metoda: vratitObnos**

- Vrácení zadaného obnosu, při kterém automat **má** na vrácení.
- Vrácení zadaného obnosu, při kterém automat **nemá** na vrácení.
- Vrácení zadaného obnosu, při kterém automat sice **má** na vrácení, ale pouze v nižších hodnotách mincí.

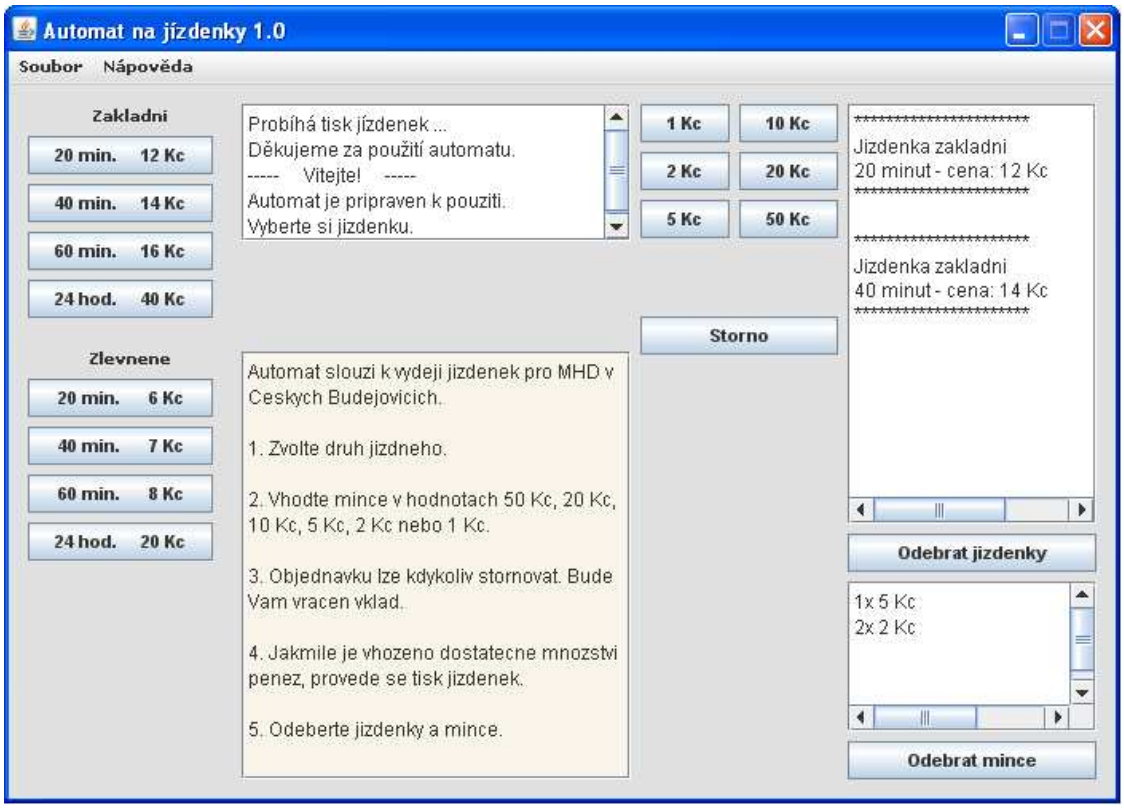
### 1.5.2. Grafické rozhraní

Testování grafického uživatelského rozhraní bylo provedeno prakticky. Testovány byly tyto části:

- Vhodit mince před objednáním jízdenky.
- Objednání jedné jízdenky.
- Objednání více jízdenek stejného i jiného druhu a jejich přiobjednání během vhazování mincí.
- Stornování objednávky před použitím automatu, po jeho použití a během objednávky.
- Objednání jízdenek ve chvíli, kdy automat nemá na vrácení.

Všechny objednávky byly testovány s přeplatkem a bez přeplatku. Jednotlivé fáze testování byly prováděny opakovaně a v kombinovaném pořadí.

## 1.6. Výsledek




**Obr. 25: Automat – Výsledek**

## 1.7. Udržování

### 1.7.1. Grafické rozhraní

V případě změny typů volených jízdenek nebo typů vhazovaných mincí, je třeba grafické rozhraní náležitě upravit. Pokud by došlo ke změně principu užívání automatu, je třeba aktualizovat zobrazovaný návod.

### 1.7.2. Algoritmická část

Přednastavené konstanty pro typy jízdenek, mincí a zásobník by bylo možné a také vhodné načítat z externího zdroje. Tato úprava je důležitá nejen kvůli případné změně jízdného nebo změně hodnot vhazovaných mincí, ale také by počet mincí v zásobníku zůstal aktuální i po ukončení aplikace.

### 1.7.3. Další možná rozšíření

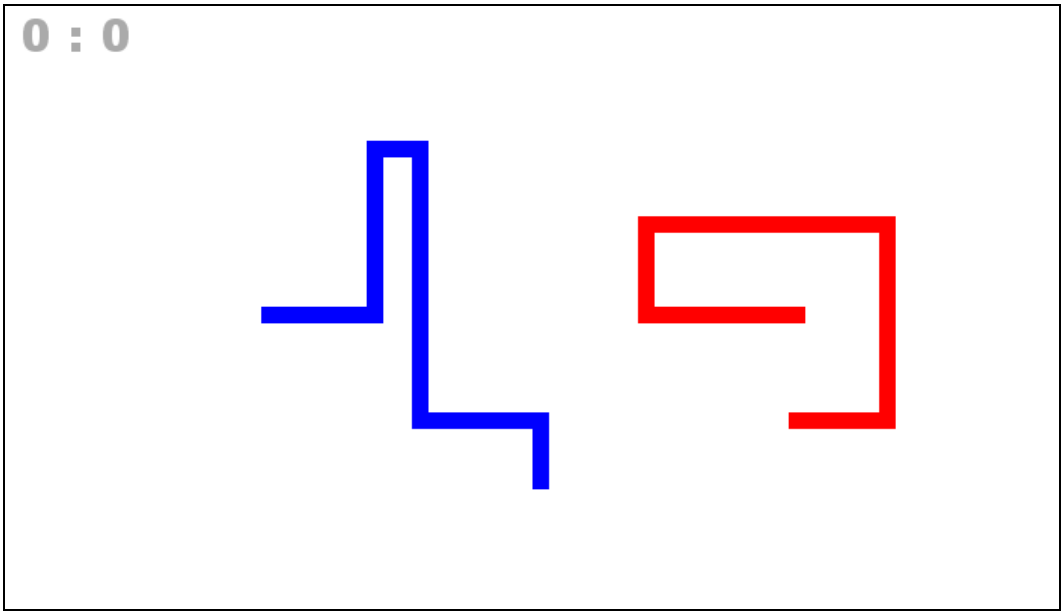
Grafickou podobu jízdenek by rovněž bylo možné načítat z externího zdroje. Grafickou i algoritmickou část by bylo možné rozšířit o správu zásobníku.

# 2. Tron

## O hře

Tron je hra realizována na čtvercové, případně obdélníkové ploše, která skládá se ze čtvercové mřížky a je tedy rozdělena do sloupců a řádků. Hry se účastní vždy 2 hráči. Každý z nich má v hrací ploše svoji počáteční pozici (viz Obr. 26). Po uplynutí určitého časového intervalu se oba hráči automaticky přesunou na vedlejší políčko v horizontálním nebo vertikálním směru, který je nastaven. Každý hráč může tento směr změnit pomocí ovládacích kláves. Všechna políčka, která již hráči navštívili, představují překážku pro oba hráče. Hra končí ve chvíli, kdy jeden z hráčů narazí na okraj hrací plochy nebo do políček, která navštívil, případně která navštívil jeho soupeř. Cílem hry je zůstat ve hře déle nežli soupeř.

S touto hrou je možné se setkat v různých modifikacích. Hráči mohou mít odlišené barvy. Vnější okraje hrací plochy mohou být průchozí na druhou stranu. Tempo hry se může zrychlovat (časový interval při přesunech hráčů mezi políčky by se snižoval). Hrací plocha může obsahovat náhodné či pevně dané překážky. Počáteční pozice hráčů se může měnit.

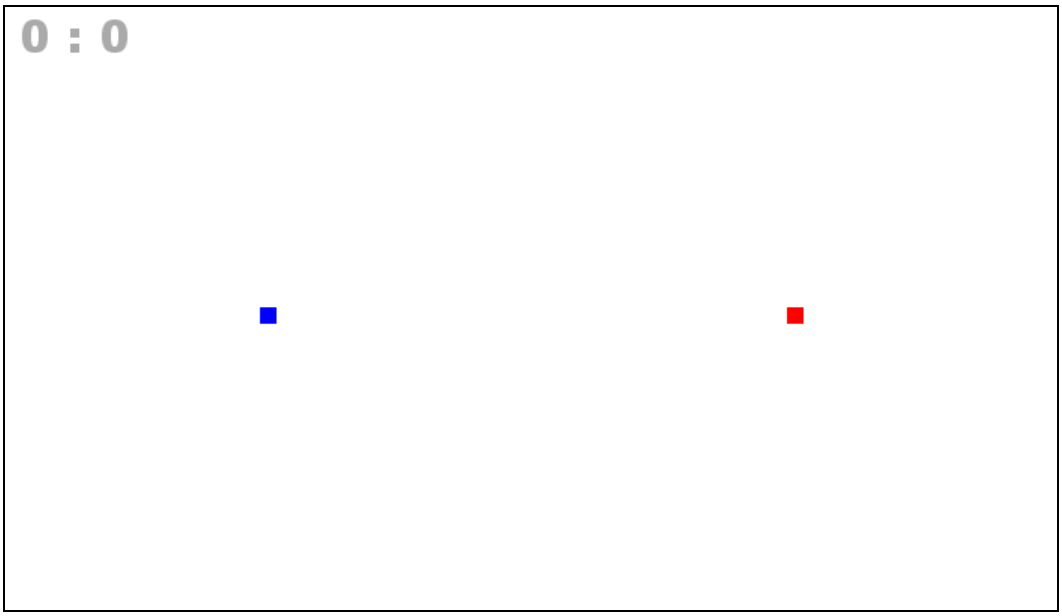



**Obr. 26: Tron – Ukázka hry**

## 2.1. Stanovení cílů

### Tron

Cílem je vytvořit aplikaci Tron podle uvedeného popisu v předchozí kapitole. Každý hráč bude mít pevně definovanou počáteční pozici (viz Obr. 27). Navštívená políčka jednotlivých hráčů budou barevně odlišena. Hráče bude možné ovládat pomocí tlačítek na klávesnici. Aplikace bude obsahovat menu s položkami pro nastavení a spuštění hry.




**Obr. 27: Tron - Počáteční pozice hráčů**

### Hrací plocha

Hrací plocha nebude obsahovat žádné překážky a její velikost bude ve tvaru obdélníku. Na výběr bude několik možností pro změnu velikosti políček a pro změnu počtu sloupců a řádků ve čtvercové mřížce. Pro větší přehlednost bude možné zobrazit či skrýt mřížku s políčky.

### Tempo hry

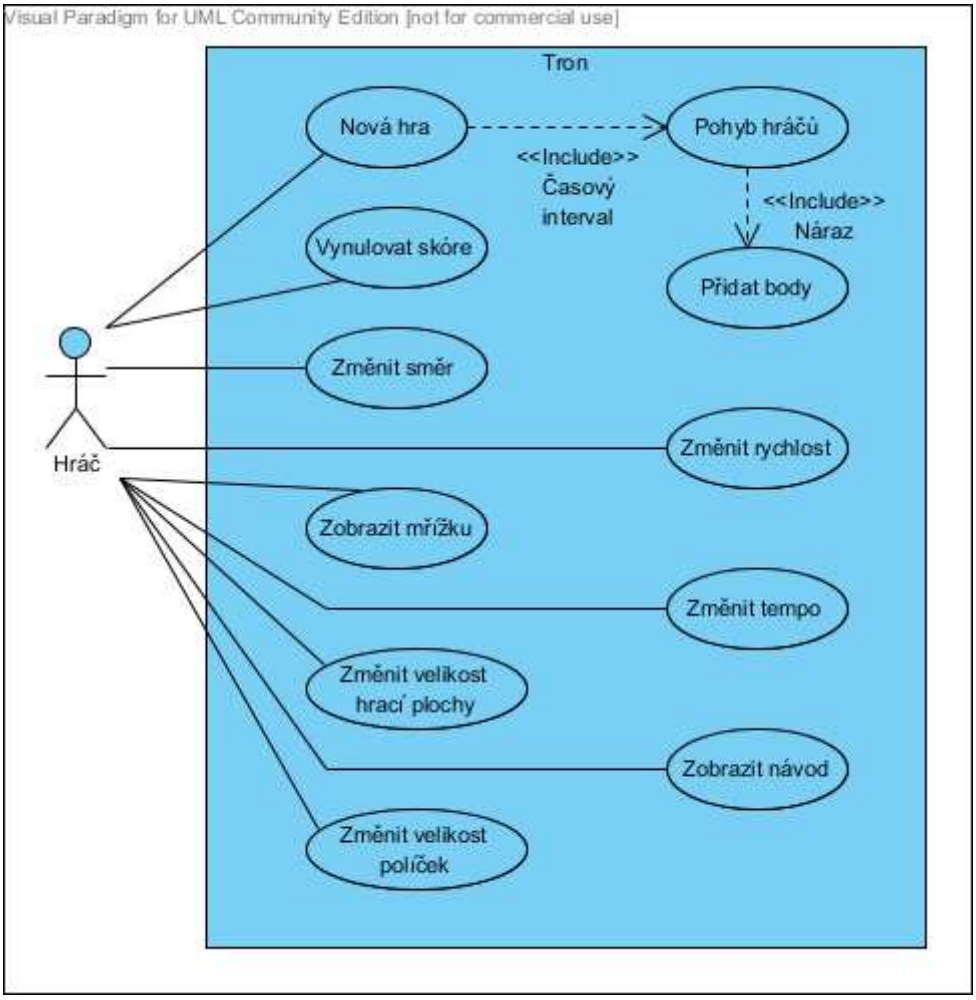
Časový interval při přesunech hráčů mezi políčky bude možné měnit. Mezi možnostmi budou 3 rychlosti: pomalu, normálně a rychle. Dále bude možné změnit tempo hry. Na výběr bude konstantní tempo a tempo, při kterém se hráči budou pohybovat rychleji (časový interval při pohybech hráčů se bude snižovat).

### Skóre

Po každé hře se vítěznému hráči připíše bod. V případě remízy dostanou bod oba hráči. Průběžné skóre bude zobrazeno na hrací ploše. Skóre bude možné vynulovat.

## 2.2. Analýza

### 2.2.1. Diagram případů užití (Use Cases)




**Obr. 28: Tron – Diagram případů užití**

**Nová hra**

Vyprázdní hrací plochu, nastaví hráčům počáteční pozici a připraví aktuální nastavení hry. Následně je proveden pohyb hráčů.

**Pohyb hráčů**

Pohyb hráčů je prováděn automaticky a opakovaně po nastaveném časovém intervalu. Pokud je nastavené zrychlené tempo hry, dojde ke snížení tohoto intervalu. Pohyb se opakuje do té doby, než jeden z hráčů narazí do okraje hrací plochy nebo do políček, která navštívil, případně která navštívil jeho soupeř. Během přesunu hráčů je možně změnit směr jejich pohybu pomocí ovládacích kláves. Jakmile celý tento proces skončí, dojde k udělení bodů.

**Přidat body**

Pokud jeden z hráčů narazí, dojde k aktualizaci bodového stavu. Po tomto procesu dojde k zastavení hry.

**Vynulovat skóre**

Vynuluje skóre obou hráčů.

**Změnit směr**

Každý hráč má k dispozici 4 ovládací klávesy. Dvě pro změnu vertikálního směru a dvě pro změnu horizontálního.

**Zobrazit mřížku**

Zobrazí čtvercovou mřížku hrací plochy. Mřížku je možné deaktivovat.

**Změnit velikost hrací plochy**

Změní velikost hrací plochy podle počtu sloupců a řádků a překreslí hrací plochu.

**Změnit velikost políček**

Změní velikost všech políček a překreslí hrací plochu.

**Změnit rychlost**

Změní nastavení časového intervalu při přesunech hráčů mezi políčky. Změna se projeví až při novém spuštění hry.

**Změnit tempo**

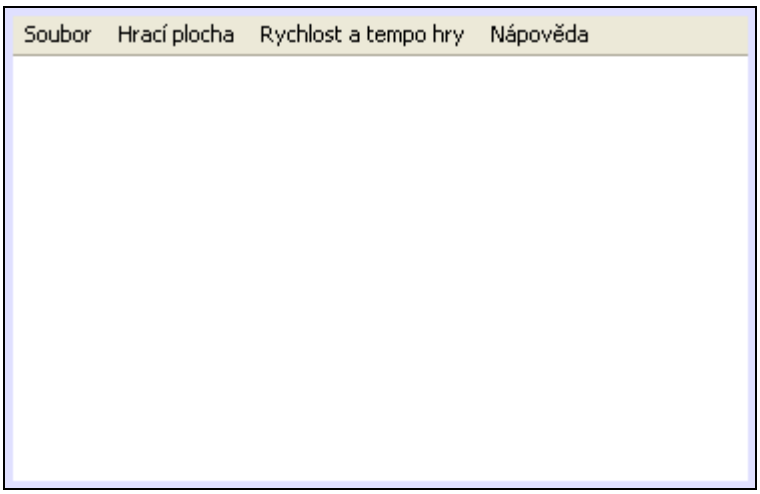
Změní tempo hry. Změna se projeví až při novém spuštění hry.

**Zobrazit návod**

Zobrazí nové okno s detailním popisem hry.

### 2.2.2. Analýza tříd

**Grafický návrh aplikace**




**Obr. 29: Tron – Návrh**

**Seznam tříd**

- Main – zajistí zobrazení grafického uživatelského rozhraní.
- Gui – grafické uživatelské rozhraní, obsahuje menu a panel s hrací plochou.
- HraciPlocha – panel, který reprezentuje hrací plochu a je součástí Gui. Vykresluje komponenty a předává události algoritmické části.
- Tron – zajišťuje chod hry. Přijímá události z grafického rozhraní.
- Hrac – obsahuje informace o jednom hráči: pozice hráče, barva, ovládací klávesy, směr pohybu, počet vítězství a navštívená políčka. Dále obsahuje informaci o tom, zda hráč narazil. Zajišťuje pohyb hráče a umožňuje změnit jeho směr.
- Policko – reprezentuje jedno políčko v hrací ploše, obsahuje informaci o své pozici a velikosti.

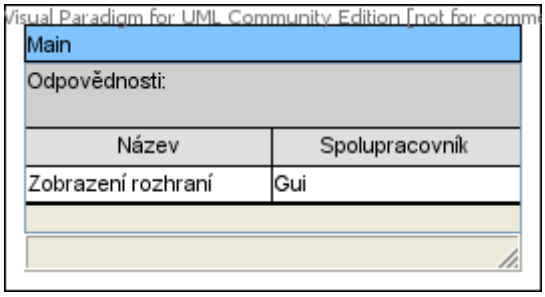
**Rozhraní**

- View, ControllerExtended – viz str. 5

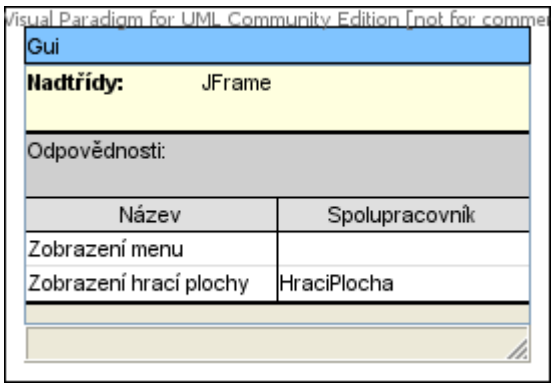
## 2.3. Objektový návrh

### 2.3.1. CRC karty

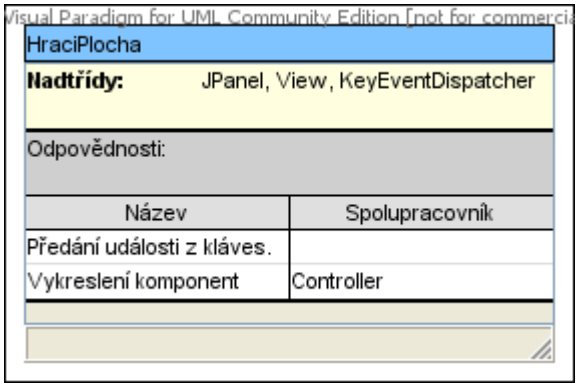
Karty: Format, JFrameUtils, OAplikaci – viz str. 7



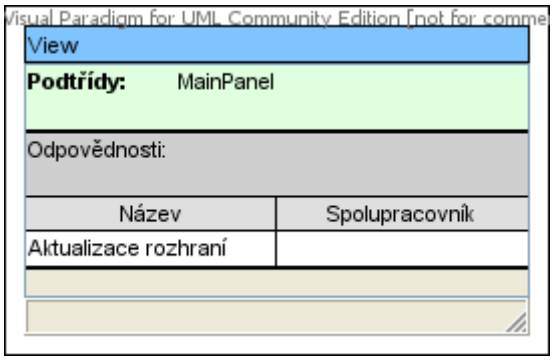

**Obr. 30: Tron – CRC karta – Main**



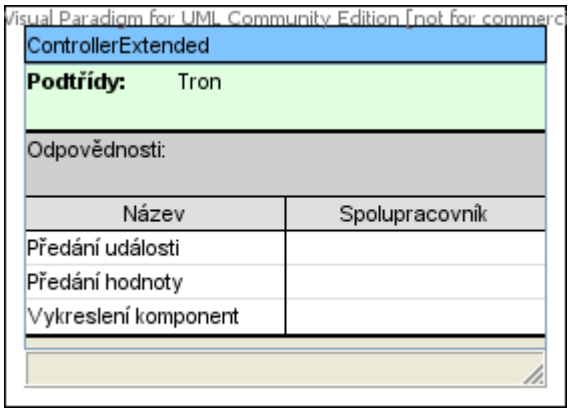

**Obr. 31: Tron – CRC karta – Gui**



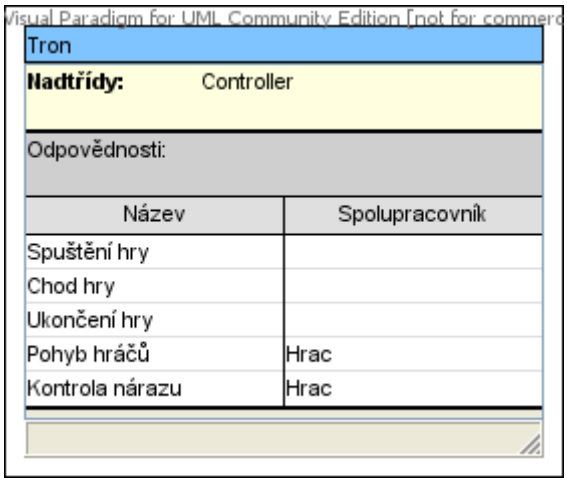

**Obr. 32: Tron – CRC karta – HraciPlocha**



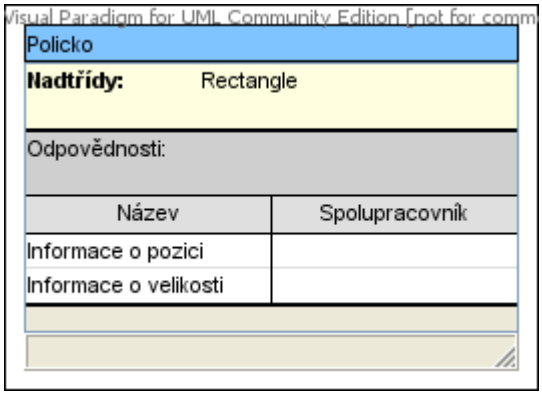

**Obr. 33: Tron – CRC karta – View**



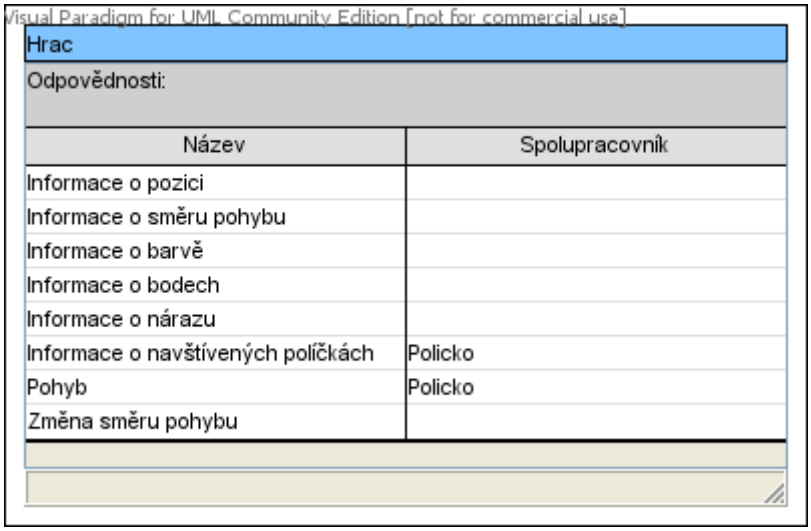

**Obr. 34: Tron – CRC karta – ControllerExtended**




**Obr. 35: Tron – CRC karta – Tron**

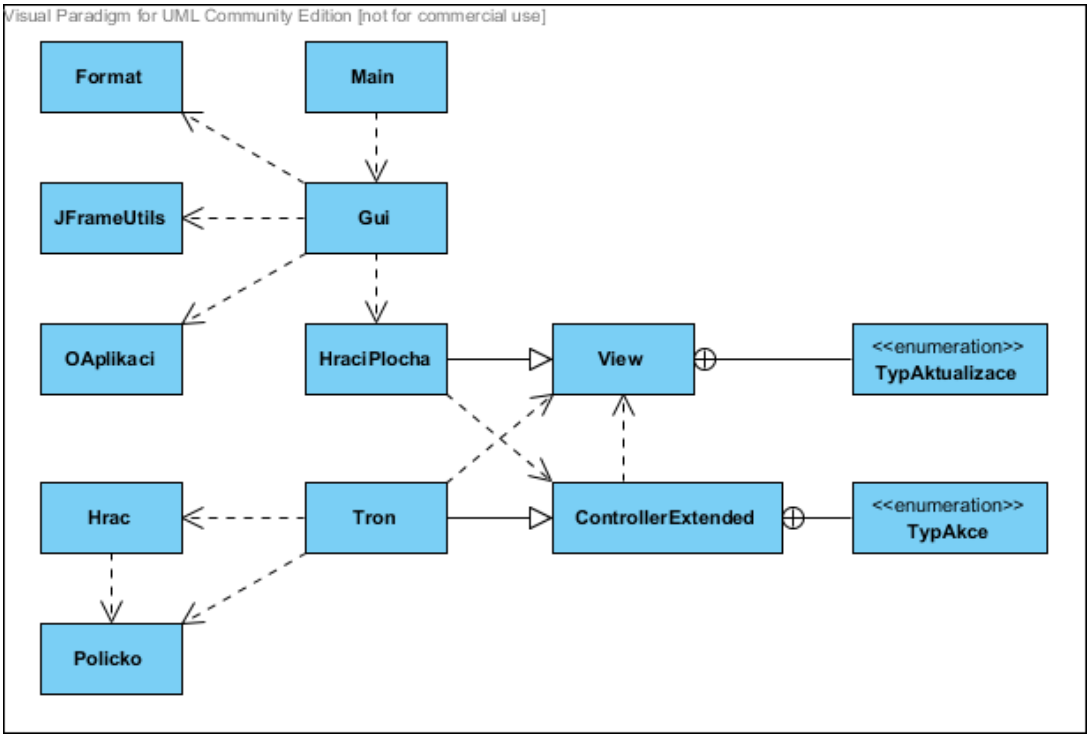



**Obr. 36: Tron – CRC karta – Policko**



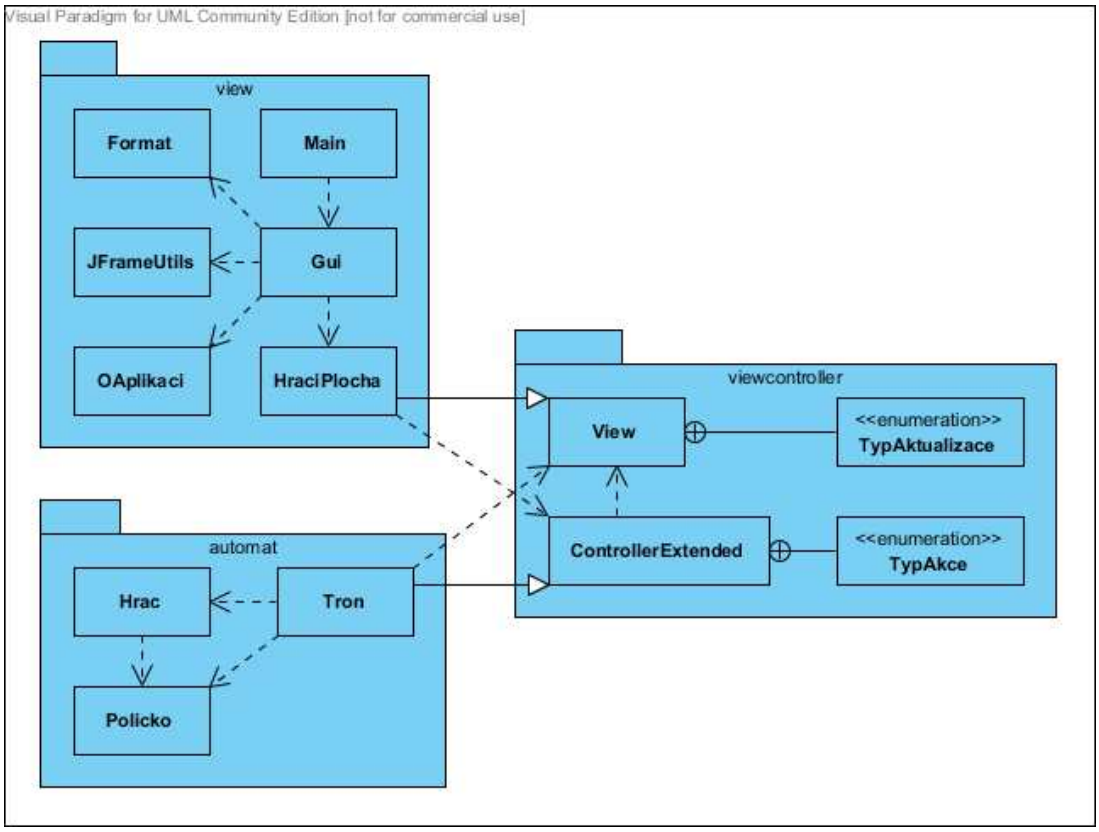

**Obr. 37: Tron – CRC karta – Hrac**

### 2.3.2. Diagram tříd




**Obr. 38: Tron – Diagram tříd**

### 2.3.3. Diagram balíčků




**Obr. 39: Tron – Diagram balíčků**

## 2.4. Podrobný návrh a implementace

### 2.4.1. Detailní popis tříd

**Balíček: tron**

Třída: Tron

- Proměnné
  - static final int[] RYCHLOSTI
  - static final int[][] VELIKOSTI
  - static final int[] POLICKA
  - View view
  - Thread thread
  - boolean spusteno
  - Hrac hrac1
  - Hrac hrac2
  - int rychlost
  - int sloupcu
  - int radku
  - int velikostPolicka
- Metody
  + void start()
  + void stop()

Třída: Hrac

- Proměnné
  + static final Color BARVA_1
  + static final Color BARVA_2
  – int x, y
  – ArrayList<Policko> policka
  – byte smerX
  – byte smerY
  – Color barva
  – int body
  – boolean narazil
  – int[] sipky
- Metody
  + ArrayList<Policko> getPolicka()
  + Color getBarva()
  + boolean isNarazil()
  + void pohyb()
  + boolean zmenitSmer(int keyCode)
  + void incrementBodyVyhra()
  + void incrementBodyRemiza()
- Směr pohybu hráče by bylo možné nahradit výčtovým typem.

Třída: Policko

- Proměnné
  + static int velikost
- Konstruktory
  + Policko(int x, int y)

**Balíček: viewcontroller**

Rozhraní: View, ControllerExtended

- Viz str. 10

**Balíček: view**

Třída: Main

- Metody
  + static void main(String[] args)

Třída: Gui

- Proměnné
  – static final String NAZEV
  – static final String VERZE
  – static final String AUTOR
  – HraciPlocha hraciPlocha
- Metody
  + void init()

Třída: HraciPlocha

- Proměnné
  – static final Color POZADI
  – static final Color SKORE
  – ControllerExtended controller
  – JLabel skore
- Metody
  + void init()
  + void update(Dimension dimension)
  + void updateSkore()
  + void update (Typ typ, int index)
  + void update(Typ typ, int index, Object object)
  + void paint(Graphics g)
  + boolean dispatchKeyEvent(KeyEvent e)

Třídy: Format, JFrameUtils, OAplikaci

- Viz str. 11

## 2.4.2. Zdrojové kódy

- Zdrojové kódy jsou uvedeny v příloze.

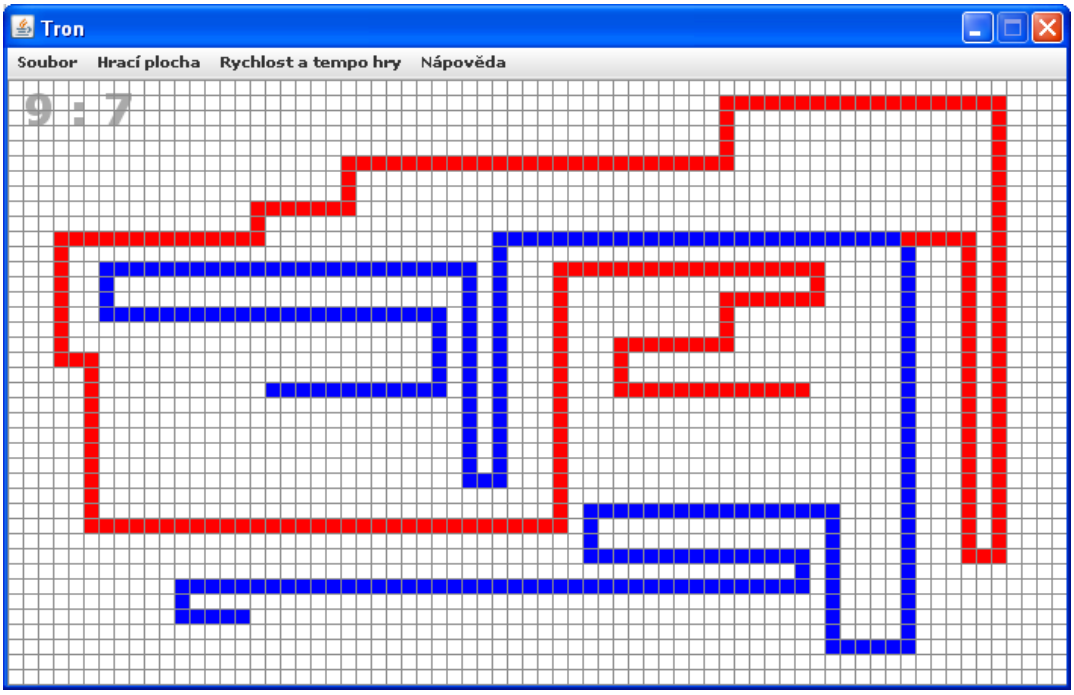
## 2.5. Testování

Vzhledem k tomu, že v aplikaci šlo především o grafickou stránku, závislou na čase, bylo testování provedeno ručně. Testovány byly tyto části:

- Náraz do vlastních políček.
- Náraz do soupeřových políček.
- Náraz do okraje hrací plochy.
- Opětovné spuštění hry (i před ukončením hry předchozí).
- Opakované zobrazení a skrytí mřížky.
- Překreslení hrací plochy při změně velikosti.
- Nastavení všech rychlostí a temp hry a následné testování průběhu hry.
- Vynulování skóre.

Jednotlivé fáze testování byly prováděny opakovaně a v kombinovaném pořadí.

## 2.6. Výsledek




**Obr. 40: Tron – Výsledek**

## 2.7. Udržování

### 2.7.1. Grafické rozhraní

Grafické rozhraní by mohlo být obohaceno o nastavení barev a ovládacích kláves obou hráčů.

### 2.7.2. Algoritmická část

Přednastavené konstanty pro políčka, hráče a hru samotnou by bylo možné a také vhodné načítat z externího zdroje. Na výběr by mohlo být více typů hracích ploch a herních pozic:

- Pevně dané nebo dynamicky vytvářené překážky.
- Průchozí strany.
- Různé počáteční pozice hráčů.

Pokud by hráč neměl svého soupeře, mohl by se účastnit hry s počítačem. Tento počítač by mohl mít více úrovní, atd.

# Zhodnocení výsledků a závěr

Dílčí cíle práce byly splněny. Implementace šablony v navržených projektech proběhla úspěšně. V jednotlivých projektech jsme mohli zaznamenat společné prvky, které by se daly zařadit do obecné šablony. Také se objevil odlišný přístup při fázi testování, kde u projektu Tron nebylo možné provést testování automatizovaně triviálním způsobem. Pro účely tohoto testování by mohly být navrženy speciální třídy a opět zahrnuty do obecné šablony. Na škodu je pouze fakt, že z důvodu omezení rozsahu práce nebylo zahrnuto více projektů, které by mohly posloužit k dalšímu hodnocení a srovnání.

Univerzální šablona byla navržena podle současných metod a postupů, nicméně jsme se přesvědčili o tom, že pokus o kompletní abstrakci, aby zahrnovala všechny případy, které by mohly nastat, není možný. Bylo by možné navrhnout sadu šablon, které by byly zaměřené na jednotlivé typy úloh.

Výsledky práce, univerzální šablona a vypracované projekty, by mohly posloužit k výuce objektově orientovaného programování na Jihočeské univerzitě v Českých Budějovicích.

# Seznam použitých zdrojů


1. BJORK, Russell C. In: *Object-Oriented Software Development* [online]. 2004 [cit. 2012-04-23]. Dostupné z: http://www.cs.gordon.edu/courses/cs211/ATMExample/index.html

2. BJORK, Russell C. In: *Object-Oriented Software Development* [online]. 2008 [cit. 2012-04-23]. Dostupné z: http://www.cs.gordon.edu/courses/cs211/AddressBookExample/index.html

3. TSANG, Curtis. In: *Visual Paradigm* [online]. [cit. 2012-04-23]. Dostupné z: http://www.visual-paradigm.com/documentation/

4. FALK, Alexander. In: *Altova* [online]. Aktualizace… [cit. 2012-04-23]. Dostupné z: http://www.altova.com/umodel.html

5. LARMAN, Craig. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition). Prentice Hall, 2004. ISBN 978-0131489066

6. BARNES, David J., KÖLLING Michael. Objects First with Java: A Practical Introduction Using BlueJ (5th Edition). Prentice Hall, 2011. ISBN 978-0132492669

# Přílohy

Součástí přílohy jsou zdrojové kódy, které jsou zahrnuty v přiložených souborech ve složce s projekty.