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 2013

Bibliografické údaje

Roháček B., 2013: Soubor projektů pro potřeby výuky objektově orientovaného programování. [A set of projects for teaching object-oriented programming. Bc. Thesis, in Czech.] – 47 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Cílem této bakalářské práce je vytvořit univerzální scénář 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 vytvořeny podle navrženého scénáře. Výsledky práce bude možné využít při výuce objektově orientovaného programování.

Abstract

The goal of this thesis is to create an universal scenario 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 developed by proposed scenario. 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, 24. 4. 2013

.....

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

Slovníček pojmů

OOP – Object-Oriented-Programming či objektově orientované programování

CRC – Class-Responsibility-Collaboration, neboli Třída-Odpovědnost-Spolupráce, je technika užívaná při vývoji softwaru

UML – Unified Modeling Language je jazyk, který slouží pro vizualizaci návrhů projektů

Observer – návrhový vzor, který je založen na pozorování jednoho objektu, který může být pozorován mnoha jinými objekty – pozorovateli

MVC – Model-View-Controller je návrhový vzor určen k návrhu architektury aplikace

JRE – Java Runtime Environment, běhové prostřední (virtuální stroj) potřebné ke spuštění Java aplikací

JDK – Java Development Kit, sada nástrojů (knihovny, překladače, debugger, javadoc, …), která je nezbytná k vývoji aplikací, zároveň obsahuje běhové prostředí JRE

IDE – Integrated Development Environment, neboli vývojové prostředí, které slouží k vývoji aplikací (IDE k vývoji Java aplikací vyžaduje JDK)

GUI – graphical user interface, v překladu grafické uživatelské rozhraní, umožňuje ovládat aplikaci pomocí klávesnice, myši a dalších ovládacích prvků

Interface – neboli rozhraní, je klíčové slovo jazyka Java, jedná se o konstrukci, která definuje množinu konstant a metod, které musí implementující třídy obsahovat

Screenshot – anglický termín pro snímek obrazovky, uloží do paměti aktuální podobu obrazovky, kterou je možné vložit do grafického editoru, případně využívat dle potřeby

1 Úvod

Výuka objektově orientovaného programování je dnes zastoupena prakticky ve všech oborech informatiky na všech vysokých školách a téměř žádný informatik se bez přinejmenším základních znalostí programování takřka neobejde. S programováním se setkáváme všude, ať už se jedná o využití obecných algoritmických postupů či aplikaci v konkrétních oblastech, jako jsou databázové systémy, webové stránky nebo samotné programování komerčního softwaru, atd.

Úkolem práce je vytvořit jednotný postup, jak navrhnout a naprogramovat rozsahem menší aplikace. Důraz bude kladen na objektovou analýzu. Nejprve je třeba navrhnout sadu projektů, které budou realizovány podle univerzálního scénáře. Scénář bude rozdělen do jednotlivých fází, které budou detailně dokumentovány a mohou obsahovat alternativní řešení. Vyvrcholením všeho bude implementace návrhů v jazyce Java, následováno okomentováním zdrojového kódu a zakončeno rozborem celé realizace.

Hlavním důvodem ke zpracování tohoto tématu je fakt, že aplikace tohoto přístupu ve výuce zcela chybí, a že studenti tento pohled na věc neznají a postrádají. Tato práce může napomoci studentům k lepšímu pochopení vývoje softwaru a vyučujícím v jejich výkladu. Téma jsem si zvolil z důvodu zájmu o programování a o výuku jako takovou. Jako student a začínající programátor dobře vím, co studenti potřebují.

Při výběru projektů a návrhu univerzálního scénáře jsem se nechal vést hotovými řešeními, například projekty *Automated Teller Machine* (*ATM*) či *Address Book* od autora Bjork Russela [1, 2] a dále projekty z knihy *Objects First* [6]. Zároveň jsem využil inspirace ve vlastních projektech, nápadech a zkušenostech, které bych rád zužitkoval. Zaměřil jsem se především na obsah jednotlivých fází vývoje a na techniky, které byly využity

Výsledky této práce mohou posloužit jako výukový materiál při výuce objektově orientovaného programování či jako studijní materiál pro studenty. Jelikož projekty budou napsány v jazyce Java a jsou určeny spíše pro pokročilejší programátory, je k jejich úplnému pochopení vyžadována alespoň základní znalost programovacího jazyka Java a znalost objektově orientovaného programování obecně.

2 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 podle jednotného scénáře využitelného při řešení problémů malého rozsahu s důrazem na objektovou analýzu a návrh. Při zpracování tématu je nutné dosáhnout těchto dílčích cílů:

1. Navrhnout jednotný scénář použitelný pro řešení problémů malého rozsahu v rámci kurzu objektově orientovaného programování, který bude zahrnovat tyto fáze vývoje:
   a. Stanovení cílů
   b. Analýza
   c. Objektový návrh
   d. Detailní popis tříd
   e. Implementace
   f. Testování
   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 prostředí NetBeans (www.netbeans.org) a BlueJ (www.bluej.org).

3 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 jednotný scénář, dle kterého 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.

4 Současný stav problematiky

V současné době je možné se setkat s mnoha modely vývoje softwaru, které zahrnují seznam postupů a pravidel. Mezi nejvýznamnější patří například vodopádový (sekvenční), iterativní, spirálový či prototypový model [5]. K samotné tvorbě návrhu a celkovému vývoji aplikace slouží nástroje a techniky k tomu určené…

Techniky CRC a UML

CRC

Technika CRC karet je určena pro specifikaci odpovědností jednotlivých tříd, a proto je ideální složkou pro počátek analýzy. Class-Responsibility-Collaboration, neboli Třída-Odpovědnost-Spolupráce, jak už samotný název napovídá, obsahuje název třídy, seznam odpovědností a spolupracující třídy, které se na odpovědnostech podílejí.

UML

UML, celým názvem The Unified Modeling Language, je jazyk, který slouží pro vizualizaci návrhů programových systémů. Součástí standardu UML jsou diagramy, z nichž tyto budou využity v rámci této práce:

- Diagram tříd
- Diagram balíčků
- Diagram případů užití

Návrhový vzor Observer a MVC

Observer

Návrhový vzor Observer je založen na pozorování jednoho objektu, který může být pozorován mnoha jinými objekty. Pokud v pozorovaném objektu dojde ke změně, objekt o této změně upozorní všechny své pozorovatele. Tento vzor se stal klíčovou součástí návrhového vzoru MVC.

V Javě je tento model realizován pomocí složky interface Observer a třídy Observable. Pozorovaná třída vlastní objekt třídy Observable, pomocí kterého upozorňuje své pozorovatele, a sice voláním metody *notifyObservers()*, případně *notifyObservers(Object arg)*. Každý pozorovatel musí implementovat interface Observer spolu s metodou *update(Observer o, Object arg)*, prostřednictvím které je třída upozorněna.

MVC

Návrhový vzor Model-View-Controller, ve zkratce MVC, je jeden z architektonických vzorů, který je určen k vývoji aplikací. Základem je rozdělení projektu 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 s procesy v aplikaci
- **View:** neboli pohled, převádí uživateli data do grafické podoby
- **Controller:** reaguje na události a zajišťuje změny v datovém modelu nebo pohledu

Nástroje a vývojová prostředí

K vývoji aplikace je možné využít některé z řady volně dostupných nástrojů. Pro tvorbu UML diagramů a CRC karet je vhodný program Visual Paradigm for UML, který poskytuje mnoho funkcí a je na vytváření diagramů ideální.

K tvorbě Java aplikací je nezbytné nainstalovat JDK. Mezi využívaná vývojová prostředí patří například program NetBeans, Eclipse nebo BlueJ, který je vhodným nástrojem k výuce OOP. Projekty v této práci budou zpracovány v programu NetBeans.

5 Univerzální scénář

Zahrnuje seznam kroků, dle kterých budeme postupovat a soubor technik, které budeme využívat při tvorbě všech projektů od samotného počátku až do jejich finální podoby. Scénář zahrnuje techniku CRC karet, vybrané UML diagramy, návrhový vzor Observer a MVC.

Mezi zvolené UML diagramy patří:

- Diagram případů užití (Use Cases)
- Diagram tříd
- Diagram balíčků

Scénář se skládá z těchto fází:

1) Stanovení cílů
2) Analýza – zahrnuje diagram případů užití
3) Objektový návrh – zahrnuje CRC karty, diagram tříd a diagram balíčků
4) Detailní popis tříd
5) Implementace
6) Testování
7) Udržování

5.1 Stanovení cílů

V první řadě je třeba uvést, čeho se daná problematika týká a vysvětlit pojmy potřebné k pochopení zadání. Následuje formulace požadavků, detailní specifikace zadání – co od aplikace očekáváme, jaké by měla mít funkce a parametry, jak by se měla v konkrétních situacích chovat, případně jak by aplikace měla graficky vypadat.

Takto jednoznačně definované zadání ale samozřejmě neurčuje žádný standard, jak lze onu danou problematiku řešit. Jde pouze o jeden pohled na věc. Jedno řešení tématu, které je možné uchopit a realizovat mnoho jinými způsoby.

5.2 Analýza

Analýza se skládá z diagramu případů užití, z analýzy jednotlivých tříd, případně z grafického návrhu (pokud se jedná o aplikaci s grafickým uživatelským rozhraním).

5.2.1 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 jinými procesy. V prvním případě se jedná o procesy spouštěné pomocí grafického rozhraní. V druhém případě jde o vnitřní procesy aplikace. Každý účastník může mít k dispozici jiné procesy anebo mohou mít některé procesy společné. Jednotlivé procesy se mohou mezi sebou navzájem ovlivňovat a spouštět jeden druhého.

Tato fáze obsahuje nejen samotný diagram, ale také seznam všech procesů s detailním popisem. Přesně určuje, za kterých podmínek a v jakém pořadí je volán který proces. Diagramy budou tvořeny v programu Visual Paradigm for UML.

5.2.2 Grafický návrh aplikace

Určuje rozmístění jednotlivých komponent v aplikaci. Návrh může být popsán slovně nebo vyjádřen formou obrázku, případně obrázky. Ke konstrukci návrhu může posloužit editor v programu Netbeans, kde jsou k dispozici všechny typické základní grafické komponenty.

5.2.3 Analýza tříd

Analýza tříd zahrnuje seznam tříd a interfaces. Všechny tyto komponenty budou obsahovat stručný popis, co reprezentují a k čemu slouží. Analýza může obsahovat alternativní rozbor jednotlivých tříd.

S ohledem na vyšší přehlednost a přesné určení, kde program začíná, bude ve všech projektech existovat třída Main, obsahující pouze spouštěcí metodu main.

5.3 Objektový návrh

Objektový návrh tvoří technika CRC karet, diagram tříd a diagram balíčků. Zaměřuje se na odpovědnosti a závislosti jednotlivých tříd a jejich rozdělení do logických celků v podobě balíčků.

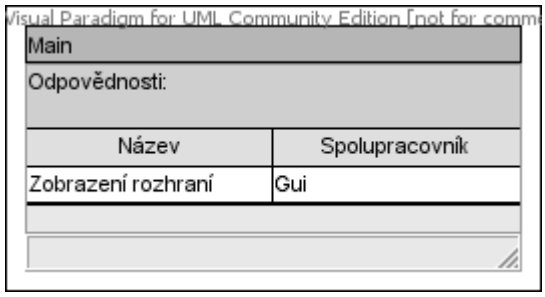
5.3.1 CRC karty

Technika CRC karet je odvozena z těchto pojmů:

- Class (třída)
- Responsibility (odpovědnost)
- Collaboration (spolupráce)

V této kapitole půjde o sadu karet, přičemž každá karta reprezentuje právě jednu třídu. Jednotlivé karty obsahují seznam odpovědností, které třída má a informaci o tom, se kterými třídami na těchto odpovědnostech spolupracuje. Karta může dále obsahovat seznam nadtříd, podtříd a seznam implementovaných interfaces. [6]

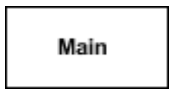
CRC karta třídy Main:

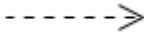
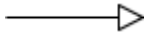


Obrázek 1: Scénář - CRC karta - Main

5.3.2 Diagram tříd

Diagram obsahuje seznam tříd a jejich vazby mezi sebou. Znázorňuje přehledný a uspořádaný systém mezi třídami. Diagramy budou tvořeny pomocí programu Visual Paradigm for UML. Značení uvnitř diagramu v tomto programu vypadá následovně:

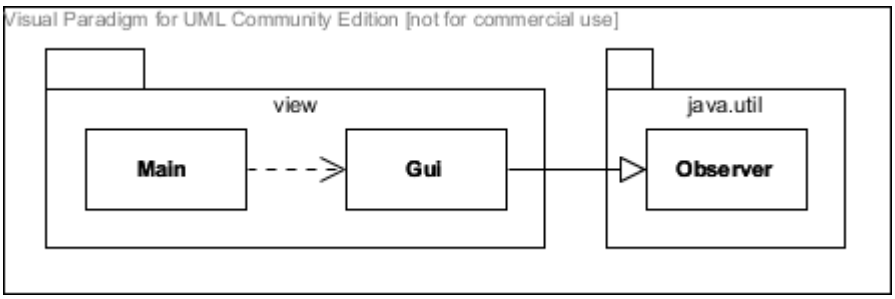
- Třída: 

- Závislost: závislá třída  závisející třída
- Dědičnost: podtřída  nadtřída
- Implementace rozhraní: implementující třída  rozhraní

5.3.3 Diagram balíčků

Diagram balíčků graficky vyjadřuje zapouzdření tříd a závislost mezi samotnými balíčky. K vytváření diagramů bude opět použit program Visual Paradigm for UML.

Příklad:



Obrázek 2: Scénář - Diagram balíčků

- Balíčky: view, java.util
- Závislosti: třída Main je závislá na třídě Gui
- Implementace: třída Gui implementuje rozhraní Observer

5.4 Detailní popis tříd

Tato fáze obsahuje seznam tříd s podrobným popisem jejich struktury. Nejprve je uvedena nadtřída (pokud existuje) a všechna implementovaná rozhraní včetně popisu jejich využití v dané třídě. Následuje rozdělení třídy:

- Konstanty
- Výčtové typy
- Instanční proměnné
- Konstruktory
- Metody
- Vnitřní třídy a rozhraní

Tyto jednotlivé části obsahují výčet prvků s jejich detailním rozborem. U všech těchto prvků je uvedena jedna z následujících přístupností:

– pro private

∗ pro protected

\+ pro public

Seznam tříd je pro lepší orientaci rozdělen do balíčků, ve kterých se třídy nacházejí.

5.5 Implementace

Všechny analýzy a návrhy jsou za námi a je na řadě samotná realizace navrhnutého řešení, tedy tvorba zdrojového kódu. Jako vývojové prostředí byl zvolen program NetBeans.

Zdrojové kódy všech projektů budou uvedeny v příloze a budou zahrnovat komentáře (především dokumentační). Řádkové komentáře budou sloužit (nejen) k objasnění algoritmů.

Na závěr bude vytvořena dokumentace, která shrne průběh implementace a její výsledky a bude se skládat ze dvou částí:

- Problémy a změny při implementaci
- Výsledný screenshot

5.5.1 Problémy a změny při implementaci

V této kapitole jsou popsány problémy, které při psaní kódu vznikly a jakým způsobem byly řešeny. Dále jsou zde popsány změny v jednotlivých třídách oproti původnímu návrhu. Zásadní je si uvědomit, že čím kvalitnější návrh aplikace je, tím méně bude nutné provést změn při jeho implementaci.

5.5.2 Výsledný screenshot

Tato část bude obsahovat oříznutý screenshot s aplikací za jejího běhu. Je zde vidět srovnání, jak se liší výsledná podoba aplikace od původního grafického návrhu.

5.6 Testování

Testování aplikace bývá ze strany studentů často zanedbáváno, někdy dokonce zcela vynecháno, nicméně této fázi vývoje je třeba věnovat zvýšenou pozornost, jelikož během testů je možné odhalit mnoho nedostatků a chyb. Testovací třídy a metody je možné zavádět již během tvorby jednotlivých tříd.

Dokumentace provedených testů bude obsahovat seznam testovaných metod včetně popisu, jak byly testy provedeny. V našem případě implementace Javy budou využity jednotkové testy JUnit. Grafické rozhraní bude testováno manuálně a jednotlivé testy budou rovněž dokumentovány. Rozdělení testů v dokumentaci bude následovné:

- Test grafického rozhraní
- Test jednotlivých tříd

Zdrojové kódy testovacích tříd budou uvedeny v příloze.

5.7 Udržování

Tato fáze zahrnuje návrhy a rady, jak postupovat při dalším vývoji aplikace.

V první řadě je třeba projít zdrojové kódy a provést refaktoring. Upravit a vyladit zdrojový kód tak, aby byl co nejsnadněji rozšiřitelný. Dále se zamyslet nad optimalizací současného řešení, případně navrhnout alternativní postupy. Navrhnout rozšíření aplikace o další možnosti. Vracíme se tak zpět na počátek, kde si můžeme stanovit nové cíle, nové požadavky a projít celým procesem od začátku.

6 Projekty

6.1 Automat na jízdenky

6.1.1 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

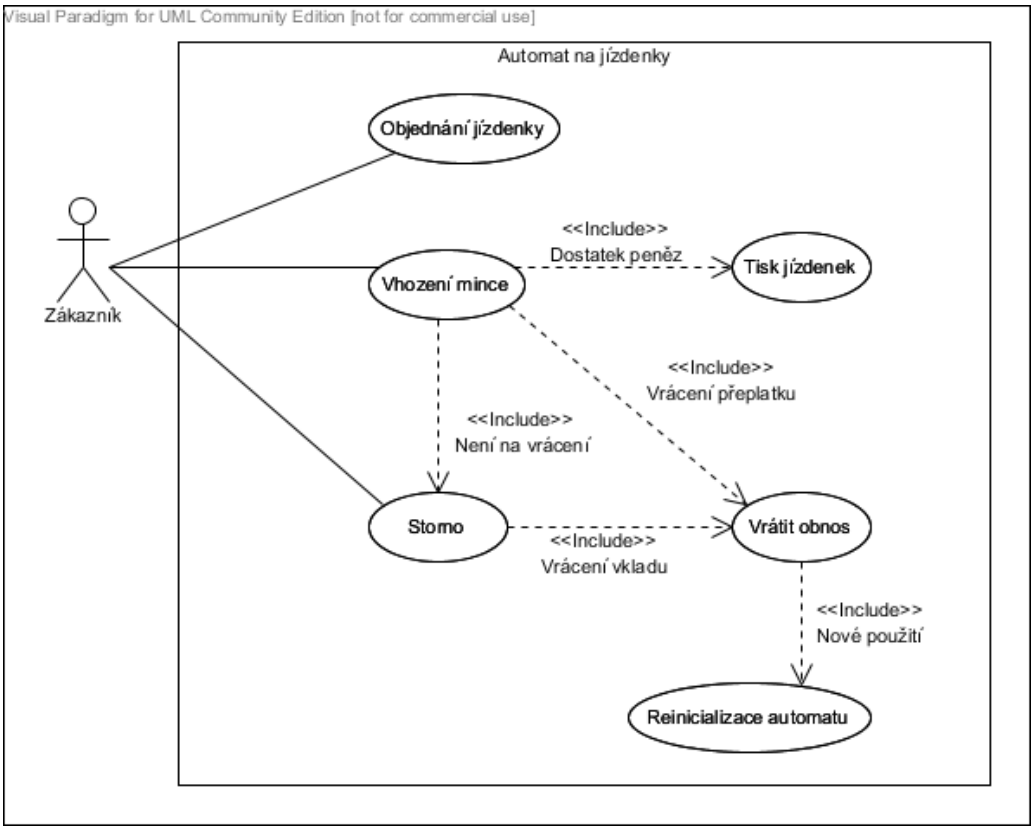
	20 minut	40 minut	60 minut	24 hodin
Základní	12 Kč	14 Kč	16 Kč	40 Kč
Zlevněné	6 Kč	7 Kč	8 Kč	20 Kč

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 o těchto činnostech:
  - Stornování objednávky.
  - Automat nemá na vrácení.
- Po řádném dokončení objednávky se zobrazí informace o tisku jízdenek s poděkováním za použití automatu.

6.1.2 Analýza

Diagram případů užití (Use Cases)



Obr. 1: 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.

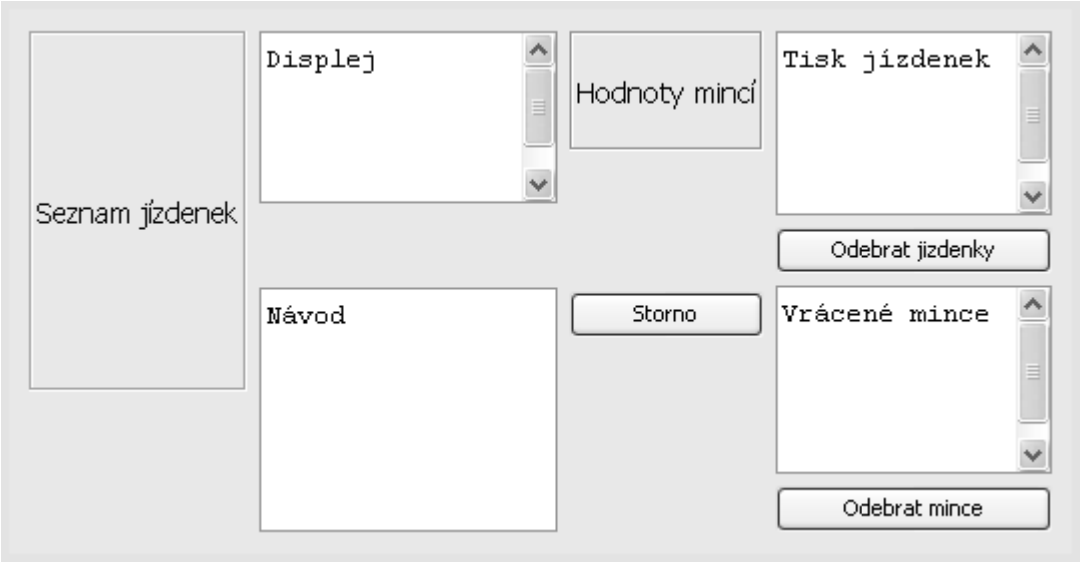
Vrátit obnos

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.

Grafický návrh aplikace



Obr. 2: Automat – Grafický návrh

Analýza tříd

Třídy

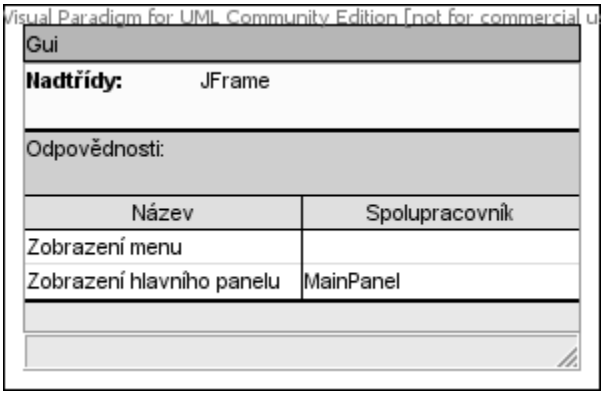
- Main – zajistí zobrazení grafického uživatelského rozhraní.
- Gui – grafické uživatelské rozhraní, obsahuje hlavní panel s komponentami.
- MainPanel – hlavní panel s komponentami, který je součástí Gui a zajistí interakci uživatele s automatem.
- Automat – představuje model reálného automatu, který 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.
- Observable – systémová třída z balíčku java.util pro aktualizaci grafického rozhraní

Interfaces

- Observer – systémový interface z balíčku java.util pro aktualizaci grafického rozhraní, implementováno v hlavním panelu s komponentami

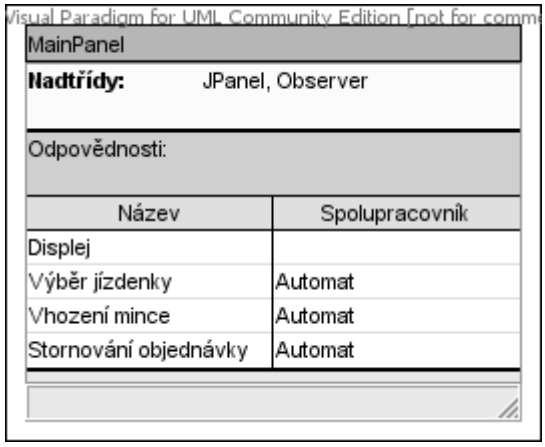
6.1.3 Objektový návrh

CRC karty



Gui	
Nadtřídy:	JFrame
Odpovědnosti:	
Název	Spolupracovník
Zobrazení menu	
Zobrazení hlavního panelu	MainPanel

Obr. 3: Automat – CRC karta – Gui

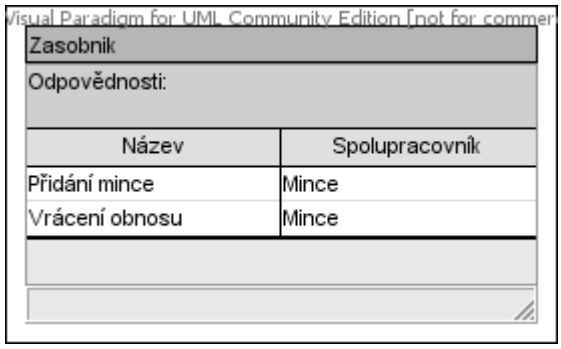


MainPanel	
Nadtřídy:	JPanel, Observer
Odpovědnosti:	
Název	Spolupracovník
Displej	
Výběr jízdenky	Automat
Vhození mince	Automat
Stornování objednávky	Automat

Obr. 4: Automat – CRC karta – MainPanel

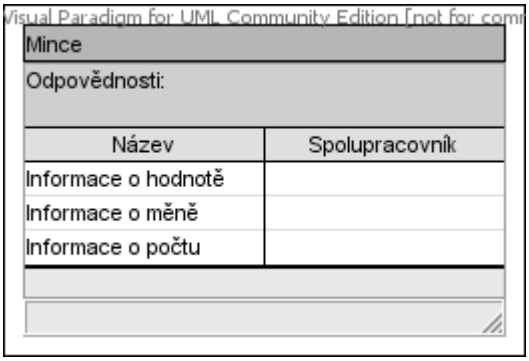
Automat	
Nadtřídy:	Observable
Odpovědnosti:	
Název	Spolupracovník
Výběr jízdenky	Jizdenka
Vhození mince	Zasobnik, Mince
Tisk jízdenek	Jizdenka
Stornování objednávky	
Vrácení obnosu	Zasobnik
Reinicializace	

Obr. 5: Automat – CRC karta – Automat



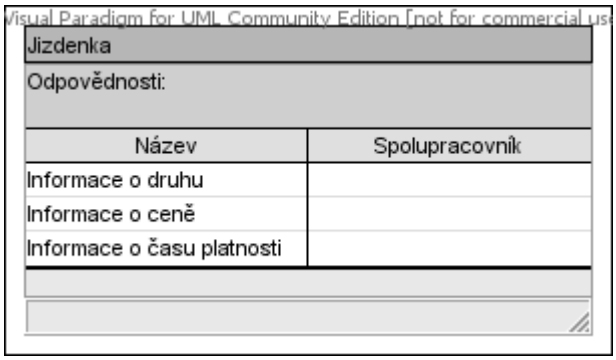
Zasobnik	
Odpovědnosti:	
Název	Spolupracovník
Přidání mince	Mince
Vrácení obnosu	Mince

Obr. 6: Automat – CRC karta – Zasobnik



Mince	
Odpovědnosti:	
Název	Spolupracovník
Informace o hodnotě	
Informace o měně	
Informace o počtu	

Obr. 7: Automat – CRC karta – Mince

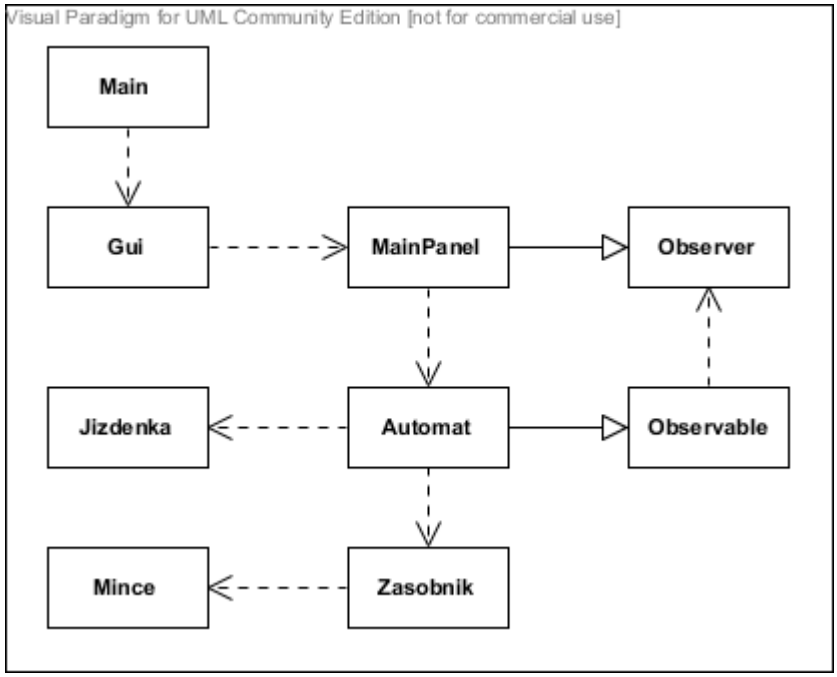


Jizdenka	
Odpovědnosti:	
Název	Spolupracovník
Informace o druhu	
Informace o ceně	
Informace o času platnosti	

Obr. 8: Automat – CRC karta – Jizdenka

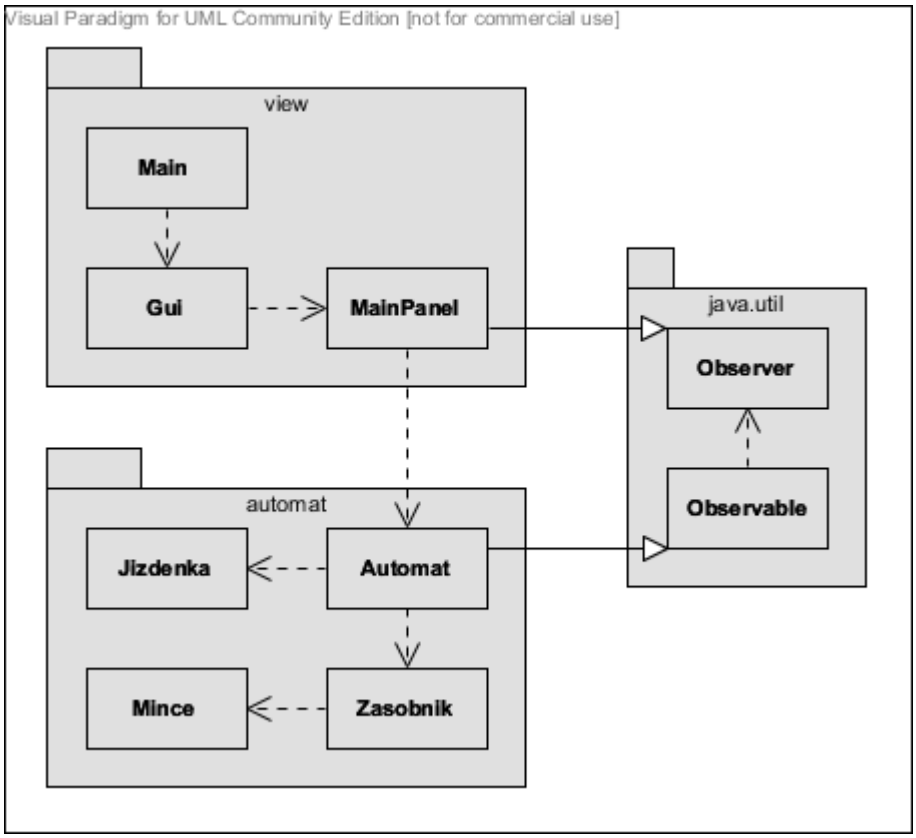
Karta Main je uvedena v příloze spolu se všemi ostatními.

Diagram tříd



Obr. 9: Automat – Diagram tříd

Diagram balíčků



Obr. 10: Automat – Diagram balíčků

6.1.4 Detailní popis tříd

Balíček: automat

Třída: Automat

Je podtřídou třídy Observable pro aktualizaci grafického rozhraní.

- Konstanty
  - + int UPDATE_DISPLAY: určuje aktualizaci displeje
  - + int UPDATE_TISK: určuje aktualizaci pro tisk jízdenek
  - + int UPDATE_MINCE: určuje aktualizaci pro vrácení mincí
- Instanční proměnné
  - – ArrayList<Jizdenka> objednavka: objednané jízdenky
  - – int vklad: celkem vhozený obnos
  - – Zasobnik zasobnik: zásobník s mincemi, které jsou v automatu
  - – Jizdenka[] jizdenky: seznam typů jízdenek, které je možné objednat
- Metody
  - + void objednatJizdenku(Jizdenka jizdenka)
    - o Přidá jízdenku do objednávky a o akci informuje GUI (update displej).
  - + void vhoditMinci(int hodnota)
    - o Pokud je objednána alespoň 1 jízdenka, přidá minci do zásobníku.
    - o Pokud bylo vhozeno dostatečné množství peněz a automat má navracení, provede se tisk jízdenek.
    - o Pokud automat nemá na vrácení, dojde ke stornování objednávky.
    - o Ve všech případech je GUI (update displej) informováno.
  - + void storno()
    - o Stornuje objednávku, případně vrátí vhozené mince.
  - – void tiskJizdenek()
    - o Vytiskne pomocí GUI (update tisk) všechny objednané jízdenky.
  - – void vratitObnos()
    - o Zajistí vrácení zadaného obnosu pomocí zásobníku a jeho zobrazení v GUI (update mince).
  - – void reinicializace()
    - o Připraví automat k novému použití.

Třída: Zasobnik

- Konstanty
  + int[] HODNOTY: seznam povolených hodnot mincí
- Instanční proměnné
  – ArrayList<Mince> mince: počet jednotlivých typů mincí
- Metody
  + void pridatMinci(int hodnota)
    - Zkontroluje, zda hodnota mince existuje a přidá minci do zásobníku.
  + ArrayList<Mince> vratitObnos(int obnos)
    - Ověří, zda je v zásobníku dostatek mincí na vrácení, poté vrátí zadaný obnos v mincích (ve formě kolekce) od nejvyšší hodnoty.

Třída: Mince

- Konstanty
  + String MENA: stanovená měna pro všechny mince
- Instanční proměnné
  – int hodnota: hodnota mincí
  – int počet: počet mincí
- Metody
  + int getHodnota(): navrátí hodnotu mincí
  + int getPocet(): navrátí počet mincí
  + void increment(): navýší počet mincí o 1
  + void odebrat(int pocet): sníží počet mincí podle zadaného parametru

Třída: Jizdenka

- Konstanty (viz Tabulka 1 – str. 12)
  + String[] DRUHY: všechny druhy jízdenek
  + String[] PLATNOSTI: povolené platnosti jízdenek
  + int[][] CENY: dvourozměrné pole pro všechny typy jízdenek
- Instanční proměnné
  – String druh: druh jízdenky – základní / zlevněná
  – int cena: cena jízdenky
  – String platnost: platnost jízdenky
- Metody
  + String toTisk(): navrátí znakovou reprezentaci jízdenky při tisku

Balíček: view

Třída: Main

- Metody
  + static void main(String[] args): zobrazí grafické uživatelské rozhraní (Gui)

Třída Gui

- Konstanty
  – Dimension SIZE: přednastavená velikost okna aplikace
- Instanční proměnné
  – MainPanel mainPanel: hlavní panel aplikace
- Metody
  – void initComponents(): inicializuje komponenty aplikace

Třída: MainPanel

Je podtřídou třídy JPanel a implementuje interface Observer pro přijímání aktualizací z automatu.

- Instanční proměnné
  – Automat automat: automat na jízdenky
- Metody
  – initComponents(): inicializuje komponenty
  + update(Observable o, Object arg): přijímá aktualizace z automatu

6.1.5 Implementace

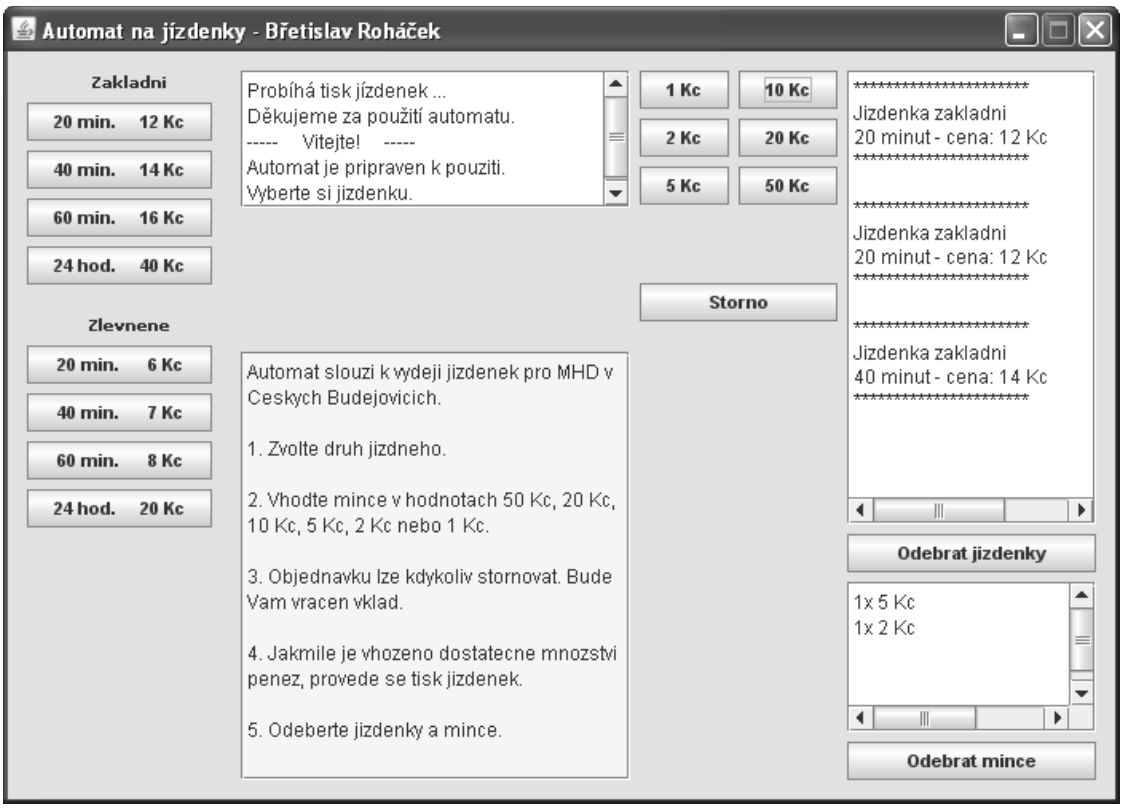
Problémy a změny při implementaci

Třída Jizdenka: Vyžadovala přepsání metody toString() pro vyjádření znakové reprezentace jízdenky při jejím objednání. Jednotlivé konstanty by bylo možné nahradit výčtovým typem nebo je načíst z externího souboru.

Třída Zasobnik: Nastal problém při kontrole, zda má automat na vrácení. Kontrola by vyžadovala stejný (úplně netriviální) proces, jako při reálném vrácení přeplatku. Oba procesy by počítaly s vracením mincí od nejvyšší hodnoty s tím rozdílem, že při kontrole by nedocházelo k odebírání mincí ze zásobníku a nesnižoval by se průběžně vrácený přeplatek. S ohledem na zamezení duplicity kódu, byl tento problém vyřešen pomocí Exception. Proces skutečného vrácení přeplatku je vložen do bloku try-catch a v případě, že není vrácen celý přeplatek, je vyhozena výjimka a tím pádem budou veškeré operace provedené při vrácení přeplatku stornovány.

Třída Automat: Bylo zapotřebí doplnit několik privátních metod. Metoda kolikObjednano() pro výpočet sumy k zaplacení a 3 metody pro usnadnění aktualizace grafického rozhraní za asistence třídy ActionEvent – jako zdroj uvádějící automat, identifikátor určující, která položka je aktualizována a řetězec, jako zpráva, která má být zobrazena.

Výsledný screenshot



Obr. 11: Automat – Výsledný screenshot

6.1.6 Testování

Test grafického rozhraní

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

- Vhození mincí 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 i bez něj. Jednotlivé části testování byly prováděny opakovaně a v kombinovaném pořadí.

Test jednotlivých tříd

Třída: Zasobnik

Metoda: void pridatMinci(int hodnota)

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

Metoda: ArrayList<Mince> vratitObnos(int obnos)

- 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.

6.1.7 Udržování

Pro typy jízdenek a mincí by bylo možné založit výčtové typy nebo ještě lépe tabulky, které by se načítaly z externích souborů, což by poskytlo mnohem snadnější editaci. Tato úprava je důležitá nejen kvůli případné změně jízdného či hodnot vhazovaných mincí, ale také by počet mincí v zásobníku zůstal aktuální i po ukončení aplikace. Grafická podoba jízdenek může být rovněž načítána z externího zdroje v textové podobě, případně nahrazena obrázkem.

Přidáním třídy Objednavka, by se ulehčilo třídě Automat – objednávání jízdenek, režie ceny objednávky formou privátní proměnné, atd. Program může být rozšířen o správu zásobníku. 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.

6.2 Geometrický kalkulátor

6.2.1 Stanovení cílů

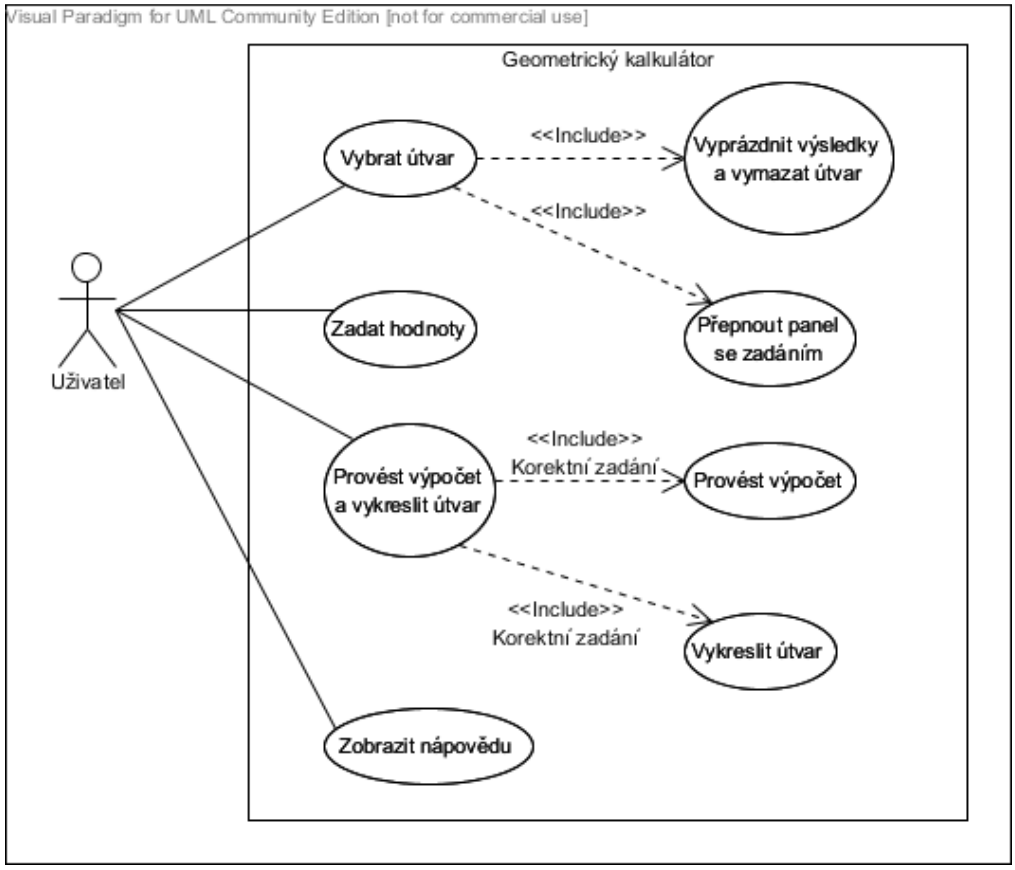
Aplikace bude umět vypočítat základní vzorce z geometrie, konkrétně obvod a obsah, pro zvolené útvary. Na výběr bude z těchto útvarů a budou definovány dle následujících parametrů:

- Čtverec: strana *a*
- Obdélník: strana *a* a *b*
- Kruh: poloměr *r*
- Trojúhelník: strana *a*, *b* a *c*

Útvary budou vykreslovány podle zadaných údajů do postranního panelu. Výpočet hodnot a vykreslení útvaru bude možné jen v případě, že zadání bude řešitelné – tj. zadané hodnoty budou kladné a u trojúhelníku bude platit pravidlo, že součet délek libovolných dvou stran bude větší než délka třetí strany. Všechny hodnoty bude možné zadávat v desetinných číslech.

6.2.2 Analýza

Diagram případů užití (Use Cases)



Obr. 12: Geometrický kalkulátor – Diagram případů užití

Vybrat útvar

Při zvolení útvaru ze seznamu dojde k přepnutí panelu se zadáním, vyprázdnění panelu s výsledky a vymazání předchozího útvaru.

Přepnout panel se zadáním

Zobrazí panel pro zadání parametrů pro odpovídající útvar (čtverec, obdélník, kruh nebo trojúhelník) dle zvolené komponenty.

Vyprázdnit výsledky a vymazat útvar

Vyprázdní výsledné hodnoty obvodu a obsahu předchozího útvaru – tj. nastaví hodnoty na 0. A vymaže vykreslený útvar.

Zadat hodnoty

Hodnoty, parametry útvaru, bude možné zadat pomocí vstupních polí. Hodnoty musí být kladné a mohou obsahovat desetinnou čárku.

Provést výpočet a vykreslit útvar

Nejprve se ověří, zda jsou zadané parametry korektní. Pokud ne, zobrazí se dialogové okno s chybovým hlášením. V opačném případě se provedou všechny výpočty a vykreslí se zvolený útvar.

Provést výpočet

Provede výpočet obvodu a obsahu vybraného útvaru a zobrazí výsledky.

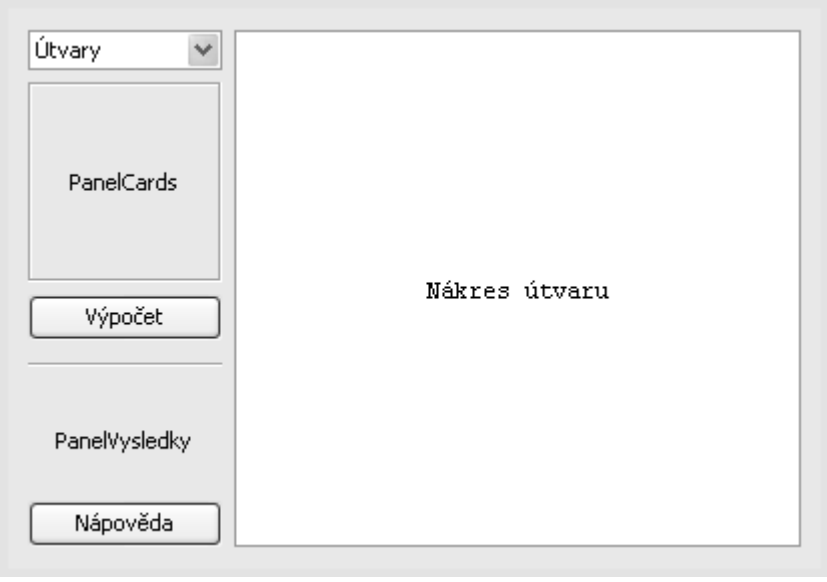
Vykreslit útvar

Vykreslí zvolený útvar dle zadaných parametrů doprostřed postranního panelu.

Zobrazit nápovědu

Zobrazí dialogové okno s nápovědou.

Grafický návrh aplikace



Obr. 13: Geometrický kalkulátor – Grafický návrh

Analýza tříd

Třídy

- Main – zajistí zobrazení grafického uživatelského rozhraní.
- Gui – grafické uživatelské rozhraní, obsahuje komponentu pro výběr útvaru, panel pro zadání parametrů, tlačítko pro provedení výpočtu, panel s výsledky a panel s vykresleným útvarem.
- PanelNakres – vykreslí zvolený útvar uprostřed panelu ve velikosti dle zadaných parametrů
- PanelCards – obsahuje jednotlivé PanelUtvary: PanelCtverec, PanelObdelnik, PanelKruh a PanelTrojuhelnik, které se budou zobrazovat (přepínat) podle zvoleného útvaru.
- PanelUtvar – panel určující standard pro panely zadávající parametry pro svůj útvar.
- PanelCtverec – PanelUtvar, který umožňuje zadat stranu čtverce.
- PanelObdelnik – PanelUtvar, který umožňuje zadat parametry pro obdélník.
- PanelKruh – PanelUtvar, který umožňuje zadat poloměr kruhu.
- PanelTrojuhelnik – PanelUtvar, který umožňuje zadat všechny strany trojúhelníku.
- Ctverec – implementuje rozhraní Utvar a nese informaci o délce strany *a*.
- Obdelnik – implementuje rozhraní Utvar a nese informaci o délkách stran *a* a *b*.
- Kruh – implementuje rozhraní Utvar a nese informaci o poloměru *r*.
- Trojuhelnik – implementuje rozhraní Utvar a nese informaci o délkách stran *a*, *b* a *c*.

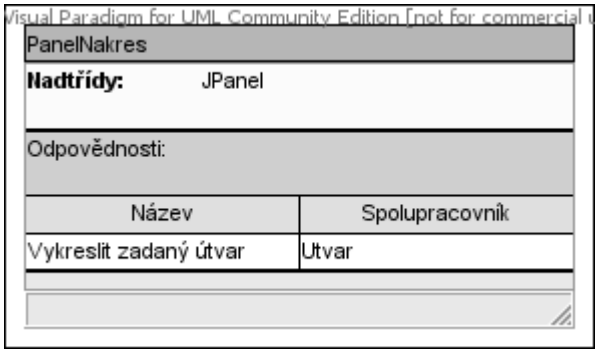
Interfaces

- Utvar – definuje metody pro výpočet obvodu a obsahu útvaru a metodu pro navrácení útvaru k vykreslení.

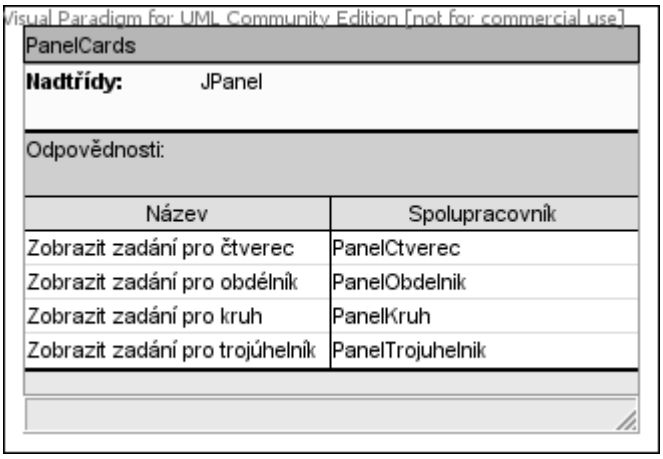
6.2.3 Objektový návrh

CRC karty

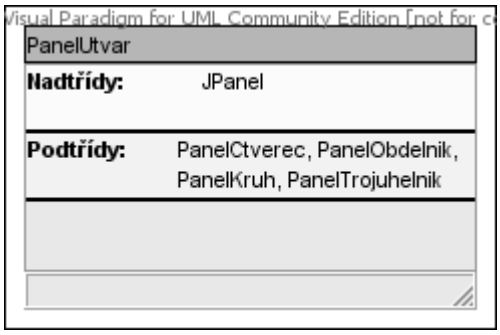
Obr. 14: Geom. Kalk. – CRC karta – Gui



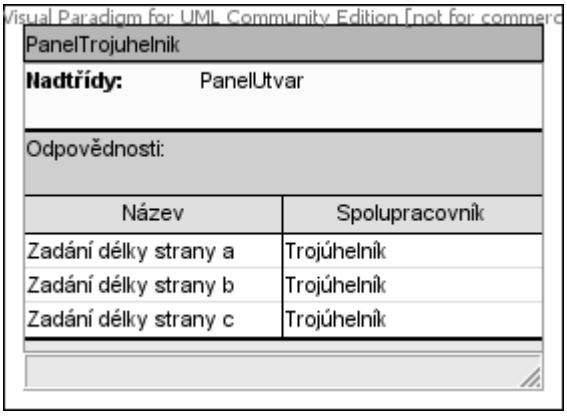
Obr. 15: Geom. Kalk. – CRC karta – PanelNakres



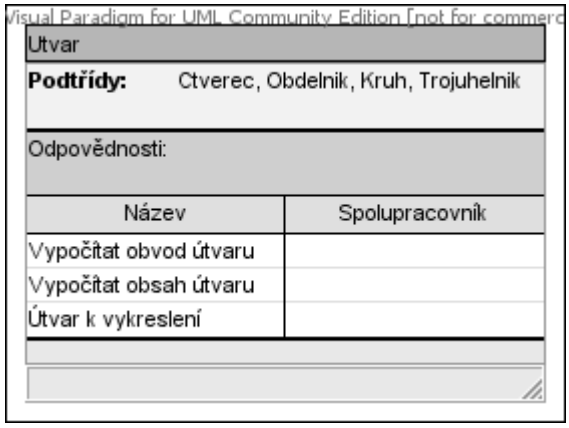
Obr. 16: Geom. kalk. – CRC karta – PanelCards



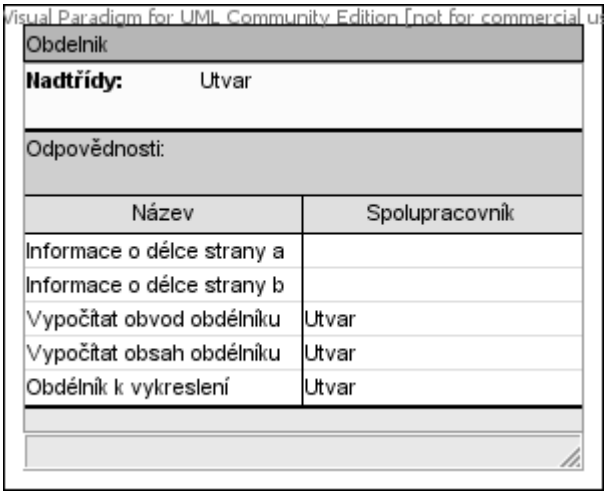
Obr. 17: Geom. kalk. – CRC karta – PanelUtvar



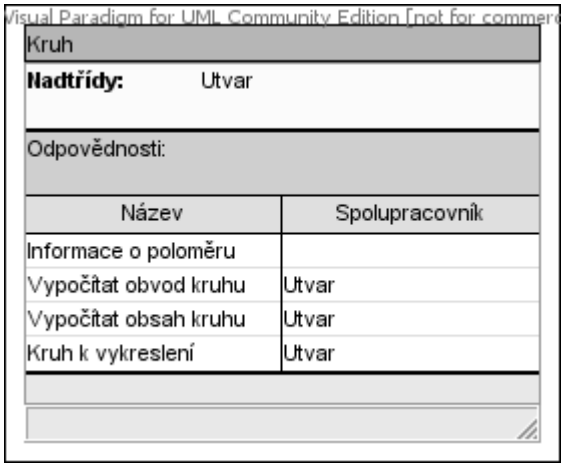
Obr. 18: Geom. kalk. – CRC karta – PanelTrojuhelnik



Obr. 19: Geom. kalk. – CRC karta – Utvar



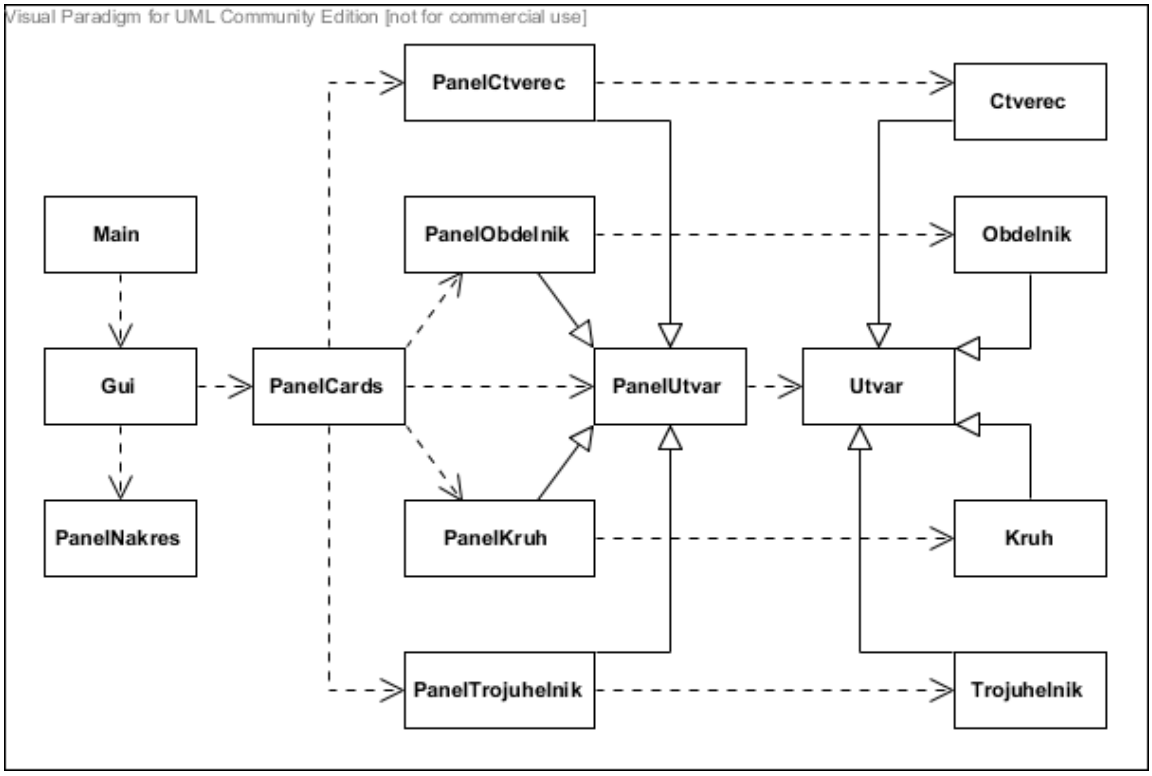
Obr. 20: Geom. kalk. – CRC karta – Obdelnik



Obr. 21: Geom. kalk. – CRC karta – Kruh

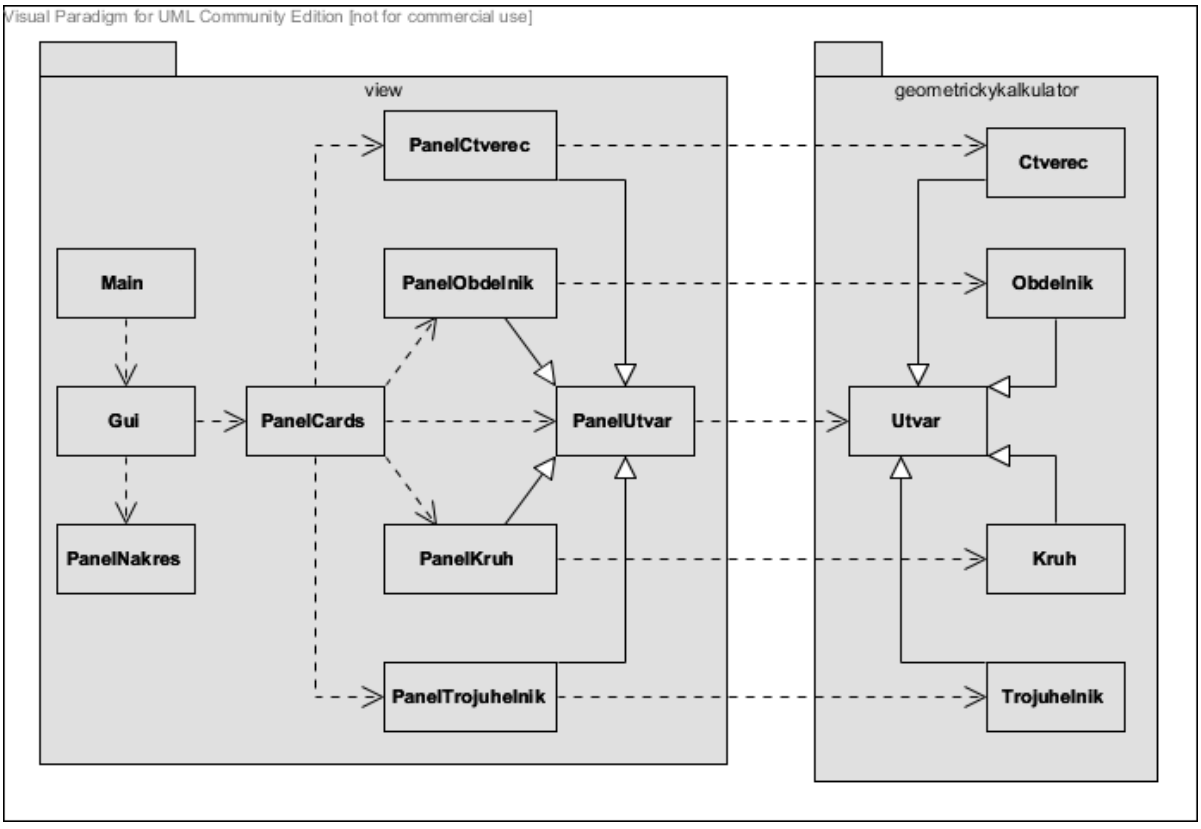
Zbylé CRC karty (PanelCtverec, PanelObdelnik, PanelKruh, Ctverec a Trojuhelnik) vypadají analogicky a jsou zahrnuty v příloze.

Diagram tříd



Obr. 22: Geometrický kalkulátor – Diagram tříd

Diagram balíčků



Obr. 23: Geometrický kalkulátor – Diagram balíčků

6.2.4 Detailní popis tříd

Balíček: geometrickykalkulator

Rozhraní: Utvar

- Metody
  + double getObvod(): navrátí obvod daného útvaru
  + double getObsah(): navrátí obsah daného útvaru
  + Shape getShape(): navrátí útvar typu Shape, který se vykreslí

Třída: Ctverec

Implementuje rozhraní Utvar – metoda getShape() navrací útvar typu Rectangle.Double.

- Instanční proměnné
  – double a: délka strany a

Třída: Obdelnik

Implementuje rozhraní Utvar – metoda getShape() navrací útvar typu Rectangle.Double.

- Instanční proměnné
  – double a: délka strany a
  – double b: délka strany b

Třída: Kruh

Implementuje rozhraní Utvar – metoda getShape() navrací útvar typu Ellipse2D.Double.

- Instanční proměnné
  – double r: poloměr kruhu

Třída: Trojuhelnik

Implementuje rozhraní Utvar – metoda getShape() navrací útvar typu Polygon.

- Instanční proměnné
  – double a: délka strany a
  – double b: délka strany b
  – double c: délka strany c

Balíček: view

Třída: Main

- Metody
  + static void main(String[] args): zobrazí grafické uživatelské rozhraní (Gui)

Třída Gui

- Konstanty
  - Dimension SIZE: přednastavená velikost okna aplikace
- Instanční proměnné
  - JPanel panelVypocty: panel obsahující panelCards a panelVysledky
  - PanelNakres panelNakres: panel vykreslující zvolený útvar
  - PanelCards panelCards: panel obsahující jednotlivé panelUtvary
  - JPanel panelVysledky: panel obsahující labely s výsledky
  - JLabel labelObvodVysledek: label pro zobrazení obvodu útvaru
  - JLabel labelObsahVysledek: label pro zobrazení obsahu útvaru
- Metody
  - void initComponents(): inicializuje komponenty aplikace

Třída: PanelNakres

- Konstanty
  - Color COLOR_BACKGROUND: barva pozadí
  - Color COLOR_SHAPE: barva výplně útvaru
- Instanční proměnné
  - Shape shape: vykreslovaný útvar
- Metody
  + paintShape(Shape shape)
    - Vykreslí zadaný útvar pomocí přepsané metody paint(Graphics g).

Třída: PanelCards

- Instanční proměnné
  - CardLayout cards: LayoutManager se všemi panelUtvary
- Metody
  + Utvar getUtvar() throws Exception
    - Navrátí útvar z aktuálně zobrazeného panelUtvaru.
    - Vyhodí výjimku, pokud útvar nelze vytvořit.

Třída: PanelUtvar

- Metody
  + abstract Utvar getUtvar() throws Exception

Třídy: PanelCtverec, PanelObdelnik, PanelKruh, PanelTrojuhelnik

Podtřídy třídy PanelUtvar, které přepisují metodu getUtvar().

6.2.5 Implementace

Problémy a změny při implementaci

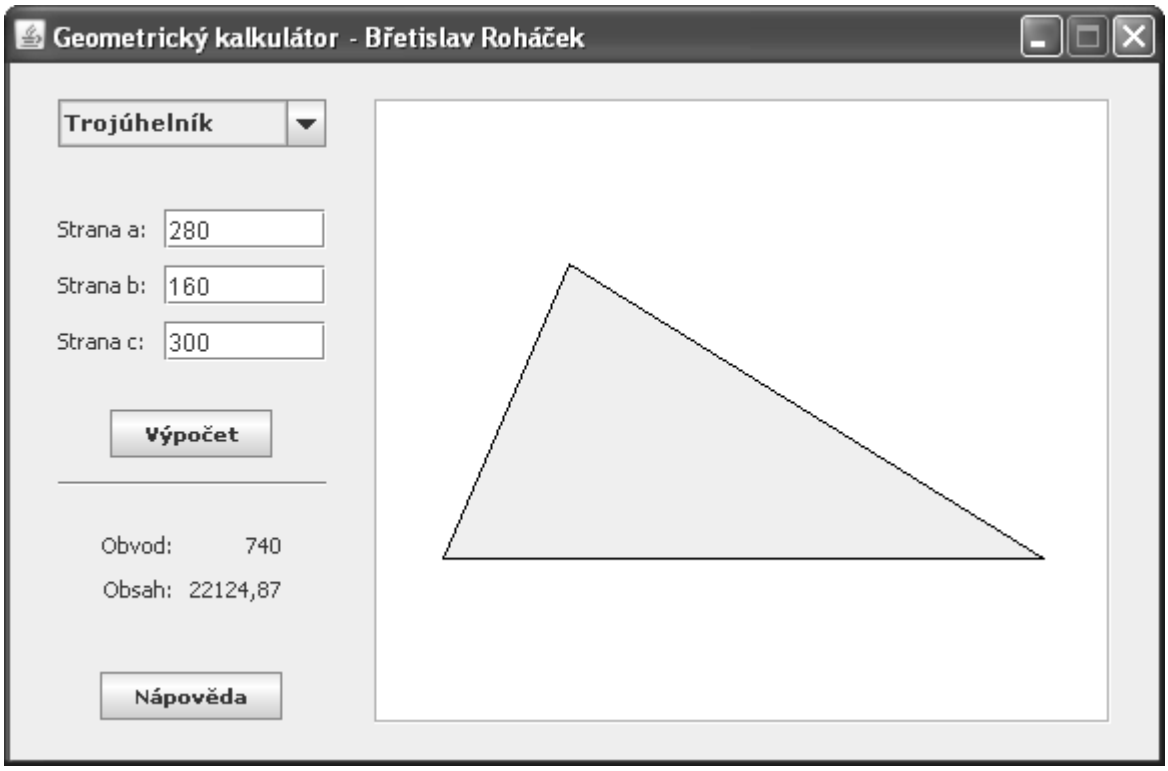
Třída PanelNakres: Při vykreslování útvarů nastal problém při jejich centrování. Objektům typu Shape totiž nelze měnit svoji pozici, a proto bylo zapotřebí útvar identifikovat pomocí konstrukce „instanceof“. V tomto případě se naštěstí rozhodovalo pouze mezi dvěma datovými typy: RectangularShape (pro čtverec, obdélník a kruh) a Polygon (pro trojúhelník).

Třída PanelCards: CardLayout nedokáže přímo určit, který panel je v danou chvíli zobrazen. Bylo tedy zapotřebí projít cyklem všechny panely obsažené v PanelCards a identifikovat zobrazený panel pomocí metody isVisible().

Třída Trojuhelnik: Objekty třídy Polygon (mnohoúhelník) nelze vytvářet na základě délek stran, ale na základě souřadnic jednotlivých bodů. Proto bylo nutné je určit. Jelikož jsem se rozhodl zvolit stranu *c* rovnoběžně s osou *x* a vykreslit trojúhelník v pořadí bodů *ABC*, jak to obvykle bývá zvykem, nebyl nakonec natolik velký problém určit souřadnice. Výšku bylo možné spočítat pomocí vzorce pro obsah trojúhelníku (a obsah pomocí Heronova vzorce), která představovala souřadnici *y* pro body *A* a *B* a souřadnici *x* pro bod C pomocí Pythagorovy věty… ostatní souřadnice byly známé.

Třída Gui: Výsledky byly s ohledem na přehlednost zaokrouhleny na 2 desetinná místa. K tomuto účelu bylo využito třídy DecimalFormat.

Výsledný screenshot



Obr. 24: Geometrický kalkulátor – Výsledný screenshot

6.2.6 Testování

Test grafického rozhraní

Testovány byly tyto části:

- Přepnutí panelu pro zadávání parametrů při výběru útvaru.
- Zadat parametry v kladných, záporných, nulových i desetinných číslech – samostatně, opakovaně a kombinovaně.
- Provést výpočet.
- Nezadat některý s parametrů a přesto provést výpočet.
- Ověřit vymazání výsledků a vykresleného útvaru při vybrání jiného útvaru.
- Zadat pro trojúhelník parametry, pro které by neexistoval a přesto provést výpočet.
- Zobrazit nápovědu.

Jednotlivé části testování byly prováděny v kombinovaném pořadí a opakovaně.

6.2.7 Udržování

Třídy Ctverec a Obdelnik by bylo vhodné sjednotit. Geometrické útvary by bylo možné obohatit o další mnohoúhelníky, kuželosečky, případně i jiné útvary. Analogickým postupem by mohly být počítány a vykreslovány 3D objekty – tělesa – krychle, kvádr, hranol, válec, koule, jehlan, kužel, …

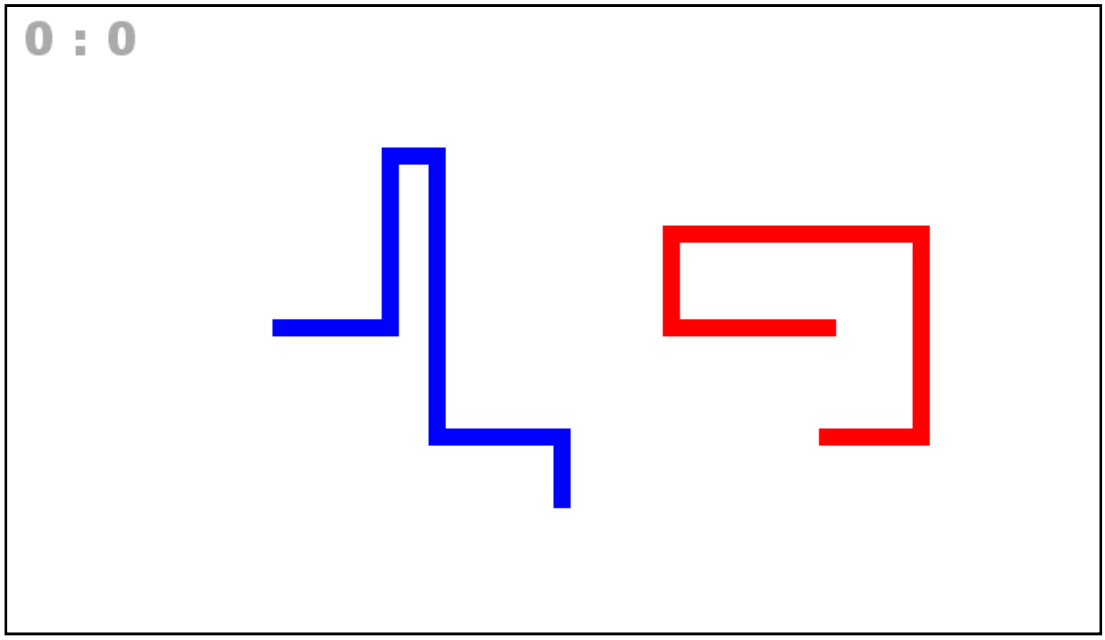
Útvarům by se dala přidělovat barva při vykreslení a jejich screenshoty všestranně využít. Zadávané parametry by mohly zahrnovat jednotky, ve kterých se výpočty provádějí, případně vytvořit převodník jednotek. Nákresy by mohly obsahovat označení bodů, stran, atd. Případně navrhnout ovládací formulář a nechat zobrazování jednotlivých prvků na uživateli.

6.3 Tron

O hře

Tron je hra realizována na čtvercové, případně obdélníkové ploše, která se skládá 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. 25). 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 náhodně či pevně dané překážky. Počáteční pozice hráčů se může měnit.

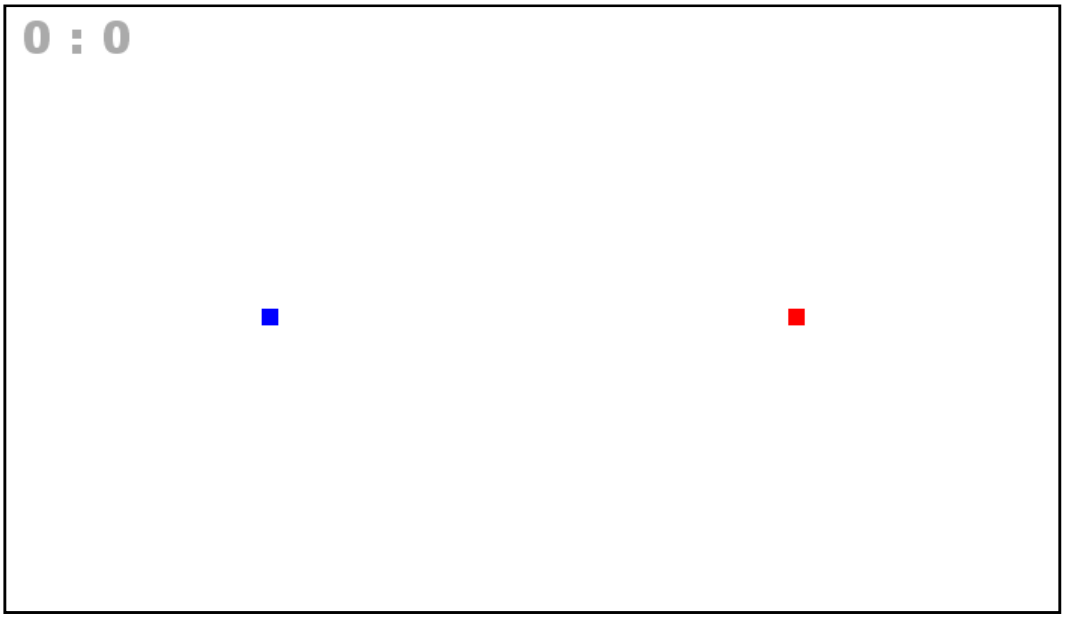


Obr. 25: Tron – Ukázka hry

6.3.1 Stanovení cílů

Tron

Cílem je vytvořit aplikaci Tron podle uvedeného popisu v předchozí kapitole. Každý hráč bude mít pevně definovanou počáteční pozici (viz Obr. 26). 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. 26: 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

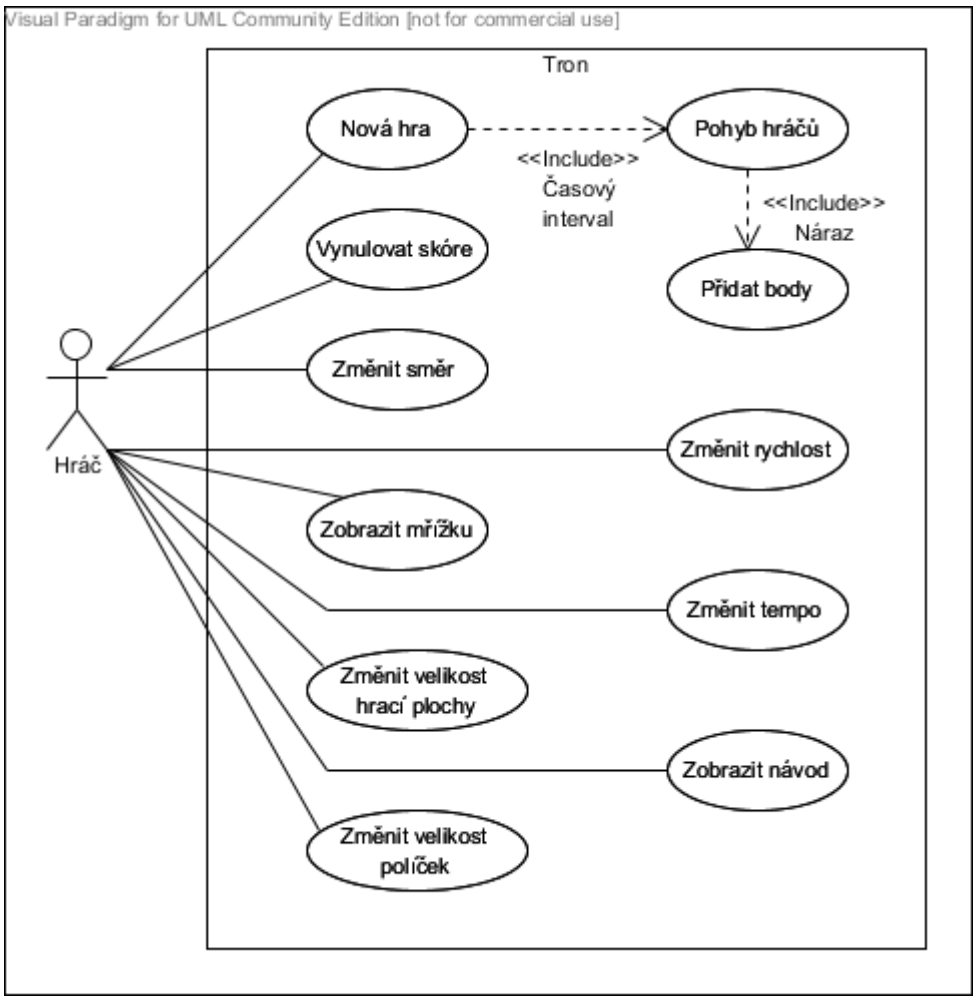
Časový interval při přesunech hráčů mezi políčky bude možné měnit. Mezi možnostmi budou 3 rychlosti: pomalu, normálně a rychle. Dále bude možné změnit tempo hry. Na výběr bude konstantní tempo a tempo, při kterém se hráči budou pohybovat rychleji (časový interval se při pohybech hráčů bude snižovat).

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.

6.3.2 Analýza

Diagram případů užití (Use Cases)



Obr. 27: 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 směru 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. Na výběr bude několik možných velikostí.

Změnit velikost políček

Změní velikost všech políček a překreslí hrací plochu. Na výběr bude ze tří velikostí políček.

Změnit rychlost

Změní nastavení časového intervalu při přesunech hráčů mezi políčky. Změna se projeví až při novém spuštění hry. Na výběr bude z těchto možností:

- Pomalu: 70 ms
- Normálně: 40 ms
- Rychle: 15 ms

Uvedená čísla udávají časový interval (pauzu – v milisekundách) mezi pohyby hráčů.

Změnit tempo

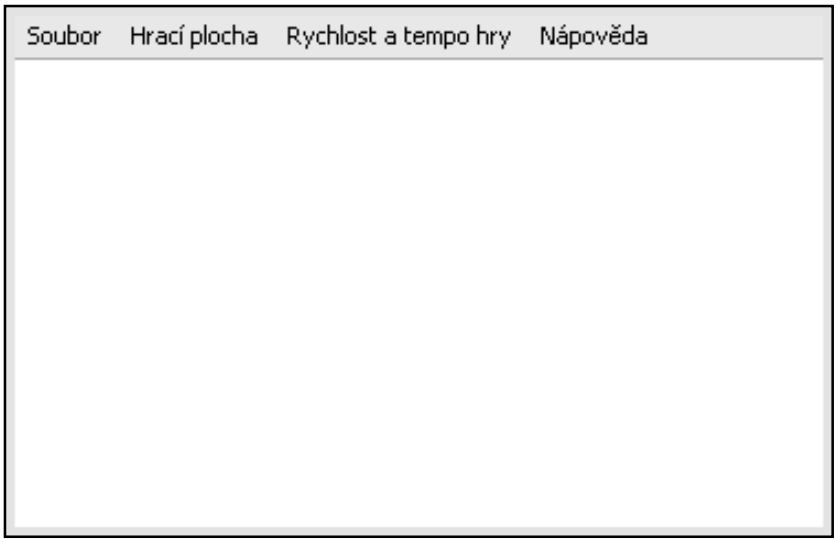
Změní tempo hry. Změna se projeví až při novém spuštění hry.

Zobrazit návod

Zobrazí nové okno s detailním popisem hry.

Analýza tříd

Grafický návrh aplikace



Obr. 28: Tron – Návrh

Třídy

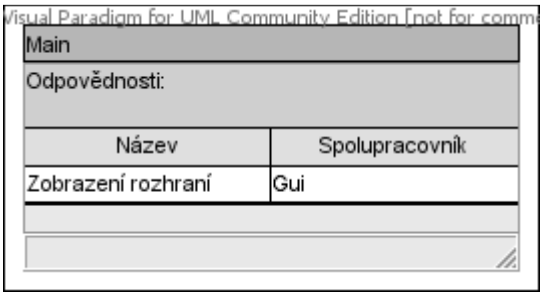
- Main – zajistí zobrazení grafického uživatelského rozhraní.
- Gui – grafické uživatelské rozhraní, obsahuje menu a panel s hrací plochou.
- HraciPlocha – panel, který reprezentuje hrací plochu a je součástí Gui. Vykresluje komponenty a předává události algoritmické části.
- Tron – zajišťuje chod hry. 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.

Interfaces

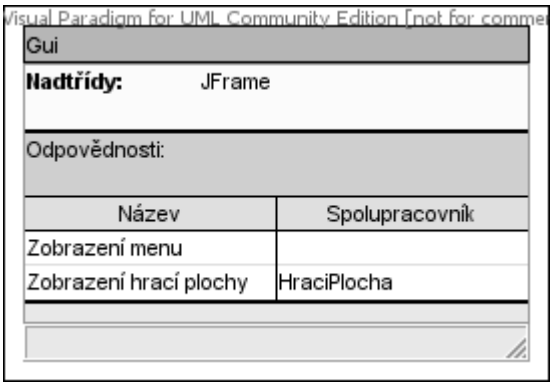
- View – aktualizuje grafické uživatelské rozhraní (Gui).
- Controller – předává události z grafického uživatelského rozhraní modelu (Tron).

6.3.3 Objektový návrh

CRC karty

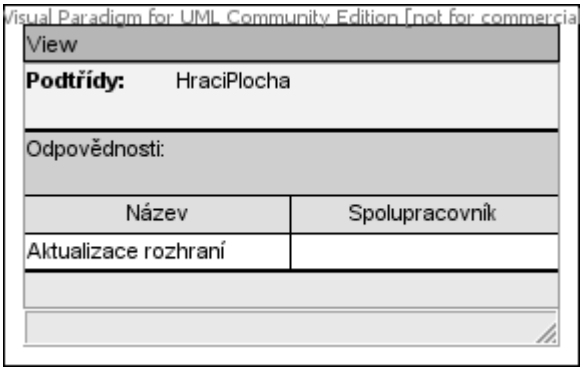


Obr. 29: Tron – CRC karta – Main

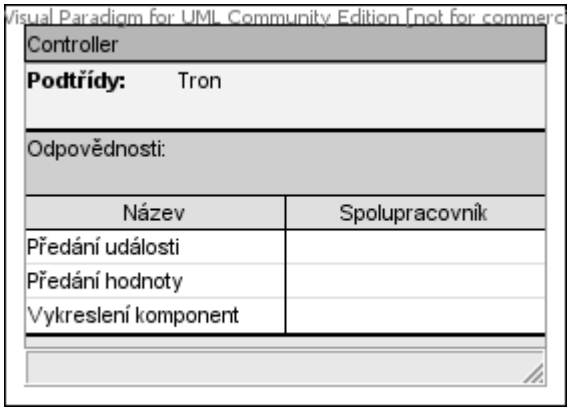


Obr. 30: Tron – CRC karta – Gui

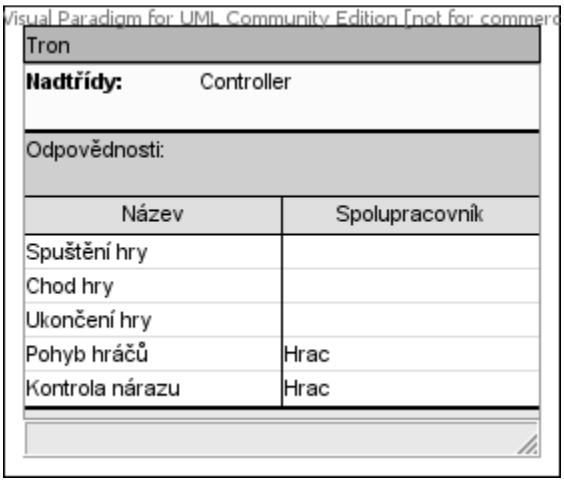
Obr. 31: Tron – CRC karta – HraciPlocha



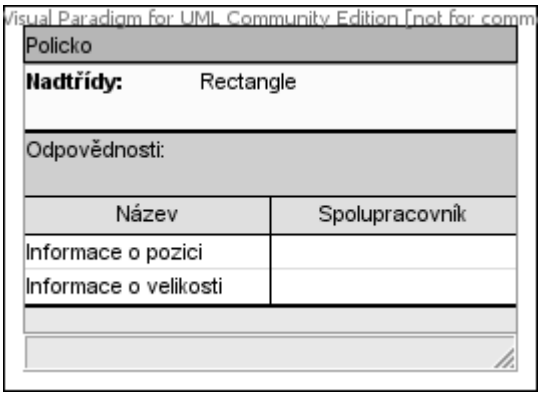
Obr. 32: Tron – CRC karta – View



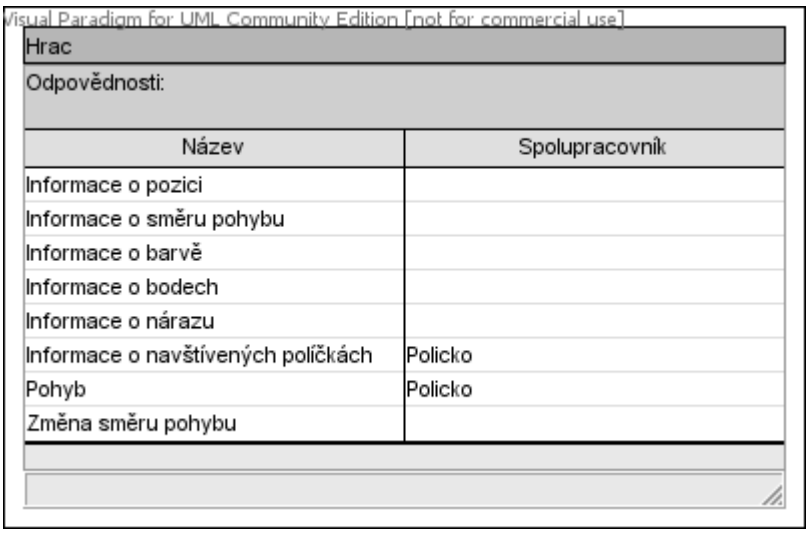
Obr. 33: Tron – CRC karta – Controller



Obr. 34: Tron – CRC karta – Tron

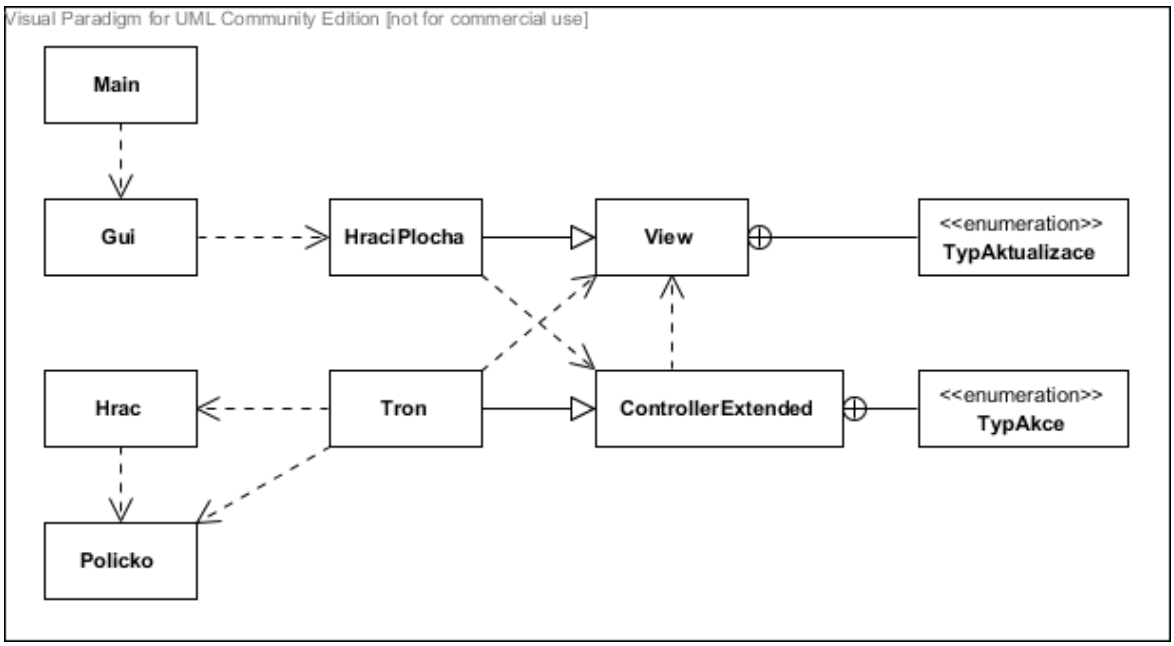


Obr. 35: Tron – CRC karta – Policko



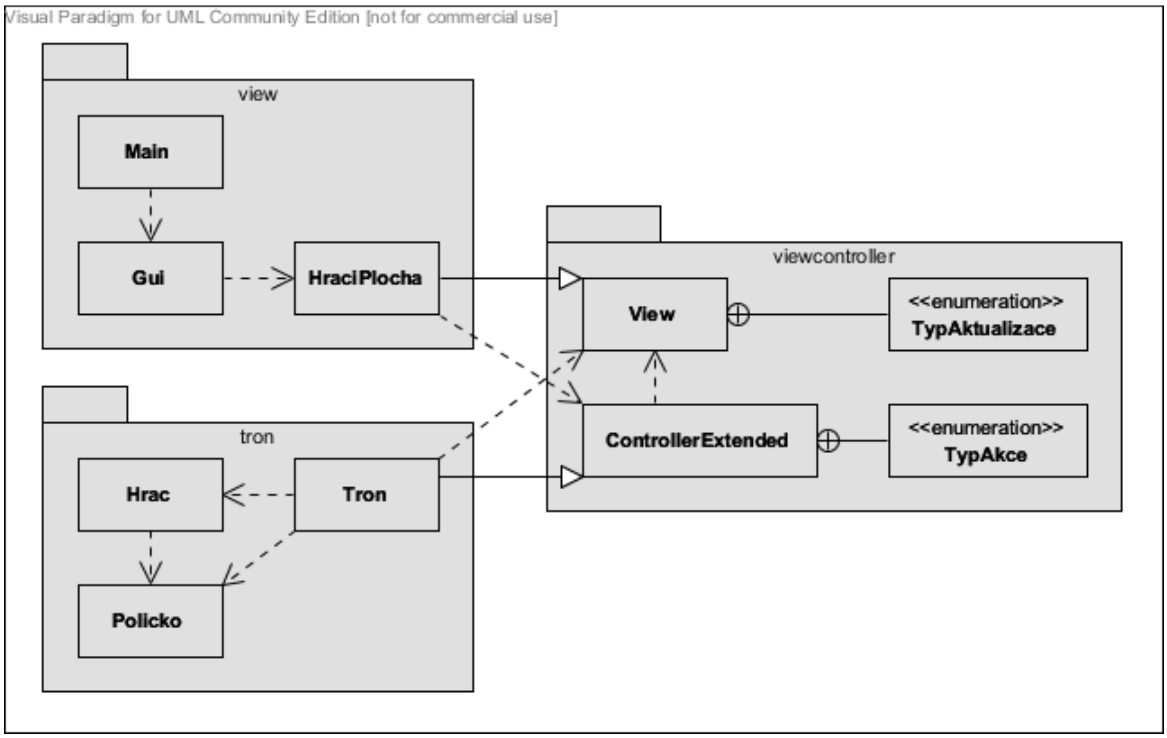
Obr. 36: Tron – CRC karta – Hrac

Diagram tříd



Obr. 37: Tron – Diagram tříd

Diagram balíčků



Obr. 38: Tron – Diagram balíčků

6.3.4 Detailní popis tříd

Balíček: tron

Třída: Tron

- Konstanty
  - int[] RYCHLOSTI: rychlosti hry
  - int[][] VELIKOSTI: velikosti hrací plochy (šířka a výška)
  - int[] POLICKA: velikosti políček
- Instanční proměnné
  - int sloupcu, radku: velikost hrací plochy
  - View view: interface pro aktualizaci grafického rozhraní
  - Thread thread: vlákno, které se stará o běžící hru
  - boolean spusteno: zda je hra spuštěna
  - int rychlost: aktuální rychlost hry
  - Hrac hrac1: první hráč
  - Hrac hrac2: druhý hráč
- Metody
  + void start(): spustí novou hru
  + void stop(): zastaví běžící hru

Třída: Hrac

- Konstanty
  + Color BARVA_1: barva prvního hráče – modrá
  + Color BARVA_2: barva druhého hráče – červená
- Instanční proměnné
  - int x, y: aktuální pozice hráče
  - ArrayList<Policko> policka: seznam navštívených políček
  - byte smerX: udává směr pohybu po ose x
  - byte smerY: udává směr pohybu po ose y
  - Color barva: barva hráče, respektive navštívených políček
  - int body: počet získaných bodů
  - boolean narazil: určuje, zda hráč do něčeho narazil
  - int[] sipky: ovládací šipky hráče
- Metody
  + void pohyb()
    - Provede pohyb hráče po x-ové a y-ové souřadnici.
    - Volá metodu pro kontrolu nárazu.
    - Přidá navštívené políčko do kolekce s políčky.

– void kontrolaNarazu()
  - Kontroluje náraz do okraje hrací plochy a do vlastních políček.

+ void zmenitSmer(int keyCode): změní směr podle zadané stisknuté klávesnice
+ void incrementBodyVyhra(): navýší počet získaných bodů v případě výhry
+ void incrementBodyRemiza(): navýší počet získaných bodů v případě remízy

Třída: Policko

- Instanční proměnné
  – int x, y: souřadnice políčka
  – int velikost: velikost políčka

Balíček: viewcontroller

Rozhraní: View

- Výčtové typy
  + TypAktualizace: SET_VALUE, ACTION: aktualizace skóre, konec hry
- Metody
  + void update(TypAktualizace typ, int index)
    - Aktualizuje grafické rozhraní.
    - Parametry typ a index určují, o kterou aktualizaci se jedná.
  + void update(TypAktualizace typ, int index, Object object)
    - Aktualizuje grafické rozhraní.
    - Parametry typ a index určují, o kterou aktualizaci se jedná
    - Parametr object obsahuje předávanou hodnotu.

Rozhraní: Controller

- Výčtové typy
  + TypAkce:
    SET_VALUE: nastavení hry – mřížka, rychlost, tempo, velikosti
    ACTION: události – start, vynulování skóre, vyčistit mřížku
- Metody
  + void setView(View view)
    - Nastaví zadaný interface View.
  + void action(TypAkce typ, int index)
    - Předá zadaný typ události.
  + void action(TypAkce typ, int index, Object object)
    - Předá zadaný typ události včetně hodnoty.
  + boolean dispatchKeyEvent(KeyEvent e)
    - Předá stisknutou klávesu modelu.

Balíček: view

Třída: Main

- Metody
  + static void main(String[] args): zobrazí grafické uživatelské rozhraní (Gui)

Třída: Gui

- Instanční proměnné
  – HraciPlocha hraciPlocha: panel vykreslující mřížku s políčky
- Metody
  + void init(): inicializuje menu a komponenty aplikace

Třída: HraciPlocha

- Konstanty
  – Color POZADI: určuje barvu pozadí
  – Color SKORE: určuje barvu skóre
- Instanční proměnné
  – Controller controller: objekt třídy tron
- Metody
  – void init(): inicializuje komponenty v panelu
  – void updateSkore(): aktualizuje skóre

6.3.5 Implementace

Problémy a změny při implementaci

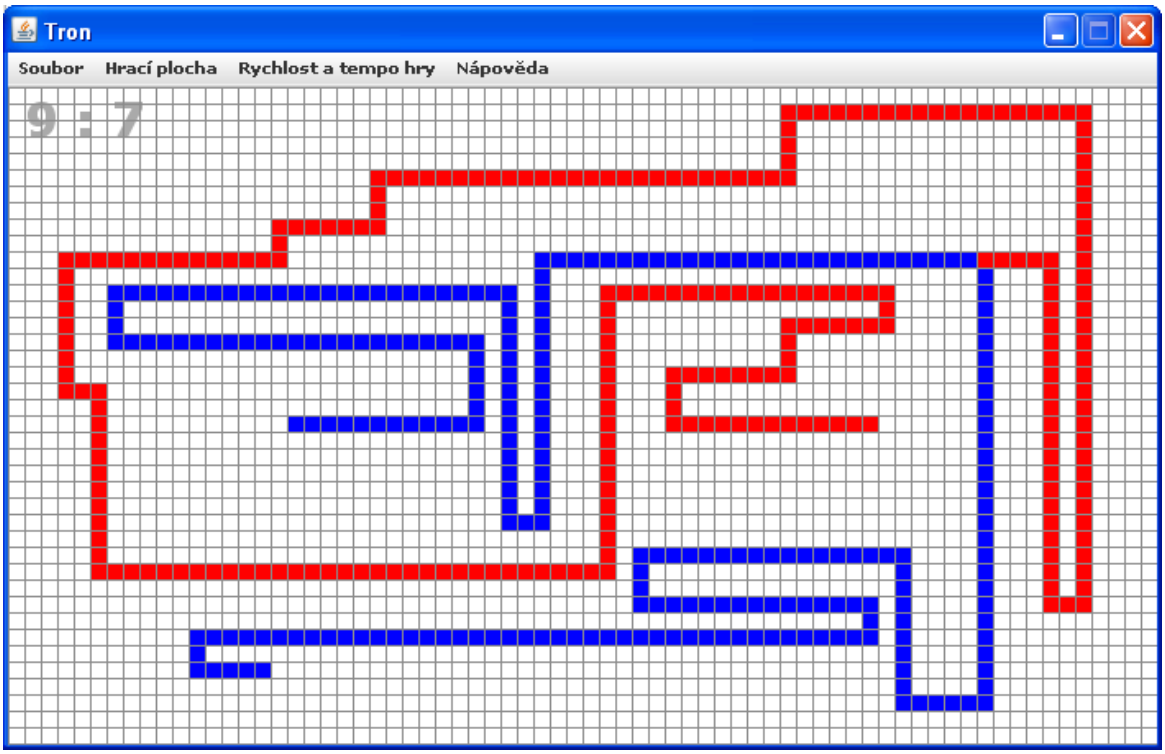
Třída Gui: Při změně velikosti hrací plochy a při změně velikosti políček, bylo zapotřebí aktualizovat velikost panelu a především celého okna aplikace. Proto byla přidána privátní metoda *updateSize()*. Tato metoda rovněž volá další přidanou metodu, a sice metodu pro vycentrování aplikace – *center()*.

Třída Hrac: Během kontroly, zda hráč nenarazil do okraje hrací plochy, případně do vlastních či soupeřovo políček, bylo zapotřebí znát velikost hrací plochy a velikost políček. Vznikla tím závislost na třídách Tron a Policko.

Třída Tron: Vzhledem k předchozímu problému ve třídě Hrac, bylo nutné zpřístupnit velikost hrací plochy ostatním třídám. Problém byl vyřešen tím, že se z instančních proměnných *radku* a *sloupcu* udělaly proměnné třídní, respektive statické. Dále přibyly privátní instanční proměnné nesoucí informace o aktuálním nastavení viditelnosti mřížky, rychlosti a tempu hry.

Třída Policko: Kvůli snadnějšímu vykreslování byla třídě přidána nadtřída Rectangle.

Výsledný screenshot



Obr. 39: Tron – Výsledek

6.3.6 Testování

Test grafického rozhraní

Vzhledem k tomu, že v aplikaci šlo především o grafickou stránku, závislou na čase, bylo testování provedeno převážně 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 (políček i celé mřížky).
- Nastavení všech rychlostí a temp hry a následné testování průběhu hry.
- Vynulování skóre.

Jednotlivé fáze testování byly prováděny opakovaně a v kombinovaném pořadí.

6.3.7 Udržování

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

Přednastavené konstanty pro políčka, hráče a hru samotnou by bylo možné nahradit výčtovým typem nebo je načíst 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.

7 Další projekty

Vzhledem k rozsahu práce nemohla zde být všem projektům věnována kompletní dokumentace, a proto jsou tyto projekty zahrnuty v příloze:

Sportka

Aplikace simuluje losovací zařízení, založené na podobném principu soutěže Sportka. Losuje se 6 náhodných čísel ze 49, která se nesmí opakovat. Vylosování jednoho čísla je označeno za tah. V každém kole se provádí 6 tahů. Sportka se může skládat z libovolného počtu kol. Před každým provedeným kolem je možné vytvořit sadu tiketů a zkusit si tipnout tažená čísla.

Program zároveň zobrazuje statistiku se třemi nejčastěji taženými čísly a seznam čísel, která nebyla tažená vůbec. Statistika rovněž zahrnuje úspěšnost tiketů v jednotlivých kolech.

Připomínající slovník

Tento slovník slouží jako učební pomůcka. Nejedná se o typický překladový slovník, ale o slovník, který připomíná zadané dvojice slov. Slovník umožňuje vytvářet kategorie, do kterých se vkládají jednotlivé dvojice. Dvojice se může skládat z anglického výrazu a jeho českého ekvivalentu nebo třeba z dvojice vět či jakékoli jiné dvojice. Slovník je při jeho prvním spuštění prázdný a neobsahuje žádnou databázi slov, takže záleží jen na uživateli, jaké dvojice bude slovník obsahovat. Dvojice jsou při jejich vkládání ukládány do externího souboru, který je možné editovat například v poznámkovém bloku.

Připomínání tkví v tom, že se v pravidelných intervalech v rohu obrazovky zobrazují náhodné dvojice slov, čímž se slova vrejou člověku do paměti i při běžném užívání počítače.

RGB Tester

Program slouží ke generování barev, které se zadávají ve formátu RGB. Umožňuje zadat přesnou intenzitu každé ze tří složek RGB (červené, zelené a modré barvy) v klasickém rozsahu 0-255.

Aplikace byla primárně určena jako pomůcka pro vývoj webu. Generování barvy samotné není samozřejmě nikterak zajímavé. To zvládne nespočet současných programů. Nicméně hlavní myšlenka tkví v tom, že barva je generována za prvé dynamicky (například tahem myši) a za druhé je zobrazena na velké ploše (třeba na celé obrazovce). Což je nesmírná výhoda, protože barva zobrazena v malém měřítku vypadá lidským okem úplně jinak, než ve „skutečnosti“ (jako třeba na pozadí).

Caesar

Projekt umožňuje šifrování zadané zprávy podle zvoleného klíče. Jedná se o realizaci Caesarovy šifry. Program obsahuje tlačítka pro šifrování a dešifrování zprávy a 3 textová pole pro: vstupní řetězec, výstupní řetězec (není možné editovat) a zvolenou šifru.

8 Zhodnocení výsledků a závěr

Ukázalo se, že hned první fáze postupu, stanovení cílů, je základem pro dobrý vývoj aplikace. Čím konkrétnější a rozpracovanější zadání je, tím spíše zabráníme výskytu nejasných otázek typu „*Co dělat s aplikací v této situaci?*“. A pokud na tuto otázku narazíme až ve chvíli implementace, je už příliš pozdě.

V druhé fázi, kde se provádí analýza, se rozhodne, jakým směrem se aplikace bude vyvíjet, jakým způsobem pak bude rozšiřitelná a do jakých logických celků bude rozdělena. Určí se, které třídy v projektu budou, co budou reprezentovat a jaké procesy v nich budou probíhat.

Kapitola objektového návrhu blíže specifikuje předchozí fázi. Definuje odpovědnosti jednotlivých tříd, jejich vazby (závislosti) mezi sebou a jejich seskupení v balíčcích.

Další fáze, nazvaná detailní popis tříd, se podrobně zabývá vnitřní strukturou tříd. Přesně definuje seznam vlastností ve všech třídách, jaké budou mít metody a jaká bude jejich přístupnost pro ostatní objekty. Ačkoli tato fáze v podstatě přesně říká, jak by výsledný kód měl vypadat, její realizace je dosti nejistá.

Protože během implementace, čímž se dostáváme do páté fáze vývoje, může nastat mnoho skutečností, které nás přinutí navržené proměnné a metody přehodnotit. Některé metody, či jiné prvky, mohou být nadbytečné a naopak některé, což se stává velice často, nám mohou chybět.

Ve chvíli, kdy aplikace funguje a zdá se, že je vše hotovo, není to tak úplně pravda. Nastává totiž fáze testování, která může odhalit řadu chyb a to nejen ve zdrojovém kódu, ale dokonce v samotném návrhu.

Poslední fáze je již spíše takové doplnění, jak bychom se o aplikaci měli postarat, aby byla co nejlépe využitelná pro případná rozšíření.

Dílčí cíle práce byly splněny. Univerzální scénář byl navržen podle současných metod, technik a nástrojů a projekty byly úspěšně vytvořeny podle něj. Výsledky práce mohou posloužit k výuce objektově orientovaného programování na Jihočeské univerzitě v Českých Budějovicích. Seznam všech projektů a jejich rozbor je samozřejmě velice obsáhlý, nicméně si vyučující mohou sami vybrat, které projekty, nebo které části projektů, do výuky zahrnou a které ne.

9 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

10 Přílohy

Součástí přílohy je složka s projekty. Každý projekt obsahuje:

- Dokumentaci – CRC karty, UML diagramy
- Zdrojové kódy
- Spustitelný .jar soubor