Jihočeská univerzita v Českých Budějovicích

Pedagogická fakulta

Katedra informatiky

Projekty pro výuku programování v jazyce Java

Bakalářská práce

Autor: **Mgr. František Přinosil**
Vedoucí bakalářské práce: **RNDr. Jaroslav Icha**

České Budějovice 2008

Anotace:

Tato práce se zabývá výukou programování v objektově orientovaném programovacím jazyce Java. Práce obsahuje okomentované zdrojové kódy na nichž jsou vysvětleny základy programování. Dále práce obsahuje zadání dílčích projektů sloužících k procvičení popisovaného problému. Řešení těchto projektů je uvedeno v příloze.

Klíčová slova:

- Objektově orientované programování
- Java
- Výuka
- Řešené projekty
- BlueJ

Abstract

This thesis deals with the tuition of programming in object oriented programming language Java. It contains source codes with comments which explain the basics of programming. Further on this thesis contains the assigments of several partial projets used for practising the described problem. The solution to this projets is given in the supplement.

Key words:

- Object oriented programming
- Java
- Tuition
- solved projets
- BlueJ

Poděkování

Na tomto místě bych rád poděkoval RNDr. Jaroslavu Ichovi, vedoucímu mé bakalářské práce, za odborné vedení, připomínky a cenné rady.

Prohlašuji, že jsem svoji bakalářskou práci vypracoval samostatně, pouze s použitím zdrojů 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.

V Českých Budějovicích dne 22.února 2008

František Přinosil

Obsah

1 ÚVOD 8

1.1 Komu je práce určena 8
1.2 Používaný software 8
1.3 Stručný popis práce 8
1.4 Uspořádání práce 9
1.5 Typografická konvence 9

2 SEZNÁMENÍ S PROGRAMOVACÍM JAZYKEM JAVA. 10

2.1 Stručná historie jazyka 10
2.2 Typy programů 10

3 PROGRAMOVÉ VYBAVENÍ 11

3.1 Instalace BlueJ 11

4 OBJEKTY - ÚVOD 13

4.1 Proměnné - atributy 15
4.2 Metody 16
4.3 Konstruktory 20
4.4 Příklad 1 - Auto(konstruktory) 23
4.5 Příklad 2 - Auto(metody) 24

5 DATOVÉ TYPY 26

5.1 Celočíselné typy 26
5.2 Reálné typy 26
5.3 Logický typ 27
5.4 Znakový typ 27
5.5 Řetězce 27
5.6 Operátor Přetypování 27
5.7 Rozšiřující konverze 27
5.8 Zužující konverze 28
5.9 Přetypování na String 28
5.10 Přetypování Stringu na základní datové typy 28
5.11 Číselné datové typy a jejich konstanty 28
5.12 Příklad 1 - Auto 29
5.13 Příklad 2 - Přetypování 29
5.14 Příklad 3 - Rozsah 30
5.15 Příklad 4 - Ztráta přesnosti 31

6 POLE ... 32
6.1 Deklarace pole ... 32
6.2 Vytvoření pole ... 32
6.3 Inicializace (plnění) pole ... 32
6.4 Přístup k prvkům pole ... 33
6.5 Délka pole ... 33
6.6 Inicializované pole ... 33
6.7 Vícerozměrná pole ... 33
6.8 Příklad 1 - Parkoviště ... 33
6.9 Příklad 2 - Parkoviště 2 ... 34
7 ŘÍDÍCÍ KONSTRUKCE ... 36
7.1 Porovnávání ... 36
7.2 Logické operátory ... 36
7.3 Příkaz if ... 37
7.4 Příkaz if - else ... 37
7.5 Příkaz switch ... 37
7.6 Příklad 1 - Banka ... 38
7.7 Příklad 2 - Známkování ... 39
7.8 Příklad 3 - Kalendář ... 40
8 CYKLY ... 42
8.1 Předpokládané znalosti ... 42
8.2 Cyklus for ... 42
8.3 Cyklus while ... 42
8.4 Cyklus do while ... 43
8.5 Příklad 1 - Cykly ... 43
8.6 Příklad 2 - Matice ... 44
8.7 Příklad 3 - Šifrování ... 45
9 KOLEKCE - KONTEJNERY ... 46
9.1 Iterátory ... 50
9.2 Příklad Škola ... 52
10 INPUT - OUTPUT ... 54
10.2 Bufferování ... 57
10.3 Příklad 1 - Čeština ... 58
10.4 Příklad 2 - šifrování ... 59
10.5 Příklad 3 - Bufferování ... 59
11 TVORBA PROJEKTU V BLUEJNETBEANS ... 61

11.1 PŘEVOD MĚN ... 61
11.2 ZALOŽENÍ PROJEKTU ... 61
11.3 VYTVOŘENÍ FORMULÁŘE ... 63

12 AUTORIZOVANÝ PŘÍSTUP ... 72

12.1 PŘÍKLAD 1 - HODINKY ... 73
12.2 PŘÍKLAD 2 – ZAMĚSTNANEC ... 73

13 DĚDIČNOST ... 75

13.1 KONSTRUKTORY A DĚDIČNOST ... 75
13.2 DĚDIČNOST - METODY ... 80
13.3 DĚDIČNOST - ABSTRAKTNÍ TŘÍDY A METODY ... 81
13.4 DĚDIČNOST - FINÁLNÍ METODY ... 82
13.5 PŘÍKLAD 1 – DĚDĚNÍ(KONSTRUKTORY) ... 82
13.6 PŘÍKLAD 2 – DĚDĚNÍ(METODY) ... 84

14 POLYMORFIZMUS ... 86

14.1 PŘÍKLAD – ŠKOLA(POLYMORFIZMUS) ... 88

15 ZÁVĚR ... 89

1 Úvod

V průběhu studia na Pedagogické fakultě mě nejvíce zaujaly předměty týkající se objektově orientovaného programování. Rozhodl jsem se proto zpracovat bakalářskou práci na téma: Projekty pro výuku programování v jazyce Java.

1.1 Komu je práce určena

Tato bakalářská práce je určena studentům prvního ročníku pedagogické fakulty jako doplňkový materiál ke studiu. A to především studentům dálkového studia. Stejně dobře ji však mohou využít i studenti denního studia jako částečnou náhradu za vynechanou lekci.

1.2 Používaný software

Pro potřeby výuky jsem zvolil vývojové prostředí BlueJ, které je vyvinuté speciálně pro výuku objektově orientovaného programování v jazyce Java. Kapitola věnující se tvorbě grafického uživatelského rozhraní je zpracována ve vývojovém studiu NetBeans s modulem BlueJ.

1.3 Stručný popis práce

Zájemce o tento programovací jazyk Java může začít pracovat s touto prací bez předešlých zkušeností a vědomostí.

V prvních kapitolách práce ukazuje založení projektu, vytvoření třídy, seznamuje s pojmem objekt. To vše je prováděno velmi přehledným způsobem využívajícím značného množství obrázků ilustrující činnost programátora.

V dalších kapitolách se zabývá základními rysy programovacího jazyka. Popisuje datové typy, cykly a podmínkové konstrukce.

Následují kapitoly týkající se využití knihovních tříd.

Závěrečné kapitoly se věnují hlubším principům objektově orientovaného programování.

1.4 Uspořádání práce

Práce je rozdělena do několika kapitol. Každá výuková kapitola začíná stručným úvodem, který informuje o znalostech nutných ke studiu dané kapitoly a o dovednostech, které čtenář studiem příslušné kapitoly získá. Dále následuje ukázkový příklad. Příklad se skládá ze zdrojového kódu, který je po částech popsán a vysvětlen. Každý ukázkový příklad si může čtenář stáhnout a vyzkoušet.

Následuje série úkolů, při kterých dochází k úpravám či k rozšíření kódu. Součástí práce je i optický disk s projekty, které slouží jako základ pro další práci. Čtenář může v rámci procvičování tyto projekty dále upravovat či rozšiřovat. Na disku jsou také umístěny vyřešené projekty, které mají sloužit jako zpětná kontrola. Projekty jsou uspořádány v adresářové struktuře, která odpovídá číselné notaci kapitol, příkladů a jednotlivých úkolů.

Na přiloženém mediu je umístěna i webová stránka, která je jakým si elektronickým obrazem práce. Jak zadávané projekty, tak správná řešení si může čtenář stáhnout ve formátu zip. Popřípadě si je může přímo prohlédnut pomocí odkazu na stránku s ukázkou kódu.

1.5 Typografická konvence

Za účelem přehlednosti a srozumitelnosti práce jsem zvolil následující grafickou úpravu klíčových slov jazyka Java, názvů proměnných, metod a tříd, výpisu zdrojového kódu.

int, this	klíčová slova jazyka Java
polomer, pocetKol	názvy proměnných, metod a tříd
`tady je` `kus kódu`	výpis ze zdrojového kódu
třída	nově zaváděný pojem

Tabulka 1: Konvence

2 Seznámení s programovacím jazykem Java.

2.1 Stručná historie jazyka

Od roku 1991 vyvíjí firma Sun Microsystems tento programovací jazyk. V roce 1993 se začala rozvíjet jako programovací jazyk sloužící k vytváření aplikací pro www a tehdy stoupla její důležitost. V roce 1995 byla Java oficiálně představena.

Od té doby prošla Java dlouhým vývojem až k současné verzi Java Tiger. Během vývoje došlo ke dvěma podstatným změnám. K první došlo koncem roku 1998, kdy se autoři rozhodli zásadním způsobem přeorganizovat hierarchii tříd. Tato verze byla prezentována jako Java 2.

2.2 Typy programů

Java se používá k vytváření dvou základních typů programů a to aplikací a apletů. Aplikace si můžeme představit jako standardní programy, kdežto aplety jako programy, které se používají na www.

viz [3]

3 Programové vybavení

Pro úspěšné studium této bakalářské práce budete na Vašem počítači potřebovat následující softwarové vybavení:

- Vývojovou sadu JDK 6.0
- Vývojové prostředí BlueJ
- Vývojové prostředí NetBeans s modulem BlueJ

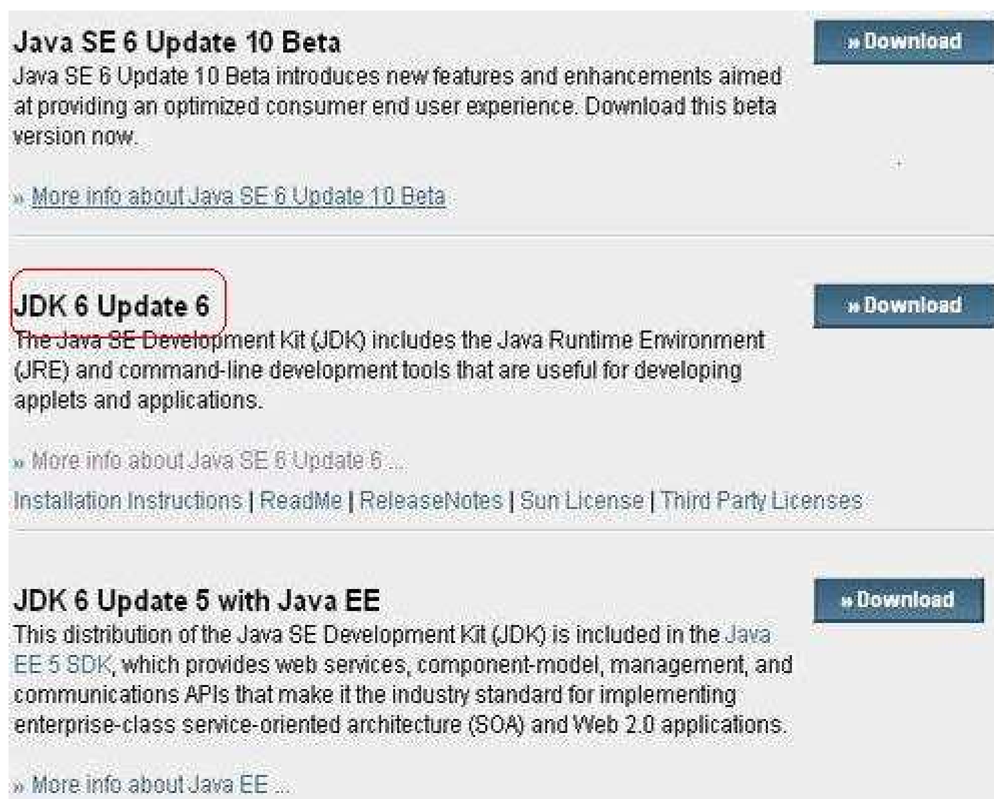
Vývojovou sadu JDK 6.0 si můžete bezplatně stáhnout na http://Java.sun.com/Javase/downloads/

Vývojové prostředí BlueJ verze 2.0 nebo vyšší si můžete stáhnout na stránkách http://www.bluej.org/download/download.html.

Vývojové prostředí NetBeans s modulem BlueJ si můžete stáhnout ze stránek http://edu.netbeans.org/bluej/.

3.1 Instalace BlueJ

Nejprve stáhněte vývojovou sadu JDK 6.0 a nainstalujte ji.



Obrázek 1: JDK

Poté stáhněte příslušnou verzi programu BlueJ vzhledem k vašemu operačnímu systému a program nainstalujte. Nastavení češtiny provedete následovně. Najděte na Vašem disku soubor bluej.defs, který bývá standardně umístěn v adresáři C:\BlueJ\lib. Záleží ovšem na adresáři, kam jste prostředí BlueJ nainstalovali. Otevřete jej v textovém editoru, pomocí dvojitého křížku zakomentujte angličtinu a okomentujte češtinu. Změny uložte.

4 Objekty - úvod

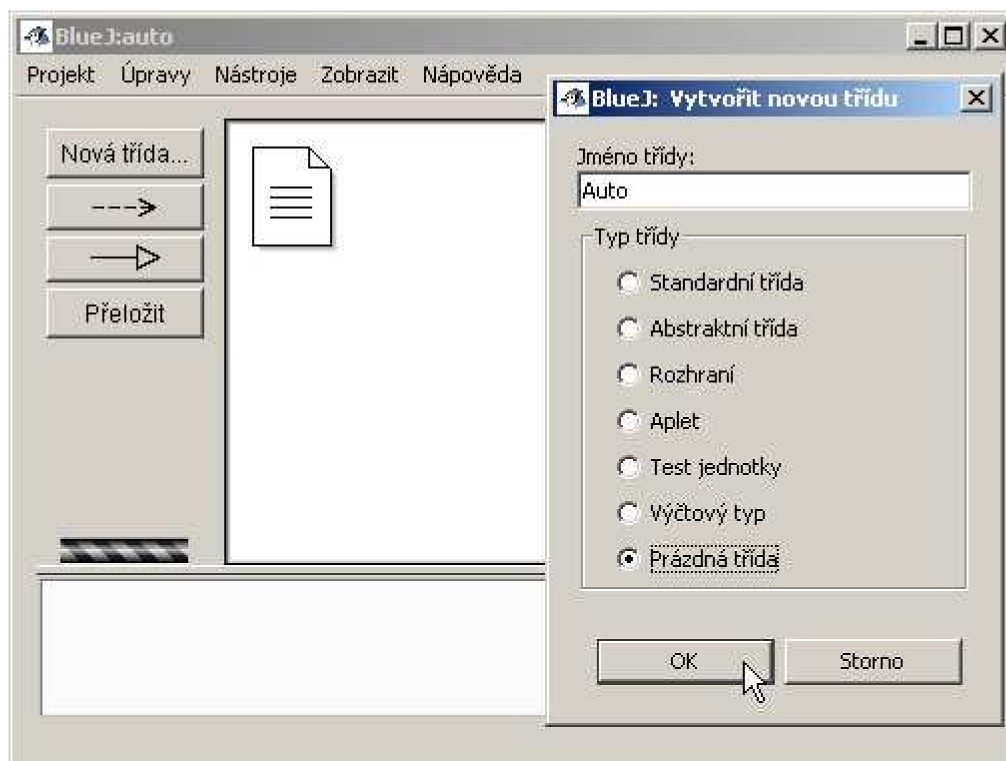
V této kapitole se seznámíme se základními pojmy objektově orientovaného programování jako jsou: třída, atribut, metoda a konstruktor. Naučíte se vytvořit vlastní třídu a pracovat s ní v prostředí BlueJ.

Základem programu vytvořeného v jazyce Java je třída(***class***). Tato třída se skládá z:

- Proměnných (***atributy***)
- Podprogramů (***metody***)

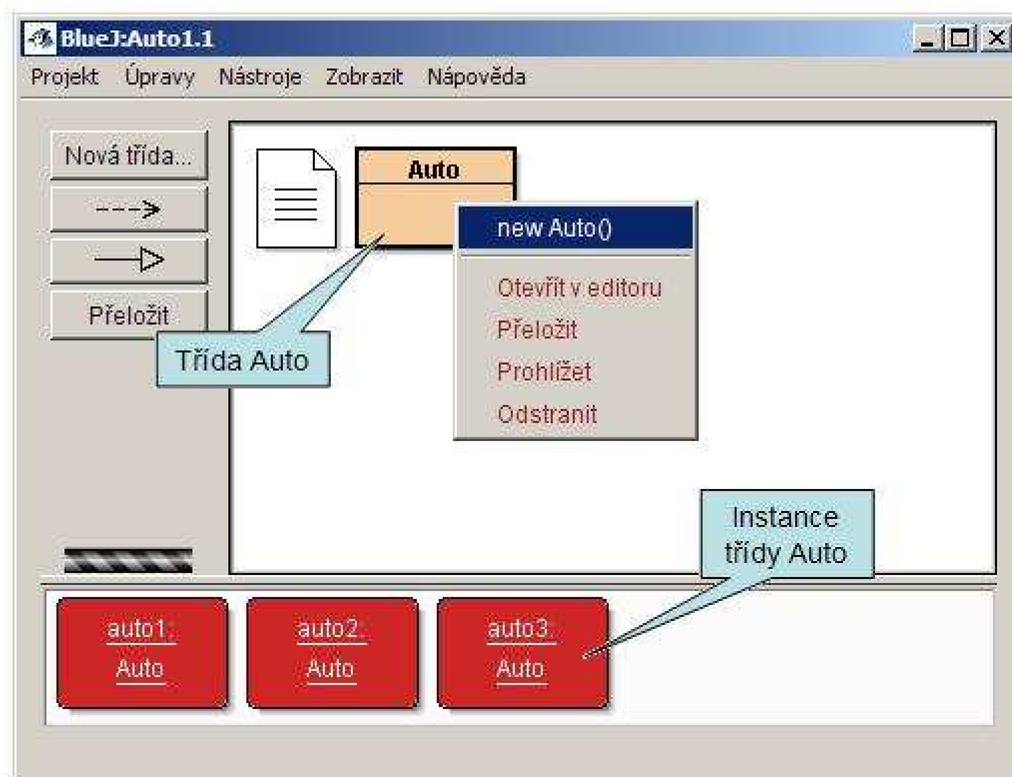
Třídu si můžeme představit jako šablonu, podle které vytváříme konkrétní objekty. Obdobně jako když podle formy na cukroví (třída), vytváříme konkrétní výrobky (instance třídy).

Založení nové třídy v prostředí BlueJ provedeme kliknutím pravého tlačítka myši na bílou pracovní plochu a výběrem možnosti nová třída. Zobrazí se formulář, ve kterém vyplníme název třídy a vybereme její typ. Poté klikneme na tlačítko ok.



Obrázek 2: Nová třída

Chceme-li vytvořit instanci třídy **Auto**, můžeme tak učinit kliknutím pravého tlačítka myši na objekt představující třídu Auto a následnou volbou **`new Auto()`**. Zobrazí se dialog tázající se na název instance. Můžeme ponechat předdefinovanou hodnotu. Po potvrzení volby dojde k vytvoření instance, kterou představuje červený obdélník v dolní části okna BlueJ. Tímto způsobem můžeme vytvořit i více instancí.



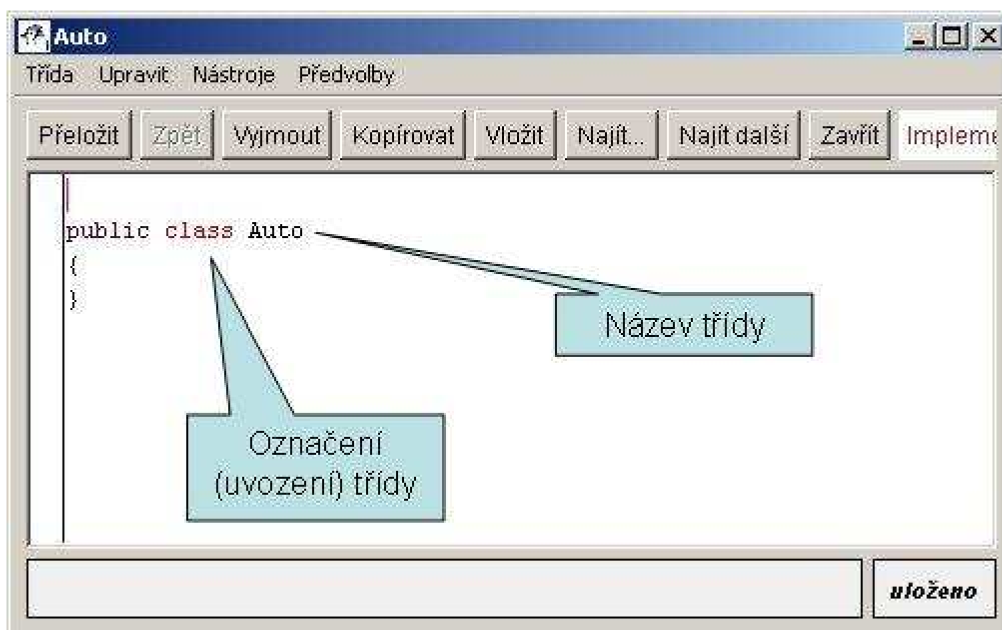
Obrázek 3: Nová instance

Poznámka:

Pokud bychom chtěli například vytvořit instanci třídy **Auto** v kódu jiné třídy, použijeme klíčové slovíčko ***new***. Kód by mohl vypadat následovně.

```
    Auto a = new Auto();
```

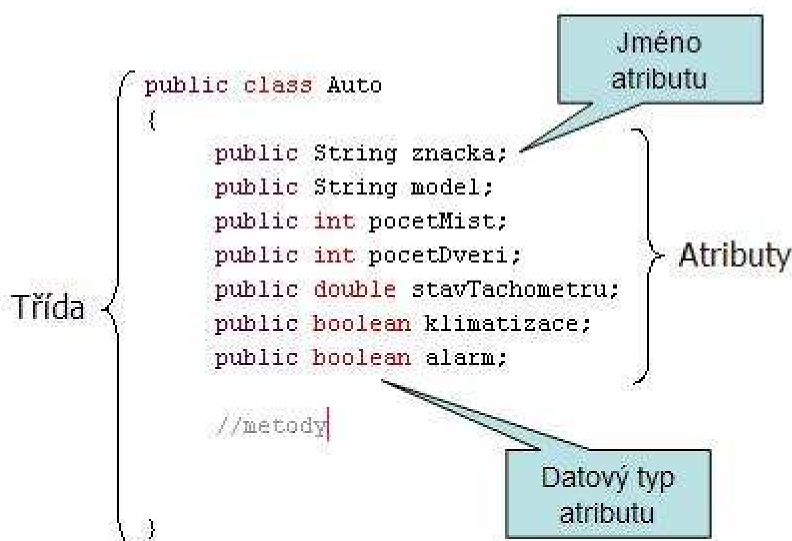
Chceme-li si zobrazit zdrojový kód třídy **Auto**, můžeme tak učinit dvojklikem na obdélník znázorňující tuto třídu. Zdrojový kód se zobrazí v editoru. Jak vidíme na obrázku níže, skládá se zdrojový kód zatím pouze ze slova ***public***(modifikátor přístupu), dále s uvozením třídy a názvu třídy.



Obrázek 4: Kód třídy

4.1 Proměnné - atributy

Proměnné, nebo-li atributy v sobě uchovávají stav objektu(instance třídy). Tyto atributy mohou být různých datových typů(přirozené číslo, reálné číslo, znak,...). S datovými typy se podrobně seznámíme v další kapitole. Je zvykem vypisovat seznam atributů hned za definici třídy, jak ukazuje následující obrázek.



Obrázek 5: Umístění atributů

Dvojklikem na instanci třídy zobrazíme její atributy a jejich aktuální hodnotu. Vše opět ukazuje následující obrázek.

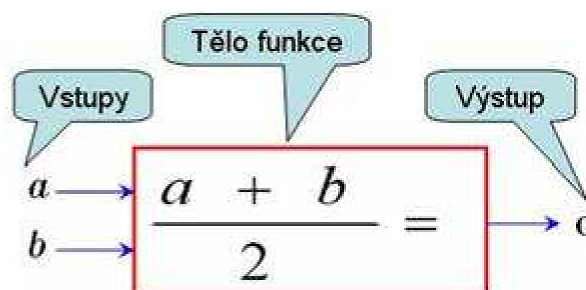
Obrázek 6: Prohlížení instancí

Budeme-li chtít přistoupit k proměnné instance třídy z jiné třídy, můžeme tak učinit pomocí tečnové notace. Vše ukazuje následující příklad.

```
    Auto a = new Auto();
    a.znacka;
```

4.2 Metody

Metodu v objektově orientovaném programovacím jazyce si můžeme představit jako funkci. Tato funkce se obecně skládá ze vstupů, se kterými se pracuje v těle funkce a výstupu, který funkce vrací. Vše znázorňuje následující obrázek.

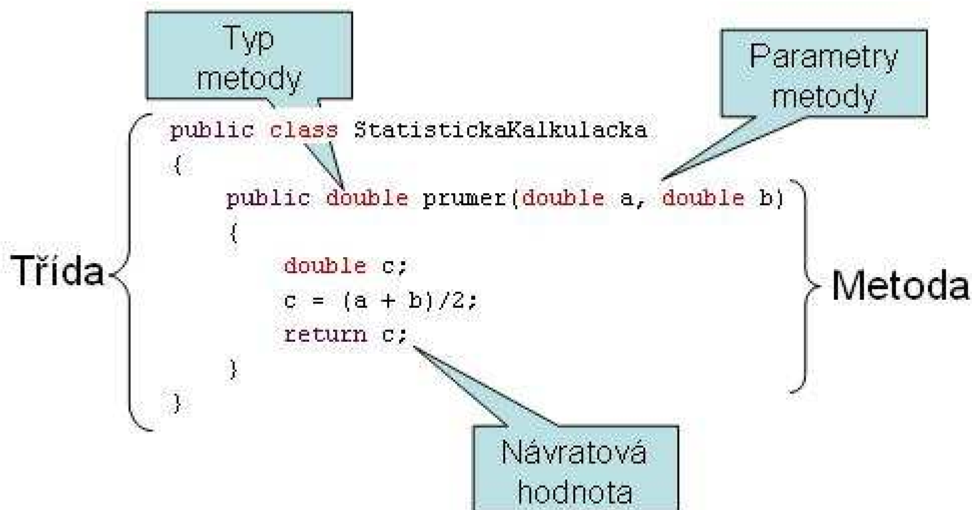


Obrázek 7: Princip práce metody

Deklarace metody ve zdrojovém kódu zahrnuje:

- Modifikátor přístupu(my budeme používat výhradně ***public***, o dalších se dozvíme v kapitole zabývající se přístupovými právy)

* Typ návratové hodnoty - typ metody(***double***, ***int***, ***char***, ... Blíže se s nimi seznámíme v kapitole o základních datových typech)
* Jméno metody
* Kulaté závorky a v nich typy a jména vstupních parametrů
* Tělo metody uzavřené do složených závorek. Tělo obvykle končí příkazem ***return*** a hodnotou, kterou má metoda vrátit.



Obrázek 8: Popis metody ve zdrojovém kódu

4.2.1 Metoda bez parametru

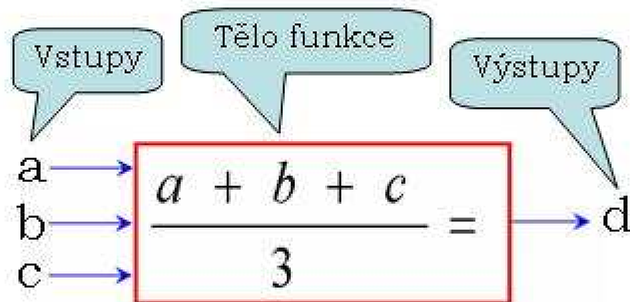
Je možné definovat také metodou bez parametru. V tomto případě ponecháme za názvem metody prázdné závorky.

4.2.2 Metoda bez návratového typu

Chceme-li vytvořit beznávratovou metodu, použijeme takzvaný prázdný typ ***void***.

4.2.3 Přetížená(overloaded) metoda

Dvě metody nemohou mít stejná jména. Výjimku tvoří takzvané přetížené metody, které se lišší od původní metody pouze počtem, či typem parametrů. Následující obrázek demonstruje přetížení metody **prumer**. Metoda tentokrát počítá průměr ze tří reálných čísel.



Obrázek 9: Princip přetížení metody

Pro úplnost ukazuje následující obrázek výpis zdrojového kódu s metodou **prumer** a s přetíženou metodou **prumer**.

```
public class StatistickaKalkulacka
{
    public double prumer(double a, double b)
    {
        double c;
        c = (a + b)/2;
        return c;
    }

    public double prumer(double a, double b, double c)
    {
        double d;
        d = (a + b + c)/3;
        return d;
    }
}
```

metoda prumer

přetížená metoda prumer

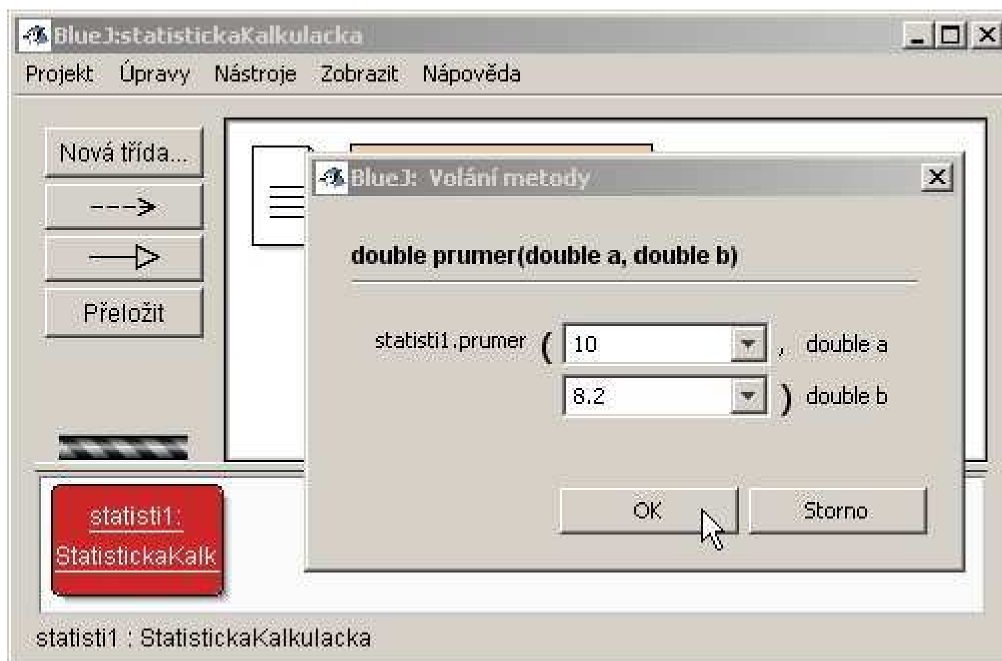
Obrázek 10: Metoda a přetížená metoda v kódu

4.2.4 Volání metody v prostředí BlueJ

Chceme-li zavolat metodu v prostředí BlueJ, stačí kliknout na instanci dané třídy pravým tlačítkem a vybrat příslušnou metodu.

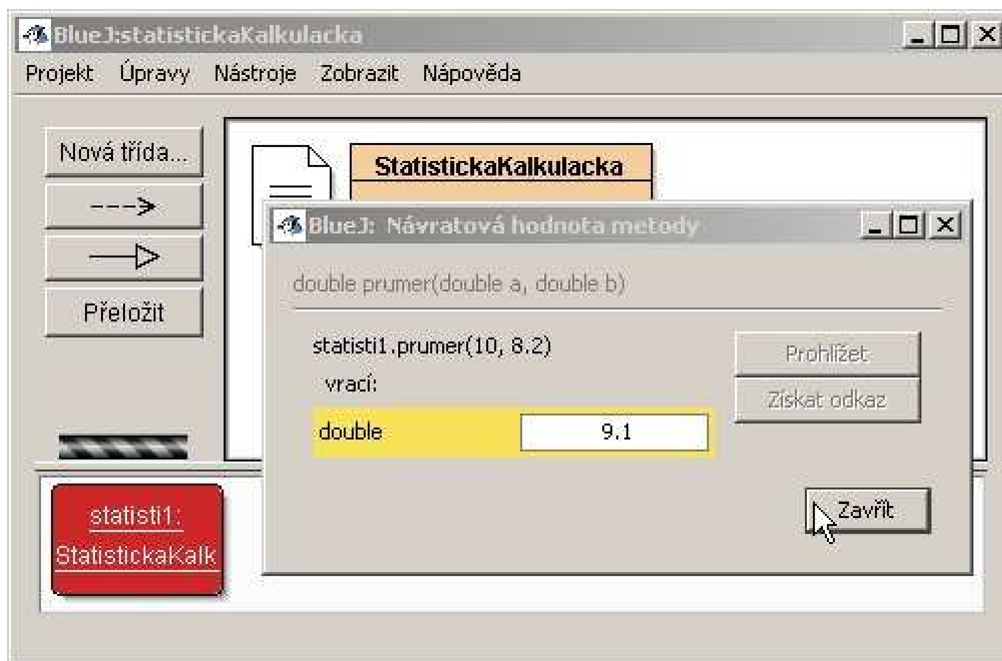
Obrázek 11: Volání metody v prostředí BlueJ

Jedná-li se o metodu s parametrem či parametry, zobrazí se formulář, ve kterém zadáme hodnoty parametru.



Obrázek 12: Zadání parametrů metody

Po vyplnění těchto hodnot a stisknutí tlačítka ok se objeví formulář s výstupní hodnotou.



Obrázek 13: Zobrazení návratové hodnoty

4.3 Konstruktory

Při vytváření instance třídy je vždy volaná speciální metoda nazývaná *konstruktor*. Tato metoda má vždy stejné jméno jako je jméno třídy, není žádného typu a nevrací žádnou hodnotu. Může ale mít parametry. Toho se právě využívá k vytvoření instance třídy, ve které už budou mít dané atributy nastavenou požadovanou hodnotu.

```
public class Auto
{
    public String znacka;
    public String model;
    public int pocetMist;
    public int pocetDveri;
    public double stavTachometru;
    public boolean klimatizace;
    public boolean alarm;

    public Auto(String znackaAuta)
    {
        znacka = znackaAuta;
    }

    public Auto(String znackaAuta, String moldelAuta)
    {
        znacka = znackaAuta;
        model = moldelAuta;
    }
```

Parametr konstruktoru

Konstruktor s parametrem

Inicializace atributů

Přetížený konsturktor

Obrázek 14: Ukázka a popis konstruktoru v kódu

Budeme-li vytvářet instanci třídy, která má více konstruktorů, můžeme si vybrat, který z nich použijeme. Následující obrázek znázorňuje vytváření instance třídy **Auto**, která má dva konstruktory.

Obrázek 15: Vytvoření instance

Poznámka:

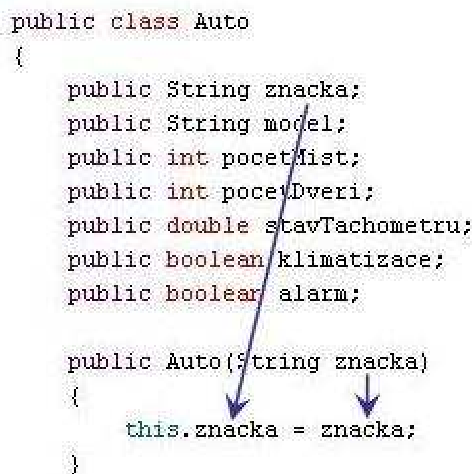
Pokud bychom chtěli například vytvořit instanci třídy **Auto** v kódu jiné třídy, použijeme klíčové slovíčko ***new***. Kód by mohl vypadat následovně.

```
Auto a = new Auto("Škoda");
Auto b = new Auto("Škoda", "Felicie");
```

Předcházející příklad ukázal vytvoření instance třídy **Auto** pomocí obou konstruktorů s parametrem.

4.3.1 Využití this

Vidíme, že v minulém příkladu jsme museli komplikovaně vymýšlet speciální názvy pro parametry metod, abychom je odlišili od názvů atributů. Tomu se můžeme vyhnout použitím klíčového slova ***this***. Vše srozumitelně znázorňuje následující obrázek.



Obrázek 16: Použití this

4.3.2 Implicitní konstruktor

Možná Vás nyní napadne otázka, jak jsme mohli vytvořit instanci třídy v prvním příkladu této kapitoly, když zdrojový kód neobsahoval definici konstruktoru. Odpověď je jednoduchá, pokud konstruktor nenapíšeme, překladač *vytvoří konstruktor implicitní*.

viz [3]

4.4 Příklad 1 - Auto(konstruktory)

4.4.1 Úkol 1

Založte nový projekt **Auto**. Vytvořte třídu **Kolo**, která bude představovat kolo u auta. Třída **Kolo** bude mít dva atributy typu ***double*** **sirka** a **polomer** představující rozměry kola. Dále bude třída **Kolo** obsahovat konstruktor s parametry **sirka** a **polomer** kola typu ***double***. Uvnitř konstruktoru inicializujte atributy. Využijte klíčového slova ***this***.

4.4.2 Úkol 2

Vytvořte novou třídu **Auto**. Třída **Auto** bude mít čtyři atributy typu kolo. Nazvěte je **predniPrave**, **predniLeve**, **zadniPrave**, **zadniLeve**. Dále bude třída obsahovat konstruktor bez parametru. Uvnitř konstruktoru inicializujte proměnné představující kola auta. Všechna kola budou mít stejné rozměry, šířku 21 a poloměr 27. Uvažováno v cm.

4.4.3 Úkol 3

Přidělejte další konstruktor s parametry **sirkaKola**, **polomerKola** typu ***double***. Pomocí tohoto konstruktoru bude možné vytvářet auto s libovolnými koly.

4.4.4 Úkol 4

Vytvořte další konstruktor s parametrem typu kolo. Parametr nazvěte **kolaAuta**.

4.4.5 Úkol 5

Poslední konstruktor, který vytvoříte, bude mít dva parametry typu kolo. Konstruktor vytvořte tak, aby mohla být různá přední a zadní kola.

4.5 Příklad 2 - Auto(metody)

4.5.1 Úkol 1

V této části budeme rozšiřovat projekt z minulého příkladu. Pokud jej nemáte, můžete si ho otevřít ve složce 4.5.1. Rozšiřte projekt o metodu **vymenKola**, která vymění všechna kola za kolo, které bude zadáno v parametru metody. Metoda bude beznávratového typu.

4.5.2 Úkol 2

Vytvořte přetíženou metodu k metodě **vymenKola**, která bude měnit kola přední za jeden typ kola a kola zadní za jiný.

4.5.3 Úkol 3

Vytvořte další metodu bez parametru. Nazvěte ji **sirkaLevehoZadnihoKola**. Tato metoda bude typu ***double*** a bude vracet šířku levého zadního kola.

4.5.4 Úkol 4

Nyní budeme chtít nafouknout naše kola u auta. Přidejte do třídy **Kola** ještě jeden atribut typu ***double*** s názvem **tlak**. Vytvořte ve třídě **Auto** novou metodu **nafukejKola** s parametrem typu ***double*** tlak, která nastaví tlak kola na příslušnou hodnotu. Tato metoda bude beznávratového typu.

4.5.5 Úkol 5

Vytvořte další metodu beznávratového typu s názvem **prifoukni**. Metoda bude bez parametrů a navýší tlak ve všech kolech o 0.1.

4.5.6 Úkol 6

Na závěr vytvořte metodu bez parametru typu ***double***, která bude vracet tlak v levém zadním kole. Metodu nazvěte **tlakVLevemZadnimKole**.

5 Datové typy

Pro studium této kapitoly se předpokládá znalost práce s objekty. Studiem této kapitoly získáte znalosti o základních datových typech a o práci s nimi(*přetypování*).

V Javě se vyskytuje několik druhů základních datových typů:

- Celočíselné
- Reálné
- Znakové
- Logické
- Prázdný datový typ

5.1 Celočíselné typy

Celočíselné datové typy jsou všechny *znaménkové* (nabývají kladných i záporných hodnot), liší se od sebe pouze svojí velikostí a tím i rozsahem zobrazitelných čísel.

Název	Bitů	MIN	MAX
byte	8	-128	127
short	16	-32 768	32 767
int	32	-2 147 483 648	2 147 483 647
long	64	-9 223 372 036 854 775 808	9 223 372 036 854 775 807

Tabulka 2: Celočíselné datové typy

5.2 Reálné typy

Java rozeznává dva reálné datové typy ***float*** a ***double***.

Název	Bitů	Rozsah	
float	32	±1.4E-45	±3.4E+38
double	64	±4.9E-324	±1.7E+308

Tabulka 3: Reálné datové typy

5.3 Logický typ

Používá se typ ***boolean*** o velikosti jeden bit. Tento typ nabývá pouze dvou hodnot, představovaných logickými konstantami ***true*** (= logická 1) a ***false*** (= logická 0).

5.4 Znakový typ

Znakový typ je v Javě pouze jeden a to znakový typ ***char***. Java pracuje se znaky v kódování *Unicode*. Znakové konstanty jsou vždy uzavřeny do apostrofů.

5.5 Řetězce

Pro řetězce se používá typ ***String***. Je nutné podotknout, že se nejedná o základní datový typ. Řetězce se neuvádějí do apostrofů jako znaky, ale do uvozovek.

5.6 Operátor Přetypování

Přetypování nebo-li konverze datových typů se používá tehdy, máme-li k dispozici jeden datový typ a potřebujeme z něj vyrobit jiný datový typ. Operátor se zapisuje ve formě kulatých závorek, uvnitř kterých je jméno datového typu, na který chceme přetypovat.

Př.:

```
int i = 5;
double d;
d = (double) i;
```

5.7 Rozšiřující konverze

Při provádění tohoto typu konverzí nedochází ke ztrátě informace. Jedná se o konverze, kdy výsledný datový typ má větší obor hodnot(rozsah) než původní typ. Operátor přetypování u těchto konverzí není nutný.

Rozšiřující konverze pro základní datové typy jsou tyto: ***byte - short - int - long - float - double***

Schéma znamená, že např. typ ***short*** lze kdykoliv převést na typ ***int***, ***long, float*** a ***double*** bez použití operátoru přetypování.

5.8 Zužující konverze

Při provádění zužující konverze může dojít ke změně nebo ztrátě původní hodnoty. Z tohoto důvodu od nás kompilátor vyžaduje zapsání operátoru přetypování, čímž nás nutí, abychom si uvědomili případné důsledky. Zužující konverze pro základní datové typy jsou tyto: ***double - float - long - int - short - byte***

5.9 Přetypování na String

Budeme-li chtít použít přetypování ostatních datových typů na typ ***String***, můžeme využít metody ***valueOf(datovy typ).***

Př:

```
int cislo = 5;
String retezec = String.valueOf(cislo);
```

5.10 Přetypování Stringu na základní datové typy

Pro převod se používají metody Tříd, které reprezentují základní datové typy. Následující příklad ukazuje přetypování datového typu ***String*** na ***Integer***.

```
String s = "10";
int i = Integer.valueOf(s);
```

5.11 Číselné datové typy a jejich konstanty

Maximální a minimální hodnoty celočíselných i reálných typů lze získat pomocí konstant **MIN_VALUE** a **MAX_VALUE**, před které se dává ještě jméno příslušné třídy - **Byte, Short, Integer, Long, Float, Double.**

Například:

```
int i = Integer.MIN_VALUE;
```

Při operacích s reálnými čísly může výsledek operace nabýt několika "nenormálních" hodnot, které ale nejsou chybné. Jedná se o hodnoty kladné a záporné

nekonečno(příslušné konstanty ze tříd **Float** a **Double** jsou **POSITIVE_INFINITY** a **NEGATIVE_INFINITY**).

Další speciální hodnotou **je NaN** (Not a Number), kterou se podaří získat operací typu dělení nuly nulou.

Všechny tyto tři hodnoty lze otestovat pomocí metod třídy **isInfinite**() a **isNaN**().

viz [3]

5.12 Příklad 1 - Auto

5.12.1 Úkol 1

Otevřete si projekt **Auto**, který je uložen v adresáři 5.12.1. Tento projekt obsahuje jedinou třídu **Auto**. Projekt si stáhněte, spusťte a vytvořte instanci třídy **Auto**. Pomocí metody **`setZnacka(String z)`** nastavte značku Vašeho auta. Poté si vše zkontrolujte pomocí metody **`getZnačka()`**. Prohlédněte si zdrojový kód třídy **Auto**.

5.12.2 Úkol 2

Proměnné **znacka** se říká atribut třídy. Rozšiřte projekt **Auto** o další atributy: **model**, **pocetMist**, **pocetDver**i, **stavTachometru**, **klimatizace**, **alarm** a zvolte pro ně vhodné datové typy. Vytvořte instanci třídy **Auto** a nastavte hodnoty atributů podle následující tabulky.

Značka	**Model**	**Počet míst**	**Počet dveří**	**Stav tachometru**	**Klimatizace**	**Alarm**
Škoda	Fabia	5	5	50132015,5	Ne	Ano

Tabulka 4: Hodnoty atributů

5.13 Příklad 2 - Přetypování

5.13.1 Úkol 1

Otevřete si projekt **Pretypovani**, který je umístěn ve složce 5.13.1. Projekt obsahuje jedinou třídu přetypování se dvěma metodami:

- **`long intNaLong(int cislo)`** - přetypovává datový typ **inteteger** na **long** pomocí implicitní konverze
- **`int longNaInt(long cislo)`** - datový typ **long** na **Integer** pomocí explicitní konverze

Vytvořte instanci třídy **Pretypovani** a obě metody vyzkoušejte. Prohlédněte si zdrojový kód metod.

5.13.2 Úkol 2

Rozšiřte projet Přetypování o metody:

- **`int byteNaInt(byte cislo)`**
- **`byte shortNaByte(short cislo)`**
- **`int stringNaInt(String text)`**
- **`String intNaString(int cislo)`**

Explicitní přetypování použijte jen tam, kde je to nezbytné.

5.14 Příklad 3 - Rozsah

5.14.1 Úkol 1

Ve složce 5.14.1 se nachází projekt Rozsah. Projekt obsahuje jedinou třídu Rozsah. Tato třída má jedinou metodu **`maximalniHodnotaByte()`**, která vrací minimální hodnotu datového typu ***byte***. Projekt si stáhněte, otevřete a vytvořte instanci třídy **Rozsah**. Vyzkoušejte metodu **`maximalniHodnotaByte()`**. Prohlédněte si zdrojový kód.

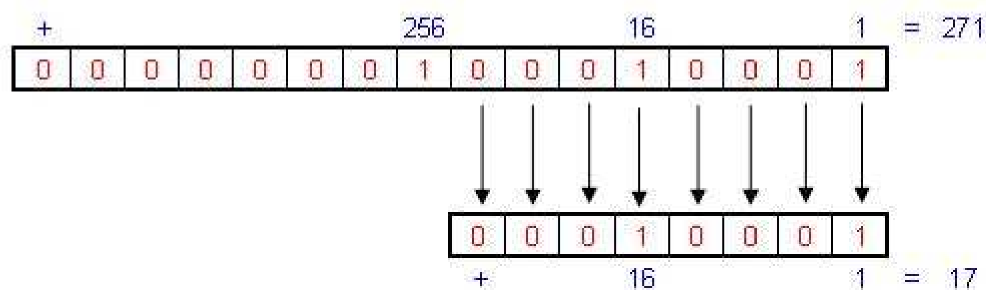
5.14.2 Úkol 2

Rozšiřte třídu Rozsah o příslušné metody tak, aby vracely minimální a maximální hodnoty všech číselných datových typů.

5.15 Příklad 4 - Ztráta přesnosti

5.15.1 Úkol 1

Otevřete si projekt Ztráta přesnosti (5.15.1), který obsahuje jedinou třídu **ZtrataPresnosti** s jedinou metodou `shortNaByte(short cislo).` Tato metoda přetypovává hodnotu typu ***short*** na ***byte***. Projekt si stáhněte, otevřete a vytvořte instanci třídy. Pro přirozená čísla od -128 do 127 dostáváme očekávaný výsledek. Co se ale stane, zadáme-li číslo mimo rozsah datového typu ***byte***? Zadáme-li například hodnotu 273, vrátí se nám výsledek 17. Proč zrovna tato hodnota? Odpověď najdeme v binární reprezentaci přirozeného čísla. Číslo 273 je uloženo jako typ ***short***, tedy typ 16 bitový, kde první bit je znaménkový. Přetypováváme-li na datový typ byte(8 bitů), zachová se osm bitů nejvíce vpravo. Ostatní bity budou ztraceny. Vše ukazuje následující obrázek.



Obrázek 17: Zobrazení čísel ve dvojkové soustavě

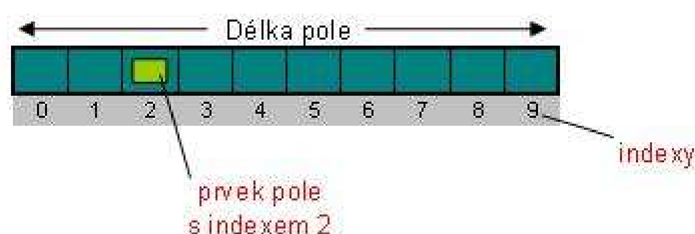
5.15.2 Úkol 2

Nalezněte přirozené číslo od 126 do 32 767, které dává po přetypování z **shortNaByte** výsledek 34. Kolik takových čísel existuje?

6 Pole

Pro studium této kapitoly je nezbytná znalost základů objektově orientovaného programování a základních datových typů v jazyce Java. Studiem této kapitoly se naučíte deklarovat, vytvořit a inicializovat jednoduchá i vícerozměrná pole.

Pole je kontejner, do kterého lze ukládat hodnoty jednoho typu. Délka pole je pevně stanovena při jeho založení. K jednotlivým prvkům pole můžeme přistupovat pomocí indexů.



Obrázek 18: Znázornění pole

6.1 Deklarace pole

Deklarace pole se skládá ze dvou částí - z typu pole a jeho jména. Př: Deklarace pole typu ***int***, pojmenované **sudaCisla**

```
int [] sudaCisla;
```

6.2 Vytvoření pole

Vytvoření pole se provádí pomocí klíčového slova ***new***, typu pole a jeho velikosti v hranatých závorkách.

```
sudaCisla = new int [10];
```

6.3 Inicializace (plnění) pole

Plnění pole provádíme tak, že pomocí indexu určíme konkrétní políčko v poli, do kterého vložíme příslušnou hodnotu.

```
sudaCisla[0] = 2;
sudaCisla[1] = 4;
sudaCisla[2] = 6;
```

6.4 Přístup k prvkům pole

Každý prvek pole je přístupný pomocí *indexu*.

```
int sudeCislo = sudaCisla[1];
```

6.5 Délka pole

Kdykoliv během výpočtů můžeme zjistit délku vytvořeného pole pomocí proměnné ***length***.

```
int delkaPole = sudaCisla.length;
```

6.6 Inicializované pole

Můžeme také vytvořit pole, které bude inicializované příslušnými hodnotami. Takové pole nevytváříme pomocí klíčového slůvka ***new***, ale pomocí *statického inicializátoru*.

```
int [] lichaCisla = {1, 3, 5, 7};
```

6.7 Vícerozměrná pole

S vícerozměrnými poli se pracuje podobně jako s poli jednorozměrnými. Deklarace a vytvoření vícerozměrného pole:

```
int [][] matice = new int [10][15];
```

Př: Inicializování dvourozměrného pole.

```
int [][] matice = {{1, 2, 3},
             {3, 2, 1}};
```

viz [3]

6.8 Příklad 1 - Parkoviště

6.8.1 Úkol 1

Ze složky 6.8.1 si otevřete projekt **Auto** známý z kapitoly datové typy. Tento projekt je rozšířen o třídu Parkoviště s jednou globální proměnnou **parkoviste** typu pole Auto. Rozšiřte tuto třídu o metodu **`void vytvorParkovaciMista(int`**

pocet), která bude sloužit k vytvoření pole **parkoviste**. Proměnná počet reprezentuje kapacitu parkoviště.

6.8.2 Úkol 2

Rozšiřte třídu **Parkoviste** o metodu **void zaparkujAuto(Auto a, int misto),** která bude ukládat instanci typu auta. Proměnná místo reprezentuje index pole, kam bude auto uloženo.

6.8.3 Úkol 3

Přidejte další metodu void **vypisAutoNaPozici(int pozice),** která vrátí značku auta umístěného v poli s indexem pozice.

6.8.4 Úkol 4

Na závěr tohoto cvičení vytvořte metodu **int kapacitaParkoviste(),** která vrátí kapacitu našeho parkoviště (tedy délku pole parkoviště).

6.9 Příklad 2 - Parkoviště 2

6.9.1 Úkol 1

Rozšiřte projekt **Auto** o třídu **ObdelnikoveParkoviste**. Auta se budou tentokrát ukládat do dvourozměrného pole typu Auto o názvu parkoviste. Třída bude obsahovat tyto metody:

- **vytvorParkovaciMista**
- **zaparkujAuto**
- **vypisAutoNaPozici**
- **kapacitaParkoviste**

6.9.2 Úkol 2

Rozšiřte tentokrát projekt o Třídu **VíceposchodoveParkoviste**. Auta se budou ukládat do trojrozměrného pole. Třída bude obsahovat tyto metody:

- **vytvorParkovaciMista**
- **zaparkujAuto**
- **vypisAutoNaPozici**
- **kapacitaParkoviste**

7 Řídící konstrukce

7.1 Porovnávání

Ke studiu této kapitoly je nutné mít základní znalosti objektově orientovaného programování, znalosti o základních datových typech. Studiem této kapitoly získáte základní dovednosti s podmínkovými příkazy ***if, if-else*** a ***switch***.

Java poskytuje šest relačních operátorů pro porovnávání dvou datových hodnot. Kterékoli z datových hodnot, které budete porovnávat, mohou být proměnné, konstanty nebo výrazy vycházející z primitivních datových typů.

Operátory porovnávání	>	>=	= =	!=	<=	<
Popis	Je větší	Je větší rovno	Je rovno	Není rovno	Je menší rovno	Je menší

Tabulka 5: Operátory porovnávání

7.2 Logické operátory

V Javě existuje celkem pět logických operátorů. Slouží k vytváření složených logických výrazů. Jejich logické vyhodnocování se řídí pravidly výrokové logiky.

<table>
<tr><td>Logické operátory</td><td>&</td><td>&&</td><td>|</td><td>||</td><td>!</td></tr>
<tr><td>Popis</td><td>Logické A</td><td>Podmíněné A</td><td>Logické NEBO</td><td>Podmíněné NEBO</td><td>Logická NEGACE</td></tr>
</table>

Tabulka 6: Logické operátory

Rozdíl mezi podmíněným a logickým operátorem je ten, že pokud u podmíněného operátoru nabude výraz na levé straně hodnoty ***false***, nebude se pravý výraz vůbec vyhodnocovat, kdežto u logického operátoru ano.

7.3 Příkaz if

```
if (výraz)
{
     // příkazy
}
```

Pokud je hodnota výrazu rovna ***true***, je proveden příkaz, který následuje za ***if***, jinak proveden není.

7.4 Příkaz if - else

```
if (výraz)
{
     // příkazy 1
}
else
{
     // příkazy 2
}
```

Základní příkaz ***if*** můžeme rozšířit přidáním části ***else***. To poskytuje možnost zadat jiný příkaz, který je proveden pokud, je výraz v příkazu ***if*** roven ***false***.

7.5 Příkaz switch

```
switch () {
  case 1:
     // Příkazy 1
     break;
  case 2:
     // Příkazy 2
     break;
  case 3:
     // Příkazy 3
     break;
  default:
     // Příkazy 4
```

Příkaz ***switch*** (přepínač) umožňuje výběr mezi více možnostmi podle hodnoty daného výrazu. Výraz musí být typu ***char***, ***byte***, ***short*** nebo ***int***. Výběr je stanoven hodnotou výrazu, kterou zadáte a která je uzavřena v závorkách za klíčovým slovem ***switch***. Možná nastavení přepínače definujeme jedním nebo více hodnotami ***case***, kterým se také říká návěsti. Pokud je uveden příkaz ***break***, znamená to, že dále budou

prováděny za uzavírací složenou závorkou pro ***switch***. Příkaz break není povinný, ale pokud nevložíte příkaz ***break*** na konec příkazů pro daný případ, příkazy pro další případy, které následují, budou také provedeny, až dokud nebude nalezen nějaký příkaz ***break*** nebo dokud se nedojde na konec bloku ***switch***.

viz [1]

7.6 Příklad 1 - Banka

7.6.1 Úkol 1

Ze složky 7.6.1 si otevřete v prostředí BlueJ projekt **Banka.** Jak vidíte, projekt obsahuje jedinou třídu **Klient**, jedinou globální proměnnou **stavKonta**. Dále obsahuje konstruktor s parametrem **stavKonta**. Parametr je typu ***int***. Veřejnou metodu typu ***void*** **DruhKlienta**. Upravte tuto metodu tak, aby prověřila velikost klientova konta. Bude-li alespoň jeden milion, vypíše o tom informaci do terminálového okna.

7.6.2 Úkol 2

Rozšiřte podmínkový příkaz ***if*** o větev ***else*** tak, aby metoda **druhKlienta** vypisovala informace o tom, zda klient je nebo není milionář.

7.6.3 Úkol 3

Modifikujte třídu Klient, tak aby informovala o klientech následujícím způsobem:

0-99 999	100 000 - 999 999	1 000 000 a více
Chudák	Průměr	Milionář

Tabulka 7: Rozřazení klientů

Nápověda: Použijte znalosti o logických operátorech.

7.7 Příklad 2 - Známkování

7.7.1 Úkol 1

Ze složky 7.7.1 si otevřete projekt **Znamkovani**. Projekt obsahuje pouze jednu prázdnou metodu **oznamkuj**. Doplňte tělo této metody, aby oznámkovala studentovu práci podle počtu získaných bodů. Známkovací kritéria zachycuje následující tabulka. Pro řešení využijte logické operátory.

Počet bodů	100 - 90	89 - 75	74 - 45	44 - 25	24 - 0
Známka	1	2	3	4	5

Tabulka 8: Známkování

7.7.2 Úkol 2

Upravte předcházející projekt tak, aby metoda **oznamkuj** pracovala bez použití logických operátorů.

7.7.3 Úkol 3

Rozšiřte třídu **Znamkovani** o metodu typu ***string*** **slovneOhodnot** s parametrem **znamka** typu ***int***. Použijte příkaz ***switch***. Známce přiřaďte hodnocení podle následující tabulky.

Známka	1	2	3	4	5
Slovní hodnocení	Výborně	Chvalitebně	Dobře	Dostatečně	Nedostatečně

Tabulka: 9 Známkování

7.7.4 Úkol 4

Na některých školách se používá pro hodnocení nikoliv číselných hodnot ale písmen. Rozšiřte třídu **Znamkovani** o metodu typu ***int*** **prevedNaCiselneHodnoceni** s parametrem typu ***char***. Uvažujte převod podle následující tabulky. Použijte příkaz ***switch***.

Známka	a	b	c	d	e
Číselná známka	1	2	3	4	5

Tabulka 10 : Převod známek

7.8 Příklad 3 - Kalendář

7.8.1 Úkol 1

Ze složky 7.8.1 si otevřete projekt **Kalendar**. Projekt obsahuje pouze jednu třídu nazvanou Kalendář. Tato třída obsahuje metodu typu **String den(int den)** s parametrem den typu ***int***, který představuje číslo dnu v týdnu. Doplňte tělo metody tak, aby vracela název příslušného dne.

7.8.2 Úkol 2

Rozšiřte třídu **Kalendar** o metodu typu **String** **`odpoledniProgram(int den)`** s parametrem číslo dne. Metoda bude vracet činnost, kterou dělám pravidelně v týdenních intervalech. Vše znázorňuje následující tabulka.

Číslo dne	1	2	3	4	5	6	7
Odpolední činnost	Plavání	Kopaná	Kopaná	Kopaná	Kolo	Kolo	Odpočinek

Tabulka 11: Týdenní rozvrh

Nápověda: Použijte znalosti o logických operátorech.

7.8.3 Úkol 3

Pokuste se vyřešit předchozí problém za použití příkazu ***switch*** a bez použití logických operátorů. Nápověda: Za každým příkazem ***case*** nemusí následovat ***break***.

8 Cykly

8.1 Předpokládané znalosti

Pro studium této kapitoly se předpokládá znalost základních datových typů, základy práce s polem a základy práce s třídou a metodami.

Student si osvojí práci s cykly, které budou umožňovat opakovaně provádět příkaz nebo blok příkazů. Existují tři druhy cyklů. *Cyklus for*, *cyklus while* a *do-while*.

8.2 Cyklus for

```
for(inicializační_výraz;
podmínka_opakování;výraz_pro_zvýšení)
{
     // příkazy
}
```

První část, *inicializační_výraz*, je provedena před začátkem cyklu. Je zpravidla využita k inicializaci počítadla pro počet opakování cyklu. U cyklu, který je řízen počítadlem, můžeme počítat nahoru nebo dolů pomocí reálné či celočíselné proměnné. Provádění cyklu pokračuje tak dlouho, dokud podmínka zadaná v druhé části, podmínka_opakování, je ***true***. Tento výraz je kontrolován na začátku každého opakování. Třetí část cyklu se používá pro zvýšení či snížení počítadla cyklů. Toto se provede na konci každého cyklu.

8.3 Cyklus while

```
while (výraz)
{
     // příkazy
}
```

Cyklus se provádí tak dlouho, dokud je daný logický výraz v závorkách ***true***. Jakmile má výraz hodnotu ***false***, provádění pokračuje příkazem, který je za blokem cyklu. Výraz je testován na začátku cyklu.

8.4 Cyklus do while

```
do {
     // příkazy
} while (výraz);
```

Cyklus je podobný cyklu ***while***, jen s tím rozdílem, že výraz, který jej řídí, je testován na konci bloku cyklu. To znamená, že blok cyklu je proveden alespoň jednou.

viz [1]

8.5 Příklad 1 - Cykly

8.5.1 Úkol 1

Ze složky 8.5.1 si otevřete projekt **Cykly** v prostředí BlueJ. Projekt obsahuje pouze jednu třídu nazvanou **Cykly**. Tato třída má tři metody, které slouží k naplnění pole přirozenými čísly. Každá z těchto metod používá jiného cyklu. Vytvořte instanci třídy a vyzkoušejte postupně všechny tři metody.

8.5.2 Úkol 2

Pokuste se všechny metody upravit tak, aby vracely přirozená čísla od 5 do 15.

8.5.3 Úkol 3

Pro potřeby dalšího cvičení si otevřete projekt ze složky 8.5.3. Tento projekt je lehkou modifikací předcházejících projektů. Metody třídy Cykly jsou tentokrát metody s parametrem, který představuje hodnotu, do které mají být vypsána přirozená čísla. Metody otestujte a prohlédněte si zdrojový kód třídy **Cykly**. Jako cvičení pozměňte definici cyklů (ne jejich těl!) tak, aby vypisovaly čísla od zadané hodnoty do jedné, čili sestupně.

8.5.4 Úkol 4

Otevřete si projekt ze složky 8.5.4 a upravte ho tak, aby metoda **cyklusFor** vracela všechna sudá čísla do stanovené hodnoty, **cyklusWhile** čísla dělitelná třemi a metoda **cyklusDoWhile** čísla dělitelná pěti.

8.6 Příklad 2 - Matice

8.6.1 Úkol 1

Tento příklad slouží k praktické ukázce použití vnořených cyklů. Otevřete si projekt **Matice**, který se nachází ve složce 8.6.1. Projekt obsahuje jednu třídu **Matice**. Třída Matice obsahuje metodu **tiskniMatici**, která vytiskne jednotkovou matici 5 krát 5. Metodu vyzkoušejte a prohlédněte si zdrojový kód.

8.6.2 Úkol 2

Rozšiřte třídu **Matice** o přetíženou metodu **tiskniMatici** s parametry typu ***int***, které budou reprezentovat počet řádků a sloupců tisknuté matice.

8.6.3 Úkol 3

Rozšiřte třídu **Matice** o další přetíženou metodu **tiskniMatici** s parametry typu ***int***, které budou reprezentovat počet řádků a sloupců tisknuté matice. Dále o parametr pole typu ***int***, představující hodnoty buněk v matici.

8.6.4 Úkol 4

Rozšiřte třídu Matice o metodu **`vynasobMatice (int radky1, int sloupce1, int [] poleCisel1, int radky2, int sloupce2, int [] poleCisel2)`**. Jak je vidět z parametrů metody, je zadávaná každá matice jako pole čísel společně s počtem řádků a sloupců.

Nápověda:

Pro přehlednost doporučuji vytvořit pomocnou **metodu vytvorMatici (int radky, int sloupce, int [] poleCisel)**, která bude vracet matici reprezentovanou pomocí dvourozměrného pole.

8.6.5 Úkol 5

Jako závěrečný bonbónek vytvořte metodu **tiskniMatici (int [][] poleCisel)**, pomocí které bude možno vytisknout matici vypočtenou v metodě **vynasobMatice**.

8.7 Příklad 3 - Šifrování

8.7.1 Úkol 1

V projektu se budeme zabývat šifrováním textu. Každému znaku bude přidělena dvojice znaků podle tabulky. Například písmeno **l** bude zašifrováno jako **cb**. Otevřete si projekt ze složky 8.7.1. Doplňte tělo metody **zasifruj(String text)** tak, aby vracela zašifrovaný zadaný text jako objekt typu string.

	a	b	c	d	e
a	a	b	c	d	e
b	f	g	h	i	j
c	k	l	m	n	o
d	p	q	r	s	t
e	u	v	w	x	y
f	z	.	!	?	

8.7.2 Úkol 2

Rozšiřte projekt Šifrování o metodu **desifruj (String text)**, která bude dešifrovat zašifrovaný text.

8.7.3 Úkol 3

Na závěr cvičení vytvořte metodu **vytvorZasifrovaneZnaky(String kod1, String kod2)**, která bude umožňovat vytvoření vlastního šifrovacího pole pro zašifrování znaků.

Námět k této úloze jsem získal na webových stránkách pana Melouna na URL adrese: http://vit-meloun.net

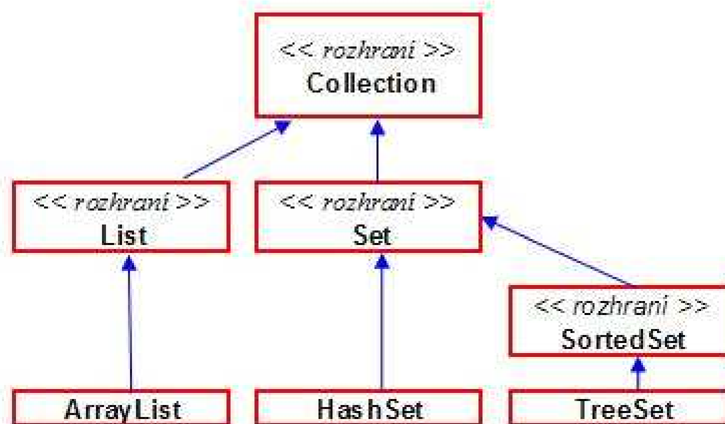
9 Kolekce - kontejnery

Studium této kapitoly předpokládá základní znalosti objektově orientovaného programování, znalost datových typu a znalost práce s polem. Studiem této kapitoly získáte základní znalosti o dynamických datových strukturách z knihovny **Java.util**.

Kolekce jsou objekty tříd, které slouží k uchovávání objektů. Tyto třídy jsou uloženy v balíku **Java.util**.

Používání kolekcí především snižuje délku kódu. Kód se stává přehlednější. Většina důležitých algoritmů(vyhledávání, řazení, ...) je již naprogramována v těchto třídách. Protože tyto třídy využívají co možná nejrychlejší algoritmy pro danou operaci, používání kolekcí urychluje program.

Protože knihovna **Java.util** je velice rozsáhlá, nebudeme si ji představovat celou. Seznámíme se pouze s nejzákladnějšími, ale zároveň nejpoužívanějšími rozhraními a třídami.



Obrázek 19: Závislost rozhraní a tříd

Rozhraní **Collection** slouží jako společný předek pro ostatní objekty. List představuje společné rozhraní pro seznamy. Prvky seznamů jsou přístupné pomocí indexů a mohou se v seznamu opakovat.

ArrayList - pole proměnné délky, při inicializaci není nutné zadávat délku pole.

Set - rozhraní k množinám. Prvky nejsou uspořádané a nemohou se opakovat.

HashSet - pro průchod kolekcí nelze použít indexaci. Je nutné použít iterátor. Pro přístup k jednotlivým prvkům se používá hashovací funkce.

HashMap - prvky jsou zde ukládány jako dvojice klíč - hodnota. Pomocí klíče můžeme tuto dvojici najít.

Následující tabulky jsou věrným obrazem dokumentace základních rozhraní z balíčku java.util.

Doporučuji si vyhledat tyto informace zároveň v dokumentaci (http://java.sun.com/javase/6/docs/api/), a tím si procvičit práci s ní.

viz[5]

Metody rozhraní Collection	
`boolean`	**add(E e)** Vloží prvek do kolekce.
`boolean`	**addAll(Collection<? extends E> c)** Vloží všechny prvky z jiné kolekce.
`void`	**clear()** Odstraní všechny prvky z kolekce.
`boolean`	**contains(Object o)** Vrátí hodnotu true, pokud kolekce obsahuje danný prvek.
`boolean`	**containsAll(Collection<?> c)** Vrátí hodnotu true, pokud kolekce obsahuje všechny prvky zadané kolekce.
`boolean`	**equals(Object o)** Porovná objekt s kolekcí.
`int`	**hashCode()** Vrátí hodnotu hešovacího kódu této kolekce.
`boolean`	**isEmpty()** Vrátí hodnotu true, pokud je kolekce prázdná.
`Iterator<E>`	**iterator()** Vrátí iterátor, pomocí kterého lez procházet kolekci.
`boolean`	**remove(Object o)** Odstraní daný prvek z kolekce. Pokud je prvků více odstraní,

	libovolný z nich.
boolean	**removeAll(Collection<?> c)** Odstaní všechny prvky, které se nacházejí v zadané kolekci c.
boolean	**retainAll(Collection<?> c)** Ponechá v kolekci pouze prvky, které se nacházejí v zadané kolekci c.
int	**size()** Vrátí počet prvků v kolekci.
Object[]	**toArray()** Vrátí pole typu objekt obsahující všechny prvky v kolekci.
<T> T[]	**toArray(T[] a)** Vrátí pole konkrétního typu obsahující všechny prvky v kolekci.

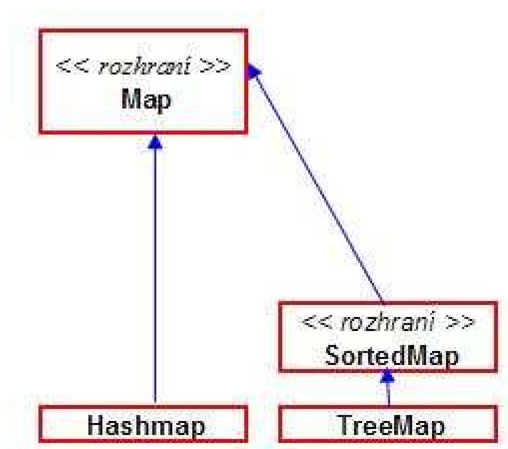
Tabulka 12: Přehled metod

Metody rozhraní List	
void	**add(int index, E element)** Vloží prvek na danou pozici. Prvky se stejným a vyšším indexem budou posunuty.
void	**addAll(int index, Collection<? extends E> c)** Vloží prvky kolekce na danou pozici.
E	**get(int index)** Vrátí prvek na dané pozici v listu.
int	**indexOf(Object o)** Vrátí hodnotu prvního nalezeného prvku shodného s parametrem metody. Pokud takový prvek nenajde, vrátí -1.
int	**lastIndexOf(Object o)** Vrátí hodnotu posledního nalezeného prvku shodného s parametrem metody. Pokud takový prvek nenajde, vrátí -1.
E	**set(int index, E element)**

	Nahradí prvek na určité pozici daným prvkem.
`List<E>`	**subList(int fromIndex, int toIndex)** Vrátí podseznam (zleva uzavřený) prvku mezi zadanými indexy.

Tabulka 13: Přehled metod

Rozhraní Set nepřináší oproti rodičovskému rozhraní **Collection** žádné nové metody.



Obrázek 20 Schéma map

Metody rozhraní map	
`void`	**`clear`**`()` Odstraní všechny položky z mapy.
`boolean`	**`containsKey`**`(Object key)` Vrátí true, pokud mapa obsahuje zadaný klíč.
`boolean`	**`containsValue`**`(Object value)` Vrátí true, pokud mapa obsahuje zadanou hodnotu.
`Set<Map.Entry<K,V>>`	**`entrySet`**`()`
`boolean`	**`equals`**`(Object o)` Porovná zadaný objekt s objekty v mapě.
`V`	**`get`**`(Object key)` Vrátí hodnotu, která přísluší k zadanému klíči.

boolean	**isEmpty**() Vrátí hodnotu true, pokud mapa neobsahuje žádný prvek.
Set<K>	**keySet**() Vrátí množinu klíčů obsažených v mapě.
V	**put**(K key, V value) Vloží dvojici klíč- hodnota do mapy.
V	**remove**(Object key) Podle zadaného klíče odebere příslušnou dvojici klíč, mapa.
int	**size**() Vrátí počet prvků v mapě.
Collection<V>	**values**() Vrátí kolekci obsahující hodnoty v mapě.

Tabulka 14 Přehled metod

viz [7]

9.1 Iterátory

Iterátor je objekt, pomocí kterého můžeme postupně získat všechny prvky v kolekci. Již v rozhraní **Collection** je definována metoda, která má vracet objekt typu **iterátor**. Jednotlivé **iteratory** implementují rozhraní **Iterator**, ve kterém jsou definovány základní metody.

Metody z rozhraní Iterátor	
boolean	**next()** Vrací další prvek získaný iterací.
E	**hasNext()** Vrátí hodnotu true, jestliže kolekce obsahuje další prvek.
Void	**remove()** Odstraní poslední prvek vrácený iterátorem.

Tabulka 15: Přehled metod

9.1.1 Ukázka práce s iterátorem

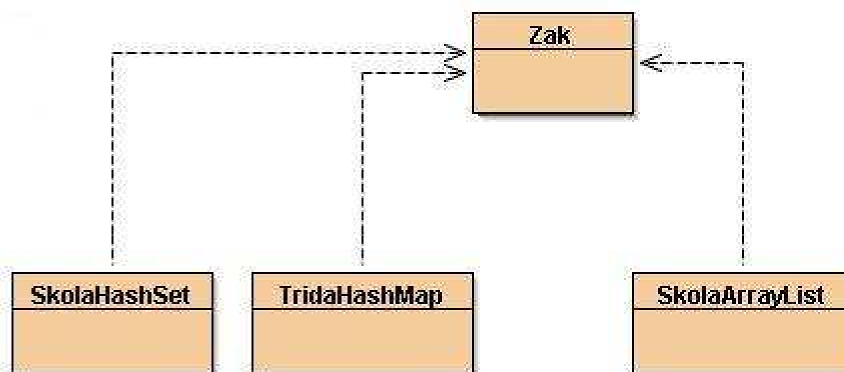
Následující kód ukazuje průchod množiny, která uchovává objekty typu **String**, iterátorm. Všiměte si, že metoda next() vrací objekt typu **Object**, který je přetypováván na typ **String**.

```
HashSet mnozina = new HashSet();
Iterator i = mnozina.iterator();
public void projdiMnozinu()
{
    while(i.hasNext())
    {
        String text = (String)i.next();
    }
}
```

9.2 Příklad Škola

9.2.1 Úkol 1

Založte nový projekt **Skola**. V tomto projektu vytvořte prázdné třídy **Zak**, **SkolaHashSet**, **SkolaHashMap**, **SkolaArrayList**. Třída **Zak** bude mít veřejné atributy **jmeno**, **prijmeni**, **rodneCislo**. Ostatní třídy budou uchovávat instance třídy **Zak** v příslušných kontejnerech. Typ kontejneru zvolte samozřejmě podle názvu třídy. Kontejnery nazvěte **mnozina**, **mapa** a **list**. V těchto třídách definujte metody pro uložení žáka do kolekce a vypsání počtu žáků ve třídě. Pro třídu **SkolaHashMap** použijte jako klíč hodnotu id typu ***int***, která bude představovat interní jedinečné označení studentů. Vytvořte instance těchto tříd. U třídy **Zaci** alespoň dvě. Vyzkoušejte uložení těchto instancí do jednotlivých tříd. Testujte hlavně chování při ukládání stejných instancí.



Obrázek 21: Třídy

9.2.2 Úkol 2

Vytvořte ve třídě **Zak** konstruktor s parametry **jmeno**, **prijmeni** a **rodneCislo**, v jehož těle se budou proměnné inicializovat. Vytvořte v prostředí BlueJ dva žáky se stejnými atributy a uložte je do instance třídy **SkolaHashSet**. Výsledek Vás možná překvapí. Z hlediska charakteru úlohy se jedná o stejné žáky, ale přesto se uložily obě instance třídy žák. Problém je v metodách **equals** a **hashCode** které dědí třída **Zak** od

třídy **Object**. Ve třídě **Zak** tyto metody překryjte tak aby, žáci se stejným rodným číslem nemohli být uloženi (dva různí žáci mohou mít stejná jména a příjmení, ale ne rodná čísla). Výsledek opět otestujte.

9.2.3 Úkol 3

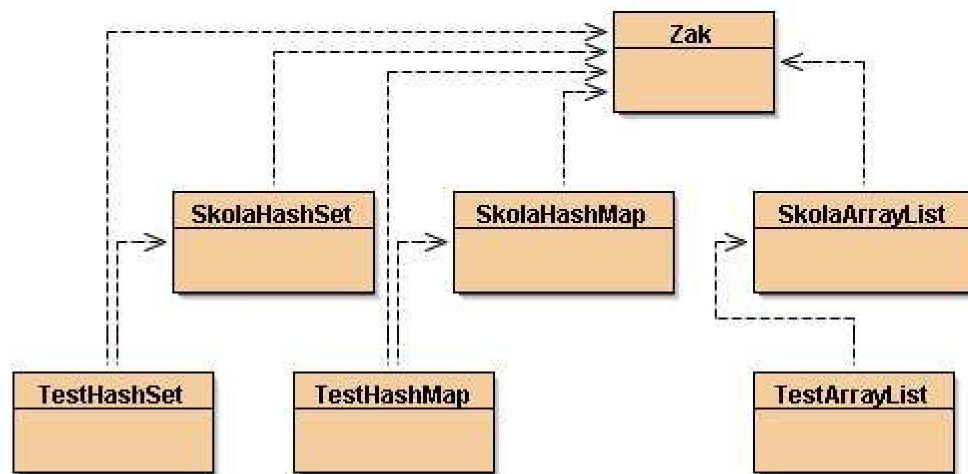
Vytvořte ve třídách **SkolaHashSet**, **SkolaHashMap** a **SkolaArrayList** metody pro smazání všech žáků, smazání určitého žáka a metodu, která zjistí, zda třída obsahuje určitého žáka. U třídy **TridaHashMap** navíc vytvořte metodu, která najde žáka podle klíče.

9.2.4 Úkol 4

V příslušných třídách vytvořte metody, které projdou kolekce pomocí iterátoru a do terminálového okna BlueJ vypíší příjmení uložených studentů.

9.2.5 Úkol 5

Vytvořte třídy **TestHashSet**, **TestHashMap** a **TestArrayList**, které budou obsahovat metody pro vytvoření předem stanoveného množství žáků a jejich uložení do příslušné školy. Metody budou vracet dobu trvání operace v milisekundách. Obdobně postupujte při vytvoření metody, která bude testovat dobu průchodu kolekcí iterátorem(zakomentujte řádek vypisující příjmení žáků do terminálového okna).



Obrázek 22: Třídy

10 Input - Output

Studium této kapitoly předpokládá základní znalosti objektově orientovaného programování, znalosti o datových typech a znalost práce s polem. Studiem této kapitoly získáte základní znalosti o vstupních a výstupních proudech, které čtou či zapisují do souboru. Dále se seznámíte s použitím *Bufferu*.

Balík na podporu vstupů a výstupů pomocí proudů se jmenuje **Java.oi**. Od verze **Javy 1.4** přibyly další dva balíčky podporující vstupní a výstupní proudy. Jsou to balíčky **Java.nio** a **Java.nio.channels**.

Proud si můžeme představit jako sekvenci bajtů. Tato sekvence může být načítána (např. ze souboru), pak mluvíme o vstupním proudu, nebo odesílaná (např. do souboru), pak mluvíme o výstupním proudu.

Rozeznáváme dále dva druhy proudů a to binární proudy, které přenášejí binární data, a proudy znakové , které obsahují znaková data.

Balíček Java.io obsahuje velké množství tříd, které vytvářejí logickou strukturu. Základ bajtových proudů tvoří dvě abstraktní třídy. Třída InputStream pro vstupní operace a třída OutputStream pro výstupní operace.

Základ znakových proudů tvoří také dvě abstraktní třídy. Třída Reader pro vstupní prouda a třídy Writer pro výstupní.

viz[5]

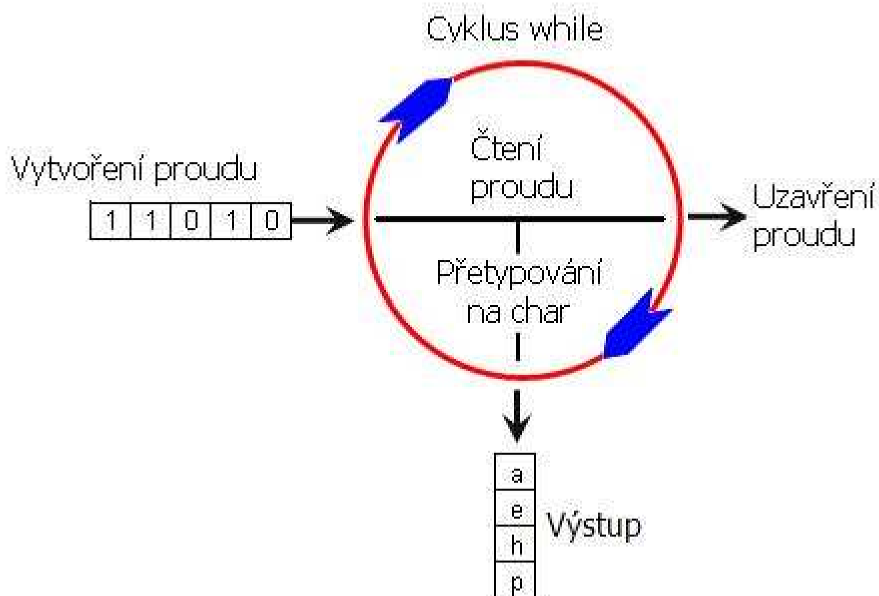
10.1.1 Vstupní proudy

My budeme nejvíce pracovat se třídou **FileReader**. Následující kód ukazuje vytvoření instance třídy **FileReader** a přečtení jednoho znaku.

```
FileReader fr = new FileReader("soubor.txt");
fr.read();
fr.close();
```

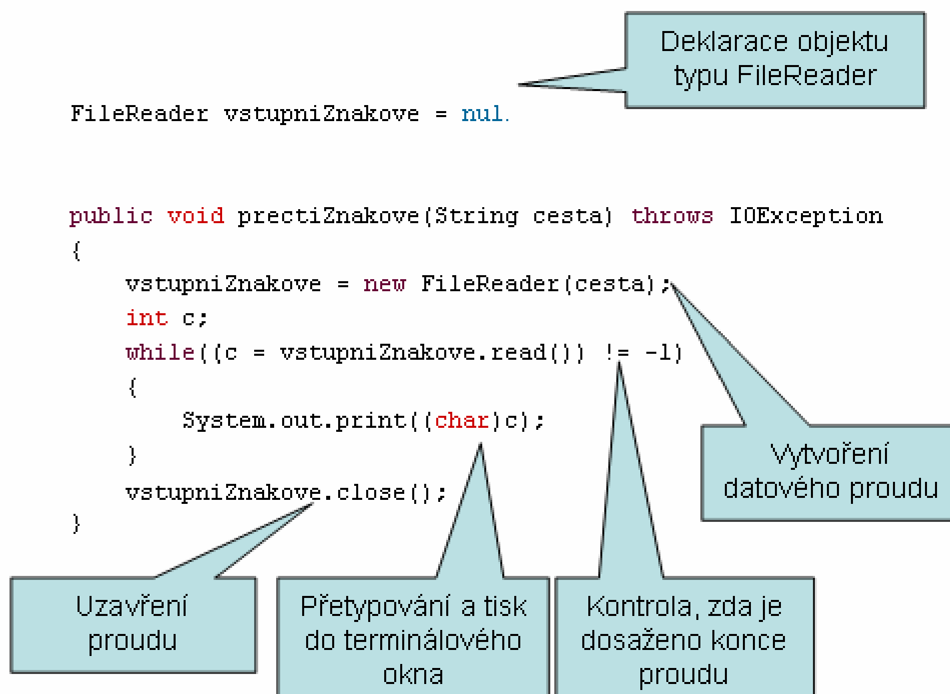
Nejdůležitější metodou tříd pracujících se vstupními proudy je metoda **read().** Tato metoda vrací vždy následující bajt ze vstupního proudu jako hodnotu typu ***int***. Pokud je dosaženo konce proudu, metoda vrátí hodnotu -1. Pokud dojde k chybě, bude vyvolána výjimka **IOException**.

Přejděme k praktické ukázce, pokusíme se načíst textový soubor a vypsat data do terminálového okna BlueJ. Myšlenku naznačuje následující obrázek.



Obrázek 23: Schéma práce s vstupním proudem

Nejdříve vytvoříme vstupní proud. Z toho budeme číst data metodou **read** v těle cyklu ***while***. V těle cyklu budeme také přetypovávat hodnoty ***int*** na ***char*** a vypisovat je do terminálového okna. Jakmile dojde k přečtení všech dat, nebude splněna podmínka pro běh cyklu ***while***, program bude pokračovat za cyklem a uzavře proud.



Obrázek 24: Popis kódu pracující se vstupním proudem

10.1.2 Výstupní proudy

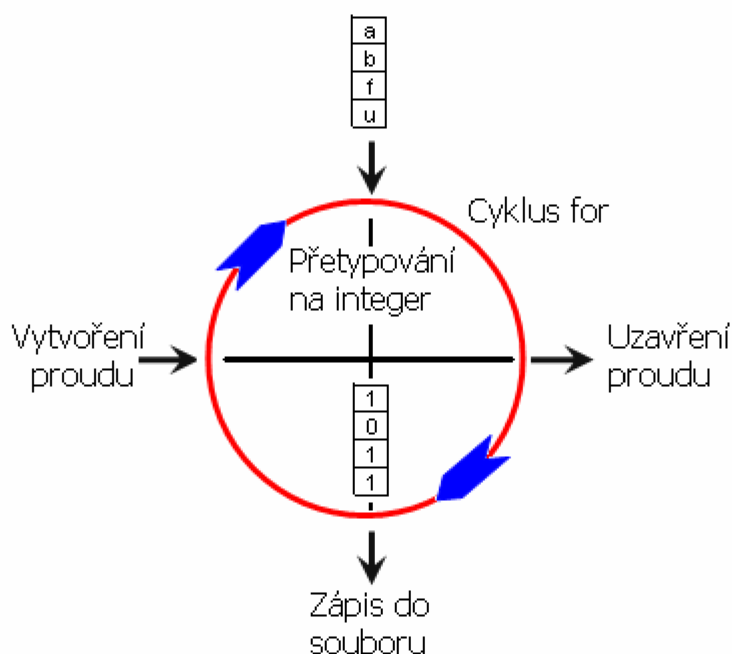
My budeme nejvíce pracovat se třídou **FileWriter**. Následující kód ukazuje vytvoření instance třídy **FileWriter** a zapsání jednoho znaku do souboru.

```
FileWriter fw = new FileWriter("soubor.txt");
fw.write((int)'b');
fw.close();
```

Podíváme-li se do souboru **soubor.txt,** bude jeho obsahem pouze písmeno b. Pokud byl již před zápisem nějaký text v souboru uložen, tak byl smazán. Budete-li chtít pouze připsat text na konec souboru, musíte použít jiný konstruktor třídy **FileWriter**.

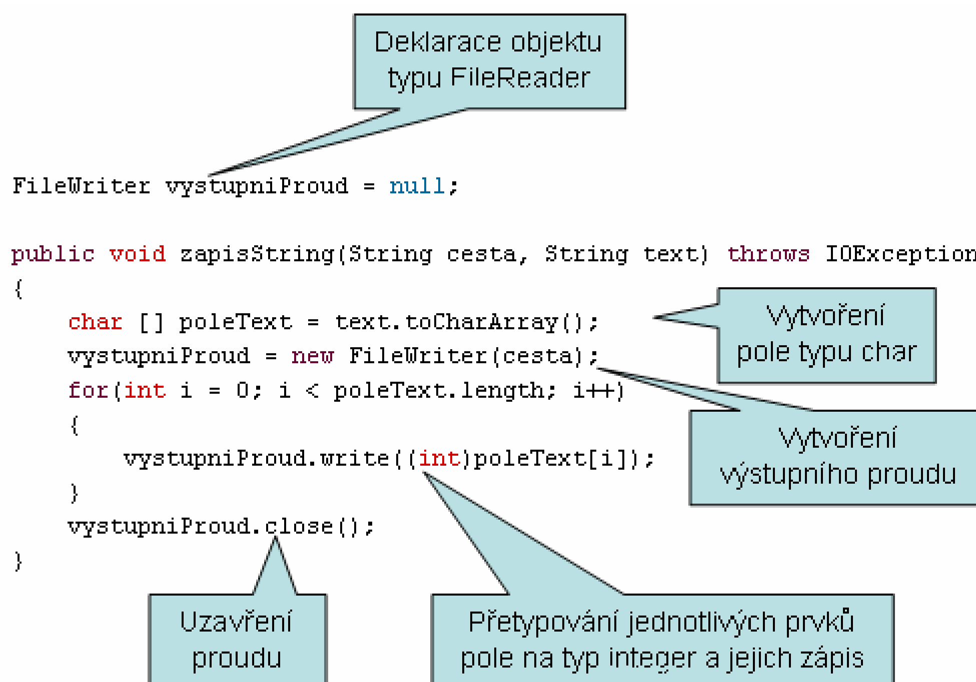
```
FileWriter fw = new FileWriter("soubor.txt", true);
```

Nejdůležitější metodou výstupních proudů je metoda **write**(), která zapisuje data do proudu. Vše si ukážeme na následujícím příkladu. Vytvoříme metodu, která převede zadaný **String** na pole typu **Char**. Pomocí cyklu for projde všechny prvky pole, přetypuje je na typ ***int*** a pomocí metody **write** je zapíše do souboru. Vše naznačuje následující obrázek.



Obrázek 25: Schéma práce s výstupním proudem

Následující kód zachycuje metodu **zapisString** s parametry typu ***String*** cesta a text. Parametr cesta představuje cestu a název souboru. Parametr text představuje text, který bude zapsán. Nejdříve dojde k vytvoření pole typu ***char*** a hned poté k vytvoření výstupního proudu. V těle cyklu for se vždy jeden prvek pole přetypuje na ***int*** a pomocí metody **write** se zapíše do souboru. Po ukončení cyklu for se uzavře proud.



Obrázek 26: Popis kódu pracující se výstupním proudem

10.2 Bufferování

Čtení (zápis) z (do) proudu znak po znaku je velice neefektivní. Java naštěstí nabízí další možnost čtení (zápisu), a to čtení (zápis) více znaků najednou. K tomu využívá objektu ***BufferedReader*** (***BufferedWriter***). Ten využívá vyrovnávací paměti **Bufferu**, kam si dočasně odkládá přečtená (zapisovaná) data. Použitím Bufferování se výrazně urychlí I/O operace.

10.2.1 Vytvoření objektu BufferReader

Následující kód ukazuje vytvoření objektu typu ***BufferedReader***. Jak vidíme, jako parametr se předává konstruktoru objekt typu ***FileReader***, který představuje znakový proud. Poslední řádek kódu přečte jeden řádek ze vstupního proudu.

```
FileReader fr = new FileReader("text.txt");
BufferedReader br = new BufferedReader(fr);
br.readLine();
```

Nejdůležitější metodou třídy ***BufferdReader*** je metoda **readLine(),** která vrací objekt typu ***String***, který představuje přečtený řádek. Je-li čtení u konce, vrátí tato metoda hodnotu ***null***.

10.2.2 Vytvoření objektu BufferedWriter

Vytvoření objektu ***BufferedWriter*** je velmi podobné vytvoření objektu třídy ***BufferedReader***. Jako parametr se tentokrát předává konstruktoru objekt typu ***FileWriter***. Pro zápis slouží metoda **write()**, která zapíše celý řádek.

```
FileWriter fw = new FileWriter("cil.txt");
BufferedWriter bw = new BufferedWriter( fw );
br.write("Jeden celý řádek.");
```

Další důležitou metodou třídy ***BufferedWriter*** je metoda **newLine()**, která vloží do proudu znak nového řádku.

10.3 Příklad 1 - Čeština

10.3.1 Úkol 1

Založte nový projekt a nazvěte ho **cestina**. Tento projekt bude obsahovat jedinou třídu **Menic**. Tato třída jedinou metodu typu ***void*** nazvanou **zamenIY** se dvěma parametry typu ***String***. První bude představovat soubor, jehož text má být změněn, druhý bude soubor, do kterého budou změny uloženy. Změny budou následující všechna měkká "i/y" se zamění za tvrdá a naopak. Zaměňovat se budu samozřejmě i dlouhá a velká "i/y".

10.3.2 Úkol 2

Jak je Vám asi jasné, předchozí příklad by neměl asi široké uplatnění, leda by jste chtěli přivést k šílenství učitelku češtiny. V následujícím kroku vytvoříme již smysluplnější metodu. Bude to metoda, která načte text, odstraní z něj všechny české

znaky a výsledek uloží do souboru. Metoda bude mít opět dva parametry, název zdrojového a cílového souboru.

10.4 Příklad 2 - šifrování

10.4.1 Úkol 1

Ve složce 10.4.1 se nachází projekt **Sifrování** z kapitoly zabývající se cykly. Tento projekt sloužil k zašifrování textu psaného bez diakritiky a bez velkých písmen. Rozšiřte projekt o metodu typu **String nactiText** s parametrem typu **String** cesta, který bude představovat cestu k souboru. Metoda bude vracet obsah textového souboru. Dále vytvořte beznávratovou metodu **ulozText** se dvěma parametry cesta a text typu **String**. Parametry budou představovat soubor, kam má být text uložen a text, který má být uložen. Na závěr vytvořte metodu **zasifruj** s parametry co a kam, představující zdrojový a cílový soubor.

10.4.2 Úkol 2

Na závěr tohoto cvičení vytvořte poslední metodu, nazvěte ji dešifruj, která načte textový soubor, dešifruje ho a uloží.

10.5 Příklad 3 - Bufferování

10.5.1 Úkol 1

Ze složky 10.5.1 si můžete otevřít projekt **Cteni**, který ukazuje čtení textu po znacích a jejich vypsání do terminálového okna. Projekt si stáhněte a vyzkoušejte. Čtení si můžete vyzkoušet na přiloženém souboru **dlouhyText.txt**.

10.5.2 Úkol 2

Třídu **Cteni** rozšiřte o metodu **cteniPoRadcich**, která bude využívat ke čtení objektu typu Buffer.

10.5.3 Úkol 3

Ze složky 10.5.3 si otevřete projekt **Sifrování**. Projekt rozšiřte o metodu **zasirfujBuff** a **desifrujBuff**, které budou pro načítání a ukládání textu používat Bufferované čtení a zápis. Vyzkoušejte rychlost na nějakém delším textu.

11 Tvorba projektu v BlueJNetBeans

Studium této kapitoly předpokládá základní znalost objektově orientovaného programování, znalost základních datových typů. Studiem se seznámíte s vývojovým prostředím NetBeans s modulem BlueJ. Dále se seznámíte se základy tvorby grafického uživatelského rozhraní.

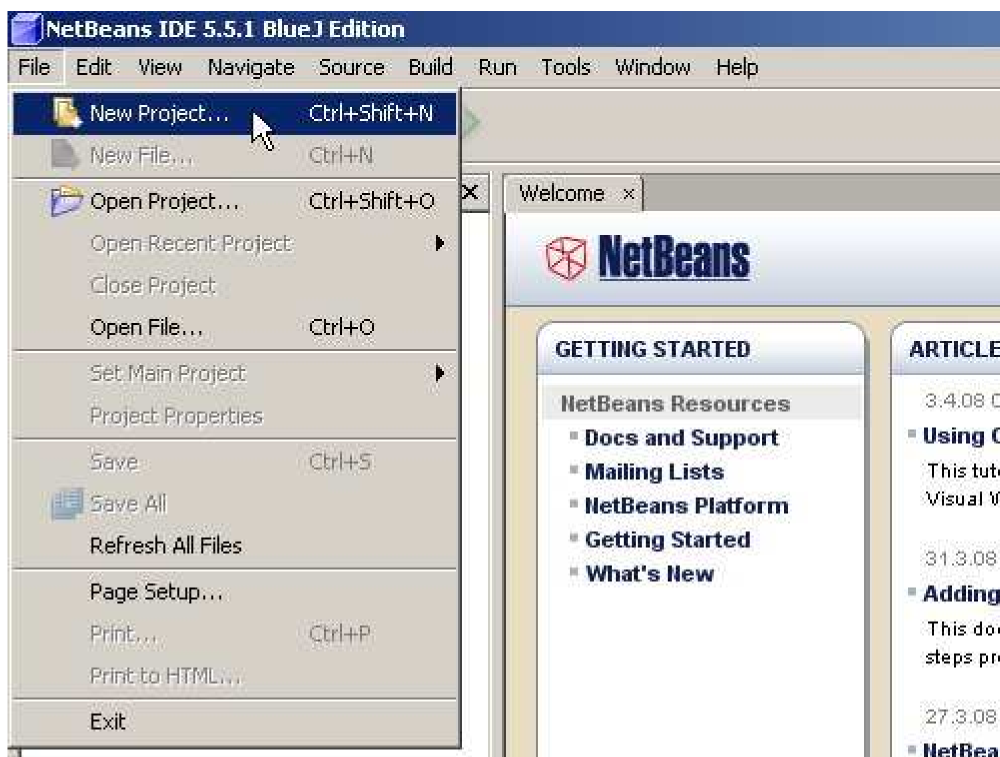
11.1 Převod měn

Cílem tohoto cvičení bude vytvořit jednoduchý program, který bude sloužit k převodu korun na dolary. Program již bude vyroben v prostředí NetBeansBlueJ a bude obsahovat grafické uživatelské rozhraní. To je zachyceno na obrázku vpravo. Po kliknutí na tento obrázek si můžete projekt uložit či spustit.

Obrázek 27: Ukázka programu

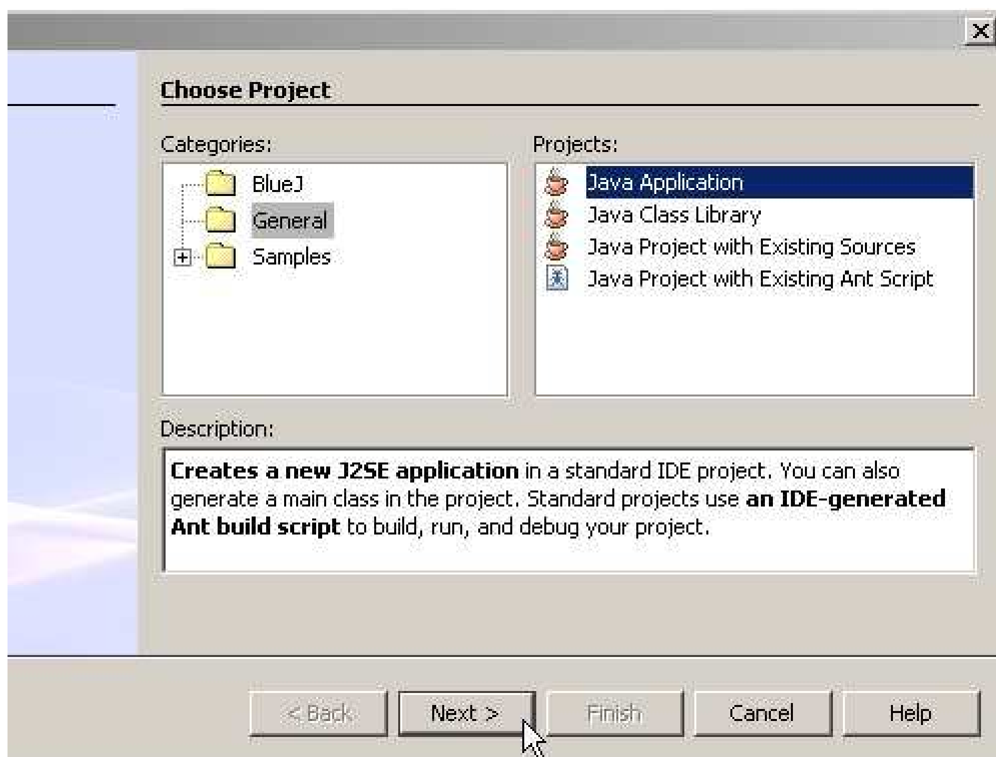
11.2 Založení projektu

Založení nového projektu začíná notoricky známým způsobem: File > New Project. Situaci znázorňuje obrázek níže.



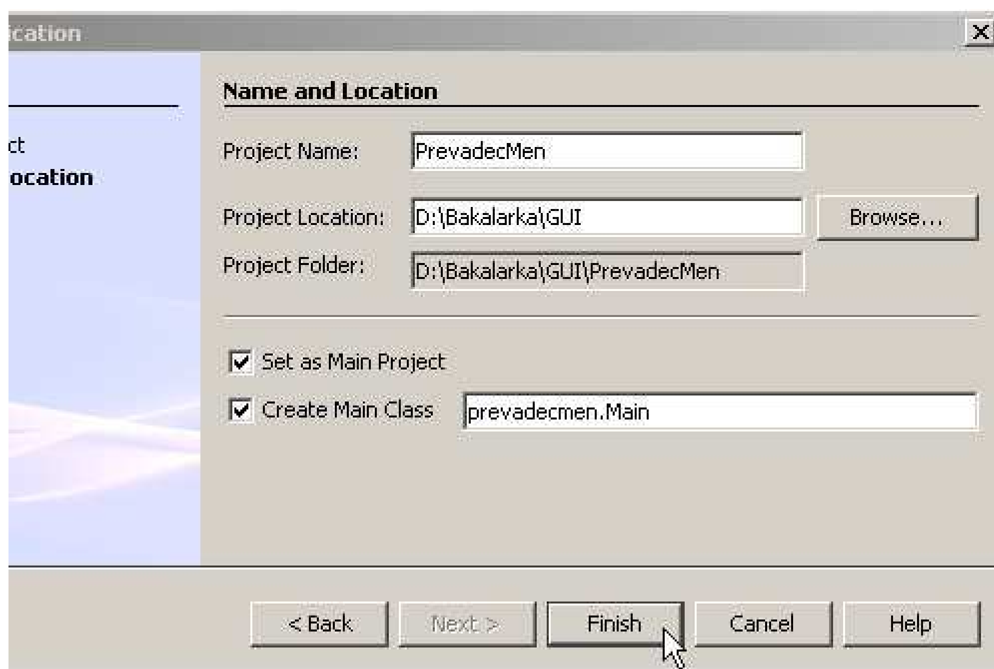
Obrázek 28: Založení projektu

Otevře se průvodce New Project. V prvním kroku vybereme jaký typ programu budeme vytvářet. My zvolíme kategorii General a projekt Java Application. Poté pokračujeme stiskem tlačítka Next.



Obrázek 29: Volba typu projektu

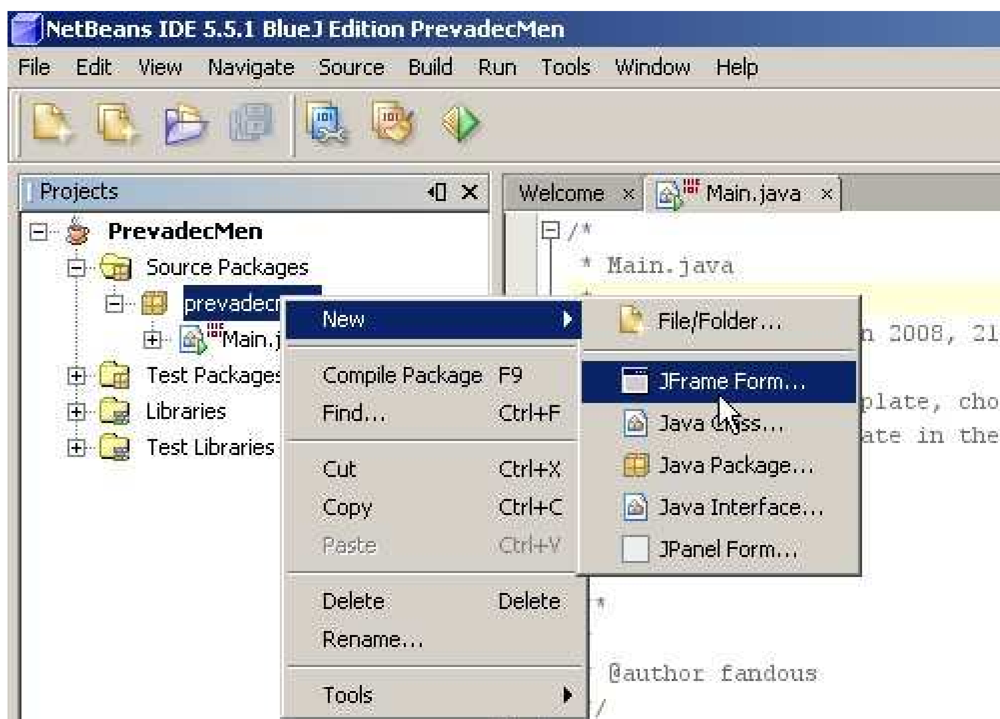
V dalším kroku zvolíme jméno a umístění našeho projektu. Check boxy Set as Main Project necháme zaškrtnutý. Stiskem tlačítka Finish dokončíme založení nového projektu.



Obrázek 30: Pojmenování a nastavení cesty

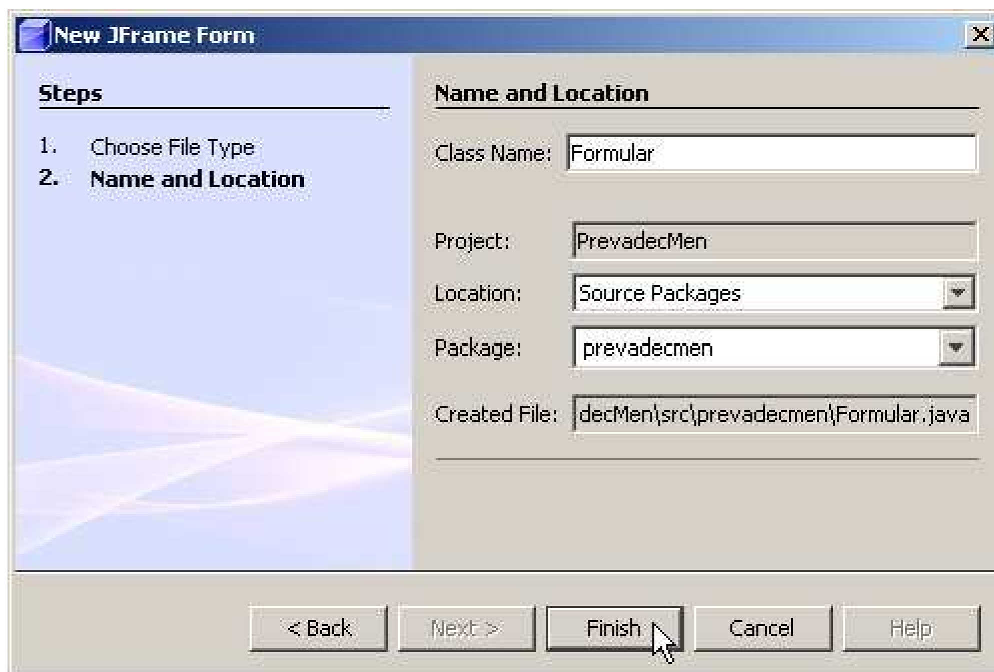
11.3 Vytvoření formuláře

Vytvoření formuláře, respektive třídy, která bude dědit od třídy JForm (budu - li dále mluvit o formuláři, budu mít na mysli právě onu třídu), provedeme následujícím způsobem. V panelu Projects rozbalte složku Source Packages a klikněte pravým tlačítkem na balíček prevadecmen. Zvolte New a následovně objekt, který chceme vytvořit, tedy JFrame Form. Vše znázorňuje následující obrázek.



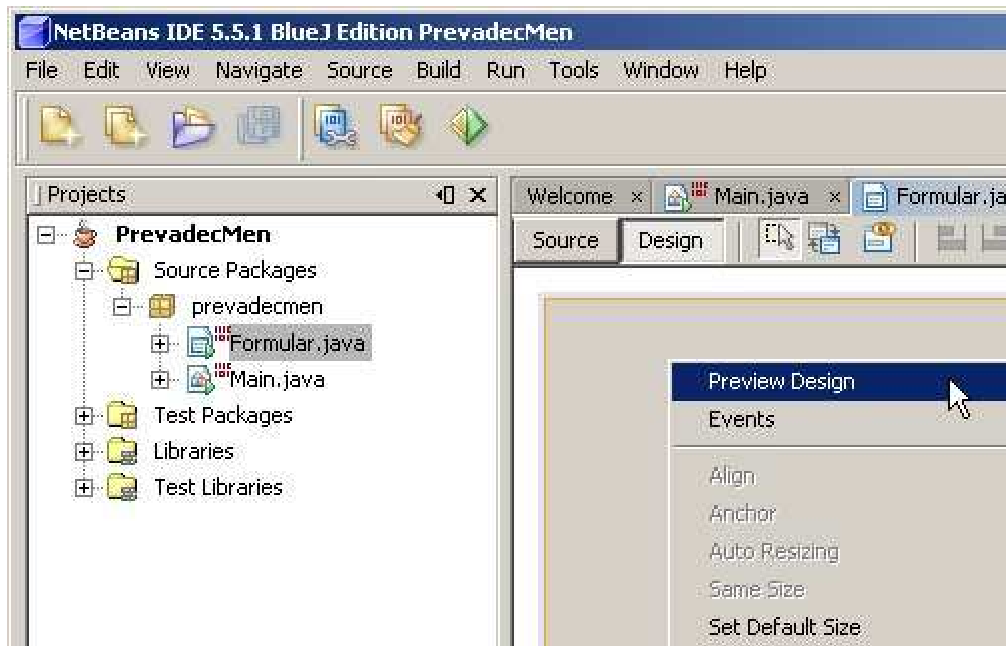
Obrázek 31: Vytvoření třídy dědící od JFrame

V průvodci New JFrame Form zvolíme název našeho formuláře. Dále zde můžeme změnit jeho umístění, my však ponecháme nastavené hodnoty. Tlačítkem Finish dokončíme vytváření formuláře. Vše znázorňuje následující obrázek.



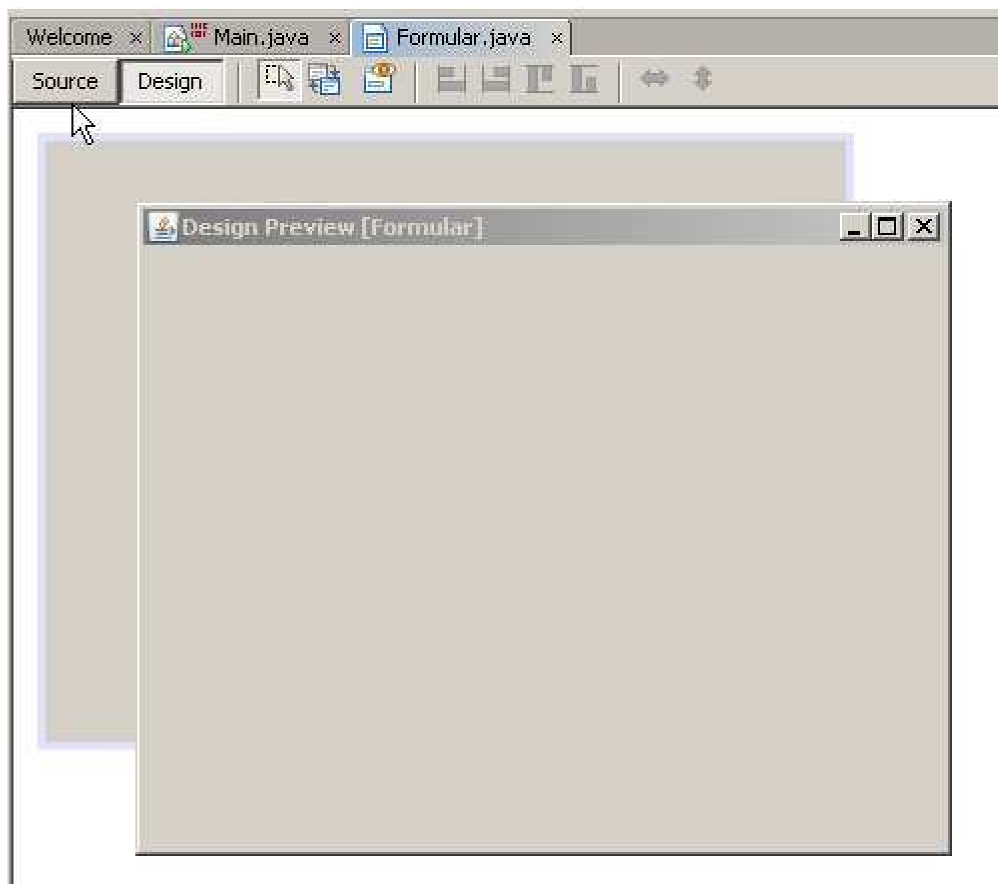
Obrázek 32: Pojmenování třídy

V hlavním okně vidíme návrh našeho formuláře. Ten si můžeme spustit a vyzkoušet. Stačí pravým tlačítkem kliknout na formulář a zvolit Preview Design.



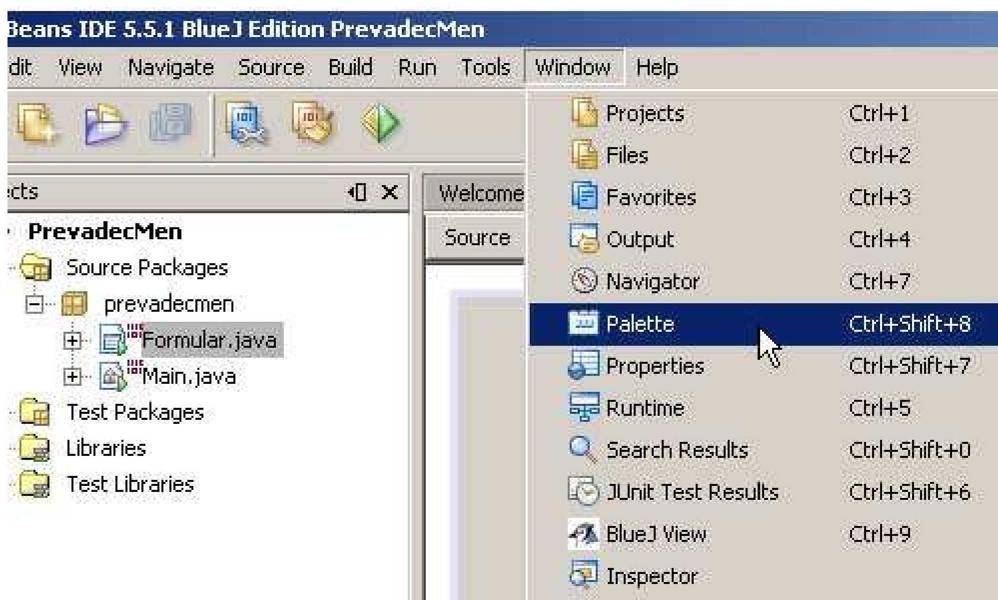
Obrázek 33: Zobrazení náhledu

Na následujícím snímku můžete vidět spuštěný formulář, se kterým může vývojář manipulovat a vyzkoušet si jeho grafickou funkčnost, hlavně co se týče jeho zvětšování, maximalizace, ...



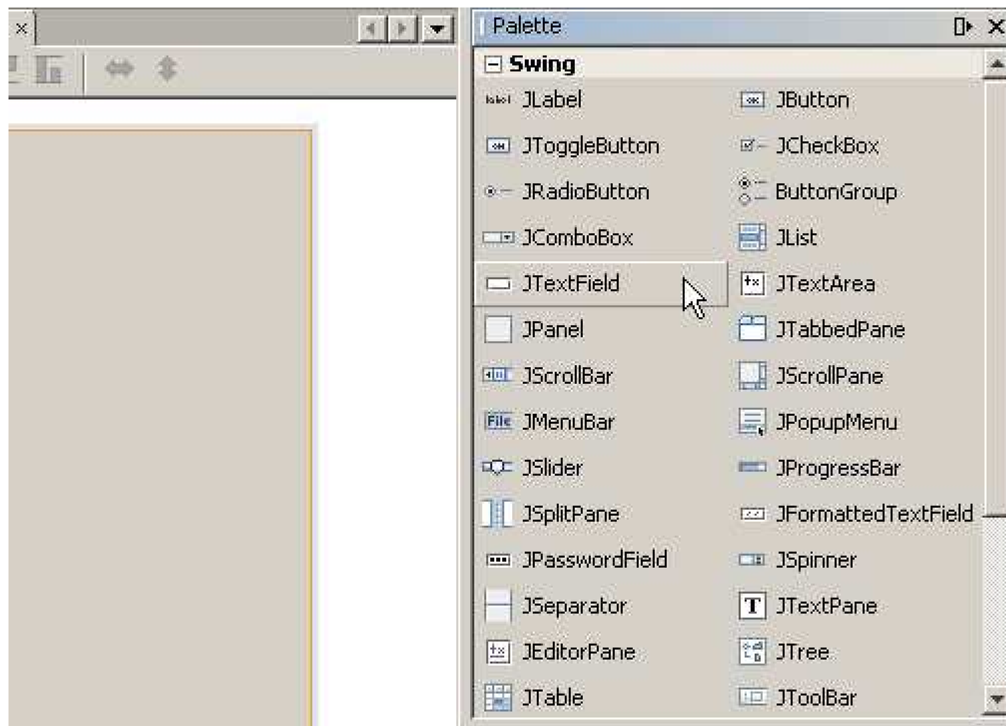
Obrázek 35: Náhled

V pravé části obrazovky bychom měli mít zobrazené panely Palette a Properties. Pokud tomu tak není, zobrazte si je výběrem Windows a příslušný panel.



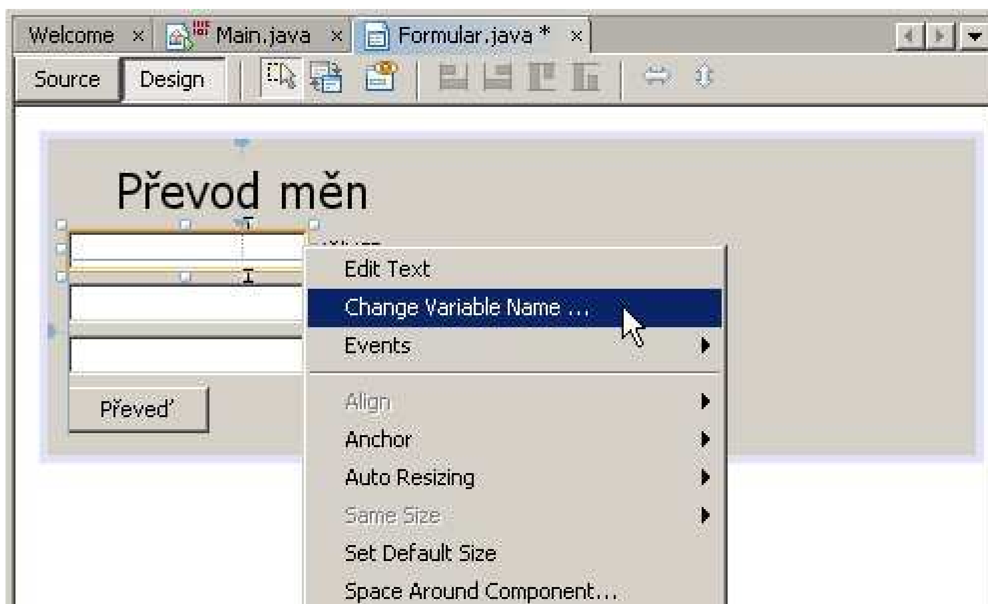
Obrázek 35 Zobrazení palety kontrolek

Na panelu Palette jsou umístěny grafické kompotenty, které můžete snadno přetáhnout na Váš formulář. S těmito komponentami můžete na panelu volně pohybovat, či měnit jejich velikost. Budeme-li chtít editovat nějaké další vlastnosti libovolné komponenty, označíme ji a danou vlastnost můžeme pozměnit v panelu Properties.



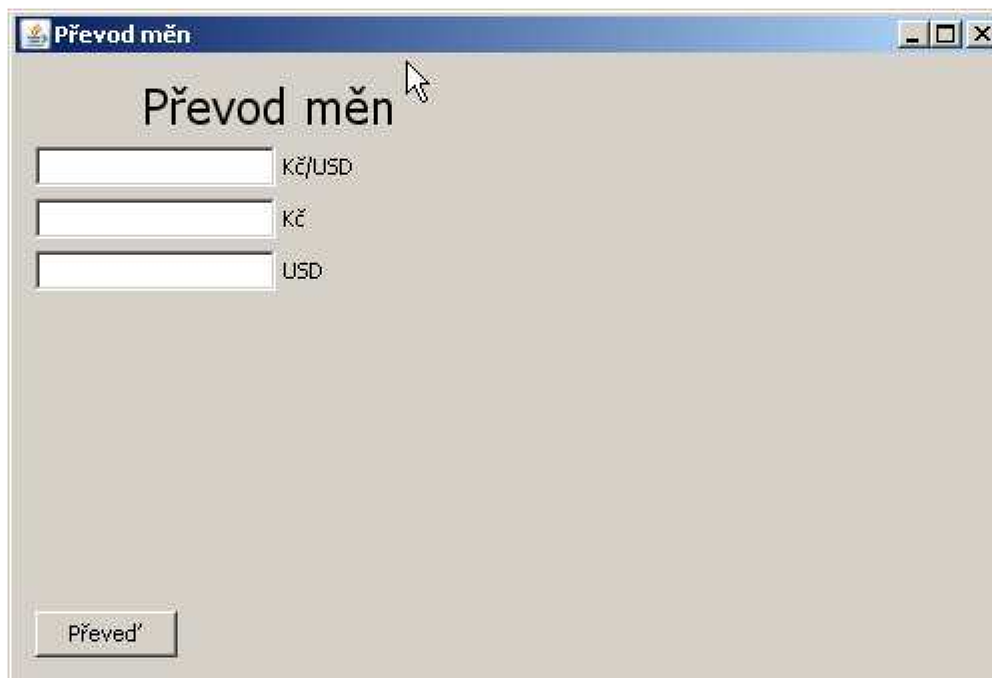
Obrázek 36: Paleta kontrolek

Další možností jak měnit vlastnosti komponenty je kliknout na ni pravým tlačítkem a vybrat příslušnou volbu. Velmi vhodné je změnit jméno komponenty. Při pozdější práci s komponentou v režimu zdrojového kódu nám označení jTextFieldKurz specifikuje komponentu lépe než standardní označení jTextField1. Samozřejmě systém pojmenování si můžete zvolit sami.



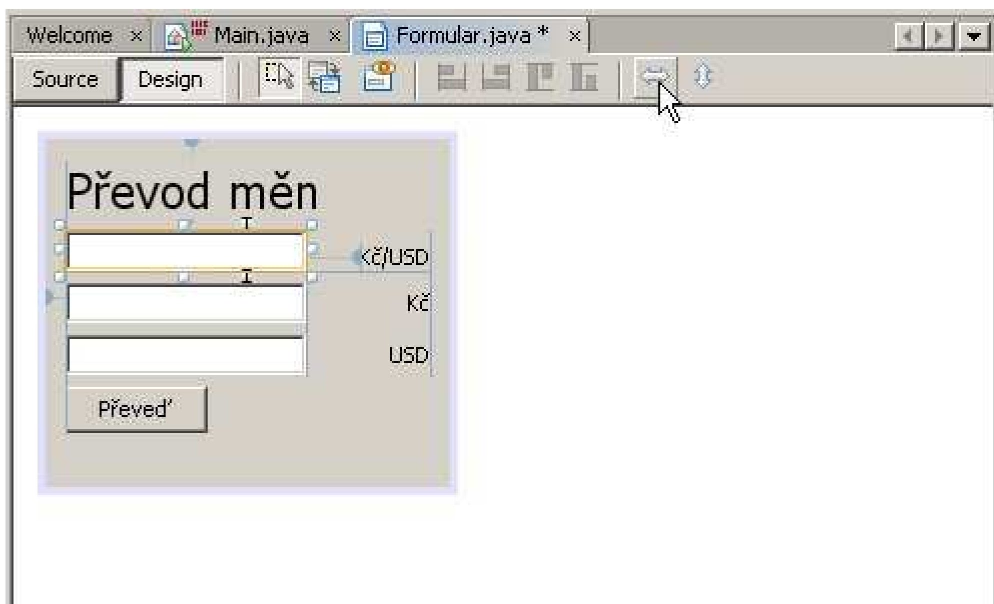
Obrázek 37: Změna jména kontrolky

Spustíte-li si nyní náhled formuláře, může se stát, že se Vám některé komponenty nebudou chovat tak, jak byste si přáli. Nejčastější chybou je asi to, že při zvětšení formuláře se nebudou proporciálně měnit jednotlivé komponenty.



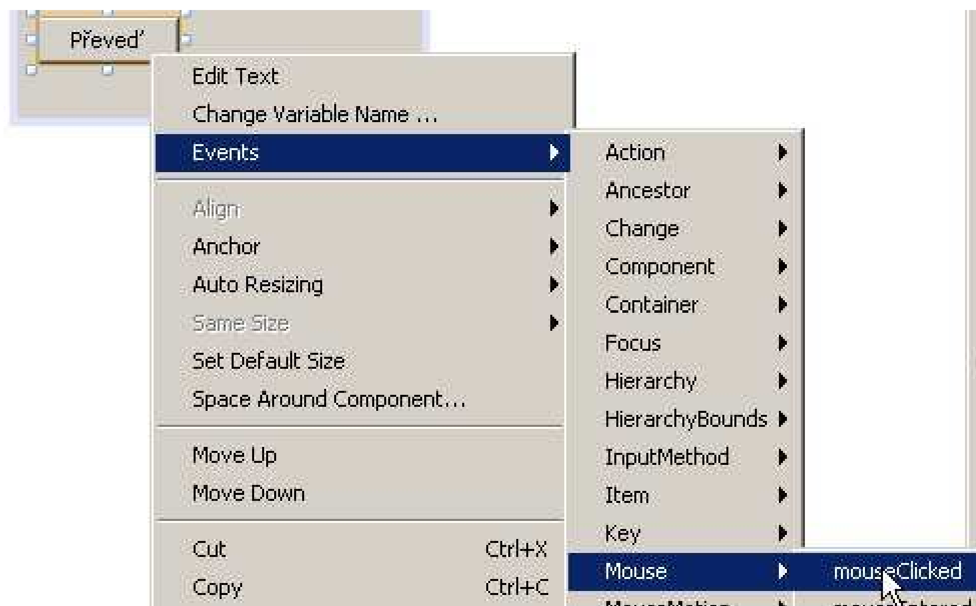
Obrázek 38: Náhled se špatným ukotvení kontrolek

Tuto vlastnost můžete změnit pomocí tlačítka change horizontal resizability. Vše ukazuje následující obrázek.



Obrázek 39: Změna velikosti formuláře a ukotvení

Teď již máme připravené naše grafické uživatelské rozhraní, takže se pustíme do úpravy kódu. Označte tlačítko Převeď, klikněte na něj pravým tlačítkem myši a vyberte možnost Events (Událost, kterou budeme chtít pro naše tlačítko zaregistrovat). Z podnabídky vyberte možnost mouseClicked (kliknutí myší).



Obrázek 40: Přidání události kontrolce

Poté se ocitnete v textovém režimu, a to přímo v těle metody, která se zavolá po kliknutí na tlačítko. Abychom mohli provést výpočet, musíme nejdříve získat hodnoty

z textových polí. Ty získáme voláním metody getText() nad instancí příslušného textového pole. Získaná hodnota však bude typu String, takže budeme muset danou hodnotu přetypovat na reálný typ. (Podrobněji se tímto problémem zabývá kapitola o základních datových typech). Poté provedeme standardním způsobem přepočet. Budeme-li chtít výslednou hodnotu zaokrouhlit, můžeme použít metodu round z knihovny Math. Na závěr budeme požadovat, aby se vypočtená hodnota zobrazila v třetím textovém poli. Toho docílíme metodou setText zvanou nad instancí příslušného textového pole. (Pokud si nejste jisti jaké metody můžete volat, či k jakým proměnným přistupovat, stačí stisknout kombinaci kláves Ctrl + Space a objeví se našeptávání s nabídkou a popisem metod).

```
 */
Generated Code

private void jButtonProvedMouseClicked(java.awt.event.MouseEvent evt) {
    double kurz = Double.valueOf(jTextFieldKurz.getText());
    double korun = Double.valueOf(jTextFieldKc.getText());
    double dolaru = Math.round(korun/kurz);
    jTextFieldUSD.setText(String.valueOf(dolaru));
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Formular().setVisible(true);
        }
```

Obrázek 41: Doplnění těla metody

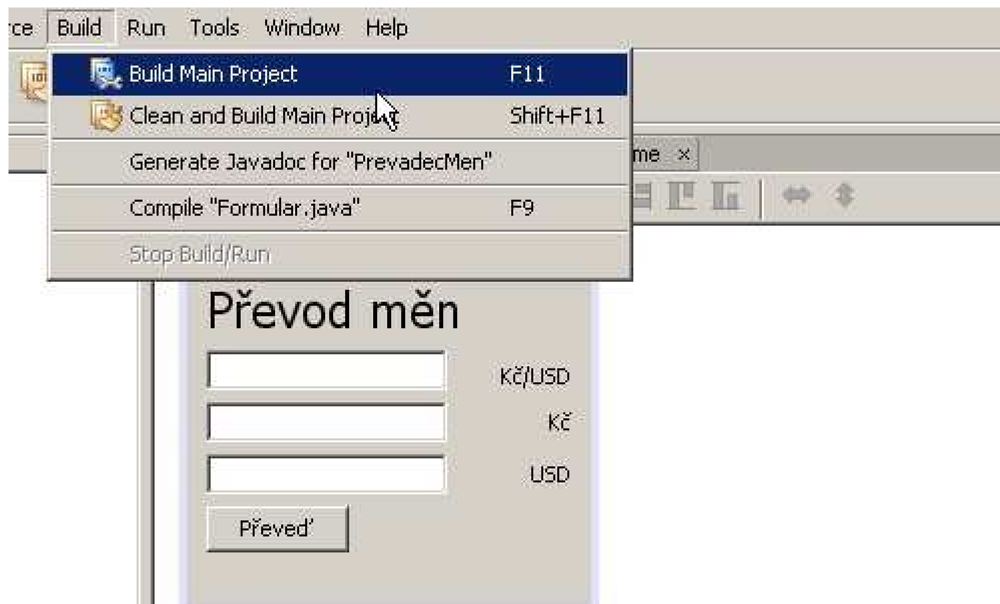
Na závěr práce se přepněte do statické třídy main. Tato třída obsahuje pouze jednu statickou metodu a to metodu main. Zde doplňte kód vytvářející instanci našeho formuláře. Nad instancí zavolejte metodu setVisible s parametrem true.

```
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    Formular f = new Formular();
    f.setVisible(true);
}
```

Obrázek 42: Metoda main

Nyní už pomocí tlačítek Run Main Project, nebo pomocí klávesy F6 můžete projekt spustit.

Na závěr v nabídce Build vyberte možnost Build Main Project. Tím se Vám v adresáři s Vaším projektem vytvoří složka dist a uvnitř spustitelný jar soubor Vašeho projektu.



Obrázek 43:Vytvoření spustitelného jar souboru

12 Autorizovaný přístup

Cílem této kapitoly je seznámit čtenáře s možností přístupu k datům zapouzdřeným uvnitř třídy pomocí metod. V těchto metodách mohou být na vstupní parametry kladeny další podmínky, které vyplývají z charakteru úlohy.

Autorizovaný přístup k datům je důsledkem zapouzdření, kdy zajistíme, aby s daty nebylo možné z vnějšku manipulovat jinak, než pomocí metod této třídy. Jak jste viděli v předchozích příkladech proměné mají před sebou klíčové slovo ***public***, které znamená, že tyto proměnné jsou přístupné i z vnějšku třídy. Nyní nahradíme toto klíčové slovo slovem ***private,*** čímž znemožníme přístup k proměnné z jiné třídy. Jak k nim ale potom přistupovat? Vytvoříme si pro získání či nastavení hodnoty proměnné metody. Obvykle názvy těchto metod začínají slovíčky get, set (getHodina, setMinuty). Následující příklad ukazuje ošetřený a neošetřený přístup k datům třídy **Hodinky**.

viz[3]

Kód nevyužívající autorizovaný přístup.

```
public class Hodinky
{
    public int hodin;
}
```

Kód využívající autorizovaný přístup.

```
public class Hodinky
{
    private int hodin;
    private int minut;

    public void setPocetHodin(int h)
    {
        if(0<=h && h<=24)
        {
            hodin = h;
        }
    }

    public int getPocetHodin()
    {
        return hodin;
    }
}
```

12.1 Příklad 1 - Hodinky

Vytvořte projekt **Hodinky** s jedinou třídou Hodinky. Tato třída bude obsahovat atributy hodin a minut. Zajistěte aby do proměnných mohly být ukládány jen hodnoty vystihující reálnou situaci. Projekt otestujte, pokuste se vložit nereálnou hodnotu.

12.2 Příklad 2 – Zaměstnanec

12.2.1 Úkol 1

Ze složky 12.2.1 si můžete otevřít projekt **AutorizovanyPristup**. Tento projekt obsahuje dvě třídy **Zamestnanec** a **GUI**.

Třída **Zamestnanec** představuje zaměstnance školy. Každý zaměstnanec bude mít tři atributy, **jmeno**, **prijmeni** a **pocetHodin**. Atribut **pocetHodin** bude představovat počet hodin odpracovaných zaměstnancem. Dále obsahuje přetíženu metodu **toString**().

Třída **GUI** bude představovat formulář, který slouží k zadávání údajů o zaměstnanci. Uvnitř třídy **GUI** je také vytvořena instance třídy **ArrayList** nazvaná **seznamZamestnancu**, která slouží k uchovávání instancí typu **Zamestnanec**.

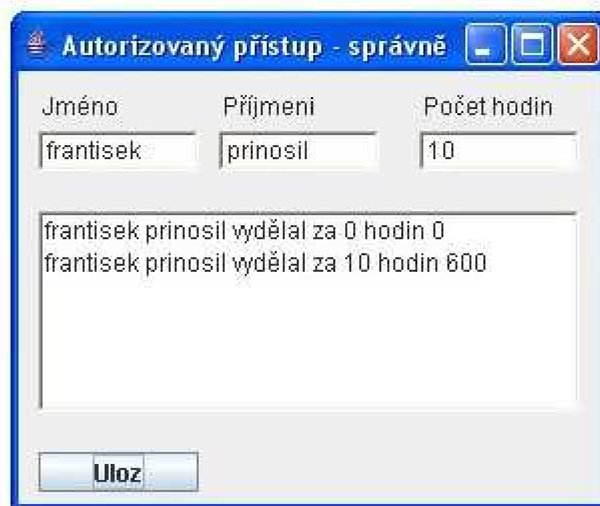
Obrázek 44: Ukázka programu

Jak vidíte z obrázku, hodnoty, které jsem zadal do textových polí, nejsou zrovna nejvhodnější. Počet odpracovaných hodin je záporný, jméno a příjmení začíná malým

písmenem. Z hlediska kódu je ale vše v pořádku **jmeno** a **prijmeni** jsou proměnné typu **String** a **pocetHodin** proměnná typu ***int***.

12.2.2 Úkol 2

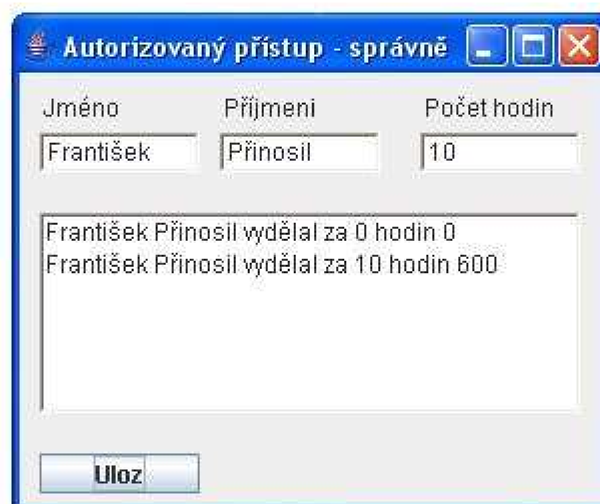
Pomocí autorizovaného způsobu zajistěte, aby do proměnné **pocetHodin** mohla být zadávaná pouze nezáporná celá čísla.



Obrázek 45: Ukázka programu

12.2.3 Úkol 3

Pomocí autorizovaného způsobu zajistěte, aby do proměnných **jmeno**, **prijmeni** mohly být zadávány pouze hodnoty začínající velkým písmenem.



Obrázek 46: Ukázka programu

13 Dědičnost

Dalším základním pilířem objektově orientovaného programování je dědičnost. Samotný název napovídá, že se bude jednat o přenos vlastností z rodiče na potomka. Rodičem rozumíme třídu, z níž budou atributy a metody přeneseny na jinou třídu. Rodiče označujeme rodičovská třída, bázová třída, nebo supertřída. Potomka označujeme jako odvozenou třídu, zděděnou třídu nebo *subtřídu*. Základní využití je zřejmé, dědění použijeme tehdy, budeme-li chtít vytvořit třídu, která je rozšířením jiné existující třídy. Dědičnost představuje možnost přidat k existující třídě další atributy či metody a vytvořit tak třídu odvozenou. Všechny třídy mají společného předka a tím je třída **Object**.

Dědění realizujeme prostřednictvím klíčového slova ***extends***. Toto slovo v překladu znamená „rozšiřuje“.

viz[3]

Ukázka deklarace třídy **Ucitel**, která je odvozena od třídy **Zamestnanec**.

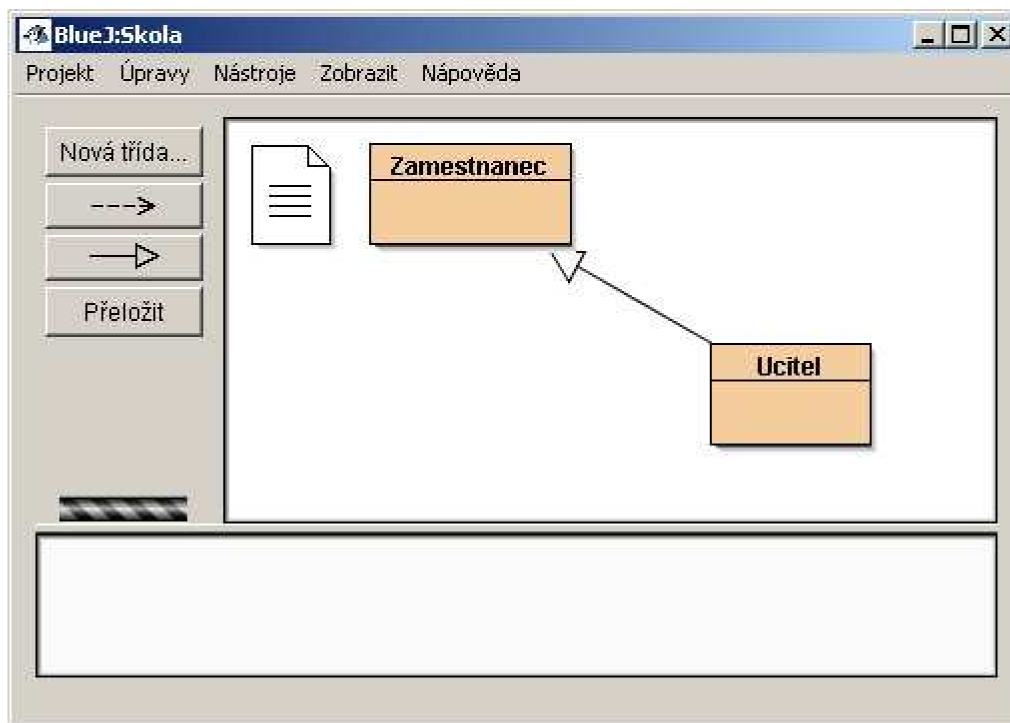
```
public class Ucitel extends Zamestnanec
{

}
```

13.1 Konstruktory a dědičnost

Studium této kapitoly předpokládá základní znalosti objektově orientovaného programování a znalost základních datových typů. Studiem této kapitoly se seznámíte s možností rozšířit existující třídu o nové parametry a vytvořit tak třídu novou.

13.1.1 Konstruktor bez parametru v předkovi i potomkovi.

Vytvořte si v prostředí BlueJ nový projekt a nazvěte jej **Skola**. Dále vytvořte prázdné třídy **Zamestnanec** a **Ucitel**. Třída zaměstnanec bude obsahovat dva veřejné atributy typu **String jmeno** a **prijmeni**. Obě třídy budou obsahovat konstruktory bez parametru(ne implicitni). Učitel je zaměstnancem, budeme tedy chtít, aby třída **Ucitel** byla třídou odvozenou od třídy **Zamestnanec**. V deklaraci třídy **Ucitel** doplňte za název třídy ***extends Zamestnanec***. Třídu přeložte.



Obrázek 47: Znázornění závislosti

V prostředí BlueJ se mezi třídami objevila šipka, která naznačuje vztah tříd. Vytvoříte-li si instanci třídy **Ucitel** a prohlédnete si ji inspektorem zjistíte, že instance má dva atributy **jmeno** a **prijmeni**, která zdědila od třídy **Zamestnanec**.

Obrázek 48: Prohlížení instance

13.1.2 Konstruktor bez parametru v předkovi a konstruktor s parametrem v potomkovi

Tato varianta je možná, i když pro tento konkrétní příklad není moc smysluplná. Ve třídě **Ucitel** vytvořte veřejný atribut typu **String** aprobace. Dále konstruktor s parametrem aprobace.

```
public class Ucitel extends Zamestnanec
{
    public String aprobace;
    public Ucitel(String aprobace)
    {
        this.aprobace = aprobace;
    }
}
```

Po přeložení a vytvoření instance třídy **Ucitel** nás asi nepřekvapí, že obsahuje parametry **jmeno**, **prijmeni** a **aprobace**.

13.1.3 Předek mající konstruktor s parametrem.

Jakým způsobem by se změnil předcházející příklad, pokud bychom chtěli získat třídu **Ucitel** rozšířením třídy **Zamestnanec**, která by používala konstruktor s parametry. Vytvořte ve třídě **Zamestnanec** konstruktor s parametry typu **String jmeno** a **prijmeni**, které se budou v jeho těle inicializovat.

```
public Zamestnanec(String jmeno, String prijmeni)
{
    this.jmeno = jmeno;
    this.prijmeni = prijmeni;
}
```

Pokud se nyní pokusíte o překlad, obdržíte chybu „Cannot find symbol – constructor Zaměstnanec()". Dědí se proměnné instance a metody, konstruktor se nedědí, je pouze využíván. Proto se v konstruktoru dceřinné třídy musí zavolat konstruktor rodičovské třídy. To se provádí pomocí klíčového slova super (s výjimkou konstruktoru bez parametru, viz předcházející příklad). Upravený konstruktor třídy **Ucitel** by mohl vypadat třeba následovně.

```
public Ucitel(String jmeno, String prijmeni ,String
aprobace)
{
```

```
    super(jmeno, prijmeni);
    this.aprobace = aprobace;
}
```

Nyní již proběhne překlad v pořádku.

13.1.4 Implicitní konstruktor

Jak by se změnila situace při používání implicitního konstruktoru v předkovi? Bude-li použit v předkovi implicitní konstruktor, může být v potomku použit implicitní konstruktor, konstruktor bez parametrů nebo konstruktor s parametrem. Bude-li však konstruktor rodičovské třídy s parametrem a my budeme chtít v potomku použít implicitní konstruktor, překladač ohlásí chybu: Cannot find symbol – constructor Zaměstnanec().

13.1.5 Více než jeden konstruktor

Jak již víme, bude-li v rodiči uveden konstruktor s parametrem a v potomku konstruktor bez parametru, obdržíme výjimku. Toto je známá věc již z předcházejícího příkladu. Existuje však ještě jedna možnost, jak této situaci předejít. A to umístit do rodičovské třídy ještě jeden konstruktor bez parametrů. Jelikož konstruktor subtřídy očekává překladač také konstruktor bez parametru v nadtřídě, ten tam ale je. Překlad proběhne v pořádku a my můžeme spustit program.

```
public class Zamestnanec
{
    public String jmeno;
    public String prijmeni;
    public Zamestnanec()
    {
    }

    public Zamestnanec(String jmeno, String prijmeni)
    {
        this.jmeno = jmeno;
        this.prijmeni = prijmeni;
    }
}

public class Ucitel extends Zamestnanec
{
```

```
    public String aprobace;

    public Ucitel()
    {
    }
}
```

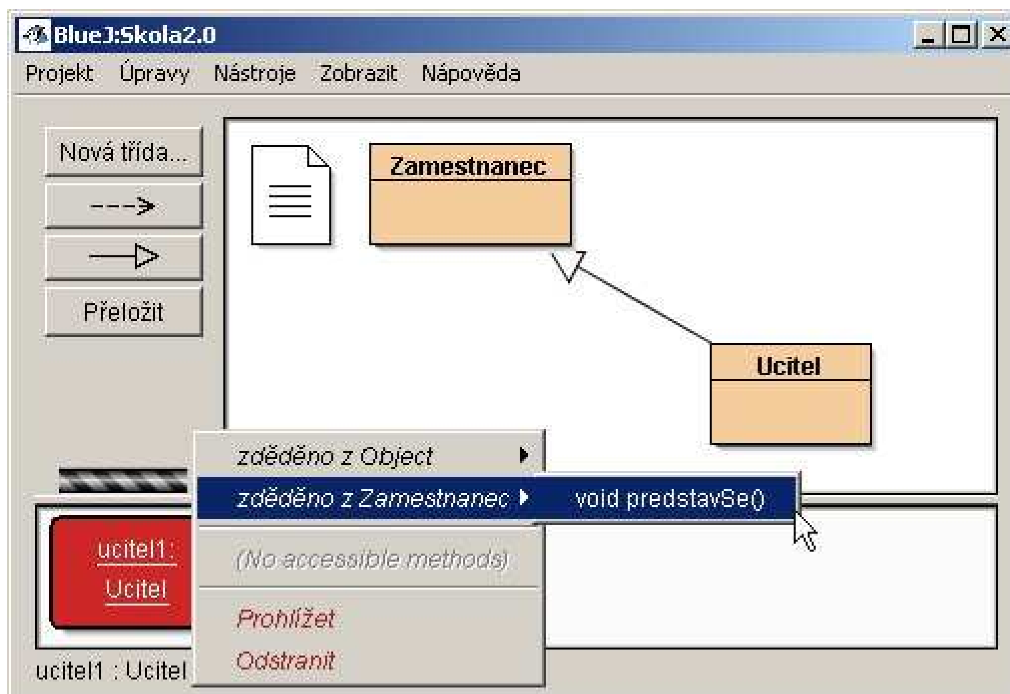
13.2 Dědičnost - metody

Pro demonstraci dědění metod použijeme projekt z předcházející kapitoly zabývající se konstruktory.

Vytvořte ve třídě **Zamestnanec** metodu bez parametru typu **String predstavSe()**, která vrátí řetězec obsahující jméno a příjmení učitele.

```
public String predstavSe()
{
    return jmeno + " " + prijmeni;
}
```

Vytvoříte instanci třídy **Ucitel**. Kliknete-li pravým tlačítkem myši na objekt instance, zobrazí se Vám nabídka metod. Tato nabídka bude také obsahovat metody zděděné od třídy **Zamestnanec** a to sice pouze jednu metodu, metodu **predstavSe()**.



Obrázek 49: Volání zděděné metody

Budeme-li chtít, aby se učitel představoval i svojí aprobací, budeme muset ve třídě **Ucitel** znovu nadefinovat metodu **predstavSe** - překrýt ji.

```
public String predstavSe()
{
   return jmeno + " " + prijmeni + " " + aprobace;
}
```

Druhý a mnohem více zajímavý způsob jak provést překrytí metody **predstavSe,** je v těle této metody využít metodu p**redstavS**e z nadtřídy. Metodu z nadtřídy zavoláme obdobně jako konstruktor pomocí klíčového slovíčka super.

```
public String predstavSe()
{
    return super.predstavSe() + " " + aprobace;
}
```

Tento způsob je zvláště u delších a složitějších metod velmi výhodný.

13.3 Dědičnost - abstraktní třídy a metody

V této kapitole budeme dále využívat projekt z předcházející kapitoly. Představte si, že naše škola bude mít více zaměstnanců, kteří budou odvozeni od třídy **Zamestnanec**. Budeme požadovat, aby autor každé další třídy, která je potomkem třídy **Zamestnanec**, byl nucen překrýt metodu **predstavSe** a tím se zároveň zamyslet nad tím, co by měl o sobě příslušný zaměstnanec sdělovat. Tuto možnost programovací jazyk Java nabízí v podobě abstraktních metod a abstraktních tříd. Abstraktní metodu, třídu uvodíme klíčovým slovem abstract. Abstraktní metoda může být obsažena pouze v abstraktní třídě, nemůže mít žádné tělo, ale ve všech třídách odvozených od abstraktní třídy musí být abstraktní metody překryty. Abstraktní třída nemůže vytvářet instance.

Označte třídu **Zamestnanec** jako abstraktní a metodu **predstavSe()** také. Dále odstraňte tělo metody **predstavSe()**. Poslední úpravu proveďte ve třídě **Ucitel** v metodě **predstavSe()**. Upravte ji tak, aby nevolala metodu z nadtřídy. Nyní projekt můžete bezproblémů zkompilovat.

```
public abstract class Zamestnanec
{
    public String jmeno;
    public String prijmeni;

    public Zamestnanec(String jmeno, String prijmeni)
    {
        this.jmeno = jmeno;
        this.prijmeni = prijmeni;
    }

    public abstract String predstavSe();
}
```

Pokud byste odstranili nebo zakomentovali metodu představ ve třídě **Ucitel**, obdrželi byste při kompilaci chybu Ucitel is not abstract and does not override abstract methot predstavSe() Zamestnanec.

13.4 Dědičnost - finální metody

Pravým opakem abstraktních metod, které bylo nutno v potomkovi překrýt, jsou finální metody, které překryty být nesmějí. Využití mohou najít v takových případech, kdy autor rodičovské třídy napíše metodu, u které si přeje, aby nebyla překryta.

Vše si zase předvedeme na příkladu naší školy. Žádná z našich tříd není tentokrát definována abstract a neobsahuje abstraktní metody. Budeme-li chtít, aby se všichni zaměstnanci představovali jednotně a to pouze jménem a příjmením, uvodíme metodu predstavSe ve třídě **Zamestnanec** klíčovým slovem ***final***.

```
public final String predstavSe()
{
   return jmeno + " " + prijmeni;
}
```

Pokusíme-li se nyní o kompilaci, neproběhne v pořádku. Kompilátor ohlásí chybu **predstavSe()** in Ucitel cannot override predstavSe in Zamestnanec; overridden methot is final.

Pokud metodu **predstavSe** ve třídě **Ucitel** odstraníme nebo zakomentujeme, překlad již proběhne v pořádku.

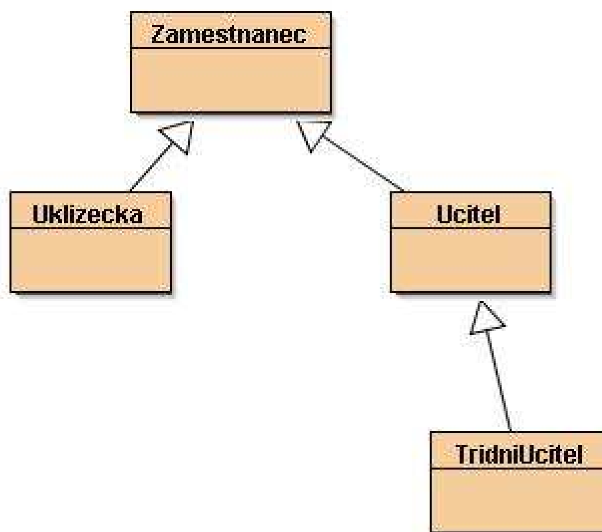
13.5 Příklad 1 – Dědění(konstruktory)

13.5.1 Úkol 1

Ze složky 13.5.1 si můžete otevřít projekt Skola, který už dobře znáte z ukázkového příkladu. Příklad obsahuje třídu **Zamestnanec** s atributy **jmeno** a **prijmeni**, konstruktor s parametry, v jehož těle si tyto atributy inicializují. Dále obsahuje třídu **Ucitel** odvozenou od třídy **Zamestnanec**, která tuto třídu rozšiřuje o atribut aprobace. Projekt si stáhněte a otevřete v prostředí BlueJ.

13.5.2 Úkol 2

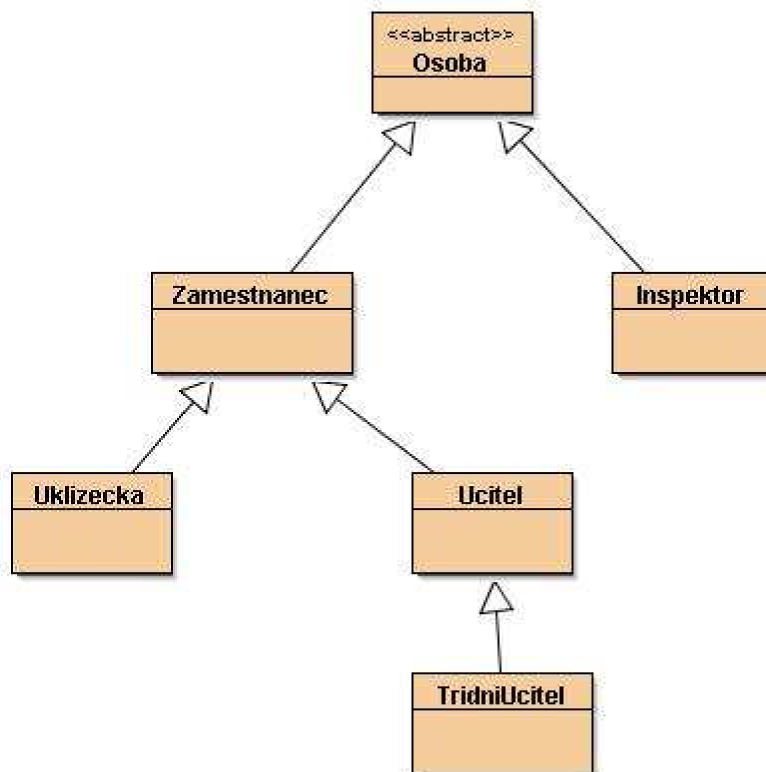
Projekt rozšiřte o třídy **TridniUcitel** a **Uklizecka** tak, aby **TridniUcitel** rozšiřoval třídu **Ucitel** o atribut **trida** (název třídy, ve které je třídním) a třída **Uklizecka** třídu **Zamestnanec** o atribut **patro** (číslo patra, které má na starosti). Třídy budou mít konstruktory, ve kterých se budou všechny atributy inicializovat.



Obrázek 50: Schéma závislosti tříd

13.5.3 Úkol 3

Rozšiřte projekt o abstraktní třídu Osoba s atributy **jmeno** a **prijmeni**. **Trida** Osoba bude mít konstruktor s parametry **jmeno** a **prijmeni**, v jehož těle se budou tyto parametry inicializovat. Dále vytvořte třídu Inspektor. Upravte třídy **Zamestnanec** a Inspektor, tak aby mohly rozšiřovaly třídu Osoba. Upravte konstruktory třídy **Zamestnanec** a Inspektor tak, aby překlad projektu proběhl v pořádku.



Obrázek 51: Schéma závislosti tříd

13.6 Příklad 2 – dědění(metody)

13.6.1 Úkol 1

V tomto příkladu budeme používat poslední verzi projektu Skola z příkladu o konstruktorech. Tento projekt si můžete otevřít ze složky 13.6.1. Projekt si otevřete v prostředí BlueJ. Vytvořte instance tříd **Uklizecka**, **TridniUcitel**, **Inspektor** a pokuste se pro ně zavolat metodu **predstavSe().**

13.6.2 Úkol 2

Ve třídách **Uklizecka** a **TridniUcitel** překryjte metodu **predstavSe**, tak aby instance těchto tříd podávaly o sobě co nejvíce informací. Využijte variaci s klíčovým slovem super.

13.6.3 Úkol 3

Ve třídě Osoba vytvořte abstraktní metodu typu ***int*** **plat**(). Upravte ostatní třídy tak, aby překlad projektu proběhl bez chyb. Pokud budete v ostatních třídách definovat metodu plat, nezalamujte se nikterak rozsáhlou definicí jejího těla.

13.6.4 Úkol 4

Třídu **Zamestnanec** rozšiřte o metodu typu ***int***, která bude vracet id zaměstnance. Id generujte jako hodnotu hash codu příjmení zaměstnance. (int id = prijmeni.hashCode()). Zajistěte, aby systém tvorby id byl nutně stejný pro všechny potomky této třídy. Výsledek ověřte tím, že se pokusíte metodu id() v rozšiřujících třídách překrýt.

14 Polymorfizmus

Pro studium této kapitoly se předpokládají hlubší znalosti objektově orientovaného programování(dědičnost), dále znalost práce se základními datovými typy. Studiem získáte základní představu o pojmu polymorfizmus a o jeho využití v objektově orientovaném programování.

Český význam slova polymorfizmus je mnohotvárnost, různorodost. Význam tohoto slova z hlediska programovacího jazyka Java je trochu specifičtější. Pojem polymorfizmu jako mnohotvárnosti je zde vnímán jako možnost uložit do proměnné jednoho datového typu proměnnou jiného datového typu, přičemž však zůstanou zachovány její charakteristické vlastnosti(definice metody). Nejde to ale vždy, podmínkou je, že vložená proměnná musí být typu, který dědí od typu proměnné, do které objekt ukládáme. Vše si ukážeme na praktickém příkladě.

viz[3]

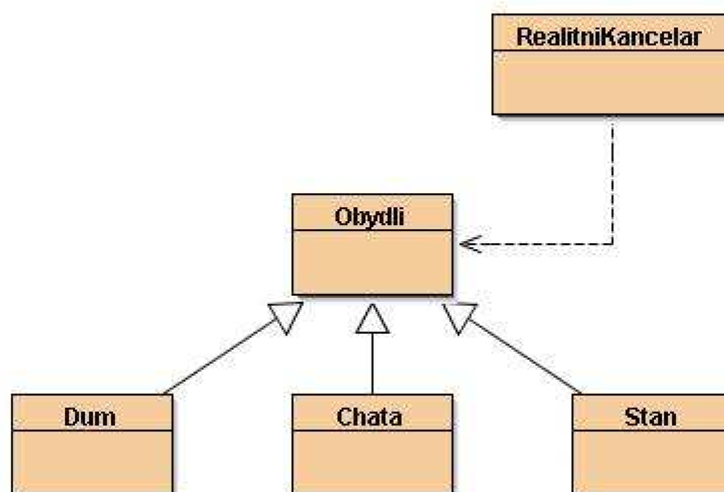
Založte si v prostředí BlueJ nový projekt a nazvěte ho obydli. Vytvořte prázdné třídy bez konstruktoru **Obydli**, **Dum**, **Chata**, **Stan** tak, aby třídy **Dum**, **Chata** a **Stan** rozšiřovaly třídu **Obydli**. Dále založte prázdnou třídu **RealitniKancelar**.

Třída realitní kancelář bude obsahovat atribut **seznamRealit** typu pole, které bude uchovávat instance třídy **Obydli**. Dále atribut **pocet**, který bude reprezentovat počet objektů uložených v poli. Dále metodu na vkládání realit a metodu na vypsání realit.

```
public class RealitniKancelar
{
    public Obydli[] seznamRealit = new Obydli[10];
    public int pocet = 0;
    public void vlozRealitu(Obydli o)
    {
        seznamRealit[pocet] = o;
        pocet ++;
    }

    public void predstavReality()
    {
        for(int i = 0; i < pocet; i ++)
        {
            seznamRealit[i].predstavSe();
        }
    }
}
```

Ve třídě **Obydli** vytvořte metodu **predstavSe**, který vypíše do terminálového okna informaci o sobě. Tuto metodu vhodným způsobem překryjte ve třídách Dum, Chata, Stan.

```
public void predstavSe()
{
   System.out.println("Ja jsem obydli.");
}
```



Obrázek 52: Náhled uspořádání tříd

Nyní již máme vše připraveno a můžeme se pustit do demonstrace. Vytvořte instance všech tříd. Do instance třídy **RealitiniKancelar** uložte instanci třídy **Obydli**. Vše proběhlo v pořádku, pole **seznamRealit** v sobě přece má uchovávat objekty typu **Obydli**. Mnohem zajímavější je to, že stejným způsobem můžeme uložit i objekty typu **Dum**, **Chata** i **Stan**. Jak je to možné? Došlo totiž k implicitnímu přetypování jednotlivých instancí na obecnější typ, tedy na typ **Obydli**. Více se o tomto problému můžete dozvědět v kapitole o datových typech. Jedná se o rozšiřující konverzi. Například:

```
        int prirozeneCislo = 10;
        double realneCislo = prirozeneCislo;
```

Ještě zajímavější výsledek dostanete, pokud nyní po vložení všech vytvořených instancí do pole **seznamRealit** zavoláte metodu **predstavReality**(). Uložené objekty byly přetypovány na objekty typu **Obydli**, takže bychom mohli očekávat odpověď, že všechny uložené reality jsou **Obydli**. Ale pozor, obdržíme výpis odpovídající původním typům objektů. Toto chování je důsledkem toho, že Java podporuje

dynamické volání metod (někdy také označováno, jako pozdní vazba). To znamená, že adresa volané metody nemusí být známa již při překladu, ale vybírá se až při zavolání metody za běhu programu v závislosti na typu proměnné.

14.1 Příklad – Škola(polymorfizmus)

14.1.1 Úkol 1

Ze složky 14.1.1 projekt **Skola** z konce minulé kapitoly, projekt otevřete v prostředí BlueJ a seznamte se s ním.

14.1.2 Úkol 2

Založte novou třídu s názvem **UcastniciZajezdu**. Tělo třídy navrhněte tak, aby instance této třídy mohla v poli uchovávat objekty typu ostatních tříd v projektu(výjimku tvoří třída **Osoba**, která je abstraktní, tedy nemůže vytvářet instance). Dále vytvořte metody pro vkládání objektů do pole a pro tisk seznamu účastníků zájezdu, kde se každý účastník představí pomocí metody **predstavSe()** a dále informuje o svém platu pomocí metody **plat()**. Pokud budou nutné změny v ostatních třídách, proveďte je.

viz[6]

15 Závěr

Tato bakalářská práce byla pojata jako „studijní pomůcka" pro výuku programování v objektově programovacím jazyku Java. Je určena spíše slabším studentům.

Jedním z cílů práce bylo přiblížit čtenáři programovací jazyk Java a motivovat k jeho užití. K tomuto účelu byly použity řešené příklady s názornou obrazovou dokumentací, která usnadňuje pochopení jazyka. Čtenář se v prvních kapitolách seznámil s úvodní teorií objektově orientovaného programování. Tu si v závěrečných kapitolách týkajících se autorizovaného přístupu k datům, dědičnosti a polymorfizmu rozšířil.

Mezi těmito kapitolami získal základní dovednosti týkající se programování. To je seznámil se s základními datovými typy, řídícími konstrukcemi a cykly. Získal i základní informace o využití knihovních tříd.

Věřím, že tuto práci využijí studenti, kteří se budou chtít přehlednu formou seznámit s tímto jazykem.

Seznam použité literatury

Literatura:

[1] Bruce, Eckel. Myslíme v jazyku Java : knihovna programátora. Praha : Grada Publishing, 2001. 431 s. ISBN 80-247-9010-6.

[2] David J. Barnes, Michael Kölling. Objects First With Java, second edition. Pearson Education Limited, 2005.

[3] Herout, Pavel. Java : Bohatství knihoven. [s.l.] : [s.n.], 2006. 252 s. ISBN 80-7232-288-5.

[4] Herout, Pavel. Učebnice jazyka Java. 1. vyd. České Budějovice : Koop, 2003. 349 s. ISBN 80-7232-115-3.

[5] Horton, Ivor. Java 5. Přeložil Petr Poledňák. 1. vyd. Praha : Neocortex, 2005. 1423 s. ISBN 1-861005-69-5.

[6] Pecinovský, Rudolf. Myslíme objektově v jazyku Java 5.0. Palasová Jaroslava. [s.l.] : Grada Publishing, 2004. 604 s. ISBN 80-247-0941-4.

Internet:

[7] BlueJ - Documentation [online]. [2006] [cit. 2008-02-02]. Dostupný z WWW: <http://www.bluej.org/doc/documentation.html>.

[8] Getting Started With the NetBeans IDE 5.0 BlueJ Edition [online]. 1994 [cit. 2008-03-03]. Dostupný z WWW: <http://java.sun.com/developer/technicalArticles/tools/bluej/>.

[9] Overview (Java Platform SE 6) [online]. 1994 [cit. 2008-04-15]. Dostupný z WWW: <http://java.sun.com/javase/6/docs/api/>.

[10] The Java Tutorials [online]. [1994] [cit. 2008-04-15]. Dostupný z WWW: <http://java.sun.com/docs/books/tutorial/>.