2. Objekty

Java je objektově orientovaný programovací jazyk přenositelný na různé platformy. Pracuje tedy s objekty. Co to vlastně jsou objekty? Jedná se o abstrakci z reality, každý objekt představuje spojení dat (údajů, proměnných, datových atributů) a činností s těmito daty (metod). Tato abstrakce je vždy účelová, o každém reálném objektu se sledují ty údaje, které jsou relevantní pro aplikaci.
Pokud chceme pracovat s objekty, je nutné vědět, jaké jsou obecné vlastnosti objektů. Nyní tyto vlastnosti uvedeme a postupně si je objasníme. V této kapitole se také seznámíme se spoustou pojmů z objektového programování.

Obecné objektové vlastnosti:

- používání abstrakce
- definování tříd objektů
- existence objektů (instancí)
- komunikace objektů (posílání zpráv, volání metod)
- zapouzdření a ukrývání implementace
- dědičnost
- polymorfismus

Základní pojmy:

- objekty
- třídy
- instance
- datové atributy
- metody
- konstruktory
- balíčky
- deklarace
- inicializace
- identifikátor
- formální parametr metody
- skutečný parametr metody
- pomocná proměnná

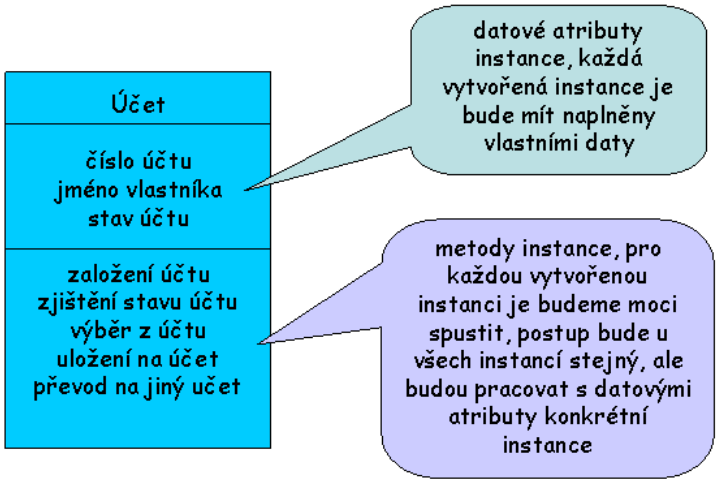
2.1. Objekty a abstrakce

Abstrakce je základní objektovou vlastností. Skutečnost, kterou chceme do programu promítnout, musíme vždy zjednodušit, pracovat jen s těmi daty, která jsou pro nás důležitá.
Uvedeme si několik jednoduchých příkladů.
První příklad:
Když chceme udělat počítačovou evidenci knih, které jsou k dispozici v obchodě, bude základem abstrakce knihy. V knihkupectví nás bude pravděpodobně u každé knihy zajímat: autor, název, ISBN, vydavatel, žánr, cena, počet kusů na skladě. S knihou budeme provádět např. tyto činnosti: založení nové knihy, změna množství na skladě, změna ceny.
Druhý příklad:
Vytváříme jednoduchou evidenci knih ve své knihovně, abychom měli přehled o knihách. Protože je často půjčujeme, chceme si půjčky evidovat. Základem je opět kniha, ale tentokrát nás o ní bude zajímat autor, název, žánr, komu je zapůjčena atd. Činnosti budou např.: zapiš výpůjčku nebo kniha vrácena.
Třetí příklad:
Chceme napsat aplikaci pro kreslení. Tvary (čtverce, obdélníky, kruhy, trojúhelníky atd.), které budou nakresleny, jsou objekty. U každého nakresleného tvaru musíme sledovat např. tato data: souřadnice umístění, rozměry tvaru, barvu čáry, barvu výplně. Metody neboli činnosti, které bude možno v takové aplikaci provádět s jednotlivými tvary, budou např. tyto: nakreslení, zvětšení, zmenšení, posun, změna barvy, vymazání.
Čtvrtý příklad:
Pokud budeme psát aplikaci pro počítání obvodů a obsahů dvourozměrných tvarů, budou zde podobné objekty, ale s jinými daty a metodami. U jednotlivých tvarů stačí sledovat rozměry stran a popřípadě velikosti úhlů. Metody, které budou k dispozici, budou metody pro výpočet obsahu a obvodu.

Pátý příklad:
Jedná se o velmi zjednodušenou „banku“. Základem každé banky jsou účty jejích klientů. Každý učet v naší bance bude mít pouze číslo účtu, jméno vlastníka a uloženou částku. Činnosti, které bude možné s jednotlivými účty provádět, mohou být např. tyto: založení účtu, zjištění stavu účtu, výběr z účtu, uložení na účet, převod na jiný učet. Tento příklad budeme dále používat pro objasňování dalších pojmů a jejich realizace v Javě.

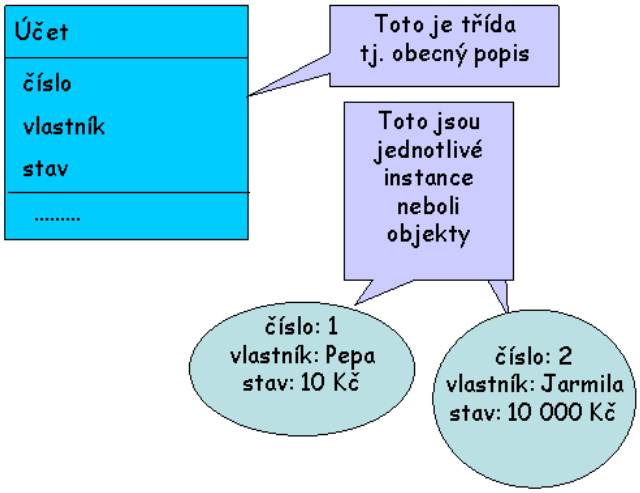
2.2. Třída a instance

Vraťme se k našemu příkladu s bankou. V bance je samozřejmě mnoho účtů. Programátor musí vytvořit obecný popis všech účtů, vytvořit třídu `Účet`.
Třída je obecný popis, ve kterém se deklarují (určí) data (datové atributy), která budou popisovat stav objektu, a metody, které definují činnosti, jaké je možné s objekty provádět.
Jak je vidět na následujícím obrázku, programátor definuje, které údaje se budou o objektech sledovat a jaké činnosti bude s těmi daty možno provádět a jak se to bude dělat.



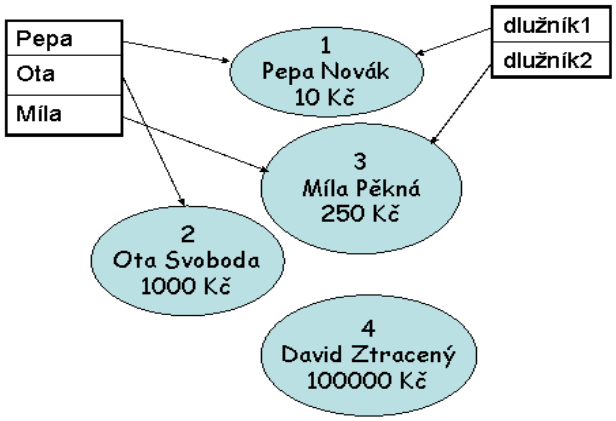
Obrázek 2.1 Popis jednotlivých částí třídy

V programu se poté vytvoří několik instancí této třídy, představující jednotlivé účty. Instance je tedy vytvořena v paměti počítače a vytváří jakýsi obraz reálného objektu např. učet Josefa Nováka nebo účet Jarmily Pavlíčkové.



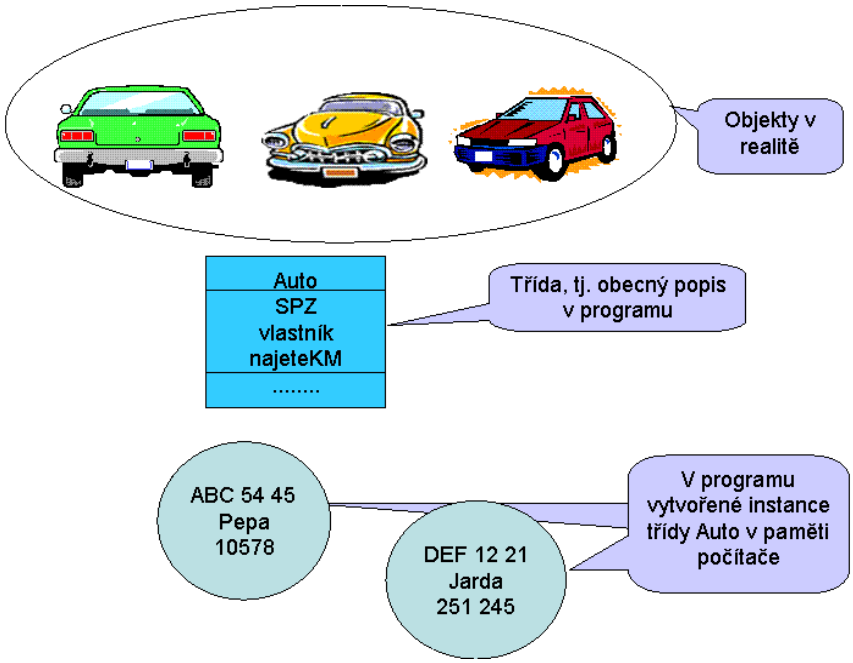
Obrázek 2.2 Znázornění rozdílu mezi třídou a jejími instancemi

Data, která budeme o každé instanci sledovat, označujeme jako datové atributy instance.
Činnosti, které je možné s danými instancemi provádět, označujeme jako metody (metody instance).
Konstruktor je specifická metoda, pomocí které se vytvářejí instance. Při spouštění konstruktoru si musíme uložit odkaz (referenci) na vznikající instanci. K již vzniklé instanci je možno přiřadit i další odkazy. Na obrázku 2.3 máme znázorněny odkazy na účty s čísly 1 až 3. Na účet číslo 4 odkaz nemáme a nemůžeme s ním dále pracovat. Nelze na něm zavolat žádnou metodu (poslat mu zprávu). Na účty s čísly 1 a 3 byly vytvořeny další odkazy. Je možné jim poslat zprávu „ze dvou míst“.



Obrázek 2.3 Znázornění odkazů na instance tříd

V objektovém programování se používají pojmy objekt a instance jako ekvivalenty, my budeme používat termín instance, protože je přesnější.
Abychom si tyto pojmy ještě více ujasnili, uvedeme si další příklad. Budeme vytvářet jednoduchou evidenci vozidel. Auta stojící na ulici jsou objekty, o kterých budou v programu „záznamy“. Pomocí abstrakce vytvoříme obecný popis auta vhodný pro náš účel a vytvoříme tak třídu `Auto`. Pokud bude aplikace fungovat, bude pro každý objekt v realitě vytvořena odpovídající instance třídy `Auto` v paměti počítače. Obrázek 2.4 znázorňuje popsanou situaci.



Obrázek 2.4 Význam pojmů objekt, třída a instance

2.3. Volání metod (posílání zpráv)

Každá aplikace je tvořena několika třídami, v rámci běhu aplikace jsou vytvářeny instance těchto tříd a volány jejich metody.
Volání metod se tak trochu podobá posílání SMS z mobilního telefonu. Můžeme poslat SMS jen tomu, na koho máme číslo. Posíláme např. zprávu „Kup chleba!“ Pokud příjemce neumí česky, je nám to na nic (voláme metodu, kterou daná instance nezná, v Javě tento omyl odchytí již překladač). Pokud pošleme tuto zprávu např. Honzovi, ten na koupení chleba zapomene, když ji pošleme Pepovi, ten chleba přinese. U objektů mohou instance různých tříd reagovat na stejnou zprávu různě, v našem případě by Honza musel být instancí třídy Lajdák a Pepa instancí třídy Svědomitý. Jedna instance může také na stejnou zprávu reagovat různě v závislosti na svém stavu – pokud bude svědomitý Pepa ve stavu spaní, tak žádný chleba nepřinese.
Příklad s tvary: Když pošleme instanci třídy `Ctverec` zprávu `kresli()`, nakreslí se čtverec. Když pošleme stejnou zprávu (zavoláme metodu `kresli()`) instanci třídy `Kruh`, nakreslí se kruh.
Tento jev, kdy posíláme stejnou zprávu (tj. voláme metodu stejného jména), ale provádí se činnosti různě implementované, se označuje jako **polymorfismus**.

2.4. Zapouzdření

Pojem **zapouzdření** (encapsulation) popisuje princip umísťování dat a souvisejících metod k sobě – do jednoho objektu, do jedné metody, atd. Zapouzdření musí být podporováno vhodnou jazykovou konstrukcí, v Javě i ostatních objektových jazycích se realizuje pomocí třída a vytváření instancí. Při zapouzdřování objektů je vhodné dodržovat tato pravidla:

- Snažit se umístit data a operace pracující s daty do stejné třídy. V našem příkladu s bankou jsou ve třídě `Účet` umístěny nejen datové atributy, ale i metody, které s účtem pracují. Chybou zapouzdření by bylo např. umístit metody pro „výběr z účtu“, „uložení na účet“ a pro „převod na jiný účet“ do samostatné třídy[1].
- Třída by neměla obsahovat jen část dat či část metod, ani by neměla obsahovat více dat či metod, než je nutné pro činnost, za kterou třída odpovídá. Častější chybou je, že třída obsahuje více dat, než kolik potřebuje. V našem příkladu s bankou by např. bylo chybou, kdyby se ve třídě `Účet` evidovala adresa banky.
- Jednotlivé zprávy posílané instanci by pokud možno měly být na sobě nezávislé, metody by měly požadovat jen nezbytně nutné parametry. Chybou tohoto typu by bylo, kdybychom metodu pro převod peněz na jiný účet rozdělili do dvou metod – v první metodě bychom nastavili číslo cílového účtu, ve druhé zadali převáděnou částku a provedli vlastní převod. Správným řešením je, že metoda bude obsahovat dva parametry – číslo cílového účtu a převáděnou částku a po počátečních kontrolách provede převod částky na zadaný účet.

2.5. Ukrývání implementace

Každý objekt poskytuje svému okolí metody, které je možné zavolat. Seznam těchto metod (a dostupných datových atributů) je označován jako **veřejné rozhraní třídy**. V tomto rozhraní by neměly být zahrnuty datové atributy, ty by měly být schovány uvnitř instance a měly by být přístupné pouze pomocí metod. Pro ilustraci v našem jednoduchém příkladě s účty by hodnota stavu účtu neměla být z vnějšku přístupná. K zjištění stavu a změnám hodnot by měly sloužit metody „zjištění stavu účtu“, „výběr z účtu“, „vložení na účet“. Důvodem je, že pomocí metod může docházet k dalším kontrolám a instance se tak nemůže dostat do nesprávného stavu. V případě našeho účtu např. není povolen výběr do mínusu a v metodě pro výběr se tedy bude kontrolovat, zda je možno výběr uskutečnit nebo ne.
Tento přístup – poskytování pouze nezbytně nutných metod pro uživatele třídy – se nazývá **ukrývání implementace** (information hiding).

---

[1] Umísťování metod do samostatných jednotek se používalo dlouho ve strukturovaném programování. Jednou z nevýhod bylo velké množství parametrů těchto metod.

Při návrhu tříd je vhodné se řídit těmito pravidly:

- ukrývejte datové atributy – datové atributy by neměly být dostupné přímo, ale pomocí metod.
- ukrývejte implementaci třídy a minimalizujte veřejné rozhraní. Metody, které nejsou nutné pro uživatele třídy, znepřístupněte pro uživatele třídy. Postupy, které jsou složitější či u kterých lze předpokládat změny (např. konkrétní algoritmus třídění či konkrétní algoritmus pro výpočet úroků) umístěte do vnitřních metod, které nebude možné přímo volat. Minimalizace rozhraní by však neměla být na úkor použitelnosti vytvářené třídy.

Předpokladem pro ukrývání implementace je, že programovací jazyk podporuje zapouzdření[2]. Oba pojmy spolu úzce souvisí, což někdy vede k jejich nepřesnému používání či vzájemnému zaměňování.

2.6. Vytváření tříd v Javě

Když chceme vytvořit třídu v Javě, musíme napsat její zdrojový kód do souboru. Zdrojové kódy se píší do textových souborů (formát prostý text). Soubor se zdrojovým kódem třídy má koncovku **.java** a jmenuje se stejně jako třída, která je v něm popsána.
V souboru mohou být na začátku uvedeny klauzule *package* a *import*, jejichž význam si objasníme později. Popis třídy začíná hlavičkou třídy, v níž musí být uvedeno klíčové slovo **class** a jméno třídy. Další níže uvedené prvky jsou volitelné a budou objasněny postupně. Tělo třídy je uvedeno mezi složenými závorkami. Zde je třeba uvést, jaké datové atributy bude třída obsahovat a jaké metody bude možné volat. Třída může obsahovat i další prvky, které budou popsány později.
Obecně vypadá deklarace třídy takto:

```
[ package jménoBalíčku; ]
[ import JménoTřídy;.... ]

[public] [final | abstract] class jmeno [extends JménoRodičovskéTřídy]
[implements JménoRozhraní …]
{
     datové atributy třídy;
     datové atributy;
     vnořené třídy;
     konstruktory;
     metody;
}
```

Hlavička třídy *Ucet* z našeho úvodního příkladu bude vypadat takto:

```
public class Ucet {
  //zde bude popsáno jaké datové atributy a metody bude třída obsahovat
}
```

2.6.1. Identifikátory

Jméno třídy, datového atributu, metody atd. se označuje jako **identifikátor**. Pojmenování jednotlivých součástí třídy, i třídy samotné, se řídí určitými pravidly.
Základní pravidla pro vytváření identifikátorů v Javě jsou následující:

- identifikátor je tvořen posloupností písmen, číslic a podtržítka, začíná písmenem,
- Java rozlišuje malá a velká písmena: *cislo* a *Cislo* jsou dva různé identifikátory,
- identifikátor nesmí obsahovat klíčové slovo Javy (seznam klíčových slov pro verzi 5.0 je uveden v tabulce 2.1).

[2] Současné objektové programovací jazyky podporují obě vlastnosti, v historii se však najdou jazyky, které podporovaly pouze zapouzdřování.

abstract	continue	for	new	switch
assert	default	if	package	synchronized
boolean	do	goto	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Tabulka 2.1 Seznam klíčových slov Javy 5.0

Tato pravidla pro identifikátory jsou závazná a kontroluje je překladač. Další pravidla pro identifikátory stanovená firmou Sun je vhodné dodržovat pro lepší čitelnost a pochopitelnost kódu, ale nejsou kontrolována překladačem. Nyní uvedeme alespoň základní doporučení.

- Identifikátor by měl vystihovat obsah proměnné/datového atributu, význam formálního parametru, třídy nebo činnost metody.
- Při vytváření identifikátorů se v Javě používá tzv. velbloudí notace (výjimku představují konstanty), u víceslovných identifikátorů je každé první písmeno slova uvedeno velkým písmenem např. `rodneCislo, stavUctu`. Zda je první písmeno identifikátoru malé či velké se řídí následujícími pravidly:
    * velké písmeno na začátku, počáteční písmena každého dalšího slova velká, ostatní písmena malá – používá se pro označení třídy a rozhraní,
    * malé písmeno na začátku, počáteční písmena dalších slov velká, ostatní písmena malá – používá se pro proměnné tříd a instancí, pro pomocné proměnné metod, pro formální parametry metod a pro jména metod,
    * všechna písmena velká, jednotlivá slova oddělena podtržítkem _ – používá se pro pojmenované konstanty.

Nedoporučujeme používání diakritiky v identifikátorech.

2.6.2. Datové atributy instance

Datové atributy uchovávají informace o instanci mezi jednotlivými voláními metod.
Každý datový atribut musí mít určený typ a jméno (identifikátor). Určení jména a typu se označuje jako **deklarace**. Přehled datových typů v Javě je uveden v kapitole 3, zde si pouze uvedeme, že vytvořením nové třídy vznikne další datový typ. Před datovým typem je uveden modifikátor přístupu, který určuje úroveň zapouzdření. V případě datových atributů se téměř vždy používá modifikátor `private` (podrobnosti viz podkapitola „Modifikátory přístupu k datovým atributům a metodám“ na straně 20). Další specifické modifikátory budou objasněny v následujících kapitolách.
Deklarace datového atributu se obecně zapisuje takto:

```
[modifikátory] typ identifikátor;
```

Následují příklady:

```
private int cisloUctu;       //typ int je celé číslo
private String jmenoVlastnika;     //typ String znamená řetězec znaků
private double castka;       //typ double je desetinné číslo
```

Nastavení počáteční hodnoty se nazývá **inicializace**. Pokud není u datových atributů explicitně uvedena, přiřadí se defaultní hodnota.
Deklaraci a inicializaci je možné spojit do jednoho příkazu.

```
[modifikátory] typ identifikátor = hodnota;
```

Příklad následuje:

```
    private double castka = 0;
```

Každý příkaz v Javě je ukončen středníkem a píše se na samostatný řádek (z důvodu lepší čitelnosti kódu). Deklarace všech datových atributů by měly být na jednom místě v kódu, obvykle se uvádějí na začátku nebo na konci třídy. My budeme deklarace datových atributů uvádět vždy na začátku třídy. Deklarace třídy *Ucet* včetně datových atributů tedy bude vypadat takto:

```
public class Ucet {

    private int cisloUctu;
    private String jmenoVlastnika;
    private double castka = 0;
  //zde budou následovat metody
}
```

2.6.3. Metody instance

Metody představují dovednosti, činnosti, které může objekt (instance) provádět. Metody jsou deklarovány ve třídě.
Metoda se skládá s hlavičky (podpisu) metody a těla metody, které je tvořeno pomocí příkazů a deklarací lokálních proměnných uvedených mezi složenými závorkami.

```
[modifikátory] typNávratovéHodnoty jménoMetody ( [formálníParametry] )
                                [throws výjimky]
{
    [deklarace_proměnných;]
    [příkazy;]
}
```

Hlavička (podpis) metody má několik částí:

- modifikátor přístupu,
- další (nepovinné) modifikátory,
- typ návratové hodnoty,
- jméno (identifikátor),
- kulaté závorky (povinné), které mohou obsahovat deklaraci formálních parametrů metody,
- vyhazované výjimky (nepovinná část) viz kapitola 12.

Metody mohou mít uvedeny různé **modifikátory přístupu**, jež jsou podrobněji popsány v podkapitole „Modifikátory přístupu k datovým atributům a metodám“. V hlavičce metody mohou být i některé další modifikátory.
Typ návratové hodnoty metody je povinný. Metoda může vracet libovolný typ platný v Javě (viz kapitola 3). Pokud žádnou hodnotu nevrací, musí být uvedeno vyhrazené slovo **void**.
Formální parametry metody slouží k předání vstupních hodnot do metody. Každý parametr je v hlavičce metody deklarován podobně jako datové atributy typem a jménem, není možné (ani smysluplné) uvádět modifikátor přístupu (parametr platí pouze v metodě). V hlavičce metody není možné přiřadit parametru defaultní hodnotu.
Lokální proměnná (pomocná proměnná metody) se v metodě používá pro uložení nějakého mezivýsledku a po ukončení činnosti metody je zrušena.
Deklarace a inicializace pomocné proměnné se od inicializace datového atributu liší tímto:

- neuvádějí se modifikátory přístupu,
- proměnné nejsou implicitně inicializovány, první hodnotu musí nastavit programátor.

Problematika rozsahu platnosti lokálních proměnných je uvedena v kapitole 4 v souvislosti se sekvencí (blokem) příkazů.

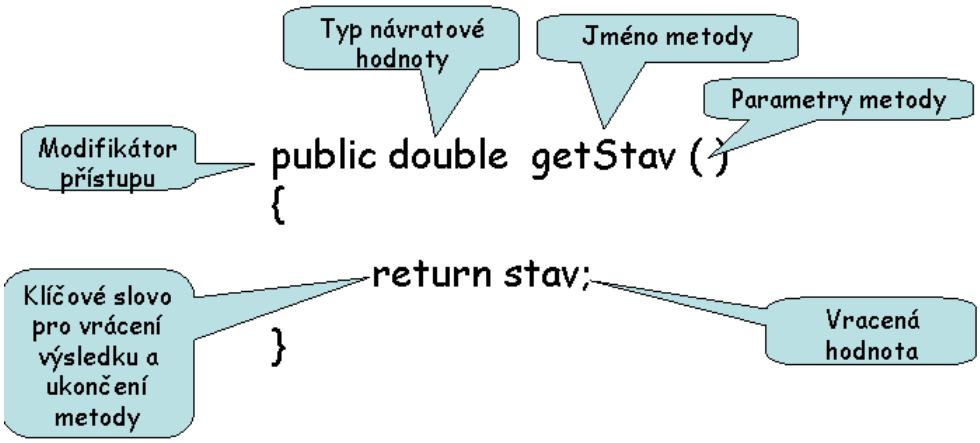
V těle metody, které je zapsáno mezi dvěma složenými závorkami, mohou být následující **příkazy**:

- volání metody,
- přiřazení,
- příkaz return,
- sekvence (posloupnost, blok příkazů),
- selekce (rozhodování, větvení),
- iterace (cyklus, opakování),
- příkaz skoku z cyklu,
- vyvolání a obsluha výjimek,
- prázdný příkaz,
- příkaz assert pro testování metody.

Postupně si jednotlivé příkazy vysvětlíme v této i dalších kapitolách.
Alespoň některé druhy metod si nyní ukážeme na našem jednoduchém příkladě s účty.

Metoda bez parametrů s návratovou hodnotou

Příkladem takovéto metody bude ve třídě `Ucet` metoda pro zjištění stavu účtu. Stav účtu je představován datovým atributem stav typu `double`, návratová hodnota metody tedy musí být také typu `double`. Pro vrácení hodnoty z metody má Java příkaz (klíčové slovo) **return**, uvedením tohoto příkazu se provádění příkazů v metodě ukončí a vrátí se hodnota za ním uvedená. V naší metodě tedy musí být tento příkaz uveden a musí vrátit hodnotu datového atributu `stav`. Metodu pojmenujeme trochu podivně `getStav()`. Důvodem pro toto pojmenování je další ze zvyklostí zavedených v Javě. Pokud metoda vrací hodnotu nějakého datového atributu, měla by se jmenovat `getJmenoAtributu()`, proto tedy `getStav()` Metoda bude mít modifikátor přístupu `public`, patří do veřejného rozhraní třídy `Ucet`. Na následujícím obrázku je znázorněno, jak bude vypadat kód této metody (deklarace metody `getStav`).



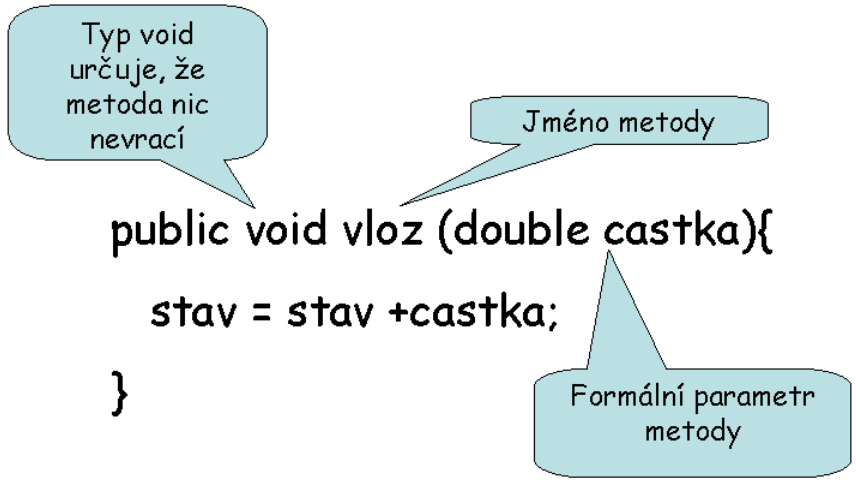
Obrázek 2.5 Popis metody s návratovou hodnotou

Důležité je si uvědomit, že ačkoli metoda nemá parametry, kulaté závorky jsou v podpisu metody vždy uvedeny.

Metoda s parametry bez návratové hodnoty

Příkladem takovéto metody ve třídě `Ucet` je metoda pro vložení další částky na účet. V tomto případě potřebujeme metodě sdělit, kolik peněz na účet vkládáme. Tento údaj představuje vstupní parametr metody. Při deklaraci metody musíme uvést, jakého typu bude parametr – vzhledem k tomu, že stav je typu `double`, tak i vkládané částky budou tohoto typu. Metoda přidá uvedenou částku na účet a nevrátí žádnou hodnotu, v deklaraci tedy bude uveden návratový typ **void**. Metoda, která nevrací hodnotu, nemá v kódu zpravidla uveden příkaz `return` (pokud ano, nesmí za ním být uvedeno nic) a

končí provedením všech příkazů uvedených v metodě. Jak bude kód metody vypadat i s vysvětlivkami, vidíte na obrázku 2.6.



Obrázek 2.6 Popis metody s parametrem

V hlavičce metody se může deklarovat více parametrů, jednotlivé deklarace jsou od sebe odděleny čárkou.

Metoda s parametry a s návratovou hodnotou

Jako příklad takovéto metody můžeme ve třídě `Ucet` uvést metodu pro výběr peněz. Požadovaná částka bude vstupním parametrem metody a bude typu `double`. Pokud bude na účtu dostatek peněz, metoda provede odpočet částky z účtu a vrátí `true` (výběr se povedl). Jestliže požadovaná částka bude vyšší než stav účtu, vrátí metoda hodnotu `false` (výběr se nepovedl) a částka odečtena nebude. Podrobnější vysvětlení příkazu `if`, který se v metodě použije, najdete v kapitole 4. Nyní si uvedeme, jak bude vypadat deklarace třídy `Ucet` včetně těchto tří metod. Povšimněte si také odsazování jednotlivých části kódu a zápisu složených závorek. Párování závorek je možné zapisovat dvěma způsoby. První byl uveden na obrázku 2.5 s kódem metody `getStav()`. Druhý je používán ve všech ostatních případech. Vyberte si ten způsob zápisu kódu, který se vám jeví jako čitelnější.

```
public class Ucet {

     private int cisloUctu;
     private String jmenoVlastnika;
     private double castka = 0;

     public double getStav(){
          return stav;
     }

     public void vloz (double castka){
          stav = stav + castka;
     }
```

```
    public boolean vyber (double castka){
        if ((stav – castka) >= 0) {
            stav = stav – castka;
            return true;
        }
        else {
            return false;
        }
    }
    //zde budou další metody
}
```

2.6.4. Modifikátory přístupu k datovým atributům a metodám

Následující modifikátory uvádějí možnost přístupu k datovému atributu nebo metodě:

- private,
- (nic neuvedeno),
- protected,
- public.

Označíme-li nějaký datový atribut nebo metodu jako **private**, znamená to, že je přístupná pouze z metod instance. V příkladě s účty jsou takto označeny všechny datové atributy. Stav účtu si v metodě jiné třídy nemohu přečíst přímo z datového atributu, ale pomocí metody `getStav()`. Obdobně z metod jiných tříd mohu změnit stav účtu zase jen prostřednictvím metod.
Jako druhý modifikátor vlastně není nic uvedeno, ale znamená to, že pokud neuvedeme žádný modifikátor přístupu, použije se **přátelský přístup**. Datové atributy a metody jsou v tomto případě přístupné v rámci balíčku (viz následující podkapitola).
Datové atributy a metody, které mají označení přístupu **protected**, jsou přístupné v rámci balíčku a také z potomků v rámci dědičné hierarchie (viz kapitola 11). Modifikátor `protected` se použije u metod, které usnadní psaní potomků, ale které nemají význam pro běžné použití instance třídy (poté by měl být použit modifikátor `public`).
Označení přístupu **public** znamená, že daný datový atribut či metoda jsou přístupné z jakékoli jiné třídy.
Jak již bylo uvedeno, pokud deklarujeme samostatnou třídu, můžeme před klíčovým slovem `class` uvést pouze modifikátor `public` nebo žádný. Jejich význam je stejný jako v případě datových atributů a metod. Třídu označenou jako `public` „vidí“ všechny ostatní třídy, třídu bez modifikátoru `public` jen třídy ze stejného balíčku.

2.6.5. Konstruktor

Jak již bylo v textu uvedeno, instance jsou vytvářeny pomocí speciálních metod, které označujeme jako konstruktory.
V Javě platí pro konstruktory několik pravidel, která je odlišují od ostatních metod:

- konstruktor se vždy jmenuje jako třída (výjimka z pravidla o malých počátečních písmenech metod),
- konstruktor nemá uveden žádný návratový typ (ani `void`),
- pokud žádný konstruktor nenapíšete, vytvoří překladač prázdný veřejný konstruktor bez parametrů,
- konstruktory se na rozdíl od ostatních metod nedědí (viz kapitola 11 o dědičnosti).

Konstruktor slouží k vytvoření instance a většinou také k inicializaci datových atributů. Ukážeme si, jak bude vypadat konstruktor naší třídy `Ucet`. V rámci vytváření instance vždy nastavíme číslo účtu a jméno vlastníka. Tyto hodnoty budou představovat vstupní parametry konstruktoru. V průběhu vytváření instance budou tyto hodnoty dosazeny do odpovídajících datových atributů. Kód konstruktoru bude vypadat takto:

```
     public Ucet (int cislo, String vlastnik){
          cisloUctu = cislo;
          jmenoVlastnika = vlastnik;
     }
```

Je tedy možné vytvořit novou instanci třídy *Ucet* s daným číslem a vlastníkem, stav tohoto účtu bude 0.00. Pokud bychom chtěli vytvořit účet s určitou počáteční hodnotou, která by byla u různých instancí různá, musíme napsat ještě jeden konstruktor. Tento konstruktor bude mít navíc ještě jeden parametr typu *double* a bude představovat počáteční vklad. Kód tohoto konstruktoru bude vypadat takto:

```
     public Ucet (int cislo, String vlastnik, double pocatecniVklad){
          cisloUctu = cislo;
          jmenoVlastnika = vlastnik;
          stav = pocatecniVklad;
     }
```

Jak vidíte, jedna třída může mít libovolný počet konstruktorů, musejí se od sebe lišit počtem, pořadím nebo typem parametrů. Této vlastnosti se říká **přetěžování** (**overloading**). Přetížit lze i „obyčejné“ metody, v jedné třídě může být libovolný počet metod stejného jména, které se liší počtem, pořadím nebo typem parametrů. Metodu ale nelze přetížit pouze typem návratové hodnoty.
Kód naší třídy *Ucet* rozšířený o konstruktory je uveden v následujícím výpise.

```
public class Ucet {

     private int cisloUctu;
     private String jmenoVlastnika;
     private double stav = 0;

     public Ucet (int cislo, String vlastnik){
          cisloUctu = cislo;
          jmenoVlastnika = vlastnik;
     }

     public Ucet (int cislo, String vlastnik, double pocatecniVklad){
          cisloUctu = cislo;
          jmenoVlastnika = vlastnik;
          stav = pocatecniVklad;
     }

     public double getStav(){
          return stav;
     }

     public void vloz (double castka){
          stav = stav + castka ;
     }

     public boolean vyber (double castka){
          if ((stav – castka) >= 0) {
               stav = stav – castka;
               return true;
          }
          else {
               return false;
          }
     }
     //zde budou další metody
}
```

2.7. Vytváření instancí a volání metod v Javě

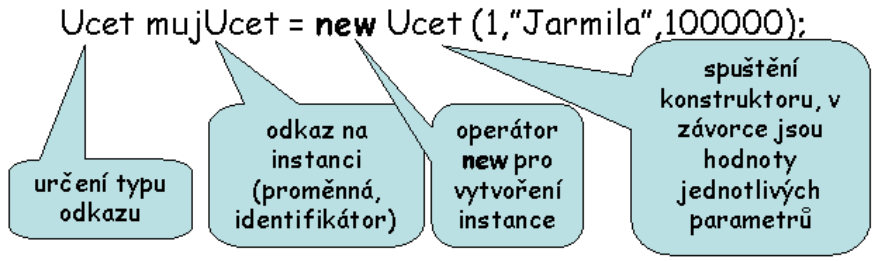
Zatím jsme si popsali, jak vytvořit třídu tj. jak udělat obecný popis nějakých objektů z reality. Nyní ukážeme, jak v programu (aplikaci) vytvářet jednotlivé instance a volat jejich metody. V Javě bude tento kód opět umístěn v nějaké třídě a její metodě. Může to být např. třída `Banka`. Nebudeme zde však uvádět celý kód třídy, ale pouze jednotlivé řádky kódu, které by byly umístěny v metodách. První věc, kterou si musíme objasnit je, jak vytvořit instanci a uložit si na ni odkaz, abychom s ní mohli dále pracovat.
Pokud chceme vytvořit instanci třídy `Ucet`, musí být identifikátor použitý na uložení odkazu typu `Ucet`, protože každá vytvořená třída je zároveň datovým typem. Identifikátor si pojmenujeme `ucet1`. Deklarace identifikátoru pro uložení odkazu na instanci třídy `Ucet` bude vypadat takto:

```
Ucet ucet1;
```

Vytvoření nové instance proběhne pomocí volání konstruktoru. Volání konstruktoru v Javě je vždy spojeno s klíčovým slovem **new**. Který ze dvou vytvořených konstruktorů se použije, záleží na počtu skutečně zadaných parametrů. Vytvoření instance a přiřazení odkazu na ni do identifikátoru bude vypadat takto:

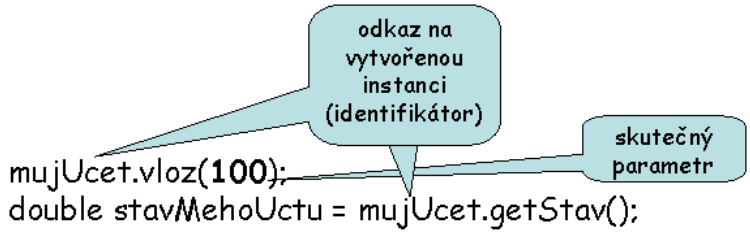
```
ucet1 = new Ucet (1,"Pepa");
```

Deklaraci a inicializaci je možné spojit dohromady do jednoho příkazu, viz obrázek 2.7.



Obrázek 2.7 Popis vytváření instance pomocí konstruktoru

Když máme v identifikátoru uložený odkaz na vytvořenou instanci, můžeme volat metody této instance (můžeme ji posílat zprávy). Pokud metoda nevrací hodnotu, uvedeme při jejím volání pouze odkaz na instanci, tečku a název metody. Pokud má metoda deklarovány formální parametry, musíme při volání uvést skutečné hodnoty, se kterými bude metoda pracovat. Můžeme zadat konstantu nebo identifikátor proměnné, která hodnotu obsahuje nebo na ni odkazuje. Třetí možností je uvést za parametr jinou metodu, která vrací hodnotu příslušného typu.
V případě, že metoda vrací hodnotu a my s ní chceme dále pracovat, musíme výsledek metody přiřadit do proměnné odpovídajícího typu. Na následujícím obrázku je uvedeno, jak zavolat metodu `vloz()` s hodnotou parametru 100 a jak získat informaci o stavu účtu pomocí metody `getStav()`. Výsledek metody `getStav()` musíme uložit do proměnné typu `double`, protože metoda vrací hodnotu tohoto typu.



Obrázek 2.8 Popis volání metod instance

2.7.1. Volání metod v rámci jedné instance (this)

Zatím jsme si ukázali, jak volat metody jiné instance, občas však potřebujeme z jedné metody zavolat jinou metodu stejné instance. V tomto případě potřebujeme odkaz sami na sebe – pro odkaz sama na sebe je v Javě vyhrazeno klíčové slovo **this**. Následuje ukázka metody pro přičtení úroků, parametrem je procento úroku, které se má přičíst.

```
public void prictiUrok (double procento) {
     this.vloz(this.getStav() * procento);
}
```

V metodě *`prictiUrok()`* se volá metoda *`getStav()`* pro zjištění aktuálního stavu účtu a metoda *`vloz()`*, která vloží vypočtený úrok na účet.
Klíčové slovo *`this`* je nepovinné, neboť překladač ho ve většině případů automaticky doplní. Metoda poté může vypadat následovně:

```
public void prictiUrok (double procento) {
     vloz(getStav() * procento);
}
```

`This` se používá i pro odkaz na datové atributy instance. Většinou se neuvádí, neboť obdobně jako u volání metod doplní překladač *`this`*. V situacích, kdy jméno datového atributu koliduje se jménem lokální proměnné či s identifikátorem parametru metody, se však *`this`* uvádět musí. Příkladem je odlišení datových atributů od parametrů metod. Pokud metoda nebo konstruktor přiřazuje hodnotu parametru k datovému atributu, většinou se pro datový atribut a parametr metody používá stejný identifikátor. Má to dvě výhody – kód metody je přehlednější a nemusí se vymýšlet dvojnásobek jmen identifikátorů.
Klíčové slovo *`this`* se používá ještě při volání jednoho konstruktoru z druhého v rámci jedné třídy. Na druhý konstruktor se odkazujeme pomocí *`this`* (tj. ne jménem třídy), správný konstruktor se vybere dle počtu a typů parametrů. Omezením je, že volání druhého konstruktoru musí být prvním příkazem v konstruktoru.
S využitím *`this`* by deklarace datových atributů a konstruktory třídy *`Ucet`* mohly vypadat následovně:

```
public class Ucet {

     private int cislo;
     private String vlastnik;
     private double castka = 0;

     public Ucet (int cislo, String vlastnik){
          this(cislo, vlastnik, 0); // volání druhého konstruktoru
     }

     public Ucet (int cislo, String vlastnik, double pocatecniVklad){
          this.cislo = cislo;
          this.vlastnik = vlastnik;
          castka = pocatecniVklad;
     }
// pokračování třídy
```

Použití stejných jmen pro datové atributy a pro parametry metod je častým důvodem chyb. Pokud použijete stejné názvy, nezapomeňte na *`this`* u datových atributů.

2.8. Modifikátor final

Modifikátor **final** je příznakem neměnnosti/konečnosti. Používá se ve více situacích:

final v deklaraci třídy

Třída s modifikátorem `final` je konečná, tj. nelze vytvářet potomky této třídy. Příkladem mohou být třídy `String`, `Math` či obalové třídy (`Integer`, `Long`, `Boolean`, atd.).

final v deklaraci metody

Metoda s modifikátorem `final` je konečná, tj. tuto metodu nelze překrývat v potomkovi. Modifikátor `final` se uvádí v případě, že chceme zajistit neměnnost metody v potomcích, tj. u metod, které mají jiný modifikátor přístupu než `private`. Metody s modifikátorem přístupu `private` nelze překrýt z principu modifikátoru. Modifikátor `final` nelze použít u konstruktoru.
Pokud je metoda `final`, může být v potomkovi přetížena – v potomkovi může být metoda stejného jména, která se bude lišit počtem či typy parametrů.

final u datového atributu

Datovému atributu s modifikátorem `final` lze přiřadit hodnotu pouze jednou, další pokusy o přiřazení hodnoty končí chybou při překladu. Pokud je datový atribut referenčního typu, je možné pomocí metod měnit obsah instance, nelze však do atributu přiřadit jinou instanci. Datové atributy s modifikátorem `final` mají obvykle i modifikátor `static` – datový atribut je poté společný pro všechny instance třídy, je v paměti pouze jednou. Popis modifikátoru `static` je v kapitole 7.
Následuje deklarace třídy `Bod`, která obsahuje konstantu referenčního typu.

```
class Bod {
     private int x, y;
     final static Bod POCATEK = new Bod(0, 0);
     Bod (int x, int y) {
          this.x = x;
          this.y = y;
     }
}
```

final u pomocné proměnné

V deklaraci pomocné proměnné může být uveden jediný modifikátor – `final`. Význam má stejný jako u datového atributu – do proměnné lze přiřadit hodnotu pouze jednou, další pokusy o přiřazení hodnoty označí překladač za chybu.

final u parametru metody

Je to jediný modifikátor, který může být uveden v deklaraci parametru metody. Při překladu se kontroluje, že se v těle metody do parametru metody nic nepřiřazuje. Přiřazovat hodnotu k parametru metody nemá význam – jak jsme si řekli již dříve, tak při volání metody se hodnoty primitivních datových typů kopírují, v případě referenčních typů se kopíruje odkaz. Tj. pokud uvedeme modifikátor `final` u parametru metody, tak nás pouze překladač upozorní na nesmyslnost přiřazení hodnoty/odkazu do parametru metody.

2.9. Rušení objektů

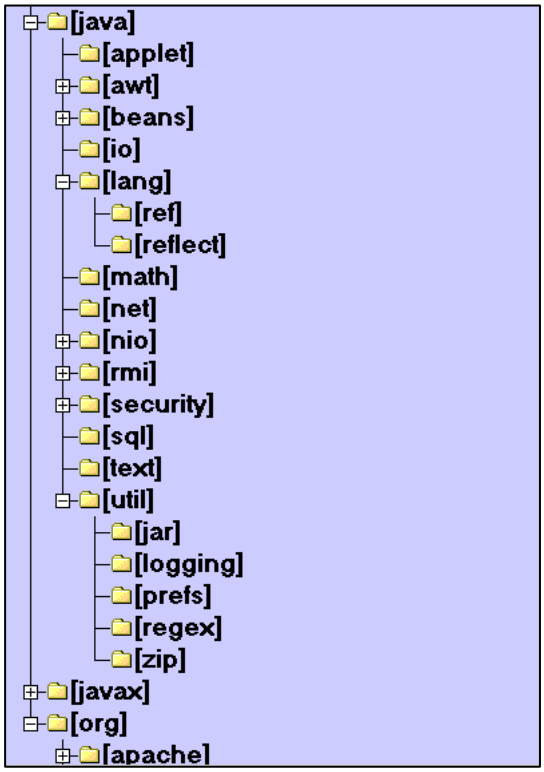
V Javě se programátor nemusí starat o rušení instancí (objektů) – JVM (Java Virtual Machine) obsahuje speciální proces **Garbage Collector** (GC), který pravidelně hledá instance, na které nevede žádný odkaz. Pokud nějakou takovou instanci nalezne (na obrázku 2.3 je takovou instancí „David Ztracený“), tak v prvním kroku zavolá metodu `finalize()`. Ve druhém kroku GC instanci vymaže z paměti.

Metodu `finalize()` má každá instance, neboť ji dědí ze třídy `Object`. Ve třídě `Object` je tato metoda prázdná a obvykle se ve třídách nepřekrývá (překrývání – viz kapitola 11 věnovaná dědičnosti). JVM nezaručuje, že se metoda `finalize()` opravdu provede – typicky při ukončení aplikace se již metoda `finalize()` nevolá.
Tím, že se programátor nemusí starat o rušení objektů, je programování v Javě jednodušší a nedochází obvykle k chybám při práci s pamětí jako v C či v C++. Přesto by se měl programátor snažit dodržovat dvě pravidla:

- Omezovat vytváření objektů, čímž se omezí i náročnost jejich rušení (a aplikace má menší paměťové nároky). Omezování vytváření objektů by však nemělo být na úkor přehlednosti či znovupoužitelnosti kódu,
- pokud nějaký identifikátor odkazuje na již nepotřebnou instanci, měli bychom přiřadit tomuto identifikátoru hodnotu `null` (přiřazovacím příkazem `promenna = null;`). Tím se odkaz na instanci zruší a GC může instanci smazat z paměti (pokud na ni není odkaz odjinud).

2.10. Balíčky (package)

Součástí distribuce Javy je několik tisíc již připravených tříd a rozhraní, které je třeba logicky uspořádat. Třídy jsou proto rozděleny do různých balíčků. Balíček (**package**) tvoří zároveň i základní jmenný prostor, v rámci kterého musí mít třída jednoznačné jméno. Ve více balíčcích mohou být různé třídy stejného jména. Jména balíčků vytvářejí stromovou (adresářovou) strukturu, část struktury balíčku standardu Javy je uvedena na obrázku 2.9.
Programátoři v rámci rozsáhlejších aplikací vytvářejí další balíčky – na začátku souborů se zdrojovým kódem uvedou klauzuli `package` následovanou jménem balíčku. Struktuře balíčků musí odpovídat i adresářová struktura, ve které jsou uloženy zdrojové a přeložené třídy.



Obrázek 2.9 Část adresářové struktury balíčků API Javy 5.0

Existuje ještě tzv. nepojmenovaný balíček (**noname package**). V nepojmenovaném balíčku jsou všechny třídy, které nemají uvedenu klauzuli `package` a jsou uloženy ve stejném adresáři. Takto umístěné třídy budeme používat v těchto skriptech.

Na třídy z nepojmenovaného balíčku se odkazujeme pouze jménem. Obdobně třídy z balíčku **java.lang** lze v programu používat jen s jejich jménem např. *String*, *Integer*, *Object*... Třídy z ostatních balíčků lze vždy označovat plným jménem včetně balíčku, např. *java.util.ArrayList*. Variantou k zadání celého jména je použití klauzule **import** pro příslušnou třídu, poté již lze používat jméno bez jména balíčku:

```
import java.util.ArrayList;
. . . . .
    private ArrayList seznam;
```

V klauzuli *import* lze zadat místo jména konkrétní třídy hvězdičku. Poté se importují všechny třídy a rozhraní z příslušného balíčku (ale ne z podřízených balíčků).

```
import java.util.*;
 . . . . . .
    private ArrayList seznam;
```

Při použití importu (a hlavně importů s hvězdičkou) může dojít k tomu, že se z různých balíčků importují dvě třídy stejného jména. V tomto případě je nutné dále používat celé jméno třídy.

2.11. Komentáře

Rozlišují se dva typy komentářů:

- dokumentační, které se zahrnují do dokumentace vytvářené programem javadoc: */** .... */*
- implementační, které jsou víceřádkové (*/* ....*/*) a jednořádkové (*//*).

Dokumentační komentáře musí být uvedeny před každou deklarací třídy, metody, rozhraní či datového atributu, které mají modifikátor přístupu *public* či *protected*. Začínají znaky */**** a končí znaky **/*. V textu komentáře by měly být použity klíčová slova pro javadoc – viz tabulka 2.2. Text může též obsahovat formátovací znaky HTML.

klíčové slovo		**popis**
@author		jméno autora třídy
@version		číselné označení verze třídy
@see		odkaz na jinou třídu/metodu či na nějaké URL
@param	jméno_parametru	popis parametru
@return		popis návratové hodnoty z metody
@exception	jméno_výjimky	popis výjimky, která může nastat v metodě

Tabulka 2.2 Základní značky pro javadoc

Následuje příklad dokumentačních komentářů:

```
/**
 *  Tato třída slouží ke generování výstupu s různými způsoby
 *  kódování českých znaků
 *
 * @author     xabcd01
 * @created    27. may 2001
 */
```

```
public class Unicode {
     /**
      * seznam českých znaků v kódování Unicode v notaci Javy
      */
     public static final String retezec = "\u00e1\u00c1\u00e9 ...";

     /**
      * metoda main dle prvního parametru příkazové řádky vytiskne
     .* řetězec na standardní výstup v požadovaném kódování
      *
      * @param  ARGS  první parametr příkazové řádky obsahuje řetězec
      *               specifikující požadované kodovani
     .* @see  Supported Encoding v dokumentaci JDK
      */
     public static void main(String ARGS[]) {
     ...
```

Pro generování dokumentace se používá program **javadoc**. Pokud chceme vygenerovat dokumentaci jedné třídy, můžeme zadat příkaz:

```
javadoc Unicode.java
```

který vygeneruje dokumentaci ve formátu html do aktuálního adresáře. Při spouštění programu javadoc je možné uvést mnoho parametrů, nejčastěji používané jsou v následující tabulce:

parametr	**popis**
-d adresář	výstupní adresář
-public -protected -package -private	tyto parametry určují na základě přístupnosti, které prvky se mají zahrnout do dokumentace

Tabulka 2.3 Nejpoužívanější přepínače pro javadoc

Na příkazové řádce je nutné uvést buď seznam zdrojových souborů či jméno balíčku, které musí odpovídat jménu adresáře. Typické spuštění pro soubory v nepojmenovaném balíčku vypadá takto:

```
javadoc –d doc –package *.java
```

Dokumentace se vygeneruje do podadresáře *doc*.
Implementační komentáře slouží ke komentování implementace třídy a měly by se používat v případě, kdy slouží k lepšímu čtení či porozumění kódu. Neměly by duplikovat informace, které lze snadno vyčíst z vlastního kódu.
Potřeba psát implementační komentáře je někdy příznakem nízké kvality návrhu kódu – v tomto případě je vhodnější přepsat kód. Následují příklady implementačních komentářů:

```
/*
 *  komentář, který je na více
 *  řádcích
 */

/* jednořádkový komentář */
příkaz           // komentář, který končí na konci řádku
```