LEKCE 2

Třídy, objekty a metody

V této lekci:

- Co je to vlastně objekt?
- Instance a metody
- Třída Objective-C pro práci se zlomky
- Oddíl @interface
- Oddíl @implementation
- Oddíl vlastního programu
- Přístup k instančním proměnným a zapouzdření dat

V této lekci se seznámíte s některými základními principy objektově orientovaného programování a začnete pracovat se třídami v jazyce Objective-C. Budete se muset naučit trochu terminologie, nebudeme se ovšem pouštět do ničeho formálního. Také si tu přiblížíme jen některé základní termíny, protože jinak byste se snadno mohli ztratit. Přesnější definice všech termínů najdete v příloze A, „Glosář“, na konci knihy.

Co je to vlastně objekt?

Objekt je nějaká věc. Objektově orientované programování můžete považovat za určitou věc a to, co s ní lze dělat. To je zásadní odlišnost oproti programovacím jazykům jako C, označovaným za procedurální programovací jazyky. V jazyce C obvykle nejprve přemýšlíte, co chcete udělat, a teprve potom se staráte o objekty – téměř pravý opak orientace na objekty.

Zvažme příklad z běžného života. Předpokládejme, že vlastníte auto, které je zjevně objektem, navíc vaším. Nemáte prostě jen nějaké auto, ale určité auto vyrobené v továrně, třeba v Detroitu, možná v Japonsku nebo někde úplně jinde. Vaše auto je vozidlem s identifikačním číslem (VIN), které je jednoznačně určuje.

V objektově orientované terminologii je vaše auto *instancí* obecného auta. Pokud bychom pokračovali v terminologii, pak je `auto` názvem třídy, z níž byla naše instance vytvořena. Kdykoli je tedy vyrobeno nové auto, vytvoří se nová instance třídy aut – každá taková instance auta se pak označuje za objekt.

Vaše auto může být stříbrné, může mít černý interiér, může to být kabriolet či mít pevnou střechu atd. Navíc můžete se svým autem provádět nějaké činnosti. Auto kupříkladu řídíte, plníte je benzinem, myjete je (doufejme), jezdíte s ním do servisu atd. Tabulka 2.1 to vše shrnuje.

Akce uvedené v tabulce 2.1 můžete provádět se svým autem a lze je provádět také s jinými auty. Kupříkladu rovněž vaše sestra jezdí svým autem, umývá je, doplňuje benzinem atd.

Tabulka 2.1: Akce na objektech

Objekt	Co s ním můžete provádět
Vaše auto	Řídit
	Doplňovat benzin
	Umývat
	Dávat do servisu

Instance a metody

Jedinečný výskyt nějaké třídy je její instancí a akce prováděné na této instanci se označují jako *metody*. V některých případech lze metodu aplikovat na instanci třídy nebo na třídu samotnou. Kupříkladu mytí auta platí pro instanci (vlastně všechny metody uvedené v tabulce 3.1 lze považovat za instanční metody). Ovšem zjištění, kolik typů aut určitý výrobce nabízí, bude platit pro třídu, takže to bude metoda třídy.

Předpokládejme, že máte dvě auta z výrobní linky, která jsou zdánlivě identická: mají stejný interiér, stejnou barvu atd. Zpočátku mohou být shodná, ovšem jak každé z nich používá jiný majitel, získávají jedinečnost. Jedno může utržit škrábnutí a druhé může mít více najeto. Každá instance neboli objekt obsahuje nejen údaje o počátečních charakteristikách získaných při výrobě, ale také o aktuálním stavu. Tyto charakteristiky se mohou dynamicky měnit. Jak jezdíte autem, ubývá palivo z nádrže, auto se špiní a pneumatiky se trochu opotřebovávají.

Aplikování metody na objekt může ovlivnit jeho *stav*. Je-li vaší metodou „natankuj do mého vozidla palivo", pak po jejím vykonání budete mít plnou nádrž. Metoda tak vlastně ovlivní stav palivové nádrže vozidla.

Základním principem je tu skutečnost, že objekty představují jedinečné reprezentace třídy a že každý objekt obsahuje nějaké informace (data), jež jsou pro něj obvykle soukromé. Metody zajišťují prostředky přístupu k datům a jejich změně.

Programovací jazyk Objective-C má následující syntaxi aplikování metod na třídy a instance:

```
[ TřídaNeboInstance metoda ];
```

V této syntaxi následuje za levou hranatou závorkou název třídy či instance dané třídy a pak jedna či více mezer, za nimiž je metoda, kterou chcete vykonat (respektive zpráva, kterou objektu posíláte, viz níže). Vše je ukončeno pravou hranatou závorkou a zakončujícím středníkem. Když požadujete po nějaké třídě či instanci vykonání určité akce, pak se říká, že jí posíláte určitou *zprávu*; adresát této zprávy se označuje jako *příjemce*. Výše uvedený obecný formát lze tedy také přepsat do tvaru:

```
[ příjemce zpráva ] ;
```

Vraťme se zpět k předchozímu výpisu a přepišme vše do této nové syntaxe. Ovšem nejprve si musíte pořídit nové auto. Zajděte si pro ně do továrny tímto způsobem:

```
vaseAuto = [Auto new];        vzít si nové auto
```

Odešlete zprávu třídě `Auto` (příjemci zprávy) s žádostí předat vám nové auto.Výsledný objekt (reprezentující vaše jedinečné auto) se pak uloží do proměnné `vaseAuto`. Od této chvíle můžete používat `vaseAuto` k odkazování na danou instanci auta, které jste získali z továrny.

Jelikož jste si přišli pro auto do továrny, označuje se metoda `new` za *tovární* metodu nebo také metodu *třídy*. Ostatní akce na vašem novém autě budou metodami instance, protože platí pro vaše auto. Zde máme ukázkové výrazy zpráv, které můžete využít u svého auta:

```
[vaseAuto pripravit];         připravit na první použití
[vaseAuto ridit];             řídit auto
[vaseAuto umyt];              umýt auto
[vaseAuto natankovat];        dotankovat palivo, je-li potřeba
[vaseAuto servisovat];        odjet do servisu

[vaseAuto strechaDole];       pokud je to kabriolet
[vaseAuto strechaNahore];
najeto = [vaseAuto stavTachometru];
```

Tento poslední příklad ukazuje instanční metodu vracející informace – zřejmě aktuální stav tachometru. V tomto případě ukládáme údaj do proměnné v našem programu nazvané `najeto`.

Vaše sestra Zuzka může používat stejné metody na její vlastní instanci auta:

```
[zuzkyAuto ridit]
[zuzkyAuto umyt];
[zuzkyAuto natankovat];
```

Aplikování shodných metod na různé objekty je jedním z klíčových principů objektově orientovaného programování – více se dozvíte za chvíli.

Ve svých programech zřejmě nebudete muset pracovat s auty. Vašimi objekty budou spíše nějaké počítačové věci, jako jsou okna, obdélníky, části textu nebo třeba kalkulačka či seznam písní. Ovšem jejich metody mohou vypadat podobně jako u aut, například:

```
[mojeOkno smazat];                        Vymazat obsah okna

[mujObdelnik zjistitPlochu];              Vypočítat plochu obdélníku

[uzivatelskyText zkontrolovatPravopis];   Zkontrolovat překlepy v textu

[stolniKalkulacka vymazatZadani];         Vymazat poslední zadání

[seznamOblibenych zobrazitPisne];         Ukázat písně na seznamu oblíbených

[telefonniCislo vytocit];                 Vytočit nějaké telefonní číslo
```

Poznámka

Velmi často hovoříme o volání či uplatňování metod na různých objektech. Je nutné podotknout, že to není terminologicky zcela korektní, jelikož správně bychom měli ve většině případů hovořit o posílání zpráv. Výraz `[příjemce zpráva]` totiž v Objective-C nevolá přímo metodu shodnou se jménem zprávy. Ve skutečnosti můžeme objektu poslat jakoukoliv zprávu a on se až při jejím přijetí rozhodne, co s ní udělá. Nejčastěji zavolá „sám na sobě“ patřičnou metodu, ale také může zprávu předat jinému objektu nebo ji odmítnout. Tento princip si podrobněji probereme v lekci 8, Polymorfizmus, dynamické typování a dynamická vazba, jelikož se jedná o zásadní vlastnost dobře navrženého objektového jazyka, jakým Objective-C bezesporu je.

Třída Objective-C pro práci se zlomky

Nastal čas definovat skutečnou třídu v Objective-C a naučit se pracovat s jejími instancemi.

Opět se nejprve seznámíme s obecnou procedurou. Díky tomu se nemusejí jevit ukázky vlastního programu jako výrazně praktické. K takovým záležitostem se dostaneme později.

Představme si, že potřebujete napsat program pracující se zlomky. Třeba potřebujete zpracovávat jejich sčítání, odečítání, násobení atd. Kdybyste nevěděli nic o třídách, mohli byste začít vytvořením následujícího jednoduchého programu:

Výpis 2.1

```
// Jednoduchý program práce se zlomky
#import <Foundation/Foundation.h>

int main (int argc, char *argv[])
```

```
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int citatel = 1;
    int jmenovatel = 3;
    NSLog (@"Zlomek je %i/%i", citatel, jmenovatel);

    [pool drain];
    return 0;
}
```

Výstup výpisu 2.1

```
Zlomek je 1/3
```

Ve výpisu 2.1 je zlomek reprezentován svým čitatelem a jmenovatelem. Jakmile je vytvořen automatický zásobník, tak dva řádky v proceduře `main` současně deklarují proměnné `citatel` a `jmenovatel` jako celá čísla a přiřazují jim výchozí hodnoty 1 a 3. To odpovídá následujícím řádkům:

```
int citatel, jmenovatel;

citatel = 1;
jmenovatel = 3;
```

Zlomek 1/3 jsme reprezentovali uložením 1 do proměnné `citatel` a hodnoty 3 do proměnné `jmenovatel`. Pokud byste potřebovali ve svém programu ukládat mnoho zlomků, byl by tento systém nepraktický. Kdykoli byste se chtěli odkazovat na nový zlomek, museli byste se odkazovat na odpovídající čitatel a jmenovatel. A provádění operací s takovými zlomky by bylo stejně nešikovné.

Bylo by lepší definovat zlomek jako entitu a odkazovat se na jeho čitatel i jmenovatel jediným názvem, např. `mujZlomek`. To lze v Objective-C provést – vše začíná definováním nové třídy.

Výpis 2.2 duplikuje funkčnost výpisu 2.1 s využitím nové třídy nazvané `Zlomek`. Zde je tedy nový program i s podrobným vysvětlením jeho funkčnosti.

Výpis 2.2

```
// Program práce se zlomky - verze se třídou
#import <Foundation/Foundation.h>

//---- oddíl @interface ----

@interface Zlomek: NSObject
{
    int citatel;
    int jmenovatel;
}

-(void) tisk;
-(void) zadatCitatel: (int) n;
-(void) zadatJmenovatel: (int) d;

@end
```

```
//---- oddíl @implementation ----

@implementation Zlomek
-(void) tisk
{
    NSLog (@"%i/%i", citatel, jmenovatel);
}

-(void) zadatCitatel: (int) n
{
    citatel = n;
}

-(void) zadatJmenovatel: (int) d
{
    jmenovatel = d;
}
@end

//---- oddíl program ----

int main (int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    Zlomek *mujZlomek;

    // Vytvořit instanci třídy Zlomek

    mujZlomek = [Zlomek alloc];
    mujZlomek = [mujZlomek init];

    // Nastavit zlomek na 1/3

    [mujZlomek zadatCitatel: 1];
    [mujZlomek zadatJmenovatel: 3];

    // Zobrazit zlomek metodou tisk

    NSLog (@"Hodnota promenne mujZlomek je:");
    [mujZlomek tisk];
    [mujZlomek release];

    [pool drain];
    return 0;
}
```

Výstup výpisu 2.2

```
Hodnota promenne mujZlomek je:
1/3
```

Jak je zřejmé z komentářů ve výpisu 2.2, je celý logicky rozdělen do tří oddílů:

- oddíl `@interface`
- oddíl `@implementation`
- oddíl `program`

Oddíl (sekce) `@interface` popisuje třídu, její datové součásti a její metody, zatímco oddíl `@implementation` obsahuje vlastní kód implementující příslušné metody. Konečně programový oddíl obsahuje kód programu vykonávající určený účel.

Každý z těchto oddílů je součástí všech programů v Objective-C, i když nemusíte vždy každý oddíl vytvářet sami. Jak uvidíte, jednotlivé oddíly se často vkládají do samostatných souborů. Prozatím je však ponecháme všechny v jednom.

Oddíl @interface

Když definujete novou třídu, musíte provést několik věcí. Především musíte říci kompilátoru Objective-C, odkud třída pochází. To znamená, že musíte určit její *rodičovskou* třídu. Pak musíte zadat, jaký typ dat se bude ukládat do objektů této třídy. To znamená, že musíte popsat data, jež budou instance dané třídy obsahovat. Tyto členy se označují jako *instanční proměnné*. Nakonec musíte definovat typ operací, tedy metody, jež lze používat při práci s objekty této třídy. To vše se provádí ve zvláštním oddílu programu označovaném jako oddíl `@interface`. Obecný formát tohoto oddílu vypadá následovně:

```
@interface NázevNovéTřídy: NázevRodičovskéTřídy
{
    deklaraceČlenů;
}

deklaraceMetod;
@end
```

Podle konvence začínají názvy tříd velkým písmenem, není to ovšem vyžadováno. Na druhou stranu pak ale dokážete při čtení programu odlišit názvy tříd od ostatních typů proměnných prostým pohledem na první znak v jejich názvu. Tak krátce odbočme a povězme si něco o názvech v Objective-C.

Výběr názvů

V lekci 1, „Programování v Objective-C“, jste používali několik proměnných k ukládání celočíselných hodnot. Tak jsme kupříkladu použili proměnnou `soucet` ve výpisu 1.4 k uložení výsledku sečtení dvou celých čísel, 50 a 25.

Jazyk Objective-C vám dovoluje ukládat do proměnných také jiné datové typy než jen celá čísla, ovšem před použitím takové proměnné v programu je zapotřebí řádně ji deklarovat. Proměnné lze používat k ukládání desetinných čísel, znaků, a dokonce i objektů (nebo, přesněji řečeno, odkazů na objekty).

Pravidla vytváření názvů proměnných jsou poměrně jednoduchá. Musejí začínat písmenem nebo znakem podtržítka (_) a pak může následovat libovolná kombinace písmen (velkých i malých), podtržítek a číslic 0 až 9. Zde máme ukázku platných názvů:

- `soucet`
- `priznakCasti`
- `i`
- `mojeMisto`
- `pocetPohybu`
- `_sysPriznak`
- `Sachovnice`

Oproti tomu následující názvy nejsou z výše uvedených důvodů platné:

- `soucet$hodnot` – `$` není platný znak.
- `priznak casti` – vložené mezery nejsou povoleny.
- `3Kusy` – názvy nemohou začínat číslicí.
- `int` – jedná se o rezervované slovo.

Označení `int` nelze použít jako název proměnné, protože má pro kompilátor Objective-C zvláštní význam. Takové použití se označuje jako *klíčové neboli rezervované slovo*. Obecně platí, že žádný název se zvláštním významem pro kompilátor Objective-C nemůže být použit jako jméno proměnné. Příloha B, „Přehled jazyka Objective-C 2.0“, uvádí úplný seznam takových rezervovaných slov.

Vždy pamatujte, že v Objective-C je rozdíl mezi malými a velkými písmeny. Proto názvy `soucet`, `Soucet` a `SOUCET` představují tři odlišné proměnné. Jak bylo zmíněno, při pojmenovávání třídy začněte velkým písmenem. Názvy instančních proměnných, objektů a metod zase většinou začínají malými písmeny. Kvůli zvýšení čitelnosti se v názvech používají velká písmena k vyznačení nových slov, jak to vidíte zde:

- `SeznamAdres` – toto by mohl být název třídy.
- `soucasnaPolozka` – toto by měl být objekt.
- `soucasna_polozka` – někteří programátoři oddělují slova podtržítky.
- `pridatNovouPolozku` – toto by mohl být název metody.

Když vymýšlíte nějaký název, pamatujte na jedno doporučení: Nebuďte líní! Vybírejte názvy zachycující určený smysl dané proměnné či objektu. Důvod je jasný: stejně jako komentáře i smysluplné názvy výrazně zvýší čitelnost programu a vyplatí se ve fázích ladění a dokumentování. Dokumentování bude jistě výrazně jednodušší, protože program se bude do značné míry vysvětlovat sám.

Zde máme ještě jednou oddíl `@interface` z výpisu 2.2:

```
//---- oddíl @interface ----

@interface Zlomek: NSObject
{
    int citatel;
    int jmenovatel;
}
```

```
-(void) tisk;
-(void) zadatCitatel: (int) n;
-(void) zadatJmenovatel: (int) d;

@end
```

Názvem nové třídy je `Zlomek` a jeho rodičovskou třídou je `NSObject`. (O rodičovských třídách si popovídáme detailně později, v lekci 7, „Dědičnost".) Třída `NSObject` je definována v souboru `NSObject.h`, který se do vašeho programu automaticky zahrnuje při každém importu `Foundation.h`.

Instanční proměnné

Oddíl *deklaraceČlenů* specifikuje, jaké typy dat se ukládají v instancích třídy `Zlomek` a také názvy těchto datových typů. Jak vidíte, tento oddíl je uzavřen ve vlastní dvojici složených závorek. V případě naší třídy `Zlomek` deklarace říkají, že instance třídy `Zlomek` má dva celočíselné členy nazvané `citatel` a `jmenovatel`:

```
int citatel;
int jmenovatel;
```

Členy deklarované v tomto oddílu se označují jako instanční proměnné. Jak uvidíte, kdykoli vytvoříte nějaký nový objekt, vytvoří se také nová a jedinečná množina instančních proměnných. Když tedy máte dva objekty třídy `Zlomek`, jeden nazvaný `zlomekA` a druhý označený `zlomekB`, pak bude mít každý svou vlastní sadu instančních proměnných. To znamená, že `zlomekA` a `zlomekB` budou mít každý svůj vlastní `citatel` a `jmenovatel`. Systém Objective-C to vše sleduje za vás a právě to je na práci s objekty nejhezčí.

Metody třídy a instancí

Je zapotřebí definovat metody pracující s objekty třídy `Zlomek`. Budete potřebovat, aby bylo možné nastavit hodnotu zlomku. Protože nebudete mít přímý přístup k interní reprezentaci zlomku (jinými slovy přímý přístup k instančním proměnným), musíte napsat metody nastavení čitatele a jmenovatele. Dále napíšete metodu označenou `tisk`, která zobrazí hodnotu zlomku. Zde máme deklaraci metody `tisk` v hlavičkovém souboru:

```
-(void) tisk;
```

Úvodní znaménko minus (–) říká kompilátoru Objective-C, že tato metoda je instanční metodou. Jedinou další možností je znaménko plus (+) označující metodu třídy. Metoda třídy je taková, která vykonává nějakou operaci na třídě samotné, kupříkladu vytváří její nové instance. To se podobá výrobě nového auta v tom, že auto je třídou a vy chcete určitou metodou třídy vytvořit nové.

Instanční metoda vykonává určitou operaci na konkrétní instanci třídy, např. nastavuje hodnotu, navrací hodnotu, zobrazuje hodnotu atd. Když se vrátíme k příkladu s autem, tak po jeho vytvoření můžete do auta např. doplnit palivo. Operace doplnění paliva se provádí s konkrétním autem, čímž je analogická instanční metodě.

Návratové hodnoty

Když deklarujete novou metodu, musíte říci kompilátoru Objective-C, zda tato metoda vrací nějakou hodnotu, a pokud ano, jakého je typu. To provedete uzavřením návratového typu do závorek za úvod-

ním znaménkem minus nebo plus. To znamená, že následující deklarace specifikuje, že instanční metoda nazvaná `zjistitCitatel` vrací celočíselnou hodnotu:

```
-(int) zjistitCitatel;
```

Podobně následující řádek deklaruje metodu vracející hodnotu s dvojnásobnou přesností. (Více se o tomto datovém typu dozvíte v lekci 3, „Datové typy a výrazy".)

```
-(double) zjistitDvojitouHodnotu;
```

Hodnota se z metody vrací s využitím příkazu `return` jazyka Objective-C podobným způsobem, jako jsme vraceli hodnotu z procedury `main` v předchozích ukázkách programů.

Jestliže metoda nevrací žádnou hodnotu, tak to oznámíte použitím typu `void`, jako je tomu zde:

```
-(void) tisk;
```

Toto deklaruje instanční metodu nazvanou `tisk`, jež nevrací žádnou hodnotu. V takovém případě nemusíte vykonávat příkaz `return` na konci své metody. Alternativně můžete provést `return` bez zadané hodnoty takto:

```
return;
```

Nemusíte specifikovat návratové typy svých metod, ovšem je to dobrá programovací praxe. Pokud nezadáte typ, je výchozím `id`. Více se o datovém typu `id` dozvíte v lekci 8, „Polymorfizmus, dynamické typování a dynamická vazba". Jen si řekněme, že `id` lze v zásadě použít k odkazu na libovolný typ objektu.

Argumenty metod

V oddílu `@interface` výpisu 2.2 jsou deklarovány dvě další metody:

```
-(void) zadatCitatel: (int) n;
-(void) zadatJmenovatel: (int) d;
```

Obě jsou to instanční metody, jež nevracejí žádnou hodnotu. Každá metoda přebírá celočíselný argument, jak to označuje zadání `(int)` před názvem argumentu. V případě metody `zadatCitatel` je názvem argumentu `n`. Tento název může být libovolný a představuje označení, pod jakým se metoda na předaný argument odkazuje. Proto deklarace `zadatCitatel` určuje předání jednoho celočíselného argumentu nazvaného `n`. Navíc je určeno, že se nebude z metody vracet žádná hodnota. Metoda `zadatJmenovatel` je velmi podobná, jen její argument je označen `d`.

Všimněte si syntaxe deklarace uvedených metod. Název každé metody končí dvojtečkou oznamující kompilátoru Objective-C, že tato metoda očekává nějaký argument.

Pak je zadán typ argumentu v závorkách velmi podobně, jako je určen návratový typ metody samotné. Nakonec je zadán symbolický název identifikující tento argument v metodě. Celá deklarace končí středníkem. Obrázek 2.1 popisuje celou syntaxi.

Když nějaká metoda přebírá argument, přidáte rovněž k jejímu názvu dvojtečku. Proto jsou `zadatCitatel:` a `zadatJmenovatel:` správné způsoby identifikace obou metod, jež přebírají jediný argument. Dále určení metody `tisk` bez přidané dvojtečky značí, že tato metoda nepřebírá žádné argumenty. V lekci 6, „Třídy zblízka", uvidíte možnosti definování metod přebírajících více argumentů.

```
-          (void)          zadatCitatel:          (int)          n;
```

typ metody — *návratový typ* — *název metody* — *metoda přebírá argument* — *typ argumentu* — *název argumentu*

Obrázek 2.1: Deklarování metody

Oddíl @implementation

Jak bylo zmíněno, oddíl `@implementation` obsahuje vlastní kód metod deklarovaných v oddílu `@interface`. Z hlediska terminologie říkáme, že se daná metoda deklaruje v oddílu `@interface` a že se definuje (tedy zadává se vlastní kód) v oddílu `@implementation`.

Obecný formát oddílu `@implementation` vypadá takto:

```
@implementation NázevNovéTřídy
    definiceMetod;
@end
```

NázevNovéTřídy je stejný název, který jste použili v oddílu `@interface`. Můžete použít přidanou dvojtečku a název rodičovské třídy, jako jsme to udělali v oddílu `@interface`:

```
@implementation Zlomek: NSObject
```

To je ovšem nepovinné a většinou se toto zadání nepoužívá.

Část *definiceMetod* oddílu `@implementation` obsahuje kód jednotlivých metod zadaných v oddílu `@interface`. Podobně jako v oddílu `@interface` začíná definice každé metody určením jejího typu (třídy či instance), návratového typu i argumentů a jejich typů. Řádek ovšem nekončí středníkem, ale následuje kód metody uzavřený ve dvojici složených závorek.

Podívejme se na oddíl `@implementation` ve výpisu 2.2:

```
//---- oddíl @implementation ----
@implementation Zlomek
-(void) tisk
{
    NSLog (@"%i/%i", citatel, jmenovatel);
}

-(void) zadatCitatel: (int) n
{
    citatel = n;
}

-(void) zadatJmenovatel: (int) d
{
    jmenovatel = d;
}

@end
```

Metoda tisk využívá NSLog k zobrazení hodnot instančních proměnných citatel a jmenovatel. Ovšem na jaký čitatel a jmenovatel se tato metoda odkazuje? Odkazuje se na instanční proměnné obsažené v objektu, jenž je příjemcem dané zprávy. To je důležitý princip, k němuž se za chvilku vrátíme.

Metoda zadatCitatel: ukládá celočíselný argument nazvaný n do instanční proměnné citatel. Podobně zadatJmenovatel: ukládá hodnotu svého argumentu d do instanční proměnné jmenovatel.

Oddíl vlastního programu

Programový oddíl obsahuje kód řešící váš konkrétní problém a může být v případě potřeby rozmístěn v mnoha souborech. Někde musíte mít rutinu nazvanou main, jak již bylo dříve zmíněno. Tam vždy začne váš program s vykonáváním. Zde je oddíl program výpisu 2.2:

```
//---- oddíl program ----

int main (int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    Zlomek *mujZlomek;

    // Vytvoření instance třídy Zlomek

    mujZlomek = [Zlomek alloc];
    mujZlomek = [mujZlomek init];

    // Nastavení zlomku na 1/3

    [mujZlomek zadatCitatel: 1];
    [mujZlomek zadatJmenovatel: 3];

    // Zobrazení zlomku metodou tisk

    NSLog (@"Hodnota promenne mujZlomek je:");
    [mujZlomek tisk];

    [mujZlomek release];
    [pool drain];

    return 0;
}
```

Uvnitř main definujete proměnnou nazvanou mujZlomek následujícím řádkem:

```
Zlomek *mujZlomek;
```

Tento řádek říká, že mujZlomek je objekt třídy Zlomek; to znamená, že mujZlomek se používá k ukládání hodnot z nové třídy Zlomek. Hvězdička (*) před mujZlomek je nutná, ovšem zatím se o její účel nestarejte. Technicky říká, že mujZlomek je ve skutečnosti odkazem (neboli *ukazatelem*) na Zlomek.

Nyní, když máte objekt pro uložení zlomku, stačí jen vytvořit jej, podobně jako žádáte továrnu o výrobu nového auta. Toho dosáhnete následujícím řádkem:

```
mujZlomek = [Zlomek alloc];
```

Slovo `alloc` je zkratkou *alokovat* čili vyhradit. Chcete vyhradit úložný prostor v paměti pro nový zlomek. Tento výraz odešle vaší nově vytvořené třídě `Zlomek` zprávu:

```
[Zlomek alloc]
```

Zde požadujete po třídě `Zlomek` aplikování metody `alloc`, tu jste ovšem nikdy nedefinovali, takže kde se vzala? Tato metoda byla zděděna z rodičovské třídy. Lekce 2, „Třídy, objekty a metody“ probírá toto téma podrobně.

Když třídě odešlete zprávu `alloc`, získáte zpět novou instanci dané třídy. Ve výpisu 2.2 se vrácená hodnota uloží do vaší proměnné `mujZlomek`. Metoda `alloc` zaručuje vynulování všech instančních proměnných daného objektu. To ovšem neznamená, že bude takový objekt řádně inicializován a připraven k použití. Po alokování objektu jej tedy ještě musíte inicializovat.

To zajišťuje následující příkaz ve výpisu 2.2, který má tuto formu:

```
mujZlomek = [mujZlomek init];
```

Opět tu pracujete s metodou, kterou jste sami nevytvořili. Metoda `init` inicializuje instanci třídy. Všimněte si, že zprávu `init` odesíláte objektu `mujZlomek`. To znamená, že tu chcete inicializovat konkrétní objekt třídy `Zlomek` – zprávu neodesíláte třídě, ale nějaké její instanci. Pokuste se tuto skutečnost pochopit, než budete pokračovat.

Také metoda `init` vrací hodnotu – konkrétně inicializovaný objekt. Návratovou hodnotu uložíte do své proměnné `mujZlomek` třídy `Zlomek`.

Uvedená sekvence dvou řádků alokujících novou instanci třídy a následně ji inicializujících je v Objective-C tak častá, že se tyto dvě zprávy obvykle kombinují následovně:

```
mujZlomek = [[Zlomek alloc] init];
```

Nejprve se vyhodnotí vnitřní výraz:

```
[Zlomek alloc]
```

Jak víte, výsledek uvedené zprávy bude vlastním alokovaným objektem třídy `Zlomek`. Výsledek alokování ovšem neuložíme do proměnné jako před chvílí, ale přímo na něj aplikujeme metodu `init`. Opět tedy nejprve alokujeme nový objekt třídy `Zlomek` a pak jej inicializujeme. Výsledek inicializace se pak přiřadí proměnné `mujZlomek`.

Ještě jedna zkratková technika dovoluje zahrnout alokaci a inicializaci přímo do deklaračního řádku tímto způsobem:

```
Zlomek *mujZlomek = [[Zlomek alloc] init];
```

Ve zbytku knihy právě tento styl zápisu kódu často používám, takže byste mu měli rozumět. Vlastně jste takový zápis již viděli ve všech dosavadních programech při alokování automatického zásobníku:

```
NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
```

Zde se zpráva `alloc` odesílá třídě `NSAutoreleasePool` a požaduje tak vytvoření nové instance. Zpráva `init` se pak odesílá nově vytvořenému objektu, aby se inicializoval.

Pokud se vrátíme k výpisu 2.2, budeme již moci nastavit hodnotu zlomku. To provádějí následující řádky kódu:

```
// Nastavit zlomek na 1/3

[mujZlomek zadatCitatel: 1];
[mujZlomek zadatJmenovatel: 3];
```

První příkaz odesílá zprávu `zadatCitatel:` objektu `mujZlomek`. Předaným argumentem je hodnota 1. Řízení se pak předá metodě `zadatCitatel:` definované ve třídě `Zlomek`. Systém Objective-C ví, že má použít metodu této třídy, protože ví, že `mujZlomek` je instancí třídy `Zlomek`.

V metodě `zadatCitatel:` se předaná hodnota 1 uloží do proměnné `n`. Jediný programový řádek v této metodě ukládá zadanou hodnotu do instanční proměnné `citatel`. V zásadě jste tak nastavili hodnotu `citatel` objektu `mujZlomek` na 1.

Následuje zpráva volající metodu `zadatJmenovatel:` objektu `mujZlomek`. Argument s hodnotou 3 se přiřadí proměnné `d` v metodě `zadatJmenovatel:`. Uvedená hodnota se pak uloží do instanční proměnné `jmenovatel` a tím se dokončí přiřazení hodnoty 1/3 objektu `mujZlomek`. Nyní jste připraveni k zobrazení hodnoty svého zlomku, což provedete následujícími řádky ve výpisu 2.2:

```
// Zobrazit zlomek metodou tisk

NSLog (@"Hodnota promenne mujZlomek je:");
[mujZlomek tisk];
```

Volání `NSLog` prostě jen zobrazí následující text:

```
Hodnota promenne mujZlomek je:
```

Následující výraz volá metodu `tisk`:

```
[mujZlomek tisk];
```

V metodě `tisk` se zobrazují hodnoty instančních proměnných `citatel` a `jmenovatel` oddělené znakem lomítka.

Další zpráva v programu uvolní paměť použitou pro instanci třídy `Zlomek`:

```
[mujZlomek release];
```

To je pro dobrý programovací styl důležité. Kdykoli vytvoříte nějaký nový objekt, požadujete alokování čili vyhrazení paměti pro tento objekt. Když pak práci s objektem skončíte, zodpovídáte za uvolnění jím využívané paměti. Je sice pravda, že se paměť po skončení vašeho programu uvolní stejně, ovšem jakmile začnete vyvíjet sofistikovanější aplikace, můžete mít najednou stovky i tisíce objektů spotřebovávajících značný objem paměti. Kdybyste s uvolňováním paměti čekali až na skončení programu, bylo by to plýtvání pamětí a mohlo by to vést také ke zpomalení vykonávání, takže takový přístup není vhodný. Zvykněte si tedy hned od začátku uvolňovat paměť, jakmile je to možné.

Běhový systém (runtime) jazyka Objective-C verze 2.0 nabízí mechanizmus označovaný v angličtině jako „sběr odpadků" (garbage collection), jenž umožňuje automaticky spravovat paměť. Je ovšem lepší naučit se sami ovládat paměť a nespoléhat se na nějaké automatické mechanizmy. Při progra-

mování pro určité platformy dokonce ani nemůžete automatickou správu paměti využívat, jestliže na nich není tato technika k dispozici. Příkladem je iPhone. Právě z toho důvodu se nebudeme o automatické správě paměti zatím bavit.

Zdá se, že bylo nutné ve výpisu 2.2 napsat mnohem více kódu, ovšem výsledek je prakticky totožný s výpisem 2.1. To platí pro zde uvedený jednoduchý příklad; základním cílem práce s objekty je ovšem zjednodušení zápisu, správy i rozšiřování programů. To si uvědomíte později.

Poslední příklad v této lekci ukazuje, jak pracovat v programu s více než jedním zlomkem. Ve výpisu 2.3 nastavíte jeden zlomek na 2/3, druhý na 3/7. Oba pak zobrazíte.

Výpis 2.3

```
// Program práce se zlomky - pokračování

#import <Foundation/Foundation.h>

//---- oddíl @interface ----

@interface Zlomek: NSObject
{
    int citatel;
    int jmenovatel;
}

-(void) tisk;
-(void) zadatCitatel: (int) n;
-(void) zadatJmenovatel: (int) d;

@end

//---- oddíl @implementation ----

@implementation Zlomek
-(void) tisk
{
    NSLog (@"%i/%i", citatel, jmenovatel);
}

-(void) zadatCitatel: (int) n
{
    citatel = n;
}

-(void) zadatJmenovatel: (int) d
{
    jmenovatel = d;
}

@end

//---- oddíl program ----
```

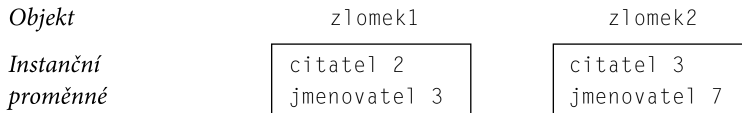
```
int main (int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    Zlomek *zlomek1 = [[Zlomek alloc] init];
    Zlomek *zlomek2 = [[Zlomek alloc] init];

    // Nastavit 1. zlomek na 2/3

    [zlomek1 zadatCitatel: 2];
    [zlomek1 zadatJmenovatel: 3];

    // Nastavit 2. zlomek na 3/7

    [zlomek2 zadatCitatel: 3];
    [zlomek2 zadatJmenovatel: 7];

    // Zobrazit zlomky

    NSLog (@"Prvni zlomek je:");
    [zlomek1 tisk];

    NSLog (@"Druhy zlomek je:");
    [zlomek2 tisk];

    [zlomek1 release];
    [zlomek2 release];

    [pool drain];
    return 0;
}
```

Výstup výpisu 2.3

```
Prvni zlomek je:
2/3
Druhy zlomek je:
3/7
```

Oddíly `@interface` a `@implementation` se neliší od výpisu 2.2. Program vytváří dva objekty třídy `Zlomek` nazvané `zlomek1` a `zlomek2`. Následně přiřazuje hodnotu 2/3 prvnímu zlomku a hodnotu 3/7 druhému zlomku. Uvědomte si, že když je metoda `zadatCitatel:` aplikována na `zlomek1` tak, aby nastavila jeho `citatel` na hodnotu 2, tak se proměnná `citatel` instance `zlomek1` nastaví na 2. Když `zlomek2` použije stejnou metodu k nastavení své proměnné `citatel` na hodnotu 3, tak se na tuto hodnotu nastaví jeho vlastní instanční proměnná `citatel`. Kdykoli vytvoříte nový objekt, bude tento mít svou vlastní sadu instančních proměnných.

To zachycuje obrázek 2.2.



Obrázek 2.2: Jedinečné instanční proměnné

Na základě objektu, kterému se zpráva odesílá, se vybírají příslušné instanční proměnné. Proto se v následujícím příkladu volání metody `zadatCitatel:` pracuje s proměnnou `citatel` objektu `zlomek1`:

```
[zlomek1 zadatCitatel: 2];
```

To je dáno tím, že příjemcem zprávy je `zlomek1`.

Přístup k instančním proměnným a zapouzdření dat

Viděli jste, jak mohou metody pracující se zlomky přistupovat ke dvěma instančním proměnným, `citatel` a `jmenovatel`, přímo zadáním názvu. Platí, že instanční metoda může vždy přímo přistupovat k proměnným této instance. To ovšem metoda třídy nedokáže, protože pracuje pouze se třídou samotnou, a nikoli s nějakou její instancí. (Chvilku se nad tím zamyslete.) Co kdybyste ovšem chtěli přistupovat k instančním proměnným odjinud – například z hlavní rutiny? To nemůžete provést přímo, protože jsou skryté. Skutečnost, že jsou před vámi skryté, je klíčovým principem označovaným jako *zapouzdření dat* (data encapsulation). Ten umožňuje tvůrci definice třídy rozšiřovat je a upravovat podle potřeby, aniž by se staral o to, zda programátoři (tedy uživatelé dané třídy) náhodou nevyužívají určitých jejich vnitřních detailů. Zapouzdření dat představuje užitečnou vrstvu izolace mezi programátorem a vývojářem třídy.

K instančním proměnným můžete čistě přistupovat po zápisu speciálních metod vracejících odpovídající hodnoty. Můžete kupříkladu vytvořit dvě nové metody nazvané celkem zjevně `citatel` a `jmenovatel`, jež budou přistupovat k odpovídajícím instančním proměnným objektu třídy `Zlomek`, který je příjemcem dané zprávy. Výsledkem bude příslušná vrácená celočíselná hodnota. Zde jsou deklarace těchto dvou nových metod:

```
-(int) citatel;
-(int) jmenovatel;
```

A zde máme odpovídající definice:

```
-(int) citatel
{
    return citatel;
}
-(int) jmenovatel
{
    return jmenovatel;
}
```

Všimněte si, že názvy metod odpovídají názvům instančních proměnných, k nimž přistupují. V tom není žádný problém, a dokonce se jedná o běžnou praxi. Výpis 2.4 testuje naše dvě nové metody.

Výpis 2.4

```
// Program přístupu k instančním proměnným - pokračování

#import <Foundation/Foundation.h>

//---- oddíl @interface ----

@interface Zlomek: NSObject
{
    int citatel;
    int jmenovatel;
}

-(void) tisk;
-(void) zadatCitatel: (int) n;
-(void) zadatJmenovatel: (int) d;
-(int) citatel;
-(int) jmenovatel;

@end

//---- oddíl @implementation ----

@implementation Zlomek
-(void) tisk
{
    NSLog (@"%i/%i", citatel, jmenovatel);
}

-(void) zadatCitatel: (int) n
{
    citatel = n;
}

-(void) zadatJmenovatel: (int) d
{
    jmenovatel = d;
}

-(int) citatel
{
    return citatel;
}

-(int) jmenovatel
{
    return jmenovatel;
}

@end
```

```
//---- oddíl program ----

int main (int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    Zlomek *mujZlomek = [[Zlomek alloc] init];

    // Nastavit zlomek na 1/3

    [mujZlomek zadatCitatel: 1];
    [mujZlomek zadatJmenovatel: 3];

    // Zobrazit zlomek pomocí našich nových metod
    NSLog (@"Hodnota mujZlomek je: %i/%i",
        [mujZlomek citatel], [mujZlomek jmenovatel]);
    [mujZlomek release];
    [pool drain];

    return 0;
}
```

Výstup výpisu 2.4

```
Hodnota mujZlomek je 1/3
```

Následující příkaz `NSLog` zobrazuje výsledek odeslání dvou zpráv objektu `mujZlomek`: první vrací hodnotu `citatel` a druhá pak hodnotu `jmenovatel`:

```
NSLog (@"Hodnota mujZlomek je: %i/%i",
    [mujZlomek citatel], [mujZlomek jmenovatel]);
```

Metody, jež nastavují hodnoty instančních proměnných, se často společně označují jako typ *setter* (nastavení, zadání) a metody používané k navracení hodnot z instančních proměnných se označují jako typ *getter* (navracení, převzetí). V případě třídy `Zlomek` jsou `zadatCitatel:` a `zadatJmenovatel:` metodami typu setter metody `citatel` a `jmenovatel` jsou pak typu getter.

Poznámka

Brzy se seznámíte s užitečnou funkčností Objective-C 2.0 umožňující automatické vytváření metod typu setter a getter.

Musím ještě zdůraznit, že je tu také metoda nazvaná `new`, jež kombinuje akce metod `alloc` a `init`. K alokování a inicializování nového objektu třídy `Zlomek` lze tedy použít následující řádek:

```
Zlomek *mujZlomek = [Zlomek new];
```

Obecně je lepší používat alokování a inicializování ve dvou krocích, abyste principiálně chápali výskyt dvou oddělených událostí: nejprve vytvoříte nový objekt a pak jej inicializujete.

Souhrn

Nyní víte, jak definovat svou vlastní třídu, jak vytvářet objekty neboli instance této třídy a jak těmto objektům odesílat zprávy. Ke třídě `Zlomek` se ještě vrátíme v dalších lekcích. Naučíte se předávat metodám více argumentů, rozdělovat definici třídy do několika souborů a také používat klíčové principy jako dědičnost a dynamické vazby. Nyní je však čas dozvědět se trochu více o datových typech a zápisu výrazů v Objective-C. Nejprve si ovšem zkuste následující cvičení a přesvědčte se, že jste dobře pochopili důležité principy popisované v této lekci.

Cvičení

1. Které z následujících názvů nejsou platné? Proč?

```
Int                  hratDalsiSkladbu    6_05
_volatoc             Xx                  rutinaAlfaBeta
vymazatObrazovku     _1312               z
ZnovuInicializovat   _                   A$
```

2. Na základě příkladu auta v této lekci si vyberte nějaký jiný běžně používaný objekt. Určete tomuto objektu třídu a napište pět akcí s takovým objektem prováděných.
3. Použijte seznam ve cvičení 2 a pomocí následující syntaxe přepište seznam do následujícího formátu:

```
[instance metoda];
```

4. Představte si, že kromě auta vlastníte také loď a motorku. Uveďte akce, které lze provést se všemi takovými typy objektů. Jsou některé akce totožné?
5. Využijte odpověď z otázky 4 a představte si, že máte třídu nazvanou `Vozidlo` a objekt označený `mojeVozidlo`, který může být autem, motorkou nebo lodí. Představte si také, že jste zapsali následující:

```
[mojeVozidlo pripravit];
[mojeVozidlo natankovat];
[mojeVozidlo servisovat];
```

   Vidíte nějakou výhodu v možnosti aplikovat akci na určitý objekt, který může pocházet z jedné z několika možných tříd?

6. V procedurálních jazycích jako C nejprve zvažujete akce a pak zapisujete kód vykonávající takové akce na různých objektech. Zvážíme-li náš příklad s autem, mohli byste napsat v jazyce C proceduru, která auto umyje. V této proceduře by byl kód zajišťující umytí auta, umytí lodě, umytí motorky atd. Kdybyste takto postupovali a pak jste chtěli doplnit nový typ vozidla (viz předchozí cvičení), ukázal by se takový procedurální přístup jako výhodný nebo nevýhodný oproti objektově orientovanému přístupu?
7. Definujte třídu nazvanou `BodXY`, jež bude obsahovat kartézské souřadnice (x, y), kde jsou `x` a `y` celá čísla. Definujte metody jednotlivě nastavující souřadnice `x` a `y` bodu a také jeho hodnoty navracející. Napište v Objective-C program implementující vaši novou třídu a program otestujte.