

Objektovo-orientované programovanie

Objekty, Java a aspekty

VALENTINO VRANIĆ

Objektovo-orientované programovanie

Objekty, Java a aspekty

Slovenská technická univerzita
v Bratislave
2008

PUBLIKÁCIU PODPORILO ZDRUŽENIE

GRATEX IT INŠTITÚT

v rámci fondu GraFIIT

www.gratex.com

© Ing. Valentino Vranić, PhD.

Lektori: doc. Ing. Pavol Herout, PhD.
doc. Ing. Ján Kollár, PhD.

Vydala Slovenská technická univerzita v Bratislave vo Vydavateľstve STU, Bratislava,
Vazovova 5.

Text neprešiel jazykovou úpravou vydavateľstva.

Schválilo vedenie Fakulty informatiky a informačných technológií STU v Bratislave
pre študijný program Informatika a študijný program Počítačové systémy a siete.

ISBN 978-80-227-2830-0

OBSAH

ZOZNAM OBRÁZKOV	v
PREDHOVOR	vii
1 ÚVOD	1
2 VHĽAD DO OBJEKTIVO-ORIENTOVANÉHO PROGRAMOVANIA	3
2.1 VZNIK OBJEKTIVO-ORIENTOVANÉHO PROGRAMOVANIA	3
2.2 OBJEKTY A TRIEDY	4
2.3 STAV A SPRÁVANIE OBJEKTU	4
2.4 TYP OBJEKTU	5
2.5 AGREGÁCIA	5
2.6 ZAPUZDRENIE	6
2.7 DEDENIE	7
2.8 POLYMORFIZMUS	8
2.9 SUMARIZÁCIA	10
3 PROGRAMOVACÍ JAZYK JAVA	11
3.1 JAVA — JAZYK A PLATFORMA	11
3.2 OBJEKTY V JAVE	13
3.3 PRIMITÍVNE TYPY	14
3.4 TRIEDY	15
3.5 UVOĽŇOVANIE PAMÄTE	19
3.6 PRVÝ PROGRAM	19
3.7 ZDROJOVÉ SÚBORY A PREKLAD	20
3.8 BALÍKY	20
3.9 RIADENIE PRÍSTUPU	23
3.10 KOMENTÁR A VNORENÁ DOKUMENTÁCIA	24
3.11 OPERÁTORY V JAVE	25
3.12 RIADENIE VYKONÁVANIA PROGRAMU	30
3.13 PREŤAŽENIE METÓD	31
3.14 INICIALIZÁCIA, KONŠTRUKTORY A FINALIZÁCIA	33
3.15 POLIA	38
4 AGREGÁCIA A DEDENIE	43
4.1 AGREGÁCIA	43

4.2	DEDENIE	43
4.3	PREKONÁVANIE METÓD	46
4.4	RIADENIE PRÍSTUPU V DEDENÍ	48
4.5	INICIALIZÁCIA PRI DEDENÍ	48
4.6	TRIEDA OBJECT	49
4.7	KĹÚČOVÉ SLOVO FINAL	49
4.8	DEDENIE A TYPY	51
5	POLYMORFIZMUS	53
5.1	POLYMORFIZMUS A PREKONÁVANIE	53
5.2	ABSTRAKTNÉ TRIEDY A METÓDY	55
5.3	POLYMORFIZMUS A STATICKÉ METÓDY	56
5.4	ROZHRANIA	58
6	APLIKÁCIA OBJEKTIVO-ORIENTO VANÝCH MECHANIZMOV	63
6.1	RÔZNORODOSŤ OBJEKTIVO-ORIENTO VANÝCH PRÍSTUPOV	64
6.2	ABSTRAKCIA	65
6.3	ZAPUZDRENIE	66
6.4	MODULÁRNOSŤ	68
6.5	HIERARCHIA	68
6.6	TYPOVOSŤ A POLYMORFIZMUS	72
6.7	SUMARIZÁCIA	73
7	VHNIEZDENÉ TYPY	75
7.1	PREKLAD VHNIEZDENÝCH TYPOV	75
7.2	VNÚTORNÉ TRIEDY	76
7.3	ANONYMNÉ TRIEDY	79
7.4	STATICKÉ VHNIEZDENÉ TRIEDY	81
8	VÝNIMKY	83
8.1	PODPORA VÝNIMIEK V JAVE	83
8.2	KONTROLA VÝNIMIEK	84
8.3	VLASTNÉ VÝNIMKY	86
8.4	VÝNIMKY PRI PREKONANÝCH METÓDACH	88
9	RTTI	89
9.1	LITERÁL CLASS	89
9.2	ROZHODOVANIE NA ZÁKLADE TYPU	89
9.3	REFLEKTÍVNE TRIEDY	90
10	ZOSKUPENIA OBJEKTOV A GENERICKOSŤ	93
10.1	TYPY ZOSKUPENÍ	93
10.2	GENERICKOSŤ ZOSKUPENÍ	94
10.3	NÁHRADNÝ ZNAK PRE TYP	94
10.4	KONTROLA TYPOV V GENERICKOSTI	96

10.5	ITERÁTORY	97
10.6	ROZŠÍRENÁ SLUČKA FOR	98
10.7	VYMENOVANÉ TYPY	98
10.8	GENERICKÉ METÓDY	100
10.9	AUTOMATICKÉ BALENIE HODNÔT PRIMITÍVNYCH TYPOV	102
10.10	TRIEDA CLASS A GENERICKOSŤ	102
11	VSTUPNO/VÝSTUPNÝ SYSTÉM JAVY	105
11.1	PRÁCA S ADRESÁRMÍ	105
11.2	V/V SYSTÉM JAVY ZALOŽENÝ NA PRÚDOCH	107
11.3	ČÍTANIE SÚBORU PO RIADKOKH	108
11.4	ŠTANDARDNÝ V/V SYSTÉM	108
11.5	FORMÁTOVANÝ VÝSTUP	109
11.6	ČÍTANIE Z PAMÄTE PO ZNAKOKH	110
11.7	ČÍTANIE Z PAMÄTE PODĽA FORMÁTU ÚDAJOV	110
11.8	ZÁPIS A ČÍTANIE SÚBOROV	111
11.9	KANÁLY A VYROVNÁVACIE PAMÄTE	114
11.10	SÚBORY ZOBRAZENÉ DO PAMÄTE	115
11.11	KOMPRESIA	116
11.12	SERIALIZÁCIA OBJEKTOV	116
12	VIACNIŤOVOSŤ	119
12.1	VYTVÁRANIE NITÍ	119
12.2	RIADENIE NITÍ	120
12.3	SYNCHRONIZÁCIA NITÍ	121
13	GRAFICKÉ POUŽÍVATEĽSKÉ ROZHRAKIE V JAVE	125
13.1	POUŽÍVATEĽSKÉ ROZHRAKIE	125
13.2	SWING	126
13.3	TVORBA OKIEN	127
13.4	PRIDÁVANIE KOMPONENTOV DO JFRAME	129
13.5	SPRACOVANIE UDALOSTÍ VO SWINGU	131
13.6	NÍŤ NA ODOSIELANIE UDALOSTÍ VO SWINGU	134
14	OBJEKTOVO-ORIENTOVANÉ MODELOVANIE	137
14.1	MODELOVANIE SOFTVÉRU	137
14.2	UML	138
14.3	DIAGRAM TRIED	140
14.4	DIAGRAM PRÍPADOV POUŽITIA	146
14.5	DIAGRAM SEKVENCÍ	150
15	NÁVRHOVÉ VZORY	153
15.1	VZORY VO VÝVOJI SOFTVÉRU	153
15.2	ARCHITEKTONICKÝ VZOR MODEL-VIEW-CONTROLLER	155
15.3	GoF VZORY	156

15.4	NÁVRHOVÝ VZOR VISITOR	157
15.5	IDIÓM DOUBLE DISPATCH	161
15.6	NÁVRHOVÝ VZOR OBSERVER	162
16	ASPEKTOVO-ORIENTOVANÉ PROGRAMOVANIE	167
16.1	PRETÍNajúCE ZÁLEŽITOSTI	168
16.2	PRÍKLAD: JEDNODUCHÉ MONITOROVANIE	168
16.3	BODY SPÁJANIA	172
16.4	ZÁKLADNÉ VLASTNOSTI JAZYKA ASPECTJ	172
16.5	BODOVÉ PRIEREZY	174
16.6	VIDENIA	182
16.7	KONTEXT BODU SPÁJANIA	185
16.8	MEDZITYPOVÉ DEKLARÁCIE	186
16.9	ABSTRAKTNÉ ASPEKTY	189
16.10	REFLEKTÍVNA PODPORA PRE BODY SPÁJANIA	190
16.11	INŠTANCIÁCIA ASPEKTOV	191
16.12	ASPEKTOVO-ORIENTOVANÁ IMPLEMENTÁCIA VZORU OBSERVER	196
	LITERATÚRA	201
	REGISTER	203

ZOZNAM OBRÁZKOV

2.1	Stret rytiera a obra.	5
2.2	Agregácia.	6
2.3	Dedenie — hierarchia obrov.	9
5.1	Hierarchia grafických útvarov.	55
5.2	Hierarchia grafických útvarov s abstraktnou triedou Utvár	56
5.3	Hierarchia grafických útvarov s rozhraniami.	60
13.1	Jednoduché okno.	128
13.2	Okno s tlačidlom.	130
13.3	Okno s dvomi tlačidlami.	131
14.1	Všeobecná asociácia medzi triedami.	141
14.2	Detaily triedy Kruh	142
14.3	Agregácia.	143
14.4	Asociačná rola.	143
14.5	Násobnosť vzťahu.	144
14.6	Príklady násobnosti vzťahu.	144
14.7	Generalizácia.	145
14.8	Abstraktné triedy a operácie.	145
14.9	Realizácia rozhrania.	146
14.10	Detailný diagram tried.	147
14.11	Použitie rozhrania a väzba závislosti.	148
14.12	Hierarchia obrov.	148
14.13	Združená šípka pre generalizáciu.	149
14.14	Vhniezdená trieda.	149
14.15	Diagram objektov.	149
14.16	Príklad diagramu prípadov použitia.	150
14.17	Predregistrácia témy.	151
14.18	Potvrdenie témy.	151
15.1	Model-View-Controller (podľa [Hel]).	155
15.2	Štruktúra vzoru Visitor.	157
15.3	Štruktúra vzoru Observer.	162

PREDHOVOR

Táto kniha vznikla na základe mojich prednášok z rovnomenného predmetu na Fakulte informatiky a informačných technológií Slovenskej technickej univerzity v Bratislave. Ako vo svojich prednáškach, tak aj v tejto knihe som sa snažil vystihnúť princípy objektovo-orientovaného programovania, ale aj poskytnúť úplný obraz jedného objektovo-orientovaného programovacieho jazyka (aj keď kniha predsa predpokladá základné vedomosti z programovacieho jazyka C alebo iného podobného procedurálneho jazyka). Nie náhodou je ním Java, ktorej popularita stále stúpa, a ktorej poznanie je v súčasnosti pre programátora veľkou výhodou.

Hovoriť dnes o objektovo-orientovanom programovaní a nespomenúť aspektovo-orientované programovanie by znamenalo zamlčať najperspektívnejší smer jeho rozvoja. Preto som jednu kapitolu venoval programovaciemu jazyku AspectJ, ktorý predstavuje reálne používané aspektovo-orientované rozšírenie Javy.

Chcel by som poďakovať recenzentom za pozorné preštudovanie rukopisu a cenné pripomienky, ktoré sa týkali aj trochu odvážnejšej terminológie (predovšetkým v oblasti objektovo-orientovaného a aspektovo-orientovaného programovania). Mojim cieľom bolo čo najviac sa priblížiť pôvodnej anglickej terminológii. V texte vždy upozorňujem na ďalšie termíny, ktoré sa používajú pre daný pojem.

Ďakujem Gratex IT Inštitútu, ktorý podporil vydanie tejto publikácie v rámci fondu GraFIIT.

Bratislava, 14. január 2008

Valentino Vranič

1 ÚVOD

Základným cieľom tejto knihy je oboznámiť čitateľa s objektovo-orientovaným programovaním. Splnenie tohto cieľa vyžaduje uvedenie čitateľa do určitého objektovo-orientovaného programovacieho jazyka. V prípade tejto knihy je to programovací jazyk Java.

Skutočné pochopenie určitého spôsobu programovania vyžaduje jeho praktické vyskúšanie v kontexte ostatných číť daného programovacieho jazyka. Preto v mnohom tento text zachádza do detailov Javy, ktoré sa bezprostredne nespájajú s objektovo-orientovaným programovaním, ako sú napríklad anonymné triedy, typické a rozsiahlo využívané v Jave.

Táto kniha sa nevenuje konštrukciám na riadenie vykonávania programu a operátorom, ktoré Java prakticky prevzala z jazyka C. Nevysvetľuje ani tvorbu vnorenej dokumentácie, tvorbu appletov a mechanizmus anotácií, ktoré však nie sú kľúčové pre objektovo-orientované programovanie.

Na druhej strane, objektovo-orientované programovanie je potrebné vnímať v širšom kontexte. Preto táto kniha pootvára dvere do troch ďalších oblastí. Objektovo-orientované programovanie je úzko prepojené s objektovo-orientovanou analýzou a návrhom, ktoré sa už takmer výlučne uskutočňujú v jazyku UML. Preto sa tento text dotýka aj jazyka UML a približuje čitateľovi tie jeho vlastnosti, ktoré patria do repertoára základných vedomostí objektovo-orientovaného programátora.

Kniha sa venuje aj objektovo-orientovaným vzorom, ktorých poznanie je predpokladom úspešného vývoja objektovo-orientovaných aplikácií.

Prostredníctvom tejto knihy čitateľ bude mať príležitosť dozvedieť sa aj o aspektovo-orientovanom programovaní. Ide o prístup, ktorý významne rozširuje možnosti objektovo-orientovaného programovania. V knihe je prezentovaný v súvislosti s jazykom AspectJ, ktorý plynulo nadväzuje na Javu, a ktorý sa používa už aj v praxi.

Pre lepšiu názornosť princípov objektovo-orientovaného programovania v texte sa prelína výklad Javy so všeobecnejším pohľadom na objektovo-orientované programovanie v kontexte objektovo-orientovaného vývoja softvéru ako takého:

- Kapitola 2 poskytuje vhľad do objektovo-orientovaného programovania.
- Kapitola 3 uvádza čitateľa do programovacieho jazyka Java.

- Kapitoly 4–6 vysvetľujú principiálne záležitosti objektovo-orientovaného programovania a ich realizáciu v Jave.
- Kapitoly 7–13 sú venované ďalším prvkom Javy a Java API (rozhrania aplikačného programovania), ktoré sa nedajú označiť priamo za objektovo-orientované, ale sú postavené na objektovo-orientovaných základoch jazyka a je nevyhnutné poznať ich pre plnohodnotné využitie tohto jazyka.
- Kapitoly 14–16 poniesú čitateľa za zdanlivé hranice objektovo-orientovaného programovania — k modelovaniu, vzorom a aspektom.

2 VHLAD DO OBJEKTIVO-ORIENTOVANÉHO PROGRAMOVANIA

Spolu s procedurálnym programovaním objektovo-orientované programovanie predstavuje najpoužívanejší prístup k programovaniu. Táto kapitola približuje objektovo-orientované programovanie prostredníctvom príkladu.

2.1 VZNIK OBJEKTIVO-ORIENTOVANÉHO PROGRAMOVANIA

Podľa Thomasa Kuhna k zmene paradigmy — prevládajúceho vedeckého názoru v danej oblasti — dochádza zlomom, teda revolúciou, nie postupne, evolúciou. K zlomu prichádza, keď sa problémy platnej paradigmy stanú neúnosnými. Klasickým príkladom takého zlomu je zmena z newtonovskej mechaniky na teóriu relativity [Kuh70].

Aj keď sa nástup objektovo-orientovaného programovania často vníma ako zlom paradigmy, korene objektovo-orientovaného programovania sú v skutočnosti v procedurálnom programovaní. Prvý objektovo-orientovaný jazyk, Simula 67, ktorý ako súčasť svojho názvu má rok vzniku, za základ mal procedurálny jazyk Algol 60. Autori Simuly 67, Ole-Johan Dahl a Kristen Nygaard, prví použili pojem objekt v zmysle objektovo-orientovaného programovania.¹

Programovací jazyk Smalltalk² predstavuje významný posun vo vývoji objektovo-orientovaného programovania. Tento jazyk je založený na postuláte „Všetko je objekt.“ a dôsledne ho uplatňuje, takže objektmi sú aj samotné riadiace konštrukcie (ako napríklad `if`). Kým v Simule 67 išlo predovšetkým o organizáciu kódu, v Smalltalku je dôraz na spolupráci objektov ako entít vo vykonávaní programu, ktorých manipulácia je veľmi pružná vďaka tomu, že ide o interpretovaný jazyk.

¹<http://staff.um.edu.mt/jskl1/talk.html>

²<http://www.smalltalk.org>

2.2 OBJEKTY A TRIEDY

Objektovo-orientované programovanie predstavuje programovanie pomocou objektov. Objekt je entita, ktorá má [Boo94]:

- stav
- správanie
- identitu

Stav objektu zahŕňa všetky vlastnosti objektu a ich hodnoty. Správanie objektu predstavuje konanie objektu pri zmenách stavu a aktivácii operácií, ktoré poskytuje. Identita objektu predstavuje jeho jednoznačnú identifikáciu.

Objektovo-orientovaný program sa uskutočňuje ako *interakcia objektov*. Správanie objektu v objektovo-orientovaných jazykoch so statickými typmi (ako je Java) definuje jeho *typ*. Typ objektu sa označuje ako *trieda* (class). V zdrojových textoch programov sa najčastejšie definujú triedy, a objekty pritom predstavujú ich inštancie. Preto sa niekedy zdá, že ide skôr o programovanie pomocou tried. Iný prístup je v dynamických objektovo-orientovaných jazykoch, v ktorých niekedy úplne absentuje pojem triedy ako šablóny pre tvorbu objektov a objekty sa vytvárajú priamo.

2.3 STAV A SPRÁVANIE OBJEKTU

Hovorili sme o objektoch, ich spolupráci a triedach ako šablónach pre tvorbu objektov. Pozrime sa na tieto pojmy bližšie prostredníctvom príkladu.

Predpokladajme, že vytvárame hru. Jeden z objektov v hre je `statocnyRytier`. Jeho stav je daný týmito vlastnosťami:

- `statocnyRytier.poloha` — kde sa `statocnyRytier` nachádza
- `statocnyRytier.energia` — akú `statocnyRytier` má energiu

Bodkou vyjadrujeme to, že dané vlastnosti patria spomínanému objektu. Vlastnosti objektu sa označujú ako *atribúty*.

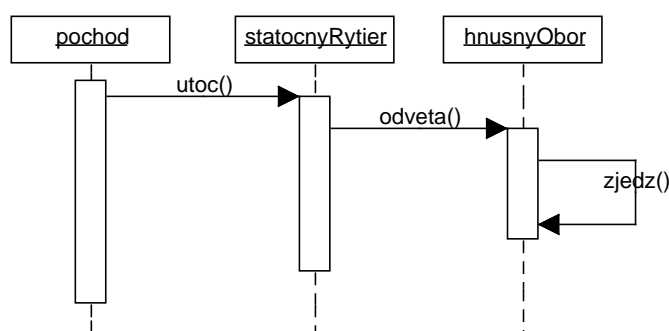
Správanie objektu `statocnyRytier` je dané operáciami, ktoré poskytuje. V našom príklade pôjde o jedinú operáciu: `statocnyRytier.utoc()`. Jej aktivácia spôsobí to, že `statocnyRytier` zaútočí na nepriateľa.

Ďalší z objektov v hre je `hnusnyObor`. Jeho stav je daný týmito vlastnosťami:

- `hnusnyObor.poloha` — kde sa `hnusnyObor` nachádza
- `hnusnyObor.energia` — akú `hnusnyObor` má energiu
- `hnusnyObor.hladny` — či je `hnusnyObor` hladný

Správanie objektu `hnusnyObor` je znovu len jedno: `hnusnyObor.odveta()`. Aktivácia tejto operácie spôsobí odvetu nepriateľovi, ktorý zaútočil na objekt `hnusnyObor`.

Stret rytiera a obra môžeme potom vnímať ako interakciu objektov, ako je to znázornené na obr. 2.1. Interakciu iniciuje objekt `pochod`, o ktorého details sa v tomto okamihu nezaujímame. Objekty interagujú prostredníctvom tzv. posielania *správ*, ktoré vlastne predstavuje volanie *operácií*.



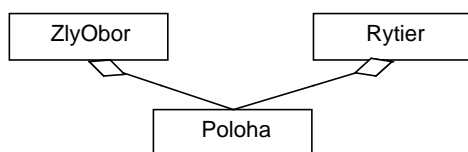
Obrázok 2.1: Stret rytiera a obra.

2.4 TYP OBJEKTU

Ako už bolo povedané, správanie objektu definuje jeho typ. Typ objektu sa označuje ako trieda (class). Napríklad `hnusnyObor` predstavuje jedného zo zlých obrov — jeho typ je daný triedou `ZlyObor`. Dalším príkladom je `statocnyRytier`, ktorý je objektom triedy `Rytier`, alebo `poloha`, ktorá je objektom triedy `Poloha`.

2.5 AGREGÁCIA

Triedy `ZlyObor` a `Rytier` obsahujú triedu `Poloha`. Táto skutočnosť je znázornená na obr. 2.2. Takýto spôsob spájania tried do väčších celkov sa označuje ako *agregácia*.



Obrázok 2.2: Agregácia.

Agregácia predstavuje spôsob tvorenia *hierarchie*: agregát, t.j. prvok, do ktorého sa zahŕňajú agregované prvky, je nadradeným prvkom vzhľadom na tieto prvky. Teda, v našom príklade triedy `ZlyObor` a `Rytier` sú nadradené triede `Poloha`.

2.6 ZAPUZDRENIE

Pokúsme sa bez zavádzania jazykových formalizmov načrtnúť implementáciu triedy `ZlyObor` — jej atribúty a operácie:

```
class ZlyObor {
    Poloha poloha;
    int energia;
    boolean hladny;

    void odvetta(Rytier r) {
        if (hladny)
            zjedz(r);
    }

    void zjedz(Rytier r) {
        int e = zistiEnergiu();
        r.uberEnergiu(e);
        zvyshEnergiu(e);
    }

    int zistiEnergiu() {
        return energia;
    }

    void zvyshEnergiu(int i) {
        energia = energia + i;
    }
}
```

```
void znizEnergiu(int i) {
    energia = energia - i;
}
}
```

Jedným z kľúčových princípov objektovo-orientovaného programovania je *zapuzdrenie* (encapsulation), označované ešte aj ako skrývanie informácií. Tento princíp hovorí, že implementácia objektu má zostať skrytá a prístup k objektu má byť zabezpečený prostredníctvom *rozhrania* (interface), ktoré tvoria vybrané operácie.

Napríklad energia zlych obrov sa nemení priamo siahnutím na atribút `energia`, ale prostredníctvom operácií `zvysEnergiu()` a `znizEnergiu()`. Cudzie objekty by nemali mať možnosť prístupu k tomuto atribútu, na čo — ako uvidíme v nasledujúcej kapitole — jestvujú príslušné jazykové konštrukcie. V uvedenom príklade však pre jednoduchosť tieto konštrukcie nie sú uvedené a princíp zapuzdrenia je porušený aj z iného hľadiska: na zmenu polohy by mali tiež byť poskytnuté zodpovedajúce operácie.

2.7 DEDENIE

`ZlyObor` je len jeden možný druh obrov. Niekedy sa potrebujeme vyjadriť o obroch vo všeobecnosti bez ohľadu na špecifické druhy obrov. *Zovšeobecnením* (generalization) obrov by v našom príklade mohla byť trieda `Obor`, ktorá zahŕňa spoločné vlastnosti všetkých obrov:

```
class Obor {
    boolean hladny;
    int energia;

    void odveta(Rytier r) {
        r.znizEnergiu(1);
    }
    int zistiEnergiu() {
        return energia;
    }
    void zvysEnergiu(int i) {
        energia = energia + i;
    }
    void znizEnergiu(int i) {
        energia = energia - i;
    }
}
```

ZlyObor potom predstavuje *špeciálizáciu* triedy Obor:

```
class ZlyObor extends Obor {  
  
    void odveta(Rytier r) {  
        if (hladny)  
            zjedz(r);  
    }  
    void zjedz(Rytier r) {  
        int e = zistiEnergiu();  
        r.uberEnergiu(e);  
        zveysEnergiu(e);  
    }  
}
```

ZlyObor *dedí správanie a štruktúru* triedy Obor. Pod štruktúrou väčšinou rozumieme atribúty alebo kód ako taký, kým pod správaním rozumieme funkcionality, ktorú implementujú operácie. Inak povedané, trieda ZlyObor *rozširuje a konkretizuje* triedu Obor — pridáva nové detaily do *abstrakcie* obra. Preto je v Jave kľúčové slovo pre dedenie práve **extends**.

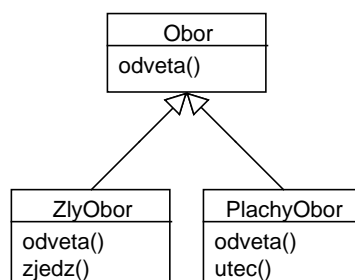
Ďalším príkladom obra môže byť PlachyObor, ktorý je, ako vidíme z operácie `odveta()`, naozaj plachý:

```
class PlachyObor extends Obor {  
    void utec() {  
        . . .  
    }  
    void odveta(Rytier r) {  
        utec();  
    }  
}
```

Dôležité je uvedomiť si, že popri agregácii, dedenie predstavuje ďalší spôsob tvorenia hierarchie v objektovo-orientovanom programovaní. Dokonca, keď sa povie len „hierarchia tried“, myslí sa na hierarchiu dedenia. Hierarchiu obrov názorne ukazuje obr. 2.3.

2.8 POLYMORFIZMUS

V dedení zďaleka nejde len o zdieľanie spoločného kódu ako sa na prvý pohľad môže zdať. Oveľa dôležitejšie je, že objekt každej triedy, ktorá je v hierarchii dedenia



Obrázok 2.3: Dedenie — hierarchia obrov.

nadradená danej triede, je zároveň aj objektom tejto nadradenej triedy. Presnejšie povedané, typ definovaný podtriedou je podtypom typu definovaným nadtriedou.

V našom príklade aj objekty typu `PlachyObor`, aj objekty typu `ZlyObor` sú typu `Obor` — preto môžeme premenným typu `Obor` priradiť objekty týchto podtypov:

```

Obor o1;
Obor o2;

o1 = new ZlyObor ();
o2 = new PlachyObor ();
  
```

Na vytvorenie objektov bolo použité kľúčové slovo **new**. Proces vytvárania objektov je detailne vysvetlený v nasledujúcej kapitole a v tomto okamihu nie je dôležitý.

Nad objektmi `o1` a `o2` môžeme volať všetky operácie definované pre typ `Obor`. Aké však bude správanie v prípade vyvolania operácie, ktorú každý z objektov definuje inak? Príkladom je operácia `odveta()`. Za predpokladu, že premenná `r` predstavuje rytiera, ktorý na obra zaútočil, pri odvete obra `o1`

```
o1.odveta(r);
```

rytier bude zjedený, kým pri odvete obra `o2`

```
o2.odveta(r);
```

sa mu nestane nič, lebo obor `o2` utečie, ako je definované v operáciách `odveta()` príslušných tried, a nie v operácii `odveta()` triedy `Obor`.

Výhoda tohto javu, ktorý sa označuje ako *polymorfizmus* začína byť zrejماً pri vyšších počtoch objektov a dopredu neznámom počte typov. Tak napríklad pre hordu sto

obrov, ktorí nemusia byť len typov `ZlyObor` a `PlachyObor`, pri ich strete s rovnako početnou rytierskou výpravou môžeme napísať jednoducho

```
for (int i = 0; i < 100; i++) {  
    obor[i].odveta(r[i]);  
}
```

a vieme, že každý obor urobí práve to, čo je pre jeho typ charakteristické. Pre úplnosť treba povedať, že výrazom `obor[]` označujeme pole premenných typu `Obor`, podobne ako v jazyku C, čo bude podrobne vysvetlené v nasledujúcej kapitole. Všimnite si veľmi dôležitú skutočnosť, že nepotrebujeme žiadne podmienené príkazy, a že v prípade zmeny v typoch obrov tento kód vôbec nie je potrebné upravovať.

2.9 SUMARIZÁCIA

Skôr ako začneme rozoberať jednotlivé záležitosti objektovo-orientovaného programovania a špecifiká programovacieho jazyka Java, táto kapitola poskytla vhľad do objektovo-orientovaného programovania. Pojmy v tejto kapitole boli definované s istou voľnosťou a príklady kódu boli prezentované bez presnej definície syntaxe a sémantiky, ale ak sa podarilo aspoň trochu demystifikovať problematiku objektovo-orientovaného programovania, kapitola splnila svoj účel.