Extrémne programovanie a iné agilné metodológie

Zimný semester 2007/08

Ing. František Gyárfáš, PhD.

Katedra aplikovanej informatiky

gyarfas@ii.fmph.uniba.sk

http://www.ii.fmph.uniba.sk/~gyarfas/

Čo je to refaktorizácia

```
void vypis(double sirka, double vyska, double hlbka){
     double temp = sirka * vyska;
     cout << "plocha = " << temp;
     temp = sirka * vyska * hlbka;
     cout << "obsah = " << temp;
}
```

Dôvod: Použitie tej istej lokálnej premennej pre rôzne účely ohrozuje čitateľnosť kódu a je to časovaná bomba pre budúce chyby.

```
void vypis(double sirka, double vyska, double hlbka){
     double plocha = sirka * vyska;
     cout << "plocha = " << plocha;
     double obsah = sirka * vyska * hlbka;
     cout << "obsah = " << obsah;
}
```

Dôvody refaktorizácie

Aké sú dôvody refaktorizácie:

- hanba, ako ten kód vyzerá
- nechuť prehrabávať sa desiatkami vetiev tej istej podmienky
- odpor opakovať tú istú konštrukciu desiatky krát
- hrozba kritiky
- chybovosť kódu
- neefektívnosť kódu
- nemožnosť ďalšej refaktorizácie

Definícia

Definícia refaktorizácie

Refaktorizácia je zmena vo vnútornej štruktúre softvéru, vedúca k jeho ***ľahšiemu pochopeniu*** *a uľahčeniu jeho ďalších úprav* ***bez zmeny*** *jeho vonkajšieho správania sa.*

Prečo je potrebné refaktorovať?

- pri písaní programov je jednoduchšie myslieť iba na funkčnosť kódu
- podobnosť kódu sa objavuje až pri ďalšom použití
- zovšeobecnenie nasleduje až po špeciálnom
- robenie poriadku je ľudskému mysleniu prirodzené
- písanie čistého kódu je omnoho ľahšie, keď už poznáme (a môžeme testovať) jeho funkčnosť
- refaktorizácia pomáha hľadať chyby

Inteligentný softvér

Zákon inteligentného softvéru

Kód, ktorému rozumie počítač, dokáže napísať hocikto.

Dobrí programátori píšu kód, ktorému rozumejú ľudia.

Hazardné hry

Varovanie

Refaktorovanie je riskantné.

Pravidlo znižovania rizika

Prv, než začnete refaktorovať, musíte mať pre daný kód sadu testov, ktoré sa vyhodnocujú samostatne.

Pravidlo minimálnych krokov

Refaktorizáciu robte pomocou čo najmenších krokov. Ľahšie sa vracia späť.

Najjednoduchší návrh

XP presadzuje najjednoduchší návrh programov. Tento návrh dosahuje splnením štyroch obmedzení v prioritnom poradí:

- program musí komunikovať so všetkým, s čím chceme, aby komunikoval
- program nesmie obsahovať duplicitný kód
- program by mal mať minimálny počet tried
- program by mal mať minimálny počet metód

Všetko v programe musí mať účel: **výpočetný** alebo **komunikačný**. Čo nespĺňa ani jeden účel, musí ísť von.

Ak dva kusy kódu spĺňajú ten istý účel, treba ich spojiť.

Do tretice

Zákon trikrát a dosť

Keď niečo robíš prvý raz, jednoducho to urobíš.

Keď niečo robíš druhý raz, rozmýšľaš o duplicite a nakoniec to predsa len urobíš.

Ak by si to mal urobiť tretí raz, refaktoruj.

Kedy refaktorovať

Babičkine pravidlo

Keď to smrdí, vymeň to.

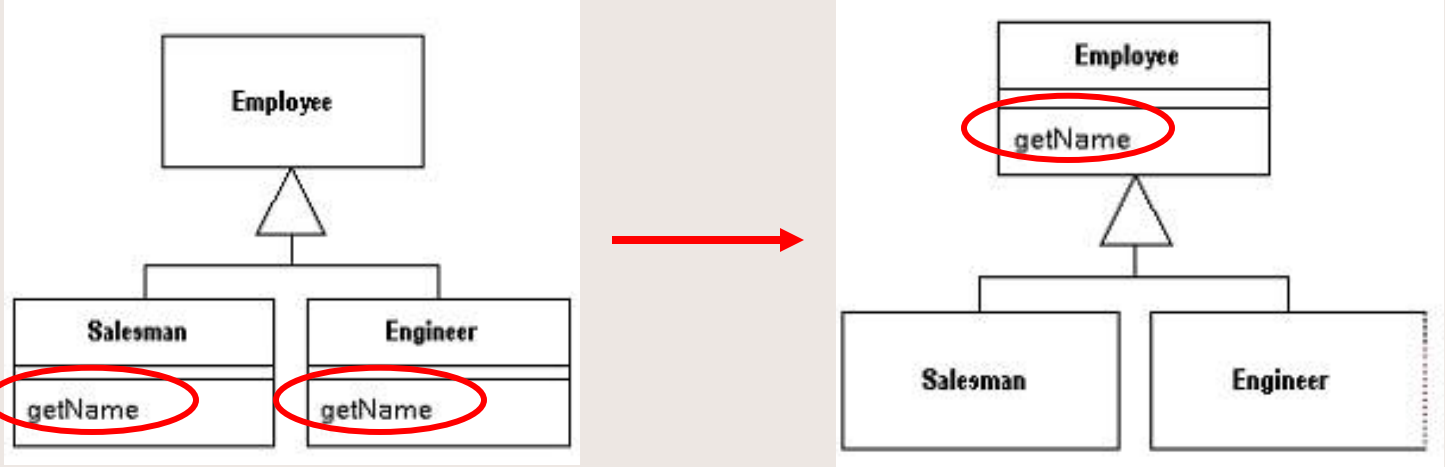
- duplicitný kód
- príliš dlhá metóda
- príliš veľká trieda
- dlhý zoznam parametrov
- protichodné zmeny
- zmeny rozptýlené na mnohé miesta

Kedy refaktorovať – II.

- chýbajúce schopnosti
- dátové zhluky
- príkazy switch
- paralelné hierarchie dedičnosti
- lenivá trieda (ktorá skoro nič nerobí)
- špekulatívna obecnosť (veci, o ktorých ste si mysleli, že sa zídu a nezišli)
- zreťazené správy (objekt žiada iný objekt, ktorý žiada iný objekt, ...)
- prílišná dôvernosť (dvoch tried)
- výskyt komentáru často signalizuje nejasný kód

Duplicitný kód: rodičovská trieda

Duplicitný kód je napr. vtedy, ak sa ten istý výraz vyskytuje v dvoch súrodeneckých podtriedach tej istej triedy. Vtedy stačí vyňať metódu a vložiť ju do rodičovskej triedy.



Duplicitný kód: vyňatie metódy

Podobný duplicitný kód je, ak sa vyskytuje ten istý výraz v dvoch metódach tej istej triedy.

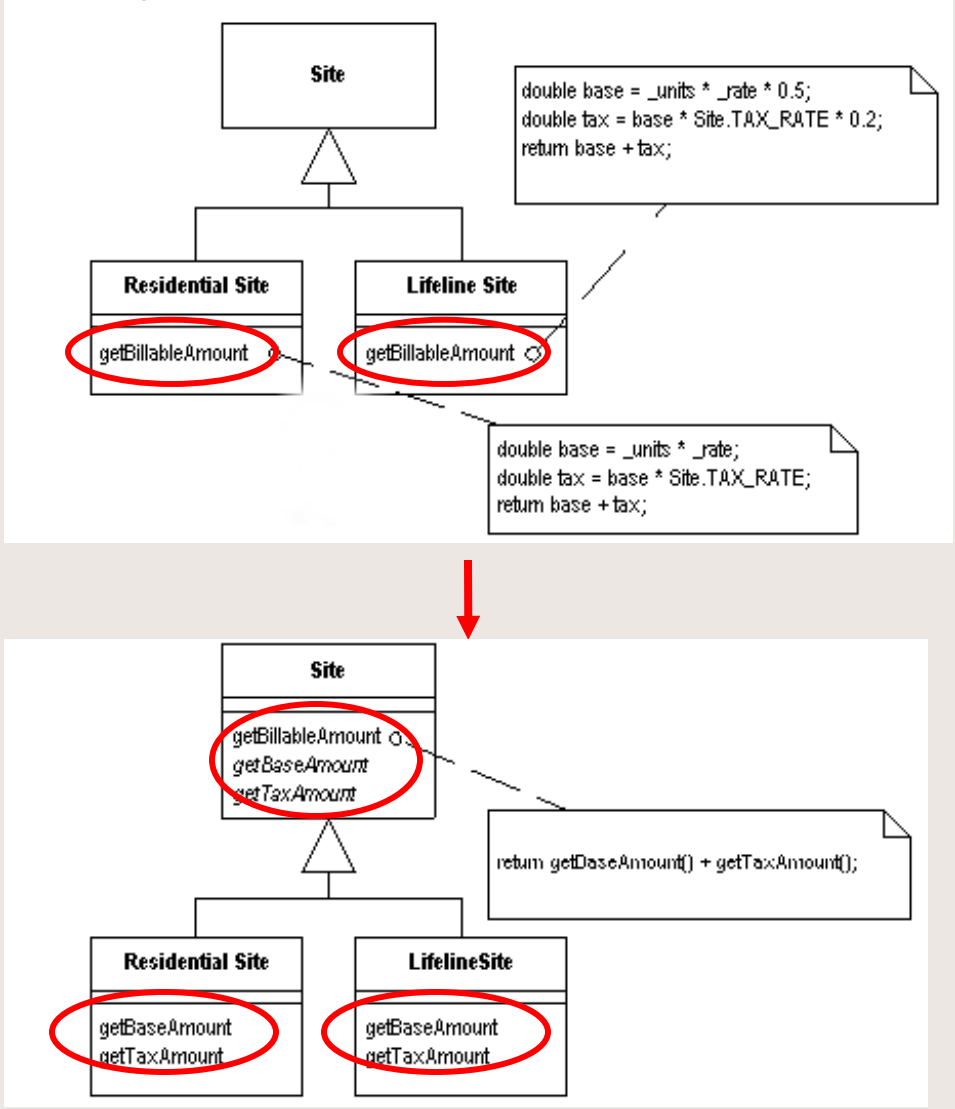
```
void printOwing(){
   printBanner();
   System.out.println ("name:" + _name);
   System.out.println ("amount " + getOutstanding());
}
```

Vyňatie metódy umožňuje volanie z viacerých miest.

```
void printOwing() {
   printBanner();
   printDetails(getOutstanding());
}

void printDetails (double outstanding) {
   System.out.println ("name:" + _name);
   System.out.println ("amount " + getOutstanding());
}
```

Duplicitný kód: abstraktné triedy



Príliš dlhá metóda

Kedysi bolo volanie funkcie drahé. Preto sa programátori snažili šetriť počet funkcií a písali dlhé funkcie a metódy.

Tieto dôvody pominuli.

- Dlhé metódy sa ťažšie chápu.
- Veľmi ťažko sa testujú.
- Ešte ťažšie sa používajú rôzne účely.
- U krátkych jednoúčelových metód stačí výstižný názov.

Vo väčšine prípadov stačí na odstránenie príliš dlhej metódy technika vyňatia metódy.

Príliš dlhá metóda: Odstráň nepotrebné premenné

```
double basePrice = anOrder.basePrice();
return (basePrice > 1000)
```

```
return (anOrder.basePrice() > 1000)
```

Príliš dlhá metóda: Vytvor objekt pre parametre

```
spocitajPrijmy(odDatumu, poDatum);
spocitajVydaje(odDatumu, poDatum);
```

Ak existuje skupina parametrov, ktoré súvisia a používajú sa spolu, tak je dobré vytvoriť pre ne triedu.

```
spocitajPrijmy(casovyInterval);
spocitajVydaje(casovyInterval);
```

Príliš dlhá metóda: Dekompozícia zložitej podmienky

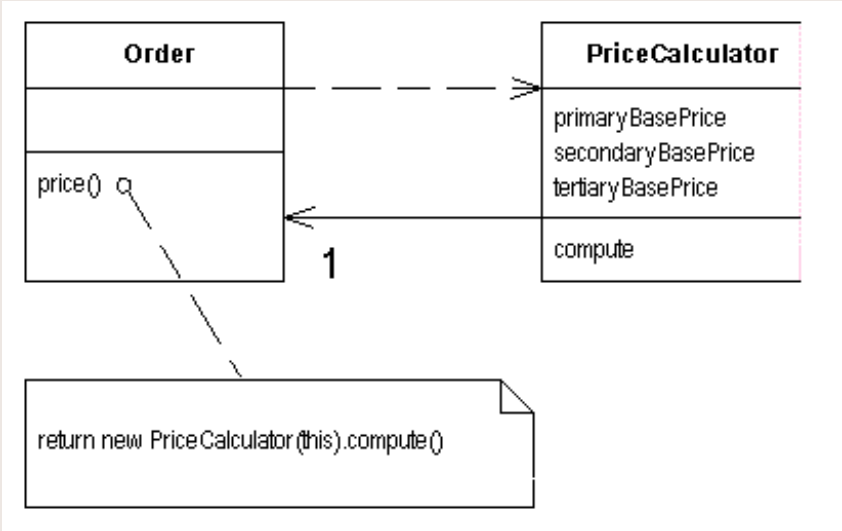
```
if (date.before (SUMMER_START) || date.after(SUMMER_END)){
   charge = quantity * _winterRate + _winterServiceCharge;
}
else {
   charge = quantity * _summerRate;
}
```

```
if (notSummer(date)) {
   charge = winterCharge(quantity);
}
else {
   charge = summerCharge (quantity);
}
```

Nahradenie metódy triedou

Vytvorenie z metódy objekt tak, že sa lokálne premenné metódy stanú položky objektu.

```
class Order...
   double price(){
         double primaryBasePrice;
         double secondaryBasePrice;
         double tertiaryBasePrice;
         // long computation;
                  ...
   }
```



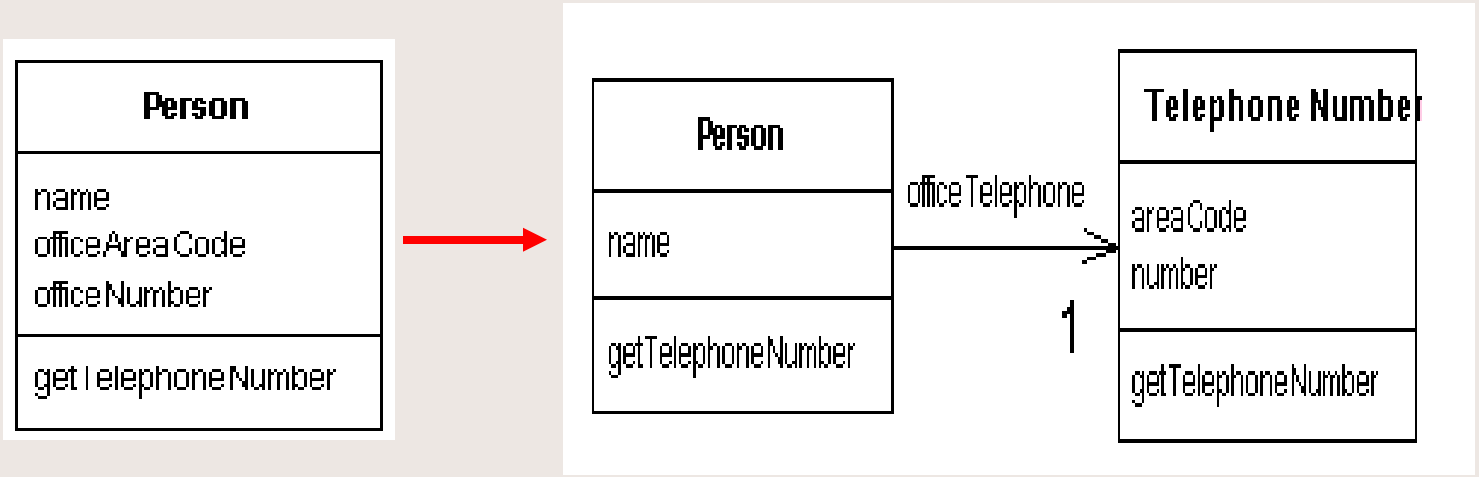
Príliš veľká trieda

Kto by čítal kód príliš dlhej triedy? Analyzoval? Pochopil?

Príliš veľké triedy sú semenište duplicitného kódu.

Príliš veľká trieda: vyňatie triedy

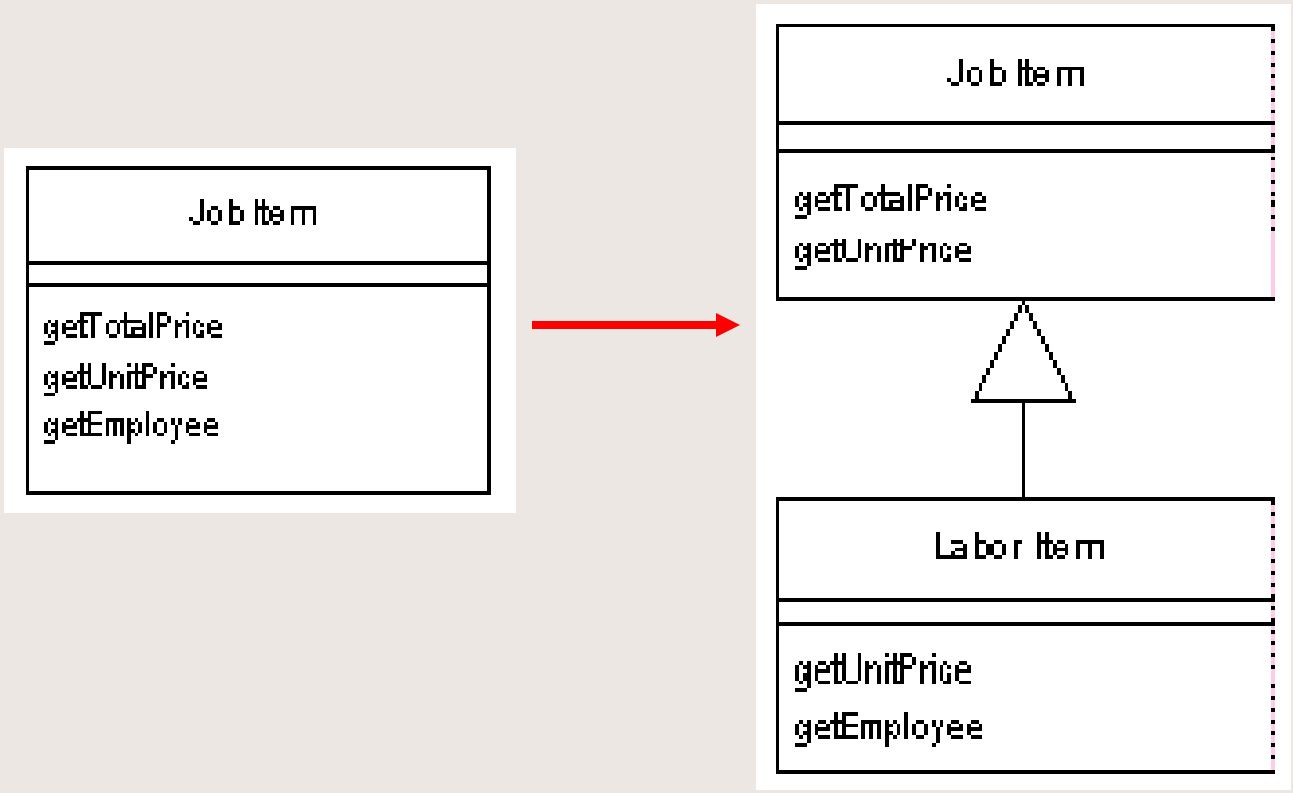
Jedna technika je vyňatie triedy.



Príliš veľká trieda: Vyňatie podtriedy

Trieda niekedy obsahuje prvky, ktoré používajú iba niektoré jej inštancie.

V takom prípade je vhodné pre túto podmnožinu prvkov vytvoriť podtriedu.



Dlhý zoznam parametrov

Vstupné parametre sa stali alternatívou globálnym premenným, ktoré boli prehlásené za zlo najvyššieho kalibru.

Dnes to už tak celkom neplatí, pretože metódy majú k dispozícii objekty svojej triedy a môžu o potrebné údaje iné triedy požiadať.

Príliš dlhý zoznam vstupných parametrov je nepohodlný a neprehľadný. Preto je snaha znižovať počet parametrov.

Dlhý zoznam parametrov:
Nahradenie parametra metódou

```
int basePrice = quantity * itemPrice;
double discountLevel = getDiscountLevel();
double finalPrice = discountedPrice(basePrice, discountLevel);
```

Ak môže metóda získať priamo hodnotu parametra, nie je potrebné jej túto hodnotu posielať ako parameter.

```
int basePrice = quantity * itemPrice;
double finalPrice = discountedPrice(basePrice);
```

Dlhý zoznam parametrov:
Predávanie celého objektu

```
int min = dennaTeplota.getMin();
int max = dennaTeplota.getMax();
mojPlan = plan.rozmedzie(min, max);
```

Namiesto vytvárania lokálnych premenných stačí metóde predať samotný objekt, nech si prečíta sama, čo potrebuje.

```
mojPlan = plan.rozmedzie(&dennaTeplota);
```

Rozptýlené úpravy

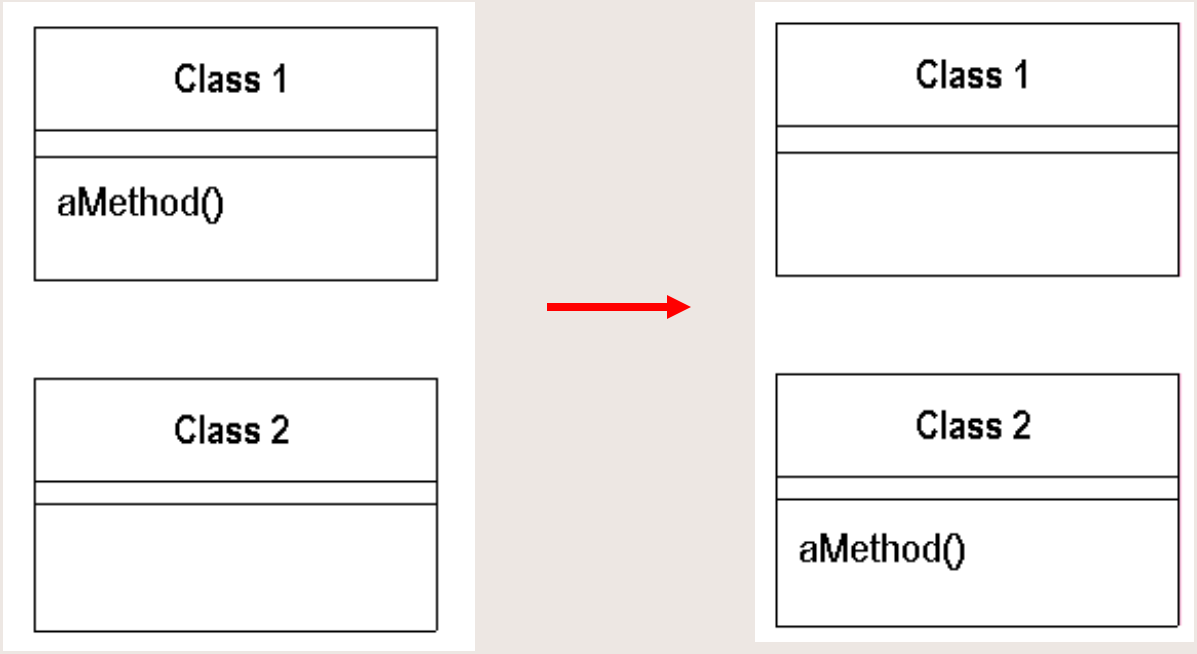
Niekedy zistíte, že pri každej zmene musíte previesť celý rad zmien v rôznych triedach, čo je ťažké nájsť aj realizovať.

Vtedy je často výhodné presunúť všetky zmeny do jednej triedy.

Ak sa taká trieda nenájde medzi existujúcimi, vytvorte novú.

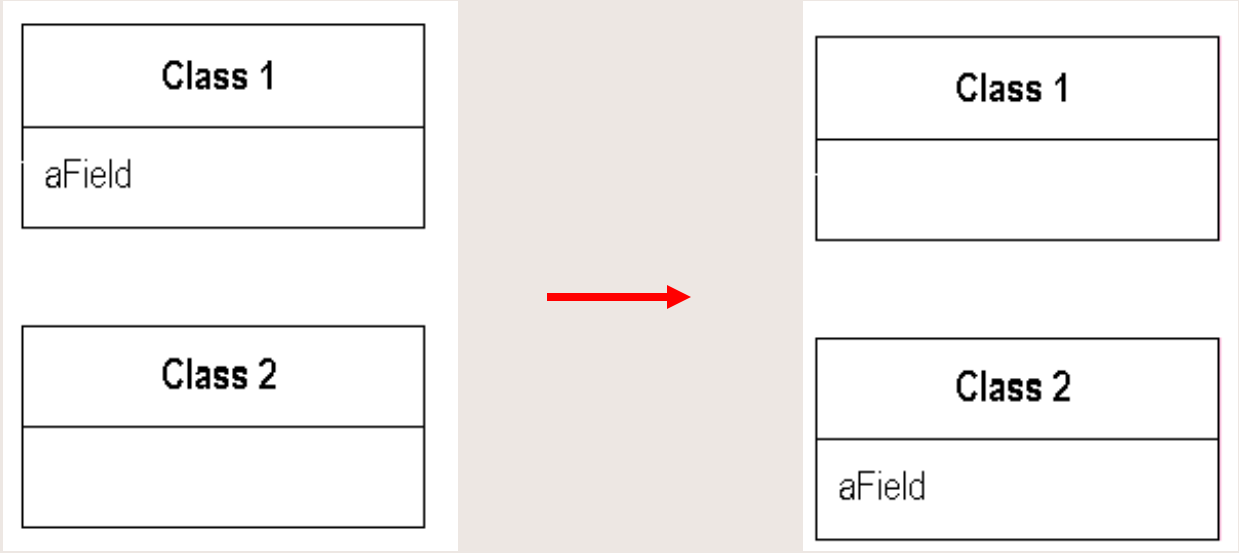
Rozptýlené úpravy: Presunutie metódy

Ak metóda spolupracuje viac s prvkami inej triedy, presuňte ju tam. Pôvodnú metódu zmeňte na púhe volanie, alebo ju odstráňte úplne.



Rozptýlené úpravy: Presunutie položky

Umiestnite položku tam, kam patrí.

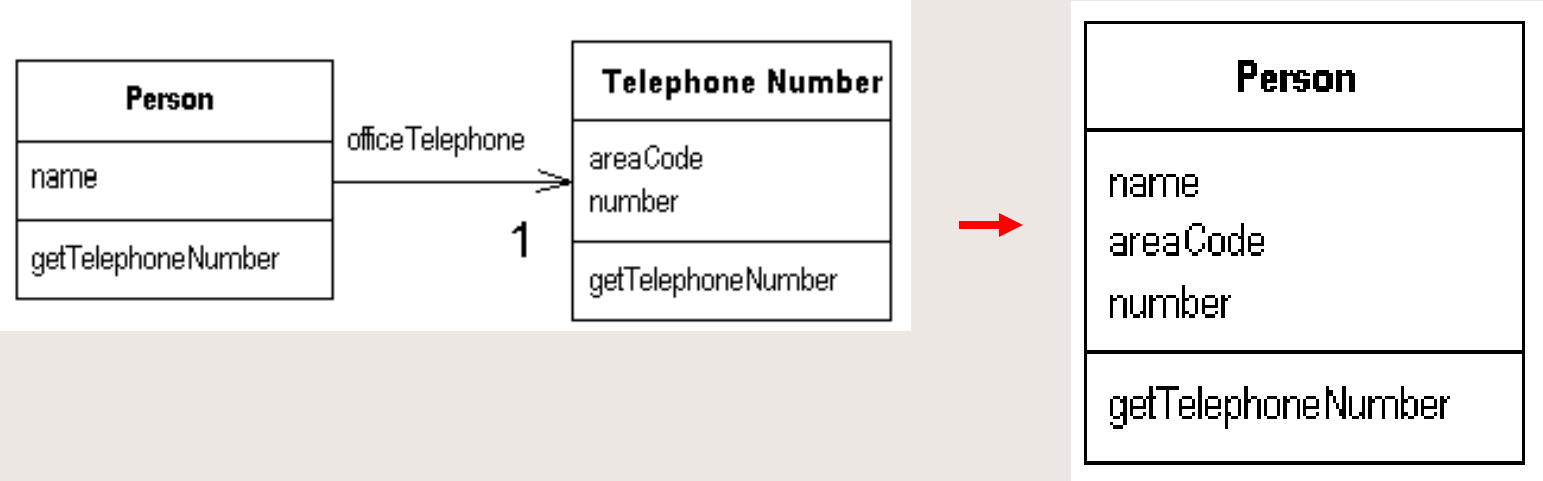


Rozptýlené úpravy: Odstránenie triedy

Triedu treba odstrániť, ak nevykazuje dostatočnú činnosť a jej samostatná existencia nemá opodstatnenie.

Táto strata činnosti je často výsledkom iného refaktorovania, ktoré odstránilo z triedy toľko metód, že v nej temer nič neostalo.

V takom prípade presuňte všetky jej prvky do inej triedy a zrušte ju.



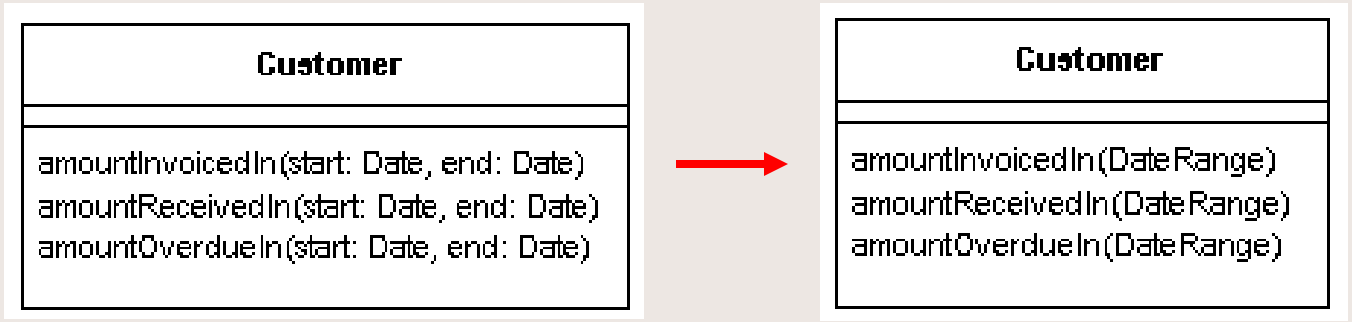
Dátové zhluky

Dátové položky sa často správajú ako deti: radi sa potĺkajú v skupinkách.

Často objavíte na rôznych miestach programu skupiny dát. Niekedy spolu súvisia aj sémantický (zoznam parametrov súboru), ale nie vždy.

V takých prípadoch sú tieto dáta zrelé na to, aby ste pre ne vytvorili objekt, ktorý by ich prirodzene združoval a obhospodaroval.

Dátové zhluky : Zavedenie objektu pre parametre



Dátové zhluky : Nahradenie pola objektom

Ak máme pole, v ktorom sa vyskytujú prvky rôzneho významu, je dobré nahradiť ich objektom, v ktorom bude mať každý prvok samostatnú položku.

```
String[] data = new String[3];
data[0] = "Jozef";
data[1] = "34";
```

```
Osoba student = new Osoba();
student.setMeno("Jozef");
student.setVek("34");
```

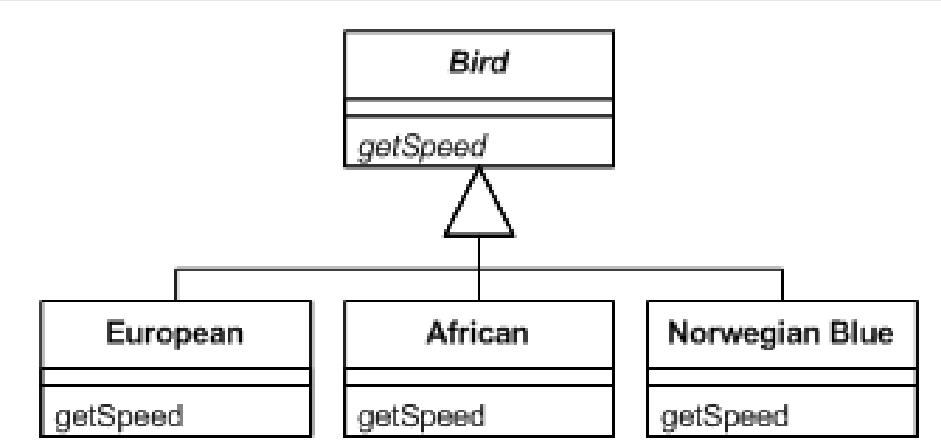
Príkazy switch (a podobné)

Jeden z najzreteľnejších znakov objektového programovania je vzácny výskyt príkazov `switch`. Dôvodom je najmä duplicita príkazov vo viacerých vetvách.

Najlepším prístupom k riešeniu tejto duplicity je použitie princípu polymorfizmu. To znamená, že ak objekty menia svoje správanie sa v závislosti na svojich typoch, nie je potrebné zahrňovať túto podmienku explicitne.

Príkazy switch: Náhrada podmienky polymorfizmom

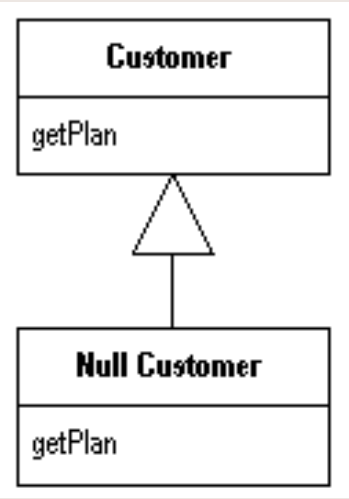
```
double getSpeed() {
   switch (_type) {
     case EUROPEAN:
         return getBaseSpeed();
     case AFRICAN:
         return getBaseSpeed() - getLoadFactor() * _numberOfCoconuts;
     case NORWEGIAN_BLUE:
         return (_isNailed) ? 0 : getBaseSpeed(_voltage);
   }
   throw new RuntimeException ("Should be unreachable");
 }
```



Príkazy switch: Zavedenie objektu null

Veľmi často sa v programoch testuje, či hodnota nejakého objektu je `null`. V takých prípadoch je výhodnejšie netestovať, ale vytvoriť objekt `null`.

```
if (customer == null)
        plan = BillingPlan.basic();
else
        plan = customer.getPlan();
```



Príkazy switch: Náhrada parametra explicitnými metódami

```
void setValue(String name, int value) {
  if (name.equals("height")) {
        height = value;
        return;
  }
  if (name.equals("width")) {
        width = value;
        return;
  }
  Assert.shouldNeverReachHere();
}
```

```
void setHeight(int arg) {
   height = arg;
}
void setWidth(int arg) {
   width = arg;
}
```