Dědičnost

Tomáš Pitner, upravil Marek Šabo

Dědičnost

- V realitě často vidíme, že objektové třídy jsou **podtřídami**, speciálními případy jiných:
  - všechny objekty podtřídy jsou zároveň objekty nadtřídy, např. každý objekt typu (třídy) `DogKeeper` je současně typu `Person` nebo
  - např. každý objekt typu (třídy) `Dog` je současně typu `Pet`
- Někdy to chce realitu trochu zjednodušit, např. zde předpokládat, že v našem výseku reality neexistují psi "nedomácí".

Podtřída

- **Podtřída** (říká se též *potomek*, *dceřinná třída*, *subclass*) je tedy specializací, zjemněním své *nadtřídy* (*předka*),
- přebírá její vlastnosti,
- zpravidla přidává další, čímž svou nadtřídu *rozšiřuje* (`extends`).

V Javě je *každá* uživatelem definovaná třída potomkem nějaké jiné — neuvedeme-li předka explicitně, je předkem vestavěná třída `Object`.

Správné použití

- Dědičnost je správně použita jen tehdy, když opravdu můžeme říci, že každý objekt podtřídy je současně objektem nadtřídy,
- např. `DogKeeper` je současně `Person`.
- Pokud to realitě neodpovídá, není dědičnost na místě.
- Často se pak popužívá *skládání* (kompozice) objektů, kdy objekt nedědí, ale nese odkaz na jiný objekt.

Proč se dědičnost používá

- Abychom zohlednili konceptuální vztah *obecnější* vs. *speciálnější* typ.
- Abychom se pomocí toho vyhnuli *opakování kódu* a dosáhli *znovupoužití* (= kód metod a atributů se podědí, nemusí jej znovu psát).
- Většinou by mělo platit oboje, aby mělo smysl dědičnost použít.

Terminologie dědičnosti

- *Nadtřídě* (superclass) se také říká "bezprostřední předek", "rodičovská třída"
- *Podtřídě* (subclass) se také říká "bezprostřední potomek", "dceřinná třída"
- Dědění může mít i více "generací", např.

- `Person` ← `Employee` ← `Manager`
- Osoba je rodičovskou třídou zaměstnance, ten je rodičovskou třídou manažera.
- Přeneseně tedy předkem (nikoli bezprostředním) manažera je člověk.

Jak zapisujeme dědění

- Klíčovým slovem `extends`:

```
public class Employee extends Person {
   // ... popis vlastností (proměnných, metod...)
   // zaměstnance navíc oproti (obecnému) člověku...
}
```

Dědičnost a vlastnosti tříd (1)

- Jak víme, třídy popisují skupiny objektů podobných vlastností.
- Třídy mohou mít tyto skupiny **vlastností**:
  - *Metody* - procedury/funkce, které pracují (především) s objekty této třídy
  - *Atributy* - pojmenované datové prvky (hodnoty) uchovávané v každém objektu této třídy
- Vlastnosti jsou ve třídě "schované", tzv. **zapouzdřené** (encapsulated)
- Co v tomto přináší dědičnost?

Dědičnost a vlastnosti tříd (2)

Dědičnost (alespoň v javovém smyslu) znamená, že dceřinná třída (podtřída, potomek):

1. má (tj. zdědí) *všechny* vlastnosti (metody, atributy) nadtřídy,
2. některé zděděné vlastnosti v potomkovi může *měnit* (překrytím metody),
3. případně v potomkovi *přidává* další vlastnosti (metody, atributy).

Příklad s `Account`

Cíl: vylepšit třídu `Account` (demo `05_Account`)

- Zdokonalíme náš příklad s účtem tak, aby si účet "hlídal", kolik se z něj převádí peněz, aby zůstatek nebyl nižší než povolený minimální
- Realizujeme tedy změnu jedné z vlastností — metody `debit` — pomocí jejího *překrytí* (overriding)
- Zdokonalenou verzi třídy `Account` nazveme `CheckedAccount`

Třída Account

```
public class Account implements Informing {
   private int balance;
   ...
   public boolean debit(int amount) {
      if(amount <= 0) return false;
      balance -= amount;
      return true;
   }
}
```

Třída CheckedAccount

```
public class CheckedAccount extends Account {
   // now it remembers also minimal balance
   private int minimalBalance;
   public CheckedAccount(Person owner, int minBal, int initBal) {
      // original initialization of Account
      super(owner, initBal);
      if(initBal < minBal) { // enough initial balance?
         throw new IllegalArgumentException("initial balance < minimal");
      }
      this.minimalBalance = minBal;
   }
   public boolean debit(int amount) {
      // min. balance is checked
      if(getBalance() - amount >= minimalBalance) {
         return super.debit(amount);   // if kept, the debit is deducted
      } else return false;             // otherwise not
   }
}
```

Co tam bylo nového

- Klíčové slovo `extends` značí, že třída `CheckedAccount` je potomkem/podtřídou/rozšířením/dceřinnou třídou (*subclass*) třídy `Account`.
- Konstrukce `super.metoda(⋯);` značí, že je volána metoda rodičovské třídy/předka/nadtřídy, tedy třídy `Account`.
- Kdyby se to `super` nepoužilo, zavolala by se metoda `debit` třídy `CheckedAccount` a program by se zacyklil!