Java 2. Zapouzdření a dědičnost

1 Zapouzdření

Zapouzdření je jedna ze tří základních charakteristik OOP. Podstatou je sloučení vlastností (atributů) a činností (metod), které logicky patří k sobě, do jediného celku, tedy objektu.

Důležitým prvkem zapouzdření je ochrana dat a řízení přístupu k nim. V Javě je nutné u každé položky ve třídě specifikovat práva přístupu. Ta mohou být **public**, **protected**, **private** nebo tzv. friend (přátelský přístup pro celý balík – package), kdy se neuvádí žádný specifikátor.

Kvůli ochraně dat je dobrým zvykem chránit atributy alespoň na úrovni protected. Veřejné atributy jsou nebezpečné, protože mohou vést k poškození dat nevhodným zacházením ze strany uživatelů objektu.

Veřejné rozhraní objektu by mělo být tvořeno pouze pomocí veřejných funkcí.

1.1 Getter a setter

Getter je metoda vracející hodnotu nějakého atributu. Je zvykem ji pojmenovat podle vzoru *getJmenoAtributu()*.

Pomocí takových metod se dají realizovat i atributy, které ve skutečnosti nejsou ukládány, ale dají se dopočítat z jiných atributů (např. ukládám atribut poloměr kruhu, ale můžu mít metodu getObvod(), která tu hodnotu na vyžádání dopočítá).

Setter je analogická metoda, která slouží pro nastavování hodnot atributů. Volí se pro ni název *setJmenoAtributu()*.

Tyto metody obvykle spolu s nastavováním hodnoty řeší různé kontroly správnosti. Například u kruhu mi nedovolí zadat záporný poloměr.

Předpony get a set jsou zažité a není dobré je například překládat do češtiny. Různá IDE často umí tyto metody automaticky generovat (obvykle přes kontextové menu), případně je rozpoznávat a při psaní kódu je rovnou nabízet.

V následující ukázce si všimněte speciálního zacházení s atributem stred, který je objektového typu Bod. Protože objektové proměnné jsou v Javě reference (ukazatele), není dobré vracet objekty přímo. Uživatel by totiž získal referenci na originál nacházející se uvnitř aktuálního objektu a mohl by měnit vnitřní stav (zde posouvat střed kruhu).

```
public class Kruh {
  protected Bod stred;
  protected float polomer;

  public Bod getStred()
  { return new Bod(stred); } // !!
  public float getPolomer()
  { return polomer; }
  public float getObvod()
  { return Math.PI*polomer*polomer; }

  public void setStred(Bod stred)
  { this.stred = new Bod(stred); }
  public void setPolomer(
    float polomer)
  {
    if (polomer > 0)
    this.polomer = polomer;
  }
}
```

2 Dědičnost

Dědičnost je další typickou charakteristikou OOP. Umožňuje rozšiřovat funkčnost existujícítřídy. Vytváří mezi nimi vztah předek–potomek. Odvozená třída, potomek, pak dědí od předka všechny atributy i metody a přidává k nim své vlastní, to vše bez nutnosti mít k dispozici zdrojový kód výchozí třídy. Díky tomu a řízení práv přístupu, je rozšiřování OO knihoven mnohem pohodlnější než v jazyce C.

Objekt typu potomek v sobě vždy obsahuje atributy a metody předka, plus atributy a metody přidané třídou potomka. Potomek má automaticky přístup k veřejným a chráněným položkám předka. Privátní položky jsou v potomkovi přítomny také, ale nemůže s nimi volně nakládat – pouze pomocí metod předka.

2.1 Překrývání/přepis metod

Kromě přidávání vlastních metod, může potomek měnit chování metod předka. Říká se tomu překrývání nebo přepisování (overriding) metod a je to důležitou součástí mechanismu polymorfismu, o němž bude řeč v dalším cvičení.

Překrývání se dělá tak, že metoda má v potomkovi stejnou hlavičku jako v předkovi, ale může dělat jinou činnost.

2.2 Dědičnost v Javě

V Javě jsou všechny třídy automaticky potomky třídy `Object`. To není potřeba nikde v kódu explicitně zmiňovat. I u jednoduchých tříd je zvykem překrýt metodu `toString()`, která je zděděná právě ze třídy `Object`.

```
public class Zvire {
  protected String jmeno;

  public Zvire()
  { /* nastavení jména */ }
  public Zvire(String jmeno)
  { /* nastavení jména */ }

  @Override
  public String toString()
  {/* vrať textovou podobu objektu */}
}
```

Slovo `@Override` je tzv. anotace. Není povinná, ale umožňuje překladači a IDE lépe kontrolovat chyby při psaní kódu. Je doporučeno tuto anotaci používat před každou překrytou metodu.

V Javě se potomek odvozuje pomocí klíčového slova **extends** v hlavičce třídy.

```
public class Pes extends Zvire {
  public Pes(String jmeno)
  { super(jmeno); }
//...
}
```

V konstruktoru potomka se na začátku buďto automaticky volá bezparametrický konstruktor předka (pokud existuje), nebo je možné volat libovolný konstruktor předka pomocí klíčového slova **super**.[1] Je nutné, aby byly atributy předka inicializovány dříve než atributy potomka.

1 Pokud by bylo potřeba volat jiný konstruktor ze stejné třídy, nikoli z předka, dělá se to pomocí klíčového slova **this**(parametry).

3 Úkol

Vytvoř projekt Farma01-PrijmeniJmeno – doplň do názvu své jméno a příjmení. (V dalších cvičeních budeme úlohu rozšiřovat, proto 01.)

1. Vytvoř třídu `Zvire` s atributy `jmeno` a `pocetNohou` (zvol vhodné datové typy).
   - Vytvoř gettery a settery pro tyto atributy. Počet nohou jde jen snižovat 😆.
2. Vytvoř třídy `Pes` a `Ptak` jako potomky třídy `Zvire`. Obě přidají atributy `rasa` a `vek`.
   - Doplň gettery a settery. Rasu měnit nepůjde, věk půjde jen zvyšovat.
3. Ve třídách Pes a Ptak bude docházet k přetěžování konstruktorů:
   - Rasa bude povinný parametr vždy
   - Když se zadá navíc jméno nebo věk nebo obojí, nastaví to odpovídající atributy.
   - Vynechané parametry se nastaví takto: jméno = „NoName“ a věk = 0.
   - Počet nohou uživatel nenastavuje. Ten nastav na správnou výchozí hodnotu.
4. Ve všech třídách přepiš metodu `toString`.
5. Přidej veřejné metody `vydejZvuk()`, které vytisknou na výstup typický zvuk každého zvířete (haf, píp, Playboy, ...).
6. Ve funkci main realizuj tento scénář:
   1. Vyrob dvouletého vlčáka Alíka.
   2. Vyrob roční andulku Julču.
   3. Vypiš jméno, rasu, věk a počet nohou obou zvířat.
      - Finta: metodu toString nemusíš volat, když objekt přičteš k textovému řetězci nebo jej použiješ na místě, kde se očekává textový řetězec.
   4. Nech obě zvířata vydat svůj typický zvuk.