Pokročilejší objektový návrh. dědičnost rozhraní, implementace více rozhraní. Abstraktní třídy.

Obsah

Objektové modelování v Javě - pokračování

- Implementace více rozhraní jednou třídou
- Rozšiřování rozhraní (dědičnost mezi rozhraními)
- Rozšiřování více rozhraní (vícenásobná dědičnost mezi rozhraními)
- Abstraktní třídy (částečná implementace)

Implementace více rozhraní současně

Implementace více rozhraní současně

Třída sice smí dědit maximálně z jedné nadtřídy (předka), ale

- zato může současně implementovat libovolný počet rozhraní!
- Podmínkou ovšem je, aby se metody ze všech implementovaných rozhraní „snesly“ v jedné třídě.
- Které že se nesnesou? Např. dvě metody se skoro stejnou hlavičkou, lišící se „jen“ návratovým typem...

Implementace více rozhraní současně - příklad

Příklad - kromě výše uvedeného intf. *Informujici* mějme ještě:

```
public interface Screaming {
    void scream();
}
```

Třída Clovek implementuje dvě rozhraní:

```
public class Person
    implements Informing, Screaming {
    ...
    public void writeInfo() {
        ...
    }
    public void scream() {
        ...
    }
}
```

Rozšiřování rozhraní

Rozšiřování rozhraní

Podobně jako u tříd, i rozhraní mohou být rozšiřována/specializována. Mohou dědit.

Na rozdíl od třídy, která dědí maximálně z jedné nadtřídy (předka) -

- z rozhraní můžeme odvozovat potomky (podrozhraní - *subinterfaces*)
- dokonce i *vícenásobně* - z více rozhraní odvodíme společného potomka slučujícího a rozšiřujícího vlastnosti všech předků.

Přesto to nepřináší problémy jako u klasické plné vícenásobné dědičnosti např. v C++, protože rozhraní samo

- nemá proměnné

- metody neimplementuje
- nedochází tedy k nejednoznačnostem a konfliktům při podědění neprázdných, implementovaných metod a proměnných

Rozšiřování rozhraní - příklad

Příklad - *Informing* informuje „jen trochu“, *WellInforming* je schopen ke standardním informacím (*writeInfo*) přidat dodatečné informace (*writeAdditionalInfo*).

```
public interface Informing {
   void writeInfo();
}


public interface WellInforming extends Informing {
   void writeAdditionalInfo();
}
```

Třída, která chce implementovat intf.*DobreInformujici*, musí implementovat*obě*metody předepsané tímto rozhraním. Např.:

```
public class Informator implements WellInforming {
   public void writeInfo() {
      ... // kód metody
   }
   public void writeAdditionalInfo() {
      ... // kód metody
   }
}
```

Rozhraní - poznámky

- Používají se i prázdná rozhraní - nepředepisující žádnou metodu
- deklarace, že třída implementuje také rozhraní, ji "k ničemu nezavazuje", ale poskytuje typovou informaci o dané třídě
- i v Java Core API jsou taková rozhraní - např. `java.lang.Cloneable` WIKIPEDIA The Free Encyclopedia

  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=java.lang.Cloneable]

Abstraktní třídy

Abstraktní třídy

I když Java disponuje rozhraními, někdy je vhodné určitou specifikaci implementovat pouze *částečně*:

Rozhraní	Specifikace
Abstraktní třída	Částečná implementace, typicky předek konkrétních tříd, plných implementací
Třída	Implementace

Abstraktní třídy (2)

Abstraktní třída je tak označena v hlavičce, např.:

```
public abstract class AbstraktniChovatel ...
```
WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=public abstract class AbstraktniChovatel ...]

Obvykle má alespoň jednu *abstraktní metodu*, deklarovanou např.:

```
public abstract void vypisInfo();
```
WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=public abstract void vypisInfo();]

Od abstraktní třídy *nelze vytvořit instanci*, nelze napsat např.:

```
Chovatel ch = new AbstraktniChovatel(...);
```
WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Chovatel ch = new AbstraktniChovatel(...);]

Příklad rozhraní - abstraktní třída - neabstraktní třída

Viz Svět chovatelství [http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/svet.html] z učebnice:

- rozhraní `svet.chovatelstvi.Chovatel`
  WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=svet.chovatelstvi.Chovatel] - specifikace, co má chovatel umět
- `svet.chovatelstvi.AbstraktniChovatel`
  WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=svet.chovatelstvi.AbstraktniChovatel] - částečná implementace chovatele
- `svet.chovatelstvi.psi.ChovatelPsu`
  WIKIPEDIA

  [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=svet.chovatelstvi.psi.Chova

telPsu] - úplná implementace chovatele psů

Pozn.: Obecný chovatel se ihned úplně implementovat nedá (ještě to neumíme), proto je definován jako *abstraktní* třída *AbstraktniChovatel* a teprve až *ChovatelPsu* je *neabstraktní* třída.

Reálný příklad použití abstraktní třídy

Rozhraní - abstraktní třída - neabstraktní třída

Viz demo `searching`WIKIPEDIA

[http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=searching] pro BlueJ:

Rozhraní - specifikuje, co má prohledávač umět	`Searcher`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=Searcher]
Abstraktní třída - předek konkrétních plných implementací prohledávače	`AbstractSearcher`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=AbstractSearcher]
Konkrétní třída - plná implementace prohledávače	`LinearSearcher`WIKIPEDIA [http://cs.wikipedia.org/wiki/Speci%C3%A1ln%C3%AD:Search?search=LinearSearcher]

Searcher

Rozhraní - specifikuje, co má prohledávač umět

```
public interface Searcher {

    /**Nastav do vyhledávače pole, kde se bude vyhledávat  */
    void set(double[] a);

    /** Zjisti, zda pole obsahuje číslo d */
    boolean contains(double d);

    /**Zjisti pozici, na níž je v poli číslo d.
       Není-li tam, vrať -1 */
    int indexOf(double d);
}
```

AbstractSearcher

Abstraktní třída - předek konkrétních plných implementací prohledávače

```
public abstract class AbstractSearcher implements Searcher { // implementuje, ale

    // úložiště prvků JE implementováno
    protected double[] array;

    // nastavení úložiště prvků JE implementováno
    public void set(double[] a) {
        array = a;
    }

    // rozhodnutí, zda prvek je přítomen na základě vyhledání jeho pozice
    public boolean contains(double d) {
        return indexOf(d) >= 0;
    }

    // samotné vyhledání prvku není implementováno
    public abstract int indexOf(double d);
}
```

LinearSearcher

Konkrétní třída - plná implementace prohledávače - tentokrát pomocí lineárního prohledání

```
public class LinearSearcher extends AbstractSearcher { // doimplementuje se, co zb

    // a to je metoda indexOf!
    public int indexOf(double d) {
        for(int i = 0; i < array.length; i++) {
            if(array[i] == d) {
                return i;
            }
        }
        return -1;
    }
}
```