

---

# Kapitola 1. Přednáška 6 - rozhraní, abstraktní třídy

## Obsah

Objektové modelování v Javě - pokračování .....	1
Rozhraní .....	1
Co je rozhraní .....	2
Deklarace rozhraní .....	2
Implementace rozhraní .....	2
Využití rozhraní .....	3
Implementace více rozhraní současně .....	3
Implementace více rozhraní současně - příklad .....	3
Rozšiřování rozhraní .....	4
Rozšiřování rozhraní - příklad .....	4
Rozhraní - poznámky .....	5
Abstraktní třídy .....	5
Abstraktní třídy (2) .....	5
Příklad rozhraní - abstraktní třída - neabstraktní třída .....	5

## Objektové modelování v Javě - pokračování

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

## Rozhraní

V Javě, na rozdíl od C++ neexistuje vícenásobná dědičnost -

- to nám ušetří řadu komplikací
- ale je třeba to něčím nahradit

Pokud po třídě chceme, aby disponovala vlastnostmi z několika různých množin (skupin), můžeme ji deklarovat tak, že

- implementuje více rozhraní

## Co je rozhraní

- Rozhraní je vlastně *popis (specifikace) množiny vlastností, aniž bychom tyto vlastnosti ihned implementovali*. Vlastnostmi zde rozumíme především *metody*.
- Říkáme, že určitá třída *implementuje rozhraní*, pokud implementuje (tedy *má* - přímo sama nebo po dědi) všechny vlastnosti (tj. metody), které jsou daným rozhraním předepsány.
- Javové rozhraní je tedy *množina hlaviček metod označená identifikátorem* - názvem rozhraní. (a celých specifikací - tj. popisem, co přesně má metoda dělat - vstupy/výstupy metody, její vedlejší efekty...)

## Deklarace rozhraní

- Vypadá i umísťuje se do souborů podobně jako deklarace třídy
- Všechny metody v rozhraní musí být `public` a v hlavičce se to ani nemusí uvádět.
- Těla metod v deklaraci rozhraní se nepíše. (Metody v rozhraní tudíž vypadají velmi podobně jako abstraktní metody ve třídách, ale nemusím psát `abstract`.)

Příklad deklarace rozhraní

```
public interface Informujici {
    void vypisInfo();
}
```

## Implementace rozhraní

Příklad

```
public class Clovek implements Informujici {
    ...
    public void vypisInfo() {
        ...
    }
}
```

Čteme: Třída `Clovek` implementuje rozhraní `Informujici`.

1. Třída v hlavičce uvede `implements` `NázevRozhraní`
2. Třída implementuje všechny metody předepsané rozhraním

## Využití rozhraní

1. Potřebujeme-li u jisté proměnné právě jen funkcionalitu popsanou určitým rozhraním,
2. tuto proměnnou můžeme pak deklarovat jako typu rozhraní - ne přímo objektu, který rozhraní implementuje.

Příklad

```
Informujici petr = new Clovek("Petr Novák", 1945);
petr.vypisInfo(); // "petr" stačí deklarovat jen jako Informujici
                 // jiné metody než předepsané tímto intf.
                 // nepotřebujeme!
```

## 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 Kricici {
    void zakric();
}
```

Třída `Clovek` implementuje dvě rozhraní:

```
public class Clovek
    implements Informujici, Kricici {
    ...
}
```

```
public void vypisInfo() {  
    ...  
}  
public void zakric() {  
    ...  
}  
}
```

## 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 - *Informujici* informuje „jen trochu“, *DobreInformujici* je schopen ke standardním informacím (*vypisInfo*) přidat dodatečné informace (*vypisDodatecneInfo*).

```
public interface Informujici {  
    void vypisInfo();  
}  
  
public interface DobreInformujici extends Informujici {  
    void vypisDodatecneInfo();  
}
```

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

```
public class Informator implements DobreInformujici {
    public void vypisInfo() {
        ... // kód metody
    }
    public void vypisDodatecneInfo() {
        ... // 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`

## Abstraktní třídy

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

- Rozhraní = Specifikace
- Abstraktní třída = Částečná implementace
- Třída = Implementace

## Abstraktní třídy (2)

Abstraktní třída je tak označena v hlavičce, např.:

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

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

```
public abstract void vypisInfo() ...
```

Od a.t. *nelze vytvořit instanci*, nelze napsat např.:

```
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` - specifikace, co má chovatel umět
- `svet.chovatelstvi.AbstraktniChovatel` - částečná implementace chovatele
- `svet.chovatelstvi.psi.ChovatelPsu` - 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*.