

# Seminář Java

## Základy objektové orientace I

Radek Kočí

Fakulta informačních technologií VUT

Únor 2010

- Charakteristika objektově orientovaných systémů
- Třída, objekt, objektové rozhraní
- Abstrakce, zapouzdření
- Vytváření objektů, konstruktory

## Objektově orientovaný přístup

- abstrakce řešené domény na základě vyhledávání podobností  $\Rightarrow$  objekty
- **objekt** = sloučení dat a funkcionality do uzavřené jednotky
- aplikace je chápána jako kolekce vzájemně komunikujících objektů

Rozlišujeme dva typy jazyků

- Prototypově založené (prototype-based)
  - pracuje se pouze s objekty
  - nové objekty se vytvářejí klonováním již existujících objektů
  - existuje alespoň jeden počáteční objekt (prototyp)
  - *Self, JavaScript, ...*
  
- Třídně založené (class-based)
  - nalezené objekty jsou klasifikovány do tříd
  - *Smalltalk, Java, C++, C#, ...*

## Třída

- vzor popisující strukturu a chování objektů stejného druhu
- množina objektů stejného druhu
- deklaruje proměnné (atributy) a metody *objektu*
- může deklarovat proměnné (atributy) a metody *třídy*

## Objekt

- instance třídy
- objekty mají vlastní data (atributy) – kopie
- objekty sdílí chování – metody

Vlastnosti objektů je třeba deklarovat

- proměnné
  - jsou nositeli "pasivních" vlastností, charakteristik objektů
  - datové hodnoty svázané (zapouzdřené) v objektu
- metody
  - jsou nositeli "výkonných" vlastností, "dovedností" objektů
  - v podstatě funkce (procedury) pracující primárně s proměnnými "mateřského" objektu
  - může mít další parametry (argumenty metody)
  - může vrátet hodnotu

Programování v Javě spočívá ve vytváření tříd, neexistují metody a atributy deklarované mimo třídy.

## Vlastnosti objektové orientace

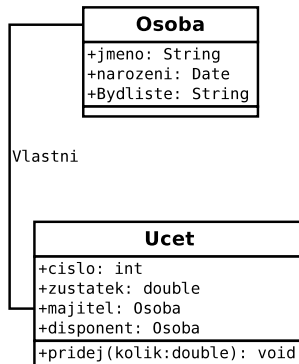
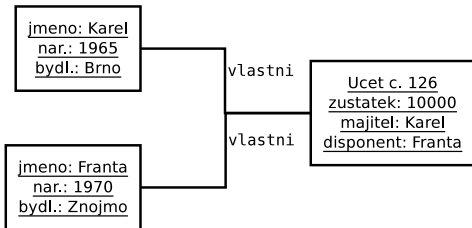
- Abstrakce (abstraction)
- Zapouzdření (encapsulation)
- *Polymorfismus (polymorphism)*
- *Dědičnost (inheritance)*

## Abstrakce

- vytvářený systém objektů je abstrakcí řešeného problému
- analýza problému  $\Rightarrow$  klasifikace do abstraktních struktur  $\Rightarrow$  objekty  $\Rightarrow$  třídy
- klasifikace je založena na rozpoznávání podobností v řešené problematice
- zjednodušený pohled na systém bez ztráty jeho významu
- objekt je abstrakcí části řešené domény, má definovanou zodpovědnost za řešení části problému



# Vlastnosti objektové orientace – Abstrakce



## Zapouzdření

- Seskupení souvisejících idejí do jedné jednotky, na kterou se lze následně odkazovat jediným názvem (objekt).
- Objektově orientované zapouzdření je seskupení operací a atributů (reprezentujících stav) do jednoho typu objektu. Stav je pak dostupný či modifikovatelný pouze prostřednictvím rozhraní (operace, metody).
- Omezení externí viditelnosti informací nebo implementačních detailů.
- Zaručené rozhraní.

```
int obsah(int x, int y) {  
    return x * y;  
}
```

---

```
struct Obdelnik {  
    int x, y;  
}  
int obsah(struct Obdelnik o) {  
    return o.x * o.y;  
}
```

---

```
struct Obdelnik {  
    int x, y;  
    int obsah() { return x * y; }  
}
```

# Vlastnosti objektové orientace – Zapouzdření

```
int obsah(int x, int y) {  
    return x * y;  
}
```

---

```
struct Obdelnik {  
    int x, y;  
}  
int obsah(struct Obdelnik o) {  
    return o.x * o.y;  
}
```

---

```
struct Obdelnik {  
    int x, y;  
    int obsah() { return x * y; }  
}
```

```
int obsah(int x, int y) {  
    return x * y;  
}
```

---

```
struct Obdelnik {  
    int x, y;  
}  
int obsah(struct Obdelnik o) {  
    return o.x * o.y;  
}
```

---

```
struct Obdelnik {  
    int x, y;  
    int obsah() { return x * y; }  
}
```

```
class Obdelnik {  
    int x;  
    int y;  
    int obsah() { return x * y; }  
}
```

Ukrývání implementačních detailů, omezení přístupu k vlastnostem tříd (zajištění integrity dat)

```
public class Obdelnik {  
    protected int x;  
    protected int y;  
    public int obsah() { return x * y; }  
}
```

```
class Obdelnik {  
    int x;  
    int y;  
    int obsah() { return x * y; }  
}
```

Ukrývání implementačních detailů, omezení přístupu k vlastnostem tříd (zajištění integrity dat)

```
public class Obdelnik {  
    protected int x;  
    protected int y;  
    public int obsah() { return x * y; }  
}
```

```
class Obdelnik {  
    int x;  
    int y;  
    int obsah() { return x * y; }  
}
```

Ukrývání implementačních detailů, omezení přístupu k vlastnostem tříd (zajištění integrity dat)

```
public class Obdelnik {  
    protected int x;  
    protected int y;  
    public int obsah() { return x * y; }  
}
```



## Operace vs. metoda

- množina operací reprezentuje chování objektu
- metoda implementuje operaci

## Rozhraní objektu

- množina operací, které objekt nabízí
- pouze definuje co objekt umí (nabízí)

```
public class Obdelnik {  
    public int obsah() { ... }  
    public int obvod() { ... }  
}
```

---

```
public interface Ctyruhelnik {  
    public int obsah();  
    public int obvod();  
}
```

```
public class Obdelnik implements Ctyruhelnik {  
    public int obsah() { // implementace operace }  
    public int obvod() { // implementace operace }  
}
```

```
public class Obdelnik {  
    public int obsah() { ... }  
    public int obvod() { ... }  
}
```

---

```
public interface Ctyruhelnik {  
    public int obsah();  
    public int obvod();  
}
```

```
public class Obdelnik implements Ctyruhelnik {  
    public int obsah() { // implementace operace }  
    public int obvod() { // implementace operace }  
}
```

## Komunikace objektů

- objekty spolu komunikují zasláním zpráv
- příjemce chápe zprávu jako požadavek na provedení služby (operace)
- zpráva obsahuje
  - identifikátor příjemce
  - název operace
  - argumenty
- obsluha zprávy (vykonání metody) reaguje podle stavu / modifikuje stav objektu
- po ukončení obsluhy může metoda vracet výsledek

- třída implementuje rozhraní (metody) a definuje atributy
- objekt je instancí třídy
- objekty stejné třídy sdílejí chování (metody), atributy má každý objekt vlastní

```
Obdelnik o = new Obdelnik();  
int obsah = o.obsah();
```

Voláním `new Obdelnik()` jsme použili:

- operátor `new`, který vytvoří prázdný objekt a
- volání *konstrukturu*, který prázdný objekt naplní počátečními údaji (daty).

## Konstruktory

- Konstruktory jsou speciální metody volané při vytváření nových instancí dané třídy.
- Typicky se v konstrukturu naplní (inicializují) proměnné objektu.
- Konstruktory lze volat jen ve spojení s operátorem `new` k vytvoření nové instance třídy – nového objektu, eventuálně volat z jiného konstrukturu.

# Implicitní konstruktor

Každá třída má *implicitní* (bezparametrický) konstruktor

- nemá žádné parametry
- nemá žádný návratový typ!
- nemusí se deklarovat
- deklarace: `JmenoTridy() {...}`

```
public class Obdelnik {  
    public Obdelnik() {  
        ...  
    }  
}
```

Použití: `new Obdelnik();`

# Další konstruktory

Každá třída může mít další (jiné) konstruktory než implicitní

- odlišují se parametry
- pokud se deklaruje alespoň jeden konstruktor, implicitní se již negeneruje!!

```
public class Obdelnik {  
    protected int x;  
    protected int y;  
    public Obdelnik(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

Použití:

```
new Obdelnik(50, 20);
```

```
new Obdelnik();
```

⇐ chyba!



Pokud chceme deklarovat další konstruktory a současně používat implicitní, musíme ho také deklarovat!

```
public class Obdelnik {  
    public Obdelnik() { }  
    public Obdelnik(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

Použití:

```
new Obdelnik(50, 20);
```

```
new Obdelnik();           ⇐ OK (ale zbytečné)
```