Přetěžování metod

```
<link rel="stylesheet"  href="http://cdnjs.cloudflare.com/ajax/libs/font-
awesome/3.1.0/css/font-awesome.min.css">
```

Přetěžování metod

- Jedna třída může mít více metod se *stejnými názvy, ale různými parametry*.
- Pak hovoříme o tzv. *přetížené* (*overloaded*) metodě.
- I když jsou to teoreticky úplně různé metody, jmenují se stejně, proto by *měly dělat něco podobného*.

Přetěžování — příklad I

```
public void transferTo(Account whereTo, double amount) {
  this.add(-amount);
  whereTo.add(amount);
}
public void transferTo(Account whereTo) {
  whereTo.add(balance);
  balance = 0;
}
```

- První metoda převede na účet příjemce `amount` peněz.
- Druhá metoda převede na účet *celý zůstatek* (`balance`) z účtu odesílatele.
- Nedala by se jedna metoda volat pomocí druhé?

Přetěžování — příklad II

```
public void transferTo(Account whereTo, double amount) {
  this.add(-amount);
  whereTo.add(amount);
}
public void transferTo(Account whereTo) {
  transferTo(whereTo, balance);
}
```

- Toto je *jednodušší, přehlednější*, udělá se tam potenciálně méně chyb.
- Kód se neopakuje, tudíž se neopakuje ani případná chyba
- Je to přesně postup divide-et-impera, rozděl a panuj, dělba práce mezi metodami!

Přetěžování — příklad III

Zkusme naší metodu lépe popsat:

```
public void transferTo(Account whereTo, double amount) {
  this.add(-amount);
  whereTo.add(amount);
}
public void transferAllMoneyTo(Account whereTo) {
  transferTo(whereTo, balance);
}
```

- Převod celého zůstatku jsme napsali jako *nepřetíženou* metodu, která popisuje přesně, co dělá.
- Z názvu metody je zřejmé, co dělá — netřeba ji komentovat!

Přetěžování konstruktorů

- Přetěžovat můžeme i konstruktory
- Můžeme tak mít více konstruktorů v jedné třídě
- Pro vzájemné volání konstruktorů použijeme klíčové slovo `this`
- Používá se hodně často, častěji než přetěžování jiných metod

```
public Person() {
  this("Default name"); // calls second constructor
}
public Person(String name) {
  this.name = name;
}
```

Přetěžování — jak ne

- Proč nelze přetížit metodu *pouze změnou typu návratové hodnoty*?
- Která metoda se zavolá?

```
public int getNumber() {
  return 5;
}
public short getNumber() { // smaller int
  return 6;
}
...
long bigInt = getNumber(); // 5 or 6?
```

- V Javě se číselné typy proměnných přetypují automaticky.
- Mělo by dojít k přetypování `int` na `long`, nebo `short` na `long`?

Obdobný příklad

- Nelze také přetížit uvedením a neuvedením návratové hodnoty
- Protože vracenou hodnotu stejně nemusíme použít:

```
new String("Sss").isEmpty(); // result is omitted
```

- Opět nevíme, která metoda se zavolá:

```
public void getNumber() {
  // do nothing
}
public int getNumber() { // smaller int
  return 6;
}
...
getNumber(); // which one is called?
```

Vracení odkazu na sebe

- Metoda může vracet odkaz na objekt, nad nímž je volána pomocí `this`:

```
public class Account {
  private double balance;

  public Account(double balance) {
    this.balance = balance;
  }

  public Account transferTo(Account whereTo, double amount) {
    add(-amount);
    whereTo.add(amount);
    return this; // return original object
  }
}
```

Řetězení volání

- Vracení odkazu na sebe lze využít k *řetězení volání*:

```
Account petrsAccount = new Account(100);
Account ivansAccount = new Account(100);
Account robertsAccount = new Account(1000);

// we can chain methods
petrsAccount
   .transferTo(ivansAccount, 50)
   .transferTo(robertsAccount, 20);
```

- Stejný princip se dost často využívá u `StringBuilder` metody `append`.