Zapouzdření

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

Co je zapouzdření

- Naprosto zásadní vlastnost objektového přístupu, možná nejzásadnější
- Jde o *spojení dat a práce s nimi* do jednoho celku - objektu
- Data jsou v atributech objektu, práce je umožněna díky metodám objektu
- Data by měla být zvenčí (jinými objekty) přístupná jen prostřednictvím metod
- Data jsou tedy "skryta" uvnitř, zapoudřena
- To zajistí větší bezpečnost a robustnost, přístup k datům máme pod kontrolou

Motivace zapouzdření

```
class Person {
   String name;
   int age;

   Person(String inputName, int inputAge) {
     name = inputName;
     age = inputAge;
   }
 }
```

Co když:

- Nechceme, aby kdokoli mohl modifikovat atributy `name`, `age` po vytvoření objektu
- Většinou ale chceme, aby konstruktor a třídu mohl používat každý

Řešení — práva přístupu

- Nastavíme *viditelnost* nebo též *práva přístupu* pomocí modifikátorů třídy, metody nebo atributu
- Nechceme modifikovat atributy `name`, `age` po vytvoření objektu? ⇒ použijeme klíčové slovo `private`
- Chceme, aby mohl konstruktor a třídu používat skutečně každý? ⇒ použijeme klíčové slovo `public`

```java
public class Person {
  private String name;
  private int age;

  public Person(String inputName, int inputAge) {
    name = inputName;
    age = inputAge;
  }
}
```

4 typy viditelnosti v zkratce

- `public` = veřejný, může používat každý i mimo balík ⇒ používejte na třídy a (některé) metody
- `private` = soukromý, nemůže používat nikdo mimo třídy ⇒ používejte na atributy
- `protected` = chráněný ⇒ používá se při dědičnosti, vysvětlíme později
- *modifikátor neuveden*, pak jde o možnost přístupu "package local"
  - v rámci balíku se chová jako `public`, mimo něj jako `private`
  - v našem kurzu tento typ nebudeme používat
- Ujistěte se, že vždy máte zadefinovaný práva přístupu/typ viditelnosti.
- V drtivé většině budete používat `public` a `private`.

Každá **veřejná** třída musí být v souburu se stejným jménem.

Metody `get` & `set` — motivace

```java
public class Person {
   private String name;
   private int age;

   public Person(String inputName, int inputAge) {
     name = inputName;
     age = inputAge;
   }
 }
```

- Klíčové slovo `public` umožňuje použít třídu `Person` všude

```java
Person p = new Person("Marek", 23); // even from another class/package
```

- Klíčové slovo `private` zabraňuje získat hodnotu atributů `p.name`, `p.age`.

Metody `get`

- Chci *získat hodnotu* atributu i po vytvoření objektu,
- ale *zabránit jeho modifikaci*?
- Do třídy přidáme metodu, která bude *veřejná* a po zavolaní vrátí hodnotu atributu.

```
public int getAge() {
  return age;
}
```

- Takové metody se slangově nazývají "gettery".
- Mají návratovou hodnotu podle typu vraceného atributu.
- Název metody je vždy `get` + *jméno atributu* s velkým písmenem (`getAge`, `getName`, ...).

Metody `set`

- Chci-li nastavit hodnotu atributu i po vytvoření objektu:

```
public void setAge(int updatedAge) {
  age = updatedAge;
}
```

- Metoda je *veřejná* a po jejím zavolaní *přenastaví* původní hodnotu atributu.
- Takové metody se slangově nazývají **settery**.
- Mají návratovou hodnotu typu `void` (nevrací nic).
- Název metody je vždy `set` + *jméno atributu* s velkým písmenem (`setAge`, `setName`, ...).

Příklad atribut a `get` & `set`

```
public class Person {
  private String name; // attribute
  public String getName() { // its getter
    return name;
  }
  public void setName(String newName) { // its setter
    name = newName;
  }
}
```

Viditelnost atributů

- Není lepší udělat atribut `public`, namísto vytváření metod `get` a `set`?
- Není, neumíme pak řešit tyhle problémy:
- Co když chci jenom získat hodnotu atributu, ale zakázat modifikaci (mimo třídy)? ⇒ *Řešení:* odstraním metodu `set`
- Chci nastavit atribut věk (v třídě `Person`) pouze na kladné číslo? ⇒ *Řešení:* upravím metodu `set`:
  - `if (updatedAge > 0) age = updatedAge;`
- Chci přidat kód provedený při získávání/nastavování hodnoty atributů? ⇒ *Řešení:* upravím metodu `get`/`set`
- Gettery & settery se dají ve vývojových prostředích (NetBeans, IDEA) generovat automaticky.

Využití `this`

```
public void setAge(int updatedAge) {
  age = updatedAge;
}
```

- Mohli bychom nahradit jméno parametru `updatedAge` za `age`?
- Ano, ale jak bychom se potom dostali k atributu objektu?
- Použitím klíčového slova `this`:

```
public void setAge(int age) {
  this.age = age;
}
```

- `this` určuje, že jde o atribut objektu, nikoli parametr (lokální proměnnou)

Korektní použití třídy I

```
public class Person {
   private String name;
   public Person(String name) {
     this.name = name;
   }
   public void writeInfo() {
     System.out.println("Person " + name);
   }
   public String getName() {
     return this.name;
   }
}
```

Korektní použití třídy II

- Vytvoříme dvě instance (konkrétní objekty) typu `Person`.

```
public class Demo {
   public static void main(String[] args) {
      Person ales = new Person("Ales");
      Person beata = new Person("Beata");
      ales.writeInfo();
      beata.writeInfo();
      String alesName = ales.getName(); // getter is used
      // String alesName = ales.name; // forbidden
   }
}
```

Třída `Account` — připomenutí

```
public class Account {
  private double balance;
  public void add(double amount) {
    balance += amount;
  }
  public void writeBalance() {
    System.out.println(balance);
  }
  public void transferTo(Account whereTo, double amount) {
    balance -= amount; // change the balance
    whereTo.add(amount);
  }
}
```

- Co je zde malý nedostatek?

- metoda `transferTo` přistupuje přímo k `balance`
- ale přístup k `balance` by měl být nějak lépe kontrolován (např. zda z účtu neberu více, než smím)!

Třída `Account` — řešení

- řešení: znovupoužijeme metodu `add`, která se o kontrolu zůstatku postará
- (i když to třeba ještě teď neumí!)

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