Konstruktory

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

Konstruktory

- Konstruktory jsou speciální *metody* volané při vytváření nových objektů (=instancí) dané třídy.
- V konstruktoru se typicky *inicializují atributy (proměnné) objektu*.
- Konstruktory lze volat jen ve spojení s operátorem **new** k vytvoření nového objektu.

Příklad konstruktoru

- Konstruktor se dvěma parametry, inicializuje hodnoty atributů:

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

  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
}
```

- identifikátor `this` znamená, že se přistupuje k atributům objektu
- nastavují se dle hodnot `name, age` předaných do konstruktoru

Konstruktory — použití

- Příklad využití tohoto konstruktoru:

```
...
Person pepa = new Person("Pepa from Hongkong", 105);
...
```

- Toto volání vytvoří objekt `pepa` a naplní ho jménem a věkem.
- Následně je možné získávat hodnoty proměnných objektů pomocí tečkové notace, např. `pepa.age`.

Výchozí (default) konstruktor

- Co když třída nemá definovaný žádný konstruktor?
- Vytvoří se automaticky *výchozí (default)* konstruktor:

```
public Person() { }
```

- Použití konstruktoru pak vypadá následovně:

```
Person p = new Person();
```

- Výchozí (default) konstruktor se vytvoří pouze v případě, že žádný jiný konstruktor v třídě neexistuje.

Nevytvoření objektu

- Co když volám nad proměnnou metody bez vytvoření objektu?

```
Person p = null;
System.out.println(p.getName());
```

- Kód po spuštění "spadne", neboli zhavaruje nebo předčasně skončí.
- Java sa snaží pád programu popsat pomocí výjimek.
- Výjimky mají své *jméno*, obvykle i určitý textový popis dokumentující příčinu havárie.

```
Exception in thread "main" java.lang.NullPointerException
```

- Výjimky budou probírány později.

Návratový typ konstruktoru?

- Jaký je návratový typ konstruktoru?
- "prázdný" typ `void`? NIKOLI!
- konstruktory vracejí odkaz na vytvořený objekt
- *návratový typ nepíšeme*, typem je odkaz na nově vytvořený objekt

Proměnná objektového typu

- Bavíme se o proměnných *lokálních* ve kódu metod.
- Proměnná objektového typu se deklaruje např. `Person p;`
- Deklarace proměnné objektového typu sama o sobě *žádný objekt nevytváří*
- Takové proměnné jsou pouze *odkazy* na *dynamicky vytvářené objekty*
- Vytvoření objektu se děje až operátorem `new` dynamicky, instance se vytvoří až za běhu programu
- V Javě sa celé objekty do proměnné *neukládají*, jde vždy o uložení pouze *odkazu* (adresy) na objekt

Přiřazení proměnné objektového typu

- Přiřazením takové proměnné pouze *zkopírujeme odkaz*
- Na jeden objekt se odkazujeme nadále ze dvou míst
- **Nezduplikujeme** tím objekt

Příklad kopie odkazu na objekt

- Proměnné `jan` a `janCopy` ukazují na ten tentýž objekt ⇒ změna objektu se projeví v obou:

```
public static void main(String[] args) {
  Person jan = new Person("Jan");
  Person janCopy = jan;
  janCopy.name = "Janko"; // modifies jan too
  System.out.println(jan.name); // prints "Janko"
}
```

Konstruktory — shrnutí

Jak je psát a co s nimi lze dělat?

- neuvádí se *návratový typ*
- mohou a nemusí mít *parametry*
- když třída nemá žádný konstruktor, automaticky se vytvoří *výchozí*
- může jich být *více* v jedné třídě, reálně se používá

Návrhové vzory

- Návrhové vzory jsou osvědčené způsoby objektové dekompozice v jasně popsaných situacích

- Jsou použitelné pro libovolný objektově orientovaný jazyk
- Jejich aplikace ale vyžaduje návrhová rozhodnutí, která mohou být ovlivněna vlastnostmi programovacího jazyka
- Mnohé si postupně stručně představíme s cílem
  - demonstrovat, že objektová dekompozice Java Core API není náhodná,
  - motivovat vás l používání vozrů při dekompozici vašeho kódu.

Vytvářecí návrhové vzory

Speciální podskupina návrhových vzorů, která nabízí alternativní způsoby k vytváření objektů, než je prosté volání konstruktoru

- Singleton: Vytvoření jediné instance třídy, kterou všichni sdílí a snadno k ní přistupují odkudkoli.
- Builder: Konstrukce složitého objektu po kouscích (např. vytvoření grafu přidáváním uzlů a hran).
- Prototype: Namísto vytváření nového objektu naklonuj existující objekt.
- Abstract Factory: Jednotné místo pro vytváření vzájemně kompatibilních instancí různých tříd.
- Factory Method: Přenechání konstrukce objektu podtřídě.