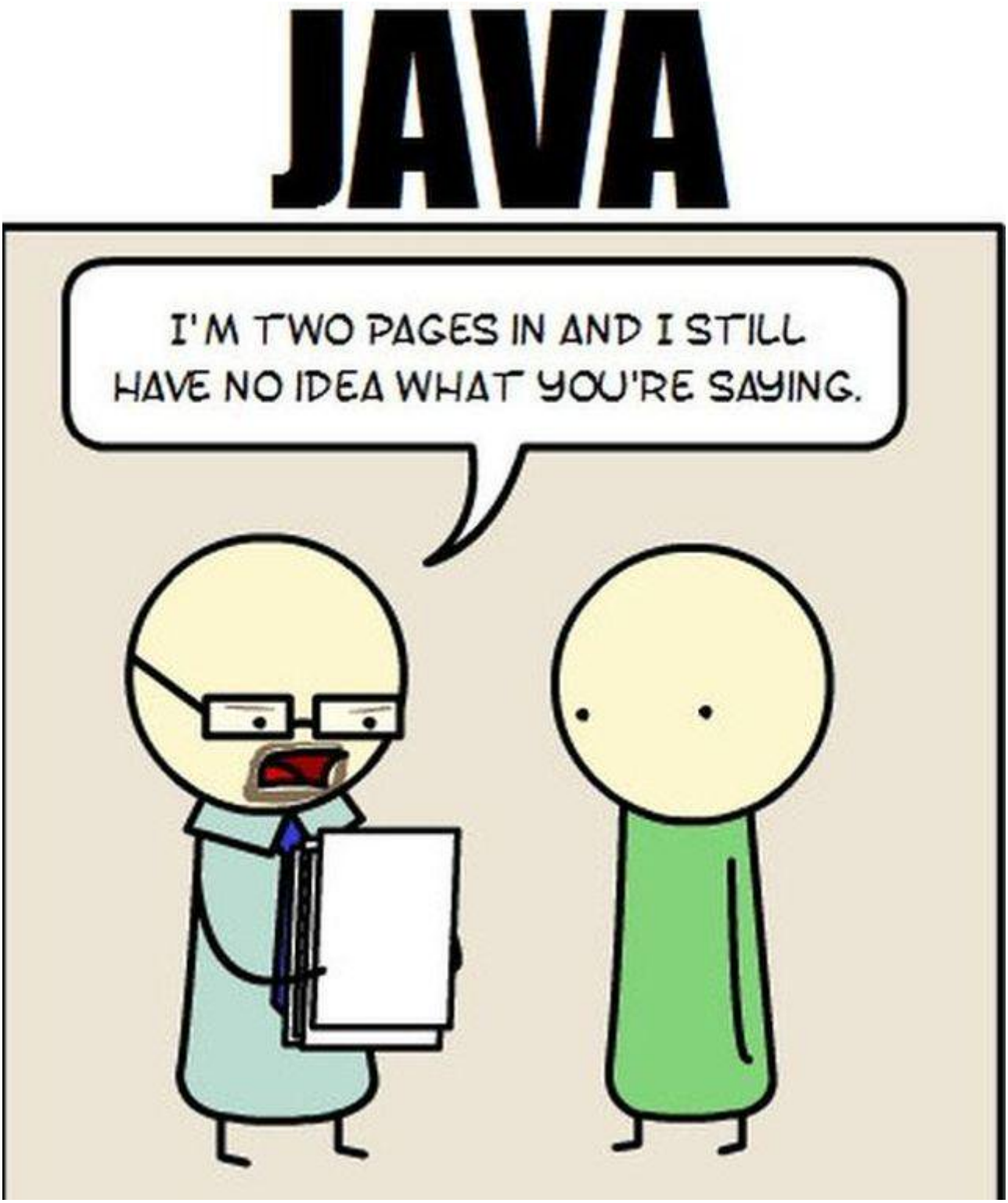
Statické proměnné a metody

Table of Contents

Java je ukecaná



Úvod

Se statickou metodou jsme se setkali už u úplně prvního programu *Hello, world!*

```
public class Demo {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

- `public` reprezentuje viditelnost — metoda `main` je veřejná, chceme spouštět
- `void` reprezentuje návratový typ — že `main` nic nevrací

Co znamená `static`

- Metoda `static` v Javě (jinde může být chápáno jinak) říká, že metoda nepotřebuje pro své fungování žádný konkrétní existující objekt, s nímž by pracovala.
- To přesně potřebujeme u `main`, neboť žádný objekt dosud nemáme.
- Sémantika `static` je +/- stejná jako v Pythonu.

Dosud byly metody a atributy *nestatické*

Nestatické atributy a metody

- Neboli *atributy (proměnné)* a *metody* objektu.
- Jméno (atribut `String name`) patří přímo jedné osobě.
- Metoda `toString` vrátí `Person jméno této osoby`.

Person.java

```
public class Person {
  private String name; // name of this person

  public String toString() {
    return "Person " + name; // returns name of this person
  }
}
```

Jak fungují statické metody a atributy

- Lze deklarovat také metody a atributy patřící *celé třídě*, tj. *všem objektům třídy*.
- Taková proměnná (nebo metoda) existuje pro jednu třídu jen *jednou*.
- Označujeme ji `static`.

Podobně v Pythonu

- Statické proměnné/atributy existují i v Pythonu.
- Označují se jako *proměnné třídy* (takto můžeme označovat i v Javě), zatímco atributy jsou *proměnné objektu*.
- V Pythonu se poznají tak, že se *neoznačují* `self`.

```
# Class for Computer Science Student
```

```python
class CSStudent:
    stream = 'cse'                  # Class Variable
    def __init__(self,name):
        self.name = name            # Instance Variable
```

Použití statických metod

Zadání: metoda `max`

- vrací maximální hodnotu dvou čísel
- K tomu žádné objekty nepotřebujeme
- Uděláme ji jako statickou
- Kdekoli zavoláme `Calculate.max(-2, 8)`

Statická ve třídě `Calculate`

```java
public class Calculate {
  public static int max(int a, int b) {
    if (a > b) return a;
    return b;
  }
}
```

Jde i jako nestatická?

Ano, vynecháním `static`

```java
public class Calculate {
  public int max(int a, int b) {
    if (a > b) return a;
    return b;
  }
}
```

Na spuštění `max` *nyní potřebuji objekt*

```java
Calculate c = new Calculate();
int max = c.max(1, 2); // method needs an object `c`
```

- Ovšem ten objekt `c` je tam úplně k ničemu, s žádnými jeho atributy se nepracuje a ani žádné nemá

Řešení

- Uděláme metodu statickou.

- Pak metoda patří celé třídě a zavoláme ji názvem třídy bez konkrétního objektu.

```
public class Calculate {
  public static int max(int a, int b) {
    if (a > b) return a;
    return b;
  }
}
...
int m = Calculate.max(1, 2);
```

Zpříjemnění použití statických metod

- Někdy v kódu často používáme statické metody určité třídy, např. naše `Calculator.max`
- Pro kratší zápis lze pak využít deklaraci `import static` dané metody
- nebo všech metod přes *, např.

```
import static cz.muni.fi.pb162.Calculator.*;

int m = max(1, 2);
```

Typické použití statických metod

- Velmi často se používají v obdobných situacích, jako výše uvedené `max`
- Tzn. jednoduše pro implementaci funkce, která nevyužívá žádné atributy (data objektu), pouze dostane vstupy a něco vrátí.
- Pak ani logicky žádný objekt nepotřebuje.
- Příklady: metody třídy `java.util.Arrays`

Statické proměnné (atributy třídy)

- Doposud jsem měli pouze proměnné (atributy) patřící konkrétnímu objektu.
- Např. ve třídě `Person`, která reprezentuje člověka, má každý člověk své (obvykle i odlišné) jméno.
- Někdy je ale situace, kdy pro celou třídu stačí určitý údaj jenom jednou.
- Příklad: chceme jsi pamatovat, kolik lidí se nám během chodu programu vytvořilo.
- Jak to udělat?

Počítaní lidí

- Vytvoříme *statickou* proměnnou peopleCount a každý člověk ji při svém vzniku zvýší o jedna.

```
public class Person {
   private String name;
   private static int peopleCount = 0;
   public Person(String name) {
      this.name = name;
      peopleCount++;
   }
   public static int howManyPeople() {
      return peopleCount;
   }
}
```

- Logicky na vrácení počtu lidí stačí *statická metoda* `howManyPeople()`.

Počítaní lidí II

- Použití bude vypadat následovně

```
Person.howManyPeople(); // 0
Person jan = new Person("Jan");
Person.howManyPeople(); // 1
Person anna = new Person("Anna");
Person.howManyPeople(); // 2
```

Více informací: Java tutorial — Class variables

Voláni statické metody

Můžeme volat statickou metodu nad konkrétním objektem (instancí) dané třídy?

```
Person anna = new Person("Anna");
anna.howManyPeople();
```

- Ano, není to problém.
- Přes třídu `Person` je to však správnější.

Voláni nestatické metody

- Můžeme volat nestatickou metodu jako statickou?

```
Person.getName();
```

- Logicky ne!
- Co by mohlo volání `Person.getName()` vrátit? Nedává to smysl.
- Jde nám přece o jméno konkrétního člověka, tj. atribut v konkrétním objektu `Person`
- Atribut `name` se nastaví až při zavolání konstruktoru

Přístup ze statické metody k nestatickému atributu?

- Obdobně platí pro atributy, tj. NELZE toto:

```
public class NonStaticTest {
  private int count = 0;
  public static void main(String args[]) {
    count++; // compiler error
  }
}
```

- Java ohlásí při překladu chybu: *non-static variable count cannot be referenced from a static context*.
- Metoda `main` je statická — může používat pouze statické metody a proměnné.

Přístup k nestatickému atributu

- Jedině po vytvoření konkrétní instance:

```
public class NonStaticTest {
  private int count = 0;
  public static void main(String args[]) {
    NonStaticTest test = new NonStaticTest();
    test.count++;
  }
}
```

Všimněte si, že ve třídě mohu vytvořit objekt té stejné třídy.

- Dalším řešením by bylo udělat `count` statický.

Problémy se statickými metodami? NE

- Ano, použití `static` není tak prosté, jak jsme dosud prezentovali :-)
- *Statické metody* většinou problém nejsou.
- Jednoduše slouží k realizaci nějaké činnosti, které stačí předané vstupy (parametry) a která nepotřebuje žádný "svůj" objekt s atributy.
- Důkazem je řada použití statických metod v Java Core API, kde jsou třídy, které mají *jen statické metody*.
- Takovým třídám se říká *utility classes* (jakési "účelové" třídy).

Raději méně

- Na obecné jednoduché úkony jsou statické metody fajn.
- Lákavé je použití statických metod v knihovnách.
- Nicméně když zrovna nepíšeme knihovnu, dvakrát před vytvořením statické metody přemýšlejme:
  - *Nebude mít příliš mnoho parametrů, aby mohla dělat, co má?*
  - *Nemusím jí předávat složitá data, která by si mohla vzít z objektu?*

Problémy se statickými proměnnými? ANO

- U *statických proměnných* je to složitější.
- Jejich použití musí být hodně dobře zdůvodněné.
- Opravdu potřebujeme danou hodnotu *pro danou třídu právě jednou*???
- Nestane se dobudoucna, že jich budeme potřebovat více — třeba dvě, tři, deset???

Možné řešení — singleton

- Často je lepší aplikovat *návrhový vzor Singleton* (jedináček)
- Jedná se běžnou třídu, např. `PersonCounter`, která má své běžné nestatické atributy a metody zajišťující potřebnou funkcionalitu
  - Např. metody `void increaseNumPeople()` a atribut `int peopleCount` zajišťující počítání vytvořených lidí.

```
public class PersonCounter {
   private int peopleCount = 0;
   public void increaseNumPeople() {
      peopleCount++;
```

```
    }
    public int howManyPeople() {
        return peopleCount;
    }
}
```

Možné řešení — singleton

Singleton navíc zajišťuje vytvoření jediné sdílené instance (např. jediného počítadla lidí pro celý systém):

- Zakáže se volání konstruktoru (např. tak se vytvoří jediný bezparametrický **privátní** konstruktor)
- Místo volání konstruktoru se nabídne veřejná metoda `getInstance()`, která zjistí, jestli (jediná) instance již existuje.
- Pokud ne, vytvoří jí (voláním privátního konstruktoru) a uloží do statického atributu třídy. Pokud již existuje, rovnou se instance vrátí.

Singleton — metoda `getInstance()`

A dále zajišťuje jednoduchý přístup k jediné instanci odkudkoliv:

- Metoda `getInstance()` je statická, tj. kdokoliv odkudkoliv může zavolat `PersonCounter.getInstance().increaseNumPersons()`.
- Pokud chceme automaticky počítat vytvoření instance třídy `Person` z předchozího příkladu, lze toto volání jednoduše přidat konstruktorů třídy `Person`

Kompletní příklad singletonu

```
public class PersonCounter {
    // here will be the singleton instance (object)
    private static PersonCounter counter = null;
    private int peopleCount = 0;
    // "private" prevents creation of instance via new PersonCounter()
    private PersonCounter() {}
    // creates the singleton unless it exists
    public static PersonCounter getInstance() {
        if(counter == null) {
            counter = new PersonCounter();
        }
        return counter;
    }
    public void increaseNumPeople() {
        peopleCount++;
    }
```

```
    public int howManyPeople() {
       return peopleCount;
    }
}
```

Import statických prvků

- Už jsme ukázali výše, že statické třídy i metody můžeme importovat:

```
import static java.lang.System.out;
public class HelloWorld {
  public static void main(String[] args) {
    out.println("Hello World!");
  }
}
```

- relevantní pouze pro *statické metody a proměnné*
- vhodně použitelné pro některé věci z Java Core API, např. Math
- Wikipedia:Static import

Problémy importu statických prvků

```
import static java.lang.System.out;
// developer is reading the code
out.println("Hello World!"); // what is out?
// few lines above:
PrintStream out;
// ahh ok, I thought it was System.out
```

- Takže někdy pak nevíme, jestli jde o statický import a nebo jen o lokální proměnnou/metodu.
- A jakmile něco nevíme na první pohled, JE TO ŠPATNĚ! :-)