# První program v Javě, třída, objekt

Tomáš Pitner, Radek Ošlejšek, Marek Šabo

# Program "Hello World!"

- Abychom měli kam náš kód psát, vytvoříme třídu Demo s hlavní funkci main, která se zavolá při spuštění programu.

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

- Metoda main musí být **veřejná** (public), **statická** (static) a **nevrací žádnou hodnotu** (void). *Klíčová slova pochopíte časem, není to teď důležité.*
- Metoda musí mít parametry typu String (řetězec), které se předávají při spuštění z příkazového řádku do pole String[] args.

# Motivace třídy I

- Jak reprezentovat složitou strukturu, aby se s ní dobře pracovalo?
- Příklad: Osoba s *jménem* a *rokem narození*

```
class Person {
   String name;
   int yearBorn;
}
```

- Části objektu nastavíme i zjistíme stejným způsobem jako v jazyce Python:

```
k.name = "Karel"; // set name to Karel
String karelsName = k.name; // get name value
```

Jednotlivé části (jméno, rok narození) nazýváme *atributy*.

# Motivace třídy II

- Někdy bychom rádi měli funkce, které pracují přímo s částmi struktury.
- Pamatujeme si rok narození, ale co když chceme zjistit věk?
- Jak lehce zjistit informace o naší ~~struktuře~~ — *třídě*?

```
public class Person {
   private String name;
   private int yearBorn;
   public int getAge() {
     return 2018 - yearBorn;
   }
   public void printNameWithAge() {
     System.out.println("I am " + name + " and my age is " + getAge());
   }
}
```

V kódu třídy se nyní objevila klíčová slova `public` a `private`. Nemají vliv na funkcionalitu, ale na "viditelnost", na možnost či nemožnost z jiného kódu danou třídu nebo její vlastnost vidět a použít. Logicky `public` asi (určitě :)) půjde použít vždy a odevšad.

# Vlastnosti třídy

- Třída představuje strukturu, která má *atributy* a *metody*.

**Atributy**

- jsou nositeli datového obsahu, údajů, "pasivních" vlastností objektů
- to, co struktura má, *z čeho se skládá*, např. auto se skládá z kol
- definují **stav** objektu, nesou **informace** o objektu

**Metody**

- jsou nositeli "výkonných" vlastností, *schopností* objektů něco udělat
- to, *co dokáže struktura dělat* — pes dokáže štěkat, osoba dokáže mluvit
- definují **chování** objektu (může být závislé na stavu)

# Vytvoření konkrétní osoby

- Máme třídu Person, to je něco jako *abstraktní šablona* pro objekty—osoby.
- Jak vytvořím **konkrétní** osobu s jménem Jan?

```
public class Demo {
 public static void main(String[] a) {
   Person jan = new Person();
   jan.name = "Jan";
   jan.yearBorn = 2000;
   System.out.println(jan.name);
   System.out.println(jan.yearBorn);
 }
}
```

== Poznámky k příkladu `Demo` - **Třída** Person má vlastnost name a age, to jsou její *atributy*. - **Objekt** jan typu Person má vlastnost name s hodnotou *Jan* a yearBorn s hodnotou *2000*. - Klíčová slova `public` a `private` vám z Pythonu nejsou známá, zde v Javě i jiných jazycích označují "viditelnost" položky — jednoduše řečeno, co je veřejné a co soukromé. Soukromé atributy "vidíme" jen z metod třídy, v níž jsou uvedeny.

== Objekt

- Objekt je jeden **konkrétní jedinec** příslušné třídy.
- Všechny vytvořené objekty nesou stejné vlastnosti, např. všechny objekty třídy Person mají vlastnost name.
- Vlastnosti mají však pro různé lidi různé hodnoty — lidi mají různá jména.
- Konkrétní objekt určité třídy se také nazývá *instance* (jedincem) své třídy.

== Vytváření objektů

- Co znamená new Person()?
- Proč musíme psát Person jan = new Person() a ne jen `Person jan`?

```
Person jan = new Person();
// why not just:
Person jan;
```

- Pouhá deklarace proměnné objektového typu (`Person jan`) žádný objekt nevytvoří.
- K vytvoření slouží operátor `new`.

== Co se děje při vytváření objektů přes `new`

- Alokuje se paměť v oblasti dynamické paměti, tedy na *haldě* (heap).
- Vytvoří se tam objekt a naplní jeho atributy výchozími hodnotami.
- Zavolá se speciální metoda objektu, tzv. konstruktor, který objekt dotvoří.

== Konstruktor

- Slouží k "oživení" vytvořeného objektu bezprostředně po jeho vytvoření:

    - Jednoduché typy, jako například `int`, se vytvoří a inicializují samy a konstruktor nepotřebují.
    - Složené typy, *objekty*, je potřeba vždy zkonstruovat!
- V našem příkladu s osobou operátor `new` *vytvoří prázdný objekt typu Person* a naplní jeho atributy výchozími (default) hodnotami.
- Další přednáška bude věnována konstruktorům, kde se dozvíte víc.

== Třída a objekt

*Třída*

- Je komplexní struktura, reprezentuje prvky z reálného světa (např. pes, člověk).
- Je určitý vzor pro tvorbu podobných objektů (konkrétních psů či lidí).
- Definice třídy sestává převážně z *atributů* a *metod* (říkáme jim také prvky nebo členy třídy).
- Skutečné objekty této třídy pak budou mít prvky, které byly ve třídě definovány.

*Objekt*

- Objekty jsou instancemi "své" třídy vytvořené dle definice třídy a obsahující atributy.
- Vytváříme je operátorem `new`.
- Odkazy na vytvořené objekty často ukládáme do proměnné typu té třídy, např. `Person jan = new Person();`

== Komplexnější příklad I ==

Následující třída `Account` modeluje jednoduchý bankovní účet.

- Každý bankovní účet má jeden *atribut* `balance`, který reprezentuje množství peněz na účtu.
- Pak má *metody*:
    - `add` přidává na účet/odebírá z účtu
    - `writeBalance` vypisuje zůstatek
    - `transferTo` převádí na jiný účet

== Komplexnější příklad II ==

public class Account { private double balance; // 0.0 public void add(double amount) { balance += amount; } public void writeBalance() { System.out.println(balance); } public void transferTo(Account whereTo, double amount) { balance -= amount; whereTo.add(amount); // whereTo is another account } }

- Metoda `transferTo` pracuje nejen se svým "mateřským" objektem, ale i s objektem `whereTo` předaným do metody.

== Komplexnější příklad - definice vs. použití třídy ==

- Třída sama je definovaná v samostatném souboru `Account.java`.
- Její použití pak třeba v `Demo.java`.

public static void main(String[] args) { Account petrsAccount = new Account(); Account ivansAccount = new Account(); petrsAccount.add(100.0); ivansAccount.add(20.0); petrsAccount.transferTo(ivansAccount, 30.0); petrsAccount.writeBalance(); // prints 70.0 ivansAccount.writeBalance(); // prints 50.0 }

== `println` vs. `return` ==

- Pozor na rozdíl mezi vypsáním řetězce a jeho vrácením:

```
public void writeString() {
  System.out.println("Sample text"); // writes it
}
```

```
public String returnString() {
  return "Sample text"; // does not write it
}
```