

# Objekty, třídy, instance

Karel Richta a kol.

Katedra počítačů  
Fakulta elektrotechnická  
České vysoké učení technické v Praze

© Karel Richta, 2016

Objektové modelování, xx36OMO  
09/2016, Lekce 1

<https://cw.fel.cvut.cz/wiki/courses/xxb36omo/start>



# Cíl předmětu

- Proč OOP:
  - Objektově-orientovaný přístup dnes drtivě převažuje ve všech aspektech vývoje softwaru: analýze, návrhu i implementaci.
  - Cílem tohoto kurzu je dovést studenty k solidnímu porozumění základním pojmům objektové teorie a k jejímu použití v praxi při návrhu software.
- Cílem OOP je především:
  - Zjednodušit pochopení modelovaného problému.
  - Usnadnit údržbu.
  - Usnadnit další rozšiřování programu.

# Sémantická mezera

- **Sémantická mezera:** Rozdíl výkonného hardware na jedné straně a „nevýkonného“ software a jeho tvorby. V současnosti totiž máme výkonný hardware, velmi složitý a hardwarově náročný software, který je schopen plnit naše požadavky avšak za cenu velkých finančních nákladů do hardware, do vývoje software jeho udržování a změn.

**Jinými slovy:** Při analýze potřeb uživatelů mluvíme jazykem blízkým uživatelům. Implementaci ale musím realizovat pomocí hardwaru typu von Neumannova stroje. Tuto sémantickou mezeru se zpravidla pokoušíme řešit pomocí různých modelů, které postupně transformujeme od úrovně uživatelské až po implementační.

# Sémantická mezera

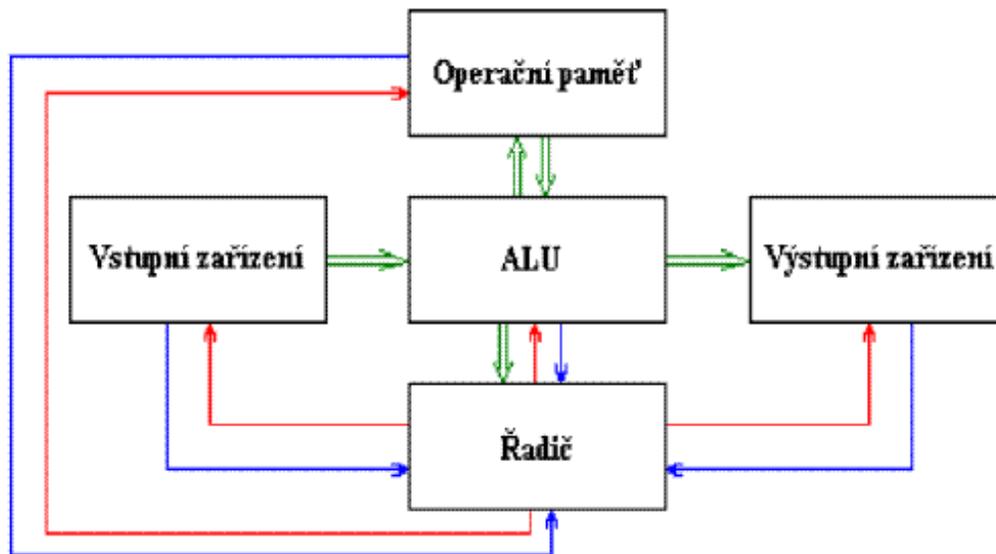
- **Problémy:** Jsme nuceni i ve vyšších programovacích jazycích používat primitivní takzvané „vN operace“. Jedná se o přiřazovací příkazy umožňující přesuny dat a skoky pro změnu toku řízení. Tento fakt se promítá i do různých vnějších podob operačních systémů a různých aplikačních programů. Algoritmy se složitě vyjadřují, rozsáhlé programy jsou neodladitelné, údržba a modifikace hotových programů je problematická a za chodu věnuje procesor více než 50% času systémovým programům.

# V čem je problém von Neumannova stroje?

- zastaralá architektura stroje (1946)
  - primitivní paměť obsahující data i program
  - sekvenční imperativní řízení výpočtu



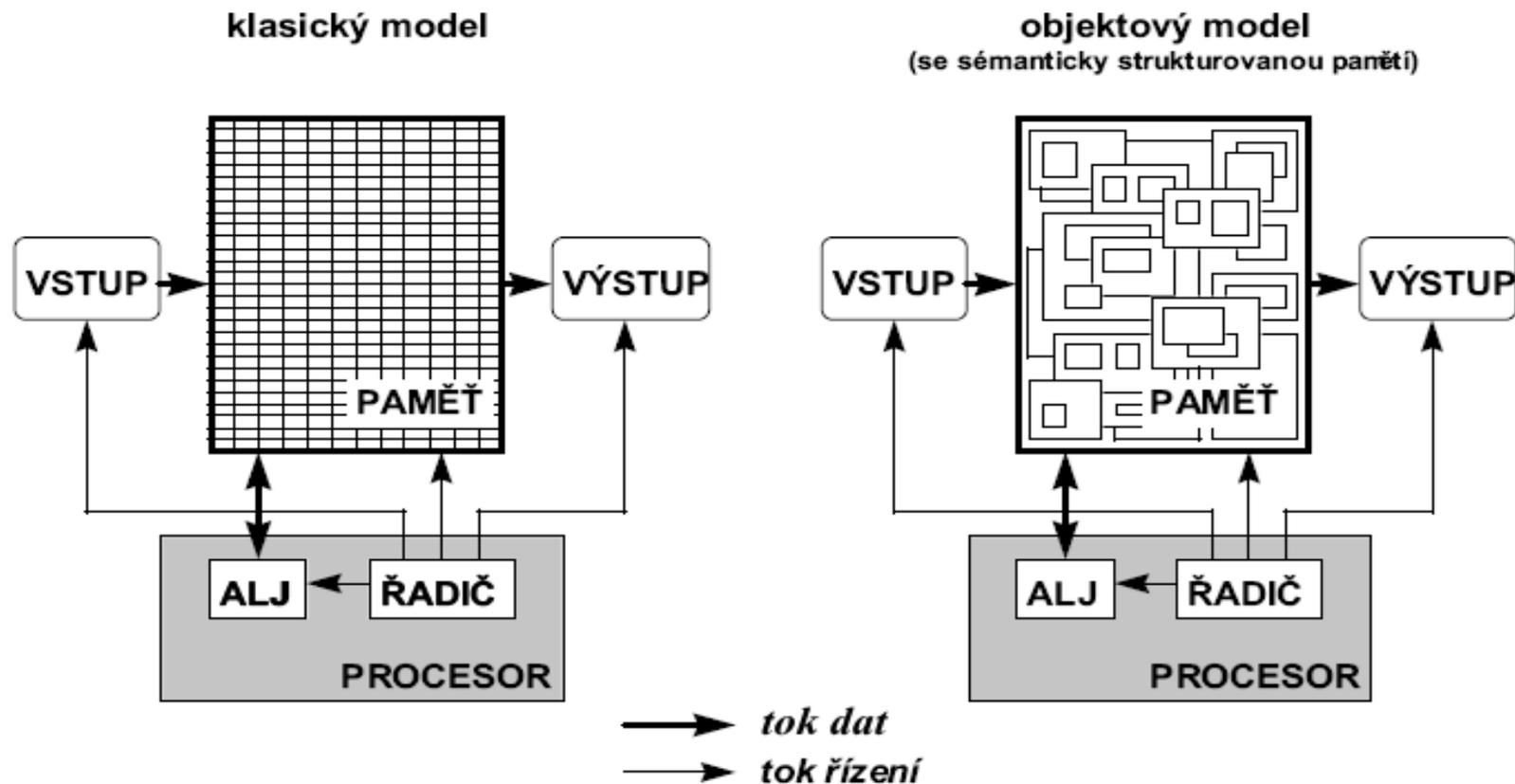
**John von Neumann**



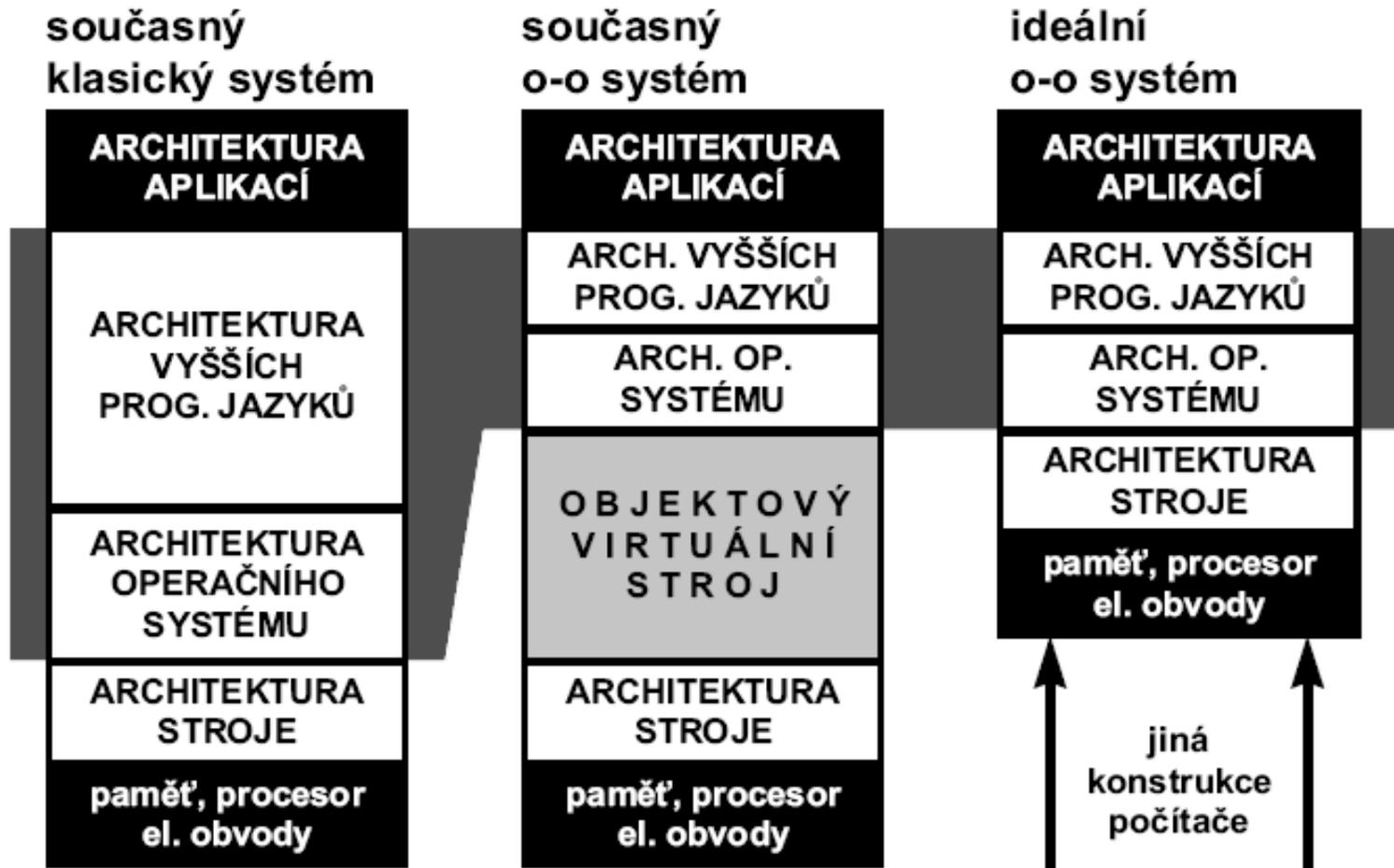
- Tok dat
- Řídicí signály řadiče
- Stavová hlášení řadiči

<http://www.ivt.mzf.cz/seminar/1-principy-a-vyvoj-pocitacu/>

# Sémanticky strukturovaná paměť



# Virtuální stroj

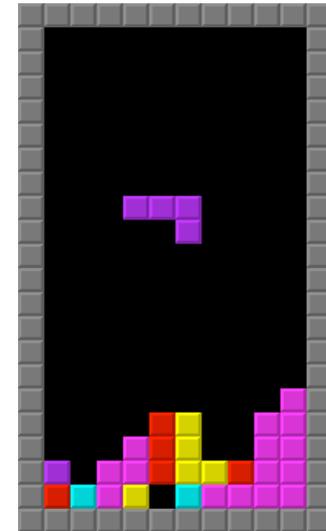


# Svět Objektů

- **Objektově Orientované Programování (OOP):**
  - Vychází z intuitivního chápání světa.
  - Základním konceptem jsou objekty.
  - Objekty mají určité *vlastnosti (properties)* a *schopnosti (capabilities)*.
- **Objekty jsou stavební bloky programu:**
  - Program je tedy soubor interagujících objektů.
- **Objekty mohou modelovat **reálné věci**:**
  - auto, hrnek, židle, zeměkoule.
- **Objekty mohou modelovat **koncepty**:**
  - schůzka, shromáždění, telefonát.
- **Objekty mohou modelovat **procesy**:**
  - hledání cesty bludištěm, seřazení balíčku karet.

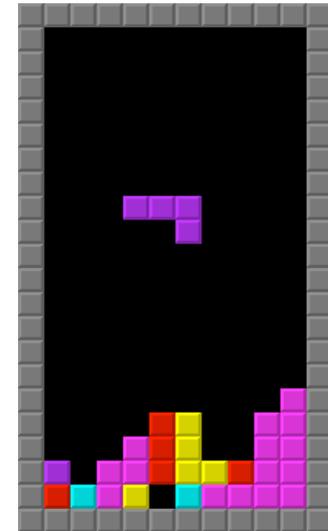
# Příklad - Tetris

- Jaké jsou herní objekty?
- Co musí tyto objekty umět vykonat?
- Jaké vlastnosti tyto objekty mají?



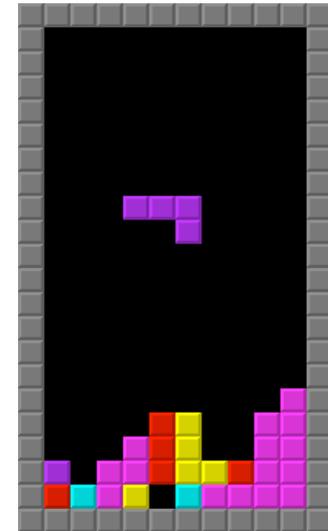
# Příklad - Tetris

- Jaké jsou herní objekty?
  - např. dílky a hrací plán
- Co musí tyto objekty umět vykonat?
- Jaké vlastnosti tyto objekty mají?



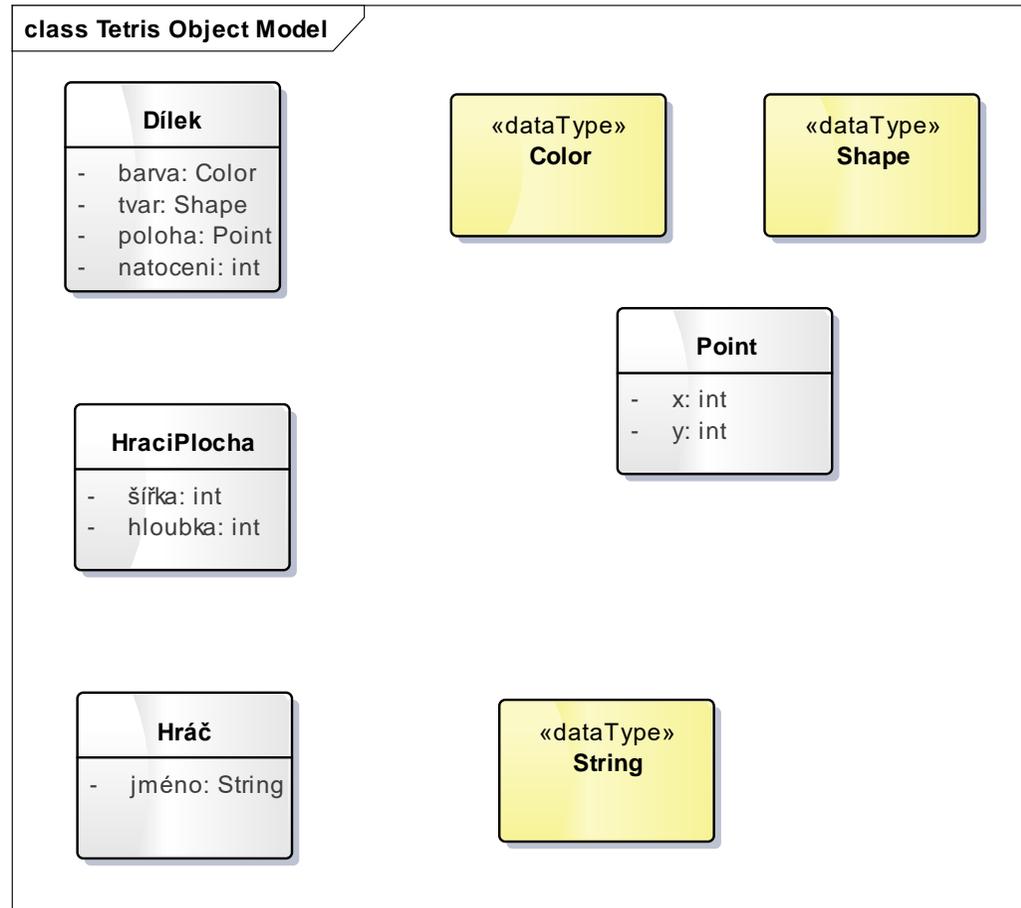
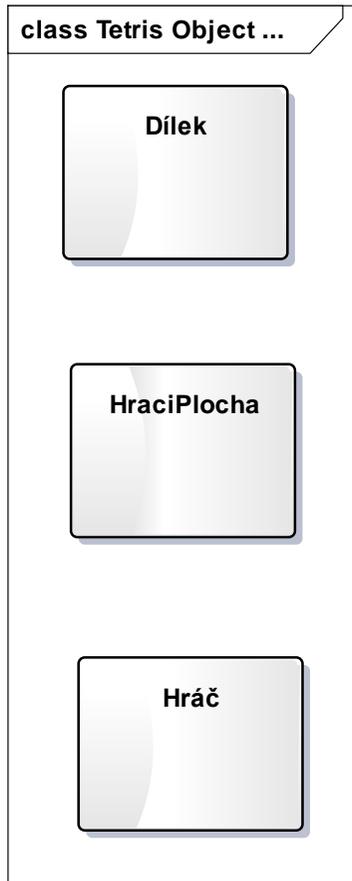
# Příklad - Tetris

- Jaké jsou herní objekty?
  - např. dílky a hrací deska.
- Co musí tyto objekty umět vykonat?
  - dílek:
    - být vytvořen, spadnout, zastavit se, otočit se,
  - hrací deska:
    - být vytvořena, odstranit řádek, zkontrolovat konec hry.
- Jaké vlastnosti tyto objekty mají?



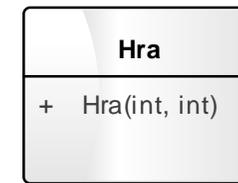
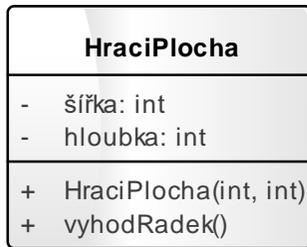
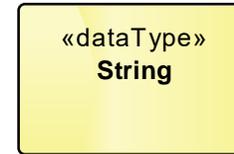
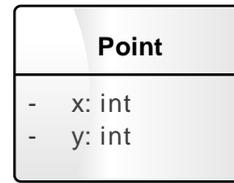
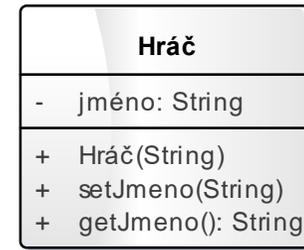
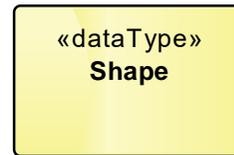
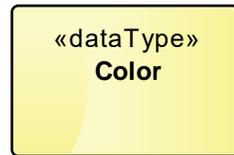
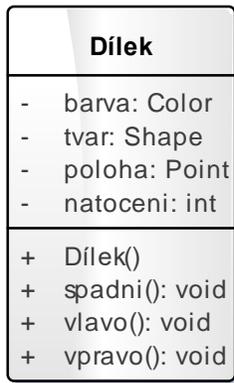


# Jak to zaznamenáme?

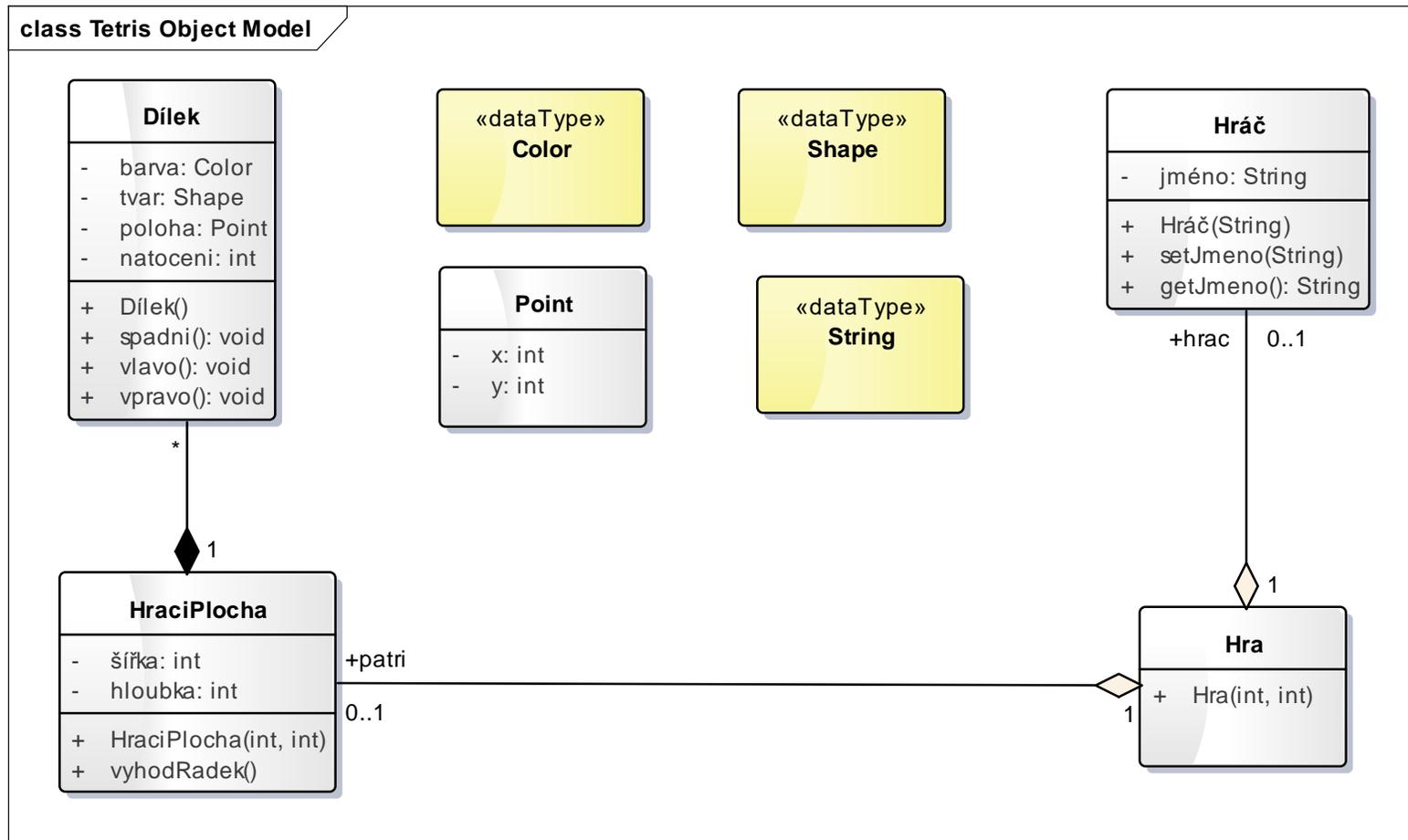


# Jak to zaznamenáme? –pokr.

## class Tetris Object Model



# Jak to zaznamujeme? –pokr.



# Schopnosti Objektů

- Schopnosti (*metody*) objektů umožňují provádět specifické akce.
- Metoda objektu musí být vyvolána nějakým objektem (včetně objektu na kterém je metoda volána).
- Metody objektů dělíme na:
  - konstruktory - inicializují počáteční stav objektu
  - příkazy - mění vlastnosti, tudíž stav objektu
  - dotazy - poskytují odpověď na základě vlastností, stavu objektu

# Příklad

- **Metody popelnice**
  - konstruktor: vytvoř popelnici
  - příkazy: přidej odpad, vyprázdni se
  - dotazy: kolik je odpadu v popelnici?, je otevřeno víko?



# Vlastnosti Objektů

- Vlastnosti (properties) zajišťují unikátnost objektů:
  - Ovlivňují jakým způsobem se metody na objektu chovají.
  - Některé vlastnosti mohou být konstantní, jiné se mohou měnit.
  - Vlastnosti mohou být také objekty.

# Vlastnosti Objektů

- Vlastnosti *objektu* dělíme na:
  - atributy - objekty, které pomáhají *objekt* popsat,
  - komponenty - objekty, které jsou součástí daného *objektu* (objekt se z nich skládá, např. auto *má* kola),
  - asociace - objekty o kterých daný *objekt* ví, ale nejsou jeho součástí (např. auto je v garáži, auto může vědět o garáži).

# Stav Objektu

- Stavem *objektu* nazýváme kolekci všech jeho vlastností společně s jejich hodnotami.
- Stav *objektu* se změní pokud se změní aktuální hodnota(y) některé jeho vlastnosti(i).

# Stav Objektu - Příklad

- Mějme *objekt Auto* s následujícími vlastnostmi:
  - rok výroby,
  - barva.
- Co popisuje stav tohoto *objektu*? Může se změnit *barva*? Může se změnit *rok výroby*?

# Stav Objektu - Příklad

- Mějme *objekt Auto* s následujícími vlastnostmi:
  - rok výroby,
  - barva.
- Co popisuje stav tohoto *objektu*? Hodnotami atributů rok výroby, barva.
- Může se změnit *barva*? Ano, auto můžeme nechat přebarvit.
- Může se změnit *rok výroby*? Ne, jedná se o konstantní vlastnost.

# Třídy a Instance

- Koncept, který jsme zavedli, modeluje každý objekt reálného světa samostatným objektem.
- Jinými slovy, každé auto bychom museli definovat od začátku se všemi vlastnostmi atd.
- Nevýhoda?

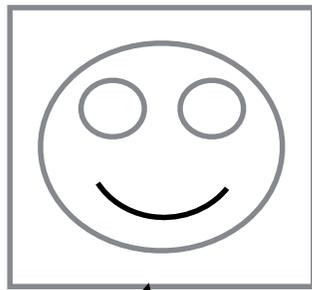
# Třídy a Instance

- Nevýhoda?
- Příliš mnoho práce pro programátora objekty specifikovat.
- Velmi nepraktické, objekty mohou mít mnoho společného:
  - Např. auto má víceméně vždy kola, volant, sedadla, barvu, atd...

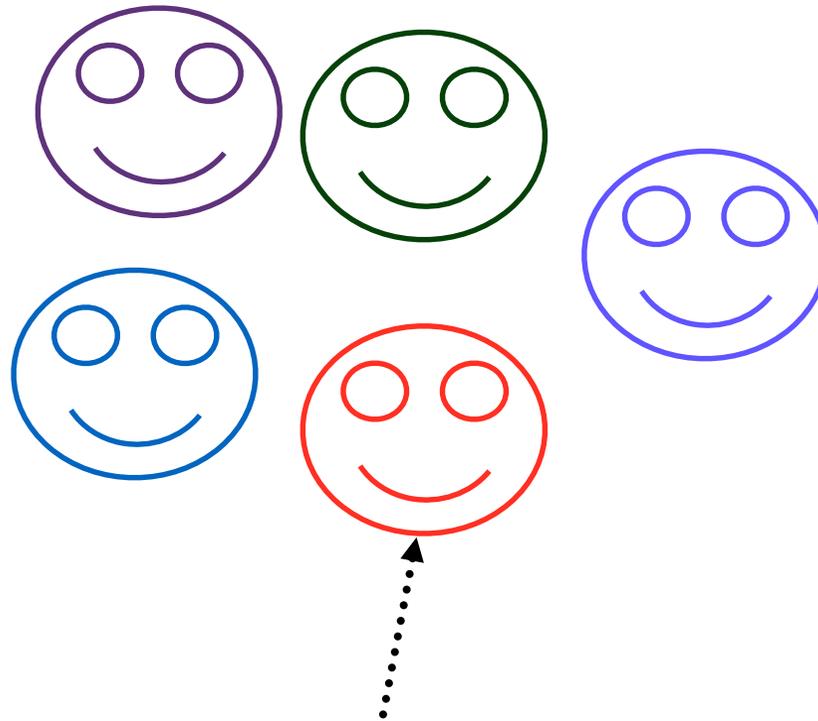
# Třídy a Instance

- Zavedení tříd nám přinese seskupení společných rysů objektů z nich “instancovaných”:
  - společné vlastnosti stačí popsat jednou,
  - poté můžeme vytvářet instance dle libosti, které budou mít tyto vlastnosti společné.

# Třídy a Instance



**Razítko (Třída)**



**Otisk Razítka (Instance)**

# Třída

- Třída je kategorie objektu.
- Definuje vlastnosti a schopnosti společné pro určitou množinu objektů.
  - Např. Všechny popelnice mohou otevírat víko, zavírat víko, vysypávat odpad.
- Třída je šablonou pro vytváření instancí objektů.
- Třídy implementují schopnosti objektů jako metody:
  - Metoda je v jazyce Java posloupnost příkazů.
  - V jazyce Java *odesílatel (sender)* zasílá zprávu *příjemci (receiver)* pomocí volání jedné z příjemcových metod.
- Třídy implementují vlastnosti objektů jako instanční proměnné.
  - Místo v paměti alokované pro daný objekt k ukládání hodnot.

# Instance

- Instance třídy jsou samotnými objekty:
  - Jsou vytvářeny z tříd.
  - Mohou být složeny z dalších instancí tříd.
  - Všechny instance dané třídy mají stejné atributy, ale u každé z nich mohou mít rozdílné hodnoty.
  - Např. Mějme dvě instance třídy `Auto`. Každá z nich má atribut *barva*. První z nich má hodnotu atributu *barva* červená, druhá *modrá*.

# Příklad: Hra v kostky

- Uvažme příklad velmi jednoduché hry s kostkami – máme dvě kostky a házíme. Pokud nám v daném hodu vyjde součet 7, vyhráli jsme, jinak jsme naopak prohráli.
- Objektové řešení vychází z podstaty úlohy – pracujeme s objekty typu „kostka“. To jsou instance obecné třídy „Kostka“.
- Kostka umí metodu „hod“, která nastaví interní hodnotu kostky „hodnotaKteraPadla“ na náhodné číslo a vrátí toto číslo jako výsledek.
- Hru hrají hráči, kteří mají jméno – to je možno jim pomocí metod nastavit a číst (setJmeno/getJmeno).
- Hra spočívá v tom, že hráč hodí kostkami – to nám může zajistit metoda „hrajHru“.



# Třída „Kostka“

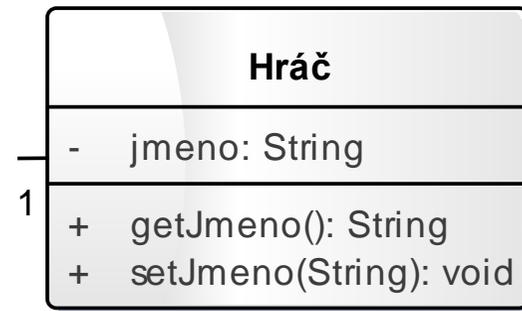
```
import java.util.Random;

class Kostka {
    private int hodnotaKteraPadla;
    private Random random;
    public Kostka() { this.random = new Random(); this.hod(); }
    public int hod() {
        hodnotaKteraPadla = this.random.nextInt(6)+1;
        return hodnotaKteraPadla;
    }
}
```

Kostka	
-	hodnotaKteraPadla: int
+	hod(): int

# Třída Hráč

```
class Hrac {  
    private String jmeno;  
    public Hrac() { jmeno = ""; }  
    public void nastavJmeno(String jm) {  
        jmeno = jm;  
    }  
    public String dejJmeno() { return jmeno; }  
}
```



# Třída HraVKostky

```
class HraVKostky {  
    private Kostka kostka1 = new Kostka();  
    private Kostka kostka2 = new Kostka();  
    private Hrac hrac1 = new Hrac();  
    private String x;  
    private int vysledek;  
    private boolean konec;
```

```
public Hra() {}
```

```
public void hrajHru() {
```

```
    hrac1.nastavJmeno("Pavel");
```

```
    do {
```

```
        System.out.println("Nazdar "+hrac1.dejJmeno()+", chces hrat?: ");
```

```
        Scanner sc = new Scanner( System.in ); x = sc.next(); konec = x.equals("n");
```

```
        if (!(konec)) { vysledek = kostka1.hod() + kostka2.hod();
```

```
            System.out.print("Padlo: "+vysledek);
```

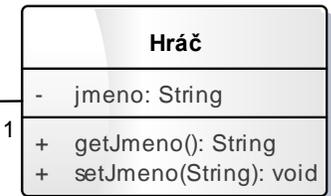
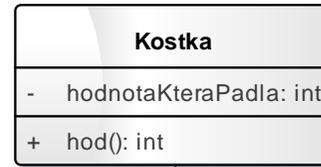
```
            if (vysledek == 7) System.out.println(", vyhral jsi!");
```

```
            else System.out.println(", prohral jsi -:(!");
```

```
        }
```

```
    } while (!(konec));
```

```
} }
```



2

1

1

1

# The End