7. přednáška - třídy, objekty

- **třídy**
- **objekty**
- **atributy tříd**
- **metody tříd**

Třída jako zdroj funkcionality

- **Třída v jazyku Java je programová jednotka** tvořená množinou identifikátorů, které mají třídou definovaný význam a které označují prostředky, které lze v téže či jiných třídách využít.

  Třída je zdrojem metod popisujících **řešení problému rozkladem na podproblémy**.

  Uvnitř třídy mohou být deklarovány **proměnné**, které jsou použitelné jako nelokální proměnné v metodách dané třídy.

Metoda `main`

- **Povinným základem aplikace**, tj. uživatelského programu, **je třída**, ve které je deklarována spouštěcí metoda přesně takto:

```
public static void main( String[ ] args ) {  ... }
```

  Pozn. 1: *Metoda main **musí být statická**, voláme ji dříve než se vytvoří nějaký objekt.*

  Pozn. 2: *V jiných jazycích, např. v C++, lze program vytvořit bez použití třídy.*

- **Každá třída může obsahovat metodu main,** pak se využívá pro:
  - testování funkčnosti objektu,
  - ukázku použití metod objektu.

- **Mnohé třídy nemají deklaraci metody *main***:
  - např. knihovní třída *java.lang.Math* poskytující matematické funkce

Třída jako datový typ

- **Třída je popisem strukturovaného datového typu**, tzn. specifikuje
  - datové prvky potřebné pro objekt (proměnné a konstanty; je v nich uložen stav objektu) a
  - množinu metod pro práci s datovými prvky (manipulují s proměnnými, čímž mění stav objektu).

  Pozn.: Třída sama o sobě nemá přidělenu žádnou paměť. Nedá sepracovat ani s proměnnými ani s metodami (pokud nejsou deklarovány jako **`static`**).

- **Objekt** (nazývá se též **instance** třídy) je datový prvek, který je vytvořen podle vzoru třídy.

  Podle jednoho vzoru třídy lze vytvořit libovolné množství objektů (instancí).

- V jazyku Java lze **objekty vytvářet pouze dynamicky pomocí operátoru *new*** a přistupovat k nim pomocí referenčních proměnných.

Třída

- **Třída je návrhový vzor**, který umožňuje definovat vlastnosti a chování

  vlastnosti … atributy (proměnné a konstanty)

  chování … metody (funkce a procedury)

Příklad: Třída (šablona) **obdélník**

atributy (vlastnosti):

- šířka,
- výška,
- barva,
- pozice na obrazovce, …

metody (chování, reakce na požadavky okolí)

- nastavení barvy,
- výpočet obvodu, obsahu,
- posunutí, …

Obdélník - příklad definice třídy

```
public class Obdelnik {
    Color barva;
    int sirka, vyska;
    Point pozice;

    int vypoctiObvod(){
        return 2*(sirka+vyska);
    }

    int vypoctiObsah(){
        return sirka*vyska;
    }

    void nastavBarvu(Color c){
        barva = c;
    }
}
```

Jméno třídy začíná velkým písmenem

Atributy objektu - definují typ a jména vlastností

Metody objektu - definují chování, schopnosti, reakce

Obdélník - příklad definice třídy

```
public class Obdelnik {
    Color barva;
    int sirka, vyska;
    Point pozice;

    int obvod(){
        return 2*(sirka+vyska);
    }

    int obsah(){
        return sirka*vyska;
    }

    void nastavBarvu(Color c){
        barva = c;
    }
}
```

Jméno třídy začíná velkým písmenem

Atributy objektu - definují typ a jména vlastností

Metody objektu - definují chování, schopnosti, reakce

Metody mohou používat atributy - proměnné.
Neobsahují-li klíčové slovo **static**, pracují s konkrétním objektem.

Speciální metoda - konstruktor třídy

Konstruktor

- nastavuje vlastnosti objektu prostřednictvím parametrů (ale může být i bez parametrů),
- neosahuje návratový typ - nic nevrací, vytváří objekt,
- jméno je totožné se jménem třídy (jediná metoda začínající velkým písmenem),
- tato metoda vytvoří objekt,

```
public class Obdelnik {
    ...
    Obdelnik(int s, int v){
       sirka = s;
       vyska = v;
    }
}
```

- volá se pomocí operátoru **new**.

```
malyObdelnik = new Obdelnik(2,5);
```

nezapomeňte na deklaraci referenční proměnné `malyObdelnik`.

Třída versus objekt

Třída

- návrhový vzor, šablona,
- reprezentovaná zápisem v Javě,
- existuje i mimo program.

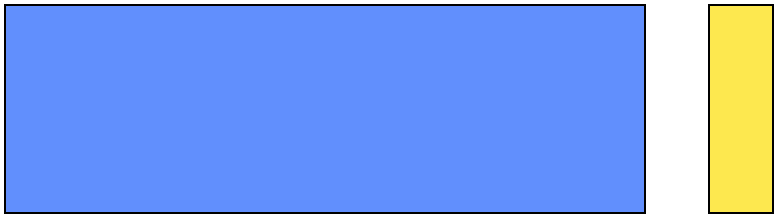
Objekt

- jeden konkrétní výrobek vyrobený podle třídy,
- je vytvořen za běhu programu,
- žije během života programu (lze jej uložit na disk, není reprezentován kódem programu.

Příklad:

třída: **`Obdelnik`**

```
Obdelnik maly = new Obdelnik(1,5);
Obdelnik velky = new Obdelnik(10,5);
```



Přetížení konstruktorů

```
public class Complex {
     double re, im;
     Complex(){
        re = 0.0;
        im = 0.0;
     }
     Complex(double re){
        this.re = re;
        this.im = 0.0;
     }
     Complex(double re, double im){
        this.re = re;
        this.im = im;
     }
}
```

re ... členská „globální“ proměnná třídy

re ... lokální proměnná metody

Operátor **this** je implicitní parametr každé metody, odkazující na „tuto instanci”.

- **Bez něj by lokální proměnná re**, reprezentovaná form. parametrem metody, **zastínila nelokální proměnnou re**, deklarovanou pro třídu.

Přetížení konstruktorů

```
public class Complex {
     double re, im;
     Complex(){
        re = 0.0;
        im = 0.0;
     }
     Complex(double re){
        this.re = re;
        this.im = 0.0;
     }
     Complex(double re, double im){
        this.re = re;
        this.im = im;
     }
}
```

Co vytvoří tyto příkazy?

```
1. Complex c1 = new Complex();
2. Complex c2 = new Complex(1);
3. Complex c3 = new Complex(1,1);
```

Přetížení konstruktorů: Obdelnik

```
import java.awt.Color;
import java.awt.Point;

public class Obdelnik {
    Color barva;
    double sirka, vyska;
    Point pozice;

    // Konstruktory
    Obdelnik(){
        sirka = vyska = 0;
    }
    Obdelnik(double a){
        sirka = vyska = a;
    }
    Obdelnik(double sirka, double vyska){
        this.sirka = sirka;
        this.vyska = vyska;
    }
    Obdelnik(Obdelnik o){
        sirka = o.sirka;          // parametrem může být i jiný objekt
        vyska = o.vyska;
    }
        ...
}
```

Přetížení konstruktorů II

Budeme chtít konstruktor, který u vytvářeného obdélníku specifikuje jeho barvu.

```
public class Obdelnik {
   ...
   Obdelnik(int s, int v){
      sirka = s;
      vyska = v;
   }
   Obdelnik(int s, int v, Color c){
      sirka = s;
      vyska = v;
      barva = c;
   }
}
```

Stejný kód.
Správné by bylo použití jednoho místa pro tento kód.
Jednodušší opravy.

Vzájemné volání konstruktorů

Vytvoříme jeden „univerzální“ konstruktor a ostatní jej budou volat.

```
public class Obdelnik {
   ...
   Obdelnik(int s, int v){
      this(s,v,Color.BLACK);
   }

   Obdelnik(int s, int v, Color c){
      sirka = s;
      vyska = v;
      barva = c;
   }
}
```

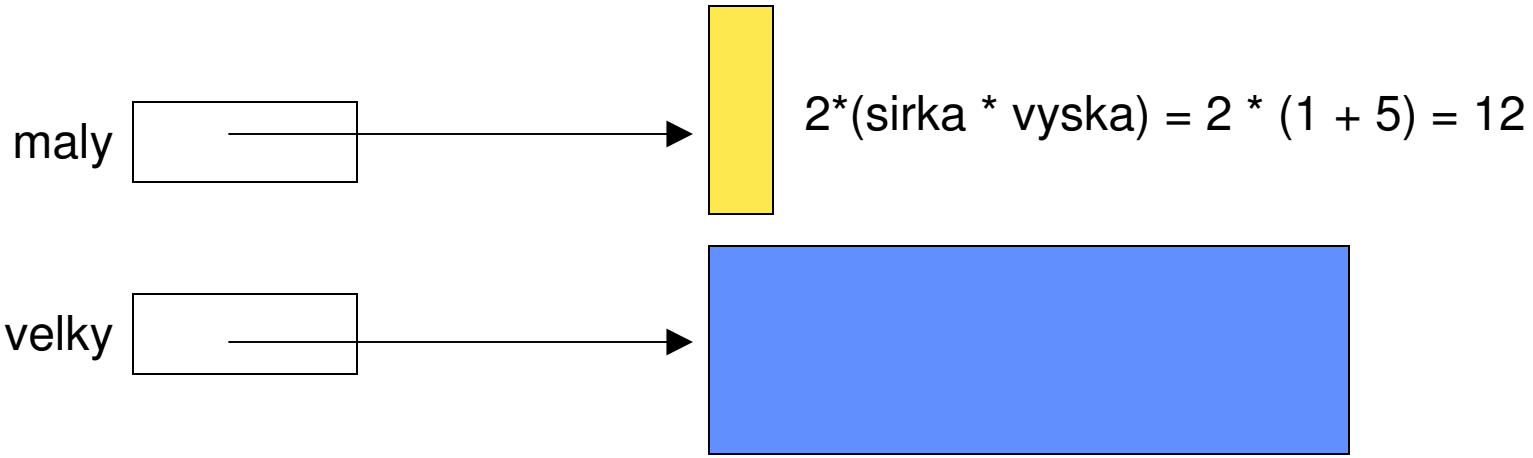
nastavení implicitní hodnoty

volání konstruktoru téže třídy

Třída pro testování obdélníka

```
public class Obdelnik {
        ...
    public static void main(String[] args) {
        Obdelnik maly;
        maly =  new Obdelnik(1,5);
        Obdelnik velky = new Obdelnik(10,5);
        System.out.println("Obvod maleho je" + maly.obvod());
    }
}
```

Není třeba předávat šířku a výšku, každá instance zná své rozměry!!



Statické a instanční metody

- Pod pojmem metoda se skrývají dva druhy metod:
  - **instanční** metody (metody objektů),
  - **statické** metody (metody třídy).

  Oba druhy mohou mít parametry a mohou vracet výsledek.
- **Instanční metoda** označuje operaci nad objektem čili instancí dané třídy.

  Je dostupná jen přes **referenci na objekt**.

  Voláme ji tedy takto:

  ```
  referenčníProměnná.jménoMetody( seznam argumentů )
  ```

  např.

  ```
  maly.vypoctiObvod()
  ```

  „vidí“ statické i nestatické atributy třídy.

  Zkráceně říkáme, že metoda **`vypoctiObvod()`** se volá na objekt **`maly`**.
- **Statická metoda** je dostupná pomocí jména třídy, aniž je nutno vytvářet nějaký objekt.

  Voláme ji tedy takto:

  ```
  JménoTřídy.jménoMetody( seznam argumentů )
  ```

  a také

  ```
  referenčníProměnná.jménoMetody( seznam argumentů )
  ```

Statické metody

```
public class Obdelnik {
   int sirka ….
   static int vratPocetRohu(){
   // sirka = 6;
      return 4;
   }
}
```

CHYBA, statická metoda nevidí instanční proměnné!

```
public class ObdelnikTest {
   public static void main(String[] args) {
      Obdelnik maly =  new Obdelnik(1,5);
      System.out.println("Pocet rohu obdelnika je " +
                         Obdelnik.vratPocetRohu());
      System.out.println("Maly obdelnik ma " +
                         maly.vratPocetRohu() + " rohu.");
   }
}
```

vypíše
4
4

Statické atributy a metody

- Některé třídy obsahují pouze statické atributy a statické metody.
- Knihovna matematických funkcí – třída **java.lang.Math** – obsahuje statické proměnné (zde konstanty typu double) **PI** a **E.**

- Statické metody reprezentující matematické funkce:
  - **sin, cos, tan, …** goniometrické funkce
  - **abs …** absolutní hodnota
  - **min, max**
  - **log …** logaritmus
  - **sqrt …** odmocnina
  - **pow(double a, double b) … a^b**
  - **random …** vrací náhodné **double** číslo z intervalu $\langle 0;1)$
  - **round …** zaokrouhlení
  - a mnohé další

Standardní metody třídy Object

- Každá třída (kromě jediné) v Javě je potomkem třídy **Object**, která implementuje několik základních metod.

  Základní metody třídy Object jsou **toString**

```
public String toString(){
   return getClass().getName() + "@" +
          Integer.toHexString(hashCode());
}
```

  Výsledkem volání *x.toString*( ) je řetěz znakové reprezentace objektu *x.*

  Metodou je zavedena implicitní typová konverze z typu objektu *x* na řetězec, použitá např. při výpisu objektu.

  a **equals**

```
public boolean equals(Object obj) {
   return (this == obj);
}
```

  Vrací true pouze pokud se jedná o stejné objekty; neporovnává položky!!!

Standardní metoda `toString`

```
public class ObdelnikTest {
   public static void main(String[] args) {
      Obdelnik prvni = new Obdelnik(5,7);
      System.out.println("Prvni: " + prvni);
   }
}
```

Vypíše:

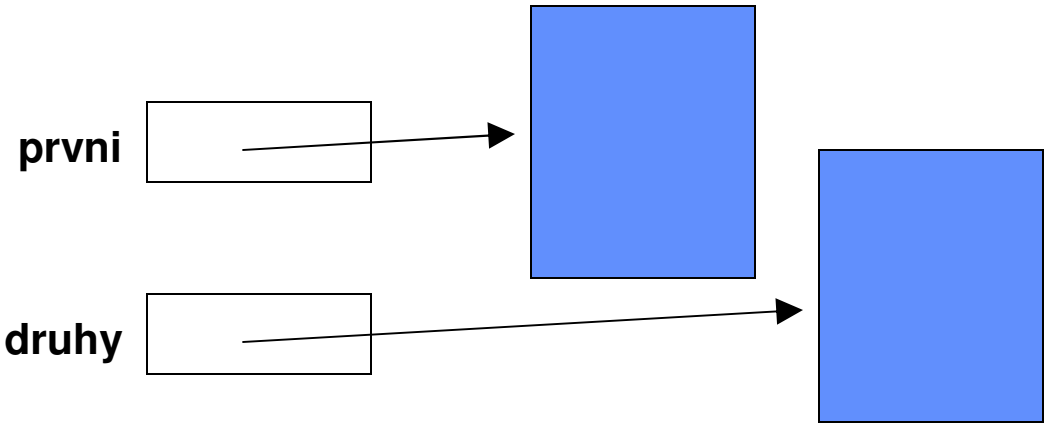
```
   Prvni: alg7.Obdelnik@11b86e7
```

Standardní metoda `equals`

```
public class ObdelnikTest {
   public static void main(String[] args) {
      Obdelnik prvni = new Obdelnik(5,7);
      Obdelnik druhy = new Obdelnik(5,7);
      System.out.println("Stejne? " + (prvni==druhy));
      System.out.println("Stejne? " + prvni.equals(druhy));
   }
}
```

Vypíše

```
Stejne? false
Stejne? false
```



Zastínění metod předka

- **Zastínění** (předefinování) metod **toString** a **equals.**

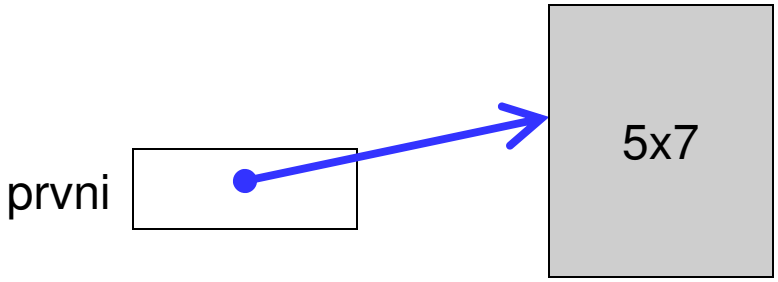
```
public String toString(){
   return ("Obdelnik: " + sirka + " x " + vyska);
}

public boolean equals(Object o){
   // porovnam jen s obdelniky
   if(!(o instanceof Obdelnik))return false;
   Obdelnik obd = (Obdelnik) o;  // Proč přetypování ?
      return (sirka == obd.sirka)&&(obd.vyska==vyska);
}
```

Pomocí operátoru **instanceof** se zjišťuje, zda je referenční proměnná typu nějaké třídy.

Více referencí na jeden objekt, smetí

```
public class ObdelnikTest {
        public static void main(String[] args) {
        Obdelnik prvni = new Obdelnik(5,7);
        Obdelnik druhy = new Obdelnik(5,2);
        druhy = prvni;
        System.out.println("Stejne? " + (prvni==druhy));
        System.out.println("Stejne? " + prvni.equals(druhy));
    }
}
```

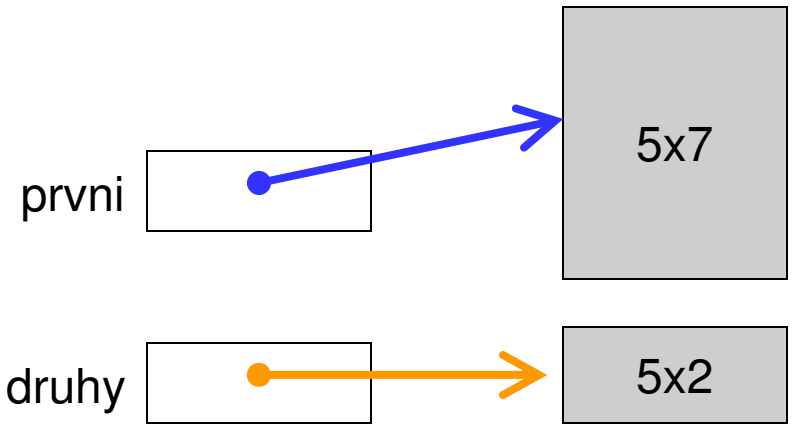
Vypíše
true
true



Více referencí na jeden objekt, smetí

```
public class ObdelnikTest {
        public static void main(String[] args) {
        Obdelnik prvni = new Obdelnik(5,7);
        Obdelnik druhy = new Obdelnik(5,2);
        druhy = prvni;
        System.out.println("Stejne? " + (prvni==druhy));
        System.out.println("Stejne? " + prvni.equals(druhy));
    }
}
```

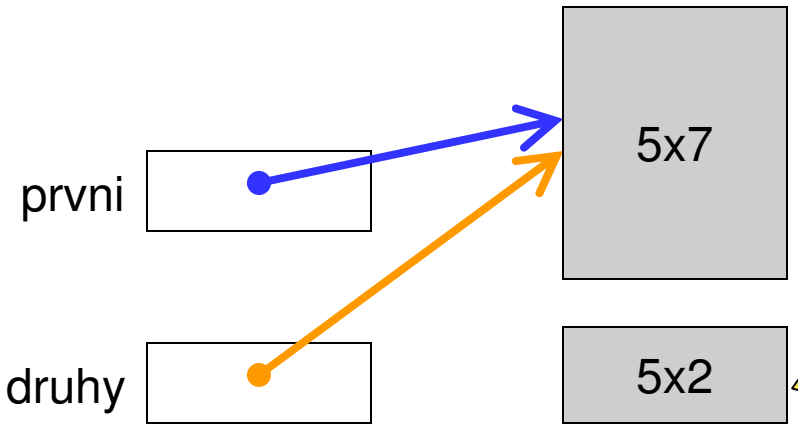
Vypíše
```
true
true
```



Více referencí na jeden objekt, smetí

```
public class ObdelnikTest {
          public static void main(String[] args) {
          Obdelnik prvni = new Obdelnik(5,7);
          Obdelnik druhy = new Obdelnik(5,2);
          druhy = prvni;
          System.out.println("Stejne? " + (prvni==druhy));
          System.out.println("Stejne? " + prvni.equals(druhy));
    }
}
```

Vypíše
```
true
true
```



Smetí - obd. 5x2 se stal nepřístupným.

Paměť přidělená nepřístupným objektům se uvolňuje automaticky (sbírání smetí, garbage collection)