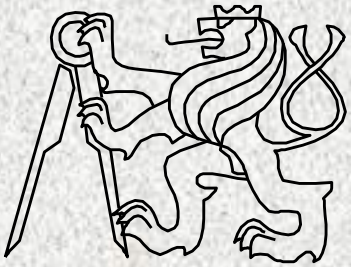
Třída jako datový typ



BD6B36PJV 01

Fakulta elektrotechnická
České vysoké učení technické

Objektový přístup programování

- Modelování problému jako systému spolupracujících tříd
- Třída modeluje jeden koncept
- Třídy umožní generování instancí, objektů příslušné třídy
- Jednotlivé objekty spolu spolupracují, „posílají si zprávy“,
- Třída je „vzorem“ pro strukturu a vlastnosti generovaných objektů
- Každý objekt je charakteristický specifickými hodnotami svých atributů a společnými vlastnostmi třídy

Třídy a objekty

Věci okolo nás lze hierarchizovat do tříd (konceptů).

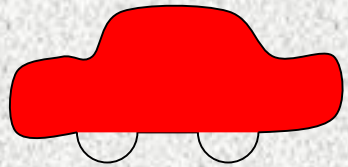
Každá třída je reprezentována svými prvky, objekty dané třídy

Každá třída je charakterizována svými vlastnostmi, svými funkčními možnostmi a svými parametry

Příklad:

Třída „automobil“

- **Funční možnosti automobilu – metody pro ovládání**
- **Parametry – charakteristická data**



Třída a objekty - příklad

definice

definice

METODY: zrychlit, brzdit, zatočit, zastavit;…

TŘÍDA:
AUTO

DATA:jmeno,barva, maxRychlost, prumSpotreba, klimatizace

objekt: autíčko

objekt: závodní

objekt: auťák

objekt: beruška

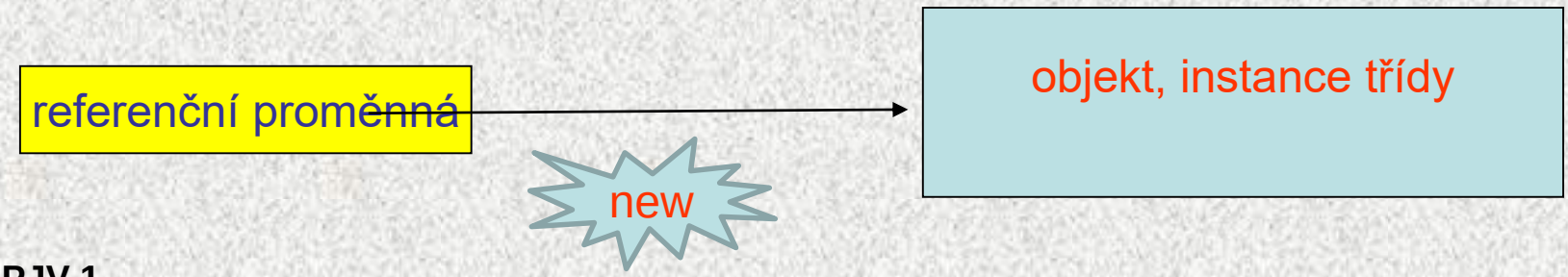
instance, objekty

Třída a objekty

- **Třída** jako šablona pro generování konkrétních *instancí* třídy, tzv. *objektů*
  - data, atributy >>> určují stav objektů
  - metody, funkce >>> určují schopnosti objektů

- **Objekt** – *instance třídy*
  - Jednotlivé instance třídy (objekty) mají stejné metody, ale nacházejí se v různých stavech
    - Stav objektu je určen hodnotami instančních, členských proměnných
  - Schopnosti objektu jsou dány instančními metodami

- V jazyku Java lze objekty (instance tříd) vytvářet ***pouze dynamicky*** pomocí operátoru **`new`** a přistupovat k nim pomocí referenčních proměnných (podobně jako u pole)

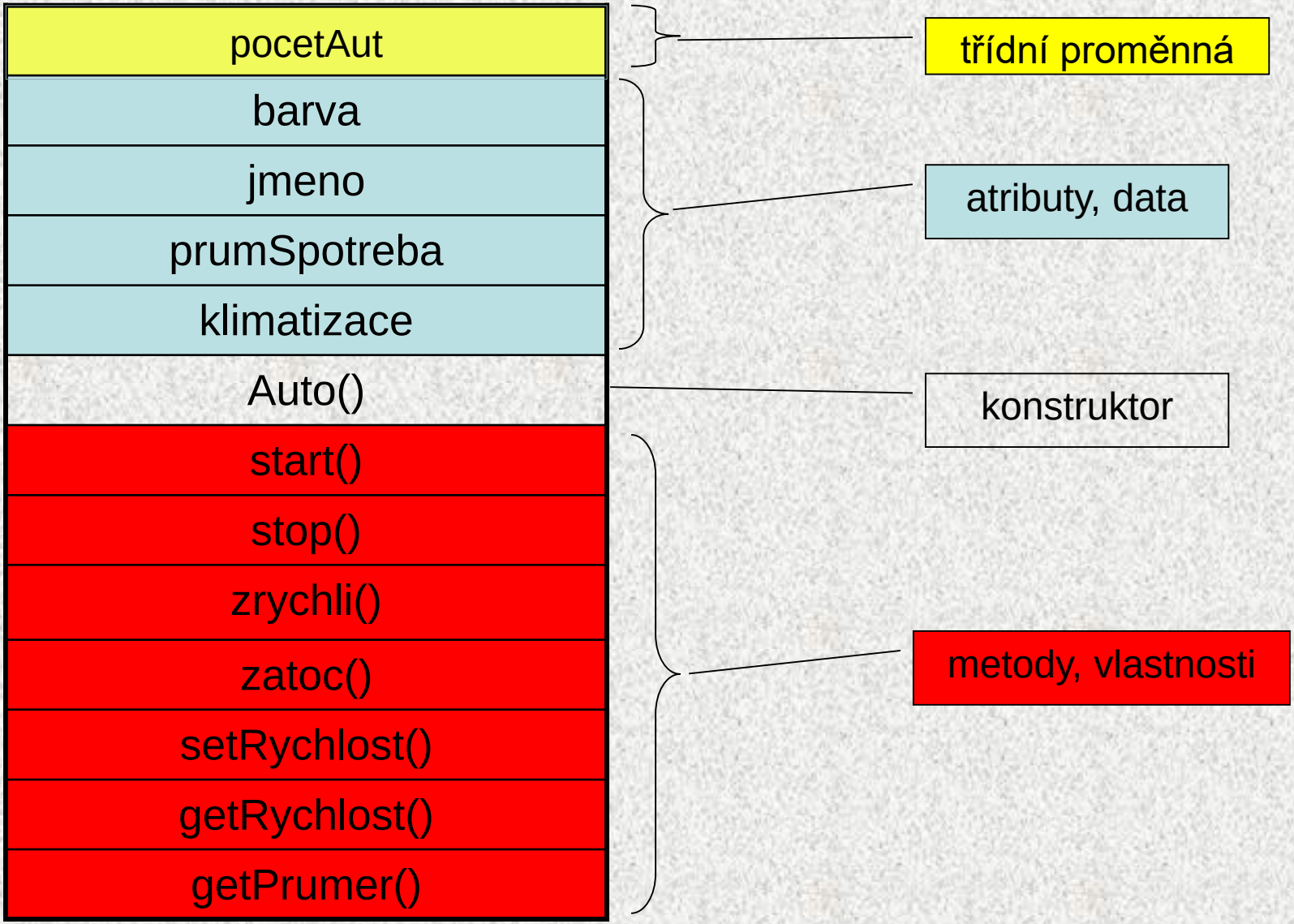


Třída – řešení problému

- Třída v jazyku Java je programová jednotka tvořená množinou identifikátorů, které mají třídou definovaný význam:
  - data – proměnné, konstanty (členské proměnné, datové složky, atributy)
  - metody – pracovně funkce a procedury

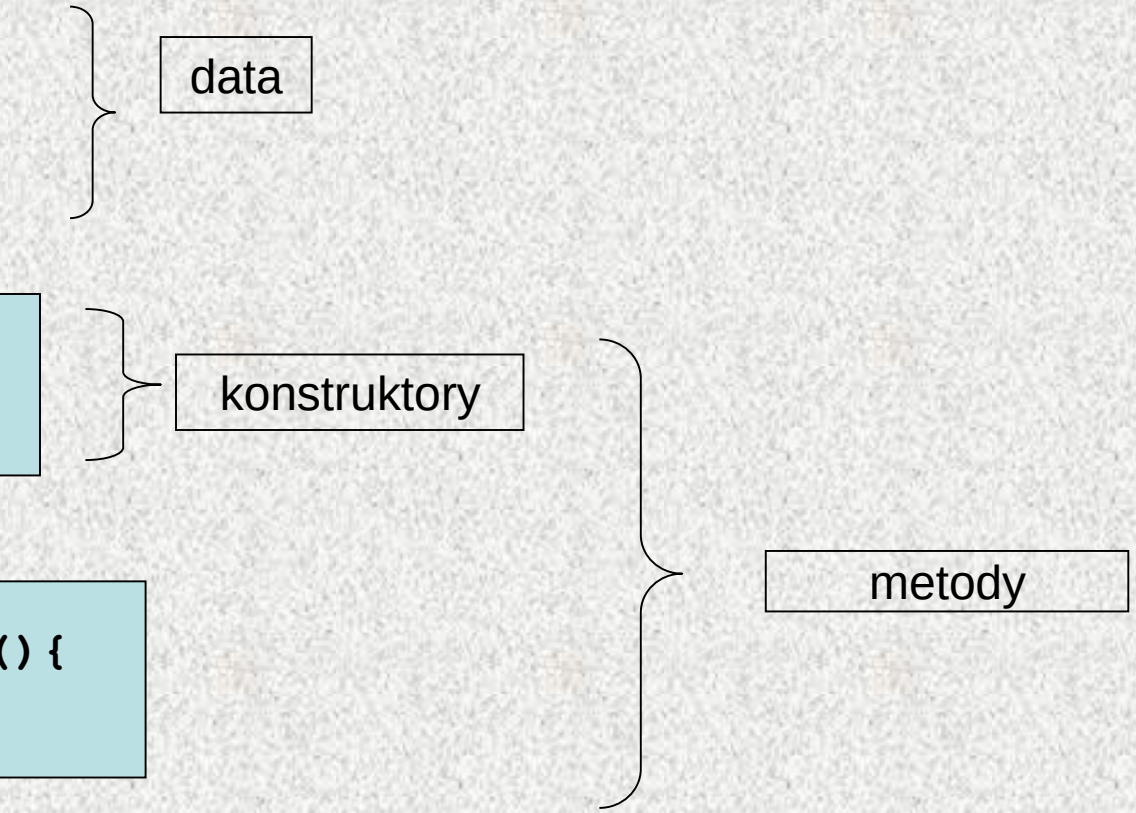
- Systém takto koncipovaných tříd je řešením problému
- Třída je „šablona“ - vzor pro vytváření jednotlivých objektů (instancí)

Třída Auto


pocetAut
barva
jmeno
prumSpotreba
klimatizace
Auto()
start()
stop()
zrychli()
zatoc()
setRychlost()
getRychlost()
getPrumer()
třídní proměnná
atributy, data
konstruktor
metody, vlastnosti

Třída Auto

```
public class Auto {

    String barva;
    …..

    public Auto(){
    }
    ……

    public void startuj(){
    }

}
```



- **konstruktory** - speciální metody pro generování instancí tříd, konkrétních objektů

Třída Auto

```
public class Auto {
 static int pocetAut = 0;// proměnná třídy
    String barva;
…
public Auto(String b, String j, double pS, boolean k){
        barva = b;
…}
public void startuj(){
…}
public void zastav(){
…}
public double getPrumSpotreba() {
 ….}
public void setPrumSpotreba(double pS) {
…}
public void zatoc(){
…}
…
}
```

data

konstruktor

metody, settery, gettery

AutoTest

- Třída pro testování třídy Auto, obsahuje **main**

```
public class AutoTest {
    public static void main(String[] args) {
        Auto a = new Auto("modrá", "autíčko", 6, true);
        a.startuj();
        Auto b = new Auto("žlutá", "závodní", 66, false);
        b.startuj();
        b.setPrumSpotreba(4);
        System.out.println("Max. spotreba " + b.jmeno);
        System.out.println(" je " + b.getSpotreba(66));
        Auto c = new Auto("červená", "beruška", 66, false);
        c.startuj();
        a.zrychli();
        b.zrychli();
        Auto d = new Auto("stříbrná", "auťák", 66, false);
        d.startuj();
        a.zastav();
        c.zatoc();
        c.zastav();
}
```

Testování třídy Auto

Startuje auto autíčko
Jede 1 aut
Startuje auto závodní
Jede 2 aut
Max. spotreba závodní
je 264.0
Startuje auto beruška
Jede 3 aut
Auto: autíčko zrychluje
Auto: závodní zrychluje
Startuje auto auťák
Jede 4 aut
Auto: autíčko zastavuje
Jede 3 aut
Auto: beruška zatáčí
Auto: beruška zastavuje
Jede 2 aut

Objekty, shrnutí

- Objekt - datový prvek, instance třídy, dynamicky vytvořen podle „vzoru“- třídy
- Objekt je strukturován tzn. skládá se z jednotlivých položek tzv. atributů

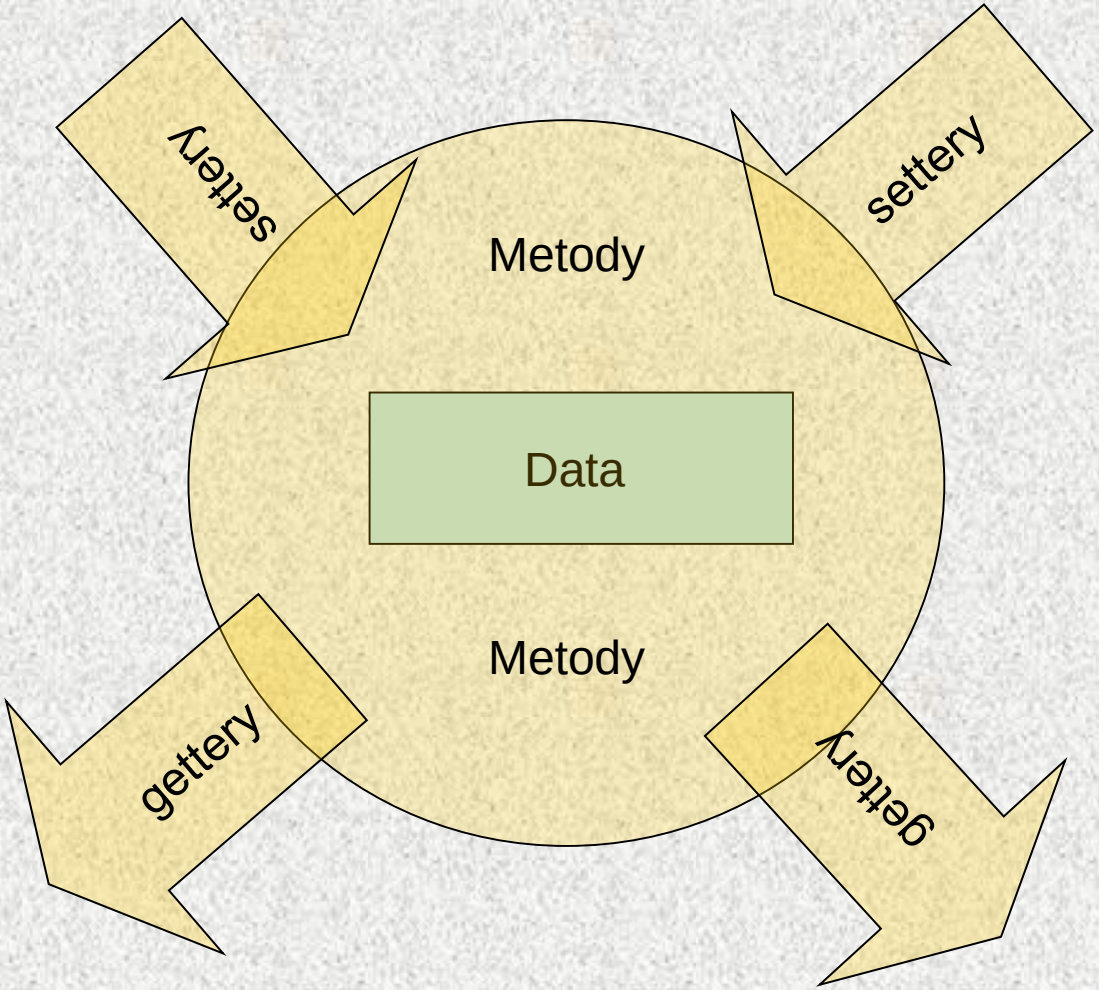
- Třída bez vytvořené instance (objektu) může „pracovat“ pouze „staticky“ (mohou být použity jen její statické metody či proměnné – procedurální přístup), instance musí být vytvořena pomocí operátoru **`new`**
- Existují dva druhy neprimitivních datových typů (referencované):
  - objekty ~ heterogenní objekt skládající se z položek různého typu
  - pole ~ homogenní objekt skládající se z položek stejného typu
- Položky objektu označujeme též jako
  - členské proměnné (member variables)
  - datové složky
  - atributy objektu

Pojmy objektového programování

- Základní vlastností objektového přístupu
  - Zapouzdření
  - Dědičnost
  - Polymorfizmus

- Základní pojmy objektového přístupu
  - Třída
  - Objekt
  - Hierarchie tříd
  - Kompozice objektů

Zapouzdření



Principy objektového programování

- Realita je modelována koncepty, třídami
- Třídy mohou generovat objekty, instance třídy, konstruktorem
- Objekty jsou charakterizovány svými vlastnostmi, individuálními hodnotami a společnými metodami
- Chod programu je tvořen vykonáváním metod jednotlivých objektů (zasílání zpráv)
- Metody jsou určeny svým rozhraním, mohou být přetíženy
- Třídy mohou vytvářet hierarchie, v Javě pouze jedna nadřízená třída, podřízená třída specializace nadřízené
- Třídy mohou být tvořeny i kompozicí jiných tříd
- Vlastnosti a metody objektů jsou zapouzdřené podle potřeby
- Metody objektů, použitelné na různé objekty, pracují podle aktuálního objektu – princip polymorfismu
- Polymorfismus lze implementovat v Javě pomocí **`abstract class`** (hierarchie tříd) nebo **`interface`** (vnucené metody)

Objektově orientovaný styl

- Hlavní zásady:
  - Při rozkladu problému specifikujeme nové datové typy (datové abstrakce), tzn. specifikujeme operace, které se budou s daty těchto typů provádět
  - Pro realizaci datových abstrakcí použijeme objekty a třídy
- Připomeňme si charakteristiku objektu:
  - datové položky a metody objektu jsou dány typem objektu - třídou
  - objekt se může nacházet v různých stavech daných hodnotami datových položek (atributů) objektu
  - chování objektu je dáno metodami (operacemi), které jsou pro něj definovány

Třídy a objekty

- Třída popisující nový datový typ obsahuje:
  - deklarace položek, ze kterých se skládají objekty daného typu
  - deklaraci konstruktoru, kterým se inicializují vytvořené objekty
  - deklarace metod, které realizují operace s objekty
- Položky se obvykle deklarují tak, aby nebyly z vnějšku objektu přístupné:
  `private` *typ jméno*;
- Konstruktor je inicializační operace, která má jméno třídy a nevrací žádnou hodnotu; schéma deklarace konstruktoru třídy *T* (konstruktor by měl být z vnějšku přístupný!! ☺):

```
public T(specifikace parametrů) {
   tělo konstruktoru
}
```

- Ve třídě může být deklarováno několik konstruktorů, které se liší počtem a/nebo typy parametrů (přetěžování)
- Obvyklé schéma deklarace metody (přístupnost metody je dána logikou problému):

```
public typ jméno(specifikace parametrů) {
   tělo metody
}
```

Čítač jako datový typ

- Čítač zavedeme jako datový typ s operacemi

  zvetsit, *zmensit, nastavit* a *hodnota*

  a s datovými položkami

  hodn a *pocHodn*

- Grafické vyjádření:

- Poznámka: hodnota položky *pocHodn* bude stanovena při vytvoření objektu

Třída Citac

```
public class Citac {
  private int hodn;
  private int pocHodn;

  public Citac(int ph) {
     pocHodn = ph;
     hodn = ph;
  }

  public void zvetsit() {hodn++;}

  public void zmensit() {hodn--;}

  public void nastavit() {hodn = pocHodn;}

  public int hodnota() {return hodn;}
}
```

Použití čítače jako objektu

```
public static void main(String[] args) {
    int volba;
    Citac citac = new Citac(0);
//        Citac citac1 = new Citac(3);
    do {
      System.out.println("Hodnota = "+citac.hodnota());
      volba = menu();
      switch (volba) {
        case 1: citac.zvetsit(); break;
        case 2: citac.zmensit(); break;
        case 3: citac.nastavit(); break;
     //       case 4: System.out.println("nic"); break;
      }
    } while (volba>0);
    System.out.println("Konec");
  }
}
```

Třída a objekty – příklad

Příklad obdélník - popisuje objekty „reálného světa“

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 setBarvu(Color c){
        barva = c;
    }
}
```

Jméno třídy začíná velkým písmenem

Atributy objektu - definují typ a jména vlastností

Metody objektu - definují chování, schopnosti, reakce

Konstruktor třídy - speciální metoda

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

Konstruktor

Konstruktor, vlastnosti:

- vytvoří objekt
- případně nastaví vlastnosti objektu
- jméno konstruktoru je totožné se jménem třídy (jediná metoda začínající velkým písmenem)
- volání pomocí operátoru **new**, např.

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

- neobsahuje návratový typ - nic nevrací, vytváří objekt
- není-li konstruktor vytvořen, je vygenerován implicitní konstruktor s prázdným seznamem parametrů
  - je-li konstruktor deklarován, implicitní zaniká

Přetěžování konstruktorů

- Konstruktory jsou metody, které vytvoří objekt a nastaví jeho počáteční hodnoty.

- Příklad
  - Požadujeme 3 typy konstruktorů pro Obdelnik:
    - bez parametrů - vytvoří obdélník o stranách 0x0,
    - 1 parametr - vytvoří čtverec,
    - 2 parametry - šířka x výška

```
public class Obdelnik {
Obdelnik(){
     sirka = vyska =0;
}
Obdelnik(int a){
     sirka =  vyska = a;
}
Obdelnik(int s, int v){
      sirka = s;
      vyska = v;
}
```

Přetěžování konstruktorů II

- 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, `this`

- Vytvoříme jedem „univerzální“ konstruktor a ostatní jej budou volat

```
public class Obdelnik {
    …
Obdelnik(int s, int v, Color c){
        sirka = s;
        vyska = v;
        barva = c;
    }
}
Obdelnik(int s, int v){
    this(s,v,Color.BLACK);
    }
Obdelnik(){
    this(0,0,Color.BLACK);
    }
```

nastavení implicitní hodnoty

volání konstruktoru téže třídy

Operátor `this`

- Každý objekt má implicitní operátor **`this`**, který obsahuje odkaz na "svou" instanci
  - Hodnotou operátoru **`this`** je odkaz na objekt pro který byla metoda zavolána (implicitní parametr odkazu na objekt)
  - Umožňuje přístup k vlastním instančním proměnným v instančních metodách
  - Používá se v přetížených konstruktorech

```
public class Bod {
//fiktivně platí Bod this=new Bod();
double x, y; //instanční proměnné
public Bod(double x, double y) {
this.x = x; this.y = y;
}
public Bod(double y) {
this(0,y);
}
```

Operátor `this`

- Podobně funguje operátor **`this`** i pro metody:
  - pokud je instanční metoda volána z jiné instanční metody té samé třídy, potom se volá pomocí operátoru **`this`** (operátor se může vynechat)
  - pokud se volá metoda z jiného kontextu, uvádí se před jejím jménem přístup k příslušné instanci (tečka notace) např. předané parametrem

Poznámka: ve statické metodě nelze použít **`this`**, není jasné k jaké třídě by se vztahoval !

Vlastnosti konstruktorů, další vlastnosti

- Jméno konstruktoru je totožné se jménem třídy
- Konstruktor nemá návratovou hodnotu (ani `void`)
- Předčasně lze ukončit činnost konstruktoru `return`
- Konstruktor má parametrovou část jako metoda - může mít libovolný počet a typ parametrů
- V těle konstruktoru použít operátor **`this`**, odkaz na příslušný konstruktor s týmž počtem, pořadím a typem parametrů - nepíše se jméno třídy
- Konstruktor je zpravidla vždy `public` (!)
  - //třída java.lang.Math jej má private – proč?

Instanční metody

- Operace s objekty se realizují pomocí instančních metod
- Metody mohou mít parametry a mohou vracet výsledek
- Volání, tj. užití, metody *m* na objekt referencovaný proměnnou *p* má tvar:

```
p.m(seznam argumentů)
```

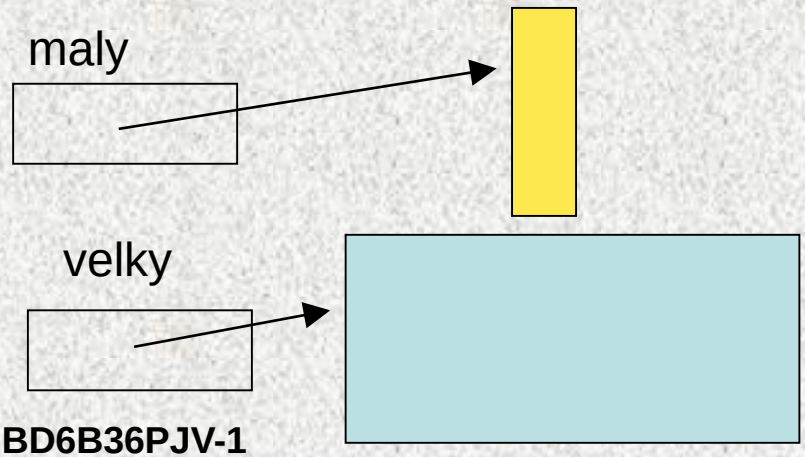
Zkráceně říkáme, že „metoda *m* se volá na objekt *p*“

Instanční metody - příklad

```
public class Obdelnik {
    Color barva;
    int sirka, vyska;
    Point pozice;
….
    int vypoctiObvod() {
        return 2 * (sirka + vyska);
    }

    int vypoctiObsah() {
        return sirka * vyska;
    }

    void setSirku(int s) {
        barva = c;
    }

    int  getSirku() {
        return sirka;
    }

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

Instanční metody - příklad

```
public class ObdelnikTest {
public static void main(String[] args) {
Obdelnik maly = new Obdelnik(1, 5);
Obdelnik velky = new Obdelnik(10, 5);

System.out.println("Obvod maleho je " + maly.vypoctiObvod());
System.out.println("Obvod velkeho je " +velky.vypoctiObvod());
System.out.println("Sirka velkeho = " + velky.getSirku());
velky.setSirka(0);
System.out.println("Sirka velkeho = " + velky.getSirku());
System.out.println("Obsah maleho je " + maly.vypoctiObsah());
System.out.println("Obsah velkeho je " +velky.vypoctiObsah());
```



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

Reprezentace objektu - metoda `toString`

- **`String toString()`**
  - je metodou každého objektu, nepřekreje-li se, pak jméno třídy + hash kód např. prednaska.Obdelnik@da5b
  - výsledkem volání **`maly.toString()`** je řetězec tvořící znakovou reprezentaci obdélníka **`maly`** (metodou je zavedena implicitní typová konverze z typu **`Obdelnik`** na typ **`String`**), často se překrývá!

Příklad

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

```
System.out.println("Velky je  " + velky);
System.out.println("Maly je  " + maly);
```

Příklad: třída `Complex`

- Příkladem třídy jako datového typu je třída **`Complex`**
  - hodnotami typu jsou komplexní čísla tvořená dvojicemi čísel typu *double* (reálná a imaginární část)
  - množinu operací tvoří obvyklé operace nad komplexními čísly (absolutní hodnota, sčítání, odčítání, násobení a dělení)

Třída Complex

```
  public class Complex {
// datové složky
  double re=0;
  double im=0;
// konstruktory
  public Complex() {}
  public Complex(double r) {re=r;}
  public Complex(double r, double i) {re=r; im=i;}
// metody (operace)
  public double getAbs() {
    return Math.sqrt(re*re+im*im);}
  public Complex prictiK(Complex c) {
    return new Complex(re+c.re, im+c.im);}
  public Complex odeMneOdecti(Complex c) {
    return new Complex(re-c.re, im-c.im);}
  public String toString() {
    return "["+re+", "+im+"]";}
```

Instanční metody - pokračování

- Příklady metod definovaných třídou *Complex* pro objekty typu *Complex*:
- **`double abs()`**
  - výsledkem volání **`c.abs()`** je absolutní hodnota komplexního čísla *c*
- **`public Complex prictiK(Complex c) {`**
  - výsledkem volání **`c1.prictiK(c2)`** je reference na nový objekt typu **`Complex`**, jehož hodnotou je součet komplexních čísel *c*1 a *c*2

Třída Complex

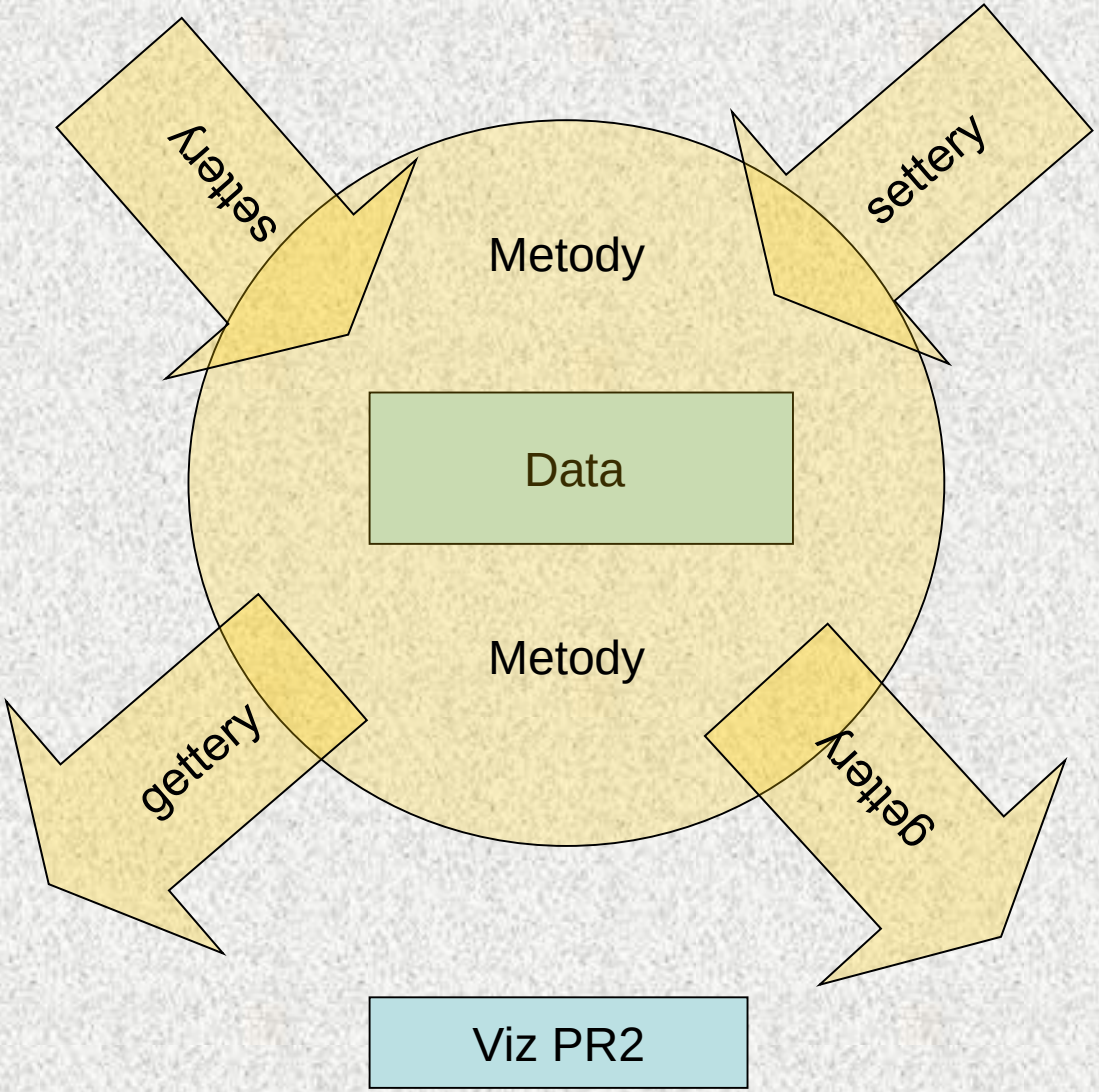
```
public static void main(String[] args) {
    Complex c1=new Complex(3,4); // konstrukce objektu
    Complex c2=new Complex(1,1); // konstrukce objektu
    System.out.println(new Complex());
    System.out.println(c1);
    System.out.println(c1.getAbs());  // |c1|
    System.out.println(c1.prictiK(c2)); // c1 + c2
  }
}
```

```
[0.0, 0.0]
[3.0, 4.0]
5.0
[4.0, 5.0]
```

Struktura objektu, přístup ke položkám

- Hodnota objektu je strukturovaná, tzn. skládá se dílčích hodnot, které mohou být obecně různého typu (heterogenní datová struktura – na rozdíl od pole)
- Objekt je tedy abstrakcí paměťového místa skládajícího se z částí, ve kterých jsou uloženy dílčí hodnoty - nazývají se položkami objektu (složkami, atributy, instančními proměnnými, fields, attributes)
- Položky objektu jsou označeny jmény, která mohou (ale nemusí) být třídou zveřejněna, zásadně se nezveřejňují

Zapouzdření



Viz PR2

Zapouzdření, settery, gettery, accessory

- Accessory = settery + gettery
- Public metody, které umožňují přístup a nastavení **`private`** či **`protected`** atributů objektů

```
public class GrO{
   private int poziceX;
...
  public void setPoziceX(int newX){
   if(newX>0)poziceX = newX;
  }
  public int getPoziceX(){
      return poziceX;
  }
  ...
 }
```

setter
- jméno = “set” + jméno atributu
- atribut nesmí být public – Proč?
- kontroluje přípustnost hodnot,

getter
- jméno = “get” + jméno atributu
- vrací hodnotu atributu

Struktura objektu, přístup ke složkám

- Příklad: objekty (instance) třídy **`Complex`** jsou tvořeny dvěma položkami typu *double*, jejichž jména jsou **`re`** a **`im`** a třída tato jména nezveřejňuje
- Veřejnou položku se jménem ***f*** objektu, který je referencován proměnnou ***p***, lze označit zápisem

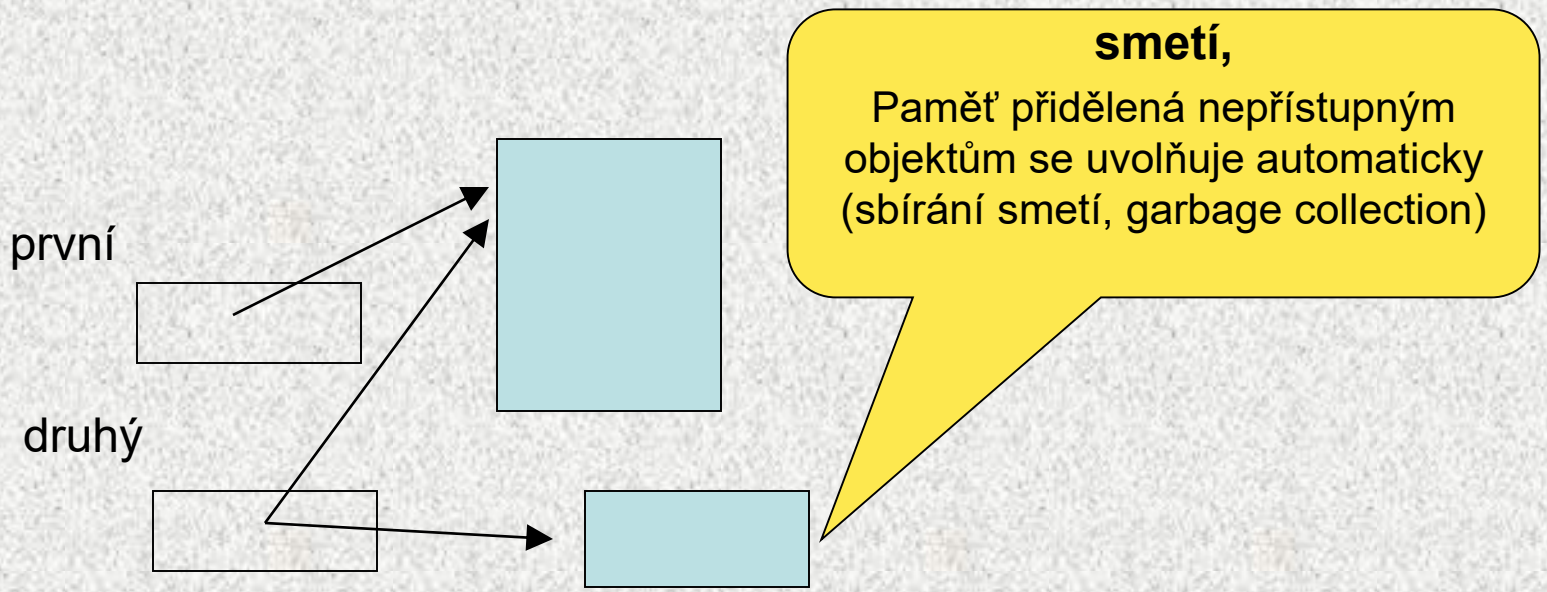
```
p.F
```

Příklad: objektu vytvořenému deklarací

```
private double re=0;
private double im=0;
  Complex c = new Complex();
  public void setRe(double reP){re = reP;}
  public void setIm(double imP){im = imP;}
  double getIm(){return this.im;}
  double getRe(){return this.re;}
  c.setRe(1);// c.re = 1;
  c.setIm(2);// c.im = 2;
```

Více referencí na jeden objekt, smetí

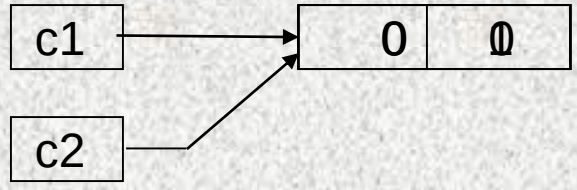
```
public class ObdelnikTest {
public static void main(String[] args) {
Obdelnik prvni = new Obdelnik(5,7);
Obdelnik druhy = new Obdelnik(5,2);
druhy = prvni;
System.out.println("Stejne? "+(prvni==druhy));
System.out.println("Stejne?"+prvni.equals(druhy));
    }
}
```



Více referencí na jeden objekt

- Objekt může být referencován i více referencemi
- Jestliže hodnotu referenční proměnné typu *T* přiřadíme jiné referenční proměnné téhož typu, pak obě proměnné referencují tentýž objekt
- Příklad:

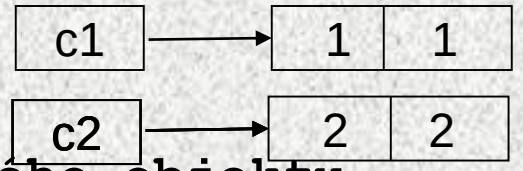
```
Complex c1 = new Complex();
Complex c2 = c1;
c1.im = 1;
System.out.println(c2.im);
// vypíše 1
```



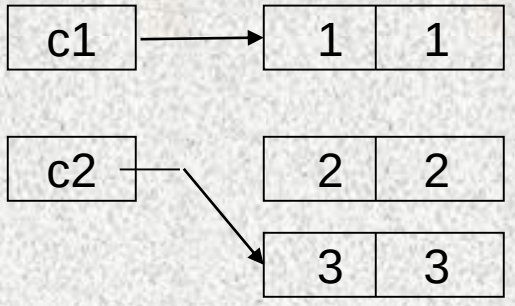
Sbírání smetí

Objekt se stane „smetím“, není-li přístupný pomocí žádné reference

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




```
c2 = c2.plus(c1); // vznik nového objektu
```



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

Role funkce **`main`**

- Třída nemusí obsahovat deklaraci hlavní funkce **`main`**
  - třída bez hlavní funkce *main* nepředepisuje program, který lze spustit, ale zavádí prostředky, které lze v jiných třídách využít – „knihovnu“
- Příkladem je:
    - knihovní třída *Math* poskytující matematické funkce
    - třída *Scanner* poskytující jednoduché funkce pro vstup a výstup
    - třída Auto
- Metoda **`main`** musí být statická, voláme ji dříve než se vytvoří nějaký objekt
- Třída, která obsahuje metodu **`main`** se využívá pro:
  - spouštění programu
  - testování funkčnosti objektu, ukázku použití metod objektu
- *Poznámka:*
  - *Třída s hlavní funkcí* **`main`** *tvořící základ programu je specialitou jazyka Java*
    - *v jiných jazycích, např. v C++, lze program vytvořit bez použití třídy*
  - *Význam parametru args v* **`public static void main( String[ ] args ) { ... }`**

Statické metody

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

Instanční metody mají přístup ke statickým strukturám „kdykoli“

```
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()+ " ro
    }
}
```

Je to možné, ale nelogické!!!

Statické versus instanční metody

- Třída může definovat dva druhy metod:
  - statické metody – metody třídy, procedury a funkce
  - instanční metody – metody objektů
- Metody obou druhů mohou mít parametry a mohou vracet výsledek nějakého typu
- Statická metoda označuje operaci (dílčí algoritmus, řešení dílčího podproblému), jejíž vyvolání (provedení) obsahuje jméno třídy, jméno metody a seznam skutečných parametrů

```
jméno_třídy.jméno_metody(seznam skutečných parametrů)
```

- (lze využít i volání pomocí referenční prom., ale není doporučeno)

Statickým metodám třídy odpovídají v jiných jazycích procedury (nevracejí žádnou hodnotu) a funkce (vracejí hodnotu nějakého typu)

- Instanční metoda označuje operaci nad objektem (instancí (!)) dané třídy, jejíž vyvolání obsahuje referenční proměnnou objektu, jméno metody a seznam skutečných parametrů

```
referenční_proměnná.jméno_metody(seznam skut. parametrů)
```

- Statickým metodám třídy budeme i nadále říkat procedury a funkce
- Instančním metodám budeme zkráceně říkat metody

Statické atributy a metody

Některé třídy obsahují pouze statické atributy a statické metody, tak to bylo dosud!!!!

Knihovna matematických funkcí - třída **`java.lang.Math`** obsahuje statické proměnné (zde konstanty typu **double**) **PI** a **E**

Pozn: Příklad je i třída Complex

Statické atributy a metody - Math

statické metody reprezentující matematické funkce:

```
double x = Math.sin(0,5);
```

- **`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 <0;1)
- **`round`** ... zaokrouhlení
- a mnohé další

$$a^{\frac{1}{3}} = \sqrt[3]{a}$$

Balíky (package) a přístupová práva

Platí pro proměnné, konstanty, metody, třídy:

- **`Private`**

  Autorizovaný přístup, zapouzdření

- **`.....`**

  Přístup jen na jeden balík

- **`Protected`**

  Z libovolné třídy téhož balíku a z odvozené třídy,

- **`public`**

  Porušený autorizovaný přístup, může se ke všemu

Balíky (package) a přístupová práva

Specifikátor	Táž třída	Jiná třída téhož balíku	V podtřítě téhož balíku	V podtřídě jiného balíku	V jiné třídě jiného baliku
`private`	Ano				
.....	Ano	Ano	Ano		
`protected`	Ano	Ano	Ano	ano	
`public`	Ano	Ano	Ano	Ano	Ano