Třída jako zdroj funkcí

- Třída v jazyku Java je programová jednotka tvořená množinou identifikátorů, které mají třídou definovaný význam
- Základem uživatelského programu v jazyku Java je třída, ve které
  - musí být deklarována hlavní funkce *main*
  - mohou být deklarovány další pomocné funkce (procedury)
  - mohou být deklarovány statické proměnné, které jsou použitelné jako nelokální proměnné ve funkcích dané třídy
- Takto koncipovaná třída je zdrojem funkcí popisujících řešení problému rozkladem na podproblémy
- 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
- Příkladem je:
  - knihovní třída *Math* poskytující matematické funkce
  - třída *Sys* poskytující jednoduché funkce pro vstup a výstup

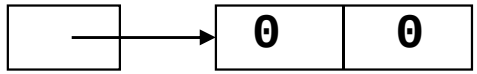
Třída jako datový typ

- Třída s hlavní funkcí *main* tvořící základ programu je specialitou jazyka Java (v jiných jazycích, např. v C++, lze program vytvořit bez použití třídy)
- Obecně je třída popisem strukturovaného datového typu, tzn. specifikuje
  - množinu hodnot datových objektů skládajících se ze složek, a
  - množinu operací s datovými objekty
- Datové objekty typu třída (zkráceně jen objekty) se nazývají též instancemi třídy
- V jazyku Java lze objekty (instance tříd) vytvářet pouze dynamicky pomocí operátoru *new* a přistupovat k nim pomocí referenčních proměnných (podobně jako u pole)
- Třídy jako datové typy se nejprve naučíme používat pasivně
- Příkladem třídy jako datového typu je třída *Complex* v balíku *sugar*
  - hodnotami 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)

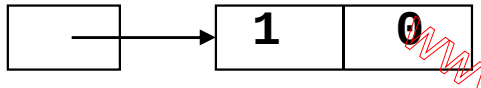
Konstruktory

- Třída *Complex* umožňuje vytvořit a inicializovat objekt třemi způsoby:

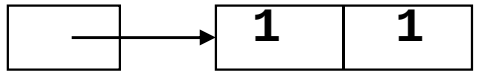
```
Complex c1 = new Complex();
```




```
Complex c2 = new Complex(1);
```




```
Complex c3 = new Complex(1,1);
```



- Tyto tři způsoby inicializace objektu jsou dány třemi konstruktory třídy *Complex*
  - konstruktor bez parametrů *Complex*() inicializuje vytvořený objekt hodnotami 0,0
  - konstruktor s jedním parametrem *Complex*(*double re*) inicializuje vytvořený objekt hodnotami *re*,0
  - konstruktor se dvěma parametry *Complex*(*double re, double im*) inicializuje vytvořený objekt hodnotami *re* a *im*

Instanční metody

- Operace s objekty se realizují pomocí instančních metod (dále jen metod)
- Metody mohou mít parametry a mohou vracet výsledek nějakého typu
- Volání (aplikace) metody *m* na objekt referencovaný proměnnou *p* má tvar:
  `p.m(`*`seznam skutečných parametrů`*`)`

  Zkráceně budeme říkat „metoda *m* se volá na objekt *p*“
- Příklady metod definovaných třídou *Complex*:
  `double abs()`
  – výsledkem volání *c.abs*() je absolutní hodnota komplexního čísla *c*
  `Complex plus(Complex y)`
  – výsledkem volání c1.*plus*(*c*2) je reference na nový objekt typu *Complex*, jehož hodnotou je součet komplexních čísel *c*1 a *c*2
  `String toString()`
  – výsledkem volání *c.toString*() je řetězec tvořící znakovou reprezentaci komplexního čísla *c* (metodou je zavedena implicitní typová konverze z typu *Complex* na typ *String*)
  `a další`

Příklad použití třídy Complex

```
package alg7;

import sugar.Complex;
import sugar.Sys;

public class KomplexniCisla {
  public static void main(String[] args) {
    Complex c1 = new Complex(10);
    Complex c2 = new Complex(3,4);
    Sys.pln("abs("+c1+")="+c1.abs());
    Sys.pln("abs("+c2+")="+c2.abs());
    Complex c3 = c1.plus(c2);
    Sys.pln(c1+"+"+c2+"="+c3);
  }
}
```


- Program vypíše:

```
abs([10.0, 0.0])=10.0
abs([3.0, 4.0])=5.0
[10.0, 0.0]+[3.0, 4.0]=[13.0, 4.0]
```

Statické ~ instanční metody

- Rekapitulace významu termínu „metoda“ v jazyku Java
- Třída může definovat dva druhy metod:
  - statické metody
  - instanční metody
- 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)

  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

Struktura objektu

- Hodnota objektu je strukturovaná, tzn. skládá se dílčích hodnot, které mohou různého typu
- Objekt je tedy abstrakcí paměťového místa skládajícího se z částí, ve kterých jsou uloženy dílčí hodnoty a které se nazývají položkami (složkami, atributy, instančními proměnnými, fields, attributes) objektu
- Položky objektu jsou označeny jmény, která mohou (ale nemusí) být třídou zveřejněna
- Příklad: objekty (instance) třídy *Complex* jsou tvořeny dvěma položkami typu *double*, jejichž jména jsou *re* a *im* a třída tato jména zveřejňuje
- Veřejnou položku se jménem *f* objektu, který je referencován proměnnou *p,* lze označit zápisem

  `p.f`
- Příklad: objektu vytvořenému deklarací

  `Complex c = new Complex();`

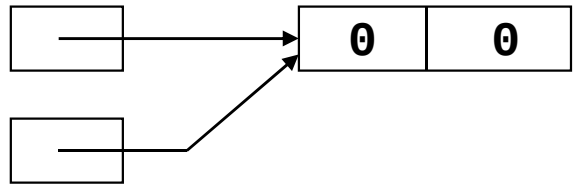
  lze změnit hodnoty položek pomocí příkazů

  `c.re = 1;`
  `c.im = 1;`

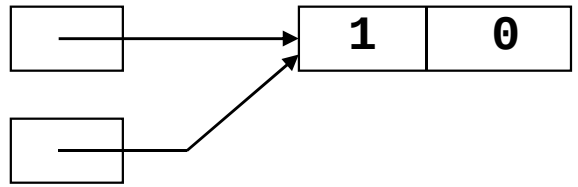
Více referencí na jeden objekt

- Objekt může být referencován několika referenčními proměnnými
- 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.re = 1;
```

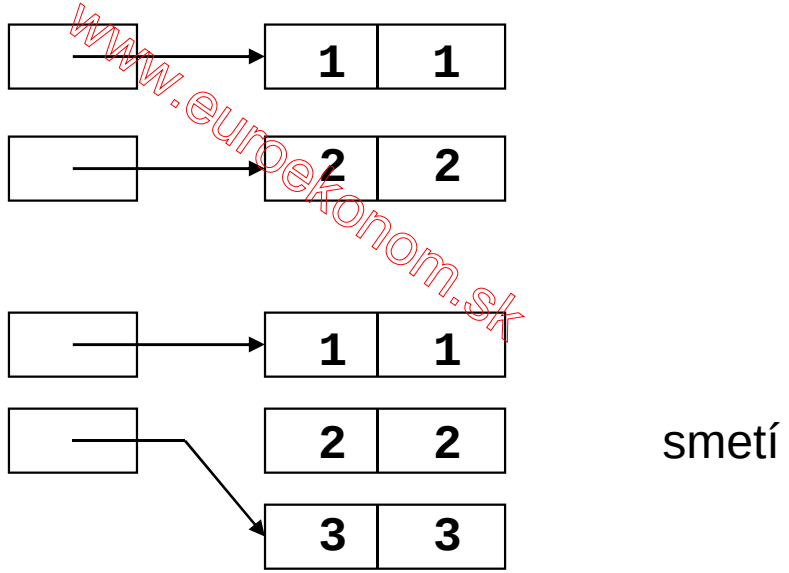



```
Sys.pln(c2.re);     // vypíše 1
```

Sbírání smetí

- Dynamicky vytvořený objekt se stane „smetím“, jestliže není přístupný prostřednictvím žádné referenční proměnné
- Příklad:

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

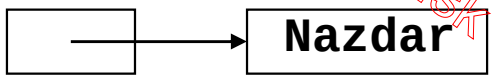


- Paměť přidělená nepřístupným objektům se uvolňuje automaticky (sbírání smětí, garbage collection)

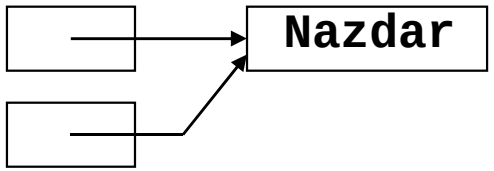
Třída String

- Objekty knihovní třídy *String* jsou řetězce znaků
- Od ostatních tříd se liší třemi specialitami:
  - objekt typu *String* lze označit literálem (posloupnost znaků uzavřená mezi uvozovky)
  - hodnotu objektu typu *String* nelze změnit
  - jedna z operací (konkatenace neboli zřetězení) není realizována metodou, ale označuje ji operátor +
- Příklady referečních proměnných typu *String:*

```
String str = "Nazdar";
```




```
String str1 = str;
```



Operace s řetězci

- Spojení řetězců (konkatenace) +
- Příklad:
  "abc" + "123" výsledek je **"abc123"**
- Jestliže jeden operand operátoru + je typu *String* a druhý je jiného typu, pak druhý operand se převede na typ *String* a výsledkem je konkatenace řetězců
- Příklady:
  "abc" + 5 výsledek je **"abc5 "**
  "a" + 1 + 2 výsledek je **"a12"**
  "a" + 1 + (-2) výsledek je **"a1-2"**
- Porovnávání řetězců
  - relační operátory == a != porovnávají reference, nikoliv obsah řetězců
  - pro porovnání řetězců na rovnost slouží metoda *equals*

```
String s1 = "abcd";
String s2 = "ab";
String s3 = s2 + "cd";
Sys.pln(s1==s3);          // vypíše false
Sys.pln(s1.equals(s3)); // vypíše true
```

Operace s řetězci

- Pro porovnání řetězců slouží další metoda *compareTo*:

```
String s = “abcd”;
Sys.pln(s1.compareTo(“abdc”)); // vypíše –1
Sys.pln(s1.compareTo(“abcd”)); // vypíše 0
Sys.pln(“abdc”.compareTo(s1)); // vypíše 1
```

- Některé další operace:

```
String s = “nazdar”;
int delka = s.length();          // délka je 5
char znak = s.charAt(1);         // znak je ‘a’
String ss = s.substring(2,4);    // ss je “zd”
int z1 = s.indexOf(‘a’);         // z1 je 1
int z2 = s.lastIndexOf(‘a’);     // z2 je 4
```


- Hodnotu referenční proměnné typu *String* lze změnit (odkazuje pak na jiný řetězec), vlastní řetězec změnit nelze

Příklad - palindrom

- Napišme program, který přečte jeden řádek a zjistí, zda se po vynechání mezer jedná o palindrom (čte se stejně zpředu jako zezadu, např. “kobyla ma maly bok”)
- Řešení – funkce s parametrem typu *String* a výsledkem typu *boolean*:

```
static boolean jePalindrom(String str) {
   int i = 0, j = str.length()-1;
   while (i<=j) {
     while (str.charAt(i)==' ') i++;
     while (str.charAt(j)==' ') j--;
     if (str.charAt(i)!=str.charAt(j))
       return false;
     i++; j--;
   }
   return true;
 }
```

Příklad - palindrom

- Výsledný program:

```
public class Palindrom {
  public static void main(String[] args) {
    Sys.pln("Zadejte jeden řádek");
    String radek = Sys.readLine();
    String vysl;
    if (jePalindrom(radek))
      vysl = "je";
    else
      vysl = "není";
    Sys.pln("Na řádku "+vysl+" palindrom");
  }

  static boolean jePalindrom(String str) {
    ...
  }
}
```

Pole znaků a řetězec

- Příklad: funkce pro převod celého čísla na řetězec tvořený zápisem čísla v hexadecimální soustavě

```
final static String hexa = "0123456789abcdef";

static String hexadecimal(int x) {
  if (x==0) return "0";
  char[] znaky = new char[9];
  int y;
  if (x<0) y=-x; else y=x;
  int prvni = 9;
  do {
    prvni--;
    znaky[prvni] = hexa.charAt(y%16);
    y = y / 16;
  } while (y>0);
  if (x<0) {
    prvni--; znaky[prvni] = '-';
  }
  return new String(znaky, prvni, 9-prvni);
}
```

Kontejnerové třídy

- Kontejner je datová struktura obsahující proměnný počet prvků, pro kterou jsou (mimo jiné) definovány operace
  - vytvoření prázdného kontejneru
  - vložení prvku
  - přístup k prvku
- V knihovnách Javy je řada tříd definujících vlastnosti různých typů kontejnerů
- Příkladem je třída *ArrayList*, ve které se vložené prvky rozlišují pomocí indexu
  - vytvoření prázdného kontejneru:

```
ArrayList kont = new ArrayList();
```

  - vložení prvků (první dostane index 0, druhý 1, atd.)

```
kont.add("abcd");
kont.add("xyz");
```

  - získání hodnoty prvku

```
Sys.pln(kont.get(0));    // vypíše se abcd
Sys.pln(kont.get(1));    // vypíše se xyz
```

  - získání počtu prvků

```
Sys.pln(kont.size());    // vypíše se 2
```

Příklad použití třídy ArrayList

- Program, který přečte řádky zakončené prázdným řádkem a vypíše je v opačném pořadí

```
import sugar.Sys;
import java.util.ArrayList;

public class KontejnerRadku {
  public static void main(String[] args) {
    ArrayList radky = new ArrayList();
    Sys.pln("zadejte řádky zakončené prázdným řádkem");
    String radek = Sys.readLine();
    while (!radek.equals("")) {
      radky.add(radek);
      radek = Sys.readLine();
    }
    Sys.pln("výpis řádků v opačném pořadí");
    for (int i=radky.size()-1; i>=0; i--)
      Sys.pln(radky.get(i));
  }
}
```

Příklad použití třídy ArrayList

- Chceme-li z kontejneru *radky* vyzvednout např. referenci na první řetězec a uložit jí do referenční proměnné

  ```
  String prvni;
  ```

  nelze použít příkaz

  ```
  prvni = radky.get(0);
  ```

  který způsobí chybu při překladu, ale je třeba použít přetypování:

  ```
  prvni = (String)radky.get(0);
  ```

- Zjednodušené vysvětlení:
  - metody *add* a *get* jsou specifikovány takto:

    ```
    void add(Object obj)
    Object get()
    ```

  - hodnotou proměnné, parametru nebo výsledkem metody typu *Object* může být reference na objekt jakéhokoliv typu
  - jestliže *o* je proměnná, parametr nebo výsledek metody typu *Object* a její hodnotou je reference na objekt typu *T*, pak pro přístup k referencovanému objektu jako k objektu typu *T* je třeba použít operaci přetypování *o* na typ *T* ve tvaru (*T*)*o*
  - operace zkontroluje, zda *o* skutečně referencuje objekt typu *T*; není-li tomu tak, nastane chyba při výpočtu

Číslo jako objekt

- Do kontejnerů vytvořených pomocí knihovních tříd lze vkládat pouze reference na objekty, nelze do nich vložit přímo hodnotu jednoduchého typu
- Chceme-li do kontejneru ukládat např. čísla typu *int*, musíme je zabalit (wrap) do objektů předdefinované třídy *Integer*
- Příklad:

```
ArrayList cisla = new ArrayList();
cisla.add(new Integer(10));
cisla.add(new Integer(20));
Sys.pln(cisla.get(0));   // vypíše se 10
Sys.pln(cisla.get(1));   // vypíše se 20
Integer prvni = (Integer)cisla.get(0);
```


- Číslo, které je v objektu typu *Integer* uloženo, získáme metodou *intValue*:

```
int n = prvni.intValue();
```

- Podobné obalovací třídy (wrapper classes) jsou v jazyku Java předdefinovany pro ostatní jednoduché typy

Hierarchie tříd

- V on-line dokumentaci jazyka Java týkající se třídy *ArrayList* najdeme následující obrázek:

```
java.lang.Object
    |
    +--java.util.AbstractCollection
        |
        +--java.util.AbstractList
            |
            +--java.util.ArrayList
```

- Tento obrázek vyjadřuje, že
  - třída *ArrayList* (definovaná v balíku *java.util*) je podtřídou třídy *AbstractList*
  - třída *AbstractList* je podtřídou třídy *AbstractCollection*
  - třída *AbstractCollection* je podtřídou třídy *Object*

Hierarchie tříd

- Třída *Tpod,* která je podtřídou třídy *Tnad,* dědí vlastnosti nadtřídy *Tnad* a rozšiřuje je o nové vlastnosti; některé zděděné vlastnosti mohou být v podtřídě modifikovány
- Pro instanční metody to znamená:
  - každá metoda třídy *Tnad* je i metodou třídy *Tpod,* v podtřídě však může mít jinou implementaci
  - v podtřídě mohou být definovány nové metody
- Pro strukturu objektu to znamená:
  - instance třídy *Tpod* mají všechny položky třídy *Tnad* a případně další
- Pro referenční proměnné to znamená:
  - proměnné typu *Tnad* může být přiřazena reference na objekt typu *Tpod*
  - na objekt referencovaný proměnnou typu *Tnad* lze vyvolat pouze metodu deklarovanou ve třídě *Tnad*; jde-li o objekt typu *Tpod,* metoda se provede tak, jak je dáno třídou *Tpod*
  - hodnotu referenční proměnné typu *Tnad* lze přiřadit referenční proměnné typu *Tpod* pouze s použitím přetypování, které zkontroluje, zda referencovaný objekt je typu *Tpod*
- Vztah *nadtřída – podtřída* je tranzitivní