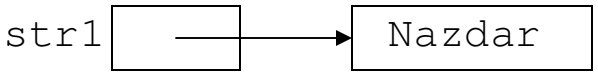
9. přednáška - třídy, objekty

- **třída String a její použití**
- **kolekce, typované kolekce**

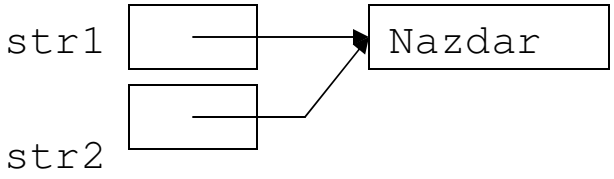
Třída String

- Objekty knihovní třídy ***String*** jsou řetězy znaků
- Od ostatních tříd se liší třemi specialitami:
  - objekt typu *String* lze vytvořit literálem
    (posloupnost znaků uzavřená mezi uvozovky)
  - hodnotu objektu typu *String* nelze jakkoli změnit
  - operací konkatenace čili zřetězení je nejen realizována metodou concat, ale i přetíženým operátorem +
- Příklady referečních proměnných typu *String:*

```
String str1 = "Nazdar";
```




```
String str2 = str1;
```



Operace s řetězy

- **Spojení řetězů (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 řetězů**
- **Příklady:**

```
"abc" + 5          výsledek je "abc5"
"a" + 1 + 2        výsledek je "a12"
"a" + 1 + (-2)     výsledek je "a1-2"
```

- **Porovnávání řetězů**
  - **relační operátory == a != porovnávají reference, nikoliv obsah řetězů**
  - **pro porovnání řetězů na rovnost slouží metoda *equals***

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

Operace s řetězy

- **Pro lexikografické porovnání řetězů slouží metoda *compareTo*:**

```
String s = "abcd";
System.out.println(s1.compareTo( "abdc" ));// vypíše -1
System.out.println(s1.compareTo( "abcd" ));// vypíše 0
System.out.println("abdc".compareTo(s1));       // vypíše 1
```

- **Některé další operace:**

```
String s = "nazdar";
int delka = s.length();              // délka je 6
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
int z3 = s.lastIndexOf('A');// z3 je -1
```

- **Hodnotu referenční proměnné typu *String* lze změnit (odkazuje pak na jiný řetěz), vlastní řetěz 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) {
      Scanner sc = new Scanner(System.in);
      System.out.println("Zadejte jeden řádek");
      String radek = sc.nextLine();
      String vysl;
      if (jePalindrom(radek)) vysl = "je" ;
      else vysl = "není" ;
      System.out.println("Na řádku " +vysl+ " palindrom");
   }

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

Pole znaků a řetěz

- Příklad: funkce pro převod celého čísla na řetěz 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);
}
```

Kolekce

- Kolekce je datová struktura obsahující proměnný počet prvků, pro kterou jsou (mimo jiné) definovány operace
  - vytvoření prázdné kolekce,
  - přidání, odebrání, vložení, přístup k prvku,
  - zjištění rozsahu, výpis a další.
- V knihovně java.util je řada tříd definujících různé druhy kolekcí.

  Příkladem je třída *ArrayList*, ve které se vložené prvky rozlišují pomocí indexu počítaných od 0.

```
ArrayList ko = new ArrayList(); // nová prázdná kolekce
ko.add( "abcd" );               // přidání na konec
ko.add( "xyz" );                // přidání na konec
System.out.println( ko.get(0) );// vypíše se abcd
System.out.println( ko.get(1) );// vypíše se xyz
System.out.println( ko.size() );//aktuální počet prvků: 2
```

Příklad použití třídy ArrayList

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

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

Příklad použití třídy ArrayList

- Chceme-li z kolekce *radky* získat např. referenci na první řetěz a uložit jí do referenční proměnné

  ```
  String prvni;
  ```

  nelze však 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

Typované kolekce v Javě 5.0

- Řešením jsou typované kolekce:

```
ArrayList<String> radky = new ArrayList();

... naplnění

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

Primitivní typy jako objekty

- Do kolekcí lze vkládat pouze reference na objekty, nikoli primitivní typ.
- Primitivní typ např. čísla typu *int*, musíme je nejprve zabalit (wrap) do objektů typu java.lang.*Integer.*
- Příklad:

```
ArrayList cisla = new ArrayList();
cisla.add( new Integer(10));
cisla.add( new Integer(20));
System.out.println(cisla.get(0));   // vypíše se 10
System.out.println(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 Javě definovány pro všechny primitivní typy, kterých je pouze osm.

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 nejvyšší třídy java.lang.*Object*

pro zájemce

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 členy 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 však *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.

pro zájemce