Třídy jazyka Java

- Vybrané třídy jazyka Java
  - String, StringBuilder, StringBuffer
  - Math, Random
  - Arrays
  - ArrayList , LinkedList
  - System
  - Scanner
  - StringTokenizer

Třída `String` – konstantní textové řetězce

- Řetězec v jazyce Java je objekt – instance třídy `String` – v balíku `java.lang`
- S řetězcem pracujeme jako s objektem a nikoli jako s polem hodnot typu `char`
- Typ `String`
  - Instance typu `String` – **konstantní textové řetězce** – jednou přiřazený textový řetězec nelze měnit
  - Pro každý textový řetězec zapsaný v uvozovkách zajistí kompilátor vytvoření instance typu `String`
  - První znak textového řetězce má index 0
  - Potřebujeme-li řetězec měnit je možné použít typy `StringBuilder, StringBuffer`

- Vytvoření textového řetězce (instance třídy `String`)
  - Přiřazením konstantního textu do proměnné typu `String`
  - Použití některého z konstruktorů třídy `String`
  - Řada metod vrací hodnotu typu `String` (tj. referenci na instanci typu `String`)
  - Inicializované pole textových řetězců

```
String s1, s2, s3, s4, s5;
char[] zn = {'C', 'a', 'u'};
s1 = "Ahoj";
s2 = new String("Ahoj");              s4 = sc.nextLine();
s3 = new String(zn);                  s5 = sc.next();
String[] s = {"prvni", "druhy"};
```

Manipulace s textovými řetězci – `String`

- Délka textového řetězce
  - `int length()`
- Získání části/částí z textového řetězce – instanční metody
  - `char charAt(int index)`
  - `String substring(int pocatecniIndex, int koncovyIndex)`
  - `String substring(int pocatecniIndex)`
  - `String[] split(String regVyraz)`
  - `String[] split(String revVyraz, int limit)`
  - `CharSequence subSequence(int pocatecniIndex, int koncovyIndex)`
  - `String trim()`
  - `void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)`
- Získání části/částí z textového řetězce – instanční metody
  - `String replace(char znak1, char znak2)`
  - `...`
- Výše uvedený typ `CharSequence` je rozhraní definované v balíku `java.lang`
- Toto rozhraní implementují třídy `String`, `StringBuffer`, `StringBuilder`, `CharBuffer`, `Segment`

Manipulace s textovými řetězci – `String`

- Vyhledávaní a porovnávání textových řetězců – instanční metody
  - `int indexOf(int znak)`
  - `int lastIndexOf(int znak)`
  - `int indexOf(int znak, int odIndexu)`
  - `int lastIndexOf(int znak, int odIndexu)`
  - `int indexOf(String str)`
  - `int lastIndexOf(String str)`
  - `int indexOf(String str, int odIndexu)`
  - `int lastIndexOf(String str, int odIndexu)`
  - `boolean contains(CharSequence s)`
  - `boolean endsWith(String sufix)`
  - `boolean startsWith(String sufix)`
  - `boolean startsWith(String sufix, int posun)`
  - `int compareTo(String dalsiRetezec)`
  - `int compareToIgnoreCase(String jinyRetezec)`
  - `boolean equals(object objekt)`
  - `boolean equalsIgnoreCase(String jinyRetezec)`
  - `boolean regionMatches(int posun, String jiny, int jinyPosun, int delka)`
  - `boolean regionMatches(boolean ignoreCase, int posun, String jiny, int jinyPosun, int delka)`
  - `boolean matches(String regVyraz)`

Použití regulárních výrazů v metodách `String`

- Některé metody třídy `String` umožňují používat regulární výrazy
  - `String[] split(String regVyraz)`
  - `String[] split(String revVyraz, int limit)`
  - `boolean matches(String regVyraz)`
  - `String replaceFirst(String regVyraz, String nahrada)`
  - `String replaceAll(String regVyraz, String nahrada)`

- Konstrukce regulárních výrazů – viz dokumentace jazyka Java, literatura
  - Metaznaky a jejich význam
  - Doslovné hodnoty, třídy znaků (výčet, rozsah, sjednocení, průnik, negace, rozdíl), předdefinované třídy znaků, opakování sekvencí, …

```
String text = "aaa      bbb ccc     ddd      eeef";
String[] slova;
slova = text.split(" ");
slova = text.split(" +");
slova = text.split(" {1,}");
```

Porovnání dvou textových řetězců

`equals()`
`equalsIgnoreCase()`

- Metody vrací hodnotu `true` nebo `false`

`compareTo()`
`compareToIgnoreCase()`

- Metody vrací hodnotu
  - zápornou
  - 0
  - kladnou

- Aplikace relačních operátorů `==`, `!=` na hodnoty typu `String`
  - Porovnání referencí a nikoli textových řetězců

```
String s1, s2;

s1 = ...;
s2 = ...;

if (s1.equals(s2))
  System.out.println("Shodne");
else
 System.out.println("Ruzne");

if (s1.compareTo(s2) == 0)
  System.out.println("Shodne");
else
 System.out.println("Ruzne");

if (s1 == s2)
  System.out.println("Totozny objekt");
else
 System.out.println("Ruzne objekty");
```

Třídy `String`, `StringBuffer`, `StringBuilder`

- `String` – konstantní textový řetězec
- `StringBuffer`, `StringBuilder` – proměnné textové řetězce, změna jednotlivých znaků textového řetězce, změna délky
  - `lenght(), capacity(), setLength(), ensureCapacity()`
  - `append(), insert(), replace(), delete()`
  - `charAt(), setCharAt(), deleteCharAt()`
  - `reverse()`
  - `indexOf(), lastIndexOf()`
  - `getChars(), substring(), subSequence()`
  - `toString()`
  - Obě třídy – téměř totožné – třída `StringBuffer` je bezpečná při používání podprocesů (threadů) díky synchronizaci svých metod
- Vytvoření instance – metody konstruktoru
  - `StringBuilder s1, s2, s3`
  - `s1 = new StringBuilder();`
  - `s2 = new StringBuilder(30);`
  - `s3 = new StringBuilder("Ahoj");`
- Konverze mezi `String` a `StringBuffer`
  - `toString()`
  - Konstruktor třídy `StringBuffer`

Práce s jednotlivými znaky – třída `Character`

- Test znaku
  - `isDigit()`
  - `isLetter()`
  - `isLetterOrDigit()`
  - `isLowerCase()`
  - `isUpperCase()`
  - `isWhiteSpace()`

- Změna velikosti písma
  - `toLowerCase()`
  - `toUppercase()`

Třída `Math`

- Knihovní třída – deklarace s modifikátorem `final`, privátní konstruktor
- Matematické funkce, konstanty PI, E

- Třída v balíku `java.lang`

- Třída obsahuje statické konstanty a statické metody
- Základní seznámení s třídou – v předmětu ALP1

Reprezentace časových, datumových hodnot

- `System.currentTimeMillis()` – počet milisekund od 1.1.1970
- `System.nanoTime()` – počet nanosekund

- Třída `Date` – uložení data a času s přesností na milisekundy
  - Řada metod dále nepodporovaných (nebo nedoporučených k užití)
  - `new Date()` – počet milisekund od 1.1.1970
  - `new Date(ms)`
  - `toString(), getTime(), before(), after()`

- Třída `Calendar` – třída reprezentující kalendář, abstraktní třída, odvozená třída `GregorianCalendar`
  - `Calendar dnes = Calendar.getInstance();`
  - Metoda vrátí instanci třídy `GregorianCalendar`

Třída Random

- Třída s prostředky pro generování náhodných čísel
- Před vlastním použitím nástrojů – vytvoření instance
  - Třída v balíku `java.util`

- `Random rnd  = new Random()`

- Metody
  - `nextBoolean()`
  - `nextBytes(byte[] poleByte)`
  - `nextDouble()`
  - `nextFloat()`
  - `nextInt()`
  - `nextInt(int rozsah)`
  - `nextLong()`
  - `setSeed(long seed)`

  - Popis činnosti a parametrů metod – viz dokumentace jazyka Java

Obalové třídy

- Obalové třídy primitivních datových typů
- Třídy v balíku `java.lang`
  - `Byte, Short, Integer, Long, Boolean, Character, Float, Double`

- Statické konstanty – rozsahy
- Statické metody – převod hodnot primitivních datových typů na textový řetězec, „parsování“ hodnot primitivního datového typu z textového řetězce – tyto metody nejsou lokalizovatelné

- Vytvoření instance – převedení hodnot primitivního datového typu na hodnotu objektového typu

```
Double dd1 = Double.valueOf("13.186");
Double dd2 = Double.valueOf(-15.465);
Double dd3 = new Double(3.15E-5);
Double dd4 = -4.186E2;
```

  - Proměnné `dd1, dd2, dd3, dd4` jsou objektového typu

Třída `Arrays`

- Knihovní třída – privátní konstruktor
- Pole v jazyce Java
  - Jediná členská proměnná `length`
  - Pole nemají metody

- Základní nástroje pro práci s poli – třída `Arrays` – knihovní třída
- Třída v balíku `java.util`

- Metody – statické
  - `Arrays.sort()`
  - `Arrays.binarySearch()`
  - `Arrays.copyOfRange()`
  - `Arrays.equals()`
  - `Arrays.deepEquals()`
  - `Arrays.fill()`
  - `Arrays.toString()`
  - `Arrays.deepToString()`

  - Popis činnosti parametrů metod – viz dokumentace jazyka Java, minimálně jeden parametr typu pole

Třídy `ArrayList`, `LinkedList`

- Pole v jazyce Java – nelze dynamicky měnit velikost, v případě nutnosti lze realokovat – realokace časově náročná – předpokládáme-li opakovanou realokaci, prodlužujeme pole o více než jednu položku
- V případě, že potřebujeme datovou strukturu, do které budeme průběžně přidávat další a další prvky, je lépe místo pole využít některou z kontejnerových třída jazyka Java
- Uvedeme dvě z těchto třídy
  - `ArrayList` – dynamické pole
  - `LinkedList` – zřetězený seznam
- Uvedené pole/seznamy mohou uchovávat pouze hodnoty objektového typu
  - Pokud potřebujeme v dynamickém poli nebo seznamu uchovávat hodnoty primitivních typů je třeba je pomocí dříve uvedených prostředků převést na instance některé z obalových tříd
- Některé prostředky obou tříd jsou shodné, jiná vnitřní implementace obou tříd

Třída `ArrayList`

- Deklarace proměnné – specifikace typu hodnot, které budeme v dynamickém poli uchovávat
- Třída v balíku `java.util`
  - `ArrayList<Bod> body;`

- Vytvoření instance
  - `new ArrayList()`
  - `new ArrayList(pocatecniKapacita)`

- Některé metody
  - `add()`
  - `clear()`
  - `get()`
  - `set()`
  - `contains()`
  - `indexOf()`
  - `lastIndexOf()`
  - `isEmpty()`
  - `ensureCapacity()`
  - `size()`
  - `toArray()`

  - Další metody, popis metod a jejich parametrů – viz dokumentace jazyka Java

Ukázka použití třídy `ArrayList`

- Deklarace proměnné – specifikace typu hodnot, které budeme v dynamickém poli uchovávat

```
ArrayList<Bod> body = new ArrayList<Bod>();
body.add(new Bod(3, 4));
body.add(new Bod(7, 16.125));
…
for (Bod b : body) {
    System.out.println(b.vzd());
}
```

- Alternativně – procházení pole/seznamu pomocí iterátoru

Třída `LinkedList`

- Deklarace proměnné – specifikace typu hodnot, které budeme v dynamickém poli uchovávat
- Třída v balíku `java.util`
  - `LinkedList<Bod> body;`

- Vytvoření instance
  - `new LinkedList()`

- Některé z řady metod
  - `add(), addFirst(), addLast()`
  - `element()`
  - `get(), getFirst(), getLast()`
  - `peek(), peekFirst(), peekLast()`
  - `pool(), poolFirst(), poolLast()`
  - `remove(), removeFirst(), removeLast()`
  - `pop(), push()`
  - `set()`
  - `contains(), indexOf(), lastIndexOf()`
  - `clear(), isEmpty(), size()`
  - `toArray()`

  - Úplný seznam metod, popis metod a jejich parametrů – viz dokumentace jazyka Java

Třída `System`

- Knihovní třída – deklarace s modifikátorem `final`, privátní konstruktor
  - Třída v balíku `java.lang`
- Statické atributy
  - `System.in`
  - `System.out`
  - `System.err`
- Některé statické metody
  - `System.arraycopy()`
  - `System.currentTimeMillis()`
  - `System.nanoTime()`
  - `System.exit()`
  - `System.gc()`
  - `System.getProperty()`
  - `System.getProperties()`
  - `System.console()`
  - `System.loadLibrary()`
  - `System.getenv()`
  - `...`
  - Popis činnosti a parametrů metod, další prostředky – viz dokumentace jazyka Java

Třída `Runtime`

- Každá aplikace v jazyce Java má k dispozici instanci třídy `Runtime`
- Instance třídy `Runtime` poskytuje prostředky prostředí, ve kterém je aplikace spuštěna
- Instanci třídy `Runtime` získáme voláním `Runtime.getRuntime()`

- Vybrané metody
  - `exec()`
  - `exit()`
  - `halt()`
  - `gc()`
  - `load()`
  - `loadLibrary()`
  - `freeMemory()`
  - `maxMemory()`
  - `totalMemory()`
  - `availableProcessors()`

  - ...

  - Popis činnosti a parametrů metod, další prostředky – viz dokumentace jazyka Java

Třída Scanner

- Separace jednotlivých hodnot (tokenů) slov z textového vstupu a jejich případný převod na hodnoty primitivních datových typů
  - Třída v balíku `java.util`
- Zdrojem dat / vstupem může být
  - Externí soubor, souborové zařízení
  - Textový řetězec typu `String`
  - Bytový kanál
  - Jiný zdroje
- Metody `hasNext(), hasNextInt(), hasNextFloat(), …`
- Metody `next(), nextInt(), nextFloat(), …, nextLine, …`
- Při zpracování externího souboru nebo bytového kanálu lze specifikovat používanou znakovou sadu
- Nástroje třídy `Scanner` jsou lokalizovatené – `useLocale()`, některé aspekty a nástroje lokalizace budou v rámci přednášek z ALP2 zmíněny později
- Oddělovače jednotlivých tokenů/symbolů implicitně dle `Character.isWhitespace()`, nastavení `useDelimiter()`

Třída `StringTokenizer`

- Separace jednotlivých hodnot (tokenů) slov z textového řetězce
  - Třída v balíku `java.util`
- Některé konstruktory
  - `StringDelimiter(text)`
  - `StringDelimiter(text, oddelovace)`
- Některé metody
  - `hasMoreTokens()`
  - `nextToken()`

```
StringTokenizer st = new StringTokenizer("libovolny text pro zpracovani");
while (st.hasMoreTokens()) {
    System.out.println(st.nextToken());
}
```

Separace symbolů/tokenů z textu

- Některé varianty separace slov z delšího textu, slova oddělená právě jednou mezerou

```
String text = "libovolny text ze ktereho budou separovana slova";
String dalsiSlovo;

Scanner sc = new Scanner(text);
while(sc.hasNext()) {
    dalsiSlovo = sc.next();
    System.out.println(dalsiSlovo);       // nebo jinak zpracuj dalsiSlovo
}

StringTokenizer st = new StringTokenizer(text);
while (st.hasMoreTokens()) {
    dalsiSlovo = st.nextToken();
    System.out.println(dalsiSlovo);       // nebo jinak zpracuj dalsiSlovo
}

String[] slova = text.split("\\s");
for (int i = 0; i < slova.length; i++){
    System.out.println(slova[i]);         // nebo jinak zpracuj slova[i]
}
```

- Uvedené kódy nejsou zcela ekvivalentní, nicméně pro výše uvedený textový řetězec poskytnou stejný výstup, každý prostředek má svoje specifika