Znaky a řetězce v Javě

Řetězec v Javě je samostatný objekt (instance třídy String, přesněji java.lang.String). To znamená, že s řetězcem se pracuje jako s celým objektem, nikoliv jako s polem prvků typu char.

Řetězce v Javě jsou konstantní, což znamená, že jednou vytvořený řetězec nelze změnit. Z tohoto důvodu je možné použít String ve switchi. Všechny řetězce se ukládají do tzv. **string constant pool**. Při vytváření nových řetězců se nejprve zkontroluje zda takový string existuje v **string constant poolu**, pokud ano použije se tato hodnota, pokud ne přidá se do něj. Tímto způsobem šetří Java místem.

Způsoby vytváření řetězců

Nejjednodušším způsobem je přiřazením textu v uvozovkách do příslušné proměnné:

String s = “ahoj”;

Nebo lze použít několik dostupných konstruktorů:

String s1, s2, s3, s4, s5, s6, s7;

byte[] bajty = {(byte)’E’, (byte)’V’, (byte)’A’};

char[] znaky = {‘P’, ‘e’, ‘t’, ‘r’, ‘a’};

StringBuffer buf = new StringBuffer(“dobry den”);

s1 = new String(“cus”);

s1 = new String(s1);

s3 = new String(bajty);

s4 = new String(bajty, 1, 2); // Eva -> va

s5 = new String(znaky);

s6 = new String(znaky, 2, 3); // Petra -> tra

s7 = new String(buf);

🛈 Při vytváření řetězce z pole bajtů se bajty považují za znaky v osmibitovém kódování a to, jak se převedou na šestnáctibitové znaky, záleží na přednastaveném kódování znaků

* Délku řetězce lze použít metodu length() , která vrací počet znaků. Jiným způsobem délka zjistit nelze

Práce s řetězci v Javě

Porovnavani

**str1.compareTo(str2)**

**str1.compareToIgnoreCase(str2)**

// Obě metody vrací int > 0 pokud je parametr metody mensi, 0 v pripade shody a int < 0 pokud je parametr metody vetsi

**str1.equals(str2)** // zjisti zda jsou retezce shodne

**str1.equalsIgnoreCase(str2)** // bez velkych a malych

🛈 U Stringu **nelze použít operátor ==** , jelikož tento operátor slouží pouze pro primitivní datové typy nebo pro porovnání stejných referencí. Pokud máme dvě různé instance Stringu, které mají stejnou hodnotu (“ahoj” a “ahoj”), operátor == nám vrátí false, jelikož se nejedná o stejné objekty. Musíme použít **.equals()**

Převody na malá a velká

**s1.toLowerCase()** // prevede vsechna pismena na mala

**s1.toUpperCase()** // prevede vsechna pismena na velka

Spojeni

**s1.concat(s2)** // pripoji s2 k s1 a vrati ho

Náhrada znaků v řetězci

**s1.replace(‘c’, ‘k’)** // nahradi vsechna pismena c za k

Získání části řetězce

**s1.substring(1, 9)** // od 1 do 9 vcetne

**s1.substring(5)** // od 5 do konce

**s1.charAt(0)** // znak na indexu 0

Hledání znaků

**s.indexOf(‘a’)**  // prvni vyskyt znaku ‘a’

**s.lastIndexOf(‘a’)** // posledni vyskyt znaku ‘a’

// pokud nenalezen vraci -1, pokud je nalezen je vracen index

Metoda toString()

Tuto metodu dědí každý objekt, jelikož je deklarována v kořenové třídě Object. Slouží nám k vypsání informace o daném objektu.

Pokud metodu nepřekryjeme, vrátí se nám String v následujícím formátu

**{Název Třídy}@{jednoznačný identifikátor objektu}**

StringBuffer a StringBuilder

Používají se v případě, že potřebujeme libovolně měnit znaky řetězce, případně ho zvětšovat nebo zmenšovat.

Oba jsou rychlejší než samotná práce se Stringy, jelikož skokově alokují velikost řetězce vždy dvojnásobek + 2 bajty a při přetečení znovu.

Hlavním rozdílem těchto dvou je, že StringBuilder není synchronizovaný tj. nepodporuje vícevláknové použití. Z tohoto důvodu je StringBuilder rychlejší než StringBuffer.

Několik metod například

reverse() // obrátí řetězec

ensureCapacity(int k) // zajistí požadovanou délku řetězce

setLength(int k) // nastaví délku, může ho i oříznout

append() // připojí k řetězci

delete(int od, int do) // smaže z řetězce

deleteCharAt(int index) // smaže znak na indexu

Konverze na string

StringBuffer buf = new StringBuffer(“ahoj”);

String s = buf.toString();

String s2 = buf.substring(0, 4);

Znaky

Pro práci se znaky se používá třída Character, neboli java.lang.Character.

Můžeme pomocí této třídy používat různé statické metody, např.:

Character.isDigit(char c);

Character.isLetter(char c);

Character.isLetterOrDigit(char c);

Character.isLowerCase(char c);

Character.isUpperCase(char c);

Character.isWhitespace(char c);

Character.toUpperCase(char c);

Character.toLowerCase(char c);