Stringovi

- Objektno-orjentisano programiranje 1

Stringovi

- String – niz karaktera
- Klasa **String** iz paketa **java.lang**
  - Klase iz paketa java.lang ne moramo eksplicitno importovati
  - Osnovne operacije u radu sa stringovima
  - Naprednije metode za procesiranje tekstualnih podataka
- Final klasa – ne može se nasleđivati
- Klasa String sakriva internu reprezentaciju samog stringa
  - Nije dozvoljena direktna manipulacija nizom karaktera
- **Objekti klase String su imutabilni**
  - Stanje objekta ne može da se promeni
  - Operacije koje “transformišu” string zapravo proizvode nove string objekte

Klasa String

- **“Privilegovana” klasa**
  - Objekte klase **String** možemo kreirati bez korišćenja operatora new
    - `String p = "Zdravo";`
    - `String q = new String("Zdravo");`
  - Stringovi se mogu konkatenirati koristeći operator + (međusobno, ali i sa objektima drugih klasa)
    - `String p = "Mika", q = "Zika";`
    - `String r = p + q;`
    - `String w = r + 123`
    - ```
      NekaKlasa t = new NekaKlasa();
      String x = "Objekat " + t;      // poziva se t.toString()
      ```
      ```

Stringovi

- Implementira interfejs **Comparable<String>**
  - Leksikografsko poređenje stringova koristeći **compareTo** metod
- Mnoštvo konstruktora
  - String()
  - String(char[] val)
  - String(byte[] val, Charset c)
  - String(String s)
  - String(StringBuillder sb)
  - …
- Metode možemo podeliti u dve kategorije
  - informativne – vraćaju neku informaciju o stringu (npr. dužina)
  - transformativne – kreiraju nove string objekte na osnovu postojećih

Informativne metode

- **char charAt(int pos)**
  - vraća karakter na poziciji **pos**
- **boolean equals(Object o)**
- **int compareTo(String o)**
  - **< 0 (leksikografski manji), == 0 (identični), > 0 (leksikofgrafski veći)**
- **int length()**
  - vraća dužinu stringa
- **boolean endswith(String suff)**
  - proverava da li se string završava stringom suff
- **boolean startswith(String pref)**
  - proverava da li string počinje stringom pref
- **boolean equalsIgnoreCase(String p)**
  - proverava da li je string identičan p ignorišući razliku između velikih i malih slova

Informativne metode

- **int indexOf(String s)**
  - Proverava da li se u stringu nalazi string **s**
  - Vraća -1 ako se ne nalazi, odnosno indeks prve pojave **s**
- **int indexOf(String s, int p)**
  - Proverava da li se u stringu nalazi string **s** počev od pozicije **p**

```
public static void svePojave(String pat, String txt) {
	int pos = txt.indexOf(pat);
	while (pos != -1) {
		System.out.println(pos);
		pos = txt.indexOf(pat, pos + 1);
	}
}
```

- **int lastIndexOf(String s)**
- **int lastIndexOf(String s, int p)**
  - poslednja pojava pre indeksa p

Transformativne metode

- **String trim()**
  - odstranjuje praznine sa početka i kraja stringa
- **String concat(String d)**
  - konkatenacija
- **String toLowerCase()**
  - sva velika slova u mala
- **String toUpperCase()**
  - sva mala slova u velika
- **String substring(int beg, int end)**
  - Formiranje podstringa od pozicije **beg** do pozicije **end – 1**
- **String substring(int beg)**
  - Formiranje podstringa od pozicije **beg** do kraja stringa
- **String replace(CharSequence target, CharSequence replacement)**
  - Zamenjuje sve pojave **target** sa **replacement**

Napredne transformativne metode

- Napredne transformativne metode se zasnivaju na **regularnim izrazima**
- **Regularni izraz je string koji opisuje skup stringova sa istom struktorom**
- Regularni izraz se sastoji od karaktera i operatora
  - **[abcd]** – izraz koji opisuje bilo koji od navedenih karaktera
  - **[^abcd]** – izraz koji opisuje bilo koji karakter osim navedenih
  - **[a-z]** – izraz koji opisuje jedan karakter iz datog opsega
  - **[a-zA-Z]** – unija dva opsega
  - **[a-m&&c-z]** – presek dva opsega
  - **\d** – bilo koja cifra
  - **\s** – belina (space, tab, new-line)
  - **\w** – isto što i [a-zA-Z0-9]

Napredne transformativne metode

- **Regularni izraz je string koji opisuje skup stringova sa istom struktorom**
- Regularni izraz se sastoji od karaktera i operatora
  - **X?** – tačno jedna ili nijedna pojava stringa koji zadovoljava X
  - **X*** – nula, jedna ili više pojava stringa koji zadovoljava X
  - **X+** – jedna ili više pojava stringa koji zadovoljava X
  - **X{n}** – tačno n projava stringa koji zadovoljava X
  - **XY** – string koji zadovoljava X konkateniran sa stringom koji zadovoljava Y
  - **X | Y** – string koji zadovoljava ili X ili Y
  - **(X)** – zagradama možemo regulisati prioritet

Napredne transformativne metode

- **boolean matches(String regex)**
  - Proverava da li string zadovoljava regex
- **String replaceAll(String regex, String replacement)**
  - Zamenjuje svaki podstring koji zadovoljava regex sa datom zamenom
- **String replaceFirst(String regex, String replacement)**
  - Zamenjuje samo prvi podstring koji zadovoljava regex datom zamenom
- **String[] split(String regex)**
  - Deli string u niz tokena naspram delimitera opisanog regularnim izrazom

Statičke metode klase String

- **String sadrži statičke metode za konverziju promenljivih primitivnih tipova u stringove**
  - `static String valueOf(boolean b)`
  - `static String valueOf(char c)`
  - `static String valueOf(int i)`
  - `static String valueOf(long l)`
  - `static String valueOf(double d)`
  - `static String valueOf(float f)`

Napredne transformativne metode

```
public static void main(String[] args) {
	String ulaz = "B434 ;. Truc ... Tika;Tak111#2Mak2@3Lak";

	// separator je sve sto nije niz slova i/ili cifara
	String sepRegex = "[^a-zA-Z0-9]+";

	String[] tok = ulaz.split(sepRegex);
	for (int i = 0; i < tok.length; i++) {
		String t = tok[i].trim();

		// izbacujemo sve brojeve
		t = t.replaceAll("[0-9]+", "");
		System.out.println(t);
	}
}
```

Izlaz:

B

Truc

Tika

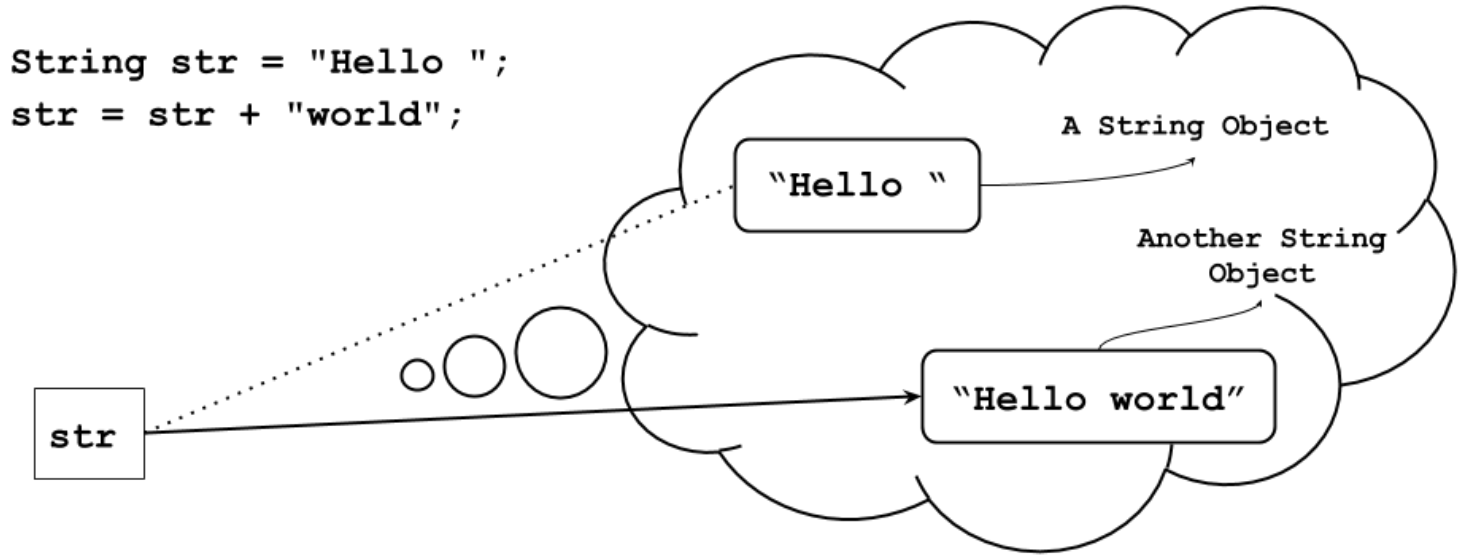
Tak

Mak

Lak

Nepromenljivost i performanse

- **Objekti klase string su nepromenljivi (imutabilni)**
- **Prilikom svake transformacije se kreira novi string objekat**




```
String rez = "";
for (int i = 0; i < k; i++)
        rez += s;
```

Nepromenljivost i performanse

- Klasa **StringBuilder** – mutabilni stringovi
  - klasa kojom pravimo stringove primenjujući stringovne operacije koje ne kreiraju nove objekte
- **Operacije insert i append koje ne postoje u klasi String**
  - insert – dodavanje stringa u string (dodavanje na proizvoljnu poziciju)
  - append – dodavanje stringa na kraj stringa
- **Konstruktori**
  - StringBuilder()
  - StringBuilder(int capacity)
  - StringBuilder(CharSequence c)
    - CharSequence – interfejs koji implementiraju klase koje realizuju stringove (String, StringBuilder, StringBuffer)
  - StringBuilder(String s)

StringBuilder

- **Klasa StringBuilder implementira interfejs CharSequence**
  - `char charAt(int index)`
  - `int length()`
  - `CharSequence subSequence(int start, int end)`
- **Append i insert metode**
  - append za primitivne tipove, npr. `StringBuilder append(int i)`
  - `StringBuilder append(Object o)`
  - `StringBuilder append(String s)`
  - `StringBuilder append(CharSequence c)`
  - insert za primitivne tipove, npr. `StringBuilder insert(int pos, int i)`
  - `StringBuilder insert(int pos, Object o)`
  - `StringBuilder insert(int pos, String s)`
  - `StringBuilder insert(int pos, CharSequence c)`

StringBuilder

- **Ostale metode klase StringBuilder**
  - `StringBuilder delete(int start, int end)`
    - briše podstring od pozicije start do pozicije end – 1
  - `StringBuilder deleteCharAt(int index)`
  - `StringBuilder replace(int start, int end, String repl)`
  - `int indexOf(String s)`
  - `int indexOf(String s, int pos)`
  - `StringBuilder reverse()`
  - `String substring(int start)`
  - `String substring(int start, int end)`

StringBuilder

```
public class StringBuildeTest {

	public static String kputa(String s, int k) {
		String r = s;
		for (int i = 0; i < k - 1; i++)
			r += s;

		return r;
	}

	public static String kputaSB(String s, int k) {
		StringBuilder sb = new StringBuilder(s);
		for (int i = 0; i < k - 1; i++)
			sb.append(s);

		return sb.toString();
	}
	...
```

```java
public static void main(String[] args) {
	int k = 100000;

	long start = System.nanoTime();
	kputa("Ana voli Milovana", k);
	long end = System.nanoTime();
	double t1 = (end - start) / 1000000000.0;
	long m1 = Runtime.getRuntime().totalMemory();
	System.out.println("Vreme (sec): " + t1);
	System.out.println("Memorija (bajtovi): " + m1);
	System.gc(); System.gc(); System.gc();
	m1 = Runtime.getRuntime().totalMemory();
	System.out.println("Memorija (bajtovi) posle GC: " + m1);

	start = System.nanoTime();
	kputaSB("Ana voli Milovana", k);
	end = System.nanoTime();
	double t2 = (end - start) / 1000000000.0;
	long m2 = Runtime.getRuntime().totalMemory();
	System.out.println("Vreme (sec): " + t2);
	System.out.println("Memorija (bajtovi): " + m2);
	System.gc(); System.gc(); System.gc();
	m2 = Runtime.getRuntime().totalMemory();
	System.out.println("Memorija (bajtovi) posle GC: " + m2);
}
```

Performanse

- **kputa**
  - Vreme (sec): 81.3822327
  - Memorija (bajtovi): 73400320 (~70mb)
  - Memorija (bajtovi) posle GC: 8388608 (~8mb)

- **kputaSB**
  - Vreme (sec): 0.0060213
  - Memorija (bajtovi): 17825792 (~17mb)
  - Memorija (bajtovi) posle GC: 8388608 (~8mb)