Matematički fakultet, Univerzizet u Beogradu
Katedra za računarstvo i informatiku

Objektno orijentisano programiranje

vežbe
školska 2016/2017

Biljana Stojanović
Nemanja Mićović
Nikola Milev

Čas 3: Stringovi u Javi: nepromenljivi (klasa java.lang.String) i promenljivi (klase java.lang.StringBuilder i java.lang.StringBuffer).

1 Stringovi

String literali i objekti klase String

String literali su sekvence karaktera između dvostrukih navodnika:

```
"Ovo je String literal"
```

Ukoliko želimo da napravimo objekat klase String, najjednostavnije je da koristimo String literal:

```
String str = "Objektno orijentisano programiranje";
```

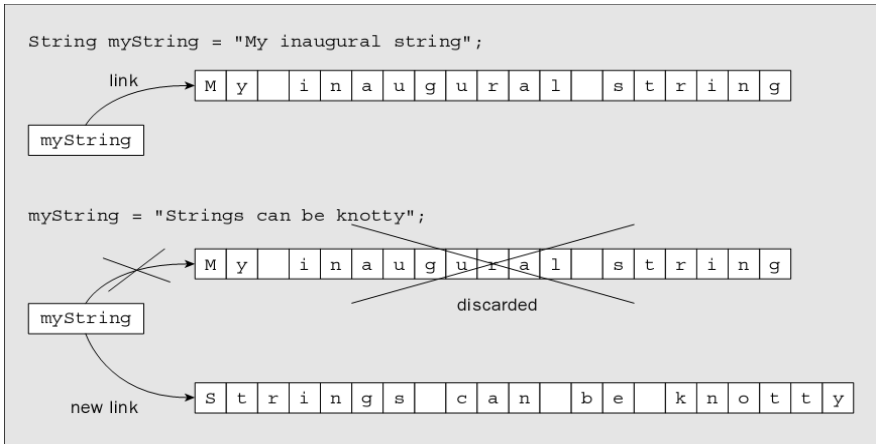
Gornjom naredbom se deklariše promenljiva *str* tipa *String* i inicijalizuje se tako da sadrži referencu na objekat klase *String* čiji je sadržaj niska *"Objektno orijentisano programiranje":*

Drugi način je pomoću operatora **new** za kojim sledi poziv konstruktora klase String kojem se kao argument prosledi String literal:

```
String str = new String("Ovo je moj string");
```

Iako deluje da su ova dva načina identična, ipak postoji izvesna razlika u ponašanju JVM u jednom i drugom slučaju.

Objekat klase *String* razlikuje se od promenljive kojom se referiše. U gornjem primeru promenljiva *str* sadrži referencu na objekat, ne i sam objekat.
Referenca je vrednost na osnovu koje Java VM može da pristupi objektu.
U istu promenljivu tipa String može se sačuvati i referenca na neki drugi objekat klase String.
Objekti klase *String* imaju svojstvo da ne mogu biti promenjeni *(immutable).* To znači da se ne može promeniti string (niska karaktera) kojim je predstavljen neki objekat klase *String*.



Kada se izvršava operacija nad postojećim objektom klase *String* kao rezultat se uvek dobija novi objekat klase *String*.
Promenom stringa na koji referiše *String* promenljiva, umesto reference na stari *String* objekat, sadržaj promenljive postaje referenca na novi *String* objekat.
Može se koristiti literal *null* kada želimo da odbacimo objekat na koji trenutno referiše promenljiva tako što joj dodelimo vrednost *null*. Promenljiva onda neće ukazivati ni na šta.

```
String s = null; /* String promenljiva koja ne referise ni na jedan string */
```

Dužina stringa

Dužina stringa određuje se pozivom ***metoda*** length() nad stringom (za razliku od nizova gde je length bio atribut).

Nadovezivanje (spajanje, konkatenacija) stringova

Konkatenacija stringova vrši se operatorom **+**.

```
String str = "OO programiranje" + " u programskom jeziku Java"
```

Kao rezultat nadovezivanje pravi se novi objekat klase *String* koji sadrži string "*OO programiranje u programskom jeziku Java*" i koji je nezavisan od dva polazna. Referenca na novi objekat upisuje se u promenljivu *str.*

```
String dan = "15. ";
String mesec = "maj";
String datum = dan + mesec; // rezultat je "15. maj"
```

Pravi se novi objekat klase *String* koji je nezavisan od dva polazna (na koje referišu promenljive *dan* i *mesec).* Promenljiva *datum* sadrži njegovu referencu.

```
String dan = "15. ";
String mesec = null; // promenljiva mesec ne referiše ni na šta
String datum = dan + mesec;
```

Sadržaj promenljive *mesec* se konvertuje u string "*null"* i to se nadovezuje na string *"15. "*. Rezultat je *"15. null"*.

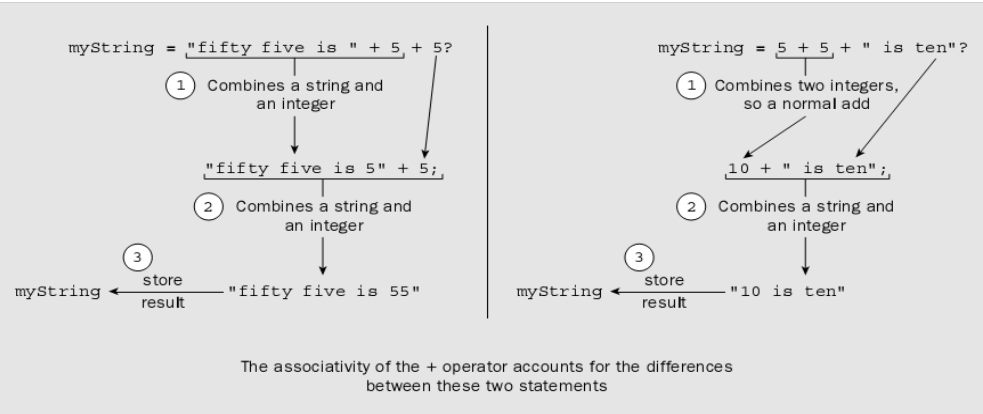
Može se koristi i operator **+=** za konkatenaciju stringova:

```
String str = "OO programiranje";
str += " u programskom jeziku Java";
```

Promenljiva *str* sada referiše na novi String objekat, što ne menja string "*OO programiranje*".

Operator + je ***levo asocijativan***.

```
String myString;
myString = "fifty five is " + 5 + 5; // "fifty five is 55"
myString = 5 + 5 + " is ten"; // "10 is ten"
```



Implicitna konverzija primitivnih tipova u String je omogućena zahvaljujući statičkom metodu *toString()* standardnih omotač-klasa koje se odnose na primitivne tipove (*Byte, Short, Integer, Long, Float, Double, Boolean, Character*).
Kad god je jedan operand operatora **+** vrednost primitivnog tipa, a drugi objekat klase String, kompajler prosleđuje vrednost primitivnog tipa kao argument metodu *toString()* koji vraća String ekvivalentan polaznoj vrednosti primitivnog tipa.
Videćemo kasnije da sve standardne klase imaju nestatički metod *toString()*.

U klasi String postoji i metod *concat(String)* koji se može koristiti za nadovezivanje stringova (na sadržaj stringa nad kojim se metod poziva dopisuje se sadržaj stringa koji je argument metoda; rezultat je novi string).
Primer:

```
String s1 = "prvi string";
System.out.println(s1.concat(" - drugi string"));
```

Sa verzijom 8 uveden je novi, statički metod za spajanje sadržaja stringova:
String.join(CharSequence delimiter, CharSequence... elements)
Prvi argument metoda je string koji predstavlja delimiter (graničnik) sadržaja koji se spajaju, a preostali argumenti (proizvoljan broj njih) su stringovi koji se spajaju.
Primer:

```
System.out.println(String.join(".", "prvi string", "drugi string"));
```

U Javi 8 uvedene su neke novine koje se tiču same implementacije operacije nadovezivanja (spajanja) stringova. Definisana je posebna klasa *StringJoiner* za ove potrebe.

Poređenje stringova

Izraz *string1 == string2* proverava da li dve String promenljive (string1 i string2) referišu na isti objekat klase String (u memoriji). Ako referišu na nezavisne objekte, rezultat izraza je *false*, bez obzira da li ta dva objekta sadrže isti string.
Zaključak: gornji izraz ne poredi same String objekte, već poredi njihove reference!

Za poređenje String objekata (stringova) na jednakost sadržaja koristi se metod *equals()*.
Za poređenje String objekata (stringova) na jednakost sadržaja uz ignorisanje veličine slova koristi se metod *equalsIgnoreCase()*.
Bilo koji od ova dva metoda primenjuje se nad objektom pomoću operatora ".", a argument metoda je String promenljiva ili String literal.

```
string1.equals(string2);
```

> *string1* i *string2* su reference na String objekte koji se porede na jednakost. Ako objekti sadrže isti string, rezultat je *true*, inače *false*.

Leksikografsko poređenje stringova vrši se pozivom metoda *compareTo(String)* čime se poredi String objekat nad kojim je metod pozvan sa String objektom čija se referenca prosleđuje kao argument metoda. Povratna vrednost može biti:

< 0	*String objekat je leksikografski "manji" od argumenta*
= 0	*String objekat je jednak argumentu*
> 0	*String objekat je leksikografski "veći" od argumenta*

Metod String.valueOf()

U klasi String postoji statički metod *String valueOf(...)* koji pravi i vraća String objekat od vrednosti proizvoljnog primitivnog tipa ili niza karaktera.

```
String doubleString = String.valueOf(3.1415926);
```

Pošto je metod *valueOf()* statički, pristupa mu se pomoću imena klase.

Metodi startsWith() i endsWith()

Upotrebom metoda *startsWith(String) i endsWith(String)* može se proveriti da li string počinje, odnosno završava datim stringom.

Metodi indexOf() i lastIndexOf()

Služe za pretraživanje stringa nad kojim se pozivaju. Metod indexOf() pretražuje string sleva udesno, a lastIndexOf() obrnutim smerom.

```
int indexOf(int ch)
int indexOf(int ch, int fromIndex)
int indexOf(String str)
int indexOf(String str, int fromIndex)
int lastIndexOf(int ch)
int lastIndexOf(int ch, int fromIndex)
int lastIndexOf(String str)
int lastIndexOf(String str, int fromIndex)
```

Svi metodi vraćaju traženi indeks ili -1 ako odgovarajući string, odnosno karakter nije pronađen.

Metod substring()

Služi za izdvajanje podstringova:

```
String substring(int) – vraća podstring datog stringa koji počinje od pozicije
zadate kao argument
String substring(int start, int end) – vraća podstring datog stringa koji
počinje na poziciji start, a završava se na poziciji end-1
```

Metodi replace() i trim()

```
String replace(char, char) – svaku pojavu prvog karaktera (prvi argument) u
stringu nad kojim se poziva zamenjuje drugim karakterom (drugi argument) i
vraća novodobijeni string.

String trim() - brisanje belina sa početka/kraja stringa. Vraća se
novodobijeni string.
```

Pravljenje String objekta od niza karaktera i obrnuto

Pravljenje niza karaktera od objekta klase String:

```
String text = "Pada kisa";
char[] textArray = text.toCharArray();
       /* pravi se niz ciji su elementi karakteri datog stringa */
```

Pravljenje objekta klase String od niza karatketa:

```
char[] textArray = { 'P', 'a', 'd', 'a', ' ', 'k', 'i', 's', 'a'};
String text = new String(textArray);
  /* poziv konstruktora klase String - pravi se novi objekat koji sadrži
     string sastavljen od karaktera datog niza */
```

2 Promenljivi stringovi

Objekti klase *String* ne mogu da menjaju svoj sadržaj.
Za rad sa stringovima koji mogu direktno da se modifikuju u Javi postoje dve standardne klase:

StringBuilder i *StringBuffer*

Ove dve klase se ne razlikuju po pitanju operacija koje omogućuju.
Jedina razlika je što se objekti klase *StringBuffer* mogu sigurno koristiti od strane više niti, pa su iz tog razloga metodi ove klase sinhronizovani. Rad sa nitima i sinhronizacija nisu deo ovog kursa, pa se za rad sa promenljivim stringovima može koristiti klasa *StringBuilder*. U nastavku je dat njen kratak opis.

Promenljive stringove treba koristiti kada su česte izmene stringova – dodavanje, brisanje, menjanje podstringova u stringu i drugo.

Pravljenje objekta klase *StringBuilder* vrši se na sledeći način:

```
  StringBuilder strbuf = new StringBuilder("Promenljivi string");
```

Za razliku od klase *String*, ne može se koristiti sledeća naredba, jer nije podržana implicitna konverzija tipa *String* u tip *StringBuilder*:

```
  StringBuilder strbuf = "Promenljivi string";
```

Moguće je sledeće:

```
String str = "Promenljivi string";
StringBuilder strbuf = new StringBuilder(str);
/* ili */
StringBuilder strbuf = new StringBuilder("Promenljivi string");
```

Objekat klase *StringBuilder* sadrži blok memorije (bafer), koji može, a ne mora da sadrži string, a ako sadrži, on ne mora da zauzima ceo bafer.
Dužina stringa u *StringBuilder* objektu može biti različita od dužine bafera sadržanog u objektu.
Veličina bafera naziva se kapacitetom (capacity) *StringBuilder* objekta.
Kada se jednom napravi *StringBuilder* objekat, dužina stringa sadržanog u njemu se može dobiti pozivom metoda length().

```
  StringBuilder strbuf = new StringBuilder("String");
  int length = strbuf.length();
```

Kada se pravi *StringBuilder* objekat iz postojećeg stringa, kapacitet objekta je *dužina stringa + 16*.

I kapacitet i dužina stringa izraženi su u broju *Unicode* karaktera, tako da će dva puta više bajtova biti zauzeto u memoriji.
Kapacitet bafera može biti i eksplicitno zadat:

```
StringBuilder newStr = new StringBuilder(50);
     // kapacitet je 50 Unicode karaktera, dakle 100 bajtova.
```

Ako se ne navede kapacitet, podrazumevana vrednost je 16 Unicode karaktera. Bafer je inicijalno prazan, tj. ne sadrži nijedan string.

Metodi za dužinu (kapacitet) bafera i dužinu stringa

```
int capacity() - vraća dužinu (kapacitet) bafera
void ensureCapacity(int) – menja podrazumevanu dužinu bafera StringBuilder
objekta. Zadaje se minimalna dužina kao argument metoda.
```

Primer:

```
newStr.ensureCapacity(40);
     // Ako je tekući kapacitet bafera objekta newStr manja od 40, kapacitet
     // će biti uvećan alokacijom novog većeg bafera kapaciteta većeg od 40
     // ili dva puta većeg od tekuće veličine uvećanog još za dva.

void setLength(int) - promena dužine stringa u StringBuilder objektu.
```

Napomena: dužina je karakteristika stringa sadržanog u *StringBuilder* objektu, dok je kapacitet karakteristika *StringBuilder* objekta.
Kada se uvećava dužina stringa u *StringBuilder* objektu, na postojeći string dodaju se karakteri sa pripadajućom UNICODE vrednošću '\u0000'.
Češće se ovaj metod koristi za odsecanje stringa.

Metod append()

Da bi se dodao string na kraj postojećeg stringa u *StringBuilder* objektu koristi se metod:

```
StringBuilder append(String)
```

Primer:

```
StringBuilder aString = new StringBuilder("Primer"");
aString.append(" nadovezivanja");  /* aString sada sadrzi string
                     "Primer nadovezivanja" */
```

Kapacitet bafera će se uvek automatski uvećati ako je potrebno prihvatiti veći string (po modelu: *2*tekući kapacitet+2*).

Metod *append()* vraća referencu na prošireni *StringBuilder* objekat i ona se može dodeliti drugom *StringBuilder* objektu:

```
StringBuilder aString = new StringBuilder("Primer");
StringBuilder bString = aString.append(" nadovezivanja");
     // aString i bString referišu na isti StringBuilder objekat.
```

Metod append postoji u više varijanti, u zavisnosti od tipa vrednosti koja se dodaje na sadržaj bafera. Na primer, za dodavanje vrednosti primitivnog tipa:

```
StringBuilder append(prim_type)
```

Dodavanje podniza datog niza karaktera:

```
StringBuilder buf = new StringBuilder("Test");
char[] text = {'a', 'b', 'c', 'd'};
buff.append(text, 1, 3);
  // na sadržaj bafera dodaju se karakteri  'b', 'c' i 'd'.
```

Metodi za pretraživanje

`int lastIndexOf(String)` - za dati argument vraća poziciju njegovog poslednjeg pojavljivanja kao podstringa unutar bafera. Pretraga se vrši zdesna ulevo.

`int lastIndexOf(String, int)` - kao i prethodni metoda, samo pretraga počinje od date pozicije.

Primer zamene podstringa u baferu:

```
StringBuilder str = new StringBuilder("jedan dva tri cetiri");
String podstring = "dva";
String zamena = "dvadeset";
int pozicija = str.lastIndexOf(podstring);

// pronalazi gde počinje podstring "dva" i zamenjuje ga stringom "dvadeset"
str.replace(pozicija, pozicija+podstring.length(), zamena);
```

Metod insert()

Umetanje stringa:

`insert(int, String)` - prvi argument je indeks pozicije u baferu gde treba umetnuti prvi karakter stringa koji je drugi argument.

Metodi setCharAt(), deleteCharAt(), delete() i reverse()

`setCharAt(int , char)` - promena jednog karaktera u StringBuilder objektu
`deleteCharAt(int)` - brisanje jednog karaktera
`delete(int, int)` - brisanje više karaktera – prvi argument je pozicija prvog karaktera za brisanje, a drugi je pozicija posle poslednjeg karaktera za brisanje.
`reverse()` - obrće sekvencu karaktera u StringBuilder-u.

Prevođenje u String

Pravljenje objekta klase *String* od objekta klase *StringBuilder* može se izvršiti upotrebom metoda ***toString()*** klase StringBuilder.

```
StringBuilder strbuf = new StringBuilder("Primer stringa");
String s = strbuf.toString();
```

Implementacija konkatenacije (nadovezivanja) stringova

Izvorno se za implementaciju operacije nadovezivanja stringova koristila klasa *StringBuffer.* Kako su njeni metodi sinhronizovani, a realne potreba za sinhronizacijom u ovom slučaju nema, ona je zamenjena klasom *StringBuilder*. Metod *toString()* klase *StringBuilder* koristi se od strane prevodioca zajedno sa metodom *append()* za impelmentaciju operacije nadovezivanja stringova.

Kada se zada naredba:

```
String str = "Objektno" + " orijentisano" + " programiranje";
```

prevodilac će je implementirati kao:

```
String str = new StringBuilder().append("Objektno")
                                .append(" orijentisano")
                                .append(" programiranje")
                                .toString();
```