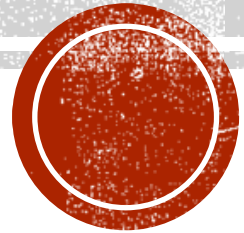
ОБЈЕКТНО ОРИЈЕНТИСАНО ПРОГРАМИРАЊЕ

ПРОГРАМСКИ ЈЕЗИК ЈАВА – 1

Низови у Јави



ПОЈАМ И ОСОБИНЕ НИЗА

- Низ је један од појмова који се јавља у великом броју програмских језика.
- Најчешће се дефинише као група променљивих истог типа
  које се појављују под заједничким именом.
- Низовни тип података у Јави има следећа својства:
  - садржи линеарно уређен, унапред познат, број чланова;
  - сви чланови су истог типа и имају заједничко име;
  - чланови могу бити примитивног или објектног типа;
  - сваком члану приступа се помоћу заједничког имена низа и индекса члана;
  - сви индекси су целобројног типа;
  - сви чланови низа се третирају као посебне променљиве
    (називају се и индексним променљивама).

НИЗ ЈЕ ОБЈЕКАТ

- Низовни тип у Јави је увек објектни тип, тј. низ је увек објекат.
- Ако низ има k индекса (k=1, 2, …), назива се k-димензионални низ.
  - Тако можемо говорити о једнодимензионалним, дводимензионалним, тродимензионалним итд. низовима.
  - Чланови k-димензионалног низа су индексне промењљиве са k индекса.
- Када је у питању једнодимензионални низ, ту разликујемо две могућности:
  1. једнодимензионални низ примитивних вредности,
  2. једнодимензионални низ објеката.

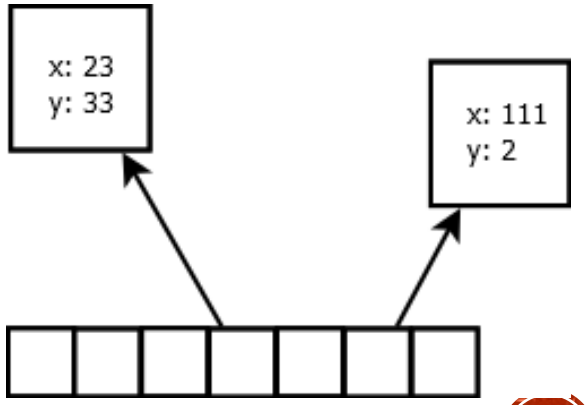
НИЗОВИ ПРИМИТИВНИХ ВРЕДНОСТИ

- Пример једнодимензионалног низа од 7 чланова је дат испод.

- Дакле, на позицији i се налази баш запис i-те вредности.
  На пример, на позицији 2 налази се број 6, односно конкретни битови 000...00110.

НИЗОВИ ОБЈЕКАТА

- Једнодимензионални низ објеката има нешто другачију меморијску организацију.
- У примеру елементи низа су тачке у дводименизионом простору (објекти) .
  - У овом случају објекти уопште нису на узастопним меморијским локацијама.
  - На узастопним меморијским локацијама су само референце ка њима.
- Низови објектних у Јави нису исто што и низови структура у C-у.
  - Показивачка аритметика C-у се ослања на величину структуре (sizeof оператор).
  - У Јави је величина референце увек иста и износи 4 бајта, без обзира на величину објекта на који се реферише.
  - Стога објекти на које показују могу бити било где у меморији.
  - Низови објеката у Јави се могу поистоветити са низовима показивача.
- Временска сложеност је, наравно, O(1).



ДЕКЛАРАЦИЈА НИЗА

- Низовна променљива се декларише у делу програма за декларацију.
- Број парова угластих заграда одређује број индекса односно димензионалност низа.

```
//Експлицитни начин - угласте заграде су задате после назива низовне променљиве
int pozicija[], a[];
String niska[][], bb[][];
Knjiga naslov[][][];

// или пре (имплицитни начин)
int [] pozicija, a;
String [][] niska, bb;
Knjiga [][][] naslov;
```

КРЕИРАЊЕ НИЗА

- Креирање низа се може реализовати на два начина:
  1. без иницијализације — помоћу оператора **new** (две под-варијанте);
  2. са иницијализацијом — набрајањем чланова.
- Други начин подразумева да су елементи низа познати већ у фази превођења, што је у аналогији са статичким низовима у C-у.
  - Међутим, за разлику од C-а, низови у Јави су увек смештени на хипу као (обе варијанте).

```
java.util.Scanner scan = new java.util.Scanner(System.in);
int n = scan.nextInt();
scan.close();
String[] niske = new String[n];// први начин – број елемената познат при извршавању
int [] brojac = new int[100]; // први начин – број елемената познат при превођењу

// следи други начин креирања
String godDoba[], s="лето";
godDoba ={"пролеће", s, "јесен", "зима"};
```

НИЗОВНА И ИНДЕКСНА ПРОМЕНЉИВА

- Приступ члановима низа је омогућен у форми **imeNiza[indeks]** где је:
  - **imeNiza** низовна променљива,
  - **indeks** је индексна променљива.
- Индексна променљива може бити:
  - целобројни литерал
  - или израз који производи целобројни литерал.
- Минимална вредност индекса је 0, а максимална је броје елемената низа мање 1.

```
int [] brojac = new int[100];
String [] godDoba={"пролеће", "лето", "јесен", "зима"};
...
brojac[50]=2; // Не би ваљало: brojac[100] = 3;
a=brojac[i+j]; // i и j могу бити познати у фази превођења или у фази извршавања
x = godDoba[3];
```

НИЗОВИ И ЦИКЛУСИ

- Будући да се **for** користи често у комбинацији са низовима, постоји и скраћена нотација (колекцијска **for** наредба).
- Она омогућава избегавање употребе бројачке променљиве.

```
for (int elemenat : niz)
    System.out.println(elemenat);
```

ПРИМЕР 2

- Написати Јава програм који одређује НЗД низа позитивних целих бројева. У реализацији користити колекцијску **for** наредбу.

```
// низ чији се НЗД тражи
int[] niz = { 24, 48, 96, 192, 36, 72, 144 };
// приказ низа
System.out.print("Низ: ");
for (int elemenat : niz)
        System.out.print(elemenat + " ");
System.out.println();
// одређивање НЗД-а
int nzd = niz[0];
for (int elemenat : niz)
        nzd = StrukturnoNzd.nzd2(nzd, elemenat);
// приказ резултата
      System.out.print("НЗД низа: " + nzd);
```

ПРИМЕР 3

- Написати Јава програм који омогућује да се оформи низ са задатим бројем елемената (који уноси корисник са стандардног улаза),
  тако да сваки члан низа добије исту (задату) вредност
  и да се применом колекцијског for циклуса прикажу вредности свих чланова низа.

ПРИМЕР 3 (2)

```
java.util.Scanner skener = new java.util.Scanner(System.in);
System.out.println("Унесите број елемената низа: ");
int n = skener.nextInt();
skener.close();
if (n <= 0) {
        System.err.println("Некоректан број елемената.");
        System.exit(1);
}
// сви низови у Јави су динамички алоцирани па није битно да ли је
// број елемената познат у фази компилације или тек при извршавању
double[] niz = new double[n];
double x = -23.34e1;
for (int i = 0; i < niz.length; i++)
        niz[i] = x;
for (double d : niz)
        System.out.printf("%8.2f ", d);
System.out.println();
```

```
Унесите број елемената низа:
5
-233.40  -233.40  -233.40  -233.40  -233.40
```

ПРИМЕР 4

- Написати Јава програм који пребројава елементе у низу бројева из интервала 1-10 и приказује их у бројчаном облику и у облику хистограма.

ПРИМЕР 4 (2)

```
int[] rezultati ={7, 3, 4, 9, 7, 6, 3, 10, 5, 6, 4, 3, 3, 3, 2, 5, 7, 9, 1};
int granica = 10;
int[] brojPojava = new int[granica];
// иницијализација
for (int i = 0; i < brojPojava.length; i++)
        brojPojava[i] = 0;
// пребројавање
for (int x : rezultati)
        brojPojava[(x - 1)]++;
// нумерички приказ
for (int i = 0; i < brojPojava.length; i++)
        System.out.printf("%d:%d %s", (i + 1), brojPojava[i],
                ((i + 1) % 8 == 0) ? "\n" : "\t");
System.out.printf("\n\n");
// графички приказ
for (int i = 0; i < brojPojava.length; i++) {
        System.out.printf("%3d:", i + 1);
        for (int j = 0; j < brojPojava[i]; j++)
                System.out.print("*");
        System.out.println();
}
```

ПРИМЕР 4 (3)

```
1:1       2:1       3:4       4:3       5:2       6:2       7:3       8:0
9:2       10:1

1:*
2:*
3:****
4:***
5:**
6:**
7:***
8:
9:**
10:*
```

ПРИМЕР 5

- Написати Јава програм који реализује стек помоћу низа.
- Потом га применити за учитавање секвенце реалних бројева и њихов приказ у обрнутом редоследу.

ПРИМЕР 5 (2)

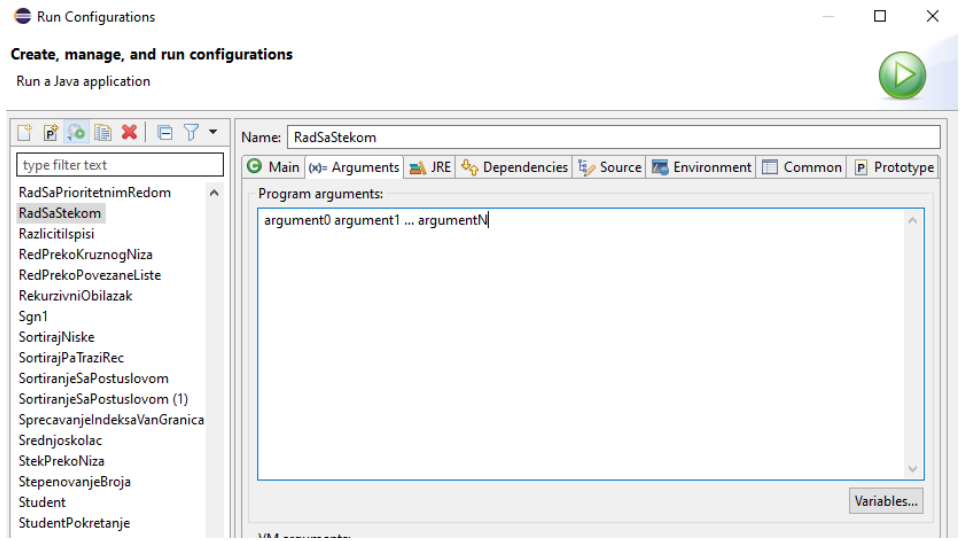
- Решење је предугачко за слајдове… Погледати у тексту поглавља.

АРГУМЕНТИ КОМАНДНЕ ЛИНИЈЕ

- Понекад је при покретању програма потребно задати разне улазне параметре.
- Ово се постиже помоћу аргумената команде линије метода **main()**.
  - Ови параметри су представљени у виду низа.
- Задавање аргумената се постиже путем конзоле при самом позивању програма.

```
java NazivPrograma argument0 argument1 ... argumentN
```

- Алтернативно се ово постиже путем одговарајућег дијалога у развојном окружењу.
  - Овај начин задавања аргумената има смисла само током развијања и тестирања програма.



ПРИМЕР 6

- Написати Јава програм за сабирање целих бројева
  који су задати као аргументи командне линије.
- Претпоставити да су аргументи дати коректно, тј. да ће бити увек цели бројеви.

ПРИМЕР 6 (2)

```
public static void main(String[] args) {
	int suma = 0;
	for(String a : args) {
		int broj = Integer.parseInt(a);
		suma+=broj;
	}
	System.out.println("Сума је "+suma);
}
```

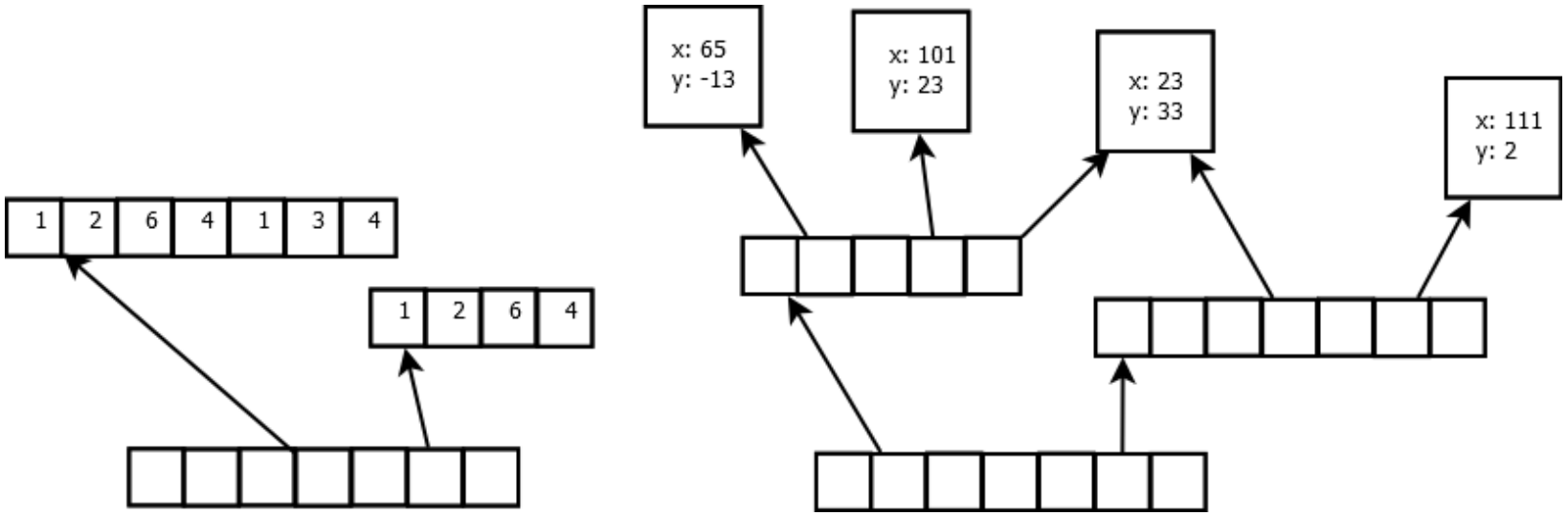
```
java SumirajArgumente 14 20 1 -2 13 352
Сума је 398
```

ВИШЕДИМЕНЗИОНАЛНИ НИЗ

- Референца у оквиру једнодимензионалног низа може показивати на други низ.
  - Што представља основ формирања вишедимензионалних низова односно низа низова.
- Временска сложеност приступа елементу k-димензионалног је исто O(1).
  - Разлог је то што се k не одређује динамички, већ се фиксира у фази превођења програм.
  - Дакле, у питању је увек константна величина.

ДВОДИМЕНЗИОНАЛНИ НИЗ

- Елементима дводимензионалног низа приступа се помоћу имена низа и два индекса.
- То подразумева низ референци где свака референца показује на једнодимензиони низ објеката или примитивних вредности.



КРЕИРАЊЕ ДВОДИМЕНЗИНАЛНОГ НИЗА

- Синтакса слична као код једнодимензионалног.
  1. Користи се **new** да би се креирао “спољашњи” низ референци ка “унутрашњим” низовима.
  2. Иницијална вредност сваке од тих референци је **null.**
  3. Након тога је потребно, помоћу **new,** креирати и сваки “унутрашњи” низ.
  4. (У случају да имамо дводимензионални низ објеката,
     потребно је имати још један ниво инстанцирања објеката.)

КРЕИРАЊЕ ДВОДИМЕНЗИОНАЛНОГ НИЗА (2)

```
int n = 10; // n може унети и корисник
int[][] nizNizova1 = new int[n][];

// неке од унутрашњих низова смо направили, а неки су остали null
nizNizova1[4]=new int[20];
nizNizova1[2]=new int[56];

// код оних које смо направили можемо подешавати и конкретну вредност
nizNizova1[4][3]=3;
// на локацијама где није експлицитно додељена вредност, биће записана вредност 0
nizNizova1[4][10]=11;

// слично и са низом низова објеката
Object[][] nizNizova2 = new Object[10][];
nizNizova2[2]=new Object[n];
nizNizova2[3]=new Object[12];

// с тим што је потребно имати и трећи ниво инстанцирања крајњих објеката
nizNizova2[2][8]=new Object();
nizNizova2[3][2]=new Object();
```

ТРОДИМЕНЗИОНАЛНИ НИЗ И НИЗОВИ ВЕЋИХ ДИМЕНЗИЈА

- Исти концепти и правила као и код дводимензионалних низова.
- Једина је разлика у додатном нивоу референци.

ПРИМЕР 9

- Написати Јава програм који задати текст
  трансформише у тродимензионални низ карактера
  такав да се на позицији (i, j, k) налази к-ти карактер j-те речи из i-те реченице.

ПРИМЕР 9 (2)

```
String tekst = "Као што се може видети, низ brojPojava чува вредности "
             + "броја појава за сваки од бројева из интервала… ";

String[] recenice = tekst.split("\\.");
char[][][] karakteri = new char[recenice.length][][];
for (int i = 0; i < recenice.length; i++) {
          String[] reci = recenice[i].trim().split(" ");
          karakteri[i] = new char[reci.length][];
          for (int j = 0; j < reci.length; j++) {
                    String rec = reci[j].replace(",", "");
                    karakteri[i][j] = new char[rec.length()];
                    for (int k = 0; k < rec.length(); k++)
                              karakteri[i][j][k] = rec.charAt(k);
                    }
          }
          for (int i = 0; i < karakteri.length; i++) {
                    System.out.println("--------------------"
                              + System.lineSeparator() + "Реченица " + (i + 1)
                              + System.lineSeparator() + "--------------------");
                    for (int j = 0; j < karakteri[i].length; j++) {
                              for (int k = 0; k < karakteri[i][j].length; k++)
                                        System.out.print(karakteri[i][j][k]);
                              System.out.println();
                    }
          }

}
```

КОРИШЋЕЊЕ КЛАСЕ ARRAYS

- За подршку у раду са низовима постоји класа **Arrays.**
- Неки од популарнијих метода ове класе користе се за:
  - сортирање **Arrays.sort**(),
  - бинарну претрагу **Arrays.binarySearch**(),
  - мешање **Arrays.shuffle**(),
  - копирање низа **Arrays.copyOf**(),
  - за попуњавање низа истим елементом **Arrays.fill**()
  - и слично.

МЕТОДИ СА АРГУМЕНТИМА ПРОМЕНЉИВЕ ДУЖИНЕ

- Подршка је додата у Јави 5 (уобичајено се зове **varargs**).
- Пре тога алтернативе за постизање сличног ефекта су подразумевале:
  1. формирање метода са истим називима и различитим аргумената, тзв. преоптерћење;
  2. прослеђивање низа аргумената методу;
- Могућност 1. је проблематична с обзиром да је већ у фази писања програма потребно предвидети све могуће комбинације аргумената, а то није увек могуће.
- Обе могућности су доводиле до проблема при одржавању кода.
- За декларацију метода са аргументом променљиве дужине користе се три тачке.

```
void f(int ... argumenti){...}
```

ПРИМЕР 11

- Написати Јава програм који користи метод са аргументом променљиве дужине.
- Метод нема повратну вредност и само исписује све прослеђене аргументе.

ПРИМЕР 11 (2)

```
public class IspisiArgumente {

    static void ispisiSve(int... argumenti) {
      System.out.println(argumenti.length + " аргумената:");
      for (int a : argumenti)
          System.out.println(a);
    }

    public static void main(String[] args) {
      ispisiSve(20);
      ispisiSve(11, 22, 34, -1);
      ispisiSve();
    }
}
```

```
1 аргумената:
20
4 аргумената:
11
22
34
-1
0 аргумената:
```

РАД СА АРГУМЕНТИМА ПРОМЕНЉИВЕ ДУЖИНЕ

- Приликом компилације аргумент променљиве дужине се преводи у низ.
  - Па се са променљивом за аргумент променљиве дужине може радити исто као са низом.
  - Предност у односу на експлицитне низове је мања количина понављајућег кода (енг. boilerplate code).
- Ако метод поред аргумента променљиве дужине има још неке аргументе, онда се аргумент променљиве дужине мора ставити на крај.

```
void f(String... a, int... b){} // грешка при компилацији
void f(int... a, String b){} // грешка при компилацији
void f(String b, int... a){} // ово је у реду
```

РАД СА АРГУМЕНТИМА ПРОМЕНЉИВЕ ДУЖИНЕ (2)

- Пре увођења аргумената промењиве дужине већ било написано доста кода.
  - Програмери су се у неким случајевима одлучивали да пређу на нову синтаксу, а у неким су задржавали стари приступ.
  - Задржавање старог приступа је оправдано, посебно кад је редослед битан.
- Постоје два принципа за безбедну употребу аргумената промењиве дужине:
  1. Аргументи промењиве дужине служе само за читање унутар метода, не и за измену.
  2. Не треба дозволити да референца ка аргументу промењиве дужине "побегне" из метода у метод која га је позвао, јер то може омогућити индиректно кршење принципа 1.
- Да ли ће се измена рефлектовати ван метода **f()**?

```
void f(int... a){
    a[...] = ...
}
```

- Неће, с обзиром да при позиву метода **f()** (из неког другог метода) нигде не фигурише назив низа, ово онемогућава да измењена вредност "побегне" у метод који позива **f()**.

ЛОШ ПРИМЕР УПОТРЕБЕ

- Међутим, постоји начин да се заобиђе претходна заштита.
  - Што се свакако не препоручује.
  - Напомињемо: ово није препоручени и дизајнирани начин употребе **varargs.**

```
public class IzmenaArgumentaPromenljiveDuzine {

    static int[] f(int ... argumenti) {
              argumenti[0] = 10;
              return argumenti;
    }

    public static void main(String[] args) {
        int[] argumenti = f(1,3,5,2,4);
        for(var a: argumenti)
            System.out.println(a);
        // исписује 10 3 5 2 4
    }
}
```

ПИТАЊА И ЗАДАЦИ

- Која су основна својства низовног типа података у програмском језику Јава?
- Примером илустровати проблем који није могуће решити без употребе низова (или неке друге колекције података), односно проблем који није могуће решити ако на располагању имамо само променљиве.
- Упоредити низове објектних типова у Јави са низовима структура података и низовима показивача ка структурама података у програмском језику C.
- Објаснити и примером илустровати разлике у декларацији низа са и без иницијализације.
- Написати Јава програм који за низ целих бројева, који се уноси са тастатуре, на екрану исписује парне бројеве који се налазе на непарним индексима.
- Написати Јава програм који за низ целих бројева, који се уноси са тастатуре, одређује најмању, највећу, просечну вредност и медијану.

ПИТАЊА И ЗАДАЦИ (2)

- Примером 5, дата је илустрација имплементације стека на којем се чувају реални бројеви преко једнодимензионалног низа. Написати Јава програм који илуструје имплементацију реда карактера преко једнодимензионалног низа. Ред је структура података коју карактерише FIFO (енг. First In First Out) принцип.
- Написати Јава програм који захтева унос два реална броја a и b са тастатуре, све док се не унесу бројеви такви да је a<b, а затим пребројава колико реалних бројева који се задају као аргументи командне линије налази у интервалу [a,b].
- Упоредити и примером илустровати задавање улазних параметара преко аргумената командне линије у програмском језику Јава и програмском језику C.
- Написати Јава програм који проверава да ли је матрица дијагонална, горња троугаона или доња троугаона. Матрица се представља дводимензионалним низом.

ПИТАЊА И ЗАДАЦИ

- По чему се разликују тродимензионални низови од дводимензионалних низова? Примером илустровати реалну ситуацију када је погодно користити тродимензионалне или четвродимензионалне низове.
- Истражити које још методе, поред метода приказаних у Примеру 10, се налазе у класи Arrays. Примером илустровати примену неких од тих метода.
- Како се пре увођења подршке за рад методима са променљивим бројем аргумената, постизао ефекат који ови методи омогућавају? Шта су предности, а шта недостаци старог и новог приступа?
- Написати метод која за аргумент узима произвољан број целих бројева и враћа највећи међу њима. Тестирати метод неколико пута задавањем различитог броја целих бројева.
- Који принципи се требају поштовати да би се постигла безбедна употреба аргумената променљиве дужине? Објаснити и илустровати примерима добру и лошу употребу.
- Упоредити механизам за рад са методима са променљивим бројем аргумената у програмском језику Јава и програмском језику C.