Podatkovni tipi in konstante

Programiranje 2, Tomaž Dobravec

Java – strogo tipiziran jezik

- Pred uporabo moramo spremenljivko deklarirati

```
int i;
```

Povemo, da bo spremenljivka `i` tipa `int` (cela števila)

- Hkrati z deklaracijo lahko spremenljivko tudi inicializiramo:

```
int i = 42;
```

tip ime vrednost

Primitivni podatkovni tipi

Java pozna 8 osnovnih (primitivnih) podatkovnih tipov:

Tip	Pomen	Velikost	Privzeta vrednost
`char`	znak	2	'\u0000'
`boolean`	logični vrednosti `true`/`false`	1	`false`
`byte`	celo število	1	0
`short`	celo število	2	0
`int`	celo število	4	0
`long`	celo število	8	0L
`float`	realno število (enojna natančnost)	4	0.0f
`double`	realno število (dvojna natančnost)	8	0.0d

število bajtov,
ki jih zasede
spr. tega tipa

Znakovni tip `char`

- Tip `char` uporabljamo za shranjevanje enega znaka

```
char znak = 'A';
```

- Znak je v *javi* predstavljen z dvema bajtoma, zato lahko hranimo poljuben ASCII ali Unicode znak.

```
char pi = '\u03C0'; //  Unicode znaka  pi
```

- Unicode tabela je razširitev ASCII tabele; v ASCII tabeli je zapisanih 128, v Unicode tabeli pa 65536 znakov.

Znakovni tip `char`

Posebni znaki

- Nekatere posebne znake zapišemo takole:

`'\n'`	prehod v novo vrstico
`'\b'`	pomik nazaj
`'\r'`	pomik na začetek vrstice
`'\t'`	tabulator
`'\\'`	znak \
`'\''`	znak '
`'\u####'`	izpis Unicode znaka

```
char c1  = 'A',        // znak A
     c2  = '\n',       // znak za prehod v novo vrstico
     c3  = '\\',       // znak \
     c4  = '\u010d';   // znak š
String s = "tek\bst"; // s se izpiše kot "test"
```

Znakovni tip `char`

Računanje z znaki

- Znak in število sta močno povezana.
- Vsakemu znaku pripada število in obratno; pretvorba poteka po ASCII oziroma po Unicode tabeli.
- Primer: znak 'A' se obravnava kot število 65 (ASCII tabela).

```
char a = 65;  // a = 'A' (ASCII tabela)

char b = 'A', c = 'D';
int i = c - b;  // i = 3 ('D' - 'A' = 3)

char p = ' ';
System.out.printf("ASCII koda presledka je: %d", (int) p);
```

Celoštevilski podatkovni tipi

- Java pozna 4 celoštevilske tipe:

Tip	Bajti	Obseg		
byte	1	-128	127	
short	2	-32.768	+32.767	(32kb)
int	4	-2.147.483.648	+2.147.483.647	(2ib)
long	8	–9,223,372,036,854,775,808	+9,223,372,036,854,775,807	(9Eib)

Vsi *java*nksi celoštevilski tipi so predznačeni.

- Konstante:

```
int a = 45;
int b = -1;
int c = 012;       // število 10  (osmiški zapis)
int d = 0xFF;      // število 255 (šestnajstiški zapis)
int e = 0b101010; // število 42  (dvojiški zapis)
```

Realni podatkovni tipi

- Java pozna 2 realna tipa:

`float` in `double`

- Konstante:

```
double d1 = 123.4;
double d2 = 1.234e2;
float  f1 = 123.4f;
```

Podatkovni tip za nize
Objekti razreda `String`

- Deklaracija in inicializacija:

```
String nizi ="To je nek niz";

String niz2;
niz2 = "Danes je lep dan";

String niz3 = new String("Tudi tako gre");
```

Niz zapišemo v "dvojnih narekovajih"

- Dolžina niza dobim z metodo `length()`:

```
String a="POMLAD";
int dolzina = a.length() ;        //   dolzina =  6
```

- `i`-ti znak niza dobim s klicem metode `charAt()`:

```
System.out.println(a.charAt(3));      // izpise L
```

Podatkovni tip za nize
Primerjanje dveh nizov

- Nizov **NE** primerjamo z operatorjem ==
- Za primerjavo uporabimo metodo `equals()`

```
String a = preberiIme(); // prebere ime in ga shrani v a

// NAROBE!                          // PRAVILNO!!!
if (a == "Lojze") ...               if (a.equals("Lojze"))  ...
```

Podatkovni tip za nize
Metode razreda `String`

Za delo z nizi uporabljamo različne metode:

- `compareToIgnoreCase()` ... primerja dva niza in pri tem zanemari velikost črk
- `startsWith()` ... ali se niz začne z danim podnizom
- `endsWith()` ... ali se niz konča z danim podnizom
- `indexOf()` ... poišče mesto prve pojavitve podniza
- `substring()` ... vrne podniz
- `replace()` ... zamenja prvo pojavitev podniza
- `replaceAll()` ... zamenja vse pojavitve podniza
- `split()` ... razbije na podnize
- ...

Primer:

```
String niz = "Danes_je_lep_dan!";
niz = niz.replaceAll("_", " ");
System.out.println(niz);  // izpis: Danes je lep dan!
```

Podatkovni tip za nize
Niza ne morem spreminjati

- Javanskega niza se ne da spreminjati!

- Če spremenim vrednost, se ustvari NOV objekt.

```
String dan = "ponedeljek";
dan = "torek";
System.out.println("Danes je " + dan);
```

objekt, ki hrani vrednost `"ponedeljek"` se zavrže, ustvari se nov niz, ki hrani vrednost `"torek"`

program izpiše `"Danes je torek"`

- Zato: objekte razreda `String` uporabljam samo za nize, ki se ne spreminjajo!

Razred `StringBuffer`

- Za "spreminjajoče se" nize uporabimo razred `StringBuffer`

```
StringBuffer ime = new StringBuffer("miha");
ime.setCharAt(0, 'M');
System.out.println(ime);
```

- Objektom tipa `StringBuffer` lahko spreminjamo vsebino (zgornji primer: z metodo `setCharAt()` smo spremenili prvo črko niza).

- Metode razreda `StringBuffer` (mnogo jih je enakih kot pri razredu `String`, dodane so metode za spreminjanje vsebine):
  - `setCharAt(), chatAt()`
  - `append(), insert(), delete(),`
  - `indexOf, substring(),`
  - `reverse(), replace(), getChars()`
  - …

Opomba: če ni skrbi za sinhronizacijo, lahko uporabimo tudi `StringBuilder`

Tabela

- Tabelo uporabljam za shranjevanje več vrednosti istega tipa.

- Do elementov tabele dostopamo z indeksom.

```
x[1]=3.14;
System.out.println(x[49]); // -42
```

```
double x[50];
```

x[0]	2
x[1]	3.14
x[2]	113.7
.	.
.	.
.	.
x[48]	
x[49]	-42

- Tabela ima nespremenljivo dolžino (*enkrat 50, vedno 50!*).

- Velikost tabele dobim z uporabo atributa `length`:

```
int velikost = x.length; // velikost = 50
```

Tabela – deklaracija in inicializacija

- **Deklaracija** tabele: `int [] tabela;`

- **Inicializacija** tabele:

```
tabela = new int[3]; // ustvarim tabelo …
tabela[0] = 1;       // … shranim podatke
tabela[1] = 6;
tabela[2] = 3;
```

- Lahko naredim tudi vse hkrati, takole:

```
int [] tabela = {1,6,3};
```

Tabela – indeksiranje podatkov

- Indeksi v tabeli gredo od `0` do `tabela.length-1`

```
tabela[0], tabela[1], …, tabela[tabela.length-1]
```

- Uporaba indeksa izven tega obsega vrže izjemo

```
ArrayIndexOutOfBoundsException
```

Razbitje nizov – metoda `split()`

Metoda `split()` razbije niz (glede na dana ločila) in ustvari tabelo posameznih delov niza.

Primer: z ukazom `vrstica.split(":")` niz razbijemo po dvopičjih:

```
String vrstica = "leo:x:131:100:Leo Novak:/home/leo:/bin/bash";
String polja[] = vrstica.split(":");
System.out.println(polja[4]);  // izpis: Leo Novak
```

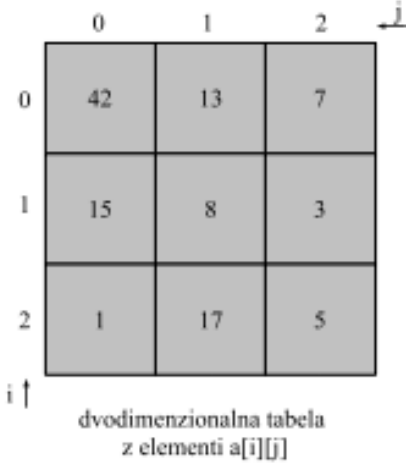
- Rezultat ukaza `split()` je vedno tabela nizov:

```
String deli[] = "3:7:15".split(":"); // deli[1]== "7"
```

- Ločilo je regularni izraz.

Večdimenzionalne tabele

- Pri večdimenzionalnih tabelah do elementov dostopamo z več indeksi:
  - 2. dimenzionalno: `a[1][2] = 3;`
  - 3. dimenzionalno: `a[4][0][7] = 1;`
  - ...



- Deklaracija in inicializacija s privzetimi vrednostmi:

```
int a[][] = new int [3][3];
```

(dobim tabelo velikosti 3x3, vrednosti v vseh poljih so 0)

Večdimenzionalne tabele

- Deklaracija in inicializacija s podanimi vrednostmi

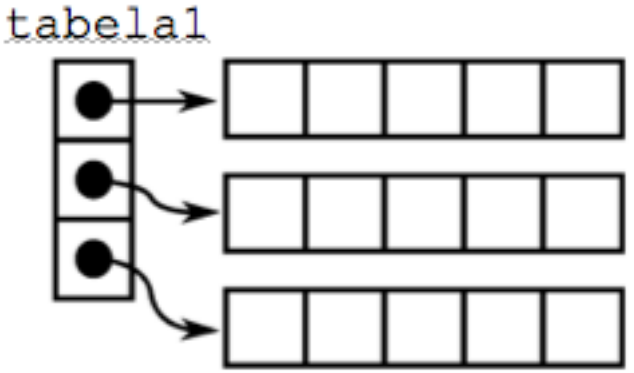
```
int y[3][3] = {
   {42,13,7},
   {15,8,3},
   {1,17,5}
}
```


42	13	7
15	8	3
1	17	5

Večdimenzionalne tabele

Večdimenzionalna tabela v javi je **tabela tabel**

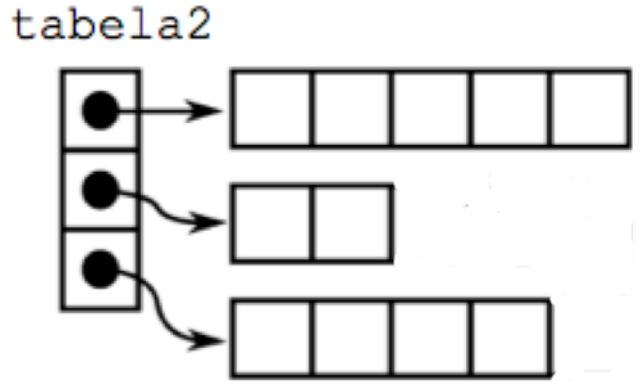
```
int tabela1[][] = new int[3][5];
```




```
int tabela2[][] = new int[3][];

tabela2[0] = new int[5];
tabela2[1] = new int[2];
tabela2[2] = new int[4];

tabela2[0][2] = 5; // OK
tabela2[1][2] = 4; // napaka
```



Primitivni in ovojni podatkovni tipi

- Java pozna 8 osnovnih podatkovnih tipov: `byte`, `short`, `int`, `long`, `float`, `double`, `char` in `boolean`.

- Za vsak osnovni podatkovni tip obstaja tudi ovojni (wrapper) podatkovni tip

Tip	Število bitov	Ovojni tip
byte	8	Byte
short	16	Short
int	32	Integer
long	64	Long
float	32	Float
double	64	Double
char	16	Character
boolean		Boolean

Primitivni in ovojni podatkovni tipi

```
int a = 5;                          // primitivni tip z vrednostjo 5
Integer aObj = new Integer(5);      // objekt tipa z "vrednostjo" 5
```

Prednost primitivnih podatkovnih tipov je, med drugim ta, da lahko nad njimi izvajamo atomarne operacije

```
int a = 5, b = 7;
int c = a + b;
```

Lahko to isto naredimo tudi z objekti?

```
Integer aObj = new Integer(5);
Integer bObj = new Integer(7);
Integer cObj = aObj + bObj;
```

Novejše verzije Jave (od 5 naprej) uporabljajo t.i. boxing / unboxing za podporo takim operacijam.

Primitivni in ovojni podatkovni tipi

- Ovojne podatkovne tipe bomo uporabljali za **pretvorbo** med nizom in osnovnim podatkovnim tipom (v obe smeri):

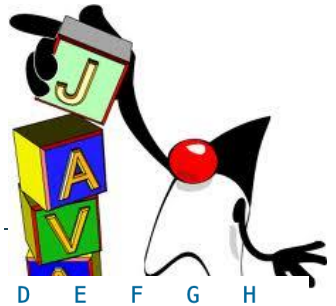
```
String odgovorStr = "42";
int    odgovor    = Integer.parseInt(odgovorStr);  // 42

String piStr = "3.14";
double pi    = Double.parseDouble(piStr);         // 3.14
```

in obratno, za pretvorbo primitivnega tipa v niz

```
String aStr = Integer.toString(15);  // "15"
String bStr = Double.toString(2.78); // "2.78"
```

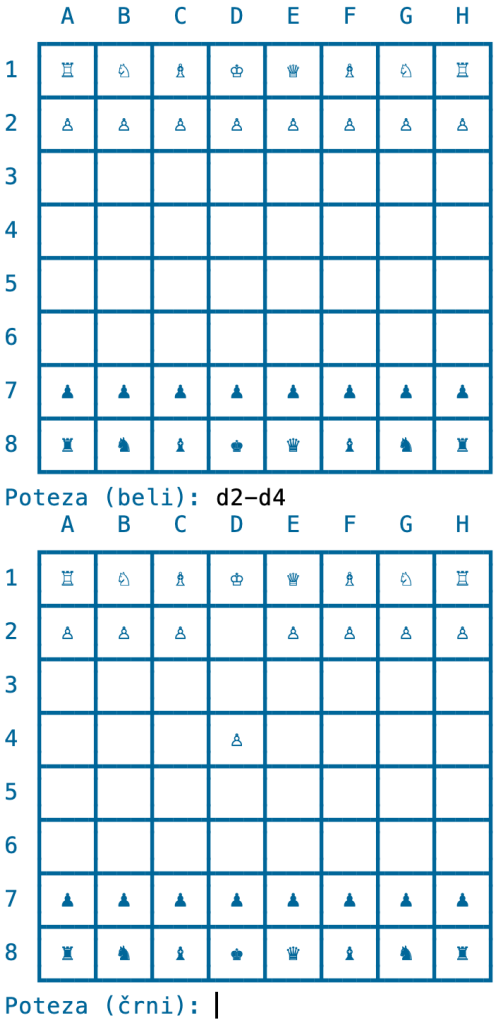
Naloga Šah



Sah.java

Naloga: Napiši program, ki dvema igralcema omogoča igranje igre šah. Program naj omogoča naslednje možnosti:

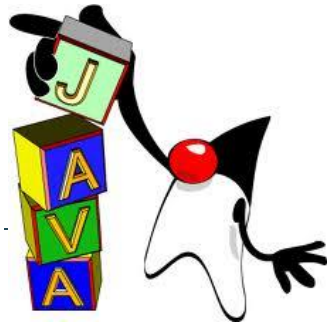
- program izdela ”igralno ploščo”, na kateri je začetna postavitev in ploščo izriše;
- program izmenjaje omogoča izvajanje potez: izmenjaje ”kliče” igralca, da vpiše potezo v obliki: izhodišče-cilj



Naloga

Šah

si/fri/sah/Sah.java

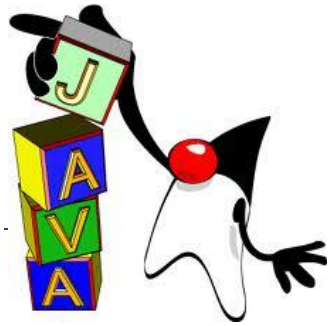


- Ob vpisani potezi program preveri njeno pravilnost:
  - ali sta izhodišče in cilj oblike ci (char, int)
  - ali je v izhodišču figura igralca, ki je na vrsti?
  - ali na cilju NI figura igralca, ki je na vrsti?

Če poteza ni pravilna, programigralca pozove k ponovnemu vpisu; v nasprotnem primeru program potezo ”izvrši”, ponovno nariše ploščo in nadaljuje z naslednjim igralcem.

- Program naj omogoča tudi naslednje možnosti:
  - izpis zgodovine vseh potez (ukaz `:zgodovina`)
  - razveljavitev potez (ukaz `:razveljavi`)
  - obrat igralne plošče (ukaz `:obrni`)
  - končanje igre (ukaz `:konec`)

Naloga Šah



Logika delovanja glavne metode:

```
igraj():
  ponavljaj
    narisi igralno ploščo
    preberi potezo ali ukaz
    če je vpisan ukaz
      če gre za pravi ukaz
        izvrši ukaz
      sicer
        javi napako (napačen ukaz)
    sicer
      preveri pravilnost poteze
      če je poteza pravilna
        izvrši potezo
        zapomni si potezo za zgodovino
        zapomni si potezo za razveljavitev
        preklopi na naslednjega igralca
      sicer
        javi napako (nepravilna poteza)
```

Podatkovne strukture programa:

- **igralna plošča**: dvodimenzionalna tabela znakov velikosti 8x8;
- **sklad za razveljavljene poteze:** tabela nizov oblike `"x:y:znak";`
- **zgodovina potez:** niz potez, ločenih z veljco (`"d2-d4,g7-g5,b1-a3"`);