Referencijalni tipovi

Nizovi i klasa `Object`

- Rekli smo da su svi referencijalni tipovi “u pozadini” realizovani kao klase, što je slučaj i sa nizovima
- Svaki niz direktno nasleđuje klasu `Object`. To znači da svaki niz sadrži sve članove klase `Object`, a takođe i da referenca bilo kog niza može biti dodeljena promenljivoj tipa `Object`
- Primer:

```
class ObjectNiz {
  public static void main(String[] args) {
    int[] x = {2, 4, 6};
    Object obj = x;
    int[] y = (int[])obj;
    x[0] = 1;
    for (int i = 0; i < y.length; i++)
      System.out.println(y[i]);
  }
}
```

- Izlaz:

```
1
4
6
```

Višedimenzionalni nizovi

- Višedimenzionalni nizovi su nizovi čiji su elementi takođe nizovi
- Broj dimenzija niza se određuje na sledeći način:
  - Niz čiji elementi nisu nizovi ima jednu dimenziju
  - Niz čiji su elementi nizovi ima za jedan veću dimenziju od dimenzije njegovog elementa
- Promenljiva čiji je tip višedimenzionalni niz se deklariše navođenjem imena tipa koji nije nizovski tip i onoliko parova otvorenih i zatvorenih uglastih zagrada koliki je broj dimenzija niza, nakon čega se navodi ime promenljive
- Pri inicijalizaciji višedimenzionalnih nizova ne moraju biti inicijalizovane sve dimenzije: mora se inicijalizovati samo prva dimenzija, a proizvoljan broj ostalih dimenzija može ostati neinicijalizovan
  - Prvo se navode uzastopne dimenzije koje se inicijalizuju brojevima elemenata (mora postojati bar jedna)
  - Zatim se navode neinicijalizovane dimenzije praznim parovima uglastih zagrada

Višedimenzionalni nizovi: primeri

Primeri: Deklaracije i inicijalizacije

```
int[][] matrica1;
int[] matrica2[];
int matrica3[][];

boolean[][] logTabela1;
logTabela1 = new boolean[][] {{true, false}, {false, true}};

boolean[][] logTabela2 = {{true, false}, {false, true}};
```

Višedimenzionalni nizovi: primeri

Primeri: Kreiranje višedimenzionalnih nizova na drugi način

```
int[][] matrica1;
matrica1 = new int[3][4];
int[] matrica2[] = new int[2][5]; // i deklaracija i kreiranje niza
System.out.println(matrica2.length);    // 2
System.out.println(matrica2[0].length); // 5
System.out.println(matrica2[1].length); // 5
int matrica3[][] = new int[3][];
System.out.println(matrica3.length);    // 3
System.out.println(matrica3[0]);        // null
matrica3[0] = new int[4]; // podnizovi mogu biti razlicitih duzina
matrica3[1] = new int[6];
matrica3[2] = new int[5];
System.out.println(matrica3[0].length); // 4
System.out.println(matrica3[1].length); // 6
System.out.println(matrica3[2].length); // 5
```

Višedimenzionalni nizovi: primer

```
class ZbirMatrica {
  public static void main(String[] args) {
    double[][] A = { {1.1, 2.2, 3.3, 4.1},
                     {0.4, -2.1, 1.9, 8.7},
                     {4.1, 2, 44, 23.2} };
    double[][] B = { {7.3, 12, 33.2, 6.2},
                     {0.0, 3.1, 2.7, 9.3},
                     {13.1, 3.8, 4.4, 23.8} };
    double[][] rez = new double[3][4];

    for (int i = 0 ; i < 3; i++)
      for (int j = 0; j < 4; j++)
        rez[i][j] = A[i][j] + B[i][j];

    System.out.println("Zbir matrica je:");
    for (int i = 0 ; i < 3; i++) {
      for (int j = 0; j < 4; j++)
        System.out.print(rez[i][j] + "\t");
      System.out.println();
    }
  }
}
```

Višedimenzionalni nizovi: primer

- Izlaz:

```
d:\PMF\Nastava\UUP\UUP2014\Predavanja\05>javac ZbirMatrica.java

d:\PMF\Nastava\UUP\UUP2014\Predavanja\05>java ZbirMatrica
Zbir matrica je:
8.4     14.2    36.5    10.3
0.4     1.0     4.6     18.0
17.2    5.8     48.4    47.0
```

Operatori nad referencijalnim tipovima

`=`	dodela
`(`*`imeTipa`*`)`	eksplicitna konverzija tipa
`==`	ispitivanje jednakosti
`!=`	ispitivanje nejednakosti
`?:`	uslovni operator
`.`	pristup članu
`instanceof`	ispitivanje tipa objekta
`+`	konkatenacija stringova
`new`	kreiranje instance
`[]`	pristup elementu niza

Dodela

- Operator dodele `=` je binarni operator kojim se promenljivoj sa leve strane operatora dodeljuje vrednost izraza sa desne strane operatora

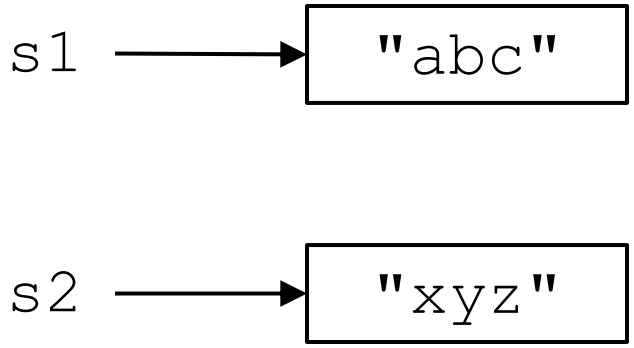
- Vrednost promenljive referencijalnog tipa nije sam objekat, već **referenca objekta** - zbog toga se dodelom vrednosti jedne promenljive drugoj ne pravi nova kopija objekta, već se kopira samo referenca na objekat, tako da posle dodele obe promenljive pokazuju na isti objekat

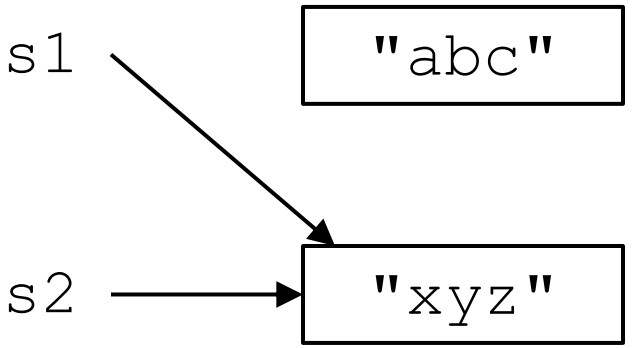
- Vrednost koja se dodeljuje promenljivoj referencijalnog tipa može biti:
  - Literal `null` ili
  - Referenca objekta čiji je tip jednak tipu promenljive ili
  - Referenca objekta čiji tip je moguće konvertovati u tip promenljive korišćenjem neke od proširujućih referencijalnih konverzija

Dodela: kopiranje referenci

```
String s1 = "abc";
String s2 = "xyz";
```

```
String s1 = "abc";
String s2 = "xyz";
s1 = s2;
```





Dodela: proširujuće konverzije

- Neke od proširujućih referencijalnih konverzija su:
  1. Konverzija iz bilo koje klase `P` u bilo koju klasu `N`, pod uslovom da je `P` podklasa klase `N`
  2. Konverzija vrednosti `null` u bilo koji referencijalni tip,
  3. Konverzija iz bilo kog niza u klasu `Object`,
  4. Konverzija iz bilo kog niza `A[]` u bilo koji niz `B[]`, pod uslovom da su `A` i `B` referencijalni tipovi i da postoji proširujuća referencijalna konverzija iz tipa `A` u tip `B`
- Proširujuće referencijalne konverzije biće detaljnije obrađene na kursu OOP1
- **Primeri:**

```
Object objekat = "abc"; // 1. String -> Object
String[] boje = {"crvena", "zelena", "plava"};
Object[] nizObj;
nizObj = boje;          // 4. String[] -> Object[]
objekat = nizObj;       // 3. Object[] -> Object
boje = null;            // 2. null -> String[]
```

Eksplicitna konverzija (kasting)

- Vrednost nekog tipa se može eksplicitnom konverzijom pretvoriti u odgovarajuću vrednost drugog tipa
- Ime ciljnog tipa se navodi u običnim zagradama i koristi se kao unarni operator
- Eksplicitna konverzija tipa se najčešće koristi prilikom dodele vrednosti promenljivoj kada se tip promenljive i tip vrednosti razlikuju i kada se oni ne mogu izjednačiti implicitnom proširujućom referencijalnom konverzijom
- Pravila koja definišu kad je moguća eksplicitna konverzija referencijalnih tipova su dosta složena i radiće se na kursu OOP1
- Za nas će biti dovoljno da se podsetimo primera:

```
int[] x = {2, 4, 6};
Object obj = x;
int[] y = (int[])obj;
```

- Dakle, ako je na neki objekat primenjena proširujuća konverzija (npr. `int[] -> Object`), dodela u "suprotnom smeru" mora da se radi pomoću kastinga

Ispitivanje (ne)jednakosti

- Operator ispitivanja jednakosti == i operator ispitivanja nejednakosti != se pored primene na vrednosti prostih tipova mogu primeniti i na vrednosti referencijalnih tipova
  - Njima se ispituje da li dve reference pokazuju ili ne pokazuju na isti objekat
- Rezultat primene operatora != je uvek suprotan od rezultata primene operatora ==
- Ovim operatorima nikada ne treba proveravati da li su dva objekta jednaka (po sadržaju), već za to treba koristiti specijalno napravljene metode
  - (standardan način je da se za tu svrhu redefiniše metod `equals` koji potiče iz klase `Object`)

Uslovni operator

- Ternarni operator `?:` može biti primenjivan i na referencijalne vrednosti
- Prvi operand ovog operatora je uvek logičkog tipa, a druga dva operanda mogu biti oba prostog tipa, ali mogu biti i oba referencijalnog tipa
- Ako su druga dva operanda ovog operatora istog referencijalnog tipa, tada će i rezultat operatora biti tog tipa. Kada su druga dva operanda različitog referencijalnog tipa, na primer jedan operand je tipa `A` a drugi operand je tipa `B`, tada za te tipove mora da važi sledeće:
  - Vrednost tipa `A` je moguće dodeliti promenljivoj tipa `B`, ili
  - Vrednost tipa `B` je moguće dodeliti promenljivoj tipa `A`
- Uslovni operator `?:` se sa referencijalnim vrednostima koristi slično kao i sa prostim

- **Primer:**

```
Object o = 1*2*3 != 1+2+3 ? new Object() : "abc";
```

Pristup članu referencijalnog tipa

- Operatorom **.** (tačka) se pristupa poljima, metodima i drugim članovima referencijalnih tipova
- Moguće je pristupiti samo onim članovima koji su u datom kontekstu vidljivi (više o vidljivosti kasnije)
- Nestatičkim članovima klase se pristupa navođenjem imena objekta, tačke i imena člana
- Statičkim članovima klase se pristupa navođenjem imena klase, tačke i imena člana, a moguće im je pristupiti i na isti način kao nestatičkim članovima, navođenjem imena nekog objekta te klase, tačke i imena statičkog člana

Operator `instanceof`

- Binarni operator `instanceof` se koristi samo kod referencijalnih tipova, njime se ispituje da li je tip prvog operanda jednak drugom operandu
- Prvi operand može biti samo neki objekat ili `null`
- Drugi operand je ime nekog referencijalnog tipa
- Ako je tip prvog operanda moguće eksplicitnom konverzijom konvertovati u tip naveden u drugom operandu, tada je vrednost izraza `true`, a inače je `false`
- Primer:

```
int[] x = {2, 4, 6};
Object obj = x;
if (obj instanceof int[])
  System.out.println("Moguca konverzija");
int[] y = (int[])obj;
```

Konkatenacija stringova

- Konkatenacija (spajanje) stringova se vrši binarnim operatorom **+**
- Ako su oba operanda stringovi, onda je rezultat novi string koji je jednak stringu koji bi nastao spajanjem stringova operanada
- Ako je samo jedan operand tipa `String` a drugi je nekog drugog tipa, onda se najpre vrednost operanda nestringovskog tipa konvertuje u tip `String` nakon čega se rezultat kreira isto kao u slučaju kada su oba operanda tipa `String`
- Konvertovanje u tip `String` je uvek moguće izvršiti:
  - Prosti tipovi – uobičajena konverzija
  - Literal `null` – u string `"null"`
  - Bilo koji objekat – u string nastao pozivom metoda `toString()` koji je deklarisan u klasi `Object`, pa ga sve klase nasleđuju
- **Primeri:**

```
System.out.println("abc" + 1 + 2); // dve konkatenacije, ispisuje abc12
System.out.println(1 + "abc" + 2); // dve konkatenacije, ispisuje 1abc2
System.out.println(1 + 2 + "abc"); // sabiranje i konkatenacija,
                                   // ispisuje 3abc
```

Klase: metodi

- Metodi nam daju način da kod koji bismo često ponavljali u programu “spakujemo” u jednu celinu i svedemo na jednu naredbu - poziv metoda
- Metode deklarišemo kao članove klasa - u istom odeljku gde deklarišemo i polja
- Ako se deklariše sa ključnom rečju `static`, metod je statički - jedinstven za ceo program, tj. klasu kojoj pripada
- U protivnom svaki objekat tipa klase sadrži svoju kopiju metoda
- Prvo ćemo se fokusirati na statičke metode

Deklaracija metoda

Primer: Recimo da u programu često imamo potrebu da štampamo sve cele brojeve iz određenog intervala. Da ne bismo ponavljali vrlo sličan kod (`for` petlju) svaki put, možemo deklarisati metod `stampajInterval`

```
static void stampajInterval(int a, int b) {
  for (int i = a; i <= b; i++) {
    System.out.print(" " + i);
  }
}
```

Deklaracija metoda

Primer: Recimo da u programu često imamo potrebu da proveravamo da li je znak jednak nekom od znakova iz skupa operacija koji nas zanima. Umesto da ponavljamo isti složen logički izraz, možemo deklaristi metod:

```
static boolean jeOperacija(char c) {
  return c == '+' || c == '-' || c == '*' || c == '/';
}
```

Deklaracija metoda

- Deklaracija metoda sadrži:
  - Zaglavlje metoda
  - Telo metoda
- Zaglavlje metoda počinje povratnim tipom, tj. tipom vrednosti koju metod vraća prilikom poziva, koji može biti:
  - Bilo koji tip, ili
  - Ključna reč `void`, čime označavamo da metod nema povratnu vrednost
- Zatim sledi ime metoda (identifikator), i u običnim zagradama lista formalnih parametara (argumenata):
  - 0 ili više deklaracija formalnih parametara, odvojene zarezima
- Formalni parametar se deklariše kao i promenljiva: navođenjem tipa i imena

Deklaracija metoda

- Telo metoda je blok, odnosno niz naredbi i deklaracija lokalnih promenljivih između { }

- Formalni parametri metoda su vidljivi u telu metoda (i samo u telu metoda) i mogu se koristiti kao i sve druge promenljive

- Štaviše, proširićemo opet definiciju promenljive, tako da sad promenljiva može biti:
  - Lokalna promenljiva
  - Polje objekta/klase
  - Formalni parametar metoda

Poziv metoda

- Ako metod ima povratnu vrednost, poziv metoda može da se pojavi:
  - U izrazu (na mestu gde je dozvoljena vrednost povratnog tipa metoda)
  - Kao naredba (tada se povratna vrednost zanemaruje)
- Ako je metod deklarisan sa `void`, tada poziv metoda može da se pojavi samo kao naredba
- Poziv metoda počinje navođenjem imena metoda, nakon čega se u običnim zagradama navode **stvarni parametri** metoda
- Stvarni parametri su vrednosti (izrazi) koji se pri pozivu metoda vezuju za formalne parametre, nakon čega se izvršava telo metoda, ali tako da se formalni parametri sada odnose na prosleđene stvarne parametre

Poziv metoda: primer

```
class MetodTest {
  static void stampajInterval(int a, int b) {
    for (int i = a; i <= b; i++) {
      System.out.print(" " + i);
    }
  }
  static boolean jeOperacija(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/';
  }
  public static void main(String[] args) {
    stampajInterval(5, 10);
    System.out.println();
    System.out.print("Unesite znak operacije (+, -, *, /): ");
    char c = Svetovid.in.readChar();
    if (jeOperacija(c)) {
      System.out.println("Uneli ste znak operacije " + c + ", hvala!");
    }
    else {
      System.out.println("Niste uneli znak operacije. C c c...");
    }
  }
}
```

Naredba `return`

- Naredba `return` izaziva vraćanje toka izvršavanja na mesto gde je metod pozvan

- Ako je metod deklarisan da ima povratnu vrednost, tada:
  - Vrednost odgovarajućeg tipa se navodi nakon ključne reči `return`
  - Izvršavanje naredbe `return` u telu metoda je **obavezno**, i to u svakoj mogućoj grani izvršavanja

- Ako je metod deklarisan sa `void`, tada:
  - Nakon ključne reči return se ne navodi ništa
  - Izvršavanje naredbe `return` u telu metoda **nije obavezno**, već se izvršavanje metoda može završiti i izvršavanjem poslednje naredbe u telu metoda

Poziv metoda: prosleđivanje parametara

- U Javi se prosleđivanje parametara pri pozivu metoda radi **po vrednosti** (engl. *call by value*)

- To znači da će u formalni parametar biti **kopirana vrednost** stvarnog parametra

- Posledica za proste tipove je da menjanjem vrednosti formalnog parametra u telu metoda ne utičemo na vrednost stvarnog parametra
  - Npr. ako metodu prosledimo promenljivu `x` kao stvaran parametar, a odgovarajući formalni parametar menjamo u telu metoda, nakon izvršavanja metoda promenljiva `x` će ostati neizmenjena

- **Napomena:** formalni i stvarni parametar mogu se zvati isto, ali to su *uvek* dve posebne stvari

Poziv metoda: prosleđivanje parametara

Primer:

```
static void stampajInc(double x) {
  x++; // menja vrednost formalnog parametra,
       // ali ne i stvarnog
  System.out.println("metod: x = " + x);
}
```

- Telo metoda `main`:

```
double x = 5.0;
stampajInc(x);
System.out.println("main: x = " + x);
```

- Izlaz:

```
metod: x = 6.0
main: x = 5.0
```

Poziv metoda: prosleđivanje parametara

- Kopiranje vrednosti stvarnog u formalni parametar za referencijalne tipove znači da će biti kopirana **referenca**

- Posledica je da promena vrednosti formalnog parametra referencijalnog tipa (tj. promena same reference) naredbom dodele i slično neće imati efekta na stvarni parametar (kao i kod prostih tipova)

- Međutim, promena **sadržaja objekta** (polja, elemenata nizova...) preko kopirane reference će biti oslikana i na stvarnom parametru

Poziv metoda: prosleđivanje parametara

```
class Tacka {
  double x, y;
}

class ParametarTest {
  static void menjajTacku(Tacka t) {
    t.x++; // promenice polje x stvarnog parametra
    t = new Tacka(); // nema efekta na stvarni parametar
  }
  public static void main(String[] args) {
    Tacka t1 = new Tacka();
    t1.x = 22.0;
    t1.y = 57.0;
    menjajTacku(t1);
    System.out.println("t1 = (" + t1.x + ", " + t1.y + ")");
  }
}
```

- Izlaz:

```
t1 = (23.0, 57.0)
```