Referencijalni tipovi

Nasleđivanje klasa

- Jedan od fundamentalnih koncepata objektno-orijentisanog programiranja je **nasleđivanje** klasa
- Kada se deklariše da klasa B nasleđuje klasu A, to znači da B preuzima članove od A, eventualno dodajući nove
- Na ovom kursu se konceptom nasleđivanja nećemo detaljno baviti, ali moramo biti svesni sledećih posledica:
  - Promenljivoj tipa A (“nadklase”) mogu se dodeljivati objekti tipa B (“podklase”)
  - Sve klase (i ostali referencijalni tipovi) implicitno nasleđuju specijalnu klasu `Object`
- Nasleđivanje će detaljno biti obrađeno na kursu OOP1

Statički i nestatički članovi

- Statički članovi klase - metodi i polja koja su deklarisana pomoću ključne reči `static`
- Mogu se koristiti i bez prethodnog kreiranja instanci klase, oni se zapravo i ne odnose na instance klase, već na samu klasu
- Svaki objekat klase sadrži svoju kopiju svih polja i metoda klase, osim onih polja i metoda koji su statički, jer statička polja i metodi pripadaju klasi a ne njenim instancama
- Kasnije ćemo detaljnije obraditi statičke metode i polja

Nizovi

- Nizovi predstavljaju grupu elemenata istog tipa koje se pojavljuju pod istim imenom
- Niz je specijalna vrsta objekta i sastoji se od elemenata kojima se pristupa pomoću njihovih indeksa (tj. rednih brojeva)
- Ubuduće, kada kažemo “niz” mislićemo na objekat nizovnog tipa podataka ili sam nizovni tip (biće jasno iz konteksta)
- Indeks prvog elementa u nizu je uvek `0`, dok je indeks poslednjeg elementa za `1` manji od ukupnog broja elemenata u nizu
- Tip elemenata niza može biti bilo koji tip, uključujući i tip niza
- Ako su elementi niza nizovi, tada takav niz zovemo više-dimenzionalan niz, inače se radi o jednodimenzionalnom nizu

Nizovi: deklaracija

- Promenljive tipa jednodimenzionalnog niza deklarišemo navođenjem tipa, imena promenljive i jednog para uglastih zagrada i može se inicijalizovati odmah prilikom deklaracije
  - Par uglastih zagrada može se navesti posle imena tipa ili posle imena promenljive

- Prilikom deklaracije niza ne navodi se veličina niza - ona se zadaje tek prilikom njegovog kreiranja `new` operatorom

- Nakon kreiranja, inicijalne vrednosti elemenata niza su nule (brojevni nizovi) , `false` vrednosti (logički nizovi) ili `null` ako su elementi niza referencijalnog tipa

Nizovi: deklaracija - primeri

Primeri: Deklaracije nizova i nezavisno kreiranje instanci

```
int[] nizCelih1;
int nizCelih2[];
String[] imena1;
String imena2[];
Lopta[] lopte;
Object objekti[];
...
nizCelih1 = new int[10];
imena1 = new String[] {"aca", "ceca", "daca"};
objekti = new Object[8];
```

Nizovi: deklaracija - primeri

Primeri: Deklaracije nizova i istovremeno kreiranje instanci

```
boolean[] logickiNiz = new boolean[18];
int[] celi1 = new int[] {1, 2, 3};
int[] celi2 = {1, 2, 3};
Tacka[] mojeTacke = new Tacka[10];
Object[] babeIZabe = new Object[] {new Tacka(),
                                   new Automobil(),
                                   new Vlasnik()};
```

Nizovi: deklaracija

- U primerima se mogu uočiti dva vida inicijalizacije nizova, koji se međusobno isključuju:
  - Navođenjem broja elemenata
  - Navođenjem samih elemenata

- Kod načina sa navođenjem elemenata dozvoljeno je izostaviti deo `new TipElementa[]`, kad se inicijalizacija radi pri deklaraciji, u protivnom se taj deo mora navesti
  - Dakle, kod sledećeg koda kompajler prijavljuje grešku:

```
int[] celi3;
celi3 = {1, 2, 3};
```

Nizovi: pristup elementima

- Elementima niza se pristupa tako što se prvo navede ime niza ili izraz čija je vrednost niz (pri čemu to ne sme biti izraz kreiranja niza), nakon čega se u uglastim zagradama navodi celobrojni izraz čija vrednost je indeks elementa kojem pristupamo
- Svaki niz ima i polje `length` koje sadrži broj elemenata niza zadat pri inicijalizaciji
- **Primeri:**

```
celi1[0]          // 1
imena1[1]         // "ceca"
logickiNiz[17]    // false
celi1.length      // 3
```

Nizovi: pristup elementima - primer 1

```
class FibonaciNiz {
  public static void main(String[] args) {
    int[] fib = new int[4];
    fib[0] = 0;
    fib[1] = 1;
    fib[2] = fib[0] + fib[1];
    fib[3] = fib[1] + fib[2];
    System.out.println("3. Fibonacijev broj je " + fib[3]);
  }
}
```

- Izlaz:

```
3. Fibonacijev broj je 2
```

Nizovi: pristup elementima - primer 2

```
class Automobil {
  String marka, proizvodjac;
  int godinaProiz;
  String boja;
  int brKonja, brVrata = 5;
  String regBroj;
}

class Vlasnik {
  String ime, prezime, JMBG;
  Automobil auto;
}
```

```
class AutomobiliNiz {
  public static void main(String[] args) {
    Vlasnik[] vlasnici = new Vlasnik[10];
    vlasnici[0] = new Vlasnik();
    vlasnici[0].ime = "Pera";
    vlasnici[0].prezime = "Peric";
    vlasnici[0].JMBG = "0101900800001";
    vlasnici[0].auto = new Automobil();
    vlasnici[0].auto.marka = "Yugo Koral 55";
    vlasnici[0].auto.proizvodjac = "Crvena zastava";
    vlasnici[0].auto.godinaProiz = 1989;
    vlasnici[0].auto.boja = "crvena";
    vlasnici[0].auto.brKonja = 55;
    vlasnici[0].auto.brVrata = 3;
  }
}
```

Nizovi: pristup elementima - primer 3

```
class MinNiz {
  public static void main(String[] args) {
    int brojeva;
    do {
      System.out.print("Unesite broj ulaznih brojeva > 0: ");
      brojeva = Svetovid.in.readInt();
    } while (brojeva <= 0);
    int[] nizBr = new int[brojeva];
    System.out.println("Unesite brojeve:");
    for (int i = 0; i < nizBr.length; i++) {
      System.out.print("Unesite " + i + ". broj: ");
      nizBr[i] = Svetovid.in.readInt();
    }
    int min = nizBr[0];
    for (int i = 1; i < nizBr.length; i++) {
      if (nizBr[i] < min) {
        min = nizBr[i];
      }
    }
    System.out.println("Minimalna vrednost u nizu je: " + min);
  }
}
```

Nizovi: pristup elementima - primer 3

- Izlaz:

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

d:\PMF\Nastava\UUP\UUP2014\Predavanja\05>java MinNiz
Unesite broj ulaznih brojeva > 0: 6
Unesite brojeve:
Unesite 0. broj: 7
Unesite 1. broj: 3
Unesite 2. broj: 5
Unesite 3. broj: 2
Unesite 4. broj: 99
Unesite 5. broj: 10
Minimalna vrednost u nizu je: 2
```

Polje `length`

- Nakon kreiranja niza, broj njegovih elemenata je fiksan i više se ne može promeniti
  - Ako se niz ponovo inicijalizuje operatorom `new`, u stvari se pravi novi niz, ne proširuje se stari
- Broj elemenata niza se može dobiti pomoću `length` polja niza. Ovo polje je konstantno (`final`), tj. njegova vrednost se ne može modifikovati
- Polje `length` može da se koristi da bismo saznali koliko argumenata je korisnik naveo prilikom poziva programa
- Navedenim argumentima pristupamo pomoću jedinog parametra metoda `main`, tipa `String[]`

Nizovi: pristup elementima - primer 4

```
class Argumenti {
  public static void main(String[] args) {
    if (args.length == 0) {
      System.out.println("Niste naveli ni jedan argument.");
    }
    else {
      System.out.println("Naveli ste " + args.length + " argumenata:");
      for (int i = 0; i < args.length; i++) {
        System.out.println(args[i]);
      }
    }
  }
}
```

Nizovi: pristup elementima - primer 4

- Izlaz:

```
d:\PMF\Nastava\UUP\UUP2014\Predavanja\05>java Argumenti
Niste naveli ni jedan argument.

d:\PMF\Nastava\UUP\UUP2014\Predavanja\05>java Argumenti 1 dva tri 4 pet
Naveli ste 5 argumenata:
1
dva
tri
4
pet
```

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, pre i/ili posle imena 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

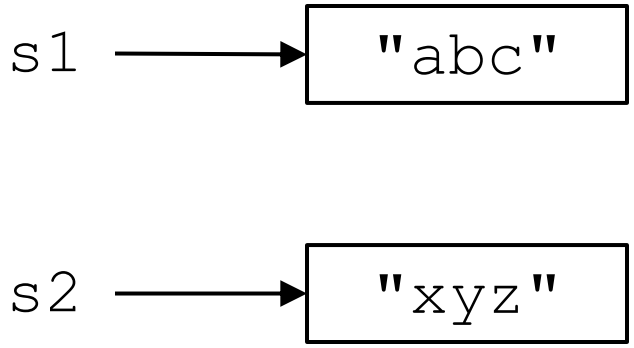
- Vrednost promenljive referencijalnog tipa nije sam objekat, već **referenca objekta** - zbog toga se dodelom vrednosti jedne promenljive drugoj ne pravi nova kopija objekta, već se kopira samo referenca na objekat, tako da posle dodele obe promenljive pokazuju na isti objekat

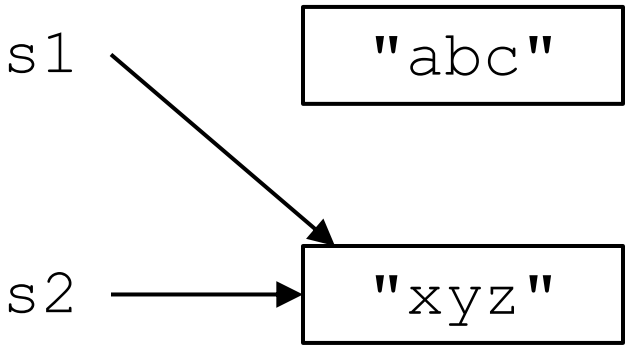
- Vrednost koja se dodeljuje promenljivoj referencijalnog tipa može biti:
  - Literal `null` ili
  - Referenca objekta čiji je tip jednak tipu promenljive ili
  - Referenca objekta čiji tip je moguće konvertovati u tip promenljive korišćenjem neke od referencijalnih konverzija

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
```