Referencijalni tipovi

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)
```

Poziv statičkog metoda

- Prilikom poziva statičkog metoda, ime metoda može se navesti u nekoliko oblika:

  - Ime klase, tačka, identifikator metoda

    Primer: `ParametarTest.menjajNesto(t1);`

  - Ime objekta, tačka, identifikator metoda

    Primer:
    ```
    ParametarTest pt = new ParametarTest();
    pt.menjajNesto(t1);
    ```

  - Samo identifikator metoda (ako je poziv u okviru iste klase)

    Primer: `menjajNesto(t1);`

Nestatički metodi

- Nestatički metodi deklarišu se bez ključne reči `static`
- Svaki objekat ima svoju kopiju metoda (koji se mehanizmom nasleđivanja može menjati)
- Koncept nestatičkih metoda je jedan od najosnovnijih u objektno-orijentisanom programiranju, jer omogućava da se u jednu celinu (objekat) “spakuju” ne samo podaci (polja), već i programski kod (nestatički metodi)
- Po pravilu, nestatički metod treba da se odnosi na i operiše nad objektom preko kog se poziva

Nestatički metodi: primer

```
class Tacka {
  double x, y;
  void translirajX(double pomeraj) {
    x += pomeraj;
  }
  void translirajY(double pomeraj) {
    y += pomeraj;
  }
}

class TackaTestOO {
  public static void main(String[] args) {
    Tacka t1 = new Tacka();
    Tacka t2 = new Tacka();
    t1.x = 1.0; t1.y = 2.0; t2.x = 5.0;
    t1.translirajY(10.0);
    t2.translirajX(5.0);
    System.out.println("t1 = (" + t1.x + ", " + t1.y + ")");
    System.out.println("t2 = (" + t2.x + ", " + t2.y + ")");
  }
}
```

Poziv nestatičkog metoda

- Prilikom poziva nestatičkog metoda, ime metoda može se navesti u nekoliko oblika:

  - Ime objekta, tačka, identifikator metoda

    Primer: `t1.translirajY(10.0);`

  - Samo identifikator metoda, ako je poziv u okviru nestatičkog metoda iz iste klase. Tada se pozivaju metodi vezani za isti objekat:

    Primer: Nov metod u klasi `Tacka`:

```
void translirajXY(double pomeraj) {
  translirajX(pomeraj);
  translirajY(pomeraj);
}
```

Statička polja

- Do sada smo posmatrli samo nestatička polja, gde svaki objekat ima svoju kopiju polja
- Slično kao kod metoda, i polja mogu da se deklarišu sa ključnom rečju `static`, čime postaju statička, odnosno jedinstvena za ceo program, tj. klasu
  - Jednom rečju, polja postaju **globalna**
- Imenima statičkih polja se pristupa analogno kao imenima statičkih metoda
- Treba imati na umu da je statičko polje uvek jedno isto, uprkos tome što mu se može pristupati preko različitih objekata

Statička polja: primer

```
class Tacka {
  double x, y;
  static final int NULA = 0;
}

class StaticTest {
  static final int JEDAN = 1;
  public static void main(String[] args) {
    Tacka t = new Tacka();
    System.out.println("Nule: " + Tacka.NULA + ", " + t.NULA);
    System.out.println("Jedan: " + JEDAN);
  }
}
```

Konstruktori

- Konstruktori se mogu posmatrati kao specijalna vrsta nestatičkih metoda koji se u životu objekta mogu izvršiti samo jednom, i to pri kreiranju objekta pomoću operatora `new` (vidi prethodno predavanje)
- Deklarišu se navođenjem imena klase, liste formalnih argumenata u običnim zagradama, i tela u vidu bloka
- Primer:

```
Tacka(double xInit, double yInit) {
  x = xInit; y = yInit;
}
```

Tada se objekat tipa `Tacka` kreira na sledeći način:

```
Tacka t1 = new Tacka(1.0, 2.0);
```

Konstruktori

- U klasi možemo deklarisati proizvoljan broj konstruktora, koji moraju imati različite liste argumenata (broj i/ili tipove)
- Ako ne deklarišemo ni jedan konstruktor, u klasi će implicitno biti deklarisan tzv. podrazumevani (engl. *default*) konstruktor, bez argumenata i sa praznim telom
- Ako deklarišemo bar jedan konstruktor, podrazumevani konstruktor se gubi
- U našem primeru, ako i dalje želimo mogućnost da kreiramo objekte klase `Tacka` bez navođenja argumenata (kao ranije), moramo konstruktor bez argumenata eksplicitno deklarisati:

```
Tacka() { }
```

Modifikatori pristupa

- Modifikatori pristupa (engl. *access modifiers*) su ključne reči koje se dodeljuju članovima klase i određuju njihovu vidljivost, tj. mogućnost da se tim članovima pristupi iz različitih delova koda
- U Javi postoje 4 modifikatora pristupa članovima: `public`, `private`, `protected`, i odsustvo modifikatora (*default*)
- Modifikator `public` označava da se članu može pristupiti iz bilo kog dela programa
- Modifikator `private` označava da se članu može pristupiti samo u kodu klase kojoj pripada
- Preostala dva modifikatora su "negde između" ova dva ekstrema
- Modifikatore pristupa ćemo detaljnije opisati kasnije

Paketi

- Paketi su jednostavan ali efektan način organizovanja referencijalnih tipova u Javi
- Svaka Java klasa (kao i drugi uvedeni referencijalni tipovi) pripada nekom paketu
- Svaki paket predstavljen je folderom istog imena kao paket, u koji se smeštaju .java fajlovi sa jedinicama prevođenja, i podfolderi koji predstavljaju pod-pakete
- Da bi se u jedinici prevođenja naznačilo kom paketu pripada, na početku .java fajla se stavlja ključna reč `package` i ime paketa
  - Ako se ovo izostavi (kao što mi radimo) klasa pripada tzv. podrazumevanom (*default*) paketu koji odgovara tekućem folderu

Paketi

- Da bi se u kodu neke klase (ref. tipa) koristili ref. tipovi iz drugih paketa, potrebno je uraditi nešto od sledećeg:
  - Koristiti puno kvalifikovano ime ref. tipa, navođenjem imena paketa, svih podpaketa, i na kraju imena tipa, razdvojenih tačkom, svuda u jedinici prevođenja (.java fajlu) gde se ref. tip koristi
    Primer: `java.io.File`
  - Na početku jedinice prevođenja pomoću ključne reči `import` uvesti ime referencijalnog tipa, nakon čega se pri navođenju imena tipa ne moraju koristiti imena paketa

    Primer: Na početku .java fajla: `import java.io.File;`

    U ostatku .java fajla navodimo samo: `File`
  - Ako želimo da koristimo više ref. tipova iz istog paketa, možemo sva imena iz paketa uvesti korišćenjem znaka `*`

    Primer: Na početku .java fajla: `import java.io.*;`

    U ostatku .java fajla postaju dostupna sva imena iz paketa `java.io`: `File`, `Reader`, `Writer`...

Važnije API klase u Javi

- Uz Javu, odnosno Java Development Kit (JDK) dolazi i velika biblioteka gotovih klasa - Application Programming Interface (API)
- Opisaćemo mali broj važnijih klasa:
  - `Object`
  - `String`
  - Referencijalni ekvivalenti prostih tipova
- Sve ove klase pripadaju paketu `java.lang` koji ima specijalan status: ne mora se navoditi u kvalifikovanom imenu, niti se njegov sadržaj mora uvoziti
- Pun pregled Java API nalazi se u zvaničnoj dokumentaciji:

  http://docs.oracle.com/javase/8/docs/api/

Klasa `Object`

- Osnovna klasa u hijerarhiji svih Java klasa, odnosno referencijalnih tipova
- Svi referencijalni tipovi implicitno nasleđuju klasu `Object`, što znači da nasleđuju njene članove
- Pomenućemo dva bitna člana - metoda:

  `public boolean equals(Object obj)`: poredi tekući objekat sa `obj`. Osnovna implementacija poredi reference operatorom ==, ali je namena metoda da se redefiniše u drugim klasama i radi poređenje *sadržaja* objekata

  `public String toString()`: vraća tekstualnu reprezentaciju tekućeg objekta. Osnovna implementacija vraća interni kod objekta, a očekuje se da svaka klasa za koju postoji potreba za `String` reprezentacijom redefiniše ovaj metod. Metod `toString` ima specijalan status - implicitno se poziva pri konverzijama objekata u string (pri štampanju, konkatenaciji stringova i sl.)

Klasa `Object`: primer metoda `toString()`

```
class Tacka {
  double x, y;
  ...
  public String toString() {
    return "(" + x + ", " + y + ")";
  }
}

...
System.out.println("t1 = " + t1.toString());
// .toString() se moze izostaviti
System.out.println("t2 = " + t2);
...
```

Klasa `String`

- Osnovna klasa za predstavljanje teksta u Javi

- Objekti tipa `String` su **nepromenljivi** (engl. *immutable*): jednom kreiran, sadržaj stringa ne može se menjati, već se uvek kreira novi objekat

- Klasa `String` ima specijalan status u Javi, po dva osnova:
  - Nove instance se mogu kreirati bez korišćenja operatora `new`, prostim navođenjem literala
  - Operator `+` može da se koristi za konkatenaciju stringova

Klasa `String`

- Bitni metodi:

  `public boolean equals(Object obj)`: ako je `obj instanceof String`, poredi sadržaj tekućeg stringa sa `obj` i vraća `true` ako su jednaki, u protivnom vraća `false`

  `public boolean equalsIgnoreCase(String anotherString)`: poredi sadržaj tekućeg stringa sa `anotherString` i vraća `true` ako su jednaki, ignorišući razliku između malih i velikih slova, u protivnom vraća `false`

  `public int length()`: vraća dužinu (broj karaktera) tekućeg stringa

  `public char charAt(int i)`: vraća znak na poziciji `i` u tekućem stringu (pozicije, odnosno indeksi kreću od `0`)

Klasa `String`: primer

```
class StringTest {
  public static void main(String[] args) {
    System.out.print("Unesite jedan string: ");
    String s1 = Svetovid.in.readLine();
    System.out.print("Unesite jos jedan: ");
    String s2 = Svetovid.in.readLine();
    System.out.println("Duzine stringova su " + s1.length() +
                       " i " + s2.length());
    System.out.println("Stringovi pocinju znacima " + s1.charAt(0) +
                       " i " + s2.charAt(0));
    System.out.println("Stringovi " +
                       (s1.equals(s2) ? "JESU" : "NISU") +
                       " jednaki");
    System.out.println("Ako se zanemari velicina slova, stringovi " +
                       (s1.equalsIgnoreCase(s2) ? "JESU" : "NISU") +
                       " jednaki");
  }
}
```

Klasa `String`: primer

- Izlaz:

```
d:\PMF\Nastava\UUP\UUP2014\Predavanja\06>javac StringTest.java

d:\PMF\Nastava\UUP\UUP2014\Predavanja\06>java StringTest
Unesite jedan string: aaaah
Unesite jos jedan: AaaaH
Duzine stringova su 5 i 5
Stringovi pocinju znacima a i A
Stringovi NISU jednaki
Ako se zanemari velicina slova, stringovi JESU jednaki
```

Referencijalni ekvivalenti prostih tipova

- Za svaki prost tip, Java API sadrži odgovarajuću klasu koja predstavlja njegov referencijalni ekvivalent
- Ovo su tzv. *wrapper* klase, odnosno klase koje služe da se vrednost prostog tipa “umota” u objekat
- Svrha ovih klasa je da omoguće korišćenje prostih tipova i u situacijama kad je neophodno koristiti referencijalni tip
  - Na primer, u neke strukture podataka kao što su standardne kolekcije iz Java API mogu da se smeštaju samo objekti
- Uz to, ove klase definišu korisne metode za rad sa prostim tipovima

Referencijalni ekvivalenti prostih tipova

Prost tip	Referencijalni tip
`boolean`	`Boolean`
`char`	`Character`
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`double`	`Double`
`float`	`Float`

Referencijalni ekvivalenti prostih tipova

Primer:

```
Integer pet = new Integer(5); // boxing
int p = pet.intValue();       // unboxing
int pedespet = Integer.parseInt("55");
Character a = new Character('a');
char c = a.charValue();
System.out.println("Brojevi: " + pet + ", " + p + ", " + pedespet);
System.out.println("Znak " + c + " " +
  (Character.isLetter(c) ? "JESTE" : "NIJE") + " slovo");
System.out.println("Znak " + c + " " +
  (Character.isDigit(c) ? "JESTE" : "NIJE") + " cifra");
System.out.println("Znak " + c + " " +
  (Character.isUpperCase(c) ? "JESTE" : "NIJE") + " veliko slovo");
System.out.println("Znak " + c + " " +
  (Character.isLowerCase(c) ? "JESTE" : "NIJE") + " malo slovo");
```

Referencijalni ekvivalenti prostih tipova

- Izlaz:

```
d:\PMF\Nastava\UUP\UUP2014\Predavanja\06>java WrapperTest
Brojevi: 5, 5, 55
Znak a JESTE slovo
Znak a NIJE cifra
Znak a NIJE veliko slovo
Znak a JESTE malo slovo
```

Nabrojivi tip

- Podsetimo se da tip podataka određuje skup vrednosti koje promenljiva može da ima
- Ako postoji potreba za malim skupom konstantnih vrednosti koji se neće menjati, sa jednostavnim operacijama koje će se primenjivati (poređenje jednakosti, i ne puno više od toga), može se koristiti nabrojivi tip (engl. *enumerated type*, skraćeno *enum*)
- Nabrojivi tip se definiše pomoću ključne reči `enum`, nakon čega se navodi ime nabrojivog tipa, i unutar `{}` imenovane konstante razdvojene zarezima
- **Primer:**

```
enum OsnovneBoje {
   CRVENA, ZELENA, PLAVA
}
```

- Lista konstanti može se završiti i znakovima `;` i `,`

Nabrojivi tip: primer

```
enum OsnovneBoje {
  CRVENA, ZELENA, PLAVA
}
enum Oblici {
  TROUGAO, KVADRAT, PETOUGAO, SESTOUGAO, KRUG
}
class EnumTest {
  public static void main(String[] args) {
    OsnovneBoje boja = OsnovneBoje.ZELENA;
    Oblici oblik = Oblici.KVADRAT; // mora kvalifikovano
    if (boja == OsnovneBoje.PLAVA)
      System.out.println(boja + ": volim tu boju");
    else
      System.out.println(boja + ": ne volim je");
  }
}
```

Nabrojivi tip

- Iako je referencijalni tip, nad nabrojivim tipom se ne primenjuje operator `new`: sve moguće instance su već kreirane i reference na njih dodeljene nabrojanim konstantama
- Nabrojivi tip može se koristiti u `switch` naredbi:

```
switch (oblik) {
  case TROUGAO: // mora nekvalifikovano
    System.out.println("Vidim trougao!");
    break;
  case KVADRAT:
    System.out.println("Vidim kvadrat!");
    break;
  default:
    System.out.println("Vidim nesto drugo...");
}
```

Nabrojivi tip

- Statički metod `values()` svakog nabrojivog tipa vratiće niz nabrojanih konstanti, što omogućava jednostavnu iteraciju kroz sve moguće vrednosti nabrojivog tipa, u redosledu u kom su nabrojane:

```
OsnovneBoje[] sveBoje = OsnovneBoje.values();
for (OsnovneBoje b : sveBoje) {
  System.out.println(b);
}
```

- Izlaz:

```
CRVENA
ZELENA
PLAVA
```