Statička implementacija polinoma

Polinom - statička implementacija

- Prema tome, pogodna struktura za predstavljanje polinoma je:

```
class Polinom {
  static final int maxSt = 100;
  double[] k = new double[maxSt+1];
  int st = -1;
}
```

- Polje `maxSt` je `static final`, što znači da predstavlja jedinstvenu konstantu za sve polinome
- Niz koeficijenata `k` sadrži redom a_0, a_1, ..., a_n, gde $0 \leq n \leq$ `maxSt`
- Polje `st` sadrži n, a na početku ga inicijalizujemo na `-1` da bi objekat tipa `Polinom` predstavljao nula-polinom
  - (elementi niza `k` se automatski inicijalizuju na `0.0`)

Polinom - statička implementacija

- Operacije nad polinomima realizovaćemo kao statičke metode u klasi `PolinomN`:

```
class PolinomN {
  /* Anulira polinom p */
  static void anuliraj(Polinom p) { ... }

  /* Kreira i vraca kopiju polinoma p */
  static Polinom kopiraj(Polinom p) { ... }

  /* Pronalazi stepen polinoma p i smesta ga u strukturu */
  static void nadjiStepen(Polinom p) { ... }

  /* Izracunava vrednost polinoma p za dato x */
  static double izracunaj(double x, Polinom p) { ... }

 /* Ucitava polinom */
  static Polinom ucitaj() { ... }

  /* Stampa polinom p */
  static void stampaj(Polinom p) { ... }
```

Polinom - statička implementacija

```
  /* Sabira polinome p1 i p2 vracajuci zbir */
  static Polinom saberi(Polinom p1, Polinom p2) { ... }

  /* Oduzima polinom p2 od polinoma p1 vracajuci razliku */
  static Polinom oduzmi(Polinom p1, Polinom p2) { ... }

  /* Mnozi broj c sa polinomom p vracajuci proizvod */
  static Polinom brojPuta(Polinom p, double c) { ... }

  /* Mnozi polinom p1 sa p2 vracajuci proizvod */
  static Polinom puta(Polinom p1, Polinom p2) { ... }

  /* Deli dva polinoma, vracajuci kolicnik i ostatak u nizu */
  static Polinom[] deli(Polinom p1, Polinom p2) { ... }
}
```

Polinom - statička implementacija

```
/* Pronalazi stepen polinoma p i smesta ga u strukturu */
static void nadjiStepen(Polinom p) {
  if (p != null) {
    final double eps = 1.0E-5;
    p.st = p.maxSt;
    while (p.st > -1 && (Math.abs(p.k[p.st]-0.0) < eps)) {
      /* Postavlja stepen polinoma za prvi koeficijent
         razlicit od 0, ili na -1 ako su svi koeficijenti
         0. Pri poredjenju sa nulom pitamo se da li
         je koeficijent dovoljno blizu nule
      */
      p.st--;
    }
  }
}
```

Polinom - statička implementacija

- Vrednost polinoma se najjednostavnije izračunava pomoću Hornerove šeme, tj. programiranjem sledećeg postupka:

$$p(c) = (\ldots(a_n c + a_{n-1})c + a_{n-2} \ldots + a_2 c + a_1)c + a_0$$

```
/* Izracunava vrednost polinoma p za dato x */
static double izracunaj(double x, Polinom p) {
  if (p == null)
    return Double.NaN;
  double rezultat;
  if (p.st == -1)
    rezultat = 0.0;
  else {
    rezultat = p.k[p.st];
    for (int i = p.st - 1; i >= 0; i--)
      rezultat = rezultat * x + p.k[i];
  }
  return rezultat;
}
```

Polinom - statička implementacija

- Učitavanje polinoma svodi se na učitavanje stepena polinoma i koeficijenata svih monoma

- Treba paziti da učitani stepen polinoma bude u dozvoljenim granicama

- Takođe, ne treba dozvoliti da koeficijent monoma najvećeg stepena bude 0, osim ako je u pitanju nula-polinom
  - Zato ćemo učitavanje koeficijenta monoma najvećeg stepena realizovati posebno

Polinom - statička implementacija

```
/* Ucitava polinom */
static Polinom ucitaj() {
  Polinom p = new Polinom();
  int pom;
  double koef;
  do {
    System.out.print("Stepen polinoma (>= 0, <= " + p.maxSt + "): ");
    pom = Svetovid.in.readInt();
  } while (pom < 0 || pom > p.maxSt);
  p.st = pom;
  do {
    System.out.print("Koeficijent uz x^" + p.st + ": ");
    koef = Svetovid.in.readDouble();
  } while (p.st > 0 && koef == 0.0);
  p.k[p.st] = koef;
```

Polinom - statička implementacija

```
  if (p.st == 0 && p.k[p.st] == 0.0) {
    p.st = -1;
  }
  else {
    for (int i = p.st - 1; i >= 0; i--) {
      System.out.print("Koeficijent uz x^" + i + ": ");
      koef = Svetovid.in.readDouble();
      p.k[i] = koef;
    }
  }
  return p;
}
```

Polinom - statička implementacija

- Polinom se može štampati po opadajućim ili rastućim stepenima monoma
- Sledeći metod štampa polinom po opadajućim stepenima
- Ako je koeficijent monoma 0, taj monom ne treba štampati
- Metod je najzgodnije podeliti na tri dela:
  - najpre štampati **vodeći monom**, jer se ne štampa vodeći predznak +,
  - zatim štampati sve **"unutrašnje" monome**, tj. monome u kojima treba štampati *x* (pri tom kod x^1 ne štampati 1),
  - na kraju štampati **slobodan član** (ne štampati *x*).

Polinom - statička implementacija

```
/* Stampa polinom p */
static void stampaj(Polinom p) {
  if (p != null) {
    if (p.st > 0) { /* ako polinom ima x stampati: */
      if (p.k[p.st] < 0.0) /* najpre vodeci monom */
        System.out.print("-");
      if (Math.abs(p.k[p.st]) != 1.0)
        System.out.print(Math.abs(p.k[p.st]));
      if (p.st > 1)
        System.out.print("x^" + p.st);
      else
        System.out.print("x");
```

Polinom - statička implementacija

```
/* stampati ostale monome sa x */
for (int i = p.st - 1; i >= 1; i--) {
  if (p.k[i] != 0.0) {
    if (p.k[i] > 0.0)
      System.out.print("+");
    else
      System.out.print("-");
    if (Math.abs(p.k[i]) != 1.0)
      System.out.print(Math.abs(p.k[i]));
    if (i > 1)
      System.out.print("x^" + i);
    else if (i == 1)
      System.out.print("x");
  }
}
```

Polinom - statička implementacija

```
      /* na kraju stampati slobodan clan */
      if (p.k[0] != 0.0) {
        if (p.k[0] > 0.0)
          System.out.print("+" + p.k[0]);
        else
          System.out.print("-" + Math.abs(p.k[0]));
      }
    }
    else { /* ako polinom nema x */
      if (p.k[0] < 0.0)
        System.out.print("-" + Math.abs(p.k[0]));
      else
        System.out.print(p.k[0]);
    }
  }
}
```

Polinom - statička implementacija

- Sabiranje i oduzimanje polinoma može da se realizuje jednim istim metodom, jer je postupak vrlo sličan: pri sabiranju polinoma koeficijenti uz monome istog stepena se sabiraju, a kod oduzimanja oduzimaju

- Jedino na šta treba paziti je da na kraju treba odrediti stepen rezultata, jer je moguće da je stepen rezultata manji od stepena sabiraka

- Uvedimo dodatan parametar `op` (operacija) koji može biti `PLUS` ili `MINUS`, što su konstante nabrojivog tipa `Znak`:

```
enum Znak {
   PLUS, MINUS
}
```

Polinom - statička implementacija

```
private static Polinom sab(Polinom p1, Polinom p2, Znak op) {
  if (p1 == null || p2 == null)
    return null;
  Polinom zbir = new Polinom();
  if (p1.st > p2.st)
    zbir.st = p1.st;
  else
    zbir.st = p2.st;
  if (op == Znak.PLUS)
    for (int i = 0; i <= zbir.st; i++)
      zbir.k[i] = p1.k[i] + p2.k[i];
  else
    for (int i = 0; i <= zbir.st; i++)
      zbir.k[i] = p1.k[i] - p2.k[i];
  nadjiStepen(zbir);
  return zbir;
}
```

Polinom - statička implementacija

```
/* Sabira polinome p1 i p2 vracajuci zbir */
static Polinom saberi(Polinom p1, Polinom p2) {
  return sab(p1, p2, Znak.PLUS);
}

/* Oduzima polinom p2 od polinoma p1 vracajuci razliku */
static Polinom oduzmi(Polinom p1, Polinom p2) {
  return sab(p1, p2, Znak.MINUS);
}
```

Polinom - statička implementacija

```
/* Mnozi broj c sa polinomom p vracajuci proizvod */
static Polinom brojPuta(Polinom p, double c) {
  if (p == null)
    return null;
  Polinom rezultat = new Polinom();
  if (p.st != -1 && c != 0.0) {
    rezultat.st = p.st;
    for (int i = 0; i <= p.st; i++)
      rezultat.k[i] = c * p.k[i];
  }
  return rezultat;
}
```

Polinom - statička implementacija

- Množenje se može realizovati na sličan način, samo što se sada množi svaki monom jednog polinoma sa svakim monomom drugog polinoma
- Treba množiti polinome ako su oba činioca različita od nule, a ako je bar jedan činilac nula-polinom, tada je množenje završeno
- Ukoliko je stepen proizvoda veći od maksimalnog dozvoljenog stepena, metod vraća `null`
- Množenje dva monoma se realizuje tako što se proizvod koeficijenata ta dva monoma doda na koeficijent monoma u proizvodu čiji je stepen jednak zbiru stepena monoma koji se množe

Polinom - statička implementacija

```
/* Mnozi polinom p1 sa p2 vracajuci proizvod */
static Polinom puta(Polinom p1, Polinom p2) {
  if (p1 == null || p2 == null)
    return null;
  int proizvodSt = p1.st + p2.st;
  if (proizvodSt > Polinom.maxSt)
    return null;
  Polinom proizvod = new Polinom();
  if (p1.st != -1 && p2.st != -1) {
    proizvod.st = proizvodSt;
    for (int i = 0; i <= p1.st; i++)
      for (int j = 0; j <= p2.st; j++)
        proizvod.k[i+j] += p1.k[i] * p2.k[j];
  }
  return proizvod;
}
```

Polinom - statička implementacija

- Deljenje se realizuje tako što se imitira poznati algoritam deljenja polinoma “peške”

- Ako je delilac jednak nula-polinomu, metod će vratiti `null` i time signalizirati grešku

Polinom - statička implementacija

```
/* Deli dva polinoma, vracajuci kolicnik i ostatak u nizu */
static Polinom[] deli(Polinom p1, Polinom p2) {
  if (p1 == null || p2 == null)
    return null;
  if (p2.st == -1)
    return null;
  int i, j, m, l;
  Polinom ostatak = kopiraj(p1);
  Polinom kolicnik = new Polinom();
  if (p1.st >= p2.st) {
    kolicnik.st = p1.st - p2.st;
    for (j = p1.st; j >= p2.st; j--) {
      i = j - p2.st;
      kolicnik.k[i] = ostatak.k[j] / p2.k[p2.st];
      for (m = 0; m <= p2.st; m++) {
        l = m + i;
        ostatak.k[l] -= kolicnik.k[i] * p2.k[m];
      }
    }
    nadjiStepen(ostatak);
  }
  Polinom[] rezultat = {kolicnik, ostatak};
  return rezultat;
}
```

Polinom - OO pristup

- Statička implementacija polinoma koju smo predstavili urađena je u proceduralnom stilu, gde su podaci (polja klase `Polinom`) odvojeni od operacija (metodi klase `PolinomN`)
- Uz male modifikacije implementacija može da se organizuje u objektno-orijentisanom stilu:
  - Metodi pripadaju klasi `Polinom`, i nisu statički
  - Umesto da se svi polinomi prosleđuju metodima kao parametri, poziv metoda odnosi se na tekući objekat preko koga se metod poziva, tako da se (prvi) parametar metoda tipa `Polinom` izostavlja
    - Primeri:
      `PolinomN.anuliraj(p)` postaje `p.anuliraj()`
      `r = PolinomN.saberi(p, q)` postaje `r = p.saberi(q)`
- Java prvenstveno podržava OO stil, tako da je u Javi ovaj pristup implementaciji prirodniji

Polinom - OO pristup

```
class Polinom {
  static final int maxSt = 100;
  double[] k = new double[maxSt+1];
  int st = -1;

  /* Anulira polinom */
  void anuliraj() {
    st = -1;
    for (int i = 0; i <= maxSt; i++)
      k[i] = 0.0;
  }
```

Polinom - OO pristup

```
/* Kreira i vraca kopiju polinoma */
Polinom kopiraj() {
  Polinom q = new Polinom();
  q.st = st;
  for (int i = 0; i <= maxSt; i++)
    q.k[i] = k[i];
  return q;
}

/* Pronalazi stepen polinoma i smesta ga u strukturu */
void nadjiStepen() {
  final double eps = 1.0E-5;
  st = maxSt;
  while (st > -1 && (Math.abs(k[st]-0.0) < eps)) {
    st--;
  }
}
```

Polinom - OO pristup

```
/* Izracunava vrednost polinoma za dato x */
double izracunaj(double x) {
  double rezultat;
  if (st == -1)
    rezultat = 0.0;
  else {
    rezultat = k[st];
    for (int i = st - 1; i >= 0; i--)
      rezultat = rezultat * x + k[i];
  }
  return rezultat;
}
```

- Metod `ucitaj` ostavićemo da bude statički, jer prirodno omogućava kreiranje i inicijalizaciju polinoma u okviru jednog poziva metoda:

```
/* Ucitava polinom (napomena: static metod)*/
static Polinom ucitaj() { ... }
```

Polinom - OO pristup

- Štampanje polinoma realizovaćemo uz pomoć metoda `toString` nasleđenog iz klase `Object`

- Iskoristićemo Javinu klasu `StringBuilder`, koja predstavlja stringove koji mogu da se modifikuju. Nama će biti potreban samo metod `append` koji spaja tekući string sa stringom prosleđenim kao parametar

- Pored deklaracije i inicijalizacije objekta `s` klase `StringBuilder`, jedina razlika u odnosu na stari metod `stampaj` je da umesto `System.out.print` pišemo `s.append`, i na kraju vratimo `s.toString()`

Polinom - OO pristup

```
public String toString() {
  StringBuilder s = new StringBuilder();
  if (st > 0) { /* ako polinom ima x stampati: */
    if (k[st] < 0.0) /* najpre vodeci monom */
      s.append("-");
    if (Math.abs(k[st]) != 1.0)
      s.append(Math.abs(k[st]));
    if (st > 1)
      s.append("x^" + st);
    else
      s.append("x");
```

Polinom - OO pristup

```
/* stampati ostale monome sa x */
for (int i = st - 1; i >= 1; i--) {
  if (k[i] != 0.0) {
    if (k[i] > 0.0)
      s.append("+");
    else
      s.append("-");
    if (Math.abs(k[i]) != 1.0)
      s.append(Math.abs(k[i]));
    if (i > 1)
      s.append("x^" + i);
    else if (i == 1)
      s.append("x");
  }
}
```

Polinom - OO pristup

```
      /* na kraju stampati slobodan clan */
      if (k[0] != 0.0) {
        if (k[0] > 0.0)
          s.append("+" + k[0]);
        else
          s.append("-" + Math.abs(k[0]));
      }
    }
    else { /* ako polinom nema x */
      if (k[0] < 0.0)
        s.append("-" + Math.abs(k[0]));
      else
        s.append(k[0]);
    }
    return s.toString();
  }
```

Polinom - OO pristup

- Sad kad imamo metod `toString`, implementacija metoda `stampaj` je trivijalna
- Prosleđivanjem `this` kao parametra metodi `System.out.print` implicitno će biti pozvan metod `this.toString()` tekućeg objekta, i rezultat će biti odštampan

```
/* Stampa polinom */
void stampaj() {
  System.out.print(this);
}
```

Polinom - OO pristup

- Nabrojivi tip `Znak` i pomoćni metod `sab` ostaju isti, samo ćemo nabrojivi tip staviti da bude privatan član klase `Polinom`:

```
private enum Znak {
  PLUS, MINUS
}

/* Sabira polinom sa polinomom p vracajuci zbir */
Polinom saberi(Polinom p) {
  return sab(this, p, Znak.PLUS);
}

/* Oduzima polinom p od polinoma vracajuci razliku */
Polinom oduzmi(Polinom p) {
  return sab(this, p, Znak.MINUS);
}
```

Polinom - OO pristup

```
/* Mnozi broj c sa polinomom vracajuci proizvod */
Polinom brojPuta(double c) {
  Polinom rezultat = new Polinom();
  if (st != -1 && c != 0.0) {
    rezultat.st = st;
    for (int i = 0; i <= st; i++)
      rezultat.k[i] = c * k[i];
  }
  return rezultat;
}
```

Polinom - OO pristup

```
/* Mnozi polinom polinomom p vracajuci proizvod */
Polinom puta(Polinom p) {
  if (p == null)
    return null;
  int proizvodSt = st + p.st;
  if (proizvodSt > maxSt)
    return null;
  Polinom proizvod = new Polinom();
  if (st != -1 && p.st != -1) {
    proizvod.st = proizvodSt;
    for (int i = 0; i <= st; i++)
      for (int j = 0; j <= p.st; j++)
        proizvod.k[i+j] += k[i] * p.k[j];
  }
  return proizvod;
}
```

Polinom - OO pristup

```
  /* Deli polinom polinomom p, vracajuci kolicnik i ostatak u nizu */
  Polinom[] deli(Polinom p) {
    if (p == null)
      return null;
    if (p.st == -1)
      return null;
    int i, j, m, l;
    Polinom ostatak = kopiraj();
    Polinom kolicnik = new Polinom();
    if (st >= p.st) {
      kolicnik.st = st - p.st;
      for (j = st; j >= p.st; j--) {
        i = j - p.st;
        kolicnik.k[i] = ostatak.k[j] / p.k[p.st];
        for (m = 0; m <= p.st; m++) {
          l = m + i;
          ostatak.k[l] -= kolicnik.k[i] * p.k[m];
        }
      }
      ostatak.nadjiStepen();
    }
    Polinom[] rezultat = {kolicnik, ostatak};
    return rezultat;
  }
} // Kraj klase Polinom
```