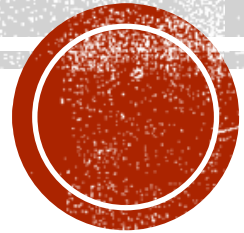
ОБЈЕКТНО ОРИЈЕНТИСАНО ПРОГРАМИРАЊЕ
ПРОГРАМСКИ ЈЕЗИК ЈАВА – 1

Неке програмске парадигме



ПРОГРАМСКА ПАРАДИГМА

- Фундаментални стил програмирања, који се користи, назива се **програмска парадигма.**
- Иако је у центру проучавања овог уџбеника ООП, биће кратко приказане и друге:
  - императивне (структурна, модуларна)
  - и декларативне (логичка, функционална).
- Решавајући исти задатак коришћењем различитих парадигми,
  - биће демостриране неке карактеристике тих парадигми
  - и приказано како се те парадигме могу користити у програмском језику Јава.

ЗАДАТАК

- **Написати Јава програм који одређује највећи заједнички делилац (НЗД) за три броја.**

- У свим решењима, која ћемо надаље навести, кључну улогу има:
  - начин на који се одређује НЗД за два броја (природни бројеви).
- Када се одреди НЗД за два броја,
  одређивање НЗД за три броја је мање-више исто у свим парадигмама.

ИМПЕРАТИВНА ПАРАДИГМА

- Основни појам на којем се заснива императивна парадигма је **команда** или **наредба.**
- Битан је и редослед команди.
- Груписање команди у тзв. **процедуре** (процедурално програмирање).
  - По неким ауторима је императивно = процедурално.
  - **По другима је процедурално варијанта императивног.**

ИМПЕРАТИВНА РЕШЕЊЕ

- Програм ћемо реализовати као монолитну целину.
- НЗД за три броја се одређује тако што се:
  - прво одреди НЗД за прва два броја,
  - а коначни резултат тако што се одреди НЗД за претходна два и за трећи број.
- Прво одређивање НЗД је реализовано коришћењем Еуклидовог алгоритма:
  - НЗД два различита броја исти као НЗД мањег од та два броја и њихове разлике.
- Друго одређивање НЗД је коришћењем модификованог Еуклидовог алгоритма:
  - уместо разлике рачуна остатак при дељењу — чиме се постиже бржа конвергенција.
- С обзиром да програмски језик Јава не подржава **goto** наредбу, уместо те наредба је коришћена обележена **break** наредба.

ИМПЕРАТИВНА РЕШЕЊЕ (2)

```
// одређивање НЗД за први и други број
nzdPrviDrugi:
for (; ; ) {
    if (prviBroj == drugiBroj)
        break nzdPrviDrugi;
    // размени бројеве тако да други број буде
        //већи од првог
    if (prviBroj > drugiBroj) {
        int privremeni = prviBroj;
        prviBroj = drugiBroj;
        drugiBroj = privremeni;
    }
    // нови пар бројева су дотадашњи мањи
    //и разлика између већег и мањег
    drugiBroj = drugiBroj - prviBroj;

}
```

```
nzdNadPrvaDvaTreci:
for (; ; ) {
      // ако су бројеви исти, НЗД је ма који од њих
      if (prviBroj == treciBroj)
          break nzdNadPrvaDvaTreci;
      // размени бројеве тако да трећи број буде већи од првог
      if (prviBroj > treciBroj) {
          int privremeni = prviBroj;
          prviBroj = treciBroj;
          treciBroj = privremeni;
      }
      // ако мањи број дели већи број, тада је мањи број НЗД
      if( treciBroj % prviBroj == 0 )
          break nzdNadPrvaDvaTreci;
      // "преживљавају" мањи од два броја и остатак при дељењу
          treciBroj = treciBroj % prviBroj;
}
```

СТРУКТУРНА ПАРАДИГМА

- Настало из процедуралног ограничавањем или потпуном елиминацијом goto.
- Дозвољена употреба ограниченог броја управљачких структура које побољшавају:
  - јасноћу и прегледност програма,
  - време развоја и одржавања (смањују).
- Потпарадигма процедуралне.

- Рачунање НЗД за два и за три броја је издвојено у посебне методе:
  - програмски кôд прегледнији,
  - читљивији
  - и лакши за одржавање и надоградњу.

СТРУКТУРНА РЕШЕЊЕ

```
    // одређивање НЗД за два броја
    static int nzd2(int prvi, int drugi) {
        while (true)
            if (prvi > drugi) {
                // први је већи
                if (prvi % drugi == 0)
                    return drugi;
                prvi %= drugi;
            } else {
                // први је мањи или једнак
                if (drugi % prvi == 0)
                    return prvi;
                drugi %= prvi;
            }
    }
    // одређивање НЗД за три броја
    static int nzd3(int prvi, int drugi, int treci) {
        return nzd2(nzd2(prvi, drugi), treci);
    }

// улазна тачка програма
    public static void main(String[] args) {
        …
```

МОДУЛАРНА ПАРАДИГМА

- Потпарадигма процедуралне.
- Модуларна усваја све добре стране структурне и заснива се на:
  - интензивном коришћењу мањих програмских целина — модула.
  - Модули се користе за апстракцију (уопштавање) појма податак.

- Једина разлика при решавању задатка је у одвајању засебног модула за покретање.

МОДУЛАРНА РЕШЕЊЕ

```
class ModulNzd {
    // одређивање НЗД за два броја
    static int nzd2(int prvi, int drugi) {
        while (true)
            if (prvi > drugi) {
                // први је већи
                if (prvi % drugi == 0)
                    return drugi;
                prvi %= drugi;
            } else {
                // први је мањи или једнак
                if (drugi % prvi == 0)
                    return prvi;
                drugi %= prvi;
            }
    }
    // одређивање НЗД за три броја
    ...
}
```

```
class ModulNzdPokretanje {
    // улазна тачка програма
    public static void main(String[] args) {
        // бројеви чији се НЗД тражи
        int prviBroj = 48;
        int drugiBroj = 120;
        int treciBroj = 56;
        …
        // одређивање НЗД за ова три броја
        int nzd = ModulNzd.nzd3(prviBroj, drugiBroj,
            treciBroj);
        …
    }
}
```

ДЕКЛАРАТИВНА ПАРАДИГМА

- У претходним парадигмама рачунару се саопштава **како** се проблем решава.
- Да ли је могуће, преко програма, описати (декларисати) проблем и на основу тога рачунар да генерише решење?
  - Овакав приступ подржан је преко тзв. **декларативних (описних)** парадигми.
- У декларативне спадају (између осталих):
  - функционална
  - и логичка.
- Понављање операција:
  - код императивних путем итерација,
  - док је код деклартивних парадигми путем рекурзије.

ЛОГИЧКА ПАРАДИГМА

- Заснована на аутоматском доказивању теорема.
- Основни појмови логичке парадигме су:
  - аксиоме (чињенице),
  - правила закључивања
  - и упити.
- Програм се реализује постављањем упита.
- При том се аутоматски:
  - претражује скуп чињеница
  - уз коришћење одређених правила закључивања.
- Због овог специфичног начина решавања проблема, логичку парадигму је најтеже реализовати у Јави.

ЛОГИЧКА РЕШЕЊЕ

- Користићемо рекурзивну верзију модификованог Еуклидовог алгоритма.
- Правила из логичког програмирања се симулирају преко метода.
- Симулирани скуп правила закључивања би могао да изгледа овако:

$$
\begin{aligned}
&m \bmod n = 0 \Rightarrow nzd(m,n) = n \\
&n \bmod m = 0 \Rightarrow nzd(m,n) = m \\
&m > n \Rightarrow nzd(m,n) = nzd(m \bmod n,\ n) \\
&m < n \Rightarrow nzd(m,n) = nzd(m,\ n \bmod m).
\end{aligned}
$$

- Напомена: ово је само симулација, логички језици користе другачији систем за резоновање, нпр. метод резолуције.

ЛОГИЧКА РЕШЕЊЕ (2)

```
// одређивање НЗД за два броја
   static int nzd2(int prvi, int drugi) {
       if (prvi % drugi == 0)
           return prvi;
       if (drugi % prvi == 0)
           return drugi;
       if (prvi > drugi)
           return nzd2(prvi % drugi, drugi);
       return nzd2(prvi, drugi % prvi);
   }
   // одређивање НЗД за три броја
   static int nzd3(int prvi, int drugi, int treci) {
       return nzd2(nzd2(prvi, drugi), treci);
   }
```

ФУНКЦИОНАЛНА ПАРАДИГМА

- Функционална парадигма је заснована на математичком појму “функција”.
- Може се окарактерисати реченицом:
  “Израчунај вредност израза и користи је за нешто”.
- Сва израчунавања у програму испуњавају се применом (позивом) функција.
- У чистим функционалним језицима:
  - не постоји концепт промењљиве па самим тим ни доделе,
  - константе су такође функције (са 0 аргумената),
  - није могућ домино-ефекат (енг. **side effect**), јер нема доделе.

ФУНКЦИОНАЛНА РЕШЕЊЕ

- Користићемо рекурзвну дефиицију НЗД за два природна броја, која гласи:

$$nzd(n,m) = n, \qquad za\ m = n$$
$$nzd(n,m) = nzd(m-n,\ n), \quad za\ m > n$$
$$nzd(n,m) = nzd(n-m,\ m), \quad za\ m < n$$

ФУНКЦИОНАЛНА РЕШЕЊЕ (2)

```
// одређивање НЗД за два броја
static BiFunction<Integer, Integer, Integer> nzd2;
static {
    nzd2 = (prvi, drugi) -> (prvi.compareTo(drugi) == 0)
            ? prvi
            : ( (prvi.compareTo(drugi) < 0)
                    ? nzd2.apply(drugi - prvi, prvi)
                    : nzd2.apply(prvi - drugi, drugi) );
}

// улазна тачка програма
public static void main(String[] args) {
    // одређивање НЗД за ова три броја
    int nzd = nzd2.apply(prviBroj, nzd2.apply(drugiBroj, treciBroj));
}
```

ПИТАЊА И ЗАДАЦИ

- Упоредити императивну и декларативну парадигму.
- Написати програм који испитује да ли су два броја узајамно проста:
  - императивном парадигмом;
  - структурном парадигмом;
  - модуларном парадигмом,
  - логичком парадигмом;
  - функционалном парадигмом.
- Написати програм који имплементира quick sort алгоритам у:
  - императивној парадигми;
  - функционалној парадигми.
- За сваку од програмских парадигми, разматраних у овом поглављу, наћи примере програмских језика који се зансивају на њој.
- Истражити које још програмске парадигме, поред наведених, постоје.