# Promenljive:
# sporedni efekti, vidljivost i život

# Sporedni efekti

- Posmatrajmo statičke metode, za koje smo rekli da su jedinstvene celine za ceo program (klasu)
- Za “komunikaciju” metoda sa spoljašnjim okruženjem postoje tri mogućnosti:
  1. Sve vrednosti se u metod unose i sve vrednosti se iz metoda iznose posredstvom “spoljašnjih” promenljivih (npr. statičkih polja klase)
  2. Sve vrednosti se u metod unose i iz njega iznose posredstvom parametara i (eventualno) povratne vrednosti (za “iznošenje” vrednosti)
  3. Kombinacijom prethodna dva načina
- Metodi koji koriste 2. način su logičke celine koje ne zavise od globalnih vrednosti i trenutnog stanja okruženja programa - njihovo ponašanje je u potpunosti određeno formalnim parametrima

# Sporedni efekti

- Pored toga, takvi statički metodi se lako i bez izmena mogu koristiti i u drugim programima
- Kad god je to moguće treba koristiti upravo ovakve statičke metode, jer su opštiji i "prenosiviji", a programi su čitljiviji, jasniji i omogućavaju lakše otkrivanje grešaka
- Rekli smo da nije dobro ako ponašanje metoda zavisi od vrednosti "spoljašnjih" promenljivih
- Još je manje dobro ako metodi menjaju vrednosti spoljašnjih promenljivih, jer se tako stvaraju programi čije se izvršavanje teško prati
- Menjanje spoljašnjih promenljivih iz metoda se često naziva sporednim efektom (engl. *side effect*) i generalno se smatra lošom praksom u programiranju

# Sporedni efekti

- Menjanje spoljašnjih promenljivih iz statičkih metoda može često biti uzrok grešaka u radu programa koje se vrlo teško otkrivaju
- Prirodno je, bar za matematičare, da (na primer) izraz `f(x)+g(x)` ima istu vrednost kao i izraz `g(x)+f(x)`, jer je + komutativna operacija
- Međutim, kada se programira uz pomoć sporednih efekata, to više ne mora biti tako
- U primeru koji sledi, kao rezultat poziva `f(vred)+g(vred)` i `g(vred)+f(vred)` dobiće se različite vrednosti, bez obzira što su polazna okruženja u kojima su se izračunavanja vršila bila ista
- Primer je jednostavan i pomalo veštački, ali je čak i u njemu teško pratiti šta se i kako menja
- Ako je program duži i promene promenljivih nesmotrene, kasnije menjanje programa će biti otežano, a greške nastale usled sporednih efekata teško će se pronaći

# Sporedni efekti: primer

```
class SideEffect {
  static int vred;
  static int f(int x) {
    vred += x;
    return vred;
  }
  static int g(int x) {
    return vred + x;
  }
  public static void main(String[] args) {
    vred = 1;
    System.out.println("f(vred) + g(vred) = " +
                       (f(vred) + g(vred)));
    vred = 1;
    System.out.println("g(vred) + f(vred) = " +
                       (g(vred) + f(vred)));
  }
}
```

- Izlaz:

```
f(vred) + g(vred) = 6
g(vred) + f(vred) = 4
```

# Sporedni efekti

- Kod nestatičkih metoda važi sličan princip
- Razlika je u tome što u tom slučaju “jedinstveni entitet” ne treba da bude sam metod kao kod statičkih metoda, već objekat kome nestatički metod pripada
- Tako da je dozvoljeno i poželjno da nestatički metod zavisi od i utiče na vrednosti polja objekta kom pripada
- Međutim, zavisnost od i uticaj na bilo koje promeljive “spoljašnjije” od polja istog objekta nisu poželjni

# Vidljivost imena i život promenljivih

- Oblast dejstva, doseg ili vidljivost imena (engl. *scope*) jesu sinonimi koji označavaju delove programa u kome važi deklaracija imena, tj. u kome ime može da se koristi (u kome se “vidi”)
- Život (ili trajanje) promenljive (engl. *extent*) je deo programa u kome promenljiva fizički postoji (“živi”), bez obzira da li je trenutno vidljiva ili ne
- U Javi, osnovni mehanizmi za određivanje vidljivosti imena i života promenljivih su paketi, klase (odnosno ref. tipovi), metodi i blokovi
- Paketi utiču samo na vidljivost imena referencijalnih tipova
- Mi ćemo se fokusirati na imena promenljivih, i njihov život

# Vidljivost imena i život promenljivih

- Na život promenljive mogu uticati dva mehanizma:
  - Blokovi (uključujući tela metoda)
  - Kreiranje objekata, odnosno postojanje reference na objekat kreiran u memoriji (heap)
- Lokalne promenljive žive i vidljive su u bloku u kom su deklarisane, i to od deklaracije do kraja bloka
  - Memorijski prostor za vrednost lokalne promenljive kreira se prilikom deklaracije, a po izlasku iz bloka se oslobađa
  - Pri tom, ako je lokalna promenljiva referencijalnog tipa, ovo važi samo za vrednost promenljive što je referenca - kreiran objekat na koji referenca pokazuje može da živi "posebnim životom"
- Argumenti metoda žive i vidljivi su u telu metoda (bloku) do njegovog kraja

# Vidljivost imena i život promenljivih

- Do kraja diskusije pod pojmom “klasa” podrazumevamo “referencijalni tip”
- Statička polja klase žive celim tokom izvršavanja programa koji koristi klasu
- Na vidljivost statičkih polja utiču modifikatori vidljivosti:
  - Polja `public` vidljivosti su vidljiva svuda (gde je vidljiva i klasa)
  - Polja `protected` vidljivosti su vidljiva u svim klasama iz istog paketa i svim podklasama klase
  - Polja *default* vidljivosti su vidljiva u svim klasama iz istog paketa
  - Polja `private` vidljivosti su vidljiva samo u okviru svoje klase

| Modifikator | Klasa | Paket | Podklasa | Sve |
|---|---|---|---|---|
| `public` | DA | DA | DA | DA |
| `protected` | DA | DA | DA | NE |
| *default* | DA | DA | NE | NE |
| `private` | DA | NE | NE | NE |

# Vidljivost imena i život promenljivih

- Nestatička polja klase žive dokle god živi objekat kom pripadaju
  - Objekat živi ako je kreiran (operatorom `new` i sl.) i na njega pokazuje bar jedna referenca u programu (tj. živi bar jedna promenljiva čija je vrednost referenca na taj objekat)

- Za vidljivost nestatičkog polja objekta neophodan preduslov je da je moguće doći do reference na taj objekat
  - Tada za vidljivost nestatičkog polja važe ista pravila kao za statička polja

# Vidljivost imena i život promenljivih

- Ključna reč `this` može da se koristi u svakom nestatičkom kontekstu (konstruktori, nestatički metodi, inicijalizacija nestatičkih polja), gde označava referencu na tekući objekat

- Dozvoljeno je da formalni parametri metoda i konstruktora, kao i lokalne promenljive u datoj klasi imaju isto ime kao i polja, čime je ime polja u metodu/konstruktoru/bloku redefinisano

- U takvim slučajevima i dalje je moguće pristupiti poljima klase
  - Za nestatička polja: navođenjem `this`, tačke, i imena polja
  - Za statička polja:
    - Isto kao za nestatička polja, ili
    - Navođenjem imena klase, tačke, i imena polja

# Vidljivost imena i život promenljivih

```
class Tacka {
  static final double JEDAN = 1.0;
  double x, y;
  Tacka(double x, double y) {
    this.x = x; this.y = y;
  }
  double testX() {
    int x = 2;
    final int JEDAN = -1;
    return this.x * x + Tacka.JEDAN + this.JEDAN;
  }
  public String toString() {
    return "(" + this.x + ", " + this.y + ")";
  }
}
class ThisTest {
  public static void main(String[] args) {
    Tacka t = new Tacka(1.0, 2.0);
    double x = t.testX();
    System.out.println("t.testX() = " + x); // 4.0
  }
}
```