

Programiranje 1

Poglavje 4: Metode

Luka Fürst

Primer

- program, ki izpiše sledeči vzorec

```
  +  
  +  
  +  
  +  
++++++  
  +  
  +  
  +  
  +
```

Rešitev

```
for (int i = 1; i <= 4; i++) {  
    System.out.println("    +");  
}
```

```
for (int i = 1; i <= 9; i++) {  
    System.out.print("+");  
}  
System.out.println();
```

```
for (int i = 1; i <= 4; i++) {  
    System.out.println("    +");  
}
```

- ponavljanje smrdi!
- načelo DRY (angl. don't repeat yourself)

Metoda

- poimenovan kos kode, ki ga lahko poljubno mnogokrat izvršimo (**pokličemo**)
- sorodni koncepti v drugih jezikih
 - funkcija, procedura, podprogram ...
- izvajanje programa se prične v metodi **main**
- metoda **main** lahko kliče druge metode, te spet lahko kličejo metode itd.
- kdaj uporabimo metodo?
 - ponavljanje istih opravil
 - smiselno zaokrožen podproblem

Rešitev s pomožno metodo

```
public static void main(String[] args) {  
    navpicniKрак();    // klic metode  
  
    for (int i = 1; i <= 9; i++) {  
        System.out.print("+");  
    }  
    System.out.println();  
  
    navpicniKрак();    // klic metode  
}  
  
public static void navpicniKрак() { // definicija metode  
    for (int i = 1; i <= 4; i++) {  
        System.out.println("    +");  
    }  
}
```

Klic in vračanje

```
public static void metoda1() {
```

```
    ...
```

```
    ...
```

```
    metoda2();
```

```
    ...
```

```
    ...
```

```
}
```

```
public static void metoda2() {
```

```
    ...
```

```
    ...
```

```
    ...
```

```
    ...
```

```
    ...
```

```
}
```

①

②

③

Križ s krakom dolžine n

```
for (int i = 1; i <= n; i++) {  
    for (int j = 1; j <= n; j++) {  
        System.out.print(" ");  
    }  
    System.out.println("+");  
}  
  
for (int i = 1; i <= 2 * n + 1; i++) {  
    System.out.print("+");  
}  
System.out.println();  
  
for (int i = 1; i <= n; i++) {  
    for (int j = 1; j <= n; j++) {  
        System.out.print(" ");  
    }  
    System.out.println("+");  
}
```

Poskus rešitve z metodami

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    navpicniKraK();
    vodoravniKraK();
    navpicniKraK();
}

public static void navpicniKraK() {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            System.out.print(" ");
        }
        System.out.println("+");
    }
}

public static void vodoravniKraK() {
    for (int i = 1; i <= 2 * n + 1; i++) {
        System.out.print("+");
    }
    System.out.println();
}
```


Parametri metode

- spremenljivka `n` v metodah `navpicniKra` in `vodoravniKra` ni vidna!
- spremenljivka obstaja samo znotraj bloka, v katerem je deklarirana
- metode si lahko podatke prenašajo prek [parametrov](#)

Rešitev s parametri

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
    int n = sc.nextInt();    // npr. 5  
    navpicniKrak(n);  
    vodoravniKrak(n);  
    navpicniKrak(n);  
}  
  
public static void navpicniKrak(int visina) {    visina = 5  
    for (int i = 1; i <= visina; i++) {  
        for (int j = 1; j <= visina; j++) {  
            System.out.print(" ");  
        }  
        System.out.println("+");  
    }  
}  
  
public static void vodoravniKrak(int n) {    n = 5  
    for (int i = 1; i <= 2 * n + 1; i++) {  
        System.out.print("+");  
    }  
    System.out.println();  
}
```

The diagram illustrates the flow of parameters in the provided Java code. A red line originates from the `n` argument in the first `navpicniKrak(n);` call within the `main` method and points to the `visina` parameter in the `navpicniKrak` method signature. An orange line originates from the `n` argument in the `vodoravniKrak(n);` call within the `main` method and points to the `n` parameter in the `vodoravniKrak` method signature. Additionally, a blue box labeled `visina = 5` is positioned next to the `navpicniKrak` method signature, and another blue box labeled `n = 5` is positioned next to the `vodoravniKrak` method signature, indicating the values passed to these methods.

Parametri in argumenti

```
public static void metoda( $T_1$   $p_1$ ,  $T_2$   $p_2$ , ...,  $T_n$   $p_n$ ) {  
    ...  
}  
  
public static void ...(...) {  
    ...  
    metoda( $a_1$ ,  $a_2$ , ...,  $a_n$ );  
    ...  
}
```

- p_1, p_2, \dots, p_n : parametri (ali formalni parametri)
- a_1, a_2, \dots, a_n : argumenti (ali dejanski parametri)

Argumenti in parametri

- vrednosti argumentov se skopirajo v istoležne parametre

$$p_1 = a_1$$

$$p_2 = a_2$$

...

$$p_n = a_n$$

- tip argumenta a_i mora biti **priredljiv** tipu parametra p_i
 - p_i je tipa `int` \implies
 a_i je lahko tipa `int`, `short`, `byte` ali `char`, ne pa `long`

Argumenti in parametri

- parametri metode
 - so povsem ločeni od argumentov
 - od argumentov dobijo samo začetne vrednosti
 - obstajajo samo znotraj metode
 - obnašajo se kot spremenljivke v metodi

```
public static void main(String[] args) {  
    int a = 42;  
    metoda(a);  
    System.out.println(a);    // 42  
}  
  
public static void metoda(int a) {  
    a++;  
    System.out.println(a);    // 43  
}
```

Metoda z več parametri

- metoda, ki n -krat izpiše podani znak in po želji doda še prelom vrstice
- parametri
 - `n`: število ponovitev
 - `znak`: znak, ki naj se izpiše
 - `prelom`: `true` \implies dodaj prelom vrstice

```
public static void zaporedje(int n, char znak,
                             boolean prelom) {
    for (int i = 1; i <= n; i++) {
        System.out.print(znak);
    }
    if (prelom) {
        System.out.println();
    }
}
```

Metoda z več parametri

- preostanek programa se precej poenostavi

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    navpicniKraK(n);
    vodoravniKraK(n);
    navpicniKraK(n);
}

public static void navpicniKraK(int visina) {
    for (int i = 1; i <= visina; i++) {
        zaporedje(visina, ' ', false);
        zaporedje(1, '+', true);
    }
}

public static void vodoravniKraK(int n) {
    zaporedje(2 * n + 1, '+', true);
}
```

Vračanje vrednosti

- metoda, ki izpiše obseg pravokotnika

```
public static void obseg(int a, int b) {  
    System.out.println(2 * (a + b));  
}
```

- rezultata metode ne moremo več uporabiti

```
public static void main(String[] args) {  
    obseg(3, 4); // izpiše 14  
    // vrednosti 14 ne moremo več uporabiti ...  
}
```


Vračanje vrednosti

- metoda lahko klicatelju vrne vrednost
 - void v glavi nadomestimo s tipom vrnjene vrednosti (T)
 - vrednost vrnemo s stavkom
`return izraz;`
 - tip vrnjene vrednosti mora biti priredljiv tipu T

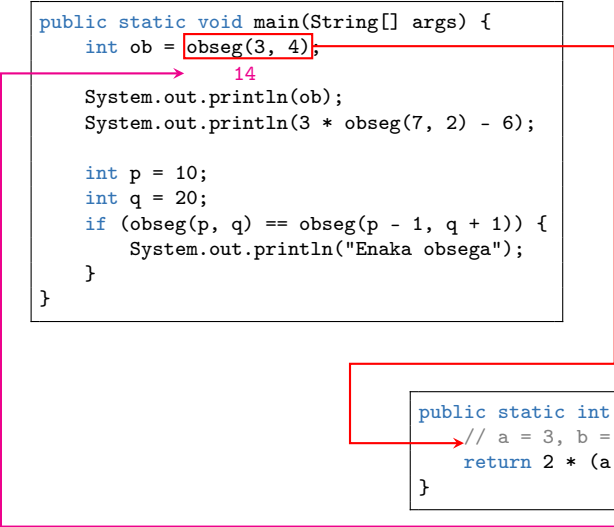
Stavek return

- takoj zaključi metodo
- nadomesti klic metode z vrnjeno vrednostjo
- če metoda ni deklarirana kot void, se mora **v vseh možnih scenarijih** zaključiti s stavkom
`return vrednost;`
- v metodi, deklarirani kot void, lahko uporabimo return brez vrednosti

```
public static void zaporedje(int n, char znak,  
                             boolean prelom) {  
    if (n <= 0) {  
        return;    // takoj zaključi metodo  
    }  
    ...  
}
```

Vračanje vrednosti

```
public static void main(String[] args) {  
    int ob = obseg(3, 4);  
    System.out.println(ob);  
    System.out.println(3 * obseg(7, 2) - 6);  
  
    int p = 10;  
    int q = 20;  
    if (obseg(p, q) == obseg(p - 1, q + 1)) {  
        System.out.println("Enaka obsega");  
    }  
}
```



```
public static int obseg(int a, int b) {  
    // a = 3, b = 4  
    return 2 * (a + b);  
}
```

Praštevila

- program, ki izpiše vsa praštevila med 2 in vključno n
- preverjanje, ali je podani kandidat praštevilo, je **smiselno zaokrožen podproblem**
 - zasluži si svojo metodo!
- metoda main

```
public static void main(String[] args) {  
    System.out.println(2);  
    for (int kandidat = 3; kandidat <= n; kandidat += 2) {  
        if (jePrastevilo(kandidat)) {  
            System.out.println(kandidat);  
        }  
    }  
}
```

Praštevila

- metoda `public static boolean jePrastevilo(int n)`
- preverjamo deljivost števila n z $d \in \{3, 5, \dots, \sqrt{n}\}$
- če odkrijemo d , ki deli n , lahko takoj vrnemo `false`
- na koncu vrnemo `true`

```
public static boolean jePrastevilo(int n) {  
    int meja = (int) Math.round(Math.sqrt(n));  
    for (int d = 3; d <= meja; d += 2) {  
        if (n % d == 0) {  
            return false;  
        }  
    }  
    return true;  
}
```

Prijateljska števila

- naj bo $S(n)$ vsota vseh deliteljev števila n brez števila n samega
- števili a in b sta **prijateljski**, če velja
 - (1) $a \neq b$
 - (2) $S(a) = b$
 - (3) $S(b) = a$
- števili 220 in 284 sta prijateljski
 - (1) $220 \neq 284$
 - (2) $S(220) = 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284$
 - (3) $S(284) = 1 + 2 + 4 + 71 + 142 = 220$

Prijateljska števila

- metoda, ki vrne true natanko tedaj, ko ima podano število prijatelja
- edini kandidat za prijatelja števila n je $k = S(n)$
- kandidat k je prijatelj števila n , če velja
 - $n \neq k$
 - $S(k) = n$
- računanje $S(.)$ je smiselno zaokrožen podproblem

Prijateljska števila

```
public static boolean imaPrijatelja(int n) {  
    int kandidat = vsotaDeliteljev(n);  
    return kandidat != n && vsotaDeliteljev(kandidat) == n;  
}  
  
public static int vsotaDeliteljev(int stevilo) {  
    int vsota = 0;  
    for (int d = 1; d < stevilo; d++) {  
        if (stevilo % d == 0) {  
            vsota += d;  
        }  
    }  
    return vsota;  
}
```


Rekurzija

- metoda lahko kliče samo sebe — **rekurzivni klic**
- rekurzivni klic se izvede popolnoma enako kot običajni klic
 - argumenti se prenesejo v parametre
 - metoda se izvrši
 - klic metode se nadomesti z vrnjeno vrednostjo
- rekurzija omogoča elegantnejšo rešitev nekaterih problemov

Fibonaccijevo zaporedje

- zaporedje F_n za $n \geq 0$

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F_{n-2} + F_{n-1} & \text{sicer} \end{cases}$$

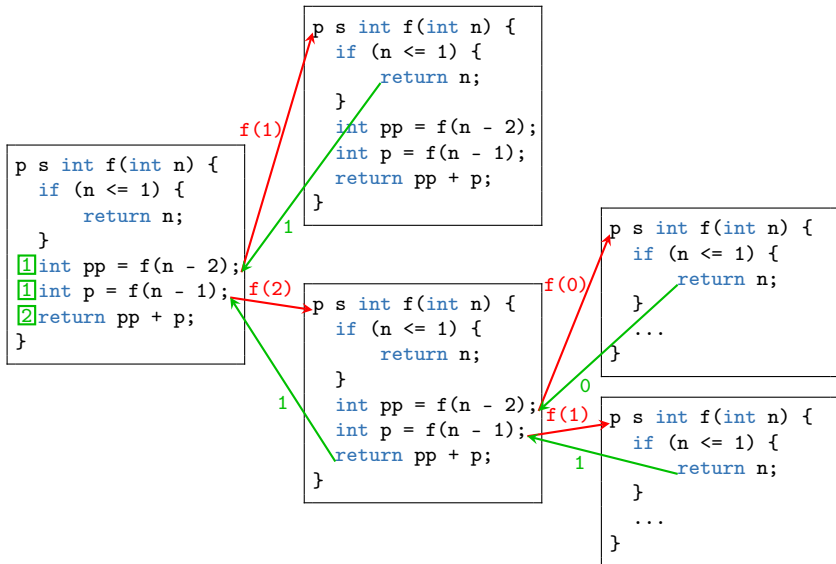
- 0, 1, 1, 2, 3, 5, 8, 13, 21 ...

Fibonaccijevo zaporedje

- metoda, ki vrne Fibonaccijevo število F_n

```
public static int f(int n) {  
    if (n <= 1) {  
        return n;  
    }  
    int pp = f(n - 2);    // predprejšnji člen  
    int p = f(n - 1);     // prejšnji člen  
    return pp + p;  
}
```

Zaporedje klicev za $n == 3$



Drevo rekurzivnih klicev za $n == 5$

