

WHILE naredba

Već je rečeno da se naredbe za ciklično ponavljanje koriste u onim situacijama kada je potrebno izvršiti neku komandu (ili više njih) više puta. FOR petlja je najkorisnija u onim situacijama kada se na početku ciklusa, pre njegovog izvršenja, zna koliko će biti iteracija. Na primer, kada je potrebno ispisati brojeve od 1 do 100 na ekranu, zna se da se to može uraditi u 100 iteracija tako što će se u svakoj iteraciji ispisati po jedan broj. Čak iako broj iteracija zavisi od nekog ulaznog parametra ili promenljive, on će biti poznat u trenutku kada petlja počne. Međutim, postoje i one situacije u kojima nije unapred poznato koliko će se iteracija izvršiti. U takvim situacijama nema smisla koristiti brojač petlje jer uslov za izlazak iz petlje ne zavisi od vrednosti brojača već od nekog drugog uslova. WHILE naredba je nastala upravo da bi mogla biti primenjena u ovakvim slučajevima. Deklaracija WHILE petlje se vrši na sledeći način:

```
while (...uslov...) komanda_p;
```

Deklaracija počinje rezervisanom reči “while” iza koje se u zagradi piše neki logički uslov. Posle logičkog uslova piše se komanda (ili više komandi) koju je potrebno ponoviti više puta (komanda_p). WHILE petlja **ponavlja izvršavanje komande (komanda_p) sve dok uslov iz zagrade važi**. Za razliku od FOR petlje, WHILE petlja **nema brojač**. Izlazak iz petlje ne zavisi od vrednosti brojača, već se dešava kad uslov prestane da važi. Ovaj **uslov je običan logički izraz**, i formira se na potpuno isti način kao bilo koji uslov iz IF naredbe (logički operatori, operatori poređenja i njihove kombinacije). Kada se petlja pokrene, redosled izvršavanja komandi i provere uslova je sledeći:

POČETAK PETLJE

1. ITERACIJA

- Proverava se uslov (uslov važi)
- izvršava se komanda_p

2. ITERACIJA

- Proverava se uslov (uslov važi)
- izvršava se komanda_p

3. ITERACIJA

- Proverava se uslov (uslov važi)
- izvršava se komanda_p

...

POSLEDNJA ITERACIJA

- Proverava se uslov (uslov NE važi - petlja se prekida)

KRAJ PETLJE

Može se desiti da uslov ne važi na početku petlje. U tom slučaju se **neće izvršiti nijedna iteracija**. Međutim, isto tako je moguće da se, usled greške, napiše uslov koji uvek važi bez obzira na sve. Efekat će biti taj da će se WHILE petlja neprestano ponavljati i postaće tzv. **“beskonačna petlja”**.

Kao i kod FOR petlje, ako je potrebno ciklično ponavljanje više naredbi, moraju se uokviriti u blok naredbi uz pomoć vitičastih zagrada:

```
while (...uslov...) {
    komanda_p_1;
    komanda_p_2;
    ...
    komanda_p_n;
}
```

Naredbe **“break”** i **“continue”** funkcionišu i kod **WHILE** ciklusa. Ovaj ciklus se može prekinuti i korišćenjem **“return”** naredbe ali se u tom slučaju prekida i metoda u kojoj se ciklus nalazi.

Primer 1

Napraviti klasu **Uvecanje** koja ima:

- Statičku metodu koja kao ulazni parametar dobija neki pozitivan ceo broj A i množi ga samim sobom sve dok ne postane veći od broja 1000. Ova metoda u stvari izračunava najmanje A^n koje je veće od 1000. Ovako umnožen broj je potrebno ispisati na ekranu.

Napraviti klasu **TestUvecanje** koja proverava koji je najmanji stepen broj 2 koji je veći od 1000.

```
class Uvecanje {

    void veciOd1000(int a) {
        int rez = 1;

        while (rez < 1000) rez = rez * a;

        System.out.println(rez);
    }
}

class TestUvecanje {

    public static void main(String[] args) {

        //Ispisace se 1024 (dva na deseti)
        Uvecanje.veciOd1000(2);
    }

}
```

While petlja iz metode **“veciOd1000”** je postavljena tako da se množenje broja samim sobom vrši sve dok je rezultat manji od 1000. U trenutku kada rezultat postane veći od 1000, petlja se prekida i rezultat se ispisuje na ekranu.

U istom primeru se vidi i to da je potrebna opreznost pri pisanju uslova za izlazak iz petlje. Ako se metodi kao ulazni parametar prosledi broj 1, WHILE petlja će da postane beskonačna petlja. To je zbog toga što će rezultat množenja jedinicom uvek biti isti i neće se uvećavati ($1^n = 1$) - biće uvek manji od 1000 bez obzira na to koliko se iteracija izvrši.

Zadaci

Zadatak 1

Napisati klasu **NaucniKalkulator2** koja ima:

- Statičku metodu koja izračunava i vraća najmanje 2 na n-ti stepen (2^n) koje je veće od zadatog broja. Zadati broj se daje u vidu ulaznog argumenta. Primer: ako je zadat broj 6, metoda vraća 8 (2 na treći stepen) jer je to najmanje 2 na n koje je veće od 6.
- Statičku metodu koja izračunava i vraća najveće 2 na n-ti stepen (2^n) koje je manje od zadatog broja. Zadati broj se daje u vidu ulaznog argumenta. Primer: ako je zadat broj 6, metoda vraća 4 (2 na kvadrat) jer je to najveće 2 na n koje je manje od 6.
- Statičku metodu koja izračunava i vraća najmanje A na n-ti stepen koje je veće od broja B. Brojevi A i B su celi brojevi i dati su u vidu ulaznih argumenata. Primer: ako su zadati brojevi 2 i 10, metoda vraća 16 jer je 16 (2 na četvrti) najmanje 2 na n koje je veći od 10.
- Statičku metodu koja izračunava i vraća najmanji stepen (broj) na koji je potrebno podići broj 2 da bi bio veći od zadatog broja. Zadati broj se daje u vidu ulaznog argumenta. Primer: ako je zadat broj 6, metoda vraća 3 jer je 2 na treći stepen jednako 8 i veće je od 6.
- Statičku metodu koja kao ulazni argument dobija ceo broj, deli ga sa brojem dva sve dok je rezultat veći od 1 i ispisuje na ekranu rezultat posle svakog deljenja. Primer: ako se unese broj 33, metoda ispisuje 16 (rezultat prvog deljenja sa dva), 8 (rezultat drugog deljenja sa dva), 4 (rezultat trećeg deljenja sa dva), 2 (rezultat četvrtog deljenja sa dva) i 1 (rezultat poslednjeg deljenja sa dva).
- Statičku metodu koja kao ulazni argument dobija ceo broj A, deli ga sa brojem B (takođe ulazni argument i ceo broj) sve dok je rezultat veći od 1 i ispisuje na ekranu rezultat posle svakog deljenja.

Napisati klasu **TestNaucnogKalkulatora2** koja poziva sve metode klase NaucniKalkulator2.

Rešenje:

```
class NaucniKalkulator2 {

    static int minStepenDva (int broj){
        int rezultat=1;
        while (rezultat<broj) rezultat=rezultat*2;
        return rezultat;
    }

    static int maxStepenDva (int broj){
        int rezultat=1;

        //U ovoj while petlji se proverava slozen uslov tj. da li
        //je trenutni stepen manji od broja i da li je sledeci stepen
        //manji od broja. Kada sledeci stepen nije manji od broja
        //to znaci da je trenutni stepen resenje a ciklus se prekida.
        while ((rezultat<broj)&&((rezultat*2)<broj)) rezultat=rezultat*2;
        return rezultat;
    }

    static int minStepenAB (int a, int b){
        int rezultat=1;
        while (rezultat<b) rezultat=rezultat*a;
        return rezultat;
    }

    static int stepenDva (int broj){
        int rezultat=1;
        int stepen = 0;
        while (rezultat<broj){
            stepen++;
            rezultat=rezultat*2;
        }
        return stepen;
    }

    static void deljenjeSaDva(int broj){
        while (broj>1){
            broj = broj / 2;
            System.out.println(broj);
        }
    }
}
```

```

    }
}

static void deljenjeAB(int a, int b){
    while (a>1){
        a = a / b;
        System.out.println(a);
    }
}

}

class TestNaucniKalkulator2 {

    public static void main (String[] args){

        System.out.println("Najmanje 2^n koje je vece od 12 je "+
            NaucniKalkulator2.minStepenDva(12));

        System.out.println("Najvece 2^n koje je manje od 12 je "+
            NaucniKalkulator2.maxStepenDva(12));

        System.out.println("Najmanje 3^n koje je vece od 30 je "+
            NaucniKalkulator2.minStepenAB(3,30));

        System.out.println("Najmanji stepen broja 2 takav da vazi 2^n > 30 je "+
            NaucniKalkulator2.stepenDva(30));

        NaucniKalkulator2.deljenjeSaDva(33);

        NaucniKalkulator2.deljenjeAB(33,3);

    }

}

```

Zadatak 2

Napraviti klasu **Ispisivac3** koja ima:

- Statičku metodu koja ispisuje na ekranu prvih 10 celih brojeva većih od nule koji su deljivi sa 9 ili sa 6.
- Statičku metodu koja ispisuje na ekranu prvih 5 celih brojeva većih od nule koji su deljivi sa 7 i sa 8.
- Statičku metodu koja ispisuje na ekranu prvih N celih brojeva većih od nule koji su deljivi sa 5 ili sa 6. N je dato kao ulazni argument metode. Primer: ako je N=3, potrebno je ispisati prva tri broja veća od nule koja su deljiva ili sa 5 ili sa 6. To su: 5, 6 i 10.

Napisati klasu **TestIspisivac3** koja poziva sve metode klase Ispisivac3.

Rešenje:

```

class Ispisivac3 {

    static void ispisiPrvihDesetDeljivihSa6Ili9() {
        int broj_deljivih = 0;
        int i = 1;
        while (broj_deljivih < 10) {
            if ((i % 6 == 0) || (i % 9 == 0)) {
                System.out.println(i);
                broj_deljivih++;
            }
            i++;
        }
    }

    static void ispisiPrvihPetDeljivihSa7I8() {

```

```

        int broj_deljivih = 0;
        int i = 1;
        while (broj_deljivih < 5) {
            if ((i % 7 == 0) && (i % 8 == 0)) {
                System.out.println(i);
                broj_deljivih++;
            }
            i++;
        }
    }

    static void ispisiPrvihNDeljivihSa5Ili6(int n) {
        int broj_deljivih = 0;
        int i = 1;
        while (broj_deljivih < n) {
            if ((i % 5 == 0) || (i % 6 == 0)) {
                System.out.println(i);
                broj_deljivih++;
            }
            i++;
        }
    }
}

class TestIspisivac3 {

    public static void main(String[] args) {

        Ispisivac3.ispisiPrvihDesetDeljivihSa6Ili9();

        Ispisivac3.ispisiPrvihPetDeljivihSa7Ili8();

        Ispisivac3.ispisiPrvihNDeljivihSa5Ili6(5);

    }

}

```

Zadatak 3

Napraviti klasu **KalkulatorKamate** koja ima:

- Statičku metodu **izracunajVremeDupliranja** koja kao ulazni parametar dobija iznos kamate u procentima (npr. 9.5%) koju banka daje na godišnjem nivou i izračunava i vraća za koliko godina (ceo broj) će se uložena sredstva udvostručiti tj. posle koliko godina će se glavnica uvećati na više od 200%. Osnovna pretpostavka je da će sredstva iz prethodne godine zajedno sa kamatom za tu godinu biti uložena u narednu godinu tako da će se računati “kamata na kamatu”.
- Statičku metodu **izracunajStednju** koja kao ulazne parametre dobija iznos kamate u procentima (npr. 9.5%) koju banka daje na godišnjem nivou, glavicu (iznos koji se ulaže) i ciljnu sumu (iznos koji se želi dobiti). Metoda bi trebalo da izračuna i ispiše na ekranu posle koliko godina uzastopnih ulaganja će glavnica da premaši ciljnu sumu. Osnovna pretpostavka je da će sredstva iz prethodne godine zajedno sa kamatom za tu godinu biti uložena u narednu godinu tako da će se računati “kamata na kamatu”.

Napraviti klasu **TestKalkulatorKamate** koja proverava za koliko godina se dupliraju uložena sredstva ako je kamata 9.5% i za koliko godina će se 1000 evra uložених po kamati od 4% godišnje uvećati na 3500 evra.

Rešenje:

```

class KalkulatorKamate {

    static int izracunajVremeDupliranja(double kamata) {
        double ukupni_procenat = 100;
        int godine = 0;
    }
}

```

```

        while (ukupni_procenat < 200){
            ukupni_procenat = ukupni_procenat * (100+kamata)/100;
            godine++;
        }

        return godine;
    }

    static void izracunajStednju(double kamata, double glavnica, double ciljnaSuma){
        double iznos = glavnica;
        int godine = 0;

        while (iznos < ciljnaSuma){
            iznos = iznos * (100+kamata)/100;
            godine++;
        }

        System.out.println(godine);
    }
}

class TestKalkulatorKamate {

    public static void main(String[] args) {

        int godine = KalkulatorKamate.izracunajVremeDupliranja(9.5);
        System.out.println(godine);

        KalkulatorKamate.izracunajStednju(4, 1000, 3500);

    }
}

```