SVEUČILIŠTE/UNIVERZITET „VITEZ“

FAKULTET INFORMACIONIH TEHNOLOGIJA

STUDIJ CIKLUSA; GODINA STUDIJA: I; III

SMJER: INFORMACIJSKE TEHNOLOGIJE

Zulka Musić

PROGRAMSKE PETLJE U JAVA PROGRAMSKOM JEZIKU

SEMINARSKI RAD

Travnik, 06.04.2025. godine

SVEUČILIŠTE/UNIVERZITET „VITEZ“

FAKULTET INFORMACIONIH TEHNOLOGIJA

STUDIJ CIKLUSA; GODINA STUDIJA: I; III

SMJER: INFORMACIJSKE TEHNOLOGIJE

PROGRAMSKE PETLJE U JAVA PROGRAMSKOM JEZIKU

SEMINARSKI RAD

IZJAVA: Ja **Zulka Musić**, student Sveučilišta/Univerziteta „VITEZ“, Indeks broj: **390-24/RIIT** odgovorno i uz moralnu i akademsku odgovornost izjavljujem da sam ovaj rad izradio potpuno samostalno uz korištenje citirane literature i pomoć predmetnog profesora.

STUDENT: Zulka Musić

PREDMET: Strukture podataka i algoritmi

MENTOR: prof.dr.sc. Mahir Zajmović

ASISTENT: Admir Sivro

SADRŽAJ

1. UVOD

1.1. PROBLEM, PREDMET I OBJEKT ISTRAŽIVANJA

Problem koji se obrađuje u ovom radu odnosi se na važnost i ulogu programskih petlji u jeziku Java, kao i njihovu pravilnu upotrebu u različitim programerskim situacijama. U savremenom programiranju, petlje su neophodan alat za ponavljanje zadataka i obradu velikih količina podataka, čime značajno utiču na efikasnost i brzinu izvršavanja programa. Međutim, neadekvatna upotreba petlji može dovesti do problema kao što su beskonačne petlje, nepotrebno složen kod i smanjenje performansi programa.

Predmet istraživanja su osnovne vrste petlji u Javi, njihova struktura, način rada i primjena u različitim kontekstima programiranja. Fokus je na analiziranju razlika između for, while i do – while petlji, kao i na mogućnostima optimizacije koda korištenjem odgovarajućih kontrolnih struktura.

U radu se mogu uočiti objekti istraživanja: programske petlje, for petlja, while petlja, do – while petlja, ugniježdene petlje, break, continue i return.

1.2. SVRHA I CILJEVI ISTRAŽIVANJA

Svrha ovog rada je detaljno objašnjenje programskih petlji u Javi kako bi se omogućilo bolje razumjevanije njihove primjene u razvoju softvera. Ciljevi rada uključuju:

- Definisanje i klasifikacija programskih petlji.
- Prikaz razlike između for, while i do-while petlji.
- Analiza kontrole toka izvršenja petlji pomoću break, continue i return naredbi.

1.3. RADNA HIPOTEZA I POMOĆNE HIPOTEZE

Na osnovu predmeta i problema istraživanja ovog seminarskog rada može se postaviti glavna hipoteza:

Pravilno korištenje petlji u programiranju može značajno poboljšati efikasnost i čitljivost koda.

Kroz svrhu i cilj istraživanja možemo izvući tri pomoćne hipoteze:

- Korištenje odgovarajuće vrste petlje zavisno od uslova smanjuje kompleksnost koda.
- Ugniježdene petlje omogućavaju efikasniju obradu podataka u strukturiranim formatima.
- Kontrola toka pomoću break, continue i return naredbi povećava fleksibilnost petlji u rješavanju specifičnih problema.

1.4. NAUČNE METODE

U istraživanju su korištene sljedeće naučne metode:

- Analitička metoda – Analizirano je funkcionisanje i struktura različitih vrsta petlji.
- Eksperimentalna metoda – Implementirani su i testirani primjeri koda kako bi se provjerila praktična primjena petlji.
- Deskriptivna metoda – Opisana su svojstva i karakteristike svake vrste petlji u Javi.

1.5. STRUKTURA RADA

Struktura seminarskog rada je usklađena sa Uputstvom za pisanje seminarskog rada na prvom ciklusu studija kao i temi seminarskog rada. On sadrži sedam poglavlja.

- Prvo poglavlje, Uvod, sadrži pet pod poglavlja:
  - Problem, predmet i objekt isprašivanja,
  - Svrha i ciljevi istraživanja,
  - Radna hipoteza i pomoćne hipoteze,
  - Naučne metode,
  - Struktura rada;
- Drugo poglavlje, Petlja u programiranju u njoj će se obraditi općenito o petlji;
- Treće poglavlje, Važnost petlji u programskom jeziku Java;
- Četvrto poglavlje, Vrste programskih petlji u Javi, sadrži tri pod poglavlja:
  - For petlja, sadrži dva pod poglavlja:
    - Algoritam,
    - Kod;
  - While petlja, sadrži dva pod poglavlja:
    - Algoritam,
    - Kod;
  - Do – while petlja, sadrži dva pod poglavlja:
    - Algoritam,
    - Kod;
- Peto poglavlje, Ugniježdene petlje, sadrži tri pod poglavlja:
  - Ugniježdene for petlje,
  - Ugniježdene while petlje,
  - Ugniježdene do – while petlje;
- Šesto poglavlje, Kontrola toka izvršavanja u petljama, sadrži tri pod poglavlja:
  - Break naredba,
  - Continue naredba,
  - Return naredba;
- Sedmo poglavlje, Zaključak, daje odgovore na postavljene hipoteze.

2. PETLJA U PROGRAMIRANJU

[1]Ukoliko rješavamo problem u programiranju koji zahtjeva da se neki blok naredbi izvrši više puta potrebne su nam naredbe ponavljanja ili petlje. Petlje se koriste za obradu velikih količina podataka, automatizaciju ponavljajućih zadataka i implementaciju složenih algoritama. Gotovo svi moderni programski jezici podržavaju neku vrstu petlji, a najčešće se koriste for, while i do – while petlje.

Svaka od ovih naredbi omogućava da implementiramo ponavljanje bloka naredbi. Jedno izvršavanje bloka naredbi u okviru petlje naziva se iteracija.

U jezicima poput C, C++, Java, Python i JavaScript, petlje funkcionišu na sličnim principima, uz manje razlike u sintaksi i mogućnostima. Neki jezici, poput Python-a, omogućavaju i napredne forme iteracija, kao što su list comprehensions i generatori.

U programiranju, petlje se mogu klasifikovati na:

- Brojačke petlje (for petlja) – Koriste se kada je broj iteracija unaprijed poznat.
- Uslovne petlje (while i do-while petlje) – Koriste se kada broj iteracija zavisi od ispunjenja nekog uslova.
- Rekurzivne petlje – U pojedinim jezicima, petlje se mogu zamijeniti rekurzijom, gdje funkcija poziva samu sebe dok se ne postigne određeni uslov.

Optimizacija petlji je ključna za efikasnost programa, a loše implementirane petlje mogu izazvati probleme kao što su beskonačne petlje, prekomjerna potrošnja memorije i smanjene performanse.

[1]https://www.petlja.org/sr-Latn-RS/biblioteka/r/lekcije/uvod-u-programiranje/nedelja_03 Pristupljeno 23.03.2025.

3. VAŽNOST PETLJI U PROGRAMSKOM JEZIKU JAVA

[2]Petlje u programskom jeziku Java igraju ključnu ulogu u automatizaciji zadataka, smanjenju redundantnog koda i poboljšanju efikasnosti programa. Njihova primjena omogućava obradu velikih skupova podataka, ponavljanje operacija i implementaciju različitih algoritamskih rješenja.

- Efikasnost i optimizacija koda: Petlje omogućavaju izvršavanje istih operacija više puta bez potrebe za ručnim pisanjem istog koda, što značajno smanjuje dužinu i složenost programa.
- Obrada velikih podataka: U softverskim aplikacijama koje rade sa velikim količinama podataka, petlje omogućavaju brzu obradu kroz iteraciju nad kolekcijama, nizovima i bazama podataka.
- Implementacija složenih algoritama: Mnogi algoritmi, kao što su sortiranje, pretraga i analiza podataka, ne bi bili mogući bez upotrebe petlji.
- Automatizacija zadataka: Korištenjem petlji mogu se automatizovati ponavljajući zadaci kao što su generisanje izvještaja, validacija podataka i izvođenje operacija nad korisničkim unosima.
- Povećanje čitljivosti i održivosti koda: Pravilna upotreba petlji poboljšava strukturu koda i olakšava njegovo razumijevanje, održavanje i nadogradnju.

Bez petlji, mnogi zadaci u programiranju bi zahtijevali znatno više vremena i resursa za implementaciju. Njihova ispravna upotreba omogućava optimizaciju performansi aplikacija i povećava efikasnost rada programera.

[2]Schildt, H.: „Java: A Beginner's Guide", Oracle Press, šesto izdanje, 63 – 102 str.

4. VRSTE PROGRAMSKIH PETLJI U JAVI

[3]U Javi postoje tri osnovne vrste petlji, a to su:

- For petlja – koristi se kada je broj ponavljanja unaprijed poznat,
- While petlja – izvršava se sve dok je dati uslov tačan i gdje nije poznat broj ponavljanja,
- Do – while petlja – slična while petlji, ali se uslov provjerava nakon prvog izvršenja petlje.

U praksi, razlike u performansama između ovih petlji su minimalne za većinu aplikacija, osim u slučajevima kada se radi sa veoma velikim skupovima podataka.

4.1. FOR PETLJA

[4]Java for petlja se koristi za iteraciju dijela programa nekoliko puta. Ako je broj iteracija fiksan, preporučuje se korištenje for petlje. Postoje tri tipa for petlje u Javi:

- Jednostavna For petlja – Jednostavna for petlja je ista kao u C/C++. Možemo inicijalizovati varijablu, provjeriti uslov i inkrementirati/dekrementirati vrijednost.

  Sintaksa za jednostavnu for petlju je:

```
for(inicijalizacija; uslov; inkr/dekr){
    //kod koji se izvršava
}
```

- For – each ili unaprijeđena (Enhanced) For petlja – For – each petlja se koristi za prolaz kroz niz ili kolekciju u Javi. Lakša je za upotrebu od jednostavne for petlje

[3]https://www.svetprogramiranja.com/petlje.html Pristupljeno 04.04.2025.
[4]https://www.znanje.org/knjige/computer/Java/ib01/061Java/061_java_for_petlja.htm Pristupljeno 04.04.2025.

zato što nema potrebe da inkrementiramo vrijednost i koristimo indeksnu notaciju. Ona radi na bazi elemenata, a ne indeksa. Ona vraća element jedan po jedan u definisanu varijablu.

Sintaksa za for – each petlju je:

```
for(Tip var: niz){
  //kod koji se izvršava
}
```

- For petlja sa labelom – Moguće je dati ime svakoj for petlji. Da bismo to uradili, koristimo labelu prije for petlje. Ovo je korisno ako imamo ugniježđenu for petlju tako da možemo prekinuti/nastaviti određenu for petlju. Normalno, ključne riječi break i continue prekidaju/nastavljaju samo krajnju unutrašnju for petlju.

  Sintaksa za for petlju sa labelom je:

```
Ime_labele: for(inicijalizacija; uslov; inkr/dekr){
//kod koji se izvršava
}
```

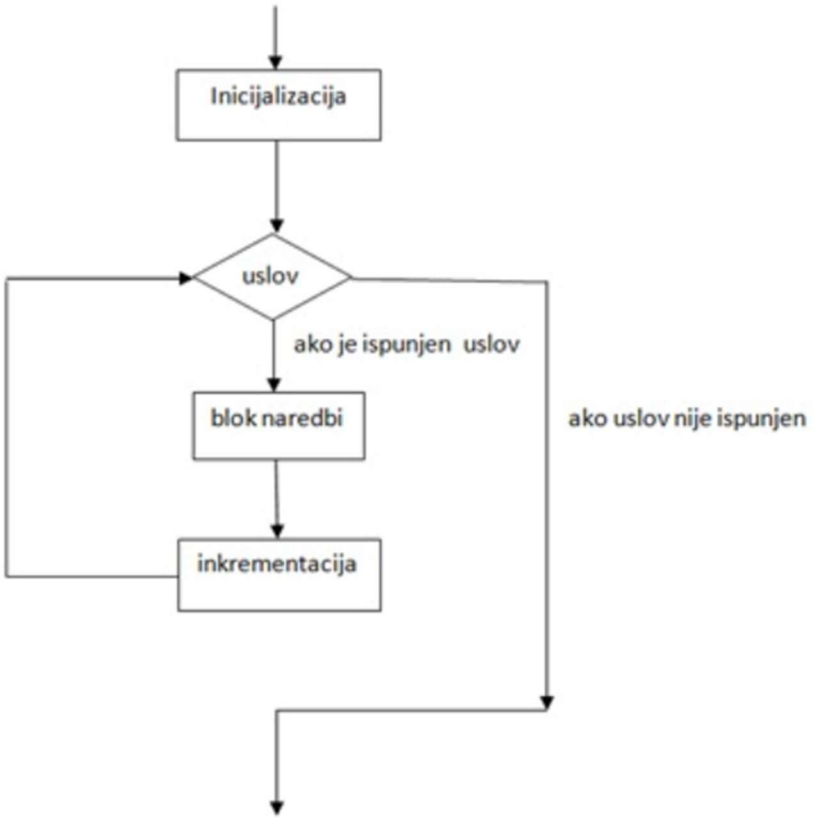
Beskonačna for petlja, nju je moguće postići ako koristimo ;; u for petlji. Sintaksa joj je sljedeća:

```
for(;;){
    //kod koji se izvršava
}
```

4.1.1. Algoritam

Algoritam za for petlju je sljedeći:

Slika 1. Algoritam for petlje



Izvor: https://www.petlja.org/sr-Latn-RS/biblioteka/r/lekcije/uvod-u-programiranje/nedelja_03
Pristupljeno 04.04.2025.

Definiše se i inicijalizuje neka promjenljiva, postavlja se logički izraz kao uslov petlje, potom ide blok naredbi i na kraju se definiše promjena vrijednosti kontrolne promjenljive tokom ciklusa

4.1.2. Kod

- Primjer za jednostavnu for petlju:

```
public class ForPetljaPrimjer1 {
   public static void main(String[] args) {
      for (int i = 1; i <= 5; i++) {
         System.out.println("Broj: " + i);
      }
   }
}
```

- Rezultat:

```
Broj: 1
Broj: 2
Broj: 3
Broj: 4
Broj: 5
```

- Objašnjenje:
  - Public class ForPetljaPrimjer1 { - Definira klasu s imenom ForPetljaPrimjer1. Sve u Javi mora biti unutar klase.
  - Public static void main(String[] args) { - Ulazna tačka svake Java aplikacije. Ovdje počinje izvršavanje programa. Public znači da je javna dostupna svima, static metod pripada klasi a ne objektu, void znači da ne vraća nikakvu vrijednost, dok string[] je niz stringova i args je naziv varijable.
  - Unutar for petlje brojaču smo dodijelili vrijednost, petlja se izvršava sve dok je i <= 5, nakon svakog kruga petlje brojač i se povećava za 1.
  - System.out.println("Broj: " + i); - Ispisuje "Broj: " + trenutnu vrijednost i.

- Primjer za for – each petlju:

```
public class ForEachPrimjer {
   public static void main(String[] args) {
```

```
        String[] imena = {"Emil", "Zulka", "Amila", "Petar"};
        for (String ime : imena) {
            System.out.println("Ime: " + ime);
        }
    }
}
```

- Rezultat:

  Ime: Emil

  Ime: Zulka

  Ime: Amila

  Ime: Petar

- Objašnjenje:
  - public class ForEachPrimjer { - Definira klasu imena ForEachPrimjer.
  - public static void main(String[] args) { - Ulazna tačka programa.
  - String[] imena = {"Emil", "Zulka", "Amila", "Petar"}; - Deklarira i inicijalizira niz stringova imena s 4 elementa.
  - for (String ime : imena) { - for – each petlja, string ime je privremena varijabla koja uzima redom svaki element iz imena, : imena znači iz kojeg niza se uzimaju elementi.
  - System.out.println("Ime: " + ime); - Ispisuje "Ime: " + trenutni string iz niza.

- Primjer za for petlju sa labelom:

```
public class ForPetljaLabela {
    public static void main(String[] args) {
        outerLoop:
        for (int i = 1; i <= 3; i++) {
            for (int j = 1; j <= 3; j++) {
                if (i == 2 && j == 2) {
                    System.out.println("Prekidam vanjsku petlju kod i=" + i + ", j=" + j);
                    break outerLoop;
```

```
            }
            System.out.println("i=" + i + ", j=" + j);
        }
    }
  }
}
```

- Rezultat:

  i=1, j=1

  i=1, j=2

  i=1, j=3

  i=2, j=1

  Prekidam vanjsku petlju kod i=2, j=2

- Objašnjenje:
  - public class ForPetljaLabela { - Definira klasu.
  - public static void main(String[] args) { - Ulazna tačka programa.
  - outerLoop: - Labela (oznaka) koju možemo koristiti s break ili continue.
  - for (int i = 1; i <= 3; i++) { - Vanjska petlja, i ide od 1 do 3.
  - for (int j = 1; j <= 3; j++) { - Unutarnja petlja, j ide od 1 do 3.
  - if (i == 2 && j == 2) { - Kad i = 2 i j = 2, ispunjen je uvjet.
  - System.out.println("Prekidam vanjsku petlju kod i=" + i + ", j=" + j); - Ispis prije prekida.
  - break outerLoop; - Prekida cijelu vanjsku petlju.
  - System.out.println("i=" + i + ", j=" + j); - Ispis trenutnih vrijednosti i i j (izvodi se samo ako se ne prekine prije).

- Primjer za beskonačnu for petlju:

```
public class BeskonacnaForPetlja {
   public static void main(String[] args) {
      int brojac = 0;
      for (;;) {
```

```
            System.out.println("Hello " + brojac);
            brojac++;
            if (brojac >= 5) {
                System.out.println("Prekidamo petlju");
                break;
            }
        }
    }
}
```

- Rezultat:

  Hello 0

  Hello 1

  Hello 2

  Hello 3

  Hello 4

  Prekidamo petlju

- Objašnjenje:
  - public class BeskonacnaForPetlja { - Klasa.
  - public static void main(String[] args) { - Ulaz u program.
  - int brojac = 0; - Inicijalizacija brojača.
  - for (;;) { - Beskonačna petlja – nema uvjeta za prekid, pa ide dok se ne prekine ručno.
  - System.out.println("Hello " + brojac); - Ispisuje "Hello " i trenutni broj.
  - brojac++; - Povećava brojač za 1.
  - if (brojac >= 5) { - Kada brojač dosegne 5 ili više...
  - System.out.println("Prekidamo petlju"); - Ispisuje poruku.
  - break; - Izlazi iz petlje.

4.2. WHILE PETLJA

[5]While petlja se koristi za uzastopno ponavljanje jednog izraza. Tačnije, while petlja ponavlja izjavu dok je zadani uvjet istinit. Sintaksa je sljedeća:

```
while (logički izraz) {
    izrazi
}
```

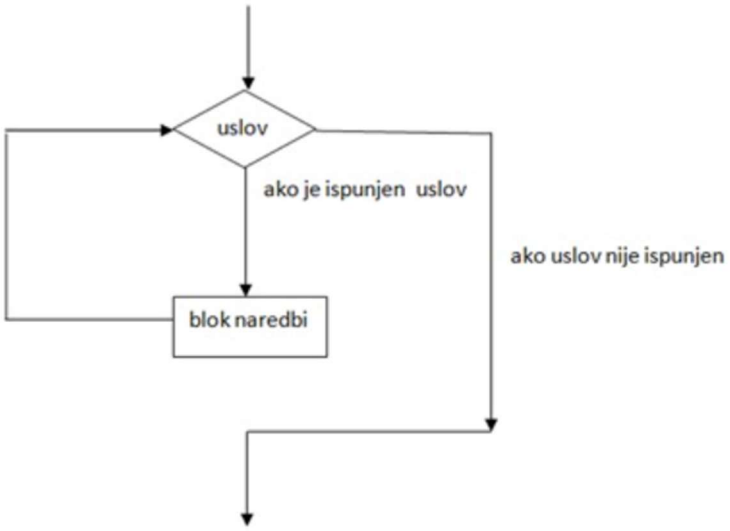
Kad računar dođe do – while izjave, procjenjuje logički izraz koji vraća vrijednost true ili false. Ako je vrijednost izraza false, računar preskače ostatak while petlje i nastavlja s izvršenjem programa. Ako je vrijednost true, računar izvršava izraze unutar petlje, zatim se vraća na početak petlje i ponavlja postupak, tj. ponovo procjenjuje logički izraz, te završava petlju ako je vrijednost false, a nastavlja ako je vrijednost true. Ovo se nastavlja dok izraz ne poprimi vrijednost false; slučaj u kojem se to nikad ne dogodi naziva se beskonačna petlja.

[5]http://laris.fesb.hr/java/blokovi_petlje.htm Pristupljeno 04.04.2025.

4.2.1. Algoritam

Algoritam za while petlju je sljedeći:

Slika 2. Algoritam za while petlju



Izvor: https://www.petlja.org/sr-Latn-RS/biblioteka/r/lekcije/uvod-u-programiranje/nedelja_03
Pristupljeno 04.04.2025.

Definiše se ponavljanje grupe naredbi sve dokle je zadovoljen neki postavljen uslov (logički izraz). Ova petlja se obično koristi kada unaprijed nije poznat broj izvršavanja ciklusa. Može se desiti da ne postoji nijedan slučaj ponavljanja.

4.2.2. Kod

- Primjer za while petlju:

```
public class WhileDjeljivost {
    public static void main(String[] args) {
        int broj = 10;
        while (broj % 7 != 0) {
            System.out.println("Broj " + broj + " nije djeljiv s 7.");
```

```
            broj++;
        }
        System.out.println("Prvi broj djeljiv s 7 je: " + broj);
    }
}
```

- Rezultat:

  Broj 10 nije djeljiv s 7.

  Broj 11 nije djeljiv s 7.

  Broj 12 nije djeljiv s 7.

  Broj 13 nije djeljiv s 7.

  Prvi broj djeljiv s 7 je: 14

- Objašnjenje:
  - public class WhileDjeljivost { - Definiramo klasu s imenom WhileDjeljivost.
  - public static void main(String[] args) { - Glavna metoda programa.
  - int broj = 10; - Postavljamo početni broj na 10.
  - while (broj % 7 != 0) { - Petlja se izvršava dok broj nije djeljiv s 7 (broj % 7 != 0 znači "nije djeljiv s 7").
  - System.out.println("Broj " + broj + " nije djeljiv s 7."); - Ispisuje trenutni broj i obavještava da nije djeljiv s 7.
  - broj++; - Povećava broj za **1**.
  - System.out.println("Prvi broj djeljiv s 7 je: " + broj); - Ispisuje prvi broj koji je djeljiv sa 7.

- Primjer za beskonačnu while petlju:

```
import java.util.Scanner;
public class WhilePin {
    public static void main(String[] args) {
        Scanner unos = new Scanner(System.in);
        int ispravanPin = 1234;
        int uneseniPin;
```

```
        while (true) {
            System.out.print("Unesite PIN: ");
            uneseniPin = unos.nextInt();
            if (uneseniPin == ispravanPin) {
                System.out.println("PIN je ispravan. Dobrodošli!");
                break;
            } else {
                System.out.println("Pogrešan PIN, pokušajte ponovo.");
            }
        }
        unos.close();
    }
}
```

- Rezultat:

  Unesite PIN: 1111

  Pogrešan PIN, pokušajte ponovo.

  Unesite PIN: 1234

  PIN je ispravan. Dobrodošli!

- Objašnjenje:
  - import java.util.Scanner; - Uvozimo Scanner klasu za unos podataka.
  - public class WhilePin { - Definiramo klasu WhilePin.
  - public static void main(String[] args) { - Glavna metoda.
  - Scanner unos = new Scanner(System.in); - Kreiramo Scanner objekt za unos s tastature.
  - int ispravanPin = 1234; - Pohranjujemo ispravni PIN.
  - int uneseniPin; - Deklariramo varijablu za PIN koji korisnik unosi.
  - while (true) { - Beskonačna while petlja – traje dok korisnik ne unese ispravan PIN.
  - System.out.print("Unesite PIN: "); - Tražimo unos PIN-a od korisnika.
  - uneseniPin = unos.nextInt(); - Čitamo broj koji korisnik unese.

- if (uneseniPin == ispravanPin) { - Ako je uneseni PIN jednak ispravnom PIN-u:
- System.out.println("PIN je ispravan. Dobrodošli!"); - Ispisujemo poruku dobrodošlice.
- break; - Izlazimo iz petlje.
- } else { - Ako PIN nije točan:
- System.out.println("Pogrešan PIN, pokušajte ponovo."); - Ispisujemo da je PIN netočan.
- unos.close(); - Zatvaramo Scanner kako bismo spriječili curenje resursa.

4.3. DO – WHILE PETLJA

[6]Do – while petlja je namijenjena slučajevima kada je potrebno da se uvjet ponavljanja testira na kraju petlje umjesto na početku. Do – while petlja je zapravo while petlja s uvjetom ponavljanja postavljenim na kraju petlje. Riječ do se koristi za označavanje početka petlje. Dakle, do – while petlja ima oblik:

```
do {
    izrazi
} while (logički izraz);
```

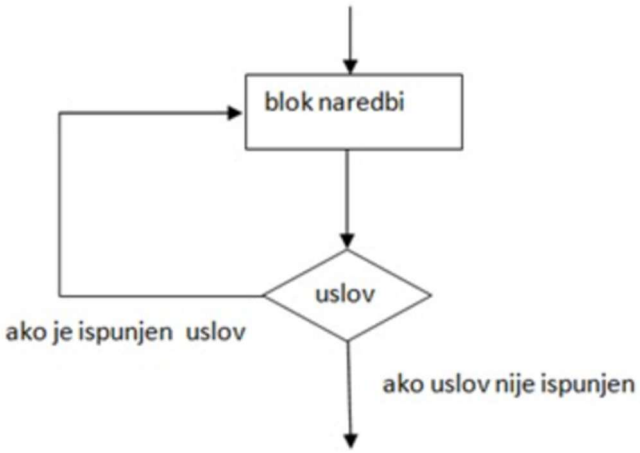
';' na kraju se ne smije izostaviti jer je to dio izraza. Izostavljanje bi prouzrokovalo bi sintaksnu grešku. Pri izvršavanju do petlje, računar prvo izvršava tijelo petlje, tj. izraze unutar petlje, a zatim procjenjuje logički izraz. Ako je vrijednost izraza true, računar se vraća na početak do petlje i ponavlja postupak; ako je vrijednost logičkog izraza false, izlazi iz petlje i nastavlja izvršavanje ostatka programa. Budući da se uvjet nastavljanja procjenjuje tek na kraju petlje, tijelo petlje se izvršava bar jedan put.

[6]http://laris.fesb.hr/java/blokovi_petlje.htm Pristupljeno 04.04.2025.

4.3.1. Algoritam

Algoritam za do – while petlju je sljedeći:

Slika 3. Algoritam do – while petlje



Izvor: https://www.petlja.org/sr-Latn-RS/biblioteka/r/lekcije/uvod-u-programiranje/nedelja_03
Pristupljeno 04.04.2025.

Definiše se ponavljanje grupe naredbi sve dokle je zadovoljen neki postavljen uslov (logički izraz). Ova petlja se obično koristi kada unaprijed nije poznat broj izvršavanja ciklusa i kada nam je iz nekog razloga važno da postoji bar jedan ciklus. Mora se izvršiti bar jedan ciklus.

4.3.2. Kod

- Primjer za do – while petlju:

```
import java.util.Scanner;
public class DoWhilePrimjer {
   public static void main(String[] args) {
      Scanner unos = new Scanner(System.in);
      int broj;
```

```
        do {
            System.out.print("Unesite broj veći od 10: ");
            broj = unos.nextInt();
        } while (broj <= 10);
        System.out.println("Unijeli ste ispravan broj: " + broj);
        unos.close();
    }
}
```

- Rezultat:

  Unesite broj veći od 10: 7

  Unesite broj veći od 10: 11

  Unijeli ste ispravan broj: 11

- Objašnjenje:
  - import java.util.Scanner; - Uvozimo Scanner klasu za unos podataka s tastature.
  - public class DoWhilePrimjer { - Definiramo klasu DoWhilePrimjer.
  - public static void main(String[] args) { - Glavna metoda programa.
  - Scanner unos = new Scanner(System.in); - Kreiramo objekt Scanner za unos s tastature.
  - int broj; - Deklariramo varijablu broj (ali je ne inicijaliziramo).
  - do { Prvi put se petlja uvijek izvrši, bez obzira na uvjet.
  - System.out.print("Unesite broj veći od 10: "); - Tražimo unos broja od korisnika.
  - broj = unos.nextInt(); - Pohranjujemo uneseni broj.
  - } while (broj <= 10); - Provjeravamo uvjet: ako je broj manji ili jednak 10, petlja se ponavlja.
  - System.out.println("Unijeli ste ispravan broj: " + broj); - Kad korisnik unese broj veći od 10, petlja prestaje i ispisujemo poruku.
  - unos.close(); - Zatvaramo Scanner da spriječimo curenje resursa.

- Primjer za beskonačnu do – while petlju:

```
import java.util.Scanner;
public class DoWhileMeni {
   public static void main(String[] args) {
      Scanner unos = new Scanner(System.in);
      int izbor;
      do {
         System.out.println("\n--- Meni ---");
         System.out.println("1. Opcija 1");
         System.out.println("2. Opcija 2");
         System.out.println("3. Izlaz");
         System.out.print("Unesite broj opcije: ");
         izbor = unos.nextInt();
         switch (izbor) {
            case 1:
               System.out.println("Izabrali ste Opciju 1");
               break;
            case 2:
               System.out.println("Izabrali ste Opciju 2");
               break;
            case 3:
               System.out.println("Izlaz iz programa...");
               break;
            default:
               System.out.println("Pogrešan unos, pokušajte ponovo.");
         }
      } while (izbor != 3);
      unos.close();
   }
}
```

- Rezultat:

  --- Meni ---
  1. Opcija 1
  2. Opcija 2
  3. Izlaz
  Unesite broj opcije: 1
  Izabrali ste Opciju 1

  --- Meni ---
  1. Opcija 1
  2. Opcija 2
  3. Izlaz
  Unesite broj opcije: 2
  Izabrali ste Opciju 2

  --- Meni ---
  1. Opcija 1
  2. Opcija 2
  3. Izlaz
  Unesite broj opcije: 5
  Pogrešan unos, pokušajte ponovo.

  --- Meni ---
  1. Opcija 1
  2. Opcija 2
  3. Izlaz
  Unesite broj opcije: 3
  Izlaz iz programa...

- Objašnjenje:
  - import java.util.Scanner; - Uvozimo Scanner klasu za unos s tastature.
  - public class DoWhileMeni { - Definiramo klasu DoWhileMeni.
  - public static void main(String[] args) { - Glavna metoda programa.
  - Scanner unos = new Scanner(System.in); - Kreiramo Scanner objekt za unos s tipkovnice.
  - int izbor; - Deklariramo varijablu izbor, u koju ćemo spremiti korisnički unos.
  - do { - Prvo izvršavamo kod unutar petlje, bez provjere uvjeta.
  - System.out.println("\n--- Meni ---"); - Ispisujemo naslov menija.
  - System.out.println("1. Opcija 1"); - Prva opcija.
  - System.out.println("2. Opcija 2"); - Druga opcija.
  - System.out.println("3. Izlaz"); - Treća opcija – izlaz iz programa.
  - System.out.print("Unesite broj opcije: "); - Tražimo unos opcije.
  - izbor = unos.nextInt(); - Čitamo unos korisnika.
  - switch (izbor) { - Koristimo switch za različite opcije.
  - case 1: - Ako korisnik unese 1:
  - System.out.println("Izabrali ste Opciju 1"); - Ispisujemo poruku.
  - break; - Prekidamo switch.
  - case 2: - Ako korisnik unese 2:
  - System.out.println("Izabrali ste Opciju 2"); - Ispisujemo poruku.
  - break; - Prekidamo switch.
  - case 3: - Ako korisnik unese 3:
  - System.out.println("Izlaz iz programa..."); - Ispisujemo poruku o izlasku.
  - break; - Prekidamo switch.
  - default: - Ako korisnik unese pogrešan broj:
  - System.out.println("Pogrešan unos, pokušajte ponovo."); - Ispisujemo upozorenje.
  - } while (izbor != 3); - Ponovo prikazujemo meni dok korisnik ne unese 3.
  - unos.close(); - Zatvaramo Scanner.

5. UGNIJEŽDENE PETLJE

[7]Ugniježdene (engl. nested) petlje su petlje koje se nalaze unutar druge petlje. Koriste se kada je potrebno izvršavati jednu petlju više puta u okviru druge, što je često slučaj kod rada sa dvodimenzionalnim nizovima, tabelama ili matrica.

Ugniježdene petlje omogućavaju složenije strukture ponavljanja, ali zahtijevaju pažljivo planiranje, jer prekomjerna upotreba može dovesti do velikog broja iteracija i smanjene efikasnosti programa. Preporučuje se izbjegavanje previše nivoa ugniježdavanja, osim kada je to apsolutno neophodno.

5.1. UGNIJEŽDENE FOR PETLJE

- Primjer:

```
public class UgnijezdeniFor {
    public static void main(String[] args) {
        int broj = 1;
        for (int red = 0; red < 3; red++) {
            for (int kolona = 0; kolona < 3; kolona++) {
                System.out.print(broj + "\t");
                broj++;
            }
            System.out.println();
        }
    }
}
```

[7]Schildt, H.: „Java: A Beginner's Guide", Oracle Press, šesto izdanje, 63 – 102 str.

- Rezultat:

  1 2 3

  4 5 6

  7 8 9

- Objašnjenje:
  - int broj = 1; - Početna vrijednost koju ispisujemo.
  - for (int red = 0; red < 3; red++) - Spoljašnja petlja – označava redove.
  - for (int kolona = 0; kolona < 3; kolona++) - Unutrašnja petlja – označava kolone u svakom redu.
  - System.out.print(broj + "\t"); - Ispisujemo broj i tabulator (\t), bez prelaska u novi red.
  - broj++; - Povećavamo broj nakon svakog ispisa.
  - System.out.println(); - Poslije svake unutrašnje petlje, pređemo u novi red.

5.2. UGNIJEŽDENE WHILE PETLJE

- Primjer:

```
public class UgnijezdeniWhile {
   public static void main(String[] args) {
      int red = 0;
      while (red < 4) {
         int kolona = 0;
         while (kolona < 5) {
            System.out.print("* ");
            kolona++;
         }
         System.out.println();
         red++;
      }
   }
```

```
}
```

- Rezultat:

```
* * * * *
* * * * *
* * * * *
* * * * *
```

- Objašnjenje:
  - int red = 0; - Brojač za redove.
  - while (red < 4) - Spoljašnja petlja ide kroz 4 reda.
  - int kolona = 0; - Unutrašnji brojač (resetuje se za svaki red).
  - while (kolona < 5) - Ispisuje 5 zvjezdica u svakom redu.
  - System.out.print("* "); - Ispis jedne zvjezdice.
  - kolona++; - Sljedeća kolona.
  - System.out.println(); - Novi red posle svake linije zvjezdica.
  - red++; - Sljedeći red.

5.3. UGNIJEŽDENE DO – WHILE PETLJE

- Primjer:

```
public class UgnijezdeniDoWhile {
   public static void main(String[] args) {
      int i = 1;
      do {
         int j = 1;
         do {
            System.out.print(i * j + "\t");
            j++;
         } while (j <= 3);
         System.out.println();
         i++;
```

```
        } while (i <= 3);
    }
}
```

- Rezultat:

```
1       2       3
2       4       6
3       6       9
```

- Objašnjenje:
  - int i = 1; - Brojač redova (broj koji množimo).
  - do { - Početak vanjske petlje (za i od 1 do 3).
  - int j = 1; - Brojač kolona (s čime množimo i).
  - do { - Unutrašnja do – while petlja (za j od 1 do 3).
  - System.out.print(i * j + "\t"); - Ispis rezultata množenja i * j.
  - j++; - Povećavamo j.
  - } while (j <= 3); - Kraj unutrašnje petlje.
  - System.out.println(); - Novi red poslije svake linije tablice.
  - i++; - Sljedeći broj (i).
  - } while (i <= 3); - Kraj vanjske petlje.

6. KONTROLA TOKA IZVRŠAVANJA U PETLJAMA

[8]Kontrola toka izvršavanja u petljama predstavlja mehanizme kojima programer upravlja načinom na koji se petlja ponaša tokom izvođenja programa. U Javi, tri ključne naredbe koje se koriste za kontrolu toka unutar petlji su: break, continue i return. Ove naredbe omogućavaju fleksibilniji i efikasniji kod, naročito u slučajevima kada je potrebno reagovati na određeni uslov ili situaciju u toku izvršavanja.

Efikasno korištenje ovih mehanizama omogućava ne samo kraći i pregledniji kod, već i smanjuje šanse za greške u logici programa. Korištenje break, continue i return treba biti pažljivo planirano kako bi se održala čitljivost i jasnoća algoritma.

6.1. BREAK NAREDBA

Naredba break prekida izvršavanje trenutne petlje i prenosi kontrolu na narednu naredbu nakon petlje. Break se koristi za trenutni izlazak iz petlje, bez obzira da li je uslov i dalje ispunjen ili ne. Kada Java naiđe na break unutar petlje, prekida izvođenje te petlje odmah, te se izvršavanje nastavlja sa prvom naredbom poslije petlje. Koristi se kada želimo prerano napustiti petlju, jer je postignut cilj (npr. Pronađen podatak.) i u pretrazi kada više nije potrebno nastaviti iteraciju. Break djeluje samo na najbližu petlju.

- Primjer:

```
import java.util.Scanner;
public class BreakPrimjer {
   public static void main(String[] args) {
      Scanner unos = new Scanner(System.in);
      while (true) {
         System.out.print("Unesi broj (negativan broj prekida unos): ");
         int broj = unos.nextInt();
```

[8]Schildt, H.: „Java: A Beginner's Guide", Oracle Press, šesto izdanje, 63 – 102 str.

```
            if (broj < 0) {
                System.out.println("Negativan broj unesen. Kraj programa.");
                break;
            }
            System.out.println("Unio si: " + broj);
        }
        unos.close();
    }
}
```

- Rezultat:

  Unesi broj (negativan broj prekida unos): 5

  Unio si: 5

  Unesi broj (negativan broj prekida unos): -3

  Negativan broj unesen. Kraj programa.

- Objašnjenje:
  - while (true) – beskonačna petlja, stalno traži unos.
  - Korisnik unosi broj.
  - if (broj < 0) – ako je broj negativan:
    - Ispisuje poruku
    - break; prekida petlju odmah.
  - Ako broj nije negativan, ispisuje što je korisnik unio.
  - unos.close(); – zatvaramo Scanner.

6.2. CONTINUE NAREDBA

[9]Naredba continue preskače ostatak trenutne iteracije i prelazi na sljedeći ciklus petlje. Ovo je korisno kada određene uslove ne želimo da obrađujemo u okviru trenutnog prolaska kroz petlju.

[9]Schildt, H.: „Java: A Beginner's Guide", Oracle Press, šesto izdanje, 63 – 102 str.

- Primjer:

```
public class ContinuePrimjer {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            if (i % 3 == 0) {
                continue;
            }
            System.out.println(i);
        }
    }
}
```

- Rezultat:

  1
  2
  4
  5
  7
  8
  10

- Objašnjenje:
  - for (int i = 1; i <= 10; i++) – petlja od 1 do 10.
  - if (i % 3 == 0) – ako je broj djeljiv s 3:
    - continue; – preskače ostatak petlje, ne ispisuje broj.
  - Inače, ispisuje broj.

6.3. RETURN NAREDBA

[10]U kontekstu metoda, return može prekinuti izvršavanje ne samo petlje, već i cijelog metoda, bez obzira dali se nalazi unutar petlje ili ne. Ovo je posebno korisno kada se nalazi unutar petlje u metodu koji treba da vrati vrijednost. Koristi se kada treba odmah da završi metodu često u provjerama i pretraživanjima, te ako su ostali koraci besmisleni nakon nekog određenog trenutka.

- Primjer:

```
public class ReturnPrimjer {
   public static void main(String[] args) {
      int[] niz = {3, 5, 8, 7, 10};
      if (sadrziSedmicu(niz)) {
         System.out.println("Niz sadrži broj 7.");
      } else {
         System.out.println("Niz NE sadrži broj 7.");
      }
   }
   public static boolean sadrziSedmicu(int[] niz) {
      for (int broj : niz) {
         if (broj == 7) {
            return true;
         }
      }
      return false;
   }
}
```

- Rezultat:

  Niz sadrži broj 7.

[10]Schildt, H.: „Java: A Beginner's Guide", Oracle Press, šesto izdanje, 63 – 102 str.

- Objašnjenje:
  - U metodi sadrziSedmicu koristimo for-each da prođemo kroz niz.
  - if (broj == 7) - čim nađe 7:
    - return true; - odmah izlazi iz metode i vraća true.
  - Ako ne nađe nijednu sedmicu:
    - nakon petlje return false;

7. ZAKLJUČAK

U ovom seminarskom radu detaljno je objašnjena važnost programskih petlji u Javi, njihovih različitih vrsta, kao i mogućnosti kontrole toka izvršavanja unutar petlji. Petlje su osnovne konstrukcije u svakom programskom jeziku, a naročito u Javi, gdje omogućavaju efikasno ponavljanje zadataka i obrada podataka. Razumjevanije osnovnih vrsta petlji, kao što su for, while i do – while, ključno je za svaku osobu koja se bavi programiranjem, jer pravilna upotreba ovih petlji čini kod kraćim, efikasnijim i lakšim za održavanje.

Pravilna kontrola toka izvršavanja petlji pomoću naredbi kao što su break, continue i return, omogućavaju programerima da riješe složene algoritamske probleme uz veću fleksibilnost i čitljivost koda. Korištenje ovih naredbi pomaže u smanjenju rizika od grešaka kao što su beskonačne petlje, čime se osigurava stabilnost i efikasnost programa. Također, ove metode omogućavaju smanjenje vremena izvršenja aplikacija i optimizaciju resursa, što je od izuzetne važnosti u modernom programiranju.

Kroz primjere, pokazano je kako se petlje mogu koristiti u realnim zadacima, čime se povećava produktivnost programera i smanjuje mogućnost grešaka. Također, istraženi su načini na koje ugniježdene petlje mogu obraditi složene strukture podataka, kao i mogućnosti optimizacije petlji u cilju postizanja boljih performansi.

U budućem razvoju aplikacija, sve veća složenost zadataka zahtjeva još bolje razumjevanije programskih petlji i tehnika optimizacije koje one omogućavaju.

Također, važno je napomenuti da se sa napredovanjem tehnologije i promjenama u softverskim inžinjerskim praksama, očekuje da će nove vrste petlji, kao i napredne metode za kontrolu toka izvršavanja, biti implementirane u budućim verzijama programskih jezika, uključujući Javu, što će dodatno unaprijediti efikasnost i performanse aplikacija.

Hipoteze postavljene na samom početku su dokazane i potvrđene.

8. LITERATURA

Knjige:

1. Schildt, H.: „Java: A Beginner's Guide", Oracle Press, šesto izdanje, 63 – 102 str.

Internet izvori:

1. https://www.svetprogramiranja.com/petlje.html Pristupljeno 04.04.2025.
2. https://www.znanje.org/knjige/computer/Java/ib01/061Java/061_java_for_petlja.htm Pristupljeno 04.04.2025.
3. http://laris.fesb.hr/java/blokovi_petlje.htm Pristupljeno 04.04.2025.
4. https://www.petlja.org/sr-Latn-RS/biblioteka/r/lekcije/uvod-u-programiranje/nedelja_03 Pristupljeno 04.04.2025.