Академија струковних
студија Шумадија
одсек у Крагујевцу

Увод у програмирање

Презентација 4

Академија струковних студија Шумадија

Одсек у Крагујевцу

Студијски програм Информатика

Крагујевац, 2020. година

Наредбе понављања - итерације

- Наредба понављања (итерација, циклус или једноставно петља) се користи за понављање извршавања једне исте акције (наредбе или скупа наредби) више пута.
- Јавине наредбе за итерације су:
  - **`while`**
  - **`do-while`**
  - **`for`**
  - посебан случај наредбе **`for`** (for-each)
- Наредба понављања (у даљем тексту петља) извршава исти скуп наредби (или једну наредбу) све док не буде испуњен услов за прекидање итерације.

Петље – обратите пажњу

- Приликом писања наредби понављања морате да обратите пажњу поред синтаксе и на логику извршавања.
- Дефинисање иницијалних вредности које ће да се користе у петљи за проверавање услова или у телу саме петље морају да буду подешене на прави начин (нпр. "бројач", следи пример за овај случај).
- Услов петље мора бити дефинисан на исправан начин.
- У телу петље се морају мењати иницијалне вредности променљивих, које сте дефинисали на почетку, на прави начин (нпр. "бројач", следи пример за овај случај).
- Ваше активности и логику приликом писања петље усмерите на: део програмског кода за иницијализацију, део за петљу и закључни део.

Петља `while`

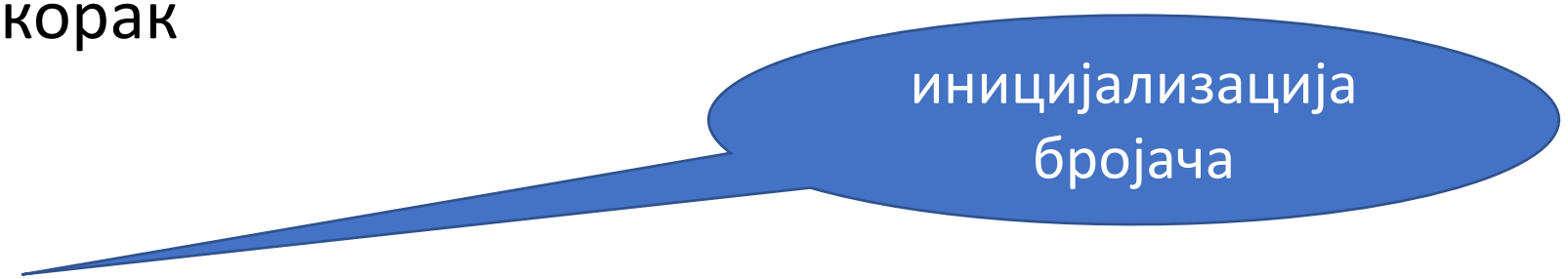
- Петља **`while`** је Јавина најосновнија наредба за понављање.
- Петља **`while`** обезбеђује извршавање неке наредбе (или више наредби) више пута, све док је логички услов задовољен, тј. **`true`**.
- Општи облик ове наредбе:

```
while (uslov){
    naredba;
}
```

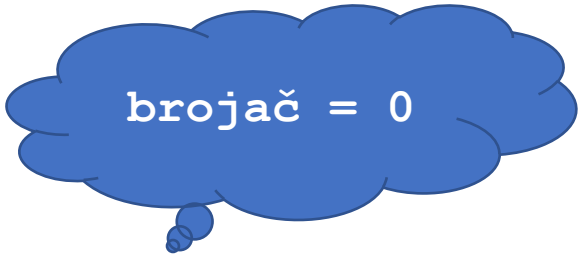
- У **`while`** петљи се **прво проверава да ли је испуњен услов** и ако јесте извршавају се наредбе циклуса које се налазе у телу петље.
- Израз `(uslov)` у наредби **`while`** је логичког типа.

Петља `while`

Петља **`while`** – корак по корак



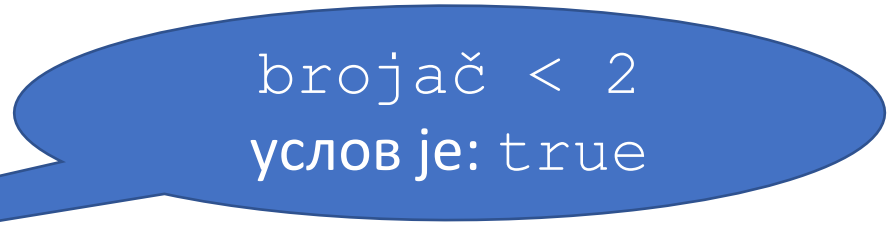

```
int brojač = 0;
while (brojač < 2) {
      System.out.println("Pozdrav!");
      brojač++;
}
```

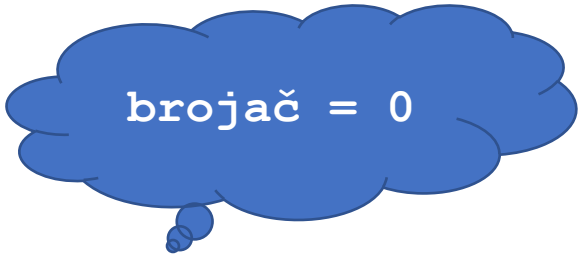


Петља while

Петља **while** – корак по корак

```
int brojač = 0;
while (brojač < 2) {
    System.out.println("Pozdrav!");
    brojač++;
}
```

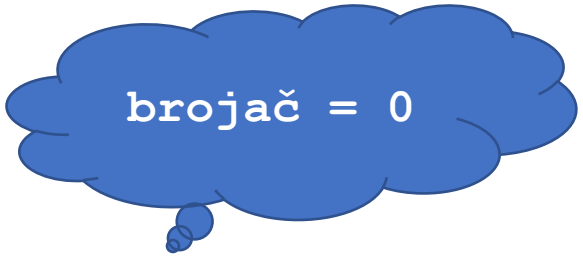




Петља while

Петља **while** – корак по корак

```
int brojač = 0;
while (brojač < 2) {
    System.out.println("Pozdrav!");
    brojač++;
}
```



Петља while

Петља **while** – корак по корак

```
int brojač = 0;
while (brojač < 2) {
      System.out.println("Pozdrav!");
      brojač++;
}
```

brojač се повећава за 1

brojač = 1

Петља while

Петља **while** – корак по корак

```
int brojač = 0;
while (brojač < 2) {
    System.out.println("Pozdrav!");
    brojač++;
}
```

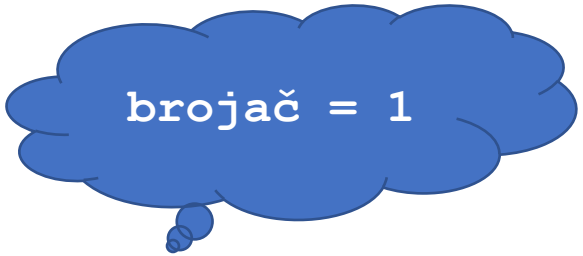
brojač < 2
услов је: true

brojač = 1

Петља while

Петља **while** – корак по корак

```
int brojač = 0;
while (brojač < 2) {
    System.out.println("Pozdrav!");
    brojač++;
}
```



Петља while

Петља **while** – корак по корак

```
int brojač = 0;
while (brojač < 2) {
      System.out.println("Pozdrav!");
      brojač++;
}
```

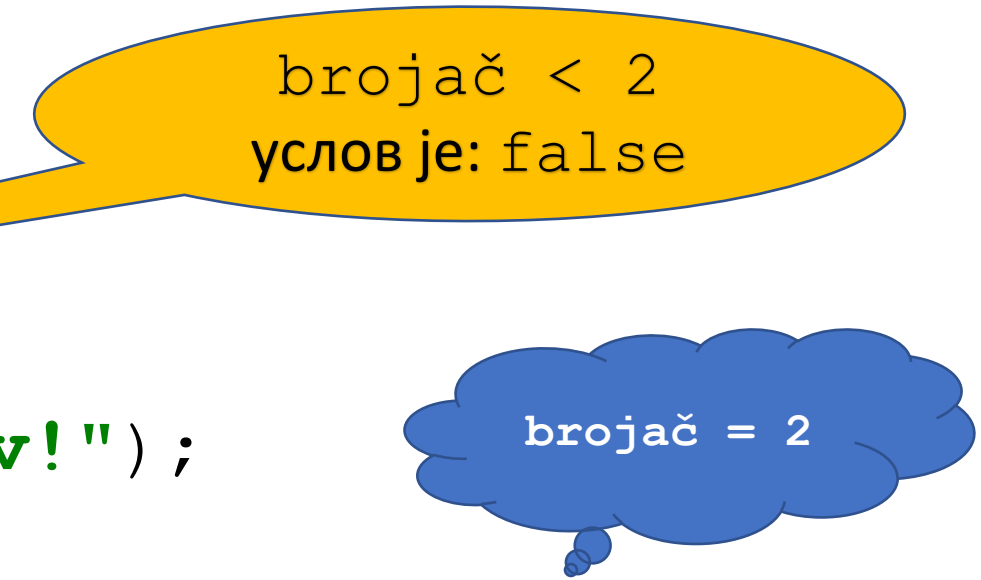
brojač се повећава за 1

brojač = 2

Петља while

Петља **while** – корак по корак

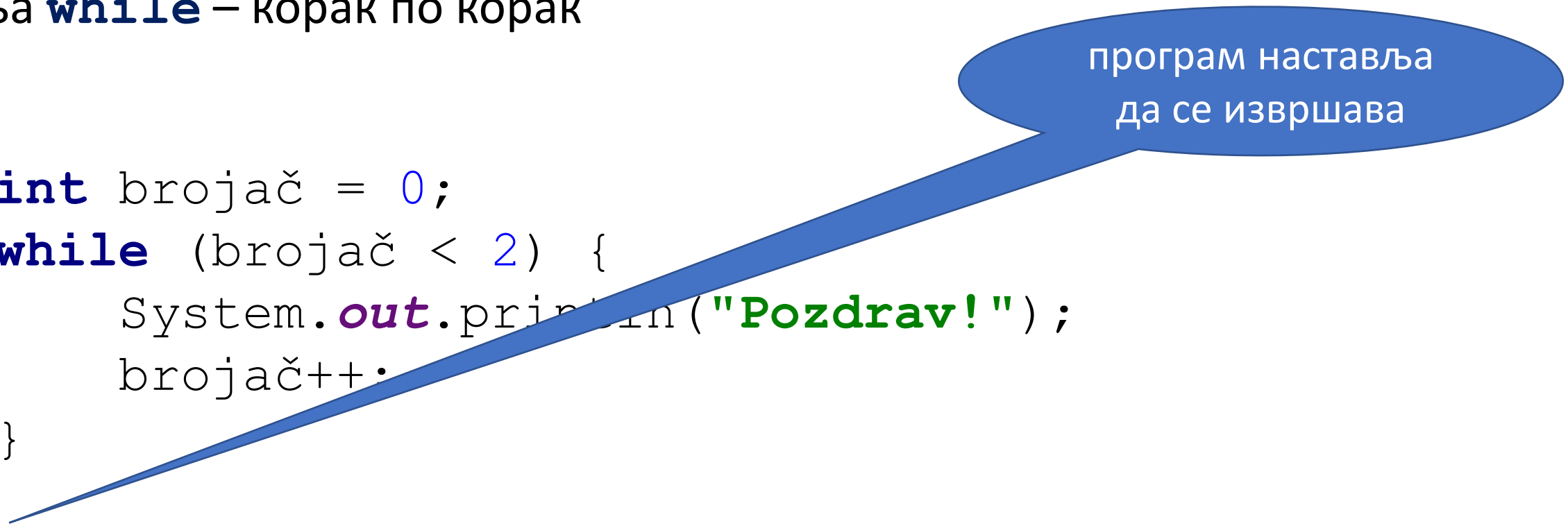
```
int brojač = 0;
while (brojač < 2) {
    System.out.println("Pozdrav!");
    brojač++;
}
```



Петља while

Петља **while** – корак по корак

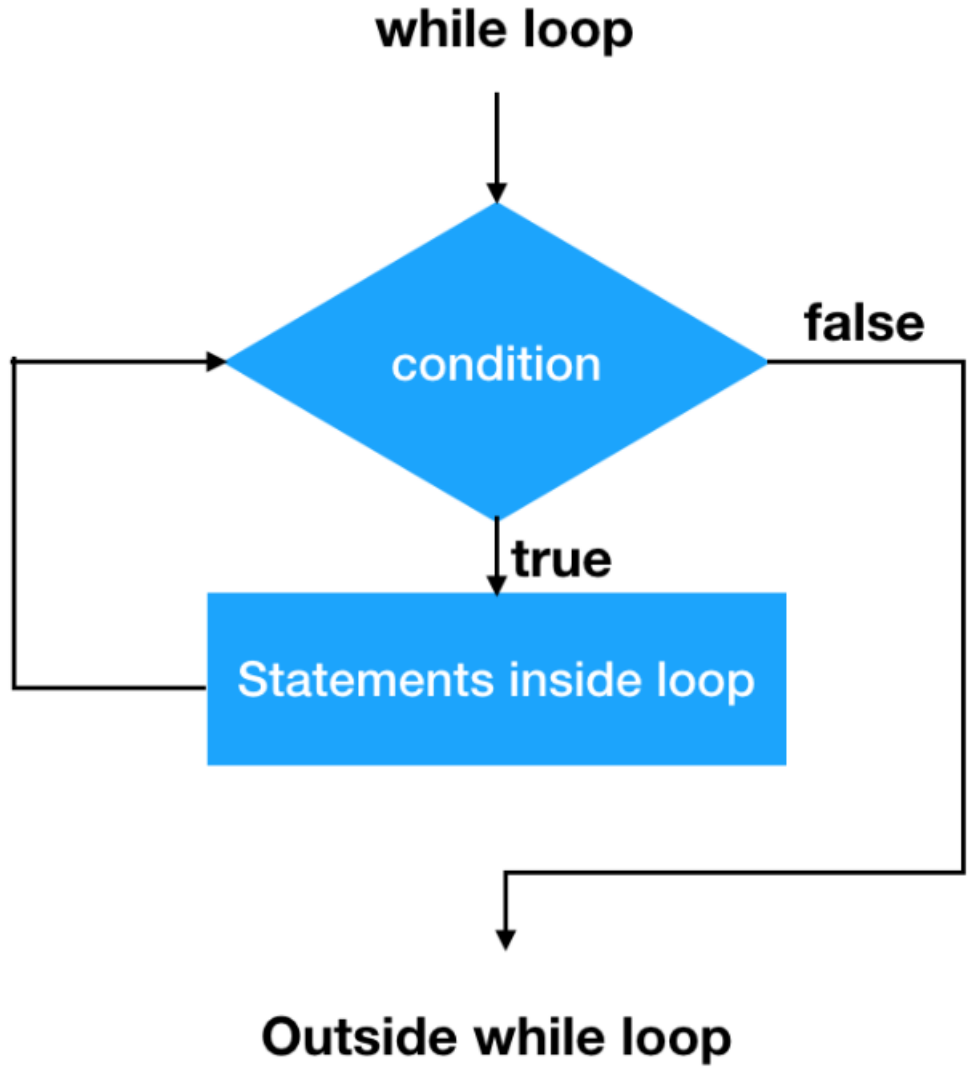
```
int brojač = 0;
while (brojač < 2) {
    System.out.println("Pozdrav!");
    brojač++;
}
```



Петља `while`

Петља **`while`** – алгоритамски приказ

- Ако је услов нетачан на почетку, извршавање **`while`** петље се одмах завршава и наредба се никад не извршава.
- У тренутку када услов постане нетачан, пример нашег бројача када је добио вредност 2, извршавање **`while`** петље се не прекида одмах већ онда када се та вредност провери у услову и услов "врати" вредност **`false`**.

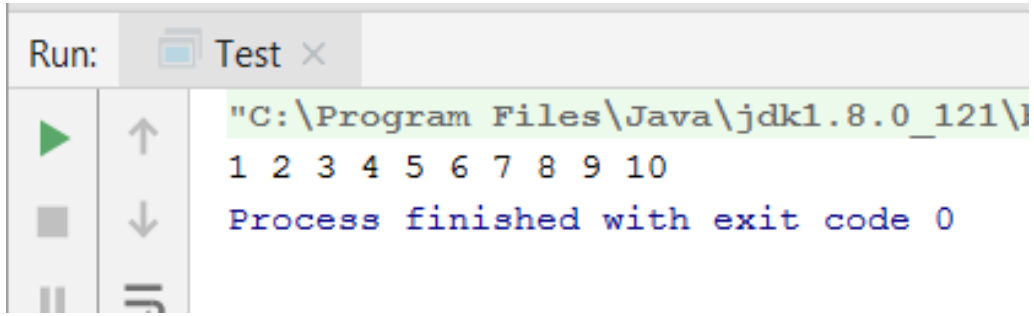


Петља `while`

Петља **`while`** – исписивање бројева од 1 до 10 у једном реду.

```
package rs.edu.asss.test;

public class Test{
    public static void main(String[] args) {
        int broj = 1;
        while (broj <= 10) {
            System.out.print(broj + " ");
            broj++;
        }
    }
}
```



Петља `while`

- Тело петље **`while`** (као и сваке друге наредбе) може да буде празно (празна наредба).
- Следећи пример је синтаксно и логички потпуно исправан.

```
public class Test {
    public static void main(String[] args) {
        int i, j;
        i = 100;
        j = 200;
        // петља која проналази средину између i и j
        while (++i < --j);
        System.out.println("Sredina je: " + i);
    }
}
```

```
Run:  Test
"C:\Program Files\Java\jdk1.8.0_121\
Sredina je: 150

Process finished with exit code 0
```

Петља do-while

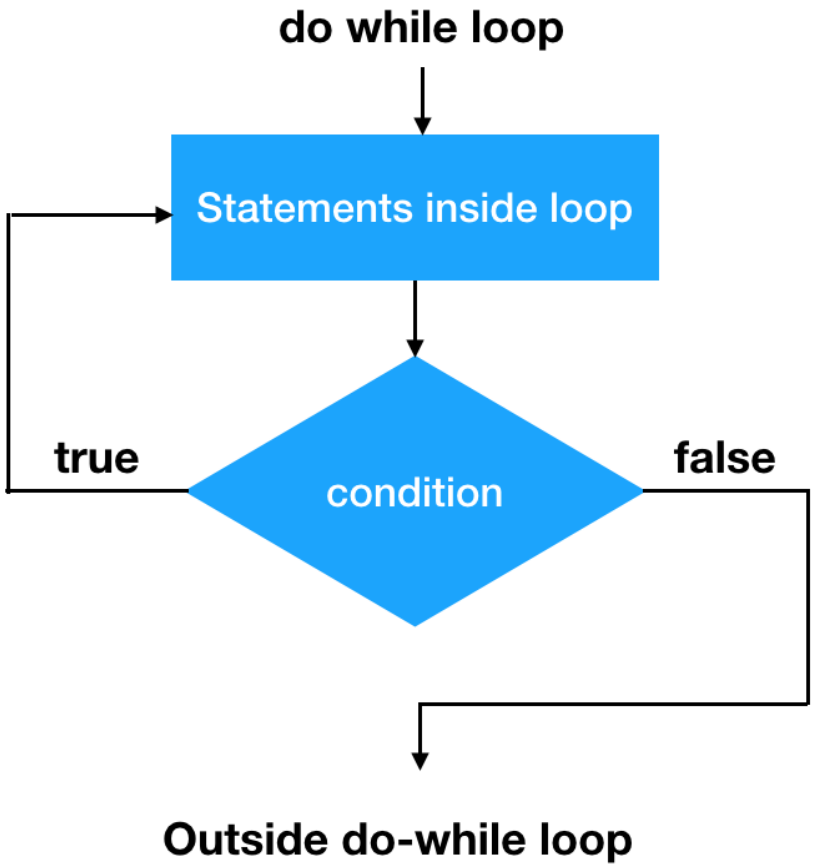
- Петља **while** проверава услов на почетку петље.
- У неким случајевима потребо је да се тело петље (наредба или блок наредби) изврши макра једном пре провераваања услова (пример менија са претходног часа).
- Тада, у тим случајевима услов се проверава **на крају петље**.
- За такве случајева користимо **do-while** петљу.
- Општи облик **do-while** петље:

```
do{
      naredba;
}
while (uslov);
```

Петља `do-while`

Петља **`do-while`** – алгоритамски приказ

- У сваком циклусу петље прво се извршава тело петље, а затим се испитује услов.
- Уколико вредност израза који се испитује "врати" **`true`**, петља наставља да се извршава. У супротном, петља се завршава.
- Као и у свим петљама, услов мора бити логички израз, тј. мора да "врати" вредност **`true`** или **`false`**.



Петља do-while

Петља **do-while** – исписивање бројева од 1 до 10 у једном реду.

```
package rs.edu.asss.test;

public class Test {
    public static void main(String[] args) {
        int broj = 1;
        do{
            System.out.print(broj + " ");
            broj++;
        }
        while (broj <= 10);
    }
}
```

```
Run:  Test
"C:\Program Files\Java\jdk1.8.0_121\
1 2 3 4 5 6 7 8 9 10
Process finished with exit code 0
```

Петља do-while

Претходни пример је технички исправан али може да се напише и на ефикаснији начин.

```
package rs.edu.asss.test;

public class Test {
    public static void main(String[] args) {
        int broj = 1;
        do{
            System.out.print(broj + " ");
        }
        while (++broj <= 10);
    }
}
```

У приказаном примеру, израз `(++broj <= 10)` комбинује увећавање вредности промењљиве за 1 и у истом исказу утврђивање да ли је та вредност мања или једнака броју 10.

Петља `for`

- Петља **`for`** се користи када је потребно извршити наредбу (или блок наредби) у телу петље за све вредности одређене променљиве у неком интервалу.
- Петља **`for`** "комбинује" три корака, одвојених са " **`;`** ", у један исказ:
  - иницијализација
  - услов
  - итерација
- Општи облик **`for`** петље:

```
for (inicijalizacija; uslov; iteracija){
    naredba;
}
```

Петља `for`

- Код **`for`** петље исказ који се налази у загради називамо контролни део:
  `(inicijalizacija; uslov; iteracija)`

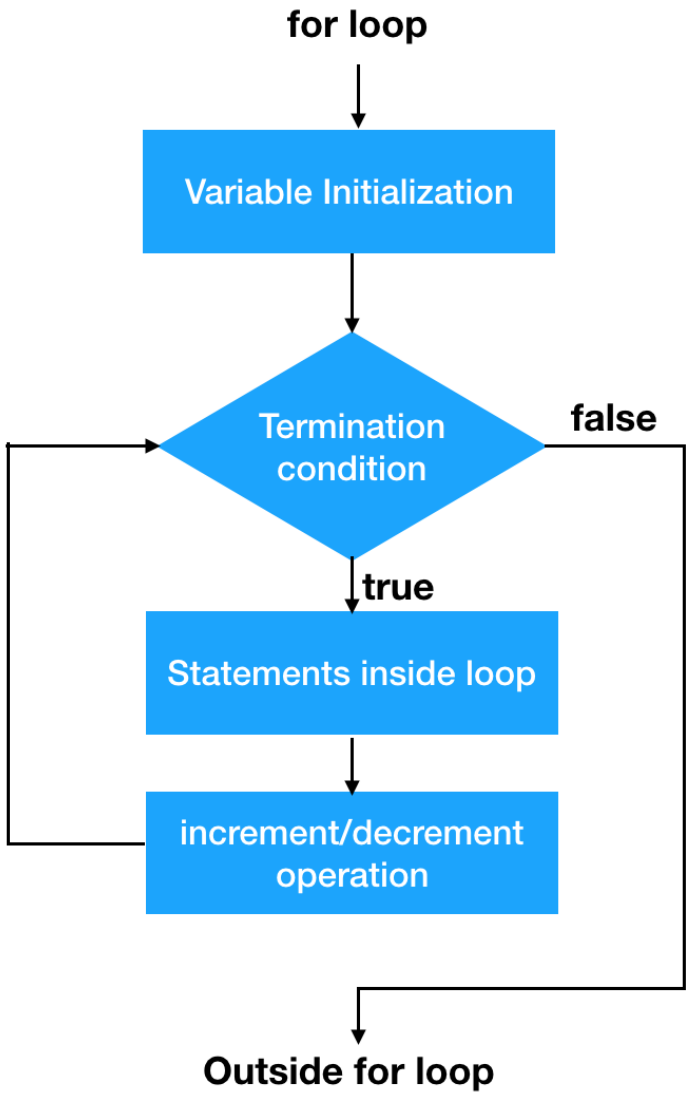
Петља **`for`** се извршава на следећи начин:

- Када петља започне рад извршава се **`inicijalizacija`**.
  - У том кораку се дефинише променљива и задаје јој се вредност (најчешће), а ту променљиву називамо управљачка променљива петље (бројач циклуса петље).
- Следећи корак је испитивање услова, који мора бити логички израз.
- Након тога се извршава **`iteracija`** – израз који увећава или смањује вредност управљачке променљиве.
- Циклус се понавља све док услов "враћа" вредност **`true`**.

Петља `for`

Петља **`for`** – алгоритамски приказ

- На почетку се извршава део иницијализација, само једанпут.
- Логички услов се израчунава пре сваког извршавања тела **`for`** петље.
- Извршавање петље се прекида кад израчуната вредност услова "врати" **`false`**.
- Иницијализација може бити било који израз, обично је то наредба доделе.
- У трећем кораку се увек извршава наредба инкрементирања или декрементирања.

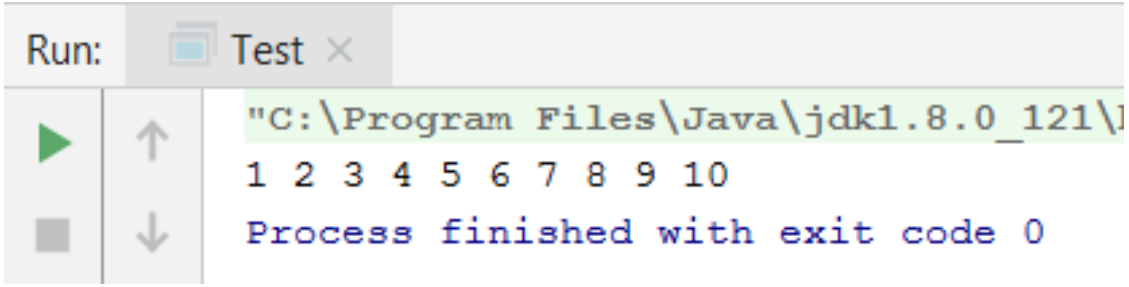


Петља `for`

Петља **`for`** – исписивање бројева од 1 до 10 у једном реду.

```
package rs.edu.asss.test;

public class Test {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++){
            System.out.print(i + " ");
        }
    }
}
```



Петља `for`

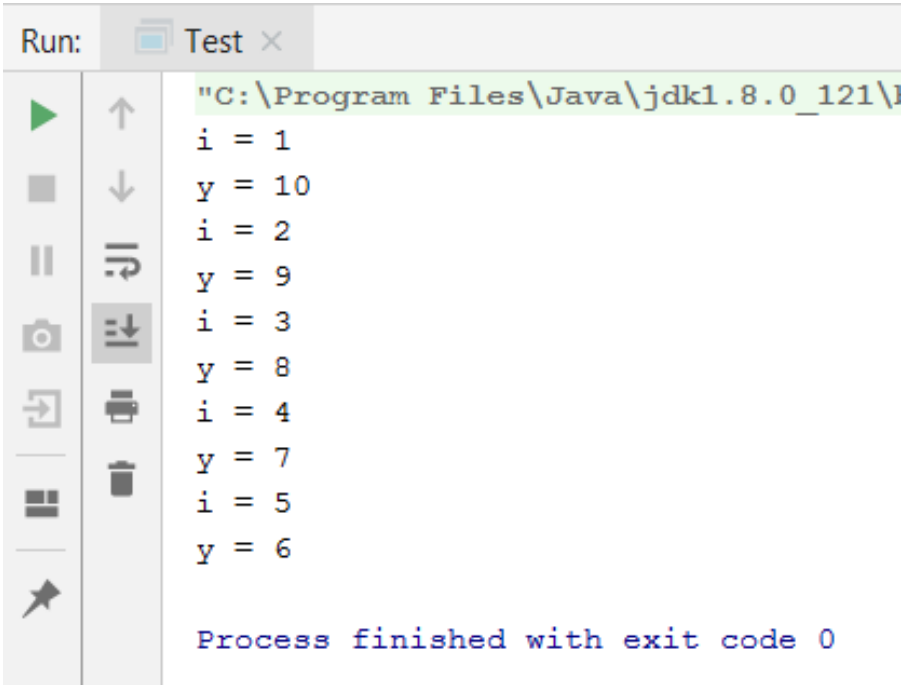
- У претходном примеру управљачка променљива **`i`** је декларисана и иницијализована унутар петље (најчешћи случај).
- Уколико желите да управљачку променљиву користите и изван петље (ређи случај) немојте да је декларишете унутар петље, већ изван ње.

```
public static void main(String[] args) {
    int i;
    for (i = 1; i <= 10; i++){
        System.out.print(i + " ");
    }
}
```

Петља for

- У неким веома ретким случајевима можда се јави потреба да петљом управљају две (или више) променљиве.
- У том случају у делу за иницијализацију све променљиве раздвајамо зарезом.

```
public class Test {
    public static void main(String[] args) {
        int i, y;
        for (i = 1, y = 10; i < y; i++, y--){
            System.out.println("i = " + i);
            System.out.println("y = " + y);
        }
    }
}
```



Петља `for`

- Петља **`for`** подржава многе варијанте које само повећавају њену применљивост.
- Флексибилност **`for`** петље се огледа у томе што се наведена три дела (иницијализација; услов; итерација) не морају користити само у те сврхе.
- То су заиста ретки случајеви коришћења **`for`** петље па их нећемо ни приказивати кодом.
- Постоји и посебна (тзв. "побољшана") верзија имплементација **`for`** петље која се назива *for-each* петља.
- *for-each* петљу веома често користимо код низова, тако да ћу њену имплементацију приказати тада (нпр. код једнодимензионалних низова).

Угнежђене петље

- Као и у свим програмским језицима и у Јави је дозвољено угнежђивање петљи.
- Другим речима, једна петља може да се уметне у другу.

```
public class Test {
    public static void main(String[] args) {
        for (int i = 0; i < 10; i++) {
            for (int j = i; j < 10; j++) {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

```
Run:  Test
"C:\Program Files\Java\jdk1.8.0_121
**********
*********
********
*******
******
*****
****
***
**
*

Process finished with exit code 0
```

Наредбе за скокове

- Јава подржава три наредбе за скокове:
  - **`break`**
  - **`continue`**
  - **`return`** (безуслован излазак из методе, биће објашњено накнадно)

Поред наведених наредби, постоји још један начин за промену извршавања тока програма, а то је обрада изузетака (о томе на предмету ООП).

- Употреба наредби **`break`**, **`continue`** и **`return`** омогућава да се извршавање програма не настави од следеће наредбе по реду, већ од неке друге.
- Наредба **`break`** одмах прекида извршавање петље и прелази се на нормално извршавање остатка програма.
- Извршавањем наредбе **`continue`** прескаче се само остатак актуелне итерације петље. Ретке су ситуације које оправдавају употребу ове наредбе унутар петље.

Литература

Градиво са четвртог предавања :

- Поглавље 5 – део 5.3

https://singipedia.singidunum.ac.rs/izdanje/40716-osnove-java-programiranja

- Ако пратите препоручене видео лекције, градиво које смо обрадили у овом предавању се односи на видео лекције 6 и 7.

https://www.youtube.com/playlist?list=PL-UTrxF0y8kK49N01V5ttb2Xaua7JfyXu

- Одабрана поглавља из књиге: **Јава JDK9: Комплетан приручник**
  - Аутор: Herbert Schildt (може и старије издање JDK7).
  - **Не морате да купујете наведену књигу**.

За испит је, поред скрипти са предавања и вежби, обавезно да учите из књиге која је линкована на почетку.