Cykly

Co je to cyklus?

Cyklus je jazyková konstrukce, která umožňuje opakování instrukcí bez přímé duplikace.

Příklad v pseudokódu

```
operace
operace
operace
operace
operace
```

je stejné jako:

```
opakuj 5:
    operace
```

Jak je vidět, příklad s cyklem má hned několik výhod:

- Operace je kratší
- Počet opakování je určen číslem
- Kód je čitelnější

Typy cyklů

Jazyk Java nabízí uživateli několik různých cyklů, každý s unikátními vlastnostmi a použitím.

1. Cyklus for

Cyklus `for` se používá v případě, že víme kolikrát má daný kód běžet.

> “Potřebuji se pohnout 5x dopředu.”

```
for (int i = 0; i < pocetOpakovani; i++) {
    // Opakující se kód
}
```

Cyklus má v závorkách 3 hodnoty:

- `int i = 0` deklaruje tzv. **iterační proměnnou** ve které je uloženo, v kolikátém opakování cyklu (tzv. iteraci) kód v cyklu je.
- `i < pocetOpakovani` nastavuje podmínku, pod kterou cyklus pokračuje. Z 99.9% v podmínce využívají operátory `<` a `>`.
- `i++` stanovuje co se stane, pokud je podmínka pravdivá.

Pokud má podmínka hodnotu `false` když má cyklus začít běžet, Cyklus se nespustí vůbec

> **Poznámka**
>
> `i` se chová jako normální proměnná, v těle cyklu k ní máš přístup, a můžeš ji i modifikovat (i když to skoro nikdy není potřeba). Proměnná se ani nemusí jmenovat `i` ale je to dobrým zvykem (a standardem).

2. Cyklus foreach

Cyklus foreach se používá pro procházení pole, nebo jiné kolekce a je k tomu syntakticky i funkčně uzpůsoben. Tento cyklus proběhne jednou pro každý prvek v dané kolekci.

```
for (int prvek : pole) {
    // Opakující se kód
}
```

Cyklus má v závorkách pouze 2 hodnoty:

- `int prvek` deklaruje proměnnou, do které se automaticky uloží stávající prvek pole. **Proměnná musí mít stejný typ jako kolekce**
- `pole` je kolekce, přes kterou bude cyklus iterovat

Tento cyklus nemá iterační proměnnou. Pokud potřebuješ v cyklu vědět, na jakém prvku jsi, je lepší využít cyklus `for`

3. Cyklus while

Cyklus `while` se používá v případě, že nevíme kolikrát má cyklus běžet.

> “Potřebuji se hýbat dopředu, dokud nenarazím do zdi.”

```
while (podminka) {
    // Opakující se kód
}
```

Cyklus `while` má v závorce jen jednu hodnotu:

- `podminka` nastavuje cyklu podmínku běhu. Cyklus poběží, dokud podmínka bude mít hodnotu `true`

Pokud bude podmínka mít hodnotu `false` už od začátku, cyklus nepoběží vůbec (přeskočí se)

Ukázka

```
while (true) {
    System.out.print(1);
    System.out.print(2);
}
```

jelikož podmínka nikdy nebude `false` jedná se o nekonečný cyklus a výstup bude následující:

```
12121212 ...
```

Cyklus poběží, dokud nebude zastaven program.

Výrazy pro kontrolu cyklů

Velká část jazyků, včetně Javy, nabízí uživateli také 2 výrazy pro kontrolu chování cyklu.

Výraz continue

Slouží k předčasnému započetí další iterace. Pokud program v průběhu cyklu narazí na výraz `continue`, běh této iterace cyklu okamžitě končí a začíná nová iterace.

Ukázka

```
while (true) {
    System.out.print(1);
    continue;
    System.out.print(2);
}
```

je také nekonečný cyklus, ale výstup bude:

```
11111111 ...
```

Protože na `System.out.print(2);` se díky `continue` nikdy nedostane.

Výraz break

Slouží k zastavení běhu cyklu. Pokud program v průběhu cyklu narazí na výraz `break`, cyklus okamžitě končí bez ohledu na podmínku či iterační proměnnou.

Ukázka

Narozdíl od předchozích 2 tohle:

```
while (true) {
    System.out.print(1);
    break;
    System.out.print(2);
}
```

už není nekonečný cyklus a výstup bude jen:

```
1
```

Protože na `System.out.println(2);` se díky `break` nedostane a cyklus hned skončí.