Anonymní metody

Anonymní metody se definují jako metody, které nemají jméno a nemají na sebe žádný indikátor -> Nemůžeme je přes něj volat.

Lambda

Lambda je kus kódu, který na vstup dostane proměnnou a další vrátí. Nemusí při tom volat metodu, oni sami jsou jim podobné, akorát nemají jméno.

Lambdy mají určitá pravidla -> Nemohou uvnitř obsahovat proměnné, podmínky či cykly. Mohou být psány více způsoby.

Dobře se využívají pro procházení array pomocí forEach a často bývají použity jako parametr na vstupu. Takto například vypíšeme všechny prvky v ArrayListu.
(Poznámka - AL se dělí od array tím, že není stanovaná předem délka)

Pro uchování lambda funkce v proměné potřebujeme importovat java.util.function.Consumer.

```
import java.util.ArrayList;

public class Main {

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<Integer>();
        list.add(5);
        list.add(2);
        list.add(3);
        list.add(4);
        list.add(7);
        list.add(6);
        list.forEach( (n) -> {System.out.println(n);});
    }
}
```

System.out.println(n) nám vrátí něco ve stylu

```
Main$$Lambda$1/471910020@1218025c
```

```
import java.util.ArrayList;
import java.util.function.Consumer;
public class Main {

    public static void main(String[] args) {
        ArrayList<Integer> list = new ArrayList<Integer>();
        list.add(5);
        list.add(2);
        list.add(3);
        list.add(4);
        list.add(7);
        list.add(6);
        Consumer<Integer> method = n -> System.out.println(n);
        System.out.println(method);
        list.forEach(method);
    }
}
```

Proč lambdu použít?

Pomáhají filtrovat a získávat data z kolekcí bez zbytečně dlouhého kódu. Jsou převážně krátké a jednoduché na pochopení.

Pozor, nejsou vždy rychlejší! Mohou být i pomalejší.

Můžeme díky nim splnit určitý interface.

Delegát

Delegát by se dal popsat jako referenční datový typ. Díky němu můžeme odkazovat na nějakou metodu skrze proměnnou. U delegátů obecně se řeší tři kroky – deklarace, nastavení reference a vyvolávání delegáta.

Díky delegátům dosahujeme čistčího kódu, protože můžeme vytvářet anonymní metody a můžeme je využít v eventech.

Abychom v C# použili delegáta, používáme klíčové slovo delegate. Když ho deklarujeme, stanovujeme, jaký typ metody do něj můžeme ukládat. Metody mají Method signature – jaké vstupní a výstupní data.

```
public delegate void printdel(int a, int b);
public static void Print(int a, int b) {
   Console.WriteLine(a+b)
}

static void Main(string[] args) {
   printdel print = Print;
   print(1,2);
}
```

V tomto případě máme proměnnou, která ukazuje na funkci.

V C# můžeme udělat multicast – Do jedné proměnné můžeme vložit více metod.

```
public delegate void printdel(int a, int b);
public delegate void Math(int a, int b);
public static void add(int a, int b) {
   Console.WriteLine(a+b)
}
public static void multiply(int a, int b) {
   Console.WriteLine(a*b);
}

static void Main(string[] args) {
   printdel print = add;
   print(1,2);

   Math add = add;
   Math mutliply = multiply;
   Math operations = add + multiply;
}
```

Přes multicast si vytvoříme dva delegáty a poté je spojíme do jednoho. Multicast můžeme volat různými způsoby, třeba operations(1,2), nebo operations.Invoke(1,2).

U multicastu můžeme dokonce i odebírat během běhu programu metody.

Můžeme tedy udělat

```
operations -= multiply
```

A díky tomu odebereme metodu z delegáta. Příklad může být tlačítko, která zabrání sbírání dat. Klikneme na něj a v programu se odebere metoda z delegáta o sbírání dat.

Delegáty můžeme dávat i jako vstupní hodnoty do metod.

```
public delegate void printdel(int a, int b);
public delegate void Math(int a, int b);
public static void add(int a, int b) {
   Console.WriteLine(a+b)
}
public static void multiply(int a, int b) {
   Console.WriteLine(a*b);
}

static void Main(string[] args) {
   invokeDel(add)
}

public static void InvokeDel(Math operation) {
   operation.Invoke(1,2)
}
```

Func – Přijímá metody, všechny kromě voidu (0-16 vstupních parametrů)

Jako první je parametr na vstupu, jako poslední je return typ.

```
public static bool je_sudy(int a)
   return a % 2 == 0;
}
static void Main(string[] args) {
   Func<int,bool> func = je_sudy;

 func.Invoke(3);
}
```

Action – Je void

Predicate – Přijímá jenom bool a jeden parametr a vrací jenom True a False