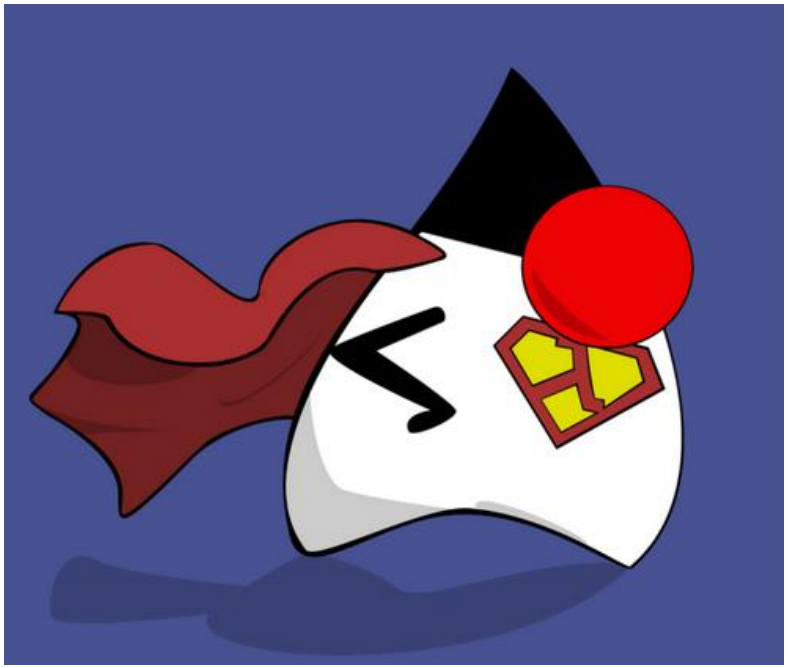
Streams
Lambdas



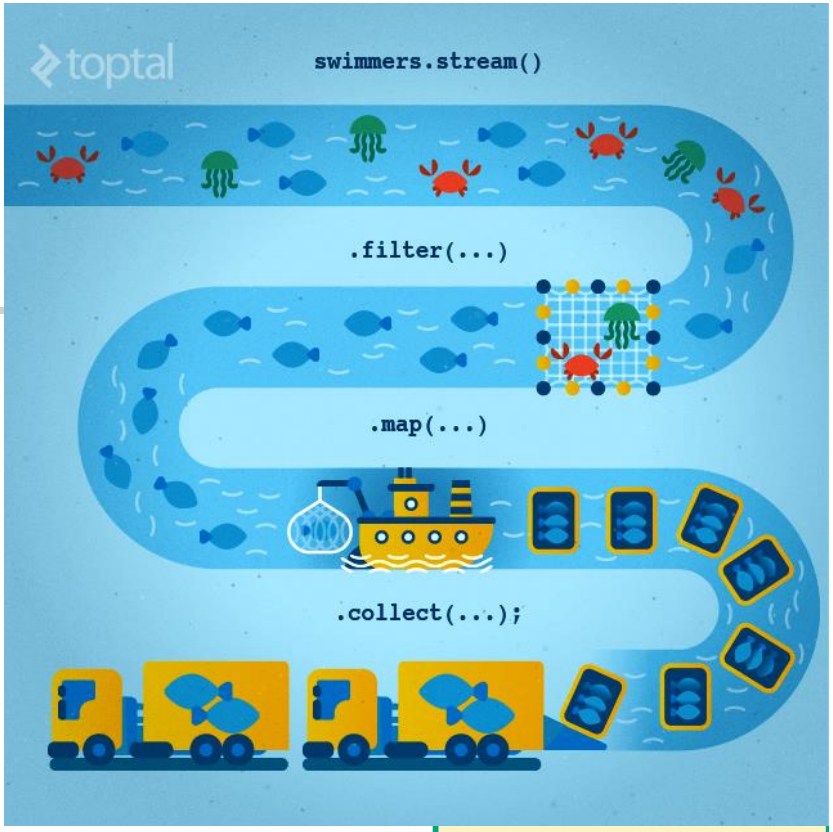
Peter Borovanský
KAI, I-18

borovan 'at' ii.fmph.uniba.sk
http://dai.fmph.uniba.sk/courses/JAVA/

Java Collections

dnes bude:

Cvičenie (bohužiaľ nemáme):

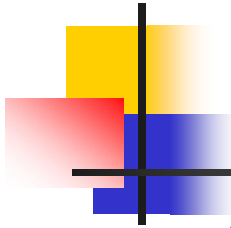


(v terminálke)

Slajd = Java 9
(v LISTe)

Slajd = Java 10
(v LISTe)

http://en.proft.me/2016/11/13/tutorial-lambda-expression-and-stream-api-java-8/



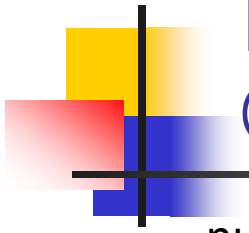
Anonymné funkcie v Jave
(lambdas)

```
(double a, double b) -> { return Math.sqrt(a*a+b*b); }
```

- Typová inferencia typov parametrov

```
(a, b) -> { return Math.sqrt(a*a+b*b); }

(a, b) -> { Math.sqrt(a*a+b*b) }

(a, b) -> Math.sqrt(a*a+b*b)

n -> n*n
```

- Lambda notácia fungujú s funkcionálnym interface = má jednu metódu
- anotácia pre FI je @FunctionalInterface



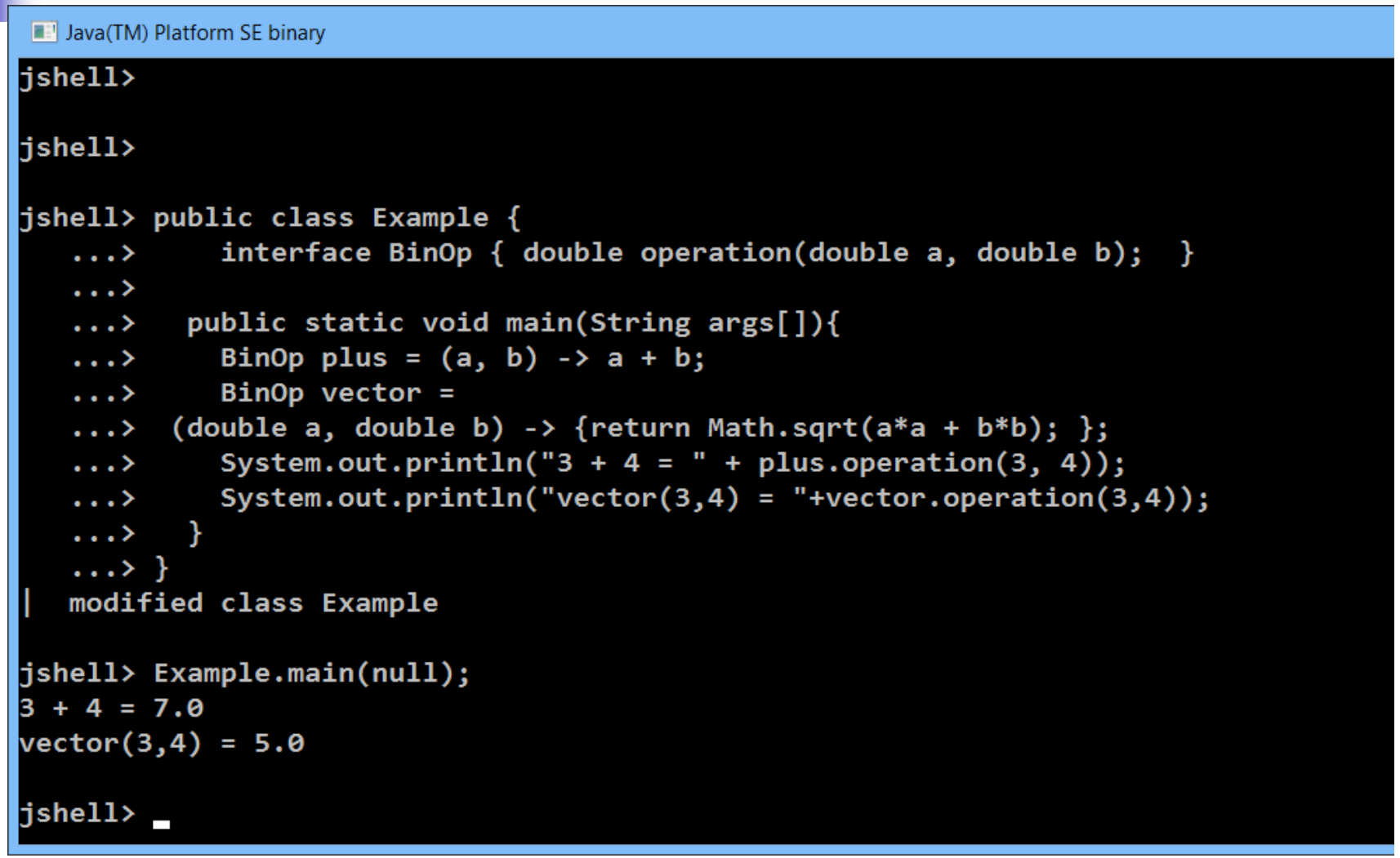
Príklad

(anonymné lambdy)

```
public class Example {

    interface BinOp { double operation(double a, double b);  }

  public static void main(String args[]){
    BinOp plus = (a, b) -> a + b;
    BinOp vector =
        (double a, double b) -> {return Math.sqrt(a*a + b*b); };
    System.out.println("3 + 4 = " + plus.operation(3, 4));
    System.out.println("vector(3,4) = "+vector.operation(3,4));
  }
}
Example.main(null);
```

Jshell

(Java 9)



JDK8 - funkcionálny interface

```
interface FunkcionalnyInterface { // koncept funkcie v J8
   public void doit(String s);    // jediná "procedúra"
}

                              // „procedúra" ako argument
public static void foo(FunkcionalnyInterface fi) {
   fi.doit("hello");
}

                           // „procedúra" ako hodnota, výsledok
public static FunkcionalnyInterface goo() {
   return (String s) -> System.out.println(s + s);
}


foo(goo())
"hellohello"
```

Funkcie.java

JDK8 - funkcionálny interface


```
public interface FunkcionalnyInterface { //String->String
   public String doit(String s); // jediná "funkcia"
}
                                  // "funkcia" ako argument
public static String foo(FunkcionalnyInterface fi) {
   return fi.doit("hello");
}
                                  // "funkcia" ako hodnota
public static FunkcionalnyInterface goo() {
   return
        (String s)->(s+s);
}
System.out.println(foo(goo()));
"hellohello"
```

Funkcie.java

JDK8 - funkcionálny interface

```
public interface RealnaFunkcia {
   public double doit(double s);   // funkcia R->R
}
public static RealnaFunkcia iterate(int n, RealnaFunkcia f){
   if (n == 0)
        return (double d)->d;        // identita
   else {
        RealnaFunkcia rf = iterate(n-1, f);      // f^(n-1)
        return (double d)->f.doit(rf.doit(d));
   }
}
RealnaFunkcia rf = iterate(5, (double d)->d*2);
System.out.println(rf.doit(1));
```

Funkcie.java

JDK8 - funkcionálny interface

java.lang.Runnable	void **run**()
java.util.concurrent.Callable	
java.io.FileFilter	boolean **accept**(File pathname)
java.util.Comparator<T>	int **compare**(<T> o1, <T> o2)
Function<T,R>	<R> apply(<T>)
Predicate<T>	boolean test(<T>)

Príklady:

```
Function<Double,Double> celsius2Fahrenheit = x -> ((x*9/5)+32);
Function<Double,Double> rad2Deg = r -> ((r/Math.PI)*180);
Function<String, Integer> string2Int = x -> Integer.valueOf(x);
Function<Integer, String> int2String = x -> String.valueOf(x);

System.out.println("C->F: "+celsius2Fahrenheit.apply(30.0)); // 86.0
System.out.println("rad2Deg: "+rad2Deg.apply(Math.PI)); // 180
System.out.println(" string2Int: " + string2Int.apply("4")); // 4
System.out.println(" int2String: " + int2String.apply(123)); // "123"
```

Java 8

```
String[] pole = { "GULA", "cerven", "zelen", "ZALUD" };
Comparator<String> comp =
(fst, snd)->Integer.compare(fst.length(), snd.length());

Arrays.sort(pole, comp);
for (String e : pole) System.out.println(e);
```

```
GULA
zelen
ZALUD
cerven
```

```
Arrays.sort(pole,
   (String fst, String snd) ->
       fst.toUpperCase().compareTo(snd.toUpperCase()));

for (String e : pole) System.out.println(e);
```

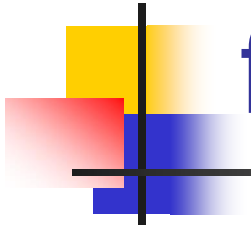
```
cerven
GULA
ZALUD
zelen
```

Funkcia.java

forEach, map, filter v Java8

```
class Karta {
   int hodnota;
   String farba;
   public Karta(int hodnota, String farba) { … }
   public void setFarba(String farba) { … }
   public int getHodnota() { … }
   public void setHodnota(int hodnota) { … }
   public String getFarba() { … }
   public String toString() { … }
}
List<Karta> karty = new ArrayList<Karta>();
karty.add(new Karta(7,"Gula"));
karty.add(new Karta(8,"Zalud"));
karty.add(new Karta(9,"Cerven"));
karty.add(new Karta(10,"Zelen"));
```

MapFilter.java



forEach, map, filter v Java8

[Gula/7, Zalud/8, Cerven/9, Zelen/10]

```
karty.forEach(k -> k.setFarba("Cerven"));
```

[Cerven/7, Cerven/8, Cerven/9, Cerven/10]

```
Stream<Karta> vacsieKartyStream =
   karty.stream().filter(k -> k.getHodnota() > 8);
List<Karta> vacsieKarty =
   vacsieKartyStream.collect(Collectors.toList());
```

[Cerven/9, Cerven/10]

```
List<Karta> vacsieKarty2 = karty
   .stream()
   .filter(k -> k.getHodnota() > 8)
   .collect(Collectors.toList());
```

[Cerven/9, Cerven/10]

MapFilter.java

forEach, map, filter v Java8

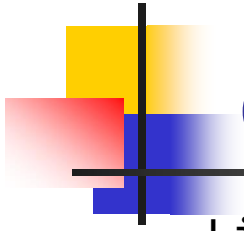
```
List<Karta> vacsieKarty3 = karty
    .stream()
    .map(k->new Karta(k.getHodnota()+1,k.getFarba()))
    .filter(k -> k.getHodnota() > 8)
    .collect(Collectors.toList());
```

[Cerven/9, Cerven/10, Cerven/11]

```
List<Karta> vacsieKarty4 = karty
    .stream()
    .parallel()
    .filter(k -> k.getHodnota() > 8)
    .sequential()
    .collect(Collectors.toList());
```

[Cerven/9, Cerven/10]

MapFilter.java



Kolekcie

(a práca s nimi – ako to poznáme)

```
List<Integer> lst = new ArrayList<Integer>();
List<Integer> lst = new ArrayList<>();

for (int i = 0; i < 100; i++)
   lst.add(i);
// explicitná inicializácia
List<Integer> lst1 = Arrays.asList(0,1,2,3,4,5,6,7,8,9);
// Nová syntax Java 9
List<Integer> list = List.of(0, 1, 2, 3, 4, 5, 6, 7, 8, 9);
Set<Integer> set = Set.of(0, 1, 2, 3, 4, 5, 6, 7, 8, 9);
Map<String,Integer> map = Map.of("Jano",1, "Palo",3, "Igor", 0);
for (Integer value : lst)              // foreach cyklus
   System.out.println(value);
lst.forEach(System.out::println);    // foreach metóda
lst.forEach(e -> System.out.println(e+e));
```

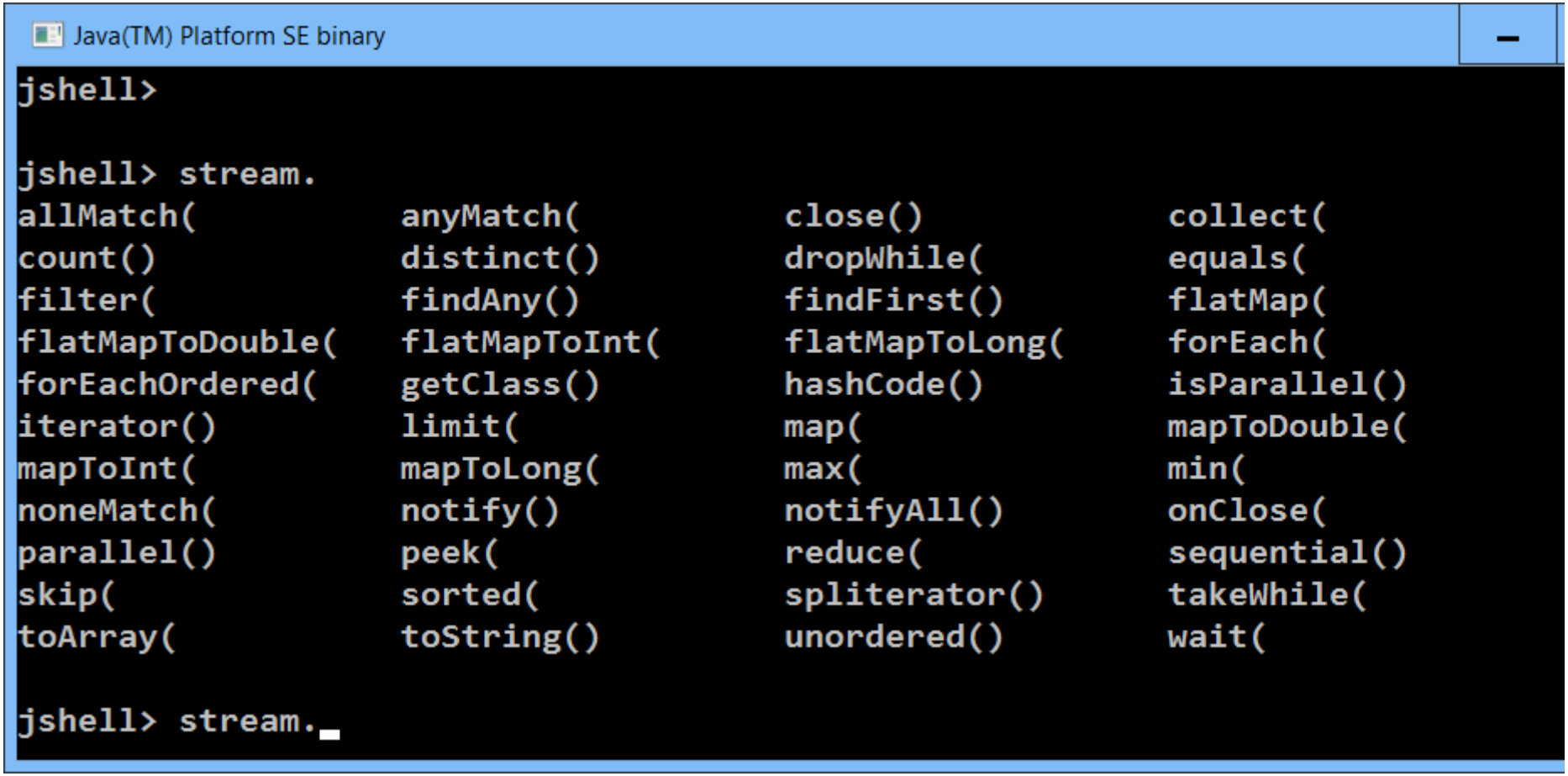
JAVA 8 Stream API

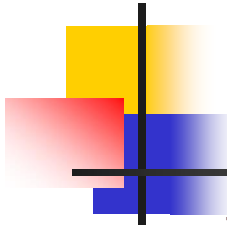
(sekvenčný a paralelný)

```
lst.stream().forEach(e -> System.out.println(e+e));

// stream() prerobí kolekciu na java.util.stream
Stream<Integer> stream = lst.stream();
                stream.count() // 100
Stream<Integer> stream = lst.stream();
stream.forEach(System.out::println);
stream.forEach(System.out::println); // !!!
// Excetion: stream has already been operated upon or closed

// toto už nedostaneme v poradí 0, 1, ...
lst.parallelStream().forEach(e -> System.out.println(e+e));
```

Jshell pozná autocompletion

- Kliknite na TAB

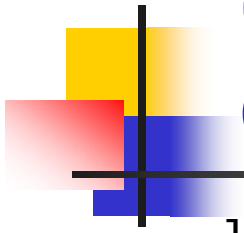




map/filter

(existuje/neexistuje/pre všetky)

```
lst.
   stream().
   filter(e -> (e % 2 == 0)).
   forEach(System.out::print);        // 024681012141618202222...
lst.
   stream().
   map(e -> e*e).
   forEach(System.out::print);        // 01491625364964 ...

lst.stream().anyMatch(e -> (e == 51))        // true
lst.stream().anyMatch(e -> (e * e == e))     // true
lst.stream().noneMatch(e -> (e > 100))       // true
lst.stream().noneMatch(e -> (e + e == e))    // false
lst.stream().allMatch(e -> e>0 )             // false
lst.stream().filter(e -> e>0 ).count()       // 99
```



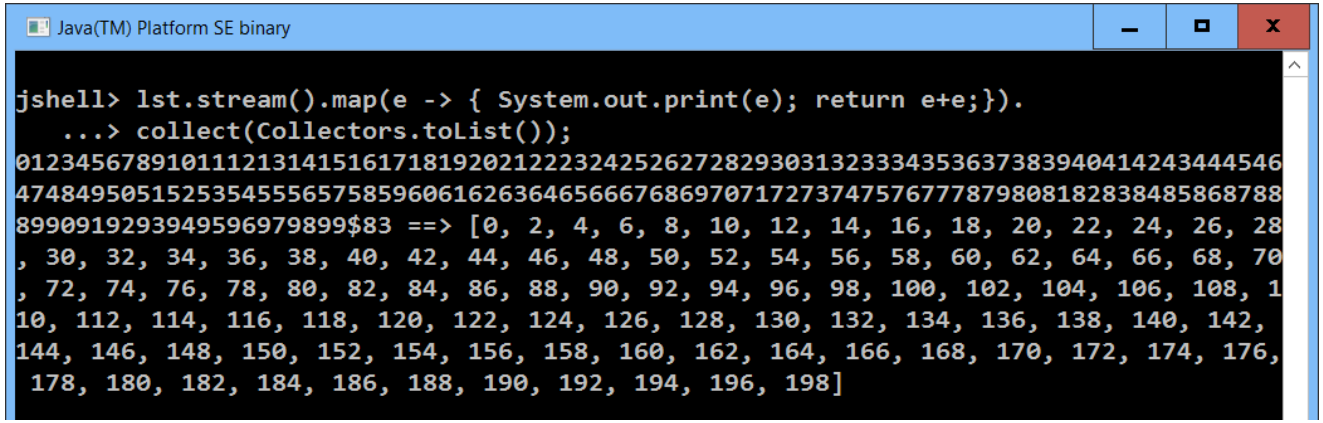
Optional

(buď existuje alebo neexistuje)

```
lst.stream().findFirst()              // Optional[0]
lst.stream().findFirst().get()        // 0

lst.parallelStream().findAny().get() // 56,65,… nejednoznačné

lst.stream().min(Integer::compare).get()           // 0
lst.stream().min(Integer::compare).isPresent()     // true
lst.stream().max(Integer::compare).get()           // 99

lst.stream().map(i->i%10).sorted().forEach(System.out::print);
000000000011111111112222222222333333333344444444455555555566666
6666667777777778888888889999999999

lst.stream().map(i->i%10).distinct().forEach(System.out::print);
0123456789
```

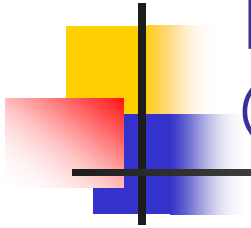
Lenivosť
(laziness)

```
lst.stream().map(e -> { System.out.print(e); return e+e;})

lst.stream().filter(e -> {System.out.print(e);return true;});

lst.stream().map(e -> { System.out.print(e); return e+e;}).
       findFirst().get()                          // 0

lst.stream().map(e -> { System.out.print(e); return e+e;}).
       collect(Collectors.toList());
```

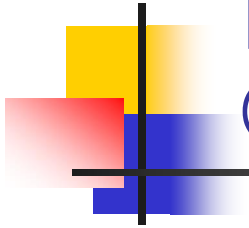


ParallelStream

(komutatívnosť)



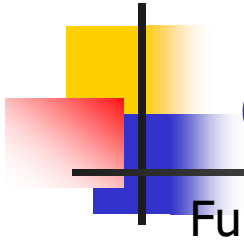
```
lst.parallelStream().
   map(e -> e+e).
   filter(e -> (e % 3 > 0)).
   forEach(e -> System.out.println(e))

lst.parallelStream().
   filter(e -> (e % 3 > 0)).
   map(e -> e+e).
   forEach(e -> System.out.println(e))

lst.parallelStream().
   map(e -> e+e).
   filter(e -> (e % 3 > 0)).
   collect(Collectors.toList()).size()  // koľko je výsledok
```



ParallelStream

(skladanie funkcií)

```
lst.parallelStream().
   map(e -> f1(e)).  // čo vieme povedať o kompozícii ?
   map(e -> f2(e)).
   collect(Collectors.toList())

lst.parallelStream().
   map(e -> f2(f1(e))).  // čo vieme povedať o kompozícii ?
   collect(Collectors.toList())

static Integer f1(Integer e) { return e+e; }
static Integer f2(Integer e) { return 5*e; }
```



ParallelStream

(funkcie so side-effect)

Funkcie poznáme *slušné* a iné:

Slušná funkcia (referenčne transparentná) vždy pre rovnaký vstup vráti rovnaký výsledok, t.j. nerobí žiaden side-effect, nepoužíva globálnu premennú, súbor, ...

Programovací jazyk je *slušný*, ak v ňom môžete písať len slušné funkcie.

Príklad (neslušný):

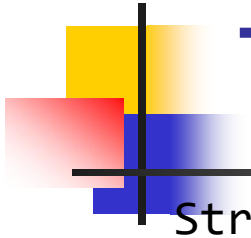
```
lst.parallelStream().
   map(e->funWithSideEffect(e)).
   filter(e -> (e % 3 > 0)).
   sorted().
   collect(Collectors.toList());

Integer globalVariable = 0;
Integer funWithSideEffect(Integer n) {
   return n+n + (++globalVariable);
}
```

Globálne premenné

(sú identifikovaná *smrť*)

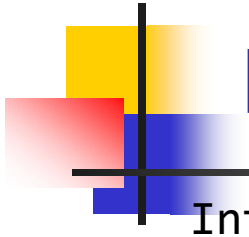
Java(TM) Platform SE binary

```
jshell> globalVariable = 0
globalVariable ==> 0

jshell> lst.parallelStream().
   ...> map(e->funWithSideEffect(e)).
   ...>    filter(e -> (e % 3 > 0)).
   ...> sorted().
   ...> collect(Collectors.toList());
$27 ==> [47, 73, 83, 112, 115, 115, 118, 118, 118, 121, 122, 122, 125, 125, 127, 127, 128, 13
5, 146, 149, 152, 166, 167, 169, 170, 172, 175, 175, 179, 179, 182, 185, 185, 194, 220, 224,

jshell> globalVariable = 0
globalVariable ==> 0

jshell> lst.parallelStream().
   ...> map(e->funWithSideEffect(e)).
   ...>    filter(e -> (e % 3 > 0)).
   ...> sorted().
   ...> collect(Collectors.toList());
$29 ==> [34, 38, 41, 44, 46, 47, 47, 49, 50, 50, 53, 53, 100, 103, 107, 110, 113, 115, 118, 1
51, 152, 154, 154, 155, 157, 158, 158, 161, 161, 163, 163, 166, 166, 169, 169, 184, 187, 190,
 241, 241, 242, 244]

jshell>
```



Trochu novej syntaxe

```
Stream.of(0,1,2,3,4,5,6,7,9).collect(Collectors.toList())
[0, 1, 2, 3, 4, 5, 6, 7, 9]

Stream.of("Palo", "Peter", "Jano", "Jana").collect(Collectors.toList())
[Palo, Peter, Jano, Jana]

Arrays.stream(new Integer[]{0,1,2,3,4,5,6,7,8,9}).collect(Collectors.t
[0, 1, 2, 3, 4, 5, 6, 7, 9]

IntStream.range(0,100).forEach(e -> System.out.print(e));
01234567891011121314151617181920212223 24…

Map<Integer, List<Integer>>mapa = lst.parallelStream().collect(
   Collectors.groupingBy( e -> (String.valueOf(e).length()) ));
{1=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], 2=[10, 11, 12, ... , 94, 95, 96, 97, 98, 99]}
mapa.forEach((len, list) -> System.out.println(len + ", "+ list));
1, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
2, [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, …
```



mapToObj

```
IntStream.range(0,10).mapToObj(e -> (char)(64+e)).
   forEach(System.out::print);
```

```
@ABCDEFGHI
```

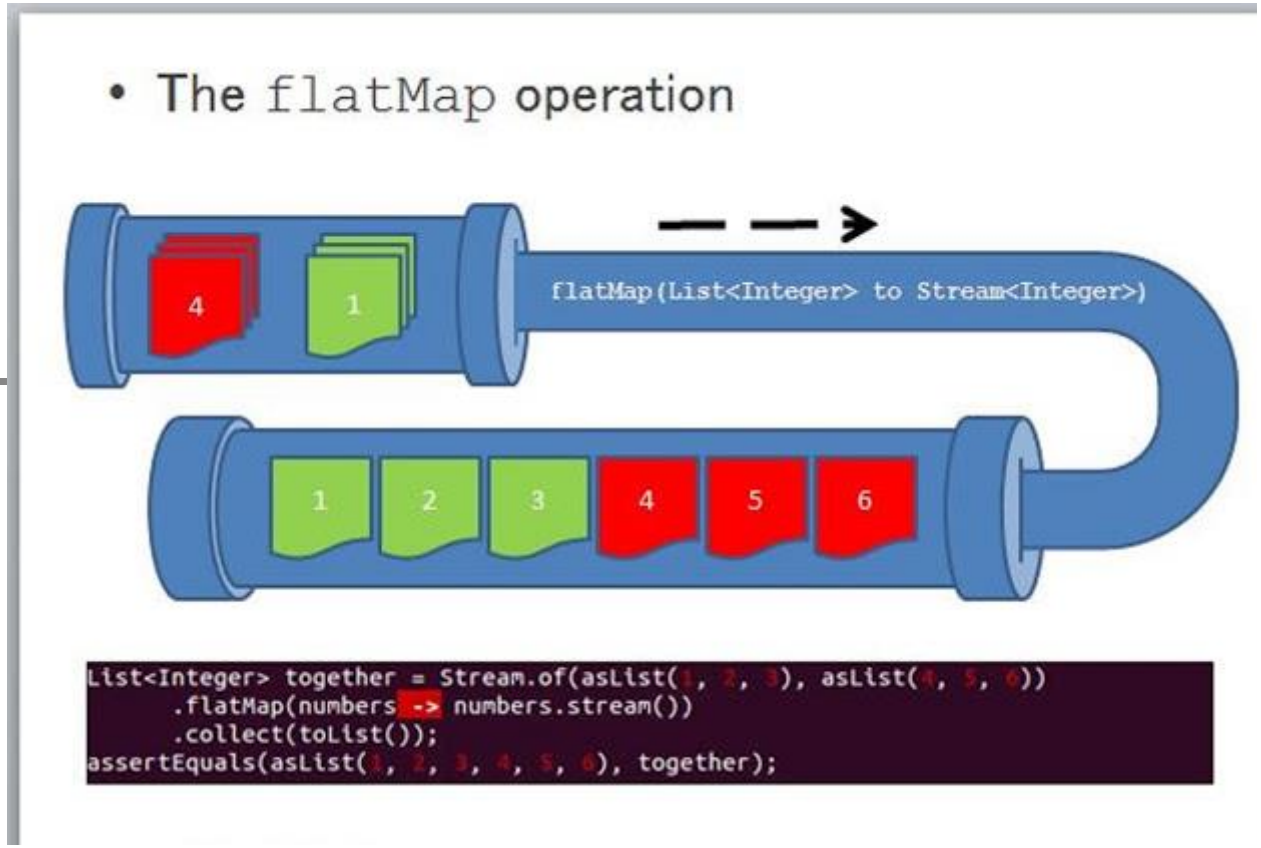
```
IntStream.range(0,10).
   mapToObj(e -> IntStream.range(0, e)).
   forEach(r -> System.out.print(r.count()));
```

```
0123456789
```

```
IntStream.range(0,10).
   mapToObj(e -> IntStream.range(0, e)).
   forEach(r -> System.out.println(
        r.boxed().collect(Collectors.toList())));
```

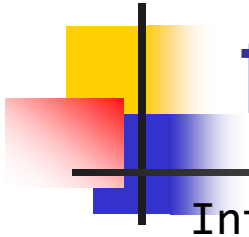
```
[]
[0]
[0, 1]
[0, 1, 2]
[0, 1, 2, 3]
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4, 5]
[0, 1, 2, 3, 4, 5, 6]
```

flatMap




```
List<List<String>> l2 = List.of(
        List.of("Palo", "Jana"),
        List.of("Peter", "Kamil", "Martina"));
[[Palo, Jana], [Peter, Kamil, Martina]]
l2.stream().flatMap(lst ->lst.stream()).
        collect(Collectors.toList());
[Palo, Jana, Peter, Kamil, Martina]
```

http://www.java67.com/2016/03/how-to-use-flatmap-in-java-8-stream.html



flatMap

```
IntStream.range(0,10).
   flatMap(e -> IntStream.range(0, e)).
   forEach(System.out::print);
0010120123012340123450123456012345670123456 78

IntStream.range(0,10).
   flatMap(e -> IntStream.range(0, e).
                        filter(i->i%2==0)).
   forEach(System.out::print);
000202024024024602468 02468
```

Binárne vektory {0,1}

(klasické riešenie)

```
List<String> binaries(int n) {
   if (n == 0) {
        return Arrays.asList("");
   } else {
        List<String> result = new ArrayList<>();
        for (String s : binaries(n-1)) {
               result.add(s + "0");
               result.add(s + "1");
        }
        return result;
   }
}
binaries(4)
```

[0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111]

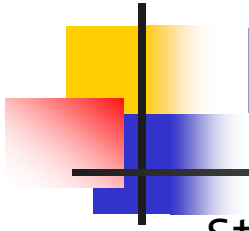
Počet = 2^n

Binárne vektory {0,1}

(streamové riešenie)

Počet = 2^n

```
Stream<String> binaries1(int n) {
   if (n == 0) {
        return Stream.of("");
   } else {
        return
               binaries1(n-1).
               flatMap(s -> Stream.of(s + "0", s + "1"));
   }
}
binaries1(4).collect(Collectors.toList())
[0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111]
```



Permutácie

```
perms(4).collect(Collectors.toList())
[4321, 3421, 3241, 3214, 4231, 2431,
2341, 2314, 4213, 2413, 2143, 2134,
4312, 3412, 3142, 3124, 4132, 1432,
1342, 1324, 4123, 1423, 1243, 1234]
```

Počet = n!

```
Stream<String> perms(int n) {
   if (n <= 0) {
        return Stream.of("");
   } else {
        return
              perms(n-1).
              flatMap(s->IntStream.range(0, n).
                    mapToObj(i -> insert(i, n, s)) );
   }
}
String insert(int i, int n, String s) {
   return      s.substring(0,i) +
               String.valueOf(n) +
               s.substring(i, s.length());
}
```

Kombinácie bez opakovania

Počet = n nad k

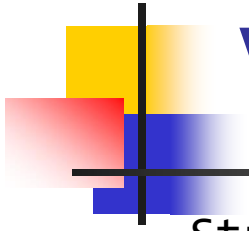
```
Stream<String> kbo(int k, int n) {
   if (k > n) {
        return Stream.of();
   } else if (k == 0) {
        return Stream.of("");
   } else {
        return Stream.concat(
                kbo(k, n-1),
                kbo(k-1, n-1).map(s -> s + String.valueOf(n-1)));
   }
}
kbo(3,6).collect(Collectors.toList())
```

```
[012, 013, 023, 123, 014, 024, 124, 034, 134, 234, 015, 025, 125, 035, 135, 235, 045, 145, 245, 345]
```

Kombinácie s opakovaním

Počet = (n+k-1) nad k

```
Stream<String> kso(int k, int n) {
   if (n == 0) {
        return Stream.of();
   } else if (k == 0) {
        return Stream.of("");
   } else {
        return Stream.concat(
                kso(k, n-1),
                kso(k-1, n).map(s -> s + String.valueOf(n-1)));
   }
}
kso(2,6).collect(Collectors.toList())
[01, 11, 02, 12, 22, 03, 13, 23, 33, 04, 14, 24, 34, 44, 05, 15, 25, 35, 45, 55]
```



Variácie s opakovaním

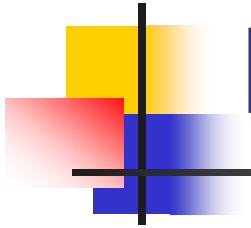
Počet = n^k

```
Stream<String> kso(int k, int n) {
   if (k > n) {
        return Stream.of();
   } else if (k == 0) {
        return Stream.of("");
   } else {
        return Stream.concat(
                kso(k, n-1),
                kso(k-1, n).map(s -> s + String.valueOf(n-1)));
   }
}
kso(2,6).collect(Collectors.toList())
[01, 11, 02, 12, 22, 03, 13, 23, 33, 04, 14, 24, 34, 44, 05, 15, 25, 35, 45, 55]
```

Variácie bez opakovania

Počet = n(n-1)…(n-k+1)

```
Stream<String> vbo(int k, int n) {
   if (k > n) {
        return Stream.of();
   } else if (k == 0) {
        return Stream.of("");
   } else {
        return Stream.concat(
                vbo(k, n-1),
                vbo(k-1, n-1).
                flatMap(s -> IntStream.range(0, k).
                        mapToObj(i -> insert(i, n-1, s))));
   }
}
vbo(3,4).collect(Collectors.toList())
[210, 120, 102, 201, 021, 012, 310, 130, 103, 301, 031, 013, 320, 230, 203, 302, 032, 023, 321,
    231, 213, 312, 132, 123]
```



Bonmot

Ak by vám (v 1.semestri) neprezradili priradenie (=) a cyklus (for/while),

tak tu máme spústu šikovných funkcionálnych programátorov...