Schleifen zu Streams

Sven Eric Panitz

Hochschule Rhein-Main
Standort Wiesbaden

```
int count(Iterable<String> xs){
  int result = 0;
  for (String x : xs) {
    result = result + 1;
  }
  return result;
}
```

```
int count(Iterable<String> xs){
   int result = 0;
   for (String x : xs) {
      result = result + 1;
   }
   return result;
}
```

```
int count(Iterable<String> xs){
  int result = 0;
  for (String x : xs) {
    result = result + 1;
  }
  return result;
}
```

```
int count(Iterable<String> xs){
  int result = 0;
  for (String x : xs) {
    result = result + 1;
  }
  return result;
}
```

```
int count(Iterable<String> xs){
  int result = 0;
  for (String x : xs) {
    result = result + 1;
  }
  return result;
}
```

```
int count(Iterable<String> xs){
  int result = 0;
  for (String x : xs) {
    result = result + 1;
  }
  return result;
}
```

- Startwert

```
int count(Iterable<String> xs){
  int result = 0;
  for (String x : xs) {
    result = result + 1;
  }
  return result;
}
```

- Startwert
- Iteration

```
int count(Iterable<String> xs){
  int result = 0;
  for (String x : xs) {
    result = result + 1;
  }
  return result;
}
```

- Startwert
- Iteration
- Aufaddieren

```
    chrNo(Iterable<String> xs){
       result =  ;
  for (String x : xs) {
   result =
  }
  return result;
}
```

```
int chrNo(Iterable<String> xs){
   int result =  ;
   for (String x : xs) {
    result =
   }
   return result;
}
```

```
int chrNo(Iterable<String> xs){
   int result = 0;
   for (String x : xs) {
    result =
   }
   return result;
}
```

```
int chrNo(Iterable<String> xs){
   int result = 0;
   for (String x : xs) {
    result = result + x.length();
   }
   return result;
}
```

```
        ap(Iterable<String> xs){
           result =     ;
  for (String x : xs) {
   result =
  }
  return result;
}
```

```
String ap(Iterable<String> xs){
   String result =    ;
   for (String x : xs) {
    result =
   }
   return result;
}
```

```
String ap(Iterable<String> xs){
   String result = " ";
   for (String x : xs) {
    result =
   }
   return result;
}
```

```
String ap(Iterable<String> xs){
   String result = " ";
   for (String x : xs) {
    result = result + " " + x;
   }
   return result;
}
```

```
        lo(Iterable<String> xs){
          result =  ;
  for (String x : xs) {
   result
   =


  }
  return result;
}
```

```
String lo(Iterable<String> xs){
 String result =  ;
 for (String x : xs) {
  result
  =


 }
 return result;
}
```

```
String lo(Iterable<String> xs){
 String result = "";
 for (String x : xs) {
  result
  =


 }
 return result;
}
```

```
String lo(Iterable<String> xs){
 String result = "";
 for (String x : xs) {
  result
  = result.length() > x.length()
    ?result:x;
 }
 return result;
}
```

```
        co(String y
           ,Iterable<String> xs){

          result =       ;
 for (String x : xs) {
  result =
 }
 return result;
}
```

```
boolean co(String y
           ,Iterable<String> xs){

 boolean result =       ;
 for (String x : xs) {
  result =
 }
 return result;
}
```

```
boolean co(String y
           ,Iterable<String> xs){

 boolean result = false;
 for (String x : xs) {
  result =
 }
 return result;
}
```

```
boolean co(String y
           ,Iterable<String> xs){

 boolean result = false;
 for (String x : xs) {
  result = result || x.equals(y);
 }
 return result;
}
```

```
public static <A  >
   falten
     ( Iterable<A> xs
     ,    result
     ,                                  ) {

   for (A x : xs) {
     result =
   }
   return result;
}
```

```
public static <A,B>
B falten
    ( Iterable<A> xs
    , B result
    ,                                ) {

  for (A x : xs) {
    result =
  }
  return result;
}
```

```
public static <A,B>
B falten
    ( Iterable<A> xs
    , B result
    , BiFunction<B, A, B> comp) {

  for (A x : xs) {
    result = comp.apply(result,x);
  }
  return result;
}
```

```java
static int count(Iterable<String> xs){
  int result = 0;
  for (String x : xs) {
    result = result + 1;
  }
  return result;
}

static int count(Iterable<String> xs){
 return falten(xs,  , (result,x) ->          );
}
```

```java
static int count(Iterable<String> xs){
  int result = 0;
  for (String x : xs) {
    result = result + 1;
  }
  return result;
}

static int count(Iterable<String> xs){
 return falten(xs, 0, (result,x) -> result + 1);
}
```

```java
static int chrNo(Iterable<String> xs){
  int result = 0;
  for (String x : xs) {
   result = result + x.length();
  }
  return result;
}
```

```java
 static int chrNo(Iterable<String> xs) {
   return falten
          ( xs,
          , (result,x) ->                    ) ;
 }
```

```java
static int chrNo(Iterable<String> xs){
  int result = 0;
  for (String x : xs) {
   result = result + x.length();
  }
  return result;
}
```

```java
 static int chrNo(Iterable<String> xs) {
   return falten
          ( xs, 0
          , (result,x) -> result + x.length()) ;
 }
```

```
String ap(Iterable<String> xs){
   String result = " ";
   for (String x : xs) {
    result = result + " " + x;
   }
   return result;
}
static String ap(Iterable<String> xs) {
   return falten
         ( xs,
         , (result,x) ->                    ) ;
}
```

```
String ap(Iterable<String> xs){
   String result = " ";
   for (String x : xs) {
    result = result + " " + x;
   }
   return result;
}
static String ap(Iterable<String> xs) {
   return falten
         ( xs, " "
         , (result,x) -> result + " " + x) ;
}
```

```
String lo(Iterable<String> xs){
 String result = "";
 for (String x : xs) {
  Result = result.length() > x.length() ? result : x;
 }
 return result;
}

static String lo(Iterable<String> xs) {
  return falten
   ( xs,
   , (result,x)->

   );
}
```

```
String lo(Iterable<String> xs){
 String result = "";
 for (String x : xs) {
  Result = result.length() > x.length() ? result : x;
 }
 return result;
}

static String lo(Iterable<String> xs) {
  return falten
   ( xs, ""
   , (result,x)->
           result.length()>x.length()?result:x
   );
}
```

```
boolean co(String y
           ,Iterable<String> xs){
 boolean result = false;
 for (String x : xs) {
  result = result || x.equals(y);
 }
 return result;
}
static boolean co( String y,
                   , Iterable<String> xs) {
   return falten
    ( xs,
    , (result,x)->
    );
}
```

```
boolean co(String y
           ,Iterable<String> xs){
 boolean result = false;
 for (String x : xs) {
  result = result || x.equals(y);
 }
 return result;
}
static boolean co( String y,
                   , Iterable<String> xs) {
   return falten
    ( xs, false
    , (result,x)-> result || x.equals(y)
    );
}
```

compact1, compact2, compact3

java.util.stream

Interface Stream<T>

Type Parameters:

`T - the type of the stream elements`

All Superinterfaces:

`AutoCloseable, BaseStream<T,Stream<T>>`

```
public interface Stream<T>
extends BaseStream<T,Stream<T>>
```

`Optional<T>`	`reduce(BinaryOperator<T> accumulator)` Performs a **reduction** on the elements of this stream, using an **associative** accumulation function, and returns an `Optional` describing the reduced value, if any.
`T`	`reduce(T identity, BinaryOperator<T> accumulator)` Performs a **reduction** on the elements of this stream, using the provided identity value and an **associative** accumulation function, and returns the reduced value.
`<U> U`	`reduce(U identity, BiFunction<U,? super T,U> accumulator, BinaryOperator<U> combiner)` Performs a **reduction** on the elements of this stream, using the provided identity, accumulation and combining functions.

T	**reduce**(T identity, **BinaryOperator**<T> accumulator) Performs a **reduction** on the elements of this stream, using the provided identity value and an **associative** accumulation function, and returns the reduced value.

```
String concat(List<String> xs){
  return xs
    .stream()
    .reduce
      (""
      ,(result,x)->result+" "+x);
}
```

<U> U	**reduce**(U identity, **BiFunction**<U,? super **T**,U> accumulator, **BinaryOperator**<U> combiner) Performs a **reduction** on the elements of this stream, using the provided identity, accumulation and combining functions.

```
<A> boolean contains
                  (List<A> xs, A y){
  return xs.stream()
    .reduce
    ( false
    , (result,x)
        -> result||y.equals(x)
    ,(a,b) -> a || b);
}
```

```
<A> boolean contains
                  (List<A> xs, A y){
  return xs.parallelStream()
    .reduce
    ( false
    , (result,x)
        -> result||y.equals(x)
    ,(a,b) -> a || b);
}
```

```java
import java.math.BigInteger;
import java.util.stream.Stream;
class BigInt8{
  static final BigInteger FIVE = new BigInteger("5");
  public static void main(String[] args){
    //mache einen unendlichen Strom der Zahlen x_1 x_2 x_3 x_4 x_5 ...
    // mit x_1 = 1 und x_{n+1} = x_n + 1
    Stream.iterate(BigInteger.ONE, n -> n.add(BigInteger.ONE))
    // mache den Stream parallel
      .parallel()
    // filtere aus diesem alle Zahlen n heraus, für die gilt
    // gcd((n+1)^5+5,n^5+5) > 1
      .filter(n -> n.add(BigInteger.ONE).pow(5).add(FIVE)
                    .gcd(n.pow(5).add(FIVE))
                    .compareTo(BigInteger.ONE) > 0)
    //Die ersten 10 Elemente
      .limit(10)
    //drucke jeden Wert des gefilterten Stroms auf der Konsole in eigener Zeile
      .forEach(n -> System.out.println(n));
  }
}
```