MATHEMA

Über den Umgang mit Streams

Java Forum Stuttgart

6. Juli 2017

```
<T> boolean allMatch(Predicate<T> p) {
    boolean result = true;
    for (T element : this) {                    // Iterable<T>
        result &= p.test(element);
    }
    return result;
}
```

```
<T> boolean allMatch(Predicate<T> p) {
    for (T element : this) {
        if (!p.test(element)) {
            return false;
        }
    }
    return true;
}
```

```
… allMatch(new Predicate<Person>() {
    boolean test(Person p) {
        return p.isEmployed(now);
    }
}) …
```

```
… allMatch(new Predicate<Person>() {
-    boolean test(Person p) -> {
         return p.isEmployed(now);
     }
}) …
```

```
… allMatch((Person p) -> {
        return p.isEmployed(now);
    }
) …
```

… allMatch((Person *p*) −> *p*.isEmployed(*now*)) …

… allMatch(~~(Person~~ *p*~~)~~ -> *p*.isEmployed(*now*)) …

… allMatch(*p* −> *p*.isEmployed(*now*)) …

final int *now* = … ;

… allMatch(*p* −> *p*.isEmployed(*now*)) …

int *now* = … ;

… allMatch(*p* −> *p*.isEmployed(*now*)) …

int *now* = … ;

… allMatch(*p* -> *p*.isEmployed(*now*)) …

now = … ; // No! (Muss *effective final* sein)

	.forAll(Consumer<T>)	void
	.allMatch(Predicate<T>)	boolean
	.count()	long
	.max(Comparator<T>)	Optional<T>
c.stream()	.map(Function<T, R>)	Stream<R>
	.filter(Predicate<T>)	Stream<T>
	.limit(long)	Stream<T>
	.reduce(BinaryOperator<T>)	Optional<T>
	.collect(Collector<T,A,R>)	R

Eigentlich:

Fertig!

Siehe auch: java.util, java.util.stream (und java.util.function)

Spaß mit:

Poker

- Royal Flush

- Straight Flush

- Four of a Kind

- Full House

- Flush

- Straight

- Three of a Kind

- Two Pair

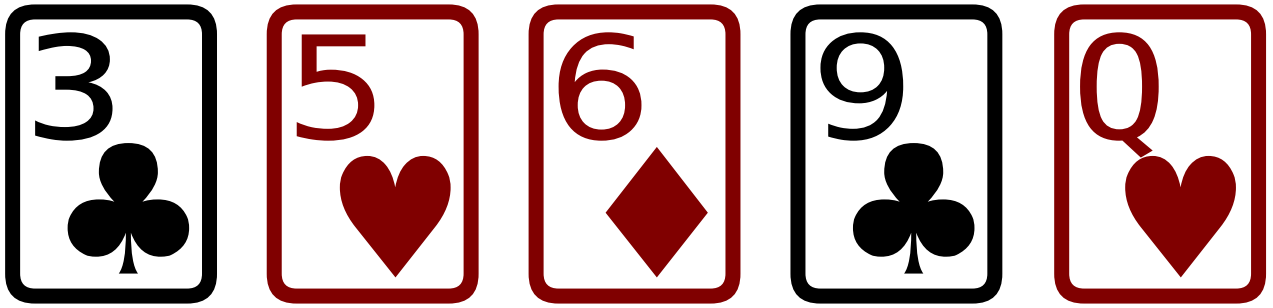
- One Pair

- No Pair / High Card

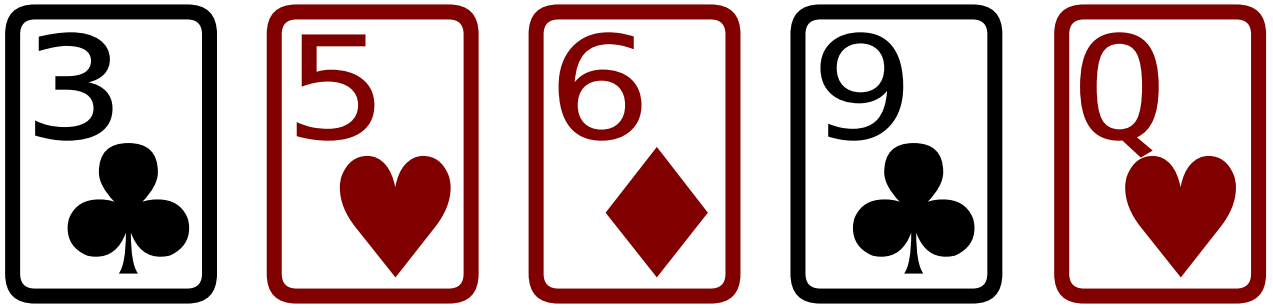
```
… c.stream().collect(
     Collectors.groupingBy(
          Employee::getDepartment      // Employee -> Key
     )
) …
```

```
R container = collector.supplier().get();
for (T element : data) {
     collector.accumulator().accept(container, element);
}
return collector.finisher().apply(container);
```

No Pair / High Card

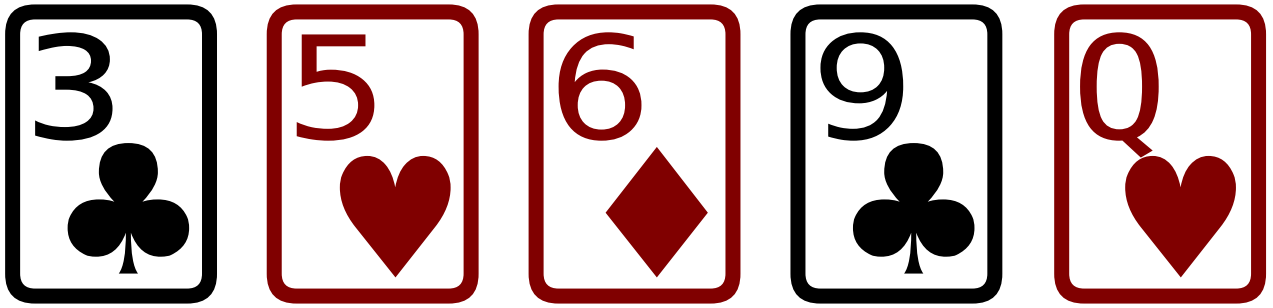


No Pair / Highest Card



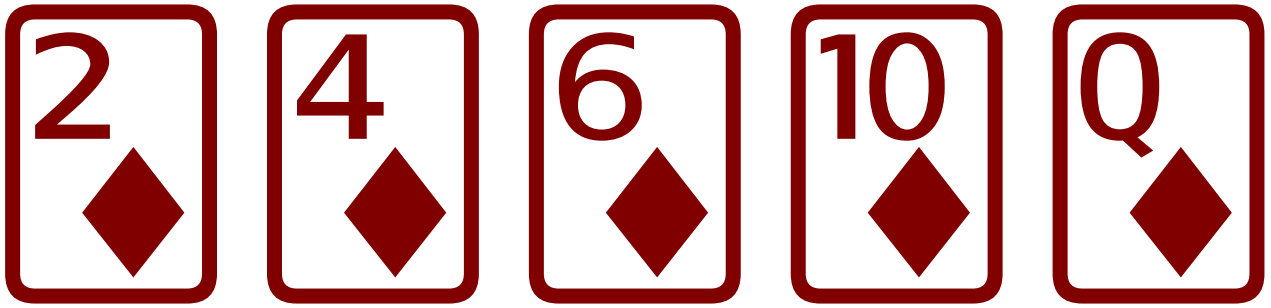
$$\text{differentRank} := (\mathit{hand} : \text{Hand}) \mapsto$$
$$\forall(\mathit{card} \in \mathit{hand})\ \nexists(c \in \mathit{hand})\ (\mathit{card}.\text{rank} = c.\text{rank})$$

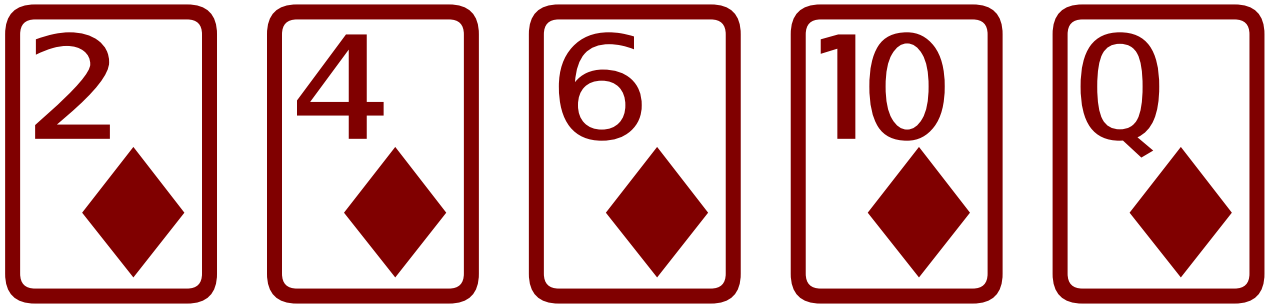
No Pair / Highest Card



$$\text{differentRank} := (\mathit{hand} : \text{Hand}) \mapsto$$
$$\forall(\mathit{card} \in \mathit{hand})\ \nexists(c \in \mathit{hand},\ c \neq \mathit{card})\ (\mathit{card}.\text{rank} = c.\text{rank})$$

Flush





$$\text{sameColor} := (\mathit{hand} : \text{Hand}) \mapsto$$
$$\forall(\mathit{card} \in \mathit{hand})\ \forall(c \in \mathit{hand})\ (\mathit{card}.\text{suit} = c.\text{suit})$$

`c.stream().forEach(U::f) // x -> U.f(x)`

```
for (int i = 0; i < n; i++) {
    f(c.get(i));
}
```

```
c.stream().forEach(
    x -> c'.stream().forEach(y -> f(x, y))
)
```

```
for (int i = 0; i < n_x; i++) {
    for (int j = 0; j < n_y; j++) {
        f(c.get(i), c'.get(j));
    }
}
```

```
c.stream().forEach(
    x -> c′.stream().forEach(
        y -> c″.stream().forEach(z -> f(x, y, z))
    )
)
```

```
for (int i = 0; i < n_x; i++) {
    for (int j = 0; j < n_y; j++) {
        for (int k = 0; k < n_z; k++) {
            f(c.get(i), c′.get(j), c″.get(k));
        }
    }
}
```

$O(1)$	konstant/beschränkt
$O(\log n)$	logarithmisch
$O(\sqrt{n})$	„wurzelig“
$O(n)$	linear
$O(n \log n)$	superlinear
$O(n^2)$	quadratisch
$O(n^3)$	kubisch
$O(n^k)$	polynomiell
$O(2^n)$	exponentiell
$O(n!)$	faktoriell

Dauer in Abhängigkeit von der Anzahl

n	1	10	100	1000
$O(1)$	1 s	1 s	1 s	1 s
$O(\log n)$	1 s	2 s	3 s	4 s
$O(n)$	1 s	10 s	100 s	1000 s
$O(n \log n)$	1 s	20 s	300 s	4000 s
$O(n^2)$	1 s	100 s	2,8 h	277,8 h
$O(n^3)$	1 s	1000 s	277,8 h	31 a
$O(2^n)$	1 s	1024 s	10^{22} a	

Dauer in Abhängigkeit von der Anzahl

n	1	10	100	1000
$O(1)$	1 ns	1 ns	1 ns	1 ns
$O(\log n)$	1 ns	2 ns	3 ns	4 ns
$O(n)$	1 ns	10 ns	100 ns	1 µs
$O(n \log n)$	1 ns	20 ns	300 ns	4 µs
$O(n^2)$	1 ns	100 ns	10 µs	1 ms
$O(n^3)$	1 ns	1 µs	1 ms	1 s
$O(2^n)$	1 ns	~1 µs	~10^{13} a	

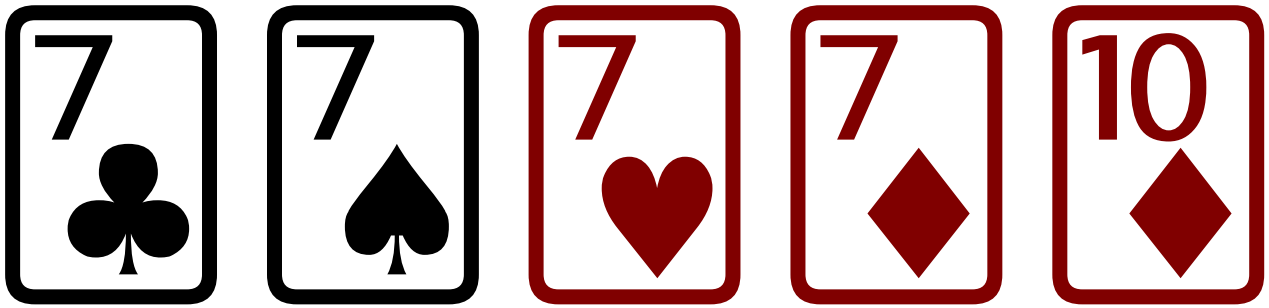
Wieviele Schritte pro Sekundezu schaffen sind

Komplexität	1 ns	1 s
$O(1)$	1	∞
$O(\log n)$	1	$10^{1\,000\,000\,000}$
$O(n)$	1	10^9
$O(n^2)$	1	31 623
$O(n^3)$	1	1000
$O(2^n)$	1	30

Vervielfachung der Rechenleistung

Komplexität	zweifach	zehnfach
$O(1)$	∞	∞
$O(\log n)$	$10 \cdot n$	$10^9 \cdot n$
$O(n)$	$2 \cdot n$	$10 \cdot n$
$O(n^2)$	$1{,}41 \cdot n$	$3{,}16 \cdot n$
$O(n^3)$	$1{,}26 \cdot n$	$2{,}15 \cdot n$
$O(2^n)$	$n + 1$	$n + 3{,}3$

Four of a Kind



- fourOfSameRank := (*hand* : Hand) ↦

  $\exists(set \in \wp_4(\text{hand})$ **where** $card := \text{arb}(set))$

  $\forall(c \in \text{set}, c \neq \text{card})\ (c.\text{rank} = \text{card.rank})$

Gibt es denn wirklich noch irgendwelche

Fragen?

Na dann:

Vielen Dank!