

COPILOT

THE `Collectors` CLASS IN JAVA 8 PROVIDES A VARIETY OF METHODS TO PERFORM REDUCTION OPERATIONS ON STREAMS. HERE ARE SOME OF THE MOST COMMONLY USED METHODS:

1. `toList()`

COLLECTS ELEMENTS INTO A `List`.

EXAMPLE:

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class ToListExample {
    public static void main(String[] args) {
        List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
        List<String> result = names.stream().collect(Collectors.toList());
        System.out.println(result); // Output: [Alice, Bob, Charlie]
    }
}
```

2. `toSet()`

COLLECTS ELEMENTS INTO A `Set`.

EXAMPLE:

```
import java.util.Arrays;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

public class ToSetExample {
    public static void main(String[] args) {
        List<String> names = Arrays.asList("Alice", "Bob", "Charlie", "Alice");
        Set<String> result = names.stream().collect(Collectors.toSet());
        System.out.println(result); // Output: [Alice, Bob, Charlie]
    }
}
```

3. `toMap()`

COLLECTS ELEMENTS INTO A `Map`.

EXAMPLE:

```
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class ToMapExample {
    public static void main(String[] args) {
        List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
        Map<String, Integer> result = names.stream()
            .collect(Collectors.toMap(name -> name, String::length));
    }
}
```

```

    System.out.println(result); // Output: {Alice=5, Bob=3, Charlie=7}
}
}

```

4. JOINING()

CONCATENATES THE ELEMENTS OF A STREAM INTO A SINGLE String.

EXAMPLE:

```

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class JoiningExample {
    public static void main(String[] args) {
        List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
        String result = names.stream().collect(Collectors.joining(", "));
        System.out.println(result); // Output: Alice, Bob, Charlie
    }
}

```

5. GROUPINGBY()

GROUPS ELEMENTS BY A CLASSIFIER FUNCTION.

EXAMPLE:

```

import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class GroupingByExample {
    public static void main(String[] args) {
        List<String> names = Arrays.asList("Alice", "Bob", "Charlie", "David", "Alice");
        Map<String, Long> result = names.stream()
            .collect(Collectors.groupingBy(name -> name, Collectors.counting()));
        System.out.println(result); // Output: {Alice=2, Bob=1, Charlie=1, David=1}
    }
}

```

6. PARTITIONINGBY()

PARTITIONS ELEMENTS INTO TWO GROUPS BASED ON A PREDICATE.

EXAMPLE:

```

import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class PartitioningByExample {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
        Map<Boolean, List<Integer>> result = numbers.stream()
            .collect(Collectors.partitioningBy(n -> n % 2 == 0));
        System.out.println(result); // Output: {false=[1, 3, 5, 7, 9], true=[2, 4, 6, 8, 10]}
    }
}

```

7. COUNTING()

COUNTS THE NUMBER OF ELEMENTS IN THE STREAM.

EXAMPLE:

```
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class CountingExample {
    public static void main(String[] args) {
        List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
        long count = names.stream().collect(Collectors.counting());
        System.out.println(count); // Output: 3
    }
}
```

8. SUMMARIZINGINT(), SUMMARIZINGDOUBLE(), SUMMARIZINGLONG()

COLLECTS STATISTICS, SUCH AS COUNT, SUM, MIN, AVERAGE, AND MAX.

EXAMPLE:

```
import java.util.Arrays;
import java.util.IntSummaryStatistics;
import java.util.List;
import java.util.stream.Collectors;

public class SummarizingIntExample {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
        IntSummaryStatistics stats = numbers.stream().collect(Collectors.summarizingInt(Integer::intValue));
        System.out.println(stats); // Output: IntSummaryStatistics{count=5, sum=15, min=1, average=3.000000, max=5}
    }
}
```

9. REDUCING()

PERFORMS A REDUCTION ON THE ELEMENTS OF THE STREAM USING AN ASSOCIATIVE ACCUMULATION FUNCTION.

EXAMPLE:

```
import java.util.Arrays;
import java.util.List;
import java.util.Optional;

public class ReducingExample {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
        Optional<Integer> sum = numbers.stream().collect(Collectors.reducing(Integer::sum));
        sum.ifPresent(System.out::println); // Output: 15
    }
}
```

10. COLLECTINGANDTHEN()

ADAPTS A COLLECTOR TO PERFORM AN ADDITIONAL FINISHING TRANSFORMATION.

EXAMPLE:

```
import java.util.Arrays;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

public class CollectingAndThenExample {
    public static void main(String[] args) {
        List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
        Set<String> result = names.stream()
            .collect(Collectors.collectingAndThen(Collectors.toSet(), Collections::unmodifiableSet));
        System.out.println(result); // Output: [Alice, Bob, Charlie]
    }
}
```

THESE ARE SOME OF THE KEY METHODS PROVIDED BY THE `Collectors` CLASS IN JAVA 8. EACH METHOD SERVES A SPECIFIC PURPOSE AND CAN BE USED TO PERFORM VARIOUS TYPES OF REDUCTION OPERATIONS ON STREAMS.

IF YOU HAVE ANY SPECIFIC QUESTIONS OR NEED MORE DETAILED EXAMPLES, FEEL FREE TO ASK!