Collecting results from streams

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming Concepts using Java

Week 9

Collecting values from a stream

- Convert collections into sequences of values — streams

Collecting values from a stream

- Convert collections into sequences of values — streams
- Process a stream as a collection?

Collecting values from a stream

- Convert collections into sequences of values — streams
- Process a stream as a collection?
- `Stream` defines a standard iterator, use to loop through values in a stream

Collecting values from a stream

- Convert collections into sequences of values — streams
- Process a stream as a collection?
- `Stream` defines a standard iterator, use to loop through values in a stream
- Alternatively, use `forEach` with a suitable function

```
mystream.forEach(System.out::println);
```

Collecting values from a stream

- Convert collections into sequences of values — streams
- Process a stream as a collection?
- `Stream` defines a standard iterator, use to loop through values in a stream
- Alternatively, use `forEach` with a suitable function
- Can convert a stream into an array using `toArray()`
  - Creates an array of `Object` by default

```
mystream.forEach(System.out::println);
```

```
Object[] result = mystream.toArray();
```

Collecting values from a stream

- Convert collections into sequences of values — streams
- Process a stream as a collection?
- `Stream` defines a standard iterator, use to loop through values in a stream
- Alternatively, use `forEach` with a suitable function
- Can convert a stream into an array using `toArray()`
  - Creates an array of `Object` by default
- Pass array constructor to get a more specific array type

```
mystream.forEach(System.out::println);
```

```
Object[] result = mystream.toArray();
```

```
String[] result =
   mystream.toArray(String[]::new);
   // mystream.toArray() has type Object[]
```

Storing a stream as a collection

- What if we want to convert the stream back into a collection?

Storing a stream as a collection

- What if we want to convert the stream back into a collection?
- Use `collect()`
  - Pass appropriate factory method from `Collectors`
  - Static method that directly calls a constructor

Storing a stream as a collection

- What if we want to convert the stream back into a collection?
- Use `collect()`
  - Pass appropriate factory method from `Collectors`
  - Static method that directly calls a constructor
- Create a list from a stream

```
List<String> result =
   mystream.collect(Collectors.toList());
```

Storing a stream as a collection

- What if we want to convert the stream back into a collection?
- Use `collect()`
  - Pass appropriate factory method from `Collectors`
  - Static method that directly calls a constructor
- Create a list from a stream
- . . . or a set

```
List<String> result =
   mystream.collect(Collectors.toList());
```

```
Set<String> result =
   mystream.collect(Collectors.toSet());
```

Storing a stream as a collection

- What if we want to convert the stream back into a collection?
- Use `collect()`
  - Pass appropriate factory method from `Collectors`
  - Static method that directly calls a constructor
- Create a list from a stream
- . . . or a set
- To create a concrete collection, provide a constructor

```
List<String> result =
   mystream.collect(Collectors.toList());

Set<String> result =
   mystream.collect(Collectors.toSet());

TreeSet<String> result =
   stream.collect(
     Collectors.toCollection(
       TreeSet::new
     )
   );
```

Stream summaries

- We saw how to reduce a stream to a single result value — `count()`, `max()`, ...
  - In general, need a stream of numbers

Stream summaries

- We saw how to reduce a stream to a single result value — `count()`, `max()`, ...
  - In general, need a stream of numbers
- `Collectors` has methods to aggregate summaries in a single object
  - `summarizingInt` works for a stream of integers
  - Pass function to convert given stream to numbers — here `String::length`
  - Returns `IntSummaryStatistics` that stores count, max, min, sum, average

```
IntSummaryStatistics summary =
  mystream.collect(
     Collectors.summarizingInt(
       String::length)
     );
```

Stream summaries

- We saw how to reduce a stream to a single result value — `count()`, `max()`, ...
  - In general, need a stream of numbers
- `Collectors` has methods to aggregate summaries in a single object
  - `summarizingInt` works for a stream of integers
  - Pass function to convert given stream to numbers — here `String::length`
  - Returns `IntSummaryStatistics` that stores count, max, min, sum, average

```
IntSummaryStatistics summary =
  mystream.collect(
     Collectors.summarizingInt(
       String::length)
     );

double averageWordLength = summary.getAverage()
double maxWordLength = summary.getMax();
```

- Methods to access relevant statistics
  - `getCount()`,`getMax()`, `getMin()`, `getSum()`, `getAverage()`,

Stream summaries

- We saw how to reduce a stream to a single result value — `count()`, `max()`, ...
  - In general, need a stream of numbers
- `Collectors` has methods to aggregate summaries in a single object
  - `summarizingInt` works for a stream of integers
  - Pass function to convert given stream to numbers — here `String::length`
  - Returns `IntSummaryStatistics` that stores count, max, min, sum, average

```
IntSummaryStatistics summary =
  mystream.collect(
     Collectors.summarizingInt(
       String::length)
     );

double averageWordLength = summary.getAverage()
double maxWordLength = summary.getMax();
```

- Methods to access relevant statistics
  - `getCount()`,`getMax()`, `getMin()`, `getSum()`, `getAverage()`,
- Similarly, `summarizingLong()` and `summarizingDouble()` return `LongSummaryStatistics` and `DoubleSummaryStatistics`

Converting a stream to a map

- Convert a stream of `Person` to a map
  - For `Person p`, `p.getID()` is key and `p.getName()` is value

```
Stream<Person> people = ...;

Map<Integer, String> idToName =
   people.collect(
     Collectors.toMap(
       Person::getId,
       Person::getName
     )
   );
```

Converting a stream to a map

- Convert a stream of `Person` to a map
  - For `Person p`, `p.getID()` is key and `p.getName()` is value
- To store entire object as value, use `Function.identity()`

```
Stream<Person> people = ...;

Map<Integer, Person> idToPerson =
   people.collect(
     Collectors.toMap(
       Person::getId,
       Function.identity()
     )
   );
```

Converting a stream to a map

- Convert a stream of `Person` to a map
  - For `Person p`, `p.getID()` is key and `p.getName()` is value
- To store entire object as value, use `Function.identity()`
- What happens if we use name for key and id for value?

```
Stream<Person> people = ...;

Map<String, Integer> nameToID =
   people.collect(
     Collectors.toMap(
       Person::getName,
       Person::getId
     )
   );
```

Converting a stream to a map

- Convert a stream of `Person` to a map
  - For `Person p`, `p.getID()` is key and `p.getName()` is value
- To store entire object as value, use `Function.identity()`
- What happens if we use name for key and id for value?
  - Likely to have duplicate keys — `IllegalStateException`

```
Stream<Person> people = ...;

Map<String, Integer> nameToID =
   people.collect(
     Collectors.toMap(
       Person::getName,
       Person::getId
     )
   );
```

Converting a stream to a map

- Convert a stream of `Person` to a map
  - For `Person p`, `p.getID()` is key and `p.getName()` is value
- To store entire object as value, use `Function.identity()`
- What happens if we use name for key and id for value?
  - Likely to have duplicate keys — `IllegalStateException`
- Provide a function to fix such problems

```
Stream<Person> people = ...;

Map<String, Integer> nameToID =
   people.collect(
     Collectors.toMap(
       Person::getName,
       Person::getId,
       (existingValue, newValue) ->
          existingValue
     )
   );
```

Grouping and partitioning values

- Instead of discarding values with duplicate keys, group them

Grouping and partitioning values

- Instead of discarding values with duplicate keys, group them
- Collect all ids with the same name in a list

```
Stream<Person> people = ...;

Map<String, List<Person>> nameTopersons =
   people.collect(
     Collectors.groupingBy(
       Person::getName
     )
   );
```

Grouping and partitioning values

- Instead of discarding values with duplicate keys, group them
- Collect all ids with the same name in a list
- Instead, may want to partition the stream using a predicate

```
Stream<Person> people = ...;

Map<String, List<Person>> nameTopersons =
   people.collect(
     Collectors.groupingBy(
       Person::getName
     )
   );
```

Grouping and partitioning values

- Instead of discarding values with duplicate keys, group them
- Collect all ids with the same name in a list
- Instead, may want to partition the stream using a predicate
- Partition names into those that start with `A` and the rest
  - Key values of resulting map are `true` and `false`

```
Stream<Person> people = ...;

Map<Boolean, List<Person>> aAndOtherPersons =
   people.collect(
     Collectors.partitioningBy(
       p -> p.getName().substr(0,1).equals("A")
     )
   );

List<Person> startingLetterA =
  aAndOtherPersons.get(true);
```

Summary

- We converted collections into sequences and processed them as streams
- After transformations, we may want to process a stream as a collection
- Use iterators, `forEach()` to process a stream element by element
- Use `toArray()` to convert to an array
- Factory methods in `Collector` allow us to convert a stream back into a collection of our choice
- Can convert an arbitrary stream into a stream of numbers and collect summary statistics
- Can convert a stream into a map
- Can group values by a key, or partition by a predicate