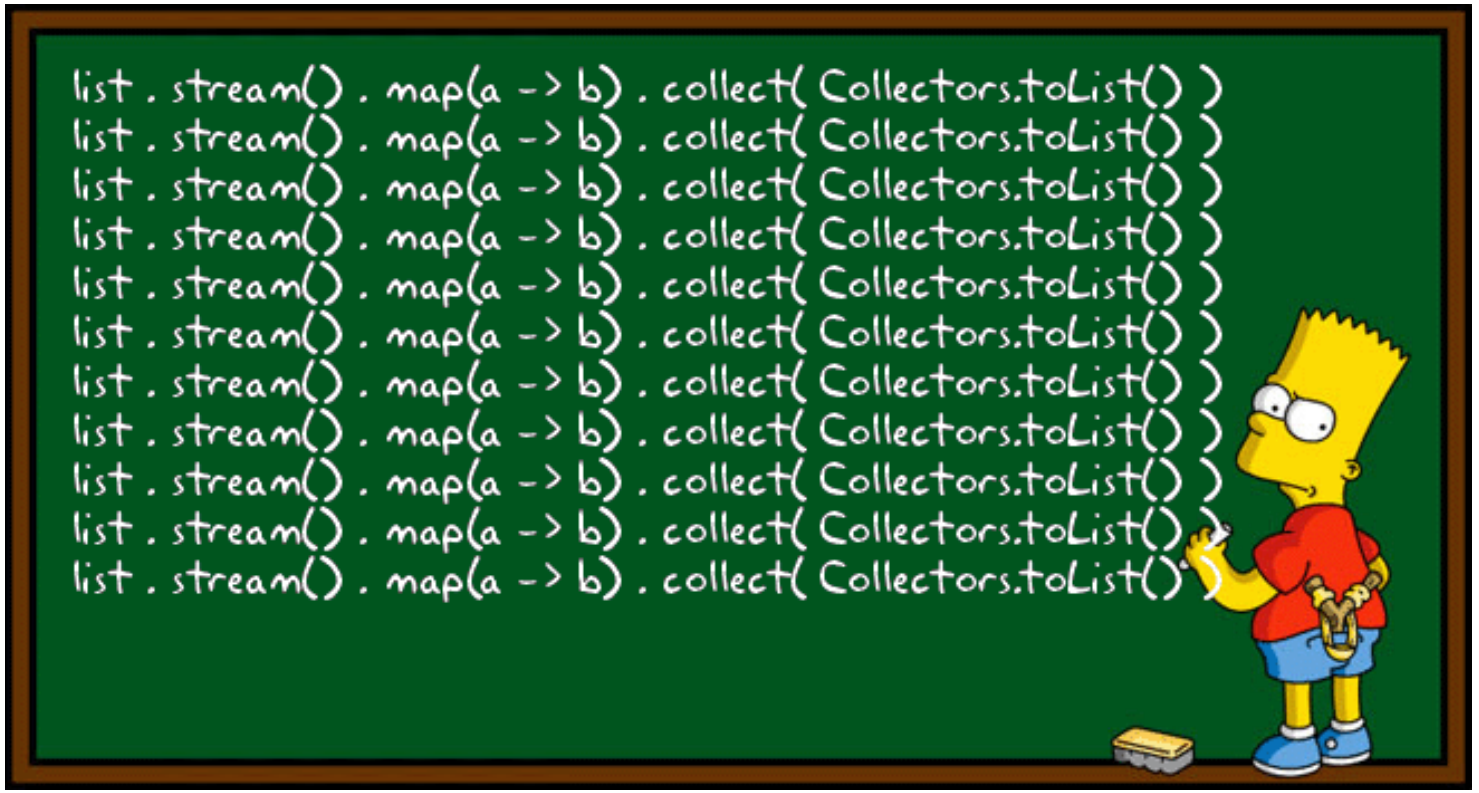
streamtuples

A fix for the ”single return value” Java 8+ Stream painpoint.

Thorbjørn Ravn Andersen
2018-03-15
tra@kb.dk

This!



https://twitter.com/lukaseder/status/639802749878697984

Why?

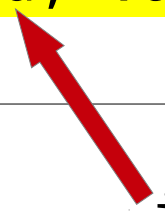
- Java methods can only return a single value.
- Java does not have pairs/tuples/value types to wrap multiple values into a single value.
- Java cannot ”auto-unbox” an object into satisfying a lambda expression with more than one parameter.

Result: Streams tend to degenerate into single-valued and independent items. Not much can be done with these except collecting them into a collection.

Why multivalued?

- Frequently you need a value later which is not directly coupled to your current stream item. Nowhere to put it!

```
Stream.of(1, 2, 3)
   .map(id → databaseLookup(id))
   .filter(dbresult → dbresult.size() > 10)
   .map(dbresult → stuff(dbresult))
   .map((id, value) → databaseSave(id, value)
...
```



`id` is lost earlier in the stream.

What if we had?

- A generic helper class which acts like a two-tuple in a stream. (Enough for standard runtime library)
- Helper methods to make it easy to switch to and use the class with the intermediate operations ”filter”, ”map” and ”flatMap”.
- No magic, but proper generics so the compiler and IDE’s has as much information as possible.

Note: Keep it simple. Do this job and this job only!

Side quests

- Idempotent jar files (compile same sources, get same binary jar any time)
- jUnit 5.
- Deploy to Maven Central as opposed to internal Nexus.
- Auto-publish Javadoc online.

General approach:

```
Stream.of(…)
   .map(StreamTuple::create)
   .X(st → st.X(v → ...))
   .Y(st → st.Y((id, v) → …))
   ……
   .collect(toMap(st → st.left(), st → st.right());
```

(Helper methods has same name as the method they are intended to be used for - `filter/map/flatMap`)

```java
@Test  // full junit 5 unit test
public void simpleMapUpdateOperationUsingStreamTupleForEach() {
    Map<Integer, String> map = new HashMap<>();
    map.put(1, "1.");
    map.put(2, "2.");

    Map result = map.keySet().stream()
            .map(StreamTuple::create) // (1, 1), (2, 2)
            // lookup value for key
            .map(st -> st.map(key -> map.get(key))) // (1, "1."), (2, "2.")
            // only process those who are interesting
            .filter(st -> st.filter(s -> s.startsWith("1"))) // (1, "1.")
            // manipulate value, no notion of key
            .map(st -> st.map(s -> s + " OK"))  // (1, "1. OK")
            // store value back for key
            .map(st -> st.map((key, s) -> map.put(key, s))) // (1, "1.") (put returns old value)
            .collect(toMap(st -> st.left(), st -> st.right()));

    Map<Integer, String> expectedResult = new HashMap<>();
    expectedResult.put(1, "1.");
    assertThat(result, is(expectedResult));

    Map<Integer, String> expectedMap = new HashMap<>();
    expectedMap.put(1, "1. OK");
    expectedMap.put(2, "2.");
    assertThat(map, is(expectedMap));
}
```

Maven Central

- Maven coordinates:

```
<groupId>dk.kb.stream</groupId>
<artifactId>streamtuples</artifactId>
<version>...</version>
```

See https://mvnrepository.com/artifact/dk.kb.stream/streamtuples for latest version.

- Used ”ossrh” approach as recommended by @nicl. Only non-trivial step is GPG key.
- mvn release:prepare -DpushChanges=false (and no release:perform, but manual deploy)

Conclusion

- Gives an extra degree of freedom similar to local variables while working with Streams.
- Quite some extra typing to get there.
  Not easily fixed without more help from the compiler (not on the horizon).
- Won over `javac` in the generics Boss Battle!

Misc. findings:

- Stream debug trick: .peek(System.err::println)
- JDBC does *not* work well with Streams. Much pain! RowSets may work better than ResultSets.
- Generics are a kludge.
- Streams too.
- I miss Haskell.

Cat tax - Felix