STREAMS, LAMBDAS, COLLECTIONS

TYPICAL APPROACH

- Common to iterate over a collection, operating on elements
  - boilerplate code for iterating
  - meaning can be less clear (need to read through entire loop)
- Depending on style of loop - actually creates iterator underneath

STREAMS

- Not a data structure
- Don't store elements
- Get elements on demand
- Allow for "functional programming" style in Java

STREAMS

- An interface
- Have an input source (collections, others)
- Allow for operations on them (see Stream documentation)

OPERATIONS

- terminal
  - return nothing (void)
  - return something other than a stream (primitive, collection, etc.)
- intermediate
  - return another stream
  - can be chained together in a pipeline

COMMON OPERATIONS

COMMON OPERATIONS

- `sorted()` - intermediate
  - Return stream of same objects in a sorted order
- `forEach()` - terminal
  - do something for each object
- `reduce()` - terminal
  - combine together somehow
- `sum()` - terminal
  - combine together by summing (need to be summable)
- `max()` - terminal
  - return maximum element

LAZY EVALUATION

- Intermediate methods are "lazy"
  - Examples: `filter`, `map`, etc.
  - Build up stream recipe
  - Don't force a new value to be generated at end
  - Won't actually bother doing anything until we add a "terminal" operation