# STREAMS, LAMBDAS, COLLECTIONS

## COLLECTIONS

- Group of objects (elements)
- Gathers/organizes elements
- Interface in Java - specifies how to interact with (access/manage) group of objects
- An abstraction - details of implementation hidden

## COLLECTIONS

- Some are ordered, others unordered
- Some allow duplicates, others don't
- Example: `ArrayList` implements `Collection`
    - Implementation differs from other implementers of `Collection`
    - Stil interact in the same way

## 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 - TERMINAL

- build result list from stream

```
collect(Collectors.toList())
```

- build ArrayList from stream

```
collect(Collectors.toCollection(ArrayList::new)
```

## COMMON OPERATIONS

- `sum()` - terminal
  - combine together by summing (need to be summable)
- `average()` - terminal
  - combine together by taking average

## COMMON OPERATIONS

- `max()` - terminal
    - combine together by taking max
- `min()` - terminal
    - combine together by taking min
- `reduce()` - terminal
    - combine together somehow

## COMMON OPERATIONS

- `forEach()` - terminal
  - do something for each object
- `count()` - terminal
  - counts the number of items in the stream

## COMMON OPERATIONS

- `map()` - intermediate
    - apply function to convert from one thing (value, type, etc.) to another
- `mapToInt()` and `mapToDouble()` - intermediate
    - convert to a special stream of Integer or Double

## COMMON OPERATIONS

- `sorted()` - intermediate
  - Return stream of same objects in a sorted order
- `filter()` - intermediate
  - filter out some of the elements (only let ones through for which condition is true)

## 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

## IMPORTANT NOTES

- Some operations return interesting "Optional" types
  - `OptionalDouble` (returned by average for instance), add on call to `.getAsDouble()` to turn into `double`
  - `OptionalInt` add on call to `.getAsInt()`
- Many operations take a lambda function
- In some cases, we can replace lambda function with `ClassName::methodName`