Learning Objectives in this Part of the Lesson

- Understand stream internals, e.g.
  - Know what can change & what can’t
- Recognize how a Java stream is constructed

```
Input x

Stream map(Function<…> mapper)

Output f(x)

Stream filter(Predicate<…> pred)

Output g(f(x))

Stream sorted()

Output h(g(f(x)))

R collect(Collector<…> collector)
```
Java Stream Construction
Recall that intermediate operations are "lazy"

Java Stream Construction

Input x

Stream map(Function<…> mapper)

Output f(x)

Stream filter(Predicate<…> pred)

Output g(f(x))

Stream sorted()

Output h(g(f(x)))

R collect(Collector<…> collector)

See www.logicbig.com/tutorials/core-java-tutorial/java-util-stream/lazy-evaluation
Recall that intermediate operations are “lazy”

i.e., they don’t start to run until a terminal operator is reached

See www.logicbig.com/tutorials/core-java-tutorial/java-util-stream/lazy-evaluation
A stream pipeline is constructed at runtime via an internal representation.

At runtime a linked list of stream source & intermediate operations is built, one per “stage” in pipeline.
Java Stream Construction

- A stream pipeline is constructed at runtime via an internal representation.
- Each pipeline stage is described by a bitmap of *stream flags* internally.

<table>
<thead>
<tr>
<th>Stream Flag</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIZED</td>
<td>Size of stream is known</td>
</tr>
<tr>
<td>DISTINCT</td>
<td>Elements of stream are distinct</td>
</tr>
<tr>
<td>SORTED</td>
<td>Elements of the stream are sorted in natural order</td>
</tr>
<tr>
<td>ORDERED</td>
<td>Stream has meaningful encounter order</td>
</tr>
</tbody>
</table>

These flags are a subset of the flags that can be defined by a spliterator.

- **Input x**
- **Output $f(x)$**
- **Output $g(f(x))$**
- **Output $h(g(f(x)))$**
- $R \text{ collect}(\text{Collector}<...> \text{ collector})$
Java Stream Construction

- A stream pipeline is constructed at runtime via an internal representation.
- Each pipeline stage is described by a bitmap of *stream flags* internally.
- Source stage stream flags are derived from spliterator characteristics, e.g.

<table>
<thead>
<tr>
<th>Collection</th>
<th>Sized</th>
<th>Ordered</th>
<th>Sorted</th>
<th>Distinct</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrayList</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HashSet</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>TreeSet</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Stream generate() & iterate() methods create streams that are *not* sized!
A stream pipeline is constructed at runtime via an internal representation.

- Each pipeline stage is described by a bitmap of *stream flags* internally.
- Source stage stream flags are derived from spliterator characteristics.
- Each intermediate operation affects the stream flags.

Java Stream Construction

```
Input x
Stream map(Function<…> mapper)
   Output f(x)
Stream filter(Predicate<…> pred)
   Output g(f(x))
Stream sorted()
   Output h(g(f(x)))
R collect(Collector<…> collector)
```
A stream pipeline is constructed at runtime via an internal representation

- Each pipeline stage is described by a bitmap of *stream flags* internally
- Source stage stream flags are derived from spliterator characteristics
- Each intermediate operation affects the stream flags, e.g.
  - `map()`
    - Clears SORTED & DISTINCT but keeps SIZED

**Java Stream Construction**

```
Input x
Stream map(Function<…> mapper)
Output f(x)
Stream filter(Predicate<…> pred)
Output g(f(x))
Stream sorted()
Output h(g(f(x)))
R collect(Collector<…> collector)
```
A stream pipeline is constructed at runtime via an internal representation.

1. Each pipeline stage is described by a bitmap of *stream flags* internally.
2. Source stage stream flags are derived from spliterator characteristics.
3. Each intermediate operation affects the stream flags, e.g.:
   - `map()`
   - `filter()`
      - Keeps SORTED & DISTINCT but clears SIZED

Java Stream Construction:

- *Input x*
- `Stream map(Function<...> mapper)`
  - *Output f(x)*
- `Stream filter(Predicate<...> pred)`
  - *Output g(f(x))*
- `Stream sorted()`
  - *Output h(g(f(x)))*
- `R collect(Collector<...> collector)`
A stream pipeline is constructed at runtime via an internal representation.

Each pipeline stage is described by a bitmap of stream flags internally.

Source stage stream flags are derived from spliterator characteristics.

Each intermediate operation affects the stream flags, e.g.

- `map()`
- `filter()`
- `sorted()`

- Keeps SIZED & DISTINCT & adds SORTED

Java Stream Construction

```
Stream map(Function<...> mapper)
Stream filter(Predicate<...> pred)
Stream sorted()
R collect(Collector<...> collector)
```
A stream pipeline is constructed at runtime via an internal representation.

- Each pipeline stage is described by a bitmap of *stream flags* internally.
- Source stage stream flags are derived from spliterator characteristics.
- Each intermediate operation affects the stream flags.
- The flags at each stage are updated as the pipeline is being constructed.

Java Stream Construction

- `Stream map(Function<…> mapper)`
- `Stream filter(Predicate<…> pred)`
- `Stream sorted()`
- `R collect(Collectors<…> collector)`

Input `x`

Output `f(x)`

Output `g(f(x))`

Output `h(g(f(x)))`
A stream pipeline is constructed at runtime via an internal representation

Each pipeline stage is described by a bitmap of *stream flags* internally

Source stage stream flags are derived from spliterator characteristics

Each intermediate operation affects the stream flags

The flags at each stage are updated as the pipeline is being constructed

e.g., flags for a previous stage are combined with the current stage’s behavior to derive a new set of flags

Java Stream Construction

Input $x$

**Stream map**($\text{Function}<\ldots>\text{mapper}$)

Output $f(x)$

**Stream filter**($\text{Predicate}<\ldots>\text{pred}$)

Output $g(f(x))$

**Stream sorted**()

Output $h(g(f(x)))$

$R$ **collect**($\text{Collector}<\ldots>\text{collector}$)
A stream pipeline is constructed at runtime via an internal representation. Each pipeline stage is described by a bitmap of stream flags internally. Source stage stream flags are derived from spliterator characteristics. Each intermediate operation affects the stream flags. The flags at each stage are updated as the pipeline is being constructed. e.g., flags for a previous stage are combined with the current stage’s behavior to derive a new set of flags.

Java Stream Construction

```java
Set<String> ts = new TreeSet<>(...);
List<String> sortedAWords =
    ts.stream()
        .filter(s -> s.startsWith("A"))
        .sorted()
        .collect(toList());
```

The streams framework removes redundant operations since the source is already sorted.
End of Understand Java Stream Internals: Construction