Visualizing Java Streams in Action

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

• Understand Java streams structure & functionality, e.g.
  • Fundamentals of streams
  • Three streams phases
  • Operations that create a stream
  • Aggregate operations in a stream
  • Visualizing streams in action
Visualizing Streams in Action
Streams enhance flexibility by forming a “processing pipeline” that composes multiple aggregate operations together.

Input $x$

$\text{Aggregate operation (behavior } f\text{)}$

Output $f(x)$

$\text{Aggregate operation (behavior } g\text{)}$

Output $g(f(x))$

$\text{Aggregate operation (behavior } h\text{)}$

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

See en.wikipedia.org/wiki/Pipeline_(software)
Streams enhance flexibility by forming a “processing pipeline” that composes multiple aggregate operations together.

Each aggregate operation in the pipeline can filter and/or transform the stream.

See [en.wikipedia.org/wiki/Water_filter#Point-of-use_filters](en.wikipedia.org/wiki/Water_filter#Point-of-use_filters)
Streams enhance flexibility by forming a “processing pipeline” that composes multiple aggregate operations together.

Array (String)
- “horatio”
- “laertes”
- “Hamlet” ...

Stream (String)
- “horatio”
- “laertes”
- “Hamlet”

Stream (String)
- “horatio”
- “Hamlet”

Stream (String)
- “Hamlet”
- “Horatio”

Array of names of (“horatio”, “laertes”, “Hamlet”, …)

Stream of names

Stream of names starting with ‘h’

Stream of capitalized names

Stream of sorted names

Visualizing Streams in Action
Streams enhance flexibility by forming a “processing pipeline” that composes multiple aggregate operations together.

Array

```
horatio
laertes
Hamlet
...
```

Stream

```
horatio
laertes
Hamlet
```

Stream

```
horatio
Hamlet
```

Stream

```
Horatio
Hamlet
```

Stream

```
Hamlet
Horatio
```

Array of names

```
(horatio, laertes, Hamlet, ...)
```

Stream of names

```
filter(s -> toLowerCase(s.charAt(0)...)```

Stream of names starting with ‘h’

```
map(this::capitalize)
```

Stream of capitalized names

```
sorted()
```

Stream of sorted names
Streams enhance flexibility by forming a “processing pipeline” that composes multiple aggregate operations together.

- **Array** `<String>`
  - “horatio”  “laertes”  “Hamlet”  ...

- **Stream** `<String>`
  - “horatio”
  - “laertes”
  - “Hamlet”

- **Stream** `<String>`
  - “horatio”
  - “Hamlet”

- **Stream** `<String>`
  - “Horatio”
  - “Hamlet”

- **Stream** `<String>`
  - “Hamlet”
  - “Horatio”

**Visualizing Streams in Action**

- **Array of names** of (“horatio”, “laertes”, “Hamlet”, …)
- **Stream of names**
  - filter(s->toLowerCase(s.charAt(0)…))
- **Stream of names starting with ‘h’**
  - map(this::capitalize)
- **Stream of capitalized names**
  - sorted()
- **Stream of sorted names**
• Streams enhance flexibility by forming a “processing pipeline” that composes multiple aggregate operations together.
Streams enhance flexibility by forming a “processing pipeline” that composes multiple aggregate operations together.

```
Stream<String> filter(s->toLowerCase(s.charAt(0))...)
```

```
Stream<String> map(this::capitalize)
```

```
Stream<String> sorted()
```

Visualizing Streams in Action

10
Streams enhance flexibility by forming a “processing pipeline” that composes multiple aggregate operations together.

- **Array**: `Array <String>`
  - "horatio"  "laertes"  "Hamlet"  …

- **Stream**: `Stream <String>`
  - "horatio"  "laertes"  "Hamlet"

- **Stream**: `Stream <String>`
  - "horatio"
  - "Hamlet"

- **Stream**: `Stream <String>`
  - "Horatio"
  - "Hamlet"

- **Stream**: `Stream <String>`
  - "Hamlet"
  - "Horatio"

- **Array of names**
  - of("horatio", "laertes", "Hamlet", …)

- **Stream of names**
  - filter(s->toLowerCase(s.charAt(0)…)

- **Stream of names starting with ‘h’**
  - map(this::capitalize)

- **Stream of capitalized names**
  - sorted()

- **Stream of sorted names**
Streams enhance flexibility by forming a “processing pipeline” that composes multiple aggregate operations together.

Visualizing Streams in Action

- Array of names
  - `Array <String>`
    - “horatio”
    - “laertes”
    - “Hamlet”
    - ...

- Stream of names
  - `Stream <String>`
    - “horatio”
    - “laertes”
    - “Hamlet”

- Stream of names starting with ‘h’
  - `Stream <String>`
    - “horatio”
    - “Hamlet”

- Stream of capitalized names
  - `Stream <String>`
    - “Horatio”
    - “Hamlet”

- Stream of sorted names
  - `Stream <String>`
    - “Hamlet”
    - “Horatio”

`Stream of sorted names`
• Streams enhance flexibility by forming a “processing pipeline” that composes multiple aggregate operations together
Streams enhance flexibility by forming a “processing pipeline” that composes multiple aggregate operations together.
Visualizing Streams in Action

- Streams enhance flexibility by forming a “processing pipeline” that composes multiple aggregate operations together.
End of Visualizing Java Streams in Action