Contrasting Java Streams with Java Collections

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 how Java collections contrast with Java streams

```
java.util.ArrayList size: 5
```

```
Aggregate operation (Function f)
Input x
Output f(x)
```

```
Aggregate operation (Function g)
Output g(f(x))
```

```
Aggregate operation (Function h)
Output h(g(f(x)))
```

```
Point x:0.0 y:0.0
```

```
Point x:0.0 y:0.0
```

```
Point x:0.0 y:0.0
```
Learning Objectives in this Part of the Lesson

- Understand how Java collections contrast with Java streams
- Know how to program with Java collections & streams

```java
List<String> urls = new ArrayList<>(Arrays.asList(urlArray));
for (int i = 0; i < urls.size(); ++i) {
    if (!urls.get(i).contains("cse.wustl")) {
        urls.remove(i); continue;
    }
    urls.set(i, urls.get(i).replace("cse.wustl", "dre.vanderbilt");
}

List<String> urls = Arrays.asList(urlArray).stream()
    .filter(s -> s.contains("cse.wustl"))
    .map(s -> s.replace("cse.wustl", "dre.vanderbilt"))
    .collect(toList());
```
Contrasting Java Collections & Java Streams
Contrasting Java Collections & Java Streams

- Java collections are different from Java streams!

java.util.ArrayList size: 5

elementData

0 1 2 3 4

Point x:0.0 y:0.0
Point x:0.0 y:0.0
Point x:0.0 y:0.0

Input x

Aggregate operation (Function f)

Output f(x)

Aggregate operation (Function g)

Output g(f(x))

Aggregate operation (Function h)

Output h(g(f(x)))

See www.oracle.com/technetwork/articles/java/ma14-java-se-8-streams-2177646.html
A Java collection is an in-memory data structure that can store, retrieve, & manipulate groups of elements.
Contrasting Java Collections & Java Streams

- A Java collection is an in-memory data structure that can store, retrieve, & manipulate groups of elements
- It is somewhat analogous to the contents on a DVD

\[\text{e.g., its content can be read from & written to (sometimes)}\]
A stream is a fixed-size pipeline that processes elements on-demand.

A stream can manipulate elements obtained from a collection without explicitly iterating over them.

Contrasting Java Collections & Java Streams

See tutorials.jenkov.com/java-collections_streams.html
Contrasting Java Collections & Java Streams

- A Java stream is a fixed-size pipeline that processes elements on-demand.
- It is somewhat analogous to a flow of elements in a video stream.

\[ \text{e.g., its content is processed as it flows through the stream} \]
Contrasting Java Collections & Java Streams

• These analogies are not perfect!
Contrasting Java Collections & Java Streams

- Various factory methods can convert collections to streams

```java
java.util.ArrayList
```

```java
size: 5
```

```java
stream()
```

```java
parallelStream()
```

![Diagram showing the conversion of a collection to a stream and the execution of aggregate operations with functions f, g, and h.]

Input x

Aggregate operation (Function f)

Output f(x)

Aggregate operation (Function g)

Output g(f(x))

Aggregate operation (Function h)

Output h(g(f(x)))

See upcoming lessons on “Stream Creation Operations”
Contrasting Java Collections & Java Streams

- Other terminal operations can convert streams to collections

See upcoming lessons on "Stream Terminal Operations"
Examples of Java Collections & Java Streams
A simple example of manipulating a Java collection

String[] urlArray = {
    "http://www.cse.wustl.edu/~schmidt/gifs/ka.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/robot.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/kitten.png"};

List<String> urls = new ArrayList<>(Arrays.asList(urlArray));

for (int i = 0; i < urls.size(); ++i) {
    if (!urls.get(i).contains("cse.wustl")) {
        urls.remove(i); continue;
    }
    urls.set(i, urls.get(i).replace("cse.wustl","dre.vanderbilt"));
}
A simple example of manipulating a Java collection

```java
String[] urlArray = {
    "http://www.cse.wustl.edu/~schmidt/gifs/ka.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/robot.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/kitten.png"};

List<String> urls = new ArrayList<>(Arrays.asList(urlArray));

for (int i = 0; i < urls.size(); ++i) {
    if (!urls.get(i).contains("cse.wustl")) {
        urls.remove(i); continue;
    }
    urls.set(i, urls.get(i).replace("cse.wustl","dre.vanderbilt"));
}
```

This example demonstrates external iteration

A simple example of manipulating a Java collection

```java
String[] urlArray = {
    "http://www.cse.wustl.edu/~schmidt/gifs/ka.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/robot.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/kitten.png"};

List<String> urls = new ArrayList<>(Arrays.asList(urlArray));

for (int i = 0; i < urls.size(); ++i) {
    if (!urls.get(i).contains("cse.wustl")) {
        urls.remove(i); continue;
    }
    urls.set(i, urls.get(i).replace("cse.wustl","dre.vanderbilt"));
}
```

Examples of Java Collections & Java Streams

Create a list from an array
• A simple example of manipulating a Java collection

```java
String[] urlArray = {
    "http://www.cse.wustl.edu/~schmidt/gifs/ka.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/robot.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/kitten.png"};

List<String> urls = new ArrayList<>(Arrays.asList(urlArray));

for (int i = 0; i < urls.size(); ++i) {
    if (!urls.get(i).contains("cse.wustl")) {
        urls.remove(i); continue;
    }
    urls.set(i,
        urls.get(i).replace("cse.wustl","dre.vanderbilt"));
}
```

Explicitly iterate through a list
A simple example of manipulating a Java collection

```java
String[] urlArray = {
    "http://www.cse.wustl.edu/~schmidt/gifs/ka.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/robot.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/kitten.png"};

List<String> urls = new ArrayList<>(Arrays.asList(urlArray));

for (int i = 0; i < urls.size(); ++i) {
    if (!urls.get(i).contains("cse.wustl")) {
        urls.remove(i); continue;
    }
    urls.set(i, urls.get(i).replace("cse.wustl","dre.vanderbilt"));
}
```

External iteration enables fine-grained control of loop behavior
A simple example of manipulating a Java collection

```java
String[] urlArray = {
    "http://www.cse.wustl.edu/~schmidt/gifs/ka.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/robot.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/kitten.png"};

List<String> urls = new ArrayList<>(Arrays.asList(urlArray));

for (int i = 0; i < urls.size(); ++i) {
    if (!urls.get(i).contains("cse.wustl")) {
        urls.remove(i); continue;
    }
    urls.set(i, urls.get(i).replace("cse.wustl","dre.vanderbilt"));
}
```
A simple example of manipulating a Java collection

```java
String[] urlArray = {
    "http://www.cse.wustl.edu/~schmidt/gifs/ka.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/robot.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/kitten.png"};

List<String> urls = new ArrayList<>(Arrays.asList(urlArray));

for (int i = 0; i < urls.size(); ++i) {
    if (!urls.get(i).contains("cse.wustl")) {
        urls.remove(i); continue;
    }
    urls.set(i, urls.get(i).replace("cse.wustl", "dre.vanderbilt"));
}
```

However, this code is tedious & error-prone to write, read, & optimize

See howtodoinjava.com/java8/internal-vs-external-iteration
Examples of Java Collections & Java Streams

• A simple example of manipulating a Java stream

```java
String[] urlArray = {
    "http://www.cse.wustl.edu/~schmidt/gifs/ka.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/robot.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/kitten.png"};

List<String> urls = Arrays.asList(urlArray)
    .stream()
    .filter(s -> s.contains("cse.wustl"))
    .map(s ->
        s.replace("cse.wustl", "dre.vanderbilt"))
    .collect(toList());
```

This example shows the “fluent interface” programming style, internal iteration, chaining of transformations

See [en.wikipedia.org/wiki/Fluent_interface](en.wikipedia.org/wiki/Fluent_interface)
Implicitly iterate through a pipeline of elements from a collection source, filter/transform each value, & create a collection result

Examples of Java Collections & Java Streams

• A simple example of manipulating a Java stream

```java
String[] urlArray = {
    "http://www.cse.wustl.edu/~schmidt/gifs/ka.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/robot.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/kitten.png"};

List<String> urls = Arrays.asList(urlArray)
    .stream()
    .filter(s -> s.contains("cse.wustl"))
    .map(s ->
        s.replace("cse.wustl", "dre.vanderbilt"))
    .collect(toList());
```
Examples of Java Collections & Java Streams

• A simple example of manipulating a Java stream

```java
String[] urlArray = {
    "http://www.cse.wustl.edu/~schmidt/gifs/ka.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/robot.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/kitten.png"};

List<String> urls = Arrays.asList(urlArray)
    .stream()
    .filter(s -> s.contains("cse.wustl"))
    .map(s ->
        s.replace("cse.wustl", "dre.vanderbilt"))
    .collect(toList());
```

Implicitly iterate through a pipeline of elements from a collection source, filter/transform each value, & create a collection result.
Examples of Java Collections & Java Streams

- A simple example of manipulating a Java stream

```java
String[] urlArray = {
    "http://www.cse.wustl.edu/~schmidt/gifs/ka.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/robot.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/kitten.png"};

List<String> urls = Arrays.asList(urlArray)
    .stream()
    .filter(s -> s.contains("cse.wustl"))
    .map(s ->
        s.replace("cse.wustl", "dre.vanderbilt"))
    .collect(toList());
```

Implicitly iterate through a pipeline of elements from a collection source, filter/transform each value, & create a collection result.
Examples of Java Collections & Java Streams

• A simple example of manipulating a Java stream

```java
String[] urlArray = {
    "http://www.cse.wustl.edu/~schmidt/gifs/ka.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/robot.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/kitten.png"};

List<URL> urls = Arrays.asList(urlArray)
    .stream()
    .filter(s -> s.contains("cse.wustl"))
    .map(s ->
        s.replace("cse.wustl", "dre.vanderbilt"))
    .map(rethrowFunction(URL::new))
    .collect(toList());
```

*Java streams simplifies chaining of transformations*
Examples of Java Collections & Java Streams

- A simple example of manipulating a Java stream

```java
String[] urlArray = {
    "http://www.cse.wustl.edu/~schmidt/gifs/ka.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/robot.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/kitten.png"};

List<URL> urls = Arrays.asList(urlArray)
    .stream()
    .filter(s -> s.contains("cse.wustl"))
    .map(s ->
        s.replace("cse.wustl", "dre.vanderbilt")
    )
    .map(rethrowFunction(URL::new))
    .collect(toList());
```

See stackoverflow.com/a/27661504/3312330
Examples of Java Collections & Java Streams

- A simple example of manipulating a Java stream

```java
String[] urlArray = {
    "http://www.cse.wustl.edu/~schmidt/gifs/ka.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/robot.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/kitten.png"};

List<URL> urls = Arrays.asList(urlArray)
    .parallelStream()
    .filter(s -> s.contains("cse.wustl"))
    .map(s ->
        s.replace("cse.wustl", "dre.vanderbilt"))
    .map(rethrowFunction(URL::new))
    .collect(toList());
```

Parallelizing a Java stream is easy!!
A simple example of manipulating a Java stream

```java
String[] urlArray = {
    "http://www.cse.wustl.edu/~schmidt/gifs/ka.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/robot.png",
    "http://www.cse.wustl.edu/~schmidt/gifs/kitten.png"};

List<String> urls = Arrays.asList(urlArray)
    .stream()
    .filter(s -> s.contains("cse.wustl"))
    .map(s ->
        s.replace("cse.wustl", "dre.vanderbilt"))
    .collect(toList());
```

Like iterators, elements in a stream can only be visited once during its lifetime.
End of Contrasting Java Streams with Java Collections