External vs. Internal Iterators in Java: Introduction

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

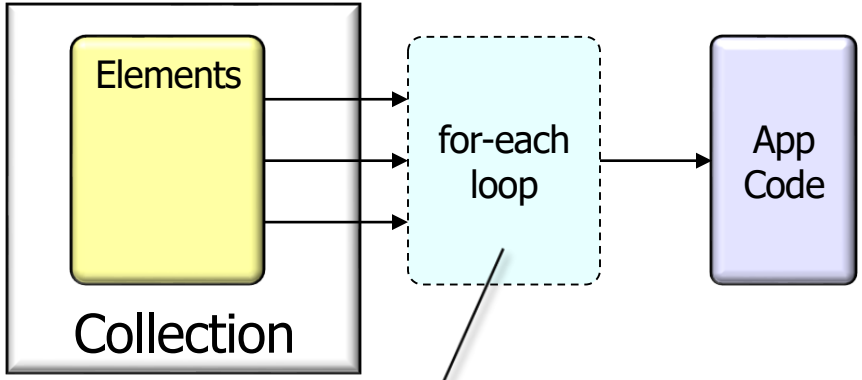
Learning Objectives in this Part of the Lesson

- Recognize the difference between external & internal iterators in Java

External Iterators vs. Internal Iterators

Overview of External Iterators vs. Internal Iterators

- Java programmers have historically iterated through collections *externally*



A Java for-each loop exists outside of any collection & invokes app-supplied code on each element during the external iteration process.

Overview of External Iterators vs. Internal Iterators

- Java programmers have historically iterated through collections *externally*

```
List<String> namesList =
  Arrays.asList("Larry",
                "Curly",
                "Moe");

for (String name : namesList)
  System.out.println(name);
```

A Java for-each loop is a common way to iterate through a collection externally.

See docs.oracle.com/javase/1.5.0/docs/guide/language/foreach.html

Overview of External Iterators vs. Internal Iterators

- Java programmers have historically iterated through collections *externally*

```
List<String> namesList =
  Arrays.asList("Larry",
                "Curly",
                "Moe");

for (String name : namesList)
  System.out.println(name);
```

Each element in the collection is accessed sequentially.

Overview of External Iterators vs. Internal Iterators

- Java programmers have historically iterated through collections *externally*

```
List<String> namesList =
  Arrays.asList("Larry",
                "Curly",
                "Moe");

for (String name : namesList)
  System.out.println(name);
```

Some app-supplied action is performed on each element.

Overview of External Iterators vs. Internal Iterators

- Java programmers have historically iterated through collections *externally*

```
List<String> namesList =
  Arrays.asList("Larry",
                "Curly",
                "Moe");

for (Iterator<String>> i =
        namesList.iterator();
     i.hasNext();)
  System.out.printLn(i.next());
```

A Java Iterator is another means to externally iterate through a collection.

See docs.oracle.com/javase/8/docs/api/java/util/Iterator.html

Overview of External Iterators vs. Internal Iterators

- Java programmers have historically iterated through collections *externally*

```
List<String> namesList =
  Arrays.asList("Larry",
                "Curly",
                "Moe");

for (Iterator<String>> i =
        namesList.iterator();
     i.hasNext();)
  System.out.println(i.next());
```

Factory method obtains an iterator to the collection.

See docs.oracle.com/javase/8/docs/api/java/util/Collection.html#iterator

Overview of External Iterators vs. Internal Iterators

- Java programmers have historically iterated through collections *externally*

```
List<String> namesList =
  Arrays.asList("Larry",
                "Curly",
                "Moe");

for (Iterator<String>> i =
        namesList.iterator();
     i.hasNext();)
  System.out.printLn(i.next());
```

Check if any elements remain in the collection.

Overview of External Iterators vs. Internal Iterators

- Java programmers have historically iterated through collections *externally*

```
List<String> namesList =
  Arrays.asList("Larry",
                "Curly",
                "Moe");

for (Iterator<String>> i =
        namesList.iterator();
     i.hasNext();)
  System.out.printLn(i.next());
```

Get the next element in the collection

Overview of External Iterators vs. Internal Iterators

- Java programmers have historically iterated through collections *externally*

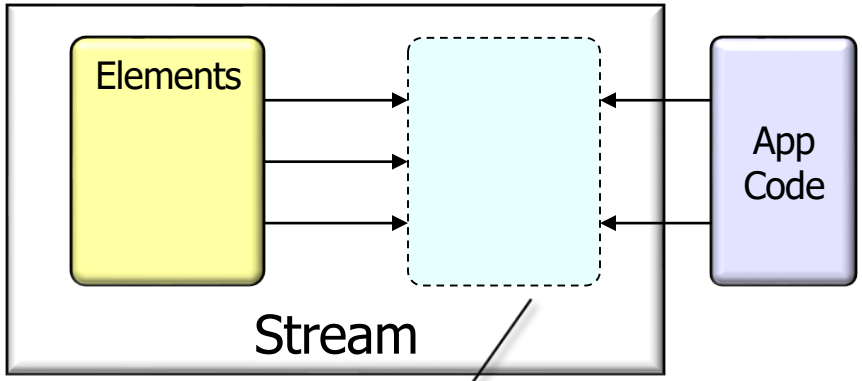
```
List<String> namesList =
  Arrays.asList("Larry",
                "Curly",
                "Moe");

for (Iterator<String>> i =
        namesList.iterator();
     i.hasNext();)
  System.out.printLn(i.next());
```

Perform some action on each element

Overview of External Iterators vs. Internal Iterators

- In contrast, aggregate operations in Java are responsible for iterating through Java streams *internally*



A Java stream invokes app-supplied code on each stream element during the internal iteration process.

See docs.oracle.com/javase/tutorial/collections/streams/#differences

Overview of External Iterators vs. Internal Iterators

- In contrast, aggregate operations in Java are responsible for iterating through Java streams *internally*

```
List<String> namesList =
  Arrays.asList("Larry",
                "Curly",
                "Moe");
```

Convert the list into a stream

```
namesList.stream().forEach
   (System.out::println);
```

Overview of External Iterators vs. Internal Iterators

- In contrast, aggregate operations in Java are responsible for iterating through Java streams *internally*

```
List<String> namesList =
  Arrays.asList("Larry",
                "Curly",
                "Moe");
```

Perform some action on each element

```
namesList.stream().forEach
  (System.out::println);
```

See docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html#forEach

Overview of External Iterators vs. Internal Iterators

- Internal iterators are useful when stream pipelines become more complex, e.g.

```
List<URL> urls = Stream
  .of(urlArray)

  .map(s ->
       s.replace("cse.wustl",
                 "dre.vanderbilt"))

  .map(url ->
       { try { return new URL(url); }
       catch(Exception ex){ ... }})


  .collect(toList());
```

Overview of External Iterators vs. Internal Iterators

- Internal iterators are useful when stream pipelines become more complex, e.g.

```
List<URL> urls = Stream
  .of(urlArray)

  .map(s ->
       s.replace("cse.wustl",
                 "dre.vanderbilt"))

  .map(url ->
       { try { return new URL(url);
       catch(Exception ex){ ... }})

  .collect(toList());
```

Convert array to a stream

Overview of External Iterators vs. Internal Iterators

- Internal iterators are useful when stream pipelines become more complex, e.g.

```
List<URL> urls = Stream
  .of(urlArray)

  .map(s ->
       s.replace("cse.wustl",
                 "dre.vanderbilt"))

  .map(url ->
       { try { return new URL(url);
       catch(Exception ex){ ... }})

  .collect(toList());
```

Replace substrings in stream

Overview of External Iterators vs. Internal Iterators

- Internal iterators are useful when stream pipelines become more complex, e.g.

```
List<URL> urls = Stream
  .of(urlArray)

  .map(s ->
       s.replace("cse.wustl",
                 "dre.vanderbilt"))

  .map(url ->
       { try { return new URL(url);
       catch(Exception ex){ ... }})

  .collect(toList());
```

Convert strings to a URLs

Overview of External Iterators vs. Internal Iterators

- Internal iterators are useful when stream pipelines become more complex, e.g.

```
List<URL> urls = Stream
  .of(urlArray)

  .map(s ->
       s.replace("cse.wustl",
                 "dre.vanderbilt"))

  .map(url ->
       { try { return new URL(url);
       catch(Exception ex){ ... }})
```

Collect results into a list

```
  .collect(toList());
```

Overview of External Iterators vs. Internal Iterators

- Internal iterators are useful when stream pipelines become more complex, e.g.

```
List<URL> urls = Stream
  .of(urlArray)

  .map(s ->
       s.replace("cse.wustl",
                 "dre.vanderbilt"))

  .map(url ->
       { try { return new URL(url); }
       catch(Exception ex){ ... }})
```

Checked exceptions are awkward!

```
  .collect(toList());
```

See slieb.org/blog/throwable-interfaces

Overview of External Iterators vs. Internal Iterators

- Internal iterators are useful when stream pipelines become more complex, e.g.

```
List<URL> urls = Stream
  .of(urlArray)

  .map(s ->
       s.replace("cse.wustl",
                 "dre.vanderbilt"))

  .map(rethrowFunction(URL::new))

  .collect(toList());
```

```
static <T, R> Function<T, R>
rethrowFunction
    (Func_WithExs<T, R> f) {
  return t -> {
   try { return f.apply(t); }
   catch (Exception ex) {
     throwAsUnchecked(ex);
      return null; }
   };
}
```

rethrowFunction() converts checked exceptions into runtime exceptions

See stackoverflow.com/a/27661504/3312330

End of External Iterators vs. Internal Iterators: Introduction