Applying the Java Consumer & Function Functional Interfaces

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

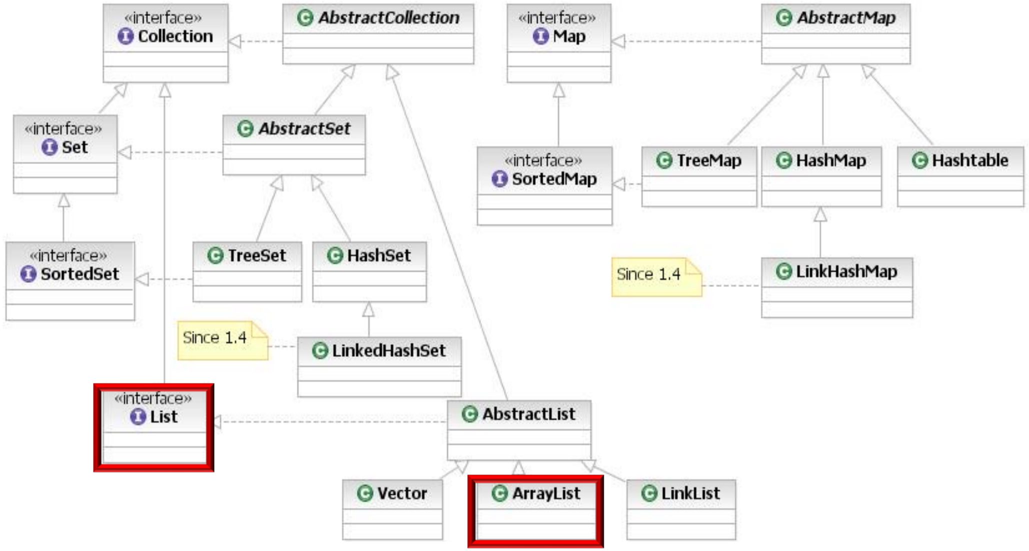
Learning Objectives in this Part of the Lesson

- Know how to apply Java Consumer & Function functional interfaces to another concise example

See github.com/douglascraigschmidt/ModernJava/tree/main/FP/ex14

Learning Objectives in this Part of the Lesson

- Know how to apply Java Consumer & Function functional interfaces to another concise example
  - This example shows the List interface & ArrayList class in the Java collection framework



See docs.oracle.com/javase/8/docs/technotes/guides/collections

Another Consumer & Function Interface Example

Another Consumer & Function Interface Example

- Here's another example of applying a *Consumer* & a *Function* together to print & sort Thread objects within a List

```
var threads = Arrays.asList(new Thread("Larry"),
                            new Thread("Curly"),
                            new Thread("Moe"));

threads.forEach(System.out::println);
threads.sort(Comparator.comparing(Thread::getName));
threads.forEach(System.out::println);
```

See github.com/douglascraigschmidt/ModernJava/tree/main/FP/ex14

Another Consumer & Function Interface Example

- Here's another example of applying a *Consumer* & a *Function* together to print & sort Thread objects within a List

```
var threads = Arrays.asList(new Thread("Larry"),
                            new Thread("Curly"),
                            new Thread("Moe"));
```

Create a List of Thread objects named after the three stooges

```
threads.forEach(System.out::println);
threads.sort(Comparator.comparing(Thread::getName));
threads.forEach(System.out::println);
```

See en.wikipedia.org/wiki/The_Three_Stooges

Another Consumer & Function Interface Example

- Here's another example of applying a *Consumer* & a *Function* together to print & sort Thread objects within a List

```
var threads = Arrays.asList(new Thread("Larry"),
                            new Thread("Curly"),
                            new Thread("Moe"));
```

Returns a fixed-size (modifiable) List backed by the specified array

```
threads.forEach(System.out::println);
threads.sort(Comparator.comparing(Thread::getName));
threads.forEach(System.out::println);
```

See docs.oracle.com/javase/8/docs/api/java/util/Arrays.html#asList

Another Consumer & Function Interface Example

- Here's another example of applying a *Consumer* & a *Function* together to print & sort Thread objects within a List

```
var threads = Arrays.asList(new Thread("Larry"),
                            new Thread("Curly"),
                            new Thread("Moe"));
```

A method reference to a Consumer is used to print threads by name

```
threads.forEach(System.out::println);
threads.sort(Comparator.comparing(Thread::getName));
threads. forEach(System.out::println);
```

See previous lesson on "*The Java Consumer Functional Interface*"

Another Consumer & Function Interface Example

- Here's another example of applying a *Consumer* & a *Function* together to print & sort Thread objects within a List

```
var threads = Arrays.asList(new Thread("Larry"),
                            new Thread("Curly"),
                            new Thread("Moe"));
```

A method reference to a Function used to sort Thread objects by name

```
threads.forEach(System.out::println);
threads.sort(Comparator.comparing(Thread::getName));
threads.forEach(System.out::println);
```

See dzone.com/articles/java-8-comparator-how-to-sort-a-list

Another Consumer & Function Interface Example

- Here's another example of applying a *Consumer* & a *Function* together to print & sort Thread objects within a List

```
var threads = Arrays.asList(new Thread("Larry"),
                            new Thread("Curly"),
                            new Thread("Moe"));
```

The comparing() method imposes a total ordering on the collection of Thread objects via the Thread::getName method reference

```
threads.forEach(System.out::println);
threads.sort(Comparator.comparing(Thread::getName));
threads.forEach(System.out::println);
```

See docs.oracle.com/javase/8/docs/api/java/util/Comparator.html#comparing

How Comparator Uses the Function Interface

How Comparator Uses the Function Interface

- Here's how the comparing() method in the Java Comparator interface uses the Function functional interface

```
interface Comparator {
  ...
  static <T, U extends Comparable<? super U>> Comparator<T>
           comparing(Function<? super T, ? extends U> keyEx) {
    return ((c1, c2) ->
            keyEx.apply(c1)
                  .compareTo(keyEx.apply(c2)); }
```

Imposes a total ordering on a collection of objects

See docs.oracle.com/javase/8/docs/api/java/util/Comparator.html#comparing

How Comparator Uses the Function Interface

- Here's how the comparing() method in the Java Comparator interface uses the Function functional interface

```
interface Comparator {
  ...
  static <T, U extends Comparable<? super U>> Comparator<T>
           comparing(Function<? super T, ? extends U> keyEx) {
    return ((c1, c2) ->
            keyEx.apply(c1)
                 .compareTo(keyEx.apply(c2)); }
```

The comparing() method is passed a Function parameter called keyEx

How Comparator Uses the Function Interface

- Here's how the comparing() method in the Java Comparator interface uses the Function functional interface

```
interface Comparator {
  ...
  static <T, U extends Comparable<? super U>> Comparator<T>
           comparing(Function<? super T, ? extends U> keyEx) {
    return ((c1, c2) ->
             keyEx.apply(c1)
                  .compareTo(keyEx.apply(c2)); }
```

`Thread::getName`

The Thread::getName method reference is bound to the keyEx parameter

How Comparator Uses the Function Interface

- Here's how the comparing() method in the Java Comparator interface uses the Function functional interface

```
interface Comparator {
  ...
  static <T, U extends Comparable<? super U>> Comparator<T>
           comparing(Function<? super T, ? extends U> keyEx) {
    return ((c1, c2) ->
            keyEx.apply(c1)
                 .compareTo(keyEx.apply(c2)); }
```

In this example c1 & c2 are Thread objects being compared by sort()

How Comparator Uses the Function Interface

- Here's how the comparing() method in the Java Comparator interface uses the Function functional interface

```
interface Comparator {
  ...
  static <T, U extends Comparable<? super U>> Comparator<T>
           comparing(Function<? super T, ? extends U> keyEx) {
    return ((c1, c2) ->
             keyEx.apply(c1)
                  .compareTo(keyEx.apply(c2)); }
```

The apply() method of the keyEx function yields String objects that are compared for their relationship

How Comparator Uses the Function Interface

- Here's how the comparing() method in the Java Comparator interface uses the Function functional interface

```
interface Comparator {
  ...
  static <T, U extends Comparable<? super U>> Comparator<T>
           comparing(Function<? super T, ? extends U> keyEx) {
    return ((c1, c2) ->
             keyEx.apply(c1)
                  .compareTo(keyEx.apply(c2)); }
```

```
c1.getName().compareTo(c2.getName())
```

The Thread::getName method reference is called to compare two thread names

End of Applying the Java Consumer & Function Functional Interfaces