Understand the Java Consumer
Functional Interface

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 Lesson

• Understand foundational functional programming features in Java, e.g.,
  • Lambda expressions
  • Method & constructor references
• Key functional interfaces
  • Predicate
  • Function
  • BiFunction
  • Supplier
  • Consumer

Interface Consumer<T>

Type Parameters:
T - the type of the input to the operation

All Known Subinterfaces:
Stream.Builder<T>

Functional Interface:
This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.
Learning Objectives in this Part of the Lesson

- Understand foundational functional programming features in Java
- Learn how to apply Java consumers in concise example programs

See [github.com/douglasraigschmidt/LiveLessons/tree/master/Java8](https://github.com/douglasraigschmidt/LiveLessons/tree/master/Java8)
Learning Objectives in this Part of the Lesson

- Understand foundational functional programming features in Java
- Learn how to apply Java consumers in concise example programs
- The examples showcase the Java collections framework

See docs.oracle.com/javase/8/docs/technotes/guides/collections
Overview of the Consumer Functional Interface
Overview of the Consumer Functional Interface

- A `Consumer` accepts a parameter & returns no results, e.g.,
  - public interface `Consumer<T>` { void accept(T t); }

See docs.oracle.com/javase/8/docs/api/java/util/function/Consumer.html
Overview of the Consumer Functional Interface

- A *Consumer* accepts a parameter & returns no results, e.g.,
  - `public interface Consumer<T> { void accept(T t); }`

*Consumer is a generic interface that is parameterized by one reference type*
Overview of the Consumer Functional Interface

- A *Consumer* accepts a parameter & returns no results, e.g.,
  - `public interface Consumer<T> { void accept(T t); }`

*Its single abstract method is passed one parameter & returns nothing*
Create a list of threads with names of the three stooges

```java
List<Thread> 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/LiveLessons/tree/master/Java8/ex5](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex5)
Overview of the Consumer Functional Interface

• A Consumer accepts a parameter & returns no results, e.g.,
• public interface Consumer<T> { void accept(T t); }

List<Thread> 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 docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html#forEach
Overview of the Consumer Functional Interface

- A Consumer accepts a parameter & returns no results, e.g.,
  - public interface Consumer<T> { void accept(T t); }

  public interface Iterable<T> {
      ...
      default void forEach(Consumer<? super T> action) {
          for (T t : this) {
              action.accept(t);
          }
      }
  }

Here’s how the forEach() method uses the Consumer passed to it
A `Consumer` accepts a parameter & returns no results, e.g.,

```java
public interface Consumer<T> { void accept(T t); }
```

```java
public interface Iterable<T> {
    ...
    default void forEach(Consumer<? super T> action) {
        for (T t : this) {
            action.accept(t);
        }
    }
}
```

The consumer parameter is bound to the `System.out::println` method reference.
Overview of the Consumer Functional Interface

• A `Consumer` accepts a parameter & returns no results, e.g.,
  • `public interface Consumer<T> { void accept(T t); }`

```java
public interface Iterable<T> {
    ...
    default void forEach(Consumer<? super T> action) {
        for (T t : this) {
            action.accept(t);
        }
    }
}
```

• The `accept()` method is replaced by the call to `System.out.println()`
End of Understand the Consumer Java Functional Interface