# Java 8 Functional Interfaces

## Predicate

Douglas C. Schmidt

# Learning Objectives in This Lesson

- Recognize foundational functional programming features in Java 8, e.g.,
  - Lambda expressions
  - Method & constructor references
  - Key functional interfaces
    - Predicate

**Interface Predicate<T>**

**Type Parameters:**

T - the type of the input to the predicate

**Functional Interface:**

This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

---

```
@FunctionalInterface
public interface Predicate<T>
```

Represents a predicate (boolean-valued function) of one argument.

This is a functional interface whose functional method is `test(Object)`.

Predicate

Douglas C. Schmidt

# Overview of Functional Interfaces

# Overview of Common Functional Interfaces: Predicate

- A *Predicate* performs a test that returns true or false, e.g.,
  - `public interface Predicate<T> { boolean test(T t); }`

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

# Overview of Common Functional Interfaces: Predicate

- A *Predicate* performs a test that returns true or false, e.g.,
  - `public interface Predicate<T> { boolean test(T t); }`

*Predicate is a generic interface that is parameterized by one reference type*

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

# Overview of Common Functional Interfaces: Predicate

- A *Predicate* performs a test that returns true or false, e.g.,
  - `public interface Predicate<T> { boolean test(T t); }`

*Its single abstract method is passed a parameter of type T & returns boolean*

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

# Overview of Common Functional Interfaces: Predicate

- A *Predicate* performs a test that returns true or false, e.g.,
  - `public interface Predicate<T> { boolean test(T t); }`

*The signature of the abstract method of the functional interface (called the "function descriptor") describes the signature of the lambda expression.*

See www.javabrahman.com/java-8/function-descriptors-java-8-explained

# Overview of Common Functional Interfaces: Predicate

- A *Predicate* performs a test that returns true or false, e.g.,
  - `public interface Predicate<T> { boolean test(T t); }`

```
Map<String, Integer> makeMap() {
   return new ConcurrentHashMap<String, Integer>() { {
     put("Larry", 100); put("Curly", 90); put("Moe", 110);
   }};
}

Map<String, Integer> iqMap = makeMap();

System.out.println(iqMap);

iqMap.entrySet().removeIf(entry -> entry.getValue() <= 100);

System.out.println(iqMap);
```

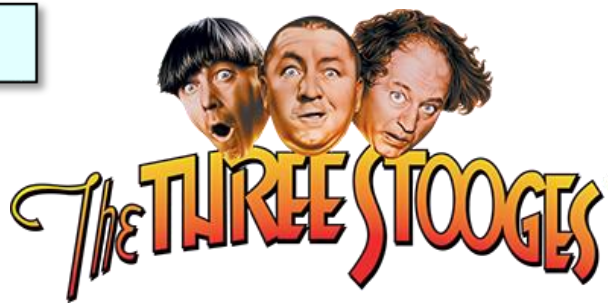
See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex10

# Overview of Common Functional Interfaces: Predicate

- A *Predicate* performs a test that returns true or false, e.g.,
  - `public interface Predicate<T> { boolean test(T t); }`

```
Map<String, Integer> makeMap() {
    return new ConcurrentHashMap<String, Integer>() { {
      put("Larry", 100); put("Curly", 90); put("Moe", 110);
    }};
}

Map<String, Integer> iqMap = makeMap();

System.out.println(iqMap);

iqMap.entrySet().removeIf(entry -> entry.getValue() <= 100);

System.out.println(iqMap);
```

*Create a map of "stooges" & their IQs!*




See en.wikipedia.org/wiki/The_Three_Stooges

# Overview of Common Functional Interfaces: Predicate

- A *Predicate* performs a test that returns true or false, e.g.,
  - `public interface Predicate<T> { boolean test(T t); }`

```
Map<String, Integer> makeMap() {
    return new ConcurrentHashMap<String, Integer>() { {
      put("Larry", 100); put("Curly", 90); put("Moe", 110);
    }};
}

Map<String, Integer> iqMap = makeMap();

System.out.println(iqMap);

iqMap.entrySet().removeIf(entry -> entry.getValue() <= 100);

System.out.println(iqMap);
```

*This predicate lambda removes all entries with iq <= 100.*

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

# Overview of Common Functional Interfaces: Predicate

- A *Predicate* performs a test that returns true or false, e.g.,
  - `public interface Predicate<T> { boolean test(T t); }`

```
Map<String, Integer> makeMap() {
    return new ConcurrentHashMap<String, Integer>() { {
      put("Larry", 100); put("Curly", 90); put("Moe", 110);
    }};
}

Map<String, Integer> iqMap = makeMap();

System.out.println(iqMap);

iqMap.entrySet().removeIf(entry -> entry.getValue() <= 100);

System.out.println(iqMap);
```

*This lambda implements the abstract test() method of Predicate directly inline*

# Overview of Common Functional Interfaces: Predicate

- A *Predicate* performs a test that returns true or false, e.g.,
  - `public interface Predicate<T> { boolean test(T t); }`

```
Map<String, Integer> makeMap() {
    return new ConcurrentHashMap<String, Integer>() { {
      put("Larry", 100); put("Curly", 90); put("Moe", 110);
    }};
}

Map<String, Integer> iqMap = makeMap();

System.out.println(iqMap);

iqMap.entrySet().removeIf(entry -> entry.getValue() <= 100);

System.out.println(iqMap);
```

*entry is short for (EntrySet <String, Integer> entry) via Java 8 type inference.*

See docs.oracle.com/javase/tutorial/java/generics/genTypeInference.html

# Overview of Common Functional Interfaces: Predicate

- A *Predicate* performs a test that returns true or false, e.g.,
  - `public interface Predicate<T> { boolean test(T t); }`

```
interface Collection<E> {
  ...
  default boolean removeIf(Predicate<? super E> filter) {
    ...
    final Iterator<E> each = iterator();
    while (each.hasNext()) {
      if (filter.test(each.next())) {
        each.remove();
      ...
```

Here's how the removeIf() method uses the predicate passed to it

# Overview of Common Functional Interfaces: Predicate

- A *Predicate* performs a test that returns true or false, e.g.,
  - `public interface Predicate<T> { boolean test(T t); }`

```
interface Collection<E> {
  ...
  default boolean removeIf(Predicate<? super E> filter) {
    ...
    final Iterator<E> each = iterator();
    while (each.hasNext()) {
      if (filter.test(each.next())) {
        each.remove();
```

*Default methods enable adding new functionality to the interfaces of libraries & ensure binary compatibility with code written for older versions of those interfaces.*

See docs.oracle.com/javase/tutorial/java/IandI/defaultmethods.html

# Overview of Common Functional Interfaces: Predicate

- A *Predicate* performs a test that returns true or false, e.g.,
  - `public interface Predicate<T> { boolean test(T t); }`

```
interface Collection<E> {
  ...
  default boolean removeIf(Predicate<? super E> filter) {
    ...
    final Iterator<E> each = iterator();
    while (each.hasNext()) {
      if (filter.test(each.next())) {
        each.remove();
      ...
```

*'super' is a lower-bounded wildcard that restricts the unknown type to be a specific type or a super type of that type.*

See docs.oracle.com/javase/tutorial/java/generics/lowerBounded.html

# Overview of Common Functional Interfaces: Predicate

- A *Predicate* performs a test that returns true or false, e.g.,
  - `public interface Predicate<T> { boolean test(T t); }`

```
interface Collection<E> {
  ...
  default boolean removeIf(Predicate<? super E> filter) {
    ...
    final Iterator<E> each = iterator();
    while (each.hasNext()) {
      if (filter.test(each.next())) {
        each.remove();
      ...
```

*entry -> entry.getValue() <= 100*

This predicate parameter is bound to the lambda expression passed to it.

# Overview of Common Functional Interfaces: Predicate

- A *Predicate* performs a test that returns true or false, e.g.,
  - `public interface Predicate<T> { boolean test(T t); }`

```
interface Collection<E> {
  ...
  default boolean removeIf(Predicate<? super E> filter) {
    ...
    final Iterator<E> each = iterator();
    while (each.hasNext()) {
      if (filter.test(each.next())) {
        each.remove();
      ...
```

```
if (each.next().getValue() <= 100)
```

The ‘entry’ in the lambda predicate is replaced by the parameter to test().

# Overview of Common Functional Interfaces: Predicate

- It's also possible to compose predicates.
  - `public interface Predicate<T> { boolean test(T t); }`

```
Map<String, Integer> iqMap = makeMap();

System.out.println(iqMap);

Predicate<ConcurrentMap.Entry<String, Integer>> lowIq =
  entry -> entry.getValue() <= 100;
Predicate<ConcurrentMap.Entry<String, Integer>> curly =
  entry -> entry.getKey().equals("Curly");

iqMap.entrySet().removeIf(lowIq.and(curly));

System.out.println(iqMap);
```

*Create two predicate objects.*

# Overview of Common Functional Interfaces: Predicate

- It's also possible to compose predicates.
  - `public interface Predicate<T> { boolean test(T t); }`

```
Map<String, Integer> iqMap = makeMap();

System.out.println(iqMap);

Predicate<ConcurrentMap.Entry<String, Integer>> lowIq =
  entry -> entry.getValue() <= 100;
Predicate<ConcurrentMap.Entry<String, Integer>> curly =
  entry -> entry.getKey().equals("Curly");

iqMap.entrySet().removeIf(lowIq.and(curly));

System.out.println(iqMap);
```

*Compose two predicates!*

See docs.oracle.com/javase/8/docs/api/java/util/function/Predicate.html#and