

Understand the Java Predicate 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 Part of the Lesson

- Recognize foundational functional programming features in Java, 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)`.

Learning Objectives in this Part of the Lesson

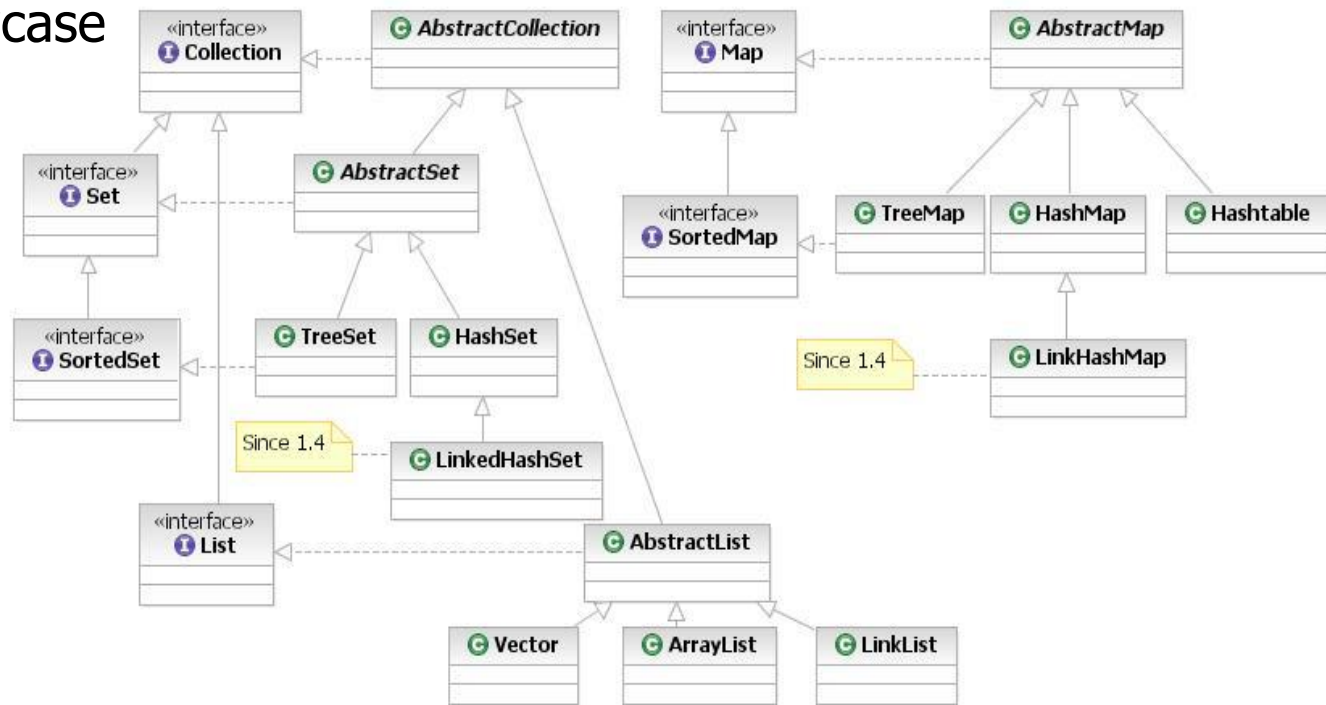
- Recognize foundational functional programming features in Java
- Understand how to apply Java predicates in concise example programs



See github.com/douglasraigschmidt/LiveLessons/tree/master/Java8

Learning Objectives in this Part of the Lesson

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



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

Overview of the Predicate Functional Interface

Overview of the Predicate Functional Interface

- 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 the Predicate Functional Interface

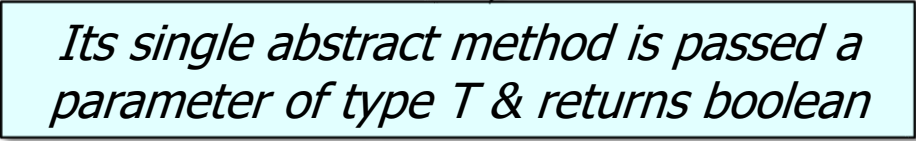
- A *Predicate* performs a test that returns true or false, e.g.,
 - `public interface Predicate<T> { boolean test(T t); }`



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

Overview of the Predicate Functional Interface

- 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

Overview of the Predicate Functional Interface

- 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.

Overview of the Predicate Functional Interface

- 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);
```

# Overview of the Predicate Functional Interface

- 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);  
    }  
};
```

Create a map of "stooges" & their IQs!

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

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

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

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



See en.wikipedia.org/wiki/The_Three_Stooges

Overview of the Predicate Functional Interface

- 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*

# Overview of the Predicate Functional Interface

- 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 the Predicate Functional Interface

- 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 (Entry
<String, Integer> entry)
via Java type inference*

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

Overview of the Predicate Functional Interface

- 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 the Predicate Functional Interface

- 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 the Predicate Functional Interface

- 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 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 the Predicate Functional Interface

- 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) {
```

```
...
```

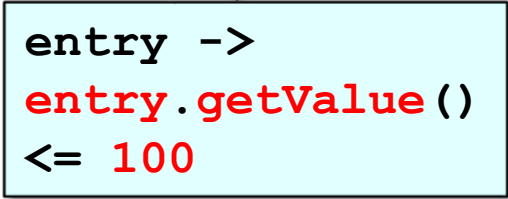
```
final Iterator<E> each = iterator();
```

```
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 the Predicate Functional Interface

- 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()

Composing Predicates

Composing Predicates

- It's also possible to compose predicates.

- ```
public interface Predicate<T> { boolean test(T t); }
```

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

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

*Create two predicate objects.*

```
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);
```



# Composing Predicates

- 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](https://docs.oracle.com/javase/8/docs/api/java/util/function/Predicate.html#and)

---

# End of Understand the Java Predicate Functional Interface