# The Java Predicate Functional Interface

Douglas C. Schmidt

d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science

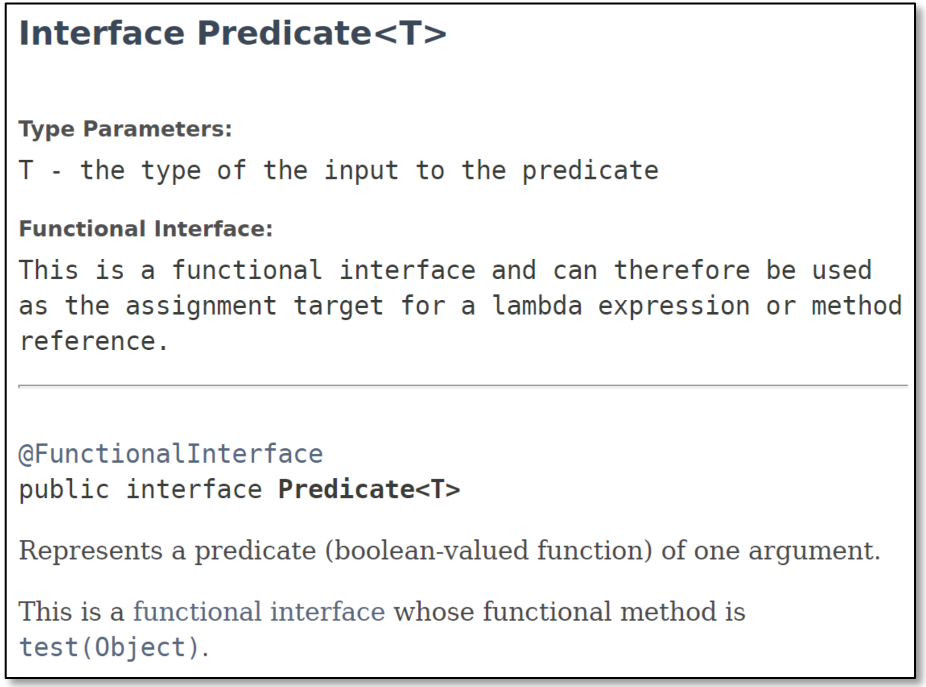
Institute for Software Integrated Systems

Vanderbilt University

Nashville, Tennessee, USA

## Learning Objectives in this Part of the Lesson

- Understand the Predicate functional interface in Java & recognize how it can be used in conjunction with lambda expressions & method references


**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)`.

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

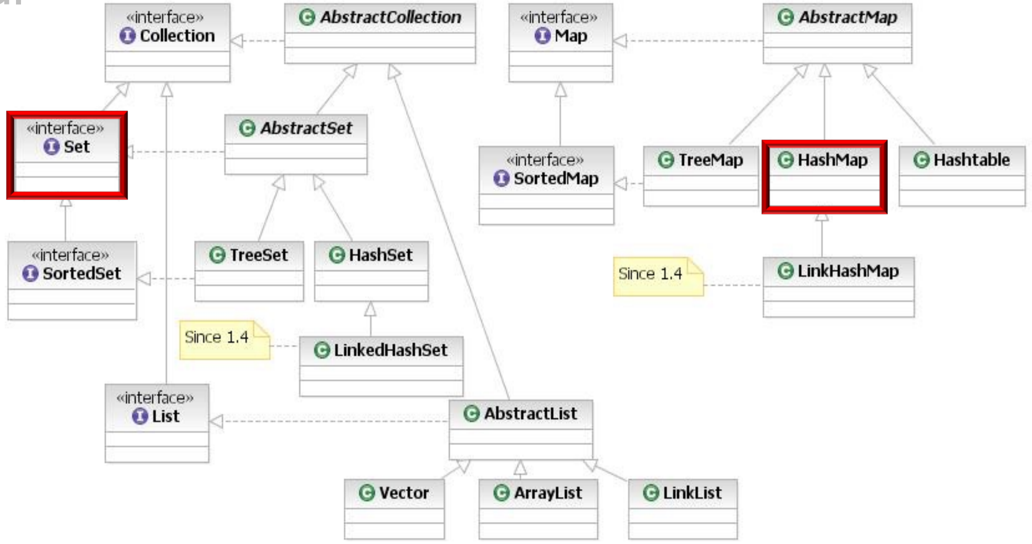
## Learning Objectives in this Part of the Lesson

- Understand the Predicate functional interface in Java & recognize how it can be used in conjunction with lambda expressions & method references
- Know how to apply Java Predicate in a concise example

See github.com/douglascraigschmidt/ModernJava/tree/main/FP/ex8

## Learning Objectives in this Part of the Lesson

- Understand the Predicate functional interface in Java & recognize how it can be used in conjunction with lambda expressions & method references
- Know how to apply Java Predicate in a concise example
  - This example showcases the HashMap class & Set interface in the Java collections framework



See [docs.oracle.com/javase/8/docs/technotes/guides/collections](docs.oracle.com/javase/8/docs/technotes/guides/collections)

## Learning Objectives in this Part of the Lesson

- Understand the Predicate functional interface in Java & recognize how it can be used in conjunction with lambda expressions & method references
- Know how to apply Java Predicate in a concise example
- Recognize how to compose Java Predicate objects

See tutorials.jenkov.com/java-functional-programming/functional-composition.html

## 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 a functional interface (called the "function descriptor") describes the signature of the lambda expression or method reference passed as a parameter to another Java method*

See [www.javabrahman.com/java-8/function-descriptors-java-8-explained](www.javabrahman.com/java-8/function-descriptors-java-8-explained)

## Applying the Predicate Functional Interface

# Applying the Predicate Functional Interface

- This example shows the use of predicate lambda expressions in the context of the Java HashMap removeIf() method

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

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

System.out.println(stooges);

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

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

See [github.com/douglascraigschmidt/ModernJava/tree/main/FP/ex8](https://github.com/douglascraigschmidt/ModernJava/tree/main/FP/ex8)

# Applying the Predicate Functional Interface

- This example shows the use of predicate lambda expressions in the context of the Java HashMap removeIf() method

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

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

System.out.println(stooges);

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

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

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

See en.wikipedia.org/wiki/The_Three_Stooges

# Applying the Predicate Functional Interface

- This example shows the use of predicate lambda expressions in the context of the Java HashMap removeIf() method

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

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

System.out.println(stooges);

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

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

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

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

## Applying the Predicate Functional Interface

- This example shows the use of predicate lambda expressions in the context of the Java HashMap removeIf() method

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

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

System.out.println(stooges);

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

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

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

# Applying the Predicate Functional Interface

- This example shows the use of predicate lambda expressions in the context of the Java HashMap removeIf() method

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

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

System.out.println(stooges);

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

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

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

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

## How Collection Uses the Predicate Functional Interface

## How Collection Uses the Predicate Functional Interface

- Here's how the Java Collection interface's removeIf() method uses the Predicate passed to it

```
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();
      ...
```

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

## How Collection Uses the Predicate Functional Interface

- Here's how the Java Collection interface's removeIf() method uses the Predicate passed to it

```
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

# How Collection Uses the Predicate Functional Interface

- Here’s how the Java Collection interface’s removeIf() method uses the Predicate passed to it

```
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

## How Collection Uses the Predicate Functional Interface

- Here's how the Java Collection interface's removeIf() method uses the Predicate passed to it

```
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

# How Collection Uses the Predicate Functional Interface

- Here's how the Java Collection interface's removeIf() method uses the Predicate passed to it

```
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 via and() & or() methods
  - `public interface Predicate<T> { boolean test(T t); }`

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

System.out.println(stooges);

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

stooges.entrySet().removeIf(iq.and(curly));

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

See tutorials.jenkov.com/java-functional-programming/functional-composition.html

# Composing Predicates

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

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

System.out.println(stooges);

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

stooges.entrySet().removeIf(iq.and(curly));

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

*Create two predicate objects.*

See github.com/douglascraigschmidt/ModernJava/tree/main/FP/ex8

## Composing Predicates

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

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

System.out.println(stooges);

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

stooges.entrySet().removeIf(iq.and(curly));

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

*Compose two predicates!*

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

## End of the Java Predicate Functional Interface