Other Properties of Java Functional Interfaces

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

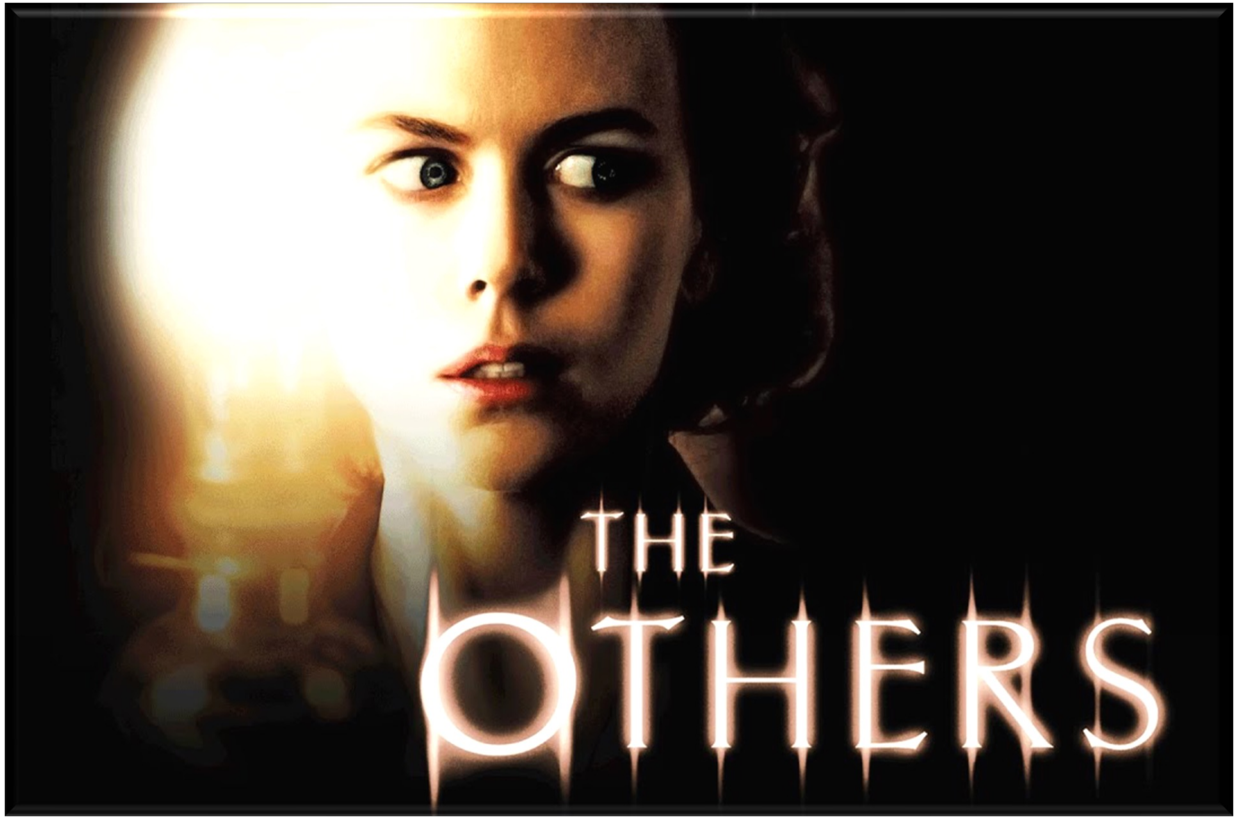
Professor of Computer Science

Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Lesson

- Understand other properties of Java functional interfaces



Learning Objectives in this Lesson

- Understand other properties of Java functional interfaces
  - Java's Comparator interface is used as an example

```
@FunctionalInterface
interface Comparator<T> {
  int compare(T o1, T o2);

  boolean equals(Object obj);

  default Comparator<T> reversed() {
    return Collections.reverseOrder(this);
  }

  static <T extends Comparable<? super T>>
  Comparator<T> reverseOrder()
  { return Collections.reverseOrder(); }
  ...
```

This interface is used to impose a total ordering on a collection of objects

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

Other Properties of Functional Interfaces

Other Properties of Functional Interfaces

- A functional interface can optionally be marked with an annotation

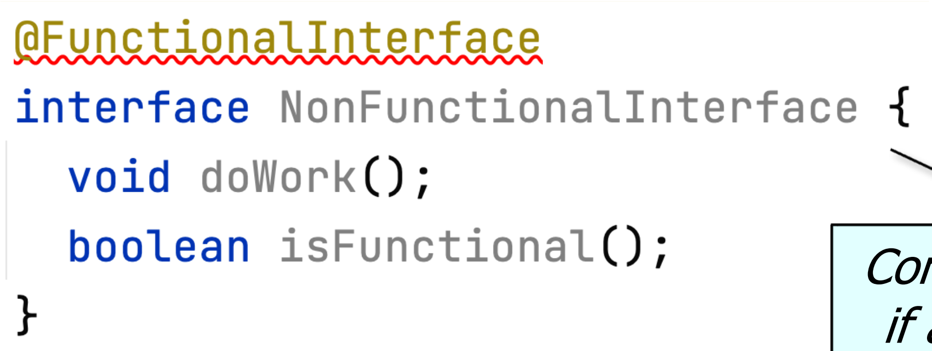
```
@FunctionalInterface
interface Comparator<T> {
  int compare(T o1, T o2);

  boolean equals(Object obj);

  default Comparator<T> reversed()
  { return Collections.reverseOrder(this); }

  static <T extends Comparable<? super T>>
  Comparator<T> reverseOrder()
  { return Collections.reverseOrder(); }
  ...
```

This annotation type indicates that this interface type declaration is intended as a functional interface

See docs.oracle.com/javase/8/docs/api/java/lang/FunctionalInterface.html

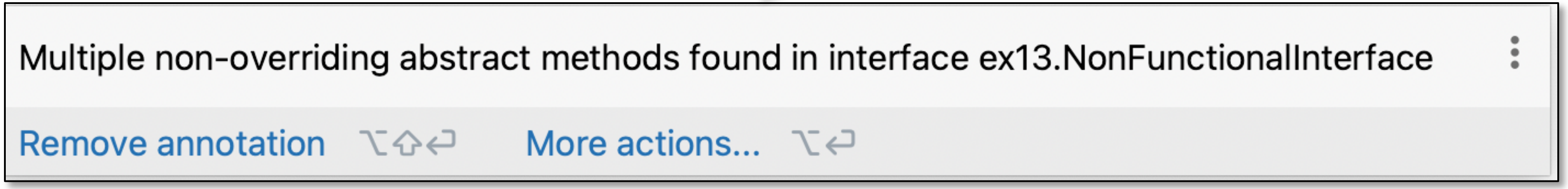
Other Properties of Functional Interfaces

- A functional interface can optionally be marked with an annotation



```
@FunctionalInterface
interface NonFunctionalInterface {
  void doWork();
  boolean isFunctional();
}
```

Compilers must generate error messages if an annotated type doesn't satisfy the requirements of a functional interface



Multiple non-overriding abstract methods found in interface ex13.NonFunctionalInterface

Other Properties of Functional Interfaces

- Functional interfaces can have abstract, default, and/or static methods

```
@FunctionalInterface
interface Comparator<T> {
  int compare(T o1, T o2);

  boolean equals(Object obj);

  default Comparator<T> reversed()
  { return Collections.reverseOrder(this); }

  static <T extends Comparable<? super T>>
  Comparator<T> reverseOrder()
  { return Collections.reverseOrder(); }
  ...
```

Comparator is an example of a functional interface with a broad range of methods

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

Other Properties of Functional Interfaces

- Functional interfaces can have abstract, default, and/or static methods

```
@FunctionalInterface
interface Comparator<T> {
  int compare(T o1, T o2);

  boolean equals(Object obj);

  default Comparator<T> reversed()
  { return Collections.reverseOrder(this); }

  static <T extends Comparable<? super T>>
  Comparator<T> reverseOrder()
  { return Collections.reverseOrder(); }
  ...
```

The primary abstract method in this functional interface

Other Properties of Functional Interfaces

- Functional interfaces can have abstract, default, and/or static methods

```
@FunctionalInterface
interface Comparator<T> {
  int compare(T o1, T o2);

  boolean equals(Object obj);

  default Comparator<T> reversed()
  { return Collections.reverseOrder(this); }

  static <T extends Comparable<? super T>>
  Comparator<T> reverseOrder()
  { return Collections.reverseOrder(); }
  ...
```

Non-intuitively, Comparator has a second abstract method, yet is still considered a functional interface

See mkyong.com/java8/is-comparator-a-function-interface-but-it-has-two-abstract-methods

Other Properties of Functional Interfaces

- Functional interfaces can have abstract, default, and/or static methods

```
@FunctionalInterface
interface Comparator<T> {
  int compare(T o1, T o2);

  boolean equals(Object obj);

  default Comparator<T> reversed()
  { return Collections.reverseOrder(this); }

  static <T extends Comparable<? super T>>
  Comparator<T> reverseOrder()
  { return Collections.reverseOrder(); }
  ...
```

An abstract method that overrides a public java.lang.Object method does not count as part of the interface's abstract method count

See docs.oracle.com/javase/8/docs/api/java/lang/FunctionalInterface.html

Other Properties of Functional Interfaces

- Functional interfaces can have abstract, default, and/or static methods

```
@FunctionalInterface
interface Comparator<T> {
  int compare(T o1, T o2);

  boolean equals(Object obj);

  default Comparator<T> reversed()
  { return Collections.reverseOrder(this); }

  static <T extends Comparable<? super T>>
  Comparator<T> reverseOrder()
  { return Collections.reverseOrder(); }
  ...
```

A default method provides an initial definition, which can be overridden (or not) by implementation classes (but not by any extending interfaces)

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

Other Properties of Functional Interfaces

- Functional interfaces can have abstract, default, and/or static methods

```
@FunctionalInterface
interface Comparator<T> {
  int compare(T o1, T o2);

  boolean equals(Object obj);

  default Comparator<T> reversed()
  { return Collections.reverseOrder(this); }

  static <T extends Comparable<? super T>>
  Comparator<T> reverseOrder()
  { return Collections.reverseOrder(); }
  ...
```

```
threads.sort
  (Comparator
   .comparing(Thread::getName)
   .reversed());
```

See earlier lesson on "*The Java Supplier Functional Interface: Case Study ex13*"

Other Properties of Functional Interfaces

- Functional interfaces can have abstract, default, and/or static methods

```
@FunctionalInterface
interface Comparator<T> {
  int compare(T o1, T o2);

  boolean equals(Object obj);

  default Comparator<T> reversed()
  { return Collections.reverseOrder(this); }

  static <T extends Comparable<? super T>>
  Comparator<T> reverseOrder()
  { return Collections.reverseOrder(); }
  ...
```

A static method provides the one-&-only implementation

See www.techopedia.com/definition/24034/static-method-java

Other Properties of Functional Interfaces

- Functional interfaces can have abstract, default, and/or static methods

```
@FunctionalInterface
interface Comparator<T> {
  int compare(T o1, T o2);

  boolean equals(Object obj);

  default Comparator<T> reversed()
  { return Collections.reverseOrder(this); }

  static <T extends Comparable<? super T>>
  Comparator<T> reverseOrder()
  { return Collections.reverseOrder(); }
  ...
```

```
threads.sort
  (Comparator
   .reverseOrder());
```

See www.geeksforgeeks.org/comparator-reverseorder-method-in-java-with-examples

Other Properties of Functional Interfaces

- Functional interfaces can have abstract, default, and/or static methods

```
@FunctionalInterface
interface Comparator<T> {
  int compare(T o1, T o2);

  boolean equals(Object obj);

  default Comparator<T> reversed()
  { return Collections.reverseOrder(this); }

  static <T extends Comparable<? super T>>
  Comparator<T> reverseOrder()
  { return Collections.reverseOrder(); }
  ...
```

There are no limits on the number of default and/or static methods!

End of Other Properties of Java Functional Interfaces