Benefits of Java Lambda Expressions

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 how lambda expressions provide a foundational functional programming feature in Modern Java
- Know the benefits of applying Java lambda expressions

Learning Objectives in this Part of the Lesson

- Understand how lambda expressions provide a foundational functional programming feature in Modern Java
- Know the benefits of applying Java lambda expressions, e.g.
  - See how Lambda expressions can work with multiple parameters more compactly

```
Arrays.sort(nameArray,
              new Comparator<String>(){
    public int compare
       (String s, String t) {
         return s.toLowerCase()
            .compareTo
              (t.toLowerCase()); }

       });
```

vs

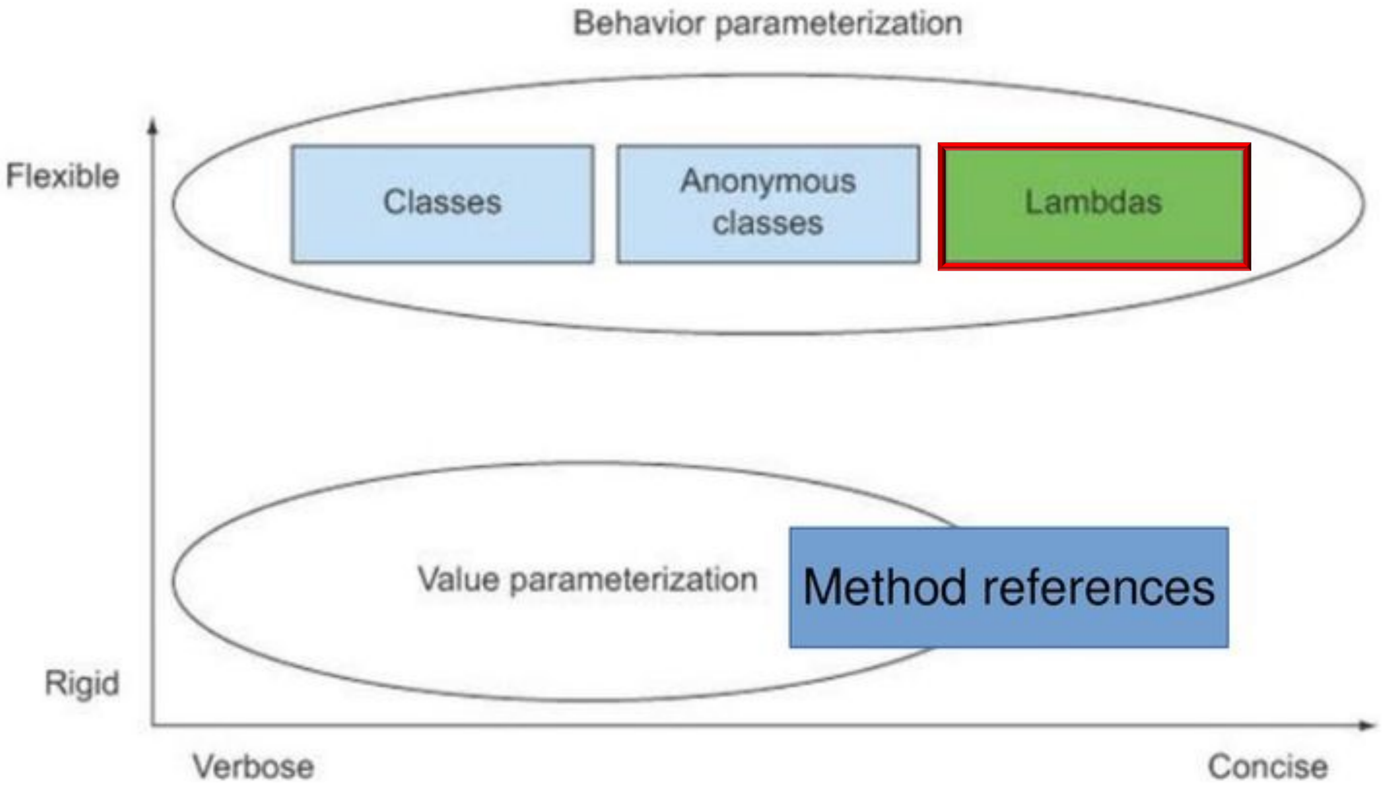
```
Arrays.sort(nameArray, (s, t) ->
            s.compareToIgnoreCase(t));
```

See github.com/douglascraigschmidt/ModernJava/tree/main/FP/ex5

Benefits of Lambda Expressions

Benefits of Lambda Expressions

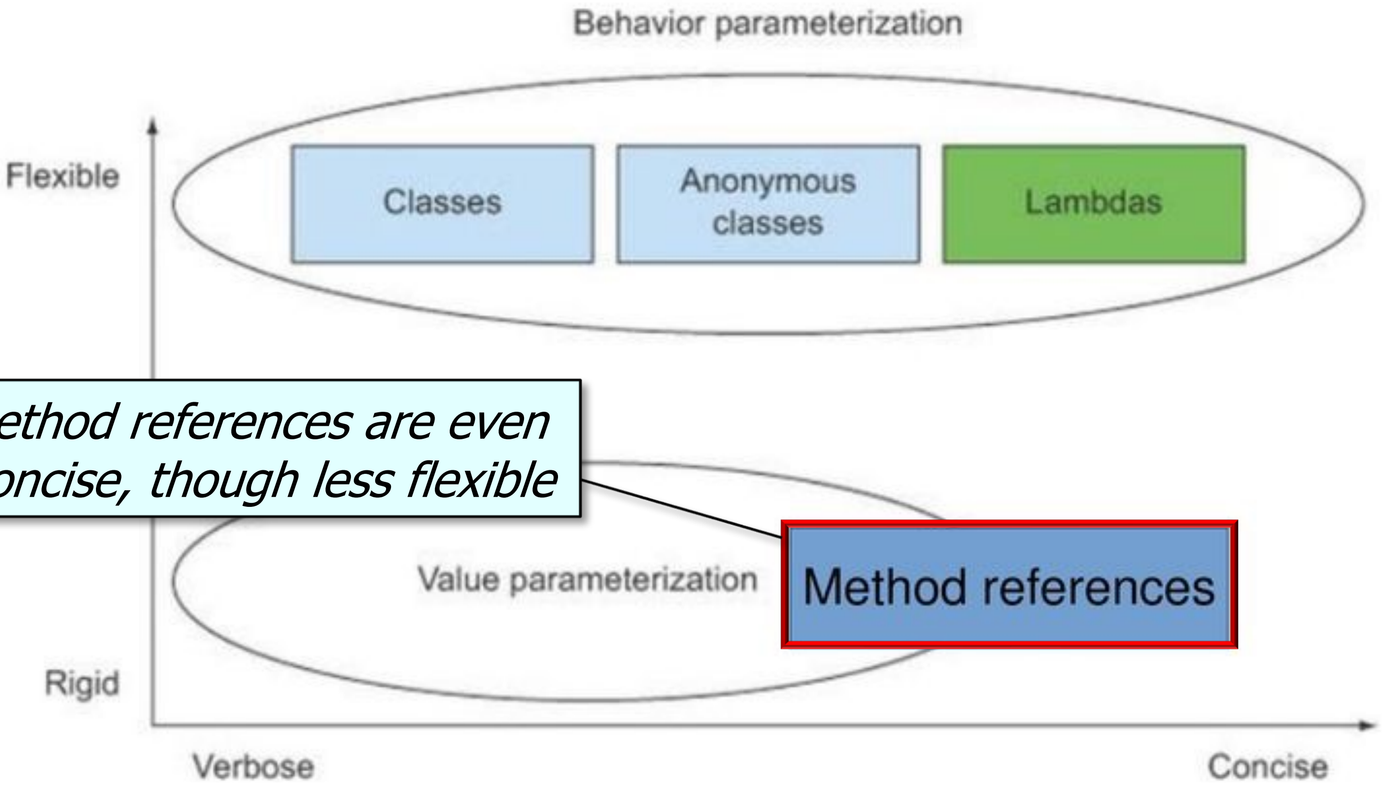
- Lambda expressions support flexible & concise "behavior parameterization"



See blog.indrek.io/articles/java-8-behavior-parameterization

Benefits of Lambda Expressions

- Lambda expressions support flexible & concise "behavior parameterization"



Java Method references are even more concise, though less flexible

See upcoming lesson *on "Java Method References"*

Benefits of Lambda Expressions

- Lambda expressions can work with multiple parameters in a *much* more compact manner than anonymous inner classes

```
String[] nameArray = {"Barbara", "James", "Mary", "John",
                "Robert", "Michael", "Linda", "james", "mary"};


Arrays.sort(nameArray, new Comparator<String>(){
  public int compare(String s, String t) { return
    s.toLowerCase().compareTo(t.toLowerCase()); }});
```

vs

```
Arrays.sort(nameArray,
            (s, t) -> s.compareToIgnoreCase(t));
```

See earlier lesson on "*Overview of Java Lambda Expressions*"

Benefits of Lambda Expressions

- Lambda expressions can work with multiple parameters in a *much* more compact manner than anonymous inner classes, e.g.

```
String[] nameArray = {"Barbara", "James", "Mary", "John",
                      "Robert", "Michael", "Linda", "james", "mary"};


Arrays.sort(nameArray, new Comparator<String>(){
  public int compare(String s, String t) { return
    s.toLowerCase().compareTo(t.toLowerCase()); }});
```

Extraneous syntax for anonymous inner class

See github.com/douglascraigschmidt/ModernJava/tree/main/FP/ex5

Benefits of Lambda Expressions

- Lambda expressions can work with multiple parameters in a *much* more compact manner than anonymous inner classes, e.g.

```
String[] nameArray = {"Barbara", "James", "Mary", "John",
                      "Robert", "Michael", "Linda", "james", "mary"};
```

```
Arrays.sort(nameArray, new Comparator<String>(){
  public int compare(String s, String t) { return
    s.toLowerCase().compareTo(t.toLowerCase()); }});
```

VS

```
Arrays.sort(nameArray,
            (s, t) -> s.compareToIgnoreCase(t));
```

This lambda expression omits the method name & extraneous syntax

Benefits of Lambda Expressions

- Lambda expressions can work with multiple parameters in a *much* more compact manner than anonymous inner classes, e.g.

```
String[] nameArray = {"Barbara", "James", "Mary", "John",
                      "Robert", "Michael", "Linda", "james", "mary"};


Arrays.sort(nameArray, new Comparator<String>(){
  public int compare(String s, String t) { return
    s.toLowerCase().compareTo(t.toLowerCase()); }});
```

VS

```
Arrays.sort(nameArray,
            (s, t) -> s.compareToIgnoreCase(t));
```

(s, t) is short for (String s, String t), which leverages Java's type inference capabilities

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

Benefits of Lambda Expressions

- Lambda expressions can work with multiple parameters in a *much* more compact manner than anonymous inner classes, e.g.

```
String[] nameArray = {"Barbara", "James", "Mary", "John",
                      "Robert", "Michael", "Linda", "james", "mary"};
```

```
Arrays.sort(nameArray, new Comparator<String>(){
  public int compare(String s, String t) { return
    s.toLowerCase().compareTo(t.toLowerCase()); }});
```

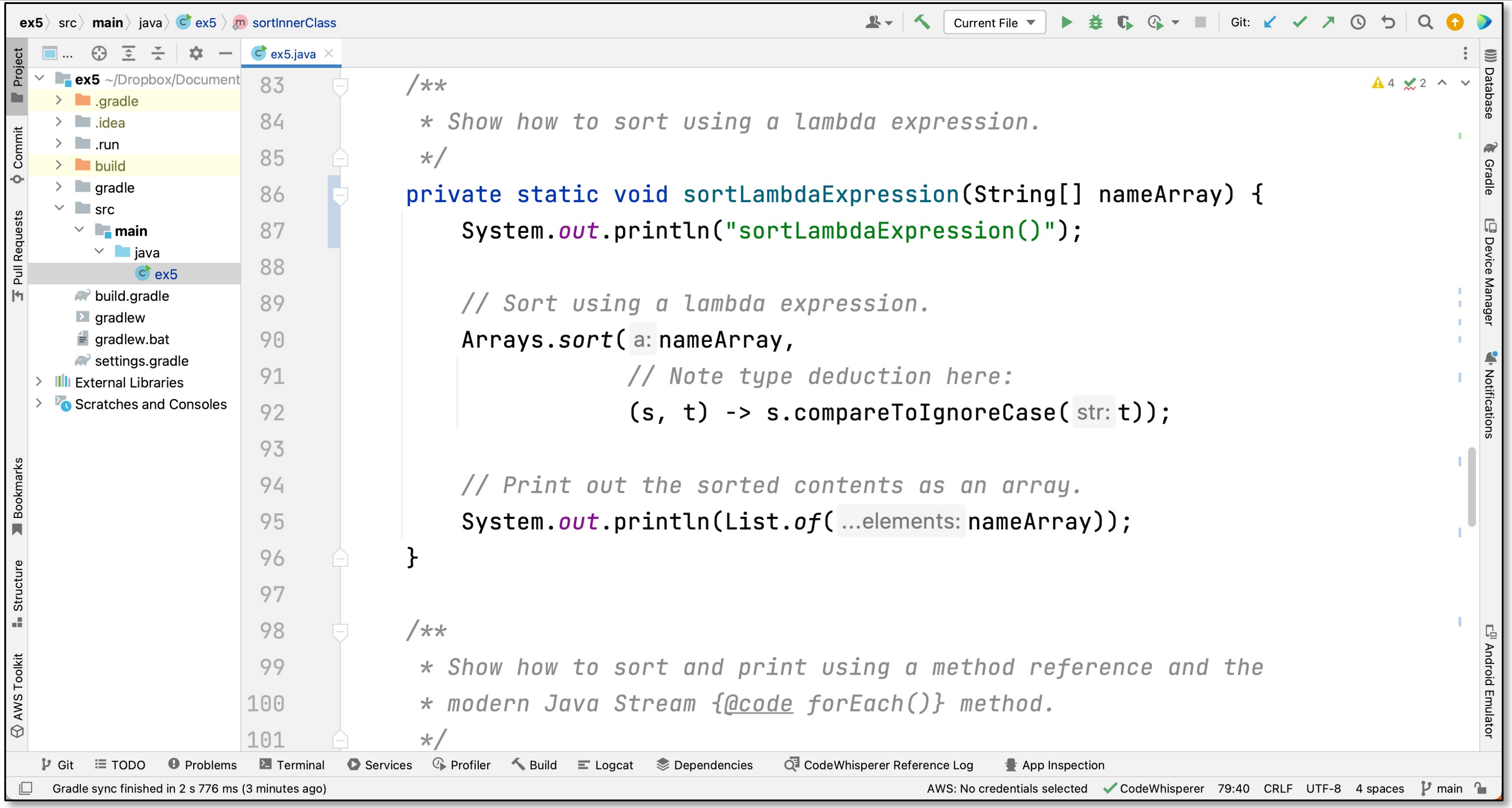
vs

```
Arrays.sort(nameArray,
            (s, t) -> s.compareToIgnoreCase(t));
```

It's therefore considered good practice to use lambda expressions when you can!

Applying Java Lambda Expressions in Case Study ex5

Applying Java Lambda Expressions in Case Study ex5



```java
    /**
     * Show how to sort using a lambda expression.
     */
    private static void sortLambdaExpression(String[] nameArray) {
        System.out.println("sortLambdaExpression()");

        // Sort using a lambda expression.
        Arrays.sort( a: nameArray,
                     // Note type deduction here:
                     (s, t) -> s.compareToIgnoreCase( str: t));

        // Print out the sorted contents as an array.
        System.out.println(List.of( ...elements: nameArray));
    }

    /**
     * Show how to sort and print using a method reference and the
     * modern Java Stream {@code forEach()} method.
     */
```

Gradle sync finished in 2 s 776 ms (3 minutes ago)

See github.com/douglascraigschmidt/ModernJava/tree/main/FP/ex5

End of Benefits of Java Lambda Expressions