

CSC 347 - Concepts of Programming Languages

Nested Classes

Instructor: Stefan Mitsch



Learning Objectives

- ❓ Functional features in an object-oriented language?
 - Identify nested and anonymous classes
 - Use object-oriented concepts to implement lambda expressions



"Higher-order functions" in Java 1.0

```
1 public class MyArray {
2     public int[] elems // ...
3
4     public MyArray map(IntMapper m) {
5         MyArray result = new MyArray();
6         for (int i = 0; i < elems.length; i++) {
7             result.elems[i] = m.apply(elems[i]);
8         }
9         return result;
10    }
11 }
```

```
1 interface IntMapper {
2     public int apply(int v);
3 }
4
5 class IntIncrement implements IntMapper {
6     public int apply(int v) {
7         return v + 1;
8     }
9 }
```

```
1 public static void main(String[] args) {
2     MyArray a = new MyArray();
3     // fill a
4     // passing in an instance of a named class
5     MyArray b = a.map(new IntIncrement());
6 }
```

❓ Can we avoid explicit named class?



"Higher-order functions" in Java 1.1

```
1 public class MyArray {
2     public int[] elems // ...
3
4     public MyArray map(IntMapper m) {
5         MyArray result = new MyArray();
6         for (int i = 0; i < elems.length; i++) {
7             result.elems[i] = m.apply(elems[i]);
8         }
9         return result;
10    }
11 }
```

```
1 interface IntMapper {
2     public int apply(int v);
3 }
```

```
1 public static void main(String[] args) {
2     MyArray a = new MyArray();
3     // fill a
4     // passing in an instance of an anonymous class
5     MyArray b = a.map(new IntMapper() {
6         public int apply(int v) { return v + 1; }
7     });
8 }
```



Higher-order functions in Java 1.8

```
1 public class MyArray {
2     public int[] elems // ...
3
4     public MyArray map(IntMapper m) {
5         MyArray result = new MyArray();
6         for (int i = 0; i < elems.length; i++) {
7             result.elems[i] = m.apply(elems[i]);
8         }
9         return result;
10    }
11 }
```

```
1 @FunctionalInterface
2 interface IntMapper {
3     public int apply(int v);
4 }
```

```
1 public static void main(String[] args) {
2     MyArray a = new MyArray();
3     // fill a
4     // passing in an instance of an anonymous class
5     MyArray b = a.map(v -> v + 1);
6 }
```



Java Functional Interface

- `java.util.function` defines many functional interfaces
- Also `java.awt.event`, `Runnable`, `Callable`, etc

```
1 @FunctionalInterface
2 interface Function<T,R> {
3     public R apply (T x);
4 }
```

```
1 Function<Integer,Integer> intIncrement = new Function<> () {
2     public Integer apply (Integer x) { return x + 1; }
3 };
```

```
1 Function<Integer,Integer> intIncrement = x -> x + 1;
```



Nonfunctional Interfaces in Java

- Some event interfaces have multiple methods
- Cannot use lambda notation: not clear which method is meant

```
1 @FunctionalInterface
2 interface MouseListener { /* from java.awt.event */
3     void mouseClicked (MouseEvent e);
4     void mouseEntered (MouseEvent e);
5     // ...
6 }
```

```
1 MouseListener.java:1: error: Unexpected @FunctionalInterface annotation
2 @FunctionalInterface
3 ^
4     MouseListener is not a functional interface
5     multiple non-overriding abstract methods found in interface MouseListener
```



Implementing Nested Classes

- `javac` (the compiler) supports nested classes
- `java` (the JVM) does not
- Compiler creates new classes with `$` in name



Nested Classes

- ❓ Should we allow nested classes with free variables?
- ❓ What could it be useful for?
- ❓ What do we have to be careful about?



1 33311000000000000000000000000000000022222222222222224444222...

- ❓ Which line seems useless in the code above?



Java Threads 1.8

```
1 public static void main (String[] args) {  
2     for (int i = 0; i < 5; i++) {  
3  
4         new Thread (() -> {  
5             while (true) System.out.print (i);  
6         }).start ();  
7     }  
8 }
```

- Compile error

```
1 error: local variables referenced from an inner class must be final or effectively final  
2     System.out.print (i);  
3                     ^
```



- ```
1 import threading
2
3 for i in range(5):
4 def worker():
5 while True:
6 print(i, end="")
7
8 threading.Thread(target=worker).start()
```

[illegible]



- ```
1 import threading
2
3 for i in range(5):
4     def worker():
5         x = i
6         while True:
7             print(x, end="")
8
9     threading.Thread(target=worker).start()
```

1 3331100000000000000000000000000000002222222222222224444222...



Some Languages Allow Mutable Variables

- Scala

```
1 for i <- (1 to 4) do
2   new Thread (() => {
3     while true do print (i)
4   }).start
```

1 33311000000000000000000000000000002222222222222224444222...

❗ Scala closure mechanism correctly treats `i` as effectively final in the loop body



Summary

- Object-oriented languages support nested anonymous classes to implement functionality of restricted scope
- Syntax resembling lambda expressions of functional languages makes such classes convenient to use
- Classes with methods that have free variables: need closures



Java Graphics 1.8

```
1 public class MyGUI extends Frame {
2     private int count = 0;
3     public MyGUI() {
4         Button button = new Button ("Go"); add (button);
5         Label out = new Label ("000", Label.CENTER); add (out);
6         // functional interface: uses closure for count/this and out
7         button.addActionListener (e -> {
8             count += 1;
9             out.setText (String.format ("%03d", count));
10            out.repaint ();
11        });
12    }
13 }
```



Implementing Nested Classes

```
1 for (int i = 0; i < 5; i++) {  
2     int x = i;  
3     new Thread (new Runnable () {  
4         public void run () {  
5             while (true) System.out.print (x);  
6         }  
7     }).start ();  
8 }
```

```
1 $ javac Run2.java  
2 $ javap -private 'Run2$1'  
3 Compiled from "Run2.java"  
4 final class Run2$1 implements java.lang.Runnable {  
5     final int val$x;  
6     Run2$1(int);  
7     public void run();  
8 }
```