

CSC 347 - Concepts of Programming Languages

Nested Classes

Instructor: Stefan Mitsch



Learning Objectives

❓ How to restrict the scope of classes?

- Understand nested classes
- Understand object-oriented implementation of lambda expressions



Higher-order functions in OOP?

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

```
1 public static void main(String[] args) {
2     TenIntArray a = new TenIntArray();
3     for (int i = 0; i < 10; i++) { a.elems[i] = i; }
4     TenIntArray b = a.map(...)
5 }
```



Higher-order functions in Java 1.0

```
1 public class TenIntArray {
2     public int[] elems = new int[10];
3
4     public TenIntArray map(IntMapper m) {
5         TenIntArray result = new TenIntArray();
6         for (int i = 0; i < 10; 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     TenIntArray a = new TenIntArray();
3     for (int i = 0; i < 10; i++) { a.elems[i] = i; }
4     // passing in an instance of a named class
5     TenIntArray b = a.map(new IntIncrement());
6 }
```



Higher-order functions in Java 1.1

```
1 public class TenIntArray {
2     public int[] elems = new int[10];
3
4     public TenIntArray map(IntMapper m) {
5         TenIntArray result = new TenIntArray();
6         for (int i = 0; i < 10; 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     TenIntArray a = new TenIntArray();
3     for (int i = 0; i < 10; i++) { a.elems[i] = i; }
4     // passing in an instance of an anonymous class
5     TenIntArray 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 TenIntArray {
2     public int[] elems = new int[10];
3
4     public TenIntArray map(IntMapper m) {
5         TenIntArray result = new TenIntArray();
6         for (int i = 0; i < 10; 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     TenIntArray a = new TenIntArray();
3     for (int i = 0; i < 10; i++) { a.elems[i] = i; }
4     // passing in an instance of an anonymous class
5     TenIntArray b = a.map(v -> v + 1);
6 }
```



Nested Classes

- Added in Java 1.1 (1997)
- Improved in Java 1.8 (2014)
- Used in, for example:
 - Graphical User Interfaces
 - Concurrency APIs
 - Collections processing

Nested Classes

- ❓ Should we allow nested classes access to the environment?
- ❓ What could it be useful for?



Java Graphics 1.0

```
1 public class Swing1 extends Frame {
2     int count = 0;
3     public Swing1 () {
4         Button button = new Button ("Go"); add (button);
5         Label out = new Label ("000", Label.CENTER); add (out);
6         button.addActionListener (new MyActionListener(this, out));
7         // set size, layout, visible etc.
8     }
9 }
```

```
1 class MyActionListener implements ActionListener {
2     private Swing1 parent;
3     private Label out;
4     public MyActionListener (Swing1 parent, Label out) { this.parent = parent; this.out = out; }
5     public void actionPerformed (ActionEvent e) {
6         parent.count += 1;
7         out.setText (String.format ("%03d", parent.count));
8         out.repaint ();
9     }
10 }
```



Java Graphics 1.1

```
1 public class Swing2 extends Frame {
2     private int count = 0;
3     public Swing2 () {
4         Button button = new Button ("Go"); add (button);
5         Label out = new Label ("000", Label.CENTER); add (out);
6         // nested anonymous class
7         button.addActionListener (new ActionListener() {
8             public void actionPerformed (ActionEvent e) {
9                 count += 1;
10                out.setText (String.format ("%03d", count));
11                out.repaint ();
12            }
13        });
14    }
15 }
```



Java Graphics 1.8

```
1 public class Swing3 extends Frame {
2     private int count = 0;
3     public Swing3 () {
4         Button button = new Button ("Go"); add (button);
5         Label out = new Label ("000", Label.CENTER); add (out);
6         // functional interface
7         button.addActionListener (e -> {
8             count += 1;
9             out.setText (String.format ("%03d", count));
10            out.repaint ();
11        });
12    }
13 }
```

Nested Classes

 What do we have to be careful about?



```
1 class MyRunnable implements Runnable {
2     private int x;
3     public MyRunnable (int x) { this.x = x; }
4     public void run () {
5         while (true) System.out.print (x);
6     }
7 }
```

13



Java Threads 1.1

```
1 class Run5 {
2     public static void main (String[] args) {
3         for (int i = 0; i < 5; i++) {
4             new Thread (new Runnable () {
5                 public void run () {
6                     while (true) System.out.print (i);
7                 }
8             }).start ();
9         }
10    }
11 }
```

- Compile error

```
1 Run5.java:7: error: local variables referenced from an inner class must be final or effectively final
2         System.out.print (i);
3                        ^
```




1 33311000000000000000000000000000000022222222222222224444222...



Some Languages Allow Mutable Variables

- Python

```
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

- Perl (old versions)

```
1 use threads;
2 my $i;
3 for ($i = 0; $i < 5; $i++) {
4     my $th = threads->create (sub {
5         while (1) { print $i; }
6     });
7     $th->detach();
8 }
9 sleep (1);
```

[illegible]



- ```
1 use threads;
2 my $i;
3 for ($i = 0; $i < 5; $i++) {
4 my $x = $i
5 my $th = threads->create (sub {
6 while (1) { print $x; }
7 });
8 $th->detach();
9 }
10 sleep (1);
```

21



# Some Languages Allow Mutable Variables

- Perl 5

```
1 use threads;
2 my $i;
3 for ($i = 0; $i < 5; $i++) {
4 my $th = threads->create (sub {
5 while (1) { print $i; }
6 });
7 $th->detach();
8 }
9 sleep (1);
```

```
1 000000000000000022222222222222221111111111111111113333333333333333444444444444444444...
```



# 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 33311000000000000000000000000000000022222222222222224444222...



## Implementing Nested Classes

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





# 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 }
```



## Recall the 1.0 implementation

```
1 for (int i = 0; i < 5; i++) {
2 new Thread (new MyRunnable (i)).start ();
3 }
```

```
1 class MyRunnable implements Runnable {
2 private int x;
3 public MyRunnable (int x) { this.x = x; }
4 public void run () {
5 while (true) System.out.print (x);
6 }
7 }
```



# Java Functional Interface

- Java 8 introduced *lambda expressions*
- How to integrate this new feature with older APIs?

```
1 button.addActionListener (new ActionListener() {
2 public void actionPerformed (ActionEvent e) {
3 count += 1;
4 out.setText (String.format ("%03d", count));
5 out.repaint ();
6 }
7 });
```

```
1 button.addActionListener (e -> {
2 count += 1;
3 out.setText (String.format ("%03d", count));
4 out.repaint ();
5 });
```



# Java Functional Interface

- Java reference types marked as `@FunctionalInterface`

This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.



# 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) {
3 return x + 1;
4 }
5 };
```

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



# Collections Processing

- Java `map` method takes a `Function`

```
1 package java.util.stream;
2
3 interface Stream<T> {
4 ...
5 // Returns a stream consisting of the results of applying
6 // the given function to the elements of this stream
7 <R> Stream<R> map (Function<? super T,? extends R> mapper);
8 ...
9 }
```



# Collections Processing

- Provide `Stream` view of `List` in order to use map

```
1 public class Nested {
2 public static void main (String[] args) {
3 List<Integer> l = new ArrayList<> ();
4 Stream<Integer> s = l.stream();
5 l.add (1); l.add (2); l.add (3);
6 s.map (x -> x + 1)
7 .collect (Collectors.toList ())
8 .forEach (x -> System.out.format ("%2d ", x));
9 }
10 }
```

1 2 3 4



# Nonfunctional Interfaces in Java

- Some event interfaces have multiple methods
- Can't use lambda notation

```
1 @FunctionalInterface
2 interface MouseListener { /* from java.awt.event */
3 void mouseClicked (MouseEvent e);
4 void mouseEntered (MouseEvent e);
5 void mouseExited (MouseEvent e);
6 void mousePressed (MouseEvent e);
7 void mouseReleased (MouseEvent e);
8 }
```

```
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
```





## 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
- Decision: how to handle access to mutable data