# Lambda and "Functional" Programming

June 28, 2017

# Reading Quiz

# What is "Lambda" in Java 8?

A. A set of removals and changes to core libraries introduced in Java 8?

B. Support for new 64 bit operations

C. Convenient syntax for describing code / functionality

D. Enhanced support for type inference

# What is Lambda for?

A. To make types less tedious to deal with in Java

B. To make it less tedious to specify functionality thats unattached to data.

C. To improve garbage collection pauses

D. To enable multiple class inheritance

# Which is an example of using Lambda syntax in Java?

A. Lambda myLambo = new Lambda().stream(a, b);

B. (a, b) -> {/* some code */}

C. void (^myLambo)(a, b);

D. function (a, b) { return a + b; }

# What does the below refer to?

`StringImprovements::makeItCute`

A. The "makeItCute" method on the StringImprovements class

B. The "makeItCute" property on the StringImprovements class

C. The "makeItCute" argument to the StringImprovements constructor

D. The "makeItCute" subclass of the StringImprovements class

# What is the result?

```
String[] input = {"🍒", "🍒"};

List<String> popQuiz = Arrays.asList(input);

String result = popQuiz.stream()
    .map((x) -> {return x + "💣";})
    .collect(Collectors.joining("🐱"));

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

A. "🐱"

B. "🍒"

C. "🍒💣"

D. "🍒💣🐱🍒💣"

E. "🐱🐱🐱🐱🐱"

Done!

# The Plan

- Homework 3: Questions and Answers
- Homework 4 and Midterm
- Coding / productivity / setup bootcamp?
- Results of "hows it going" quiz
- Lambdas
- New and improved in class programming

# Checking in Results

# How do you find the pace of class?

A. Too slow: 38%

B. Just right: 54%

C. Too fast: 8%

# Me Laptop Coding?

A. More: 54%

B. Just right: 15%

C. Less: 31%

D. Please please stop: 0%

# Homeworks

A. Too easy: 23%

B. Just right: 69%

C. Too hard: 8%

# Coding in Groups

A. Helpful: 23%

B. More time covering existing material: 46%

C. More time covering new material: 31%

# Why Lambda

(ie OOP isn't all its cracked up to be)

# Whats the Problem(s)

- OOP Difficulties:
  - **One-off (Singleton) Functionality**
    Do I want to add one-offs in my taxonomy system
  - **Generalization**
    Functionality as a variable (do "something", and then X)
  - **Composition**
    Given a methods that do X and Y, how to build a method that does X and Y
  - **Predictability**
    OOP encapsulation makes things unpredictable
- Lambda tries to solve all these problems…

# OOP Problems: Singleton Functionality

- OOP is functionality and data
  - ie functionality that acts on state
- Where to put functionality w/o state?
  - Static methods, attached to classes
  - What are we "classifying"?
  - Some kind of mismatch…

# OOP Problems: Generalization

- Functionality as a variable
- Examples:
  - Filter items from a collection meeting criteria X
  - When this slow IO operation finishes, do X
  - Perform X operations at the same time, then continue
  - Do X when a user does Y

# OOP Problems: Composition

- Assume we have 26 useful methods (A…Z)
- How to build methods that do 2 of these?
  2^25 new static methods?
- Three of these?
  !3 new static methods?!

# OOP Problems: Predictability

- Encapsulation means hiding state
- Hiding state means output of methods is unpredictable
- Unpredictability makes "correctness" difficult

Difficult.java –>

# What is the result?

```
// adder is type DifficultAddition
System.out.print(adder.add(4));
```

A. 0

B. 4

C. 16

D. 64

```
public static class DifficultAddition {
  private Integer priorValue = 0;

  public Integer add(Integer someInteger) {
    Integer newInteger = priorValue + someInteger;
    this.priorValue = newInteger;
    return newInteger;
  }
}
```

# Easy.java –>

# What is the result?

```
System.out.print(EasyAddition.add(4, 0));
```

A. 0

B. 4

C. 16

D. 64

```
public static class EasyAddition {

  public static Integer add(Integer someInteger, Integer anotherInteger) {
    Integer newInteger = someInteger + anotherInteger;
    return newInteger;
  }
}
```

# Other Functional Benefits

- Performance (sometimes)
- Parallel correctness
- Less typing / cruft

# Lambda as a Solution

# Lambda:
# The Solution to Java's OO Woes

- Short hand for two things:
  - (…args) -> {code} for declaring Function instances
  - <Class>::<Method> for referring to existing static methods
- Recall the long way…

FileFilter.java –>

NestedTypes.java –>

# Lambda is "just" Shorthand

- java.util.function.*
- A bunch of interfaces that look like "Laterable"
- And short hand syntax for creating instances
- https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html

Lambda.java –>

# Where is this useful?

- Callbacks (when something happens **in the future**, do X)
- Aggregating other functionality
- One-off functions, ex:
  - comparing items
  - collection filtering
  - collection mapping

# Collections in Java8

- Collection Interface ("act on a bunch of things")
- Stream Interface ("apply lambdas to the collection")
    - Stream::forEach
    - Stream::map
    - Stream::filter
    - Stream::reduce
- Collectors ("give me a new object back")

LambdaFilters.java –>

LambdaMap.java –>

# Programming Task

- Create a command line program that works like "tree"

- Should accept two commands
  - -f <term>: filter the list
    only show paths that include the string "term"

  - -r <from> <to>: rewrite the list
    alter the names of the printed set, replacing instances of <from> to <to>

# Steps

1. Create a class called "FileTree"

    1.1.Constructor takes a File instance

    1.2. public ArrayList<String> toTree()

    1.3.Uses recursion and a static method

2. In your main function, create the above class, and then use lambdas to apply the other functionality