Lecture 13 – Functional Programming

08-671
Java for Application Programmers

February 23, 2016

Terry Lee
Assistant Teaching Professor
School of Computer Science

08-671 Lecture Topics

(subject to change – but only a little bit)

#1 Intro
#2 Primitive Types
#3 Java Classes
#4 Reference Types
#5 Loops & Arrays
#6 Methods & Classes
#7 Lists & Maps
#8 File & Network I/O
#9 Swing Interfaces
#10 Swing Actions
#11 Threads
#12 Exceptions
#13 Functional Programming
#14 In-class Written Exam

* Programming Exam – this will be a 3-hour exam

Exam Plan

- Written Exam (1 hour)
  - In-class on Feb 25th (Thursday)
  - Location: BH A51 (Giant Eagle Auditorium)
    - **Please give us 10 minutes to set up**
  - Plan: multiple choice & fill-in the blank, etc.
    - Closed everything. Pencils, erasers and CMU ID
- Programming Exam (3 hours)
  - Date and Time: 5:30pm on Mar 1st (Tuesday)
  - Location: BH A51 (Giant Eagle Auditorium)
  - Plan: same as HW#6, but different
    - Need your laptop. Don't forget your power adapter
    - YOU are not allowed to use any tools to generate GUI

Faculty Course Evaluations

We have been working hard to give feedback to you as soon as possible.

It is your turn.
Please do FCEs!

Outline

- Questions
- Object-Oriented Programming
- Java 8
- Functional Programming
- Lambda Expressions
- Functional Interfaces
- Final Exam

Question for You

- What is Object-Oriented Programming?

Outline

Questions

⇢Object-Oriented Programming

Java 8

Functional Programming

Lambda Expressions

Functional Interfaces

Final Exam

Object-Oriented Programming

- Writing programs using objects
  - Objects that specify their (internal) data
  - Objects that specify the methods by which you can manipulate their data

Object-Oriented Programming

- You have learned:
  - Class (Blueprint or Template)
  - Instance variables and methods
  - Static variables and methods
  - Access Modifiers
  - Inheritance (Super classes and Interfaces)
  - Abstract classes vs. Interfaces
  - Overriding vs. Overloading
  - Encapsulation (private fields and methods that manipulate)
  - Polymorphism & Dynamic Binding
  - Swing & Thread
  - Reflection (java.lang.Class)

Examples

- Shape.java
- Circle.java
- Person.java
- PersonTest.java
- SwingEvent.java
- QuoteGUIAction.java
- HelloThread.java

Time to think differently!

"Pass around behaviors"

Learning Goals

- Understand what Functional Programming is
- Understand core concepts of lambda expressions and functional interfaces in Java 8
  - Examples of functional interfaces
  - Changes in interfaces in Java 8
- Get familiar with the syntax of lambda expressions in Java 8
- Review immutable objects

Outline

Questions

Object-Oriented Programming

⇢Java 8

Functional Programming

Lambda Expressions

Functional Interfaces

Final Exam

Java 8 (focusing on FP)

- Motivations
  - Less verbose (less boilerplate code)
    - Application programmers can write code focusing on core business logic
  - easy-to-write (less to write) and easy-to-read, arguably
  - easy-to-maintain

Outline

Questions

Object-Oriented Programming

Java 8

⇢Functional Programming

Lambda Expressions

Functional Interfaces

Final Exam

Functional Programming

- Think about a problem domain in terms of:
  - Immutable values (immutability)
    - Simply means values or data should never change
  - Functions that translate between those immutable values
    - Should operate only on values or data passed in as arguments
    - And should not rely on other outside values to execute

Calling a function twice with the same input will produce the same result each time (**Immutability is the key**). Think of functions in mathematics

Note: **Java 8 does not enforce this fully. We say Java 8 introduces functional-style programming**

Outline

Questions

Object-Oriented Programming

Java 8

Functional Programming

⇢ Lambda Expressions

Functional Interfaces

Final Exam

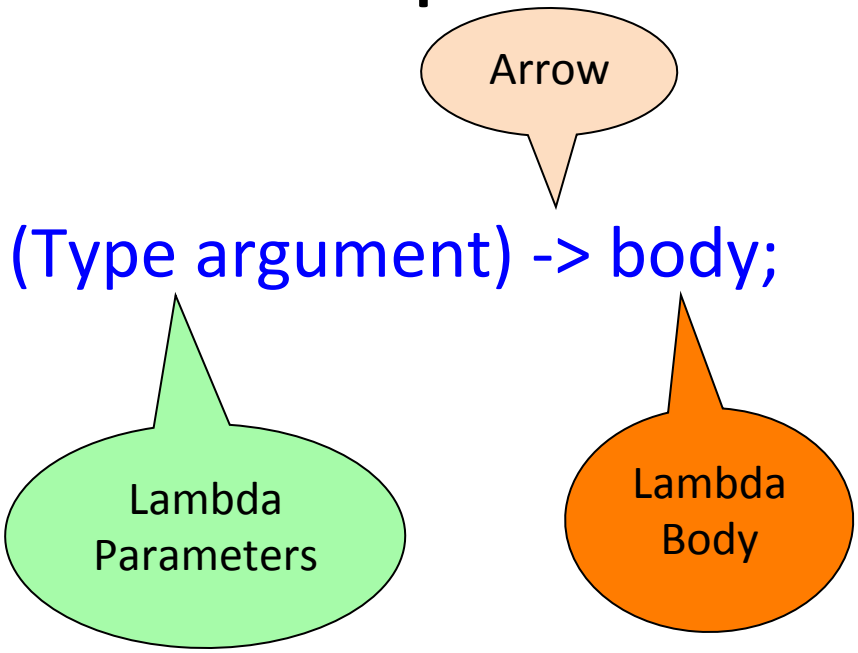
Lambda Expressions

- From the Greek letter lambda (λ)
- Simply speaking,

"*compact way of passing around behavior*"

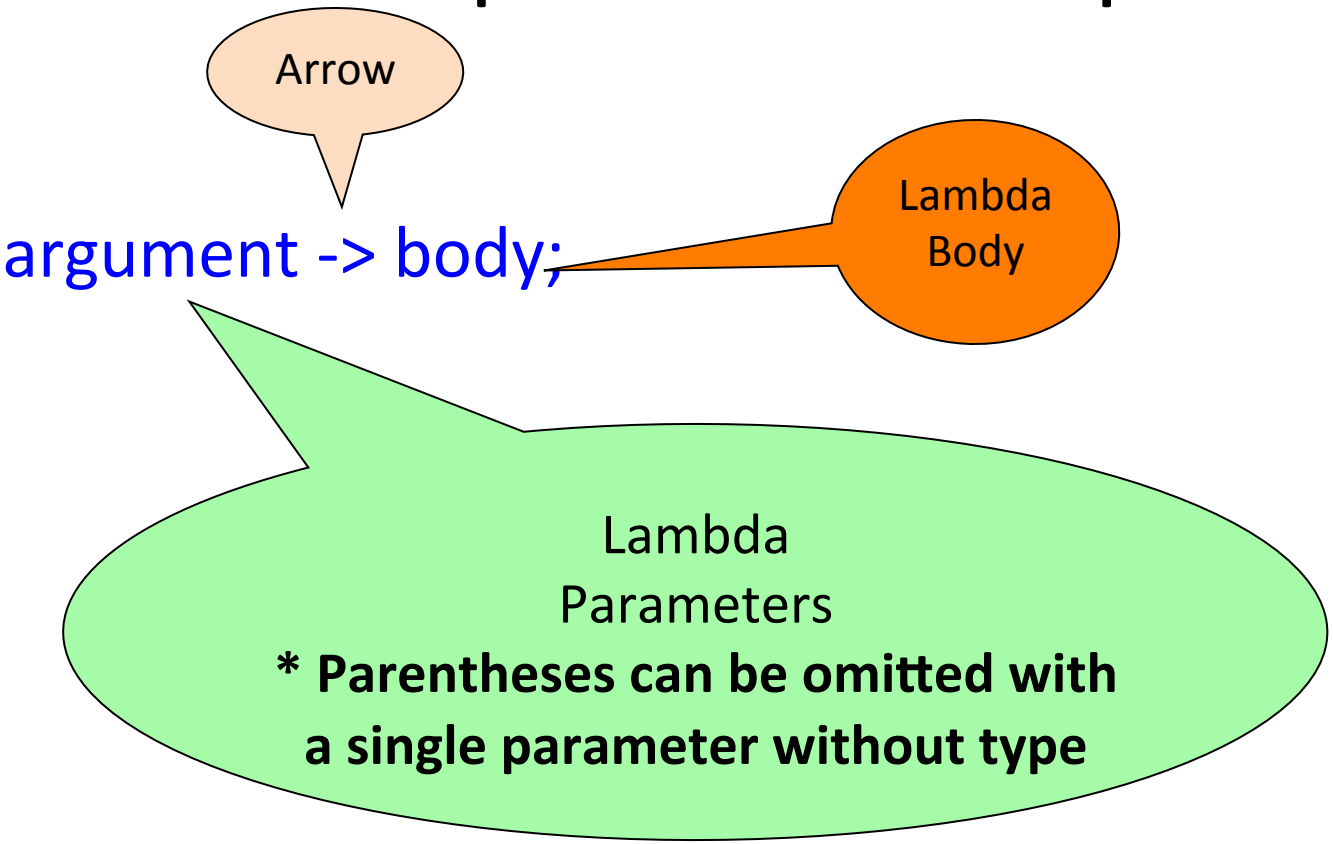
"concise representation of **anonymous function** that can be passed around"

Lambda Expression Examples in Java 8



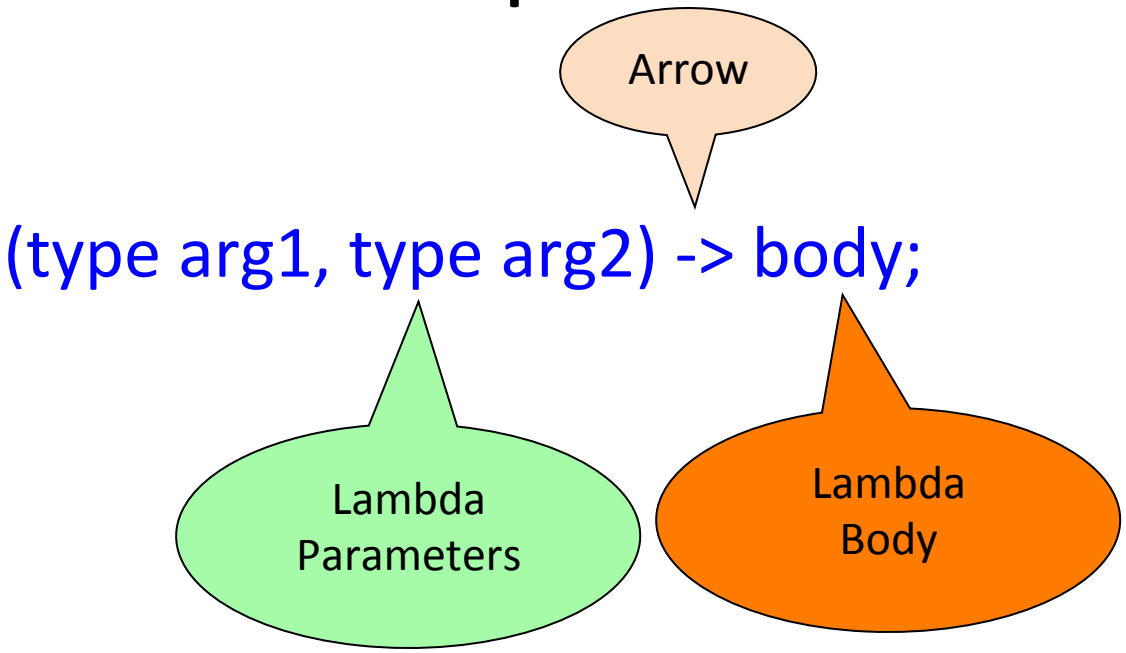
Example: SwingEventFP.java and QuoteGUIActionFP.java

Lambda Expression Examples in Java 8



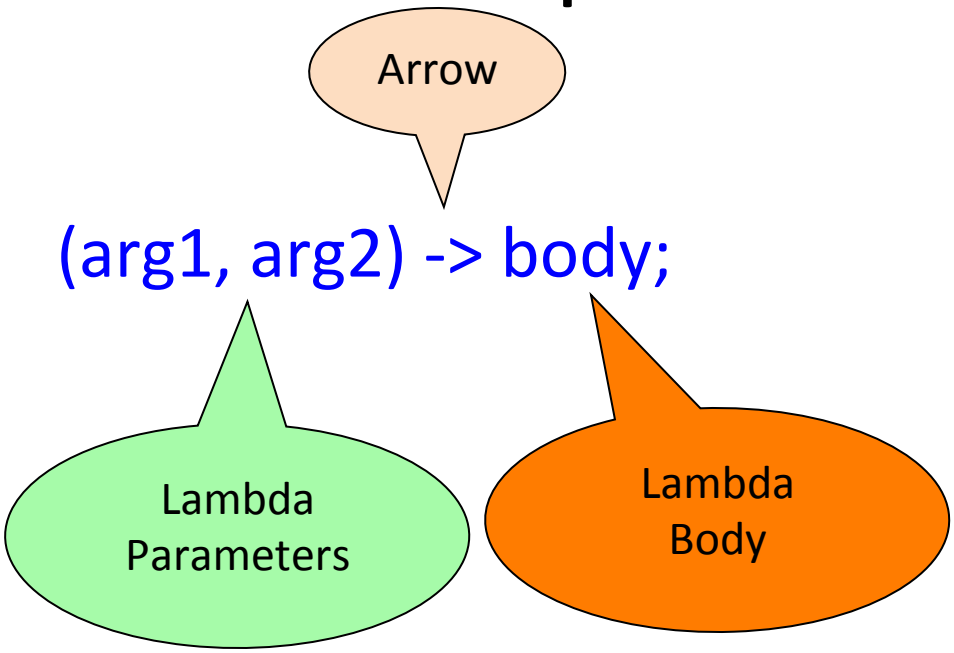
Example: SwingEventFP.java and QuoteGUIActionFP.java

Lambda Expression Examples in Java 8



Example: `ImmutablePerson.java` and PersonTestFP1.java

Lambda Expression Examples in Java 8



Example: ImmutablePerson.java and PersonTestFP2.java

Did it bother you?

In `PersonTest.java`

```
for (ImmutablePerson p : list) {
    System.out.println(p);
}
```

Default methods (new in Java 8)

- Did you notice?

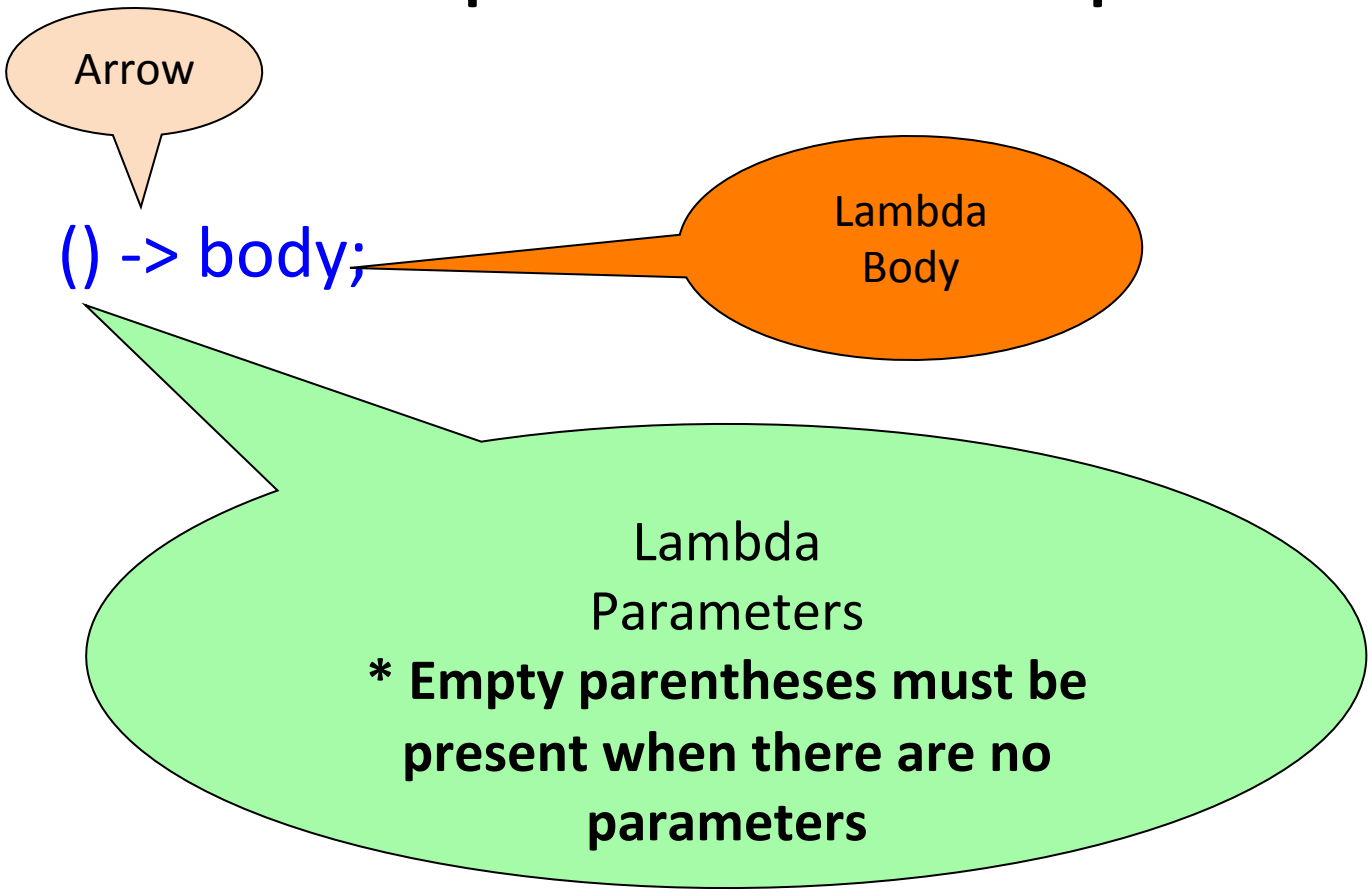
```
public interface Iterable<T> {
    …
    default void forEach(Consumer<? super T> action) {
        Objects.requireNonNull(action);
        for (T t : this) {
            action.accept(t);
        }
    }
    …
}
```

Default methods (new in Java 8)

- Allow **API designers** to enlarge interfaces without breaking existing client code
  - In fact, many interfaces in Java 8 have more methods but they do not break existing code because they are default methods with implementations in them

    (example: Collection interface in java.util package)
- Provide code reuse in interfaces

Lambda Expression Examples in Java 8



Example: HelloThreadFP.java

Lambda Expression Examples in Java 8

Lambda Parameters

Arrow

```
() -> {
        statement;
        statement;
};
```

Lambda Body
***It is possible to have multiple statements**

Example: HelloThreadFP.java

Outline

Questions

Object-Oriented Programming

Java 8

Functional Programming

Lambda Expressions

⇢Functional Interfaces

Final Exam

Functional Interface

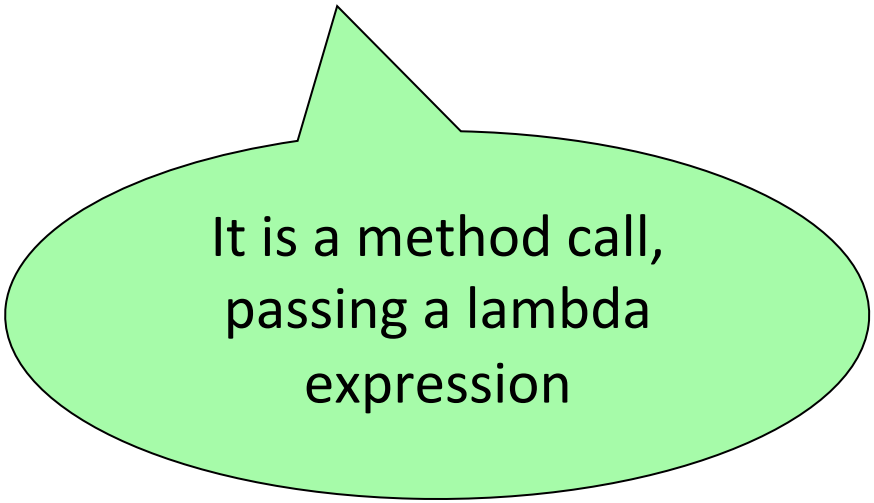
```
private static void doWork(int x) {
        System.out.println(x);
}

doWork(3);
```

Passing 3 and
it is int

In Java, all method parameters must have types

Functional Interface

new Thread(**() -> System.*out.println("Hello from lambda!")***);



In Java, all method parameters must have types!!!

Functional Interface

"**What is the type of the lambda expression?**"

It is an "**interface**" just like other interfaces but different

It is "**functional**" because it **allows you to pass a function (behavior) code**

Thus, **functional interface**

Functional Interface

- Interface with **EXACTLY ONE abstract method**
  - The method is called "Function Descriptor"
- Used as the type of a lambda expression
  - After all, all method parameters have types in Java
  - In other words, **signature of THE abstract method describes THE signature of a lambda expression**
  - **Which means Lambda expressions are STATICALLY TYPED**
- java.util.function package
- @FunctionalInterface annotation
  - Example: Runnable interface

Functional Interface

```
@FunctionalInterface
public interface Runnable {
    public abstract void run();
}
```

No parameters

Only abstract method

No parameters

```
new Thread(() -> { System.out.println("Hello from lambda!"); });
```

Implementation of abstract run method!

* A lot more Functional Interfaces predefined for you

A few Important Functional Interfaces

Interface Name	Arguments	Returns
Consumer<T>	T	void
Supplier<T>	None	T
Function<T, R>	T	R
Predicate<T>	T	boolean

Functional Interface – Consumer<T>

```
@FunctionalInterface
public interface Consumer<T> {
    void accept(T t);
}
```

T ->

Example: ConsumerExampleFP.java

Functional Interface – Supplier<T>

```
@FunctionalInterface
public interface Supplier<T> {
    T get();
}
```

Example: SupplierExampleFP.java

Functional Interface – Function<T, R>

```
@FunctionalInterface
public interface Function<T, R> {
    R apply(T t);
}
```

Example: FunctionExampleFP.java

Functional Interface – Predicate<T>

```
@FunctionalInterface
public interface Predicate<T> {
    boolean test(T t);
}
```

Example: PredicateExampleFP.java

Immutable Values

- Local variables used (referenced) but not declared in lambda expressions should be final or effectively final
  - You can assign to the variable **only once**
  - **Immutable Objects (Again!)**

Example: SwingEventWithValuesFP.java and FinalVariables.java

Immutable Values

- "Pure" functional programming languages such as Haskell and XSLT, it is hard to use iterative control structures such as "while" and "for" because you cannot update counter variables like i, j, k,….
  - Writing FP code, I try not to think of them as variables. Rather, symbols!

- **How to write code then?**

Immutable Values (Objects)

"***Classes should be immutable unless there's a very good reason to make them mutable....If a class cannot be made immutable, limit its mutability as much as possible.***" **by Joshua Bloch**

Immutable objects are harder than you think

For example, String class is only observably immutable!

Moving forward

- This is brief introduction about FP with Java 8
- There are a lot more in Java 8
  - Streams (java.util.stream package)
  - Parallel Data Processing
  - New Date and Time API (java.time package)
  - Optional (alternative to null)
  - Etc.

Moving forward

- J2EE Web Application Development (08-672)
  - Servlets
  - JSPs & JSTL
  - Databases
  - ORM
  - HTML
  - CSS
  - Etc.

Moving forward

- Data Structures for Application Programmers (08-722)
  - Arrays and List (ArrayList & LinkedList)
  - Stack and Queue
  - Hashing, HashTable, HashMap, and HashSet
  - Binary Search Trees, TreeMap, and TreeSet
  - Huffman Coding
  - Heap (PriorityQueue)
  - Searching and Sorting Algorithms
  - Recursion
  - Etc.

Sample Final Exam Questions

- What is functional programming?
- What is functional interfaces?
- What is lambda expressions?
- Would the following lambda expression compile?

```
Runnable r = e -> System.out.println("Hello World");
```

- Given Person class, how would you write a lambda expression to compare person objects by last name in a sort?

Written Exam

- See you on Thursday
- In BH A51 (Giant Eagle Auditorium)
- Be here by 12pm
  - But exam starts at 12:10pm
  - Pencils, Erasers, and CMU ID