UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

Lambda Expressions

CS 272 Software Development

Professor Sophie Engle
Department of Computer Science

Motivation

Avoid Naming Single Use Variables

```java
Map<String, Set<String>> elements = …

Set<String> set = elements.get("hello");
set.add("world");

elements.get("hello").add("world");
```

Avoid Naming Single Use Classes

```java
PathMatcher matcher = new PathMatcher() {
  @Override
  public boolean matches(Path path) {
    return path.toString().endsWith(".txt");
  }
};
```

https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/nio/file/PathMatcher.html

Anonymous Classes

- Allows the **declaration** (i.e. superclass/interface), **definition** (i.e. method implementation), and **instantiation** (i.e. constructor call) of a class
- Always an inner class
- Never an abstract, static, or final* class

https://docs.oracle.com/javase/specs/jls/se17/html/jls-15.html#jls-15.9.5

Anonymous Methods?

- Many interfaces only have one abstract method
  - `PathMatcher`, `Comparator`, `Runnable`, etc.
- Is there shortcut syntax for defining these methods?
  - e.g. array initialization, auto boxing/unboxing, ...
- What does it mean for a method versus a class to be anonymous?

https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/lang/FunctionalInterface.html

Brief History

Lambda Calculus

- Name comes from symbol **Λ λ** (upper/lower lambda)
- Invented in 1930s by Alonzo Church (1903–1995)
- Can simulate any Turing machine
- All functions are **anonymous** functions
- Computational model underlying many **functional programming** languages

https://en.wikipedia.org/wiki/Lambda_calculus

Functional Programming

- Different paradigm vs object-oriented programming
  - Uses **expressions** (returns a value) vs **statements**
  - Eliminates side effects, avoids mutable data
  - Functions may be parameters to other functions
- Produces more concise code and easier to parallelize
- Many languages support functional programming

https://en.wikipedia.org/wiki/Functional_programming

Java Implementation

Functional Interfaces

- An **annotation** applied to interfaces with exactly one abstract method
  - Does not count default methods or overriding `public Object` methods
- Instances created with **lambda expressions**, **method references**, or traditionally (implements keyword, anonymous inner class)

https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/lang/FunctionalInterface.html

Functional Interfaces

```java
package java.nio.file;

@FunctionalInterface
public interface PathMatcher {
    boolean matches(Path path);
}
```

https://github.com/openjdk/jdk/blob/master/src/java.base/share/classes/java/nio/file/PathMatcher.java
https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/lang/FunctionalInterface.html

Package java.util.function

Functional Interface	Description	Method
Function<T,…,R>	Accepts *n* args and produces a result	R apply(T t, …)
Consumer<T,…>	Accepts *n* args and returns no results	void accept(T t, …)
Predicate<T,…>	Accepts *n* args and returns a boolean	boolean test(T t, …)
Supplier<R>	Accepts *no* args and supplies results	R get()

https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/util/function/package-summary.html

Lambda Expressions

- Compact definition of a functional interface
  - *Almost* like a shortcut syntax for anonymous inner classes of interfaces with only one abstract method
- Can be passed to other methods as parameters
- Can be considered anonymous methods (methods without a name)

Lambda Expression Syntax

```
(a, ...) → { statements; ... }
```

- Parameters enclosed in parenthesis `( )` if more than one comma-separated parameter
- The `→` arrow token (a `-` dash and `>` greater than sign)
- The body enclosed in curly `{ }` braces if not a return statement or multiple statements

https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html#syntax

Anonymous Class Example

```java
PathMatcher matcher = new PathMatcher() {
  @Override
  public boolean matches(Path path) {
    return path.toString().endsWith(".txt");
  }
};
```

https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/nio/file/PathMatcher.html

```java
PathMatcher matcher1 = new PathMatcher() {
  @Override
  public boolean matches(Path path) {
    return path.toString().endsWith(".txt");
  }
};

PathMatcher matcher2 = (Path path) -> {
  return path.toString().endsWith(".txt");
};
```

https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/nio/file/PathMatcher.html

Lambda Expression Example

```
PathMatcher m1 = (Path p) -> {
  return p.toString().endsWith(".txt");
};

PathMatcher m2 = p -> p.toString().endsWith(".txt");

Predicate<Path> m3 = p -> p.toString().endsWith(".txt");
```

https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/nio/file/PathMatcher.html

Method References

- Some lambda expressions call an existing method
  - e.g. `s → s.trim()`
- Use method references to use existing methods instead of using a lambda expression
  - e.g. `String::trim`

https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html

Method References

Reference	Pattern	Example
Constructor	ClassName::new	HashSet::new
Static method	ClassName::staticMethod	String::valueOf
Instance (arbitrary)	ClassName::instanceMethod	String::trim
Instance (particular)	instance::instanceMethod	mySet::add

https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html

Functions as Objects?

- **Lambda expressions are NOT objects!**
    - The "type" is a functional interface
    - Does not inherit from `Object`
    - Cannot use the `this`, `super`, or `new` keywords*
- Can only interact with "effectively `final`" variables outside its scope

USF UNIVERSITY OF SAN FRANCISCO

CHANGE THE WORLD FROM HERE

Professor Sophie Engle
sjengle.cs.usfca.edu