CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2015
Doc 7 Java 8 Lambda
Feb 5, 2015

Java 8

Target Release March 18, 2014

New Features

- New Time, Date & Calendar classes
- Improvements to Cryptographic classes
- Nashorn JavaScript Engine
- Concurrency Improvements
  - Accumulators, Adders
- Default Methods
- Functional language features
  - Lambda Expressions
  - Collection Streams (internal iterators)

Documentation

Java lambda Tutorial

http://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

Java 8 Lambdas, Warburton, O'Reilly Media, 2014

http://libproxy.sdsu.edu/login?url=http://proquest.safaribooksonline.com/

Lambda Expression

Anonymous Function

```
(Integer a, Integer b) -> a + b
```

arguments

body

```
(Integer start, Integer stop) -> {
    for (int k = start; k < stop; k++)
        System.out.println(k);
    }
```

Short Version of Lambda Syntax

```
(String text) -> text.length();
```

```
text -> text.length();
```

```
(Integer a, Integer b) -> a + b
```

```
(a, b) -> a + b
```

Using Lambdas

```
Function<String,Integer> length =  text -> text.length();
int nameLength = length.apply("Roger Whitney");
```

```
BiFunction<Integer,Integer,Integer> adder = (a, b) -> a + b;
int sum = adder.apply(1, 2);
```

Other Types of Lambdas

```
Predicate<Integer> isLarge =  value -> value > 100;
if (isLarge.test(59))
      System.out.println("large");
```

```
Consumer<String> print = text -> System.out.println(text);
print.accept("hello World");
```

```
int size = xxx;
      Supplier<List> listType = size > 100 ? (()-> new ArrayList()): (() -> new Vector());
      List elements = listType.get();
      System.out.println(elements.getClass().getName());
```

Lambda Types

New - See java.util.function Interfaces

- Predicate<T> -- a boolean-valued property of an object
- Consumer<T> -- an action to be performed on an object
- Function<T,R> -- a function transforming a T to a R
- Supplier<T> -- provide an instance of a T (such as a factory)
- UnaryOperator<T> -- a function from T to T
- BinaryOperator<T> -- a function from (T, T) to T

Pre-existing

- java.lang.Runnable
- java.util.concurrent.Callable
- java.security.PrivilegedAction
- java.util.Comparator
- java.io.FileFilter
- java.beans.PropertyChangeListener
- etc.

Functional Interfaces

Interface with one method

Can be used to hold a lambda

```
java.lang.Runnable
        void run()
```

Runnnable Example

```
Runnable test = () -> System.out.println("hello from thread");
Thread example = new Thread(test);
example.start();
```

OnClickListener Example

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View source) {
        makeToast();
    }
});
```

```
button.setOnClickListener( event -> makeToast());
```

Internal Iterator - forEach

```
String[] rawData = {"cat", "can", "bat", "rat"};
List<String> data = Arrays.asList(rawData);
data.forEach( word ->System.out.println(word) );
```

Stream

java.util.stream.Stream

Sequence of values

Operations on the values

Operations are chained together into pipelines

Example

```
        String[] words = {"a", "ab", "abc", "abcd", "bat"};
        List<String> wordList = Arrays.asList(words);
        List<String> longWords
    longWords = wordList.stream()
                                        .filter( s -> s.length() >
2)
                                        .filter( s -> s.charAt(0)
== 'a')
                                        .map( s ->
    s.toUpperCase())

    .collect(Collectors.toList());
            System.out.println(longWords);
```

Lazy Evaluation

```
        String[] words = {"a", "ab", "abc", "abcd", "bat"};
        List<String> wordList = Arrays.asList(words);
        List<String> longWords
    longWords = wordList.stream()
                                        .filter( s -> s.length() >
2)
                                        .filter( s -> s.charAt(0)
== 'a')
                                        .map( s ->
s.toUpperCase())

.collect(Collectors.toList());
        System.out.println(longWords);
```

Only One pass of List to do all operations

4.0 gpa

```
List<Student>  = students.stream()
                                    .filter( student -> student.gpa() >= 4.0)
                                        .collect(Collectors.toList());
```

Steam methods

count()
distinct
filter
findAny
findFirst
flatMap
forEach
forEachOrdered
limit
map
max
min
nonMatch
reduce
sorted

For More Information

State of the Lambda: Libraries Edition

http://cr.openjdk.java.net/~briangoetz/lambda/lambda-libraries-final.html

http://tinyurl.com/mshjfkj

State of the Lambda

http://cr.openjdk.java.net/~briangoetz/lambda/lambda-state-final.html

http://tinyurl.com/kg5m9zu