Lambda Expressions

A Functional Interface

- Define a class in place instead of in a separate file
- Why would you do this?
  - Logically group code in one place
  - Increase encapsulation
  - Make code more readable
- `StringAnalyzer` interface

```
public interface StringAnalyzer {
  public boolean analyze(String target, String
   searchStr);
}
```

- A single method interface
  - **Functional Interface**
- Takes two strings and returns a `boolean`

Polymorphic Usage of StringAnalyzer

- An improvement to the code is to encapsulate the `forloop`:

```
 3 public class Z03Analyzer {
 4
 5   public static void searchArr(String[] strList, String
   searchStr, StringAnalyzer analyzer){
 6     for(String currentStr:strList){
 7       if (analyzer.analyze(currentStr, searchStr)){
 8         System.out.println("Match: " + currentStr);
 9       }
10     }
11   }
// A number of lines omitted
```

String Analysis Test Class with Helper Method

- With the helper method, the main method shrinks to this:

```
13   public static void main(String[] args) {
14     String[] strList01 =
15     {"tomorrow","toto","to","timbukto","the","hello","heat"};
16     String searchStr = "to";
17     System.out.println("Searching for: " + searchStr);
18
19     // Call concrete class that implments StringAnalyzer
20     ContainsAnalyzer contains = new ContainsAnalyzer();
21
22     System.out.println("===Contains===");
23     Z03Analyzer.searchArr(strList01, searchStr, contains);
24   }
```

String Analysis Anonymous Inner Class

- Create anonymous inner class for third argument.

```
19      // Implement anonymous inner class
20      System.out.println("===Contains===");
21      Z04Analyzer.searchArr(strList01, searchStr,
22        new StringAnalyzer(){
23          @Override
24          public boolean analyze(String target, String
   searchStr){
25            return target.contains(searchStr);
26          }
27        });
28    }
```

String Analysis Lambda Expression

- Use lambda expression for the third argument.

```
13   public static void main(String[] args) {
14     String[] strList =
15     {"tomorrow","toto","to","timbukto","the","hello","heat"};
16     String searchStr = "to";
17     System.out.println("Searching for: " + searchStr);
18
19     // Lambda Expression replaces anonymous inner class
20     System.out.println("==Contains==");
21     Z05Analyzer.searchArr(strList, searchStr,
22       (String target, String search) -> target.contains(search));
23   }
```

Lambda Expression Defined

Argument List	Arrow Token	Body
`(int x, int y)`	`->`	`x + y`

Basic Lambda examples

```
(int x, int y) -> x + y
(x, y) -> x + y

(x, y) -> { system.out.println(x + y);}

(String s) -> s.contains("word")
s -> s.contains("word")
```

What Is a Lambda Expression?

```
(t,s) -> t.contains(s)
```

ContainsAnalyzer.java

```
public class ContainsAnalyzer implements StringAnalyzer {
  public boolean analyze(String target, String searchStr){
    return target.contains(searchStr);
  }
}
```

What Is a Lambda Expression?

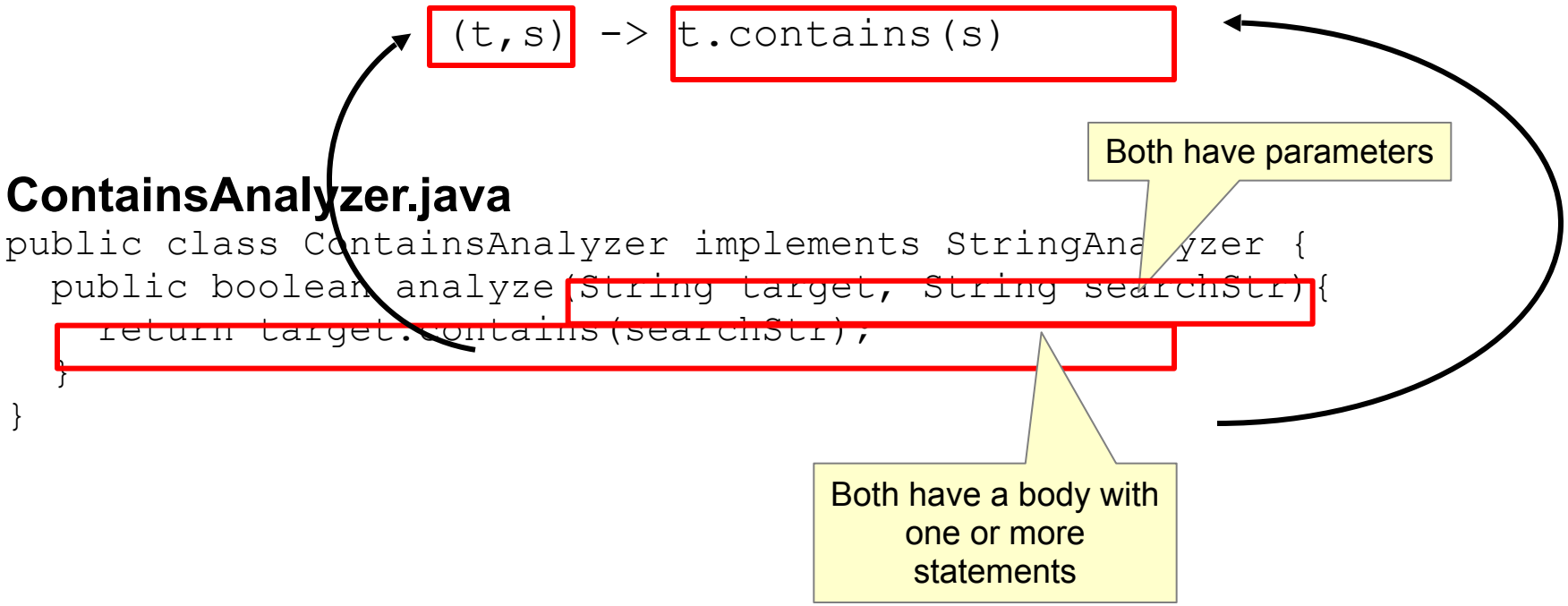
```
(t,s) -> t.contains(s)
```

Both have parameters

ContainsAnalyzer.java

```
public class ContainsAnalyzer implements StringAnalyzer {
  public boolean analyze(String target, String searchStr){
    return target.contains(searchStr);
  }
}
```

What Is a Lambda Expression?



Lambda Expression Shorthand

- Lambda expressions using shortened syntax

```
20      // Use short form Lambda
21      System.out.println("==Contains==");
22      Z06Analyzer.searchArr(strList01, searchStr,
23        (t, s) -> t.contains(s));
24
25      // Changing logic becomes easy
26      System.out.println("==Starts With==");
27      Z06Analyzer.searchArr(strList01, searchStr,
28        (t, s) -> t.startsWith(s));
```

- The `searchArr` method arguments are:

```
public static void searchArr(String[] strList, String
 searchStr, StringAnalyzer analyzer)
```

Lambda Expressions as Variables

- Lambda expressions can be treated like variables.
- They can be assigned, passed around, and reused.

```
19      // Lambda expressions can be treated like variables
20      StringAnalyzer contains = (t, s) -> t.contains(s);
21      StringAnalyzer startsWith = (t, s) -> t.startsWith(s);
22
23      System.out.println("==Contains==");
24      Z07Analyzer.searchArr(strList, searchStr,
25        contains);
26
27      System.out.println("==Starts With==");
28      Z07Analyzer.searchArr(strList, searchStr,
29        startsWith);
```