



JAVA ALGORITHMS

SUMMARY OF UTILITIES

- `java.util.Arrays` – static utilities for raw arrays
 - Searching and sorting
 - Equality comparisons and hash codes
 - Fill
 - Copy
- `java.util.Collections` – similar items for Lists. Also includes:
 - Min, max, counts
 - Reverse, shuffle
- There are many more algorithms and utilities in the java library!
- To find how to use them, go to the Java API!

EXAMPLE OF USING SORT

```
1. Scanner s = new Scanner(new File("numbers.txt"));
2. ArrayList<Integer> numbers = new ArrayList<>();
3. while (s.hasNextInt())
4.     numbers.add(s.nextInt());
5. ...elsewhere...
6. Collections.sort(numbers);
```

ADVANCED ALGORITHMS WITH LAMBDA EXPRESSIONS (JAVA 8)

- In `java.util.Collection` provides a function `stream()`. A `stream()` allows you to perform functions over the data in the collection. Examples:
 - `filter` – create a stream based on a predicate
 - `forEach` – apply an action to each element
 - `map` – create a new stream after applying an action to each element
 - Many, many more
- You can always use the classic method of having a specialized file implement the required interface.
- OR you can use anonymous classes – nameless classes
- OR you can use a lambda expression
 - A lambda is an anonymous single method class, but defined with extremely terse syntax
 - Can also loosely define them as nameless methods

ADVANCED ALGORITHMS WITH LAMBDA EXPRESSIONS (JAVA 8)

- Take the following example function

```
1. public static void
2.   printIntegersInRange(
3.     List<Integer> nums,
4.     Integer low,
5.     Integer high) {
6.   for(Integer i : nums)
7.     if(i >= low && i <= high)
8.       System.out.println(i);
9. }
```

- We should be able to generalize this. We already know how, use interfaces

```
1. public interface CheckInteger {
2.   boolean test(Integer n);
3. }
```

- Then our function becomes

```
1. public static void
2.   printIntegersIf(
3.     List<Integer> nums,
4.     CheckInteger tester) {
5.   for(Integer i : nums)
6.     if(tester.test(i))
7.       System.out.println(i);
8. }
```

ADVANCED ALGORITHMS WITH LAMBDA EXPRESSIONS (JAVA 8)

- Now with a class

```
1. public class CheckRange0To100
2.     implements CheckInteger {
3.     public static Boolean
4.         test(Integer n) {
5.         return n >= 0 && n <= 100;
6.     }
7. }
```

- However, this seems really extensive for a one off class, right?
- Of course, so Java also has the ability to write things with anonymous classes...

ADVANCED ALGORITHMS WITH LAMBDA EXPRESSIONS (JAVA 8)

- Now with a class

```
1. public class CheckRange0To100
2.     implements CheckInteger {
3.     public static Boolean
4.         test(Integer n) {
5.         return n >= 0 && n <= 100;
6.     }
7. }
```

```
1. printIntegersIf(nums,
2.     new CheckRange0To100());
```

- However, this seems really extensive for a one off class, right?
- Of course, so Java also has the ability to write things with anonymous classes...

ADVANCED ALGORITHMS WITH LAMBDA EXPRESSIONS (JAVA 8)

- Now with an anonymous class

```
1. printIntegersIf(nums,  
2.     new CheckInteger() {  
3.         public boolean  
4.             test(Integer i) {  
5.                 return i >= 0 && i <= 100;  
6.             }  
7.         }  
8.     );
```

- However, this still seems really extensive for a one off class, right?
- Of course, so Java 8 introduced the widely known concept of lambda functions

ADVANCED ALGORITHMS WITH LAMBDA EXPRESSIONS (JAVA 8)

- Now with a lambda expression

```
1. printIntegersIf(nums,  
2.     (Integer i) -> i >= 0 && i <= 100  
3. );
```

- Short and sweet!
- This allows us to write generic functions with functions as parameters easily!

ADVANCED ALGORITHMS WITH LAMBDA EXPRESSIONS (JAVA 8)

- Now with the standard Java provided functionals found in the package `java.util.function`

```
1. public static void
2.   printIntegersInRange(
3.     List<Integer> nums,
4.     Predicate<Integer> tester) {
5.   for(Integer i : nums)
6.     if(tester.test(i))
7.       System.out.println(i);
8. }
```

- And our lambda can become even shorter!

```
1. printIntegersIf(nums,
2.   i -> i >= 0 && i <= 100
3. );
```



- Sort example

- `Collections.sort(nums, (i1, i2) -> -i1.compareTo(i2));`

- [Full tutorial](#)



PROBLEM

- Generate a random list of 1000 integers between 0 and 100
 - Filter the list to numbers between 40 and 60
 - Map a function on each element that applies a random power between 2 and 4 to each element of the list
 - Sum up and average the list
 - Do this all without loops, only use lambdas and functions of stream!
 - With the rest of the time, work on the next programming assignment
- 
- 
- 