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
- `java.util.streams.*` - automatic data processing library (with parallelism included)
- 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

```
Scanner s = new Scanner(new File(“numbers.txt”));
ArrayList<Integer> numbers = new ArrayList<>();
while(s.hasNextInt())
   numbers.add(s.nextInt());
…elsewhere…
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

```java
public static void
  printIntegersInRange(
    List<Integer> nums,
    Integer low,
    Integer high) {
  for(Integer i : nums)
    if(i >= low && i <= high)
      System.out.println(i);
}
```

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

```java
public interface CheckInteger {
  boolean test(Integer n);
}
```

- Then our function becomes

```java
public static void
  printIntegersIf(
    List<Integer> nums,
    CheckInteger tester) {
  for(Integer i : nums)
    if(tester.test(i))
      System.out.println(i);
}
```

ADVANCED ALGORITHMS WITH LAMBDA EXPRESSIONS (JAVA 8)

- Now with a class

```
public class CheckRange0To100
  implements CheckInteger {
  public static Boolean
    test(Integer n) {
    return n >= 0 && n <=100;
  }
}
```

```
printIntegersIf(nums,
  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

```java
printIntegersIf(nums,
    new CheckInteger() {
      public boolean
        test(Integer i) {
        return i >= 0 && i <= 100;
      }
    }
);
```

- 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

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

- Short and sweet!
- This allows us to write generic algorithms 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`

```
public static void
  printIntegersInRange(
    List<Integer> nums,
    Predicate<Integer> tester) {
  for(Integer i : nums)
    if(tester.test(i))
      System.out.println(i);
}
```

- And our lambda can become even shorter!

```
printIntegersIf(nums,
    i -> i >= 0 && i <= 100
);
```

- Sort example

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

PROBLEM

- Lets explore the power of streams and lambdas by bulk processing and "corrupting" some data! (obviously can be used to "clean" data as well)
- Download the source code from online. Change the TODO statements to perform the required actions to bulk process data. You can only use lambda expressions in this file (except when you read from the file)
- Work in pairs. After completing, show me the source file and then work on the next programming assignment.