

Java Idioms

Jeanne Boyarsky
Wednesday February 19, 2020
DevNexus

<https://speakerdeck.com/boyarsky>

Pause for a Commercial

Amazon Best Sellers
Our most popular products based on sales. Updated hourly.

Best Sellers in Oracle Certification

#1 OCP Oracle Certified Professional Java SE 11 Programmer I STUDY GUIDE EXAM 1Z0-815

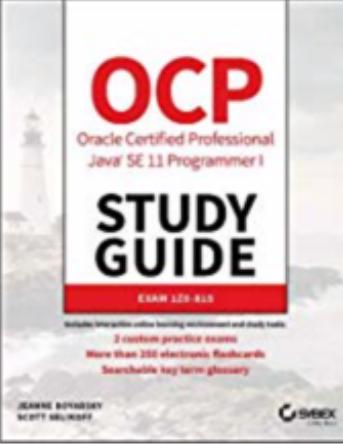
#2 OCA Oracle Certified Associate Java SE 8 Programmer I STUDY GUIDE

#3 OCP Oracle Certified Professional Java SE 11 Programmer I STUDY GUIDE EXAM 1Z0-815

#4 OCP Oracle Certified Professional Java SE 8 Programmer II STUDY GUIDE EXAM 1Z0-816

#5 OCA/OCP JAVA SE 8 PROGRAMMER CERTIFICATION KIT

#1 New Release !



Java 11 certs

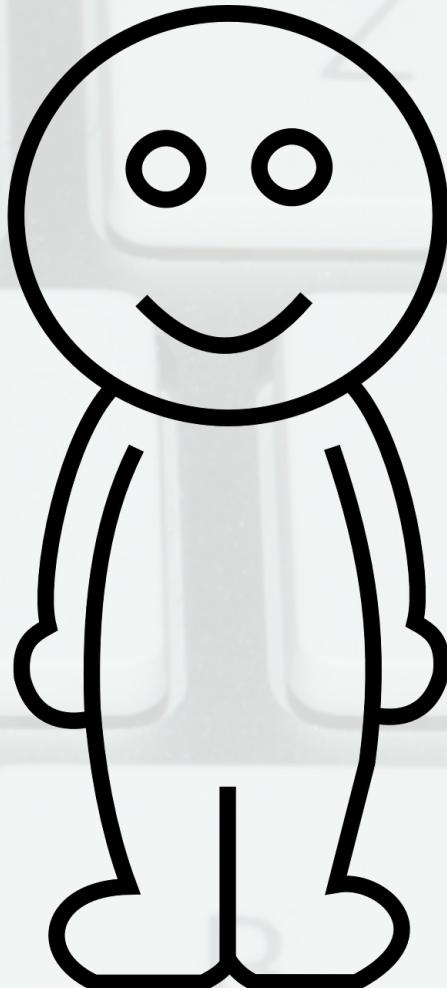
- 1Z0-815 - Out now
- 1Z0-816 - April ETA

@jeanneboyarsky

Disclaimer

- A bit of the material is from my books.
- Some of the material in this presentation may appear in our upcoming certification books.

Introductions



@jeanneboyarsky

Agenda

- Strings
- Regular Expressions
- Collections
- Streams
- Advanced Streams
- File I/O
- Other Useful APIs

Module Flow

- Review lab from previous module
- Lecture/review
- Hands on exercises
- 10 minute break

This means if a colleague needs to call you, the last 15-20 minutes of each hour is best.

Idiom

- Recurring construct
- Has meaning
- Common approach



Design Patterns

- Typically higher level
- May be language agnostic



Anti-Pattern

- Common “solution”
- Negative effects



Example #1 - What is it?

```
public class OnlyOne {  
  
    private static OnlyOne instance;  
  
    private OnlyOne(){}
  
  
    public static synchronized OnlyOne getInstance(){  
        if(instance == null){  
            instance = new OnlyOne();  
        }  
        return instance;  
    }
}
```

Design Pattern
(Singleton)

Example #2 - What is it?

```
public int countPositive(int[] nums) {  
    int count = 0;  
    for (int i = 0; i < nums.length; i++) {  
        if (nums[i] > 0) {  
            count++;  
        }  
    }  
    return count;  
}
```

Idiom
(Pre-Java 8)

Example #3 - What is it?

```
public void doEverything() {  
    try {  
        doWork();  
    } catch(Exception e) {  
        // ignore  
    }  
}
```

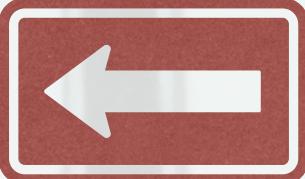
Anti-pattern

- * catches “Exception”
- * ignores exception

Example #4 - What is it?

```
public int countPositive(int[] nums) {  
    int count = 0;  
    int index = 0;  
    while(index < nums.length > index++) {  
        if (nums[index] > 0)  
            count++;  
        index++;  
    }  
    return count;  
}
```

Agenda

- Strings 
- Regular Expressions
- Collections
- Streams
- Advanced Streams
- File I/O
- Other Useful APIs

Is there a match?

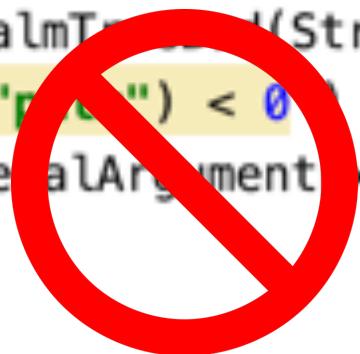
```
public void validatePalmTree(String tree) {  
    if (tree.indexOf("palm") < 0) {  
        throw new IllegalArgumentException("Not a palm tree");  
    }  
}
```



IntelliJ warning!

Is there a match?

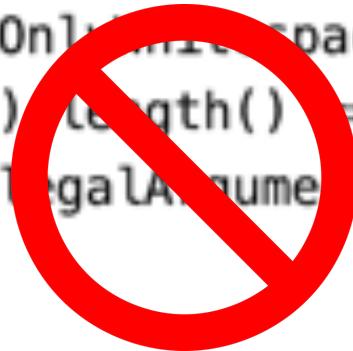
```
public void validatePalmTree(String tree) {  
    if (tree.indexOf("pam") < 0) {  
        throw new IllegalArgumentException("Not a palm tree");  
    }  
}
```



```
private void validatePalmTree(String tree) {  
    if (! tree.contains("pam")) {  
        throw new IllegalArgumentException("Not a palm tree");  
    }  
}
```

Only whitespace

```
public void validateOnlySpaceBad(String string) {  
    if (string.trim().length() == 0) {  
        throw new IllegalArgumentException("Not blank");  
    }  
}
```



```
public void validateOnlyWhitespaceBetter(String string) {  
    if (string.trim().isEmpty()) {  
        throw new IllegalArgumentException("Not blank");  
    }  
}
```

Even Better

```
public void validateOnlyWhitespaceBitBetter(String string) {  
    if (string.strip().isEmpty()) {  
        throw new IllegalArgumentException("Not blank");  
    }  
}
```

Java 11

```
public void validateOnlyWhitespaceBest(String string) {  
    if (string.isBlank()) {  
        throw new IllegalArgumentException("Not blank");  
    }  
}
```

Java 11

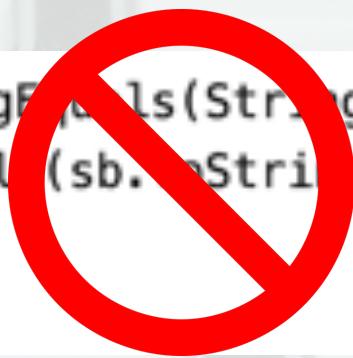
String Concatenation

```
public String builder() {  
    StringBuilder builder = new StringBuilder();  
    for (int i = 0; i < 10; i++) {  
        builder.append(i);  
        builder.append(" ");  
    }  
    return builder.toString();  
}  
  
public String concat() {  
    String result = "";  
    for (int i = 0; i < 10; i++) {  
        result += i + " ";  
    }  
    return result;  
}
```

Best choice
changes over time

Equal?

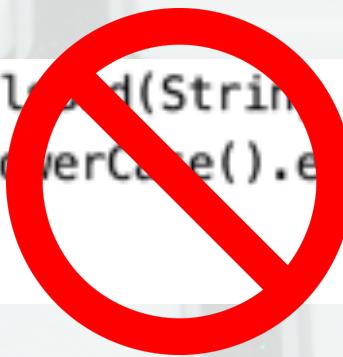
```
public boolean usingEquals(String str, StringBuilder sb) {  
    return str.equals(sb.toString());  
}
```



```
public boolean better(String str, StringBuilder sb) {  
    return str.contentEquals(sb);  
}
```

Case Insensitive Equal

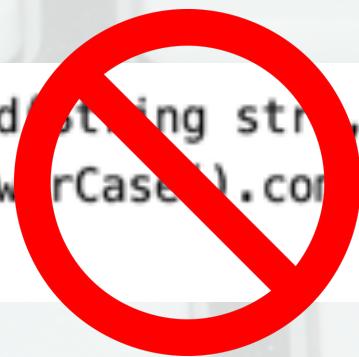
```
public boolean equalsBad(String str1, String str2) {  
    return str1.toLowerCase().equals(str2.toLowerCase());  
}
```



```
public boolean equalsBetter(String str1, String str2) {  
    return str1.equalsIgnoreCase(str2);  
}
```

Case Insensitive Compare

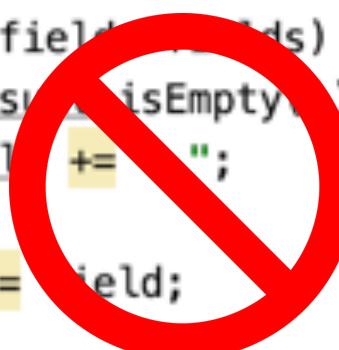
```
public int compareBad(String str1, String str2) {  
    return str1.toLowerCase().compareTo(str2.toLowerCase());  
}
```



```
public int compareBetter(String str1, String str2) {  
    return str1.compareToIgnoreCase(str2);  
}
```

Creating Csv

```
public String commaSeparatedBad(List<String> fields) {  
    String result = "";  
    for (String field : fields) {  
        if (! result.isEmpty()) {  
            result += ",";  
        }  
        result += field;  
    }  
    return result;  
}
```



Better to use
Commons
CSV Library

```
public String commaSeparated(List<String> fields) {  
    return String.join(", ", fields);  
}
```

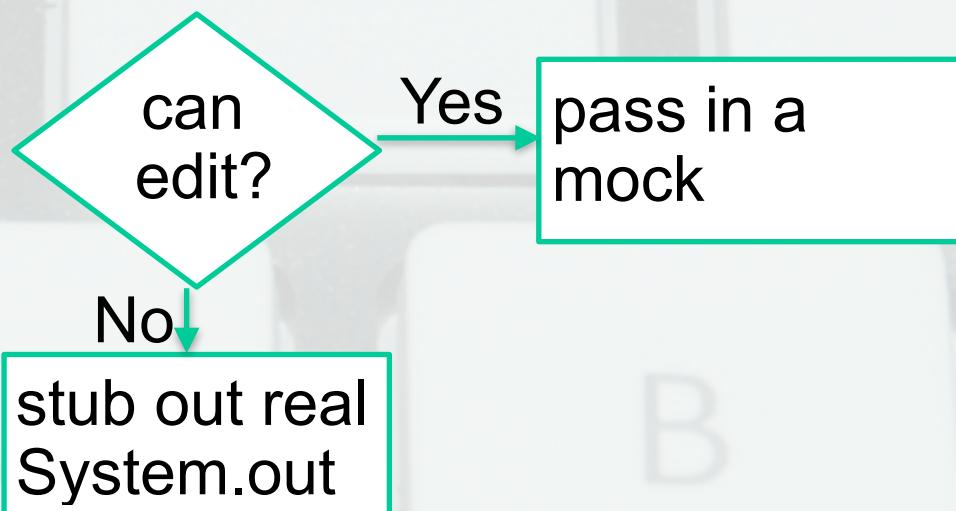
Multiples

```
public String tenStarsBad() {  
    StringBuilder builder = new StringBuilder();  
    for (int i = 0; i < 10; i++) {  
        builder.append("*");  
    }  
    return builder.toString();  
}
```

```
public String tenStars() {  
    return "*".repeat(10);  
}
```

How do I unit test this?

```
public class Printing {  
  
    public void helloAtlanta() {  
        System.out.print("Hello Atlanta");  
    }  
}
```



Pass Mock object

```
public class PrintingWithMock {  
  
    private PrintStream stream;  
  
    public PrintingWithMock() {  
        this(System.out);  
    }  
  
    PrintingWithMock(PrintStream stream) {  
        this.stream = stream;  
    }  
  
    public void helloAtlanta() {  
        stream.print("Hello Atlanta");  
    }  
}
```

And Test

```
class PrintingWithMockTest {  
  
    private PrintingWithMock target;  
    private ByteArrayOutputStream mock;  
  
    @BeforeEach  
    void setUp() {  
        mock = new ByteArrayOutputStream();  
        target = new PrintingWithMock(new PrintStream((mock)));  
    }  
  
    @Test  
    void print() {  
        target.helloAtlanta();  
        String actual = mock.toString();  
        assertEquals( expected: "Hello Atlanta", actual);  
    }  
}
```

If Cannot Change

```
private Printing target;
private ByteArrayOutputStream mock;
private PrintStream originalSystemOut;

@BeforeEach
void setUp() {
    mock = new ByteArrayOutputStream();
    target = new Printing();
    originalSystemOut = System.out;
    System.setOut(new PrintStream(mock));
}

@AfterEach
void tearDown() {
    System.setOut(originalSystemOut);
}
```

Formatting

```
public void printPi() {  
    double pi = 3.14159265359;  
    System.out.format("%s is %3.2f to the nearest %d digits",  
                      "Pi", pi, 2);  
}
```

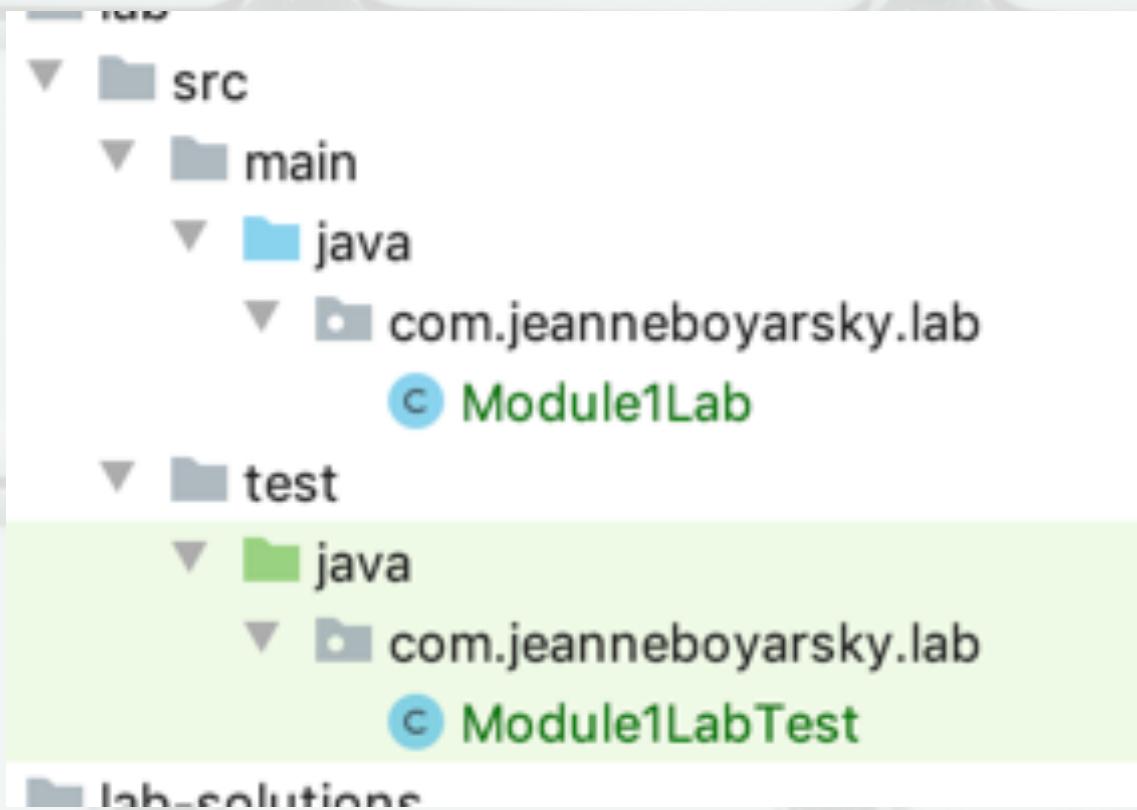
- %s - string
- %d - int
- %f - float/double (minimum width/precision)
- %n - new line

Review - What best API?

- Check if String empty?
- Duplicate String?
- Merge Strings?
- Compare case sensitive strings?
- Compare non-case sensitive strings?

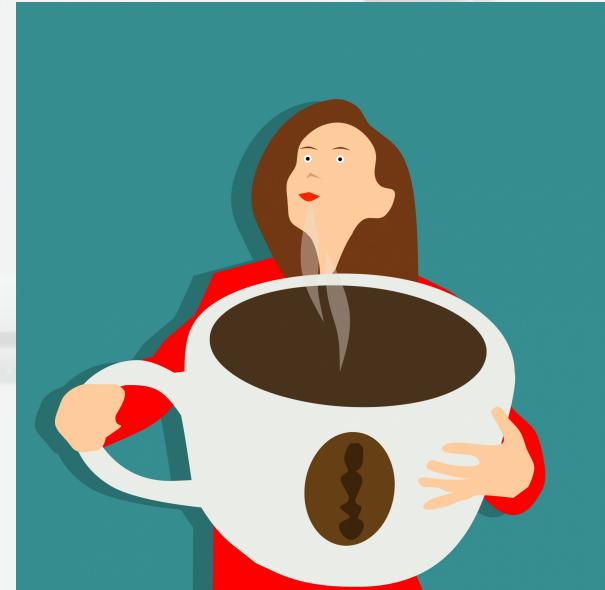
Lab

Do **NOT** use streams or lambdas
in this lab



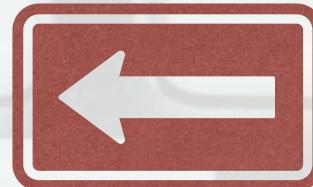
Lab & Break

<https://github.com/boyarsky/2020-devnexus-java-idioms>



Agenda

- Strings
- Regular Expressions
- Collections
- Streams
- Advanced Streams
- File I/O
- Other Useful APIs



Lab Review

- Any questions?
- How did you solve Tic Tac Toe?

Remove all Numbers

```
public static String removeAll(String text) {  
    return text.replaceAll( regex: "[0-9]", replacement: "" );  
}
```

Note replaceAll

What are two ways to remove
blanks spaces too?

Remove leading pattern

```
public static String removeLeadingNumbers(String text) {  
    return text.replaceFirst( regex: "^[0-9]+", replacement: "" );  
}
```

Three changes:
replaceFirst()

^

+

Remove trailing pattern

```
public static String removeTrailingPeriod(String text) {  
    return text.replaceFirst( regex: "\\\.$", replacement: "");  
}
```

Watch escaping

Remove middle

```
public static String removeMiddle(String csv) {  
    String anyCharsExceptComma = "[^,]*";  
    String captureCharsWithoutComma = '(' + anyCharsExceptComma + ')';  
    char comma = ',';  
    String regex = captureCharsWithoutComma + comma  
        + anyCharsExceptComma + comma + captureCharsWithoutComma;  
    return csv.replaceFirst(regex, replacement: "$1,$2");  
}
```

Off by one error?

Ugly regex

```
public static String removeMiddle(String csv) {  
    return csv.replaceFirst( regex: "([^\n]*,[^\n]*,([^\n]*)",  
                           replacement: "$1,$2");  
}
```

Better regex

```
public static String removeMiddle(String csv) {  
    String anyCharsExceptComma = "[^,]*";  
    String captureCharsWithoutComma = '(' + anyCharsExceptComma + ')';  
    char comma = ',';  
    String regex = captureCharsWithoutComma + comma  
        + anyCharsExceptComma + comma + captureCharsWithoutComma;  
    return csv.replaceFirst(regex, replacement: "$1,$2");  
}
```

Regex quick reference

0 or more	*
1 or more	+
0 or 1	?
any	.
three x's	x{3}

range	[0-9]
Not in range	[^0-9]
Digit	\d
White space	\s
Word	\w

Splitting on a Regex

```
public static String[] nameParts(String text) {  
    return text.split( regex: "[ ,]+");  
}
```

Contains only digits

```
public static boolean isOnlyDigits(String text) {  
    return text.matches( regex: "\\\d+");  
}
```

Contains any digits

```
public static boolean atLeastOneDigit(String text) {  
    return text.matches( regex: ".*\d.*");  
}
```

More to regex

Case Insensitive

```
public static String replaceShort(String text) {  
    return text.replaceAll( regex: "[xX]{3}", replacement: "" );  
}
```

Case Insensitive

```
public static String replaceLong(String text) {  
    return text.replaceAll( regex: "(?i)www\\\\.google\\\\.com", replacement: "" );  
}
```

Multi Line Replace

```
public static String remove(String text) {  
    return text.replaceAll( regex: "(?m)^\\s(.*)$",
                           replacement: "$1");  
}
```

(?m)
Also \$1

a
b
c

Dot All

```
public static String remove(String text) {  
    return text.replaceAll( regex: "(?s)<form>.*</form>",  
                           replacement: "<form></form>" );  
}
```

```
<form>  
  <input />  
  <textarea />  
</form>
```

(?s)

Pattern Idiom

```
public static Set<String> getTags(String text) {  
    Set<String> tags = new TreeSet<>();  
    String openTag = "<";  
    String closeTag = ">";  
    String tagContents = "[^>]+";  
    String regex = openTag + "(" + tagContents + ")" + closeTag;  
    Pattern pattern = Pattern.compile(regex);  
    Matcher matcher = pattern.matcher(text);  
    while (matcher.find()) {  
        String tagName = matcher.group(1);  
        tags.add(tagName);  
    }  
    return tags;  
}
```

Pattern Replace

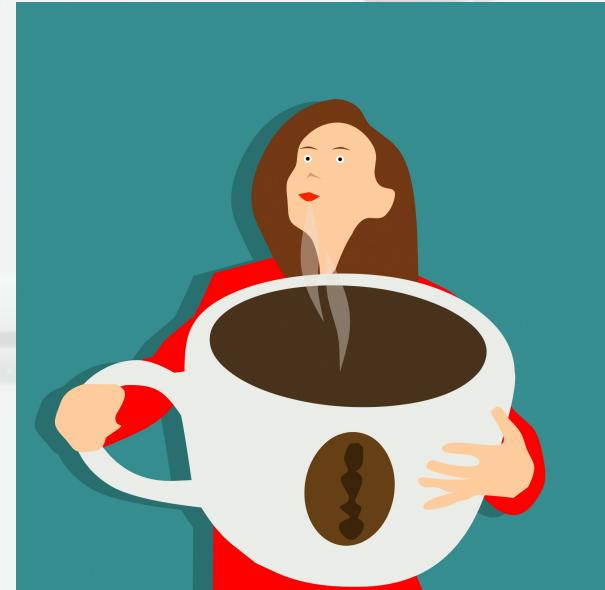
```
public static String sanitize(String text) {  
    String number = "[0-9]{1,3}";  
    String dot = "\\. ";  
    String regex = number + dot + number + dot + number + dot + number;  
    Pattern pattern = Pattern.compile(regex);  
    Matcher matcher = pattern.matcher(text);  
    StringBuilder builder = new StringBuilder();  
    while (matcher.find()) {  
        String ip = matcher.group();  
        matcher.appendReplacement(builder, replacement: "xxx");  
    }  
    matcher.appendTail(builder);  
    return builder.toString();  
}
```

Review - How do I?

- Match digit?
- Remove a single match?
- Treat dot as any character?
- Match a character 100 times?
- Loop through matches?
- Match a period?

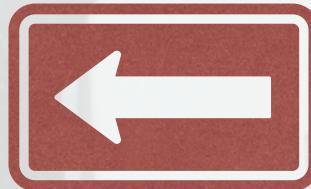
Lab & Break

<https://github.com/boyarsky/2020-devnexus-java-idioms>



Agenda

- Strings
- Regular Expressions
- Collections
- Streams
- Advanced Streams
- File I/O
- Other Useful APIs



Lab Review

- Any questions?
- Did you like replaceAll() or Pattern better?
- What was hardest?

Create a Set

```
public Set<Integer> createSet() {  
    return new HashSet<>(Arrays.asList(1, 2, 3));  
}
```

```
public Set<Integer> createSet() {  
    return Set.of(1, 2, 3);  
}
```

Java 9

Create a List

```
public List<Integer> createList() {  
    return Arrays.asList(1, 2, 3);  
}
```

```
public List<Integer> createList() {  
    return List.of(1, 2, 3);  
}
```

Java 9

Mutable List

```
public List<Integer> createMutableList() {  
    return new ArrayList<>(Arrays.asList(1, 2, 3));  
}
```

```
public List<Integer> createMutableList() {  
    return new ArrayList<>(List.of(1, 2, 3));  
}
```

Java 9

Mutable Copy

```
public <T> List<T> constructor(List<T> original) {  
    return new ArrayList<T>(original);  
}
```

Immutable Copy

```
public <T> List<T> copyOf(List<T> original) {  
    return List.copyOf(original);  
}
```

Java 10

Synchronized View

```
public <T> List<T> synchronizedList(List<T> original) {  
    return Collections.synchronizedList(original);  
}
```

Unmodifiable View

```
public <T> List<T> unmodifiableList(List<T> original) {  
    return Collections.unmodifiableList(original);  
}
```

Create a Map

```
public Map<String, Integer> createMap() {  
    Map<String, Integer> map = new HashMap<>();  
    map.put("a", 1);  
    map.put("b", 2);  
    return map;  
}
```

Create a Map

```
public Map<String, Integer> createMap() {  
    return Map.of( k1: "a", v1: 1,  
                  k2: "b", v2: 2);  
}
```

```
public Map<String, Integer> createMapClearer() {  
    return Map.ofEntries(  
        Map.entry( k: "a", v: 1),  
        Map.entry( k: "b", v: 2));  
}
```

Java 9

Sorting

```
public void sortAlphabetically(List<String> list) {  
    Collections.sort(list);  
    list.sort(Comparator.naturalOrder());  
}
```

```
public void sortBackwards(List<String> list) {  
    Collections.sort(list);  
    list.sort(Comparator.reverseOrder());  
}
```

Sorting

```
public void sortByLength(List<String> list) {  
    list.sort(Comparator.comparing(String::length));  
}
```

```
public void sortByLengthReversed(List<String> list) {  
    list.sort(Comparator.comparing(String::length).reversed());  
}
```

Removing from a List

```
public void remove(List<String>list, String target) {  
    list.remove(target);  
}
```

```
public void removeWithoutEquals(List<Name>list, String target) {  
    list.removeIf(n -> target.equals(n.getFirstName()));  
}
```

Element in List

```
public boolean findMatch(List<String>list, String target) {  
    return list.contains(target);  
}
```

```
public boolean findMatchWithoutEquals(List<Name>list, String target) {  
    return list.stream()  
        .anyMatch(n -> target.equals(n.getFirstName()));  
}
```

Replacing

```
public void doubleString(List<String> list) {  
    list.replaceAll(s -> s + s);  
}
```

Venn Diagram?

```
public Set<Integer> union(Set<Integer> set, Collection<Integer> nums) {  
    Set<Integer> copy = new HashSet<>(nums);  
    copy.addAll(set);  
    return copy;  
}
```

```
public Set<Integer> intersection(Set<Integer> set, Collection<Integer> nums) {  
    Set<Integer> copy = new HashSet<>(nums);  
    copy.retainAll(set);  
    return copy;  
}
```

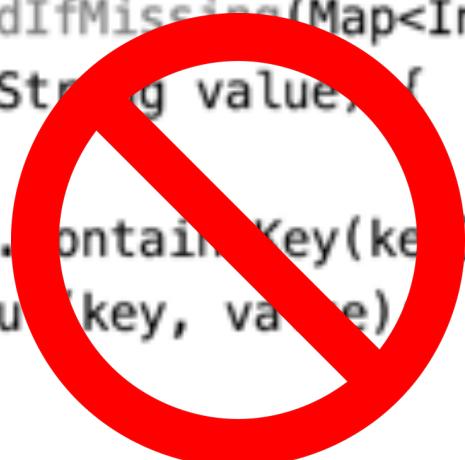
```
public Set<Integer> distinct(Set<Integer> set, Collection<Integer> nums) {  
    Set<Integer> copy = new HashSet<>(set);  
    copy.removeAll(nums);  
    return copy;  
}
```

Defaulting

```
public int total(Map<String, Integer> map,  
                 String key1, String key2) {  
  
    return map.getOrDefault(key1, defaultValue: 0)  
           + map.getOrDefault(key2, defaultValue: 0);  
}
```

Add to map if not present

```
public void addIfMissing(Map<Integer, String> map,  
    int key, String value) {  
  
    if (! map.containsKey(key)) {  
        map.put(key, value);  
    }  
}
```



```
public void addIfMissing(Map<Integer, String> map,  
    int key, String value) {  
  
    map.putIfAbsent(key, value);  
}
```

Other Useful Map Methods

- containsKey(), containsValue()
- entrySet()
- keySet(), values()
- remove(key)
- remove(key, value)
- replace(key, value)
- replace(key, oldValue, newValue)

Printing

```
public void printTwice(Set<Character> set) {  
    set.forEach(c -> System.out.println(c));  
    set.forEach(System.out::println);  
}
```

```
public void print(Map<Character, Double> map) {  
    map.entrySet().forEach(System.out::println);  
    map.forEach((k,v) -> System.out.println(k + v));  
}
```

Review - How do I?

- Create an immutable copy
- Remove all values less than 5
- Update the value in a map if the current value is 10
- Return 0 if the value isn't found
- Find the intersection of two collections

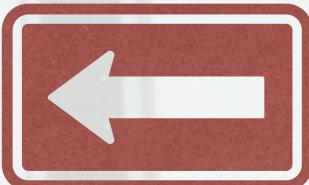
Lab & Break

<https://github.com/boyarsky/2020-devnexus-java-idioms>



Agenda

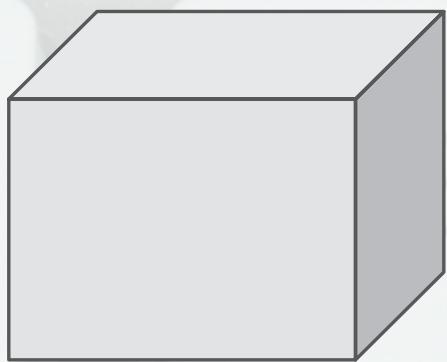
- Strings
- Regular Expressions
- Collections
- Streams
- Advanced Streams
- File I/O
- Other Useful APIs



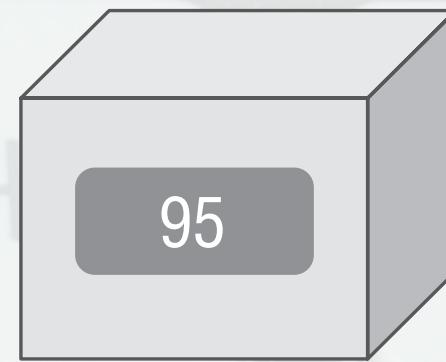
Lab Review

- Did you get them all?
- Which was your longest solution?
- How many ways did you get the challenge question?

Optional

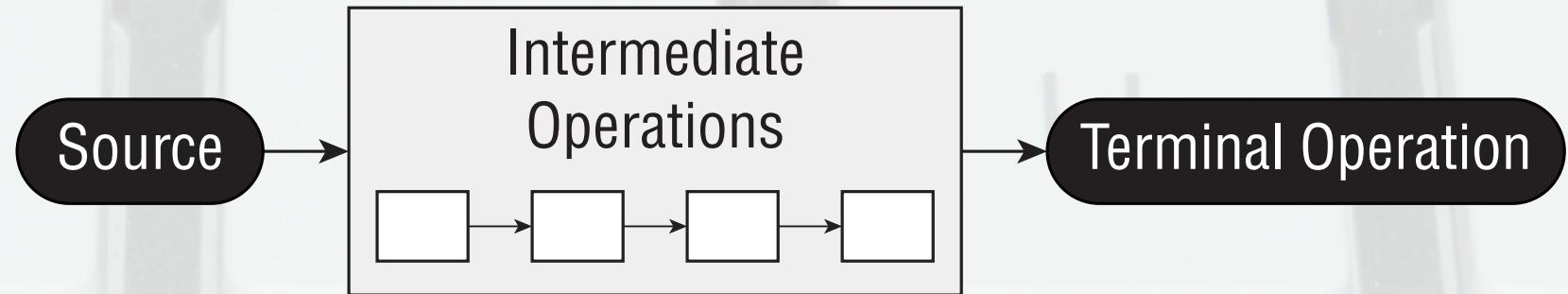


Optional.empty()

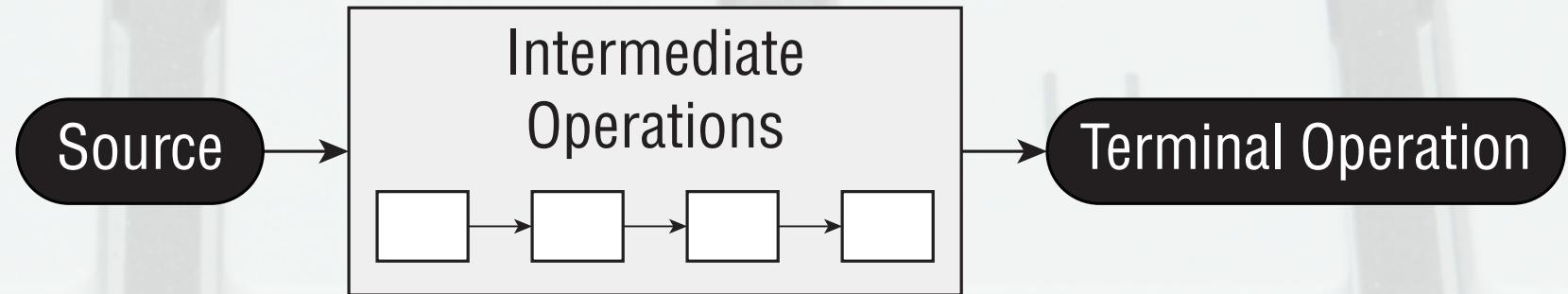


Optional.of(95)

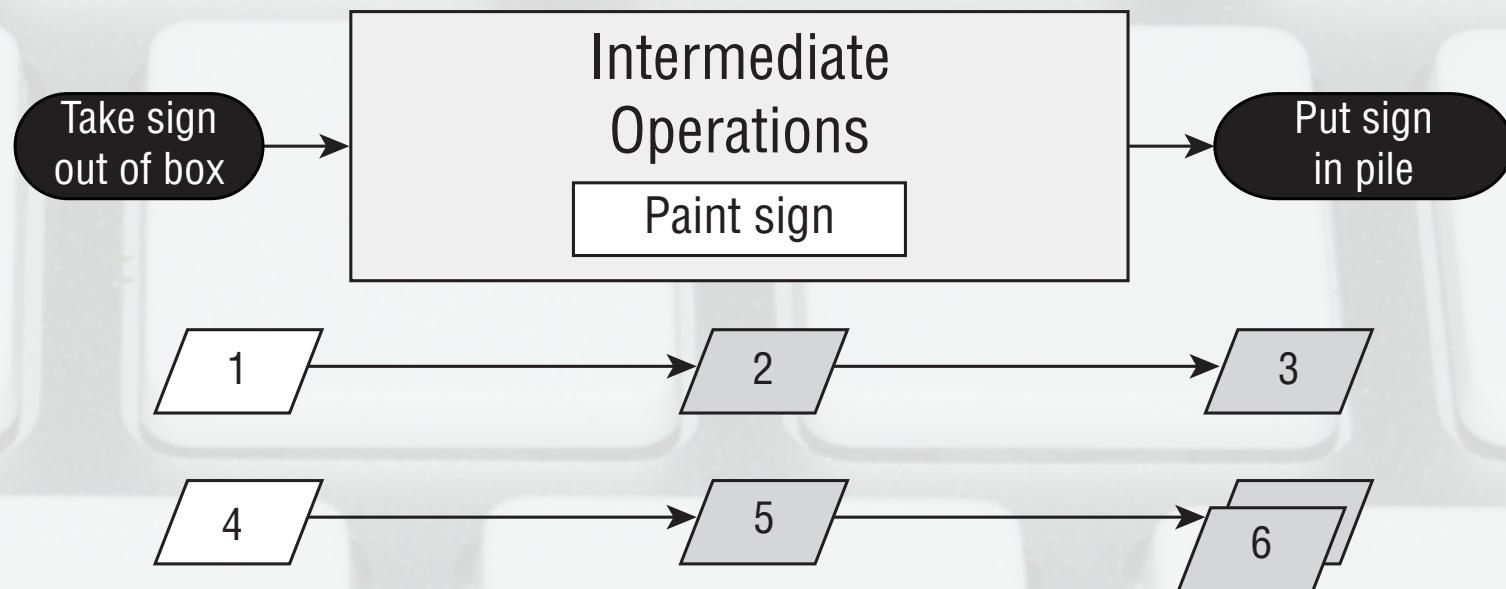
Stream Flow



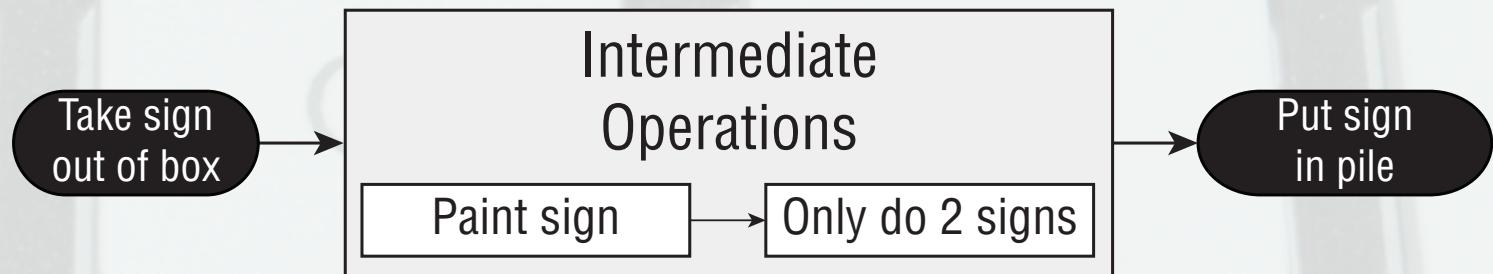
Stream Flow



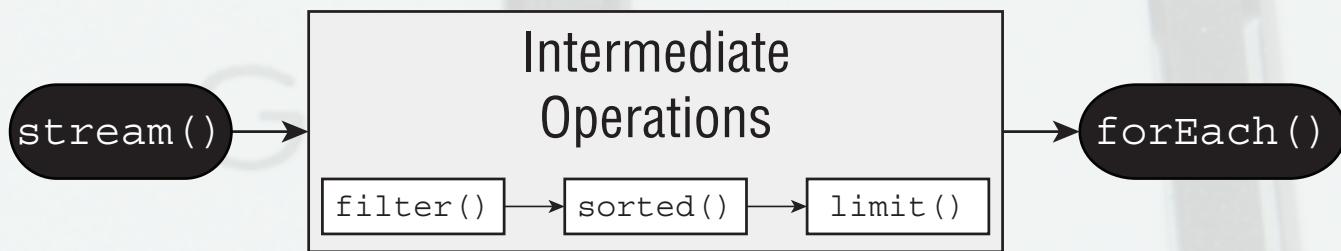
Painting Example



Painting Example



Painting Example



Basic Stream

```
return list.stream()
    .map(String::trim)
    .filter(s -> ! s.isEmpty())
    .map(String::length)
    .collect(Collectors.toList());
```

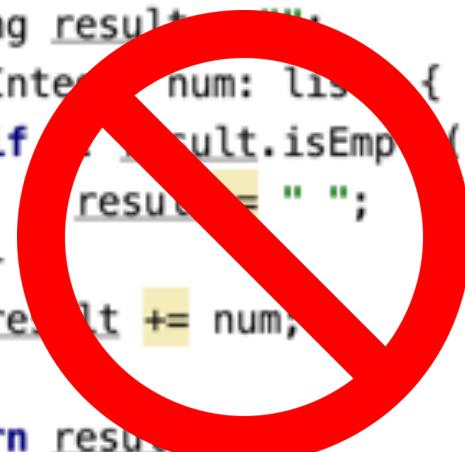
Find max

```
public OptionalInt max(List<Integer> list) {  
    if (list.isEmpty()) {  
        return OptionalInt.empty();  
    }  
    Collections.sort(list);  
    int max = list.get(list.size() - 1);  
    return OptionalInt.of(max);  
}
```

```
public OptionalInt maxStream(List<Integer> list) {  
    return list.stream().mapToInt(x -> x).max();  
}
```

Join a String

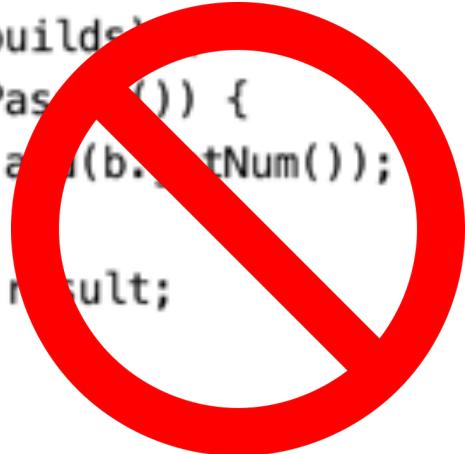
```
public String join(List<Integer> list) {  
    String result = "";  
    for(Integer num: list) {  
        if (!result.isEmpty()) {  
            result += " ";  
        }  
        result += num;  
    }  
    return result;  
}
```



```
public String joinStream(List<Integer> list) {  
    return list.stream()  
        .map(Object::toString)  
        .collect(Collectors.joining(" "));  
}
```

Get First Elements

```
List<Integer> result = new ArrayList<>();
for (Build b: builds) {
    if (! b.isPassed()) {
        result.add(b.getNum());
    } else {
        return result;
    }
}
return result;
```

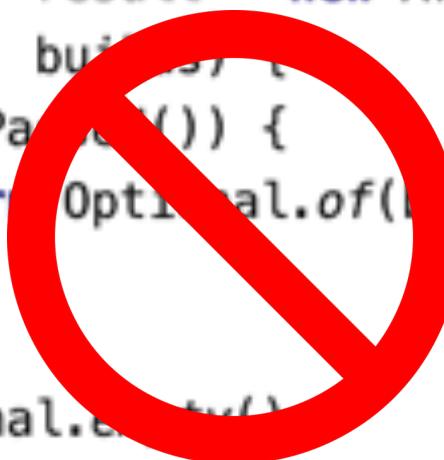


```
return builds.stream()
    .takeWhile(b -> ! b.isPassed())
    .map(Build::getNum)
    .collect(Collectors.toList())
```

Java 9

Get element

```
List<Integer> result = new ArrayList<>();
for (Build b: builds) {
    if (b.isPassed()) {
        return Optional.of(b.getNum());
    }
}
return Optional.empty();
```



```
return builds.stream()
    .dropWhile(b -> ! b.isPassed())
    .map(Build::getNum)
    .findFirst();
```

Java 9

Strings

```
public long countLinesWithQuestion(String text) {  
    return text.lines()  
        .filter(s -> s.contains("?"))  
        .count();  
}
```

```
public long countQuestionMarks(String text) {  
    return text.chars()  
        .filter(c -> c == '?')  
        .count();  
}
```

Useful Terminal Operations

- count()
- collect() - more on this next module
- allMatch(), anyMatch(), noneMatch()
- findFirst(), findAny()
- min()/max()
- foreach()

Useful Intermediate Operations

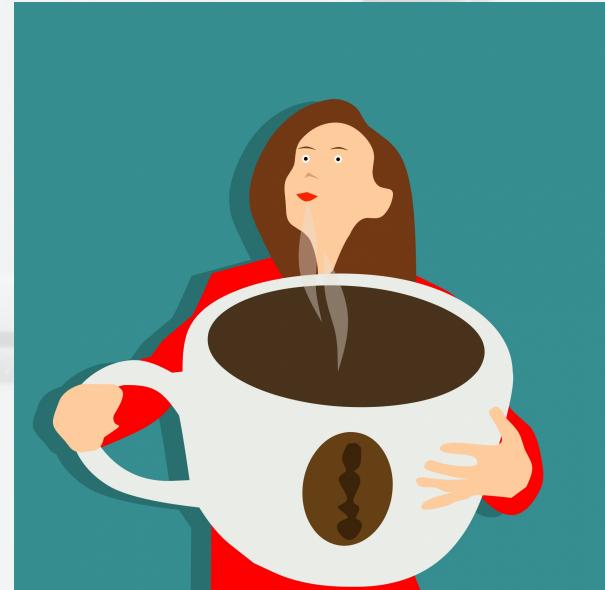
- filter()
- map()
- distinct()
- limit(), skip()
- sorted()
- peek()

Review - What method can...

- Return whether 5/5 values == 7?
- Return whether >=1 values == 7?
- Make a stream smaller?
- Change the order of stream elements?
- Loop through stream elements?
- Ignore the first 2 elements?
- Ignore any elements are the first 2?

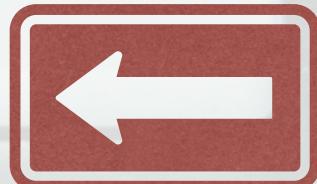
Lab & Break

<https://github.com/boyarsky/2020-devnexus-java-idioms>



Agenda

- Strings
- Regular Expressions
- Collections
- Streams
- **Advanced Streams**
- File I/O
- Other Useful APIs



Lab Review

- Did you get them all?
- Which was your longest solution?
- What did you come up with for `takeWhile()` or `dropWhile()`?

Infinite: Generating

```
public String tenStars() {  
    return Stream.generate(() -> "*")  
        .limit(10)  
        .collect(Collectors.joining());  
}
```

Infinite: Iterative

```
public String counting() {  
    return Stream.iterate( seed: 1, i-> i+1)  
        .limit(10)  
        .map(Object::toString)  
        .collect(Collectors.joining( delimiter: " "));  
}
```

```
public String countingWithLimit() {  
    return Stream.iterate( seed: 1, i -> i <= 10, i-> i+1)  
        .map(Object::toString)  
        .collect(Collectors.joining( delimiter: " "));  
}
```

Java 9

FlatMap

```
List<String> first = List.of("a", "b");
List<String> second = List.of();
List<List<String>> listOfLists = List.of(first, second);

return listOfLists.stream()
    .flatMap(Collection::stream)
    .collect(Collectors.toList());
```

Creating Map

```
public Map<String, Integer> mapByName(List<String> list) {  
    return list.stream()  
        .collect(Collectors.toMap(  
            s -> s,  
            String::length));  
}
```

Creating Map With Conflict

```
public Map<Integer, String> mapBySize(List<String> list) {  
    return list.stream()  
        .collect(Collectors.toMap(  
            String::length,  
            s -> s,  
            (a, b) -> a));  
}
```

Grouping By Key

```
public Map<Integer,List<String>> grouping(List<String> list) {  
    return list.stream()  
        .collect(Collectors.groupingBy(String::length));  
}
```

Partitioning By Key

```
public Map<Boolean,List<String>> partitioning(List<String> list) {  
    return list.stream()  
        .collect(Collectors.partitioningBy(String::isEmpty));  
}
```

Grouping By Key

```
public Map<Integer,Long> groupingWithCount(List<String> list) {  
    return list.stream()  
        .collect(Collectors.groupingBy(  
            String::length,  
            Collectors.counting()));  
}
```

Reduce

```
public Integer sum(List<Integer>list) {  
    return list.stream()  
        .reduce( identity: 0, (x, y) -> x+ y);  
}
```

```
public Optional<Integer> min(List<Integer>list) {  
    return list.stream()  
        .reduce((x, y) -> x < y ? x : y);  
}
```

Review

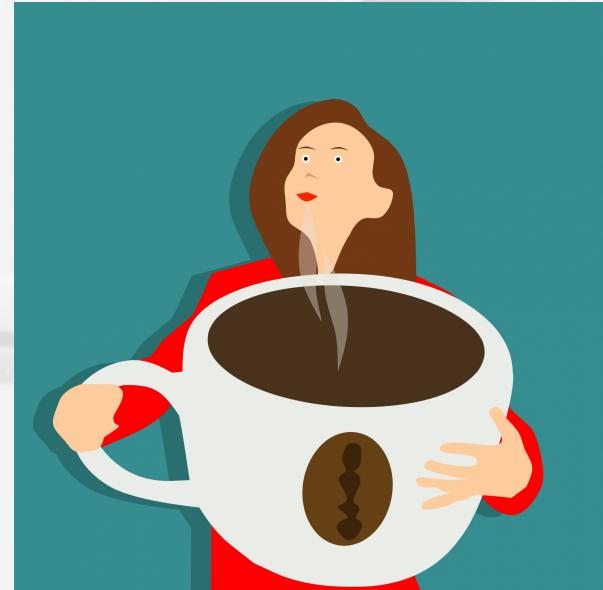
- Sources
 - generate()
 - iterate()
- Intermediate
 - flatMap()
- Terminal
 - reduce()
- Collectors
 - toMap()
 - groupingBy()
 - partitioningBy()

What method for?

- Creating an infinite stream of a times table?
- Creating a Map that always has two keys?
- Creating a Map with a custom key/value?
- Returning an Optional representing the stream?

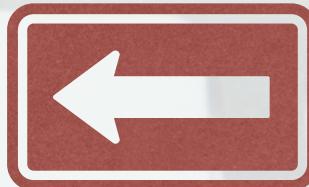
Lab & Break

<https://github.com/boyarsky/2020-devnexus-java-idioms>



Agenda

- Strings
- Regular Expressions
- Collections
- Streams
- Advanced Streams
- File I/O
- Other Useful APIs



Lab Review

- Did you get them all?
- How did you solve the challenge problem?
- Did you see how different methods are useful for different problems?

Creating a Path

```
Path path = Paths.get( first: "lecture/src");
```

```
Path path = Paths.get( first: "lecture", ...more: "src");
```

```
File file = new File( pathname: "lecture/src");
Path path = file.toPath();
```

Absolute Path

```
System.out.println(path.toAbsolutePath());
```

```
if (path.isAbsolute()) {
```

Separators

```
System.out.println(File.separator);
System.out.println(File.pathSeparator);
System.out.println(System.getProperty("file.separator"));
System.out.println(System.getProperty("path.separator"));
```

Windows:
\\ and ;

Mac/Linux:
/ and :

Read File into String

```
public String readIntoString(Path path) throws IOException {  
    return new String(Files.readAllBytes(path));  
}
```

```
public String readIntoString(Path path) throws IOException {  
    return Files.readString(path);  
}
```

Java 11

Read File into List

```
public List<String> readIntoList(Path path) throws IOException {  
    return Files.readAllLines(path);  
}
```

Read File with Stream

```
public long numEmptyLines(Path path) throws IOException {  
    try (Stream<String> lines = Files.lines(path)) {  
        return lines.map(String::trim)  
            .filter(String::isEmpty)  
            .count();  
    }  
}
```

Write File from String

```
public void writeFile(Path path, String text) throws IOException {  
    Files.write(path, text.getBytes());  
}
```

```
public void writeFromString(Path path, String text) throws IOException {  
    Files.writeString(path, text);  
}
```

Java 11

Write File from List

```
public void writeFile(Path path, List<StringBuilder> list)
    throws IOException {
    Files.write(path, list);
}
```

Append To File

```
public void append(Path path, List<String> list)
    throws IOException {
    Files.write(path, list, StandardOpenOption.APPEND);
}
```

Walk Tree

```
public Collection<String> getAll(Path path) throws IOException {  
    return Files.walk(path)  
        .map(Path::toString)  
        .collect(Collectors.toList());  
}
```

Convert to unchecked

```
public Collection<String> readAll(Path path)
    throws IOException {

    return Files.walk(path)
        .map(this::readFile)
        .collect(Collectors.toList());
}

private String readFile(Path path) {
    try {
        return Files.readString(path);
    } catch (IOException e) {
        throw new UncheckedIOException(e);
    }
}
```

Or
unchecked
wrapper

Delete File

```
Files.delete(outputPath);
```

```
Files.deleteIfExists(outputPath);
```

What API to?

- Read file into a single string?
- Write a List to a file?
- Get rid of a file?
- List all files in directory?
- Add to a file?
- Process file as a Stream<String>?

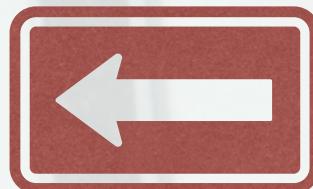
Lab & Break

<https://github.com/boyarsky/2020-devnexus-java-idioms>



Agenda

- Strings
- Regular Expressions
- Collections
- Streams
- Advanced Streams
- File I/O
- Other Useful APIs



Lab Review

- Did you get them all?
- How did you solve the challenge problem?
- Which seemed easier - reading all lines or lines() for the odd/even problem?

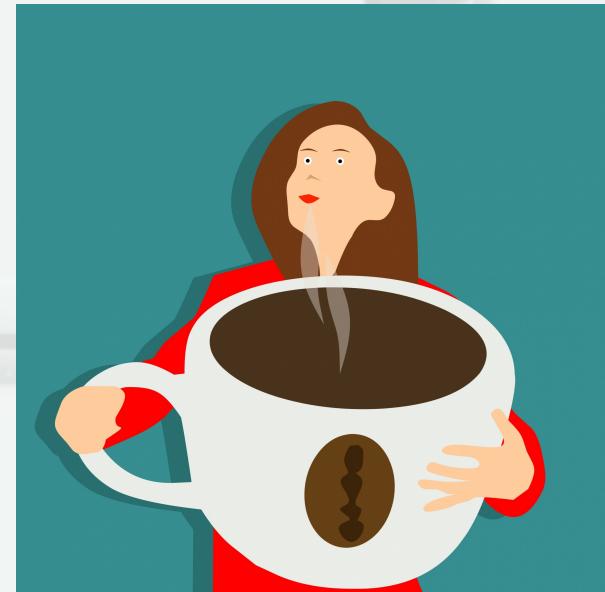
Math

```
public int absoluteValue(int x) {  
    return Math.abs(x);  
}
```

```
public int biggest(int x, int y) {  
    return Math.max(x, y);  
}
```

Lab & Break

See handout



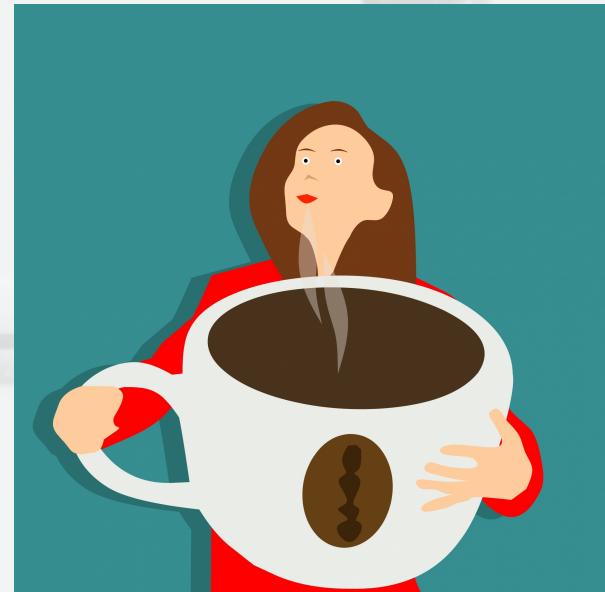
Math

```
public int smallest(int x, int y) {  
    return Math.min(x, y);  
}
```

```
public double power(double x, double y) {  
    return Math.pow(x, y);  
}
```

Lab & Break

See handout



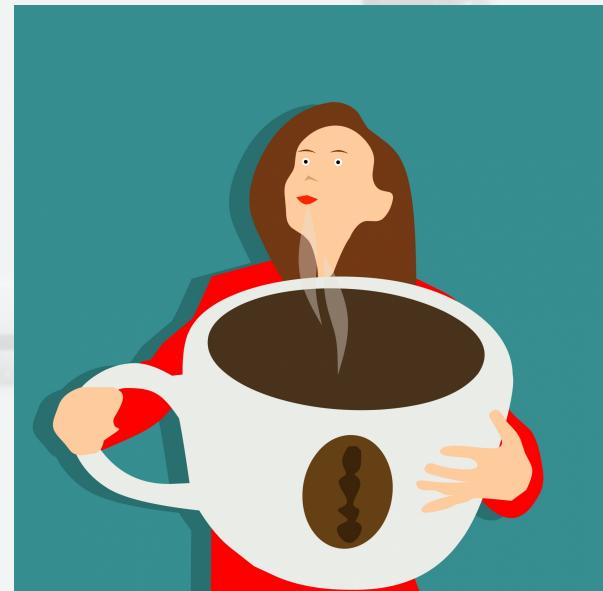
Random

```
// 0 <= x < 1
public double randomMath() {
    return Math.random();
}

// 0 <= x < 1
public double random() {
    Random random = new Random();
    return random.nextDouble();
}
```

Lab & Break

See handout



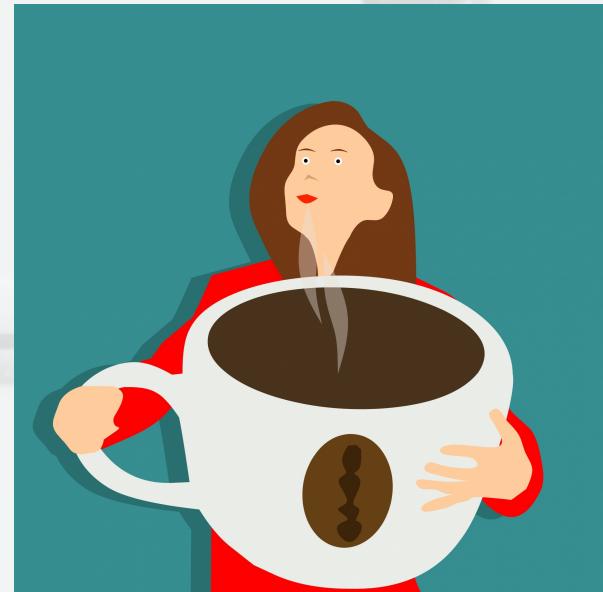
Random

```
// 0 <= x < max
public int randomIntUnderMax(int max) {
    Random random = new Random();
    return random.nextInt(max);
}

// 1 <= x <= max
public int randomIntOneToMax(int max) {
    Random random = new Random();
    return random.nextInt(max) + 1;
}
```

Lab & Break

See handout



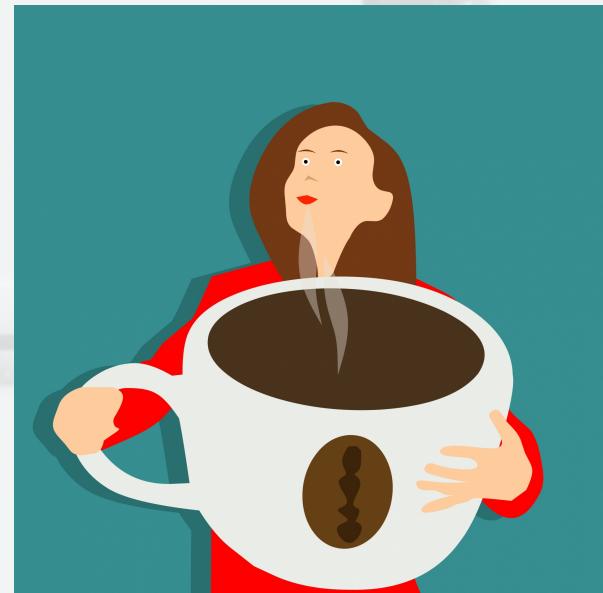
Random

```
public IntStream infiniteIntStream() {  
    Random random = new Random();  
    return random.ints();  
}
```

```
public IntStream finiteIntStream() {  
    Random random = new Random();  
    return random.ints( streamSize: 5 );  
}
```

Lab & Break

See handout

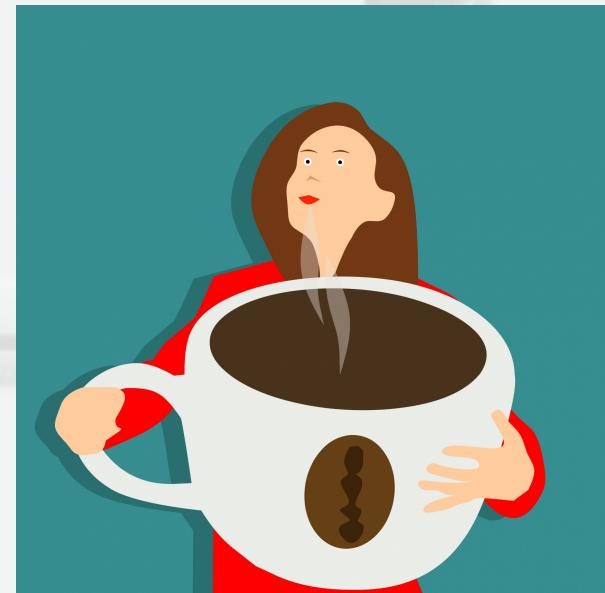


Random

```
// example: fe3957e5-d91a-48db-9631-260fedb0e341
public UUID uuid() {
    return UUID.randomUUID();
}
```

Lab & Break

See handout



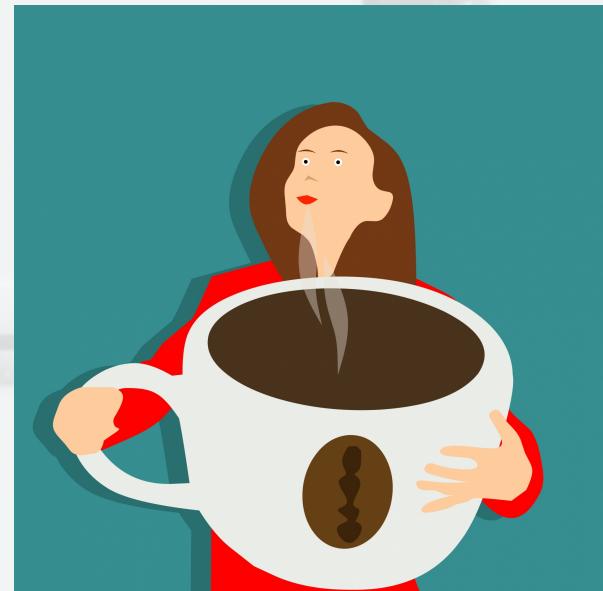
LocalDate

```
public LocalDate now() {  
    return LocalDate.now();  
}
```

```
public LocalDate presidentsDay() {  
    return LocalDate.of( year: 2020, Month.FEBRUARY, dayOfMonth: 17);  
}
```

Lab & Break

See handout



LocalDate

```
public LocalDate dateMath(LocalDate date) {  
    return date.plusDays(4).minusWeeks(1);  
}
```

```
@Test  
void presidentsDay() {  
    LocalDate actual = target.presidentsDay();  
    assertEquals( expected: 2020, actual.getYear());  
    assertEquals(Month.FEBRUARY, actual.getMonth());  
    assertEquals( expected: 17, actual.getDayOfMonth());  
}
```

Lab & Break

See handout



What API to?

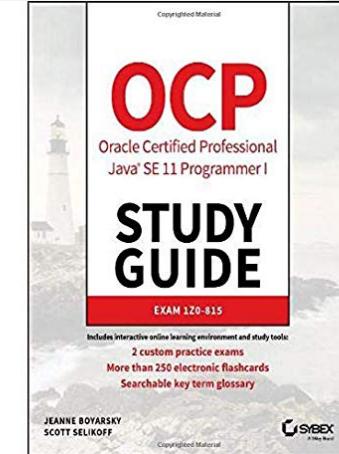
- Get today?
- Get July 4th of this year?
- Create a random number between 2 and 10?
- Get the largest value of two?
- Get a unique id?

Before the last lab:

Tomorrow 10:20am

Java: Did you know that?

+ Win a book



Lab & Break

<https://github.com/boyarsky/2020-devnexus-java-idioms>

