CompSci 201, L7: Runtime Efficiency

Logistics, Coming up

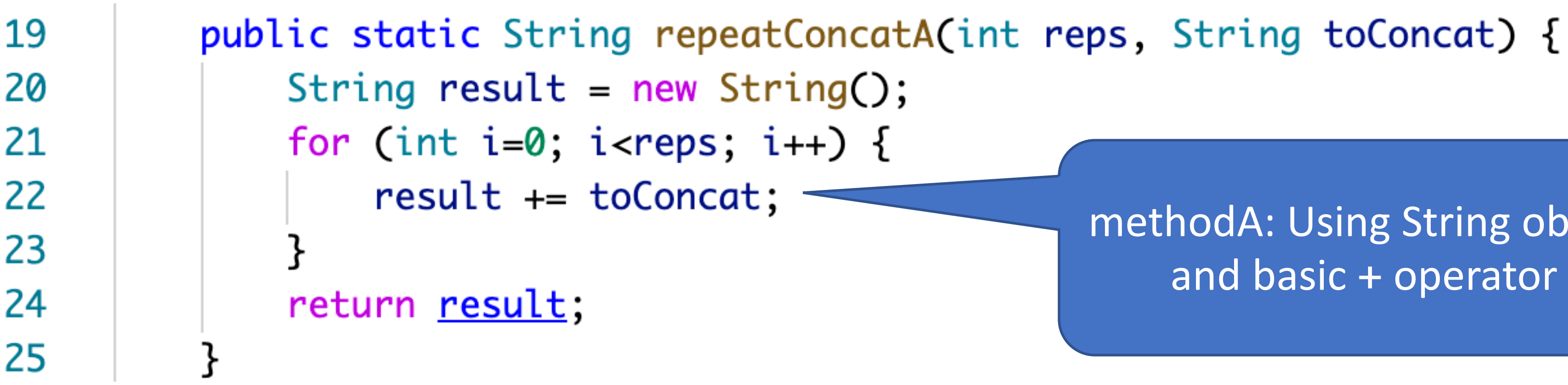
- Today
  - Project 1 Nbody due today
  - Runtime efficiency
  - Project 2 Markov releasing later (due in 2 weeks)

- Wednesday 9/21
  - APT 3 due
  - Big O / Asymptotic Analysis

- Friday 9/23
  - Discussion: Maps, Big O, hashCode

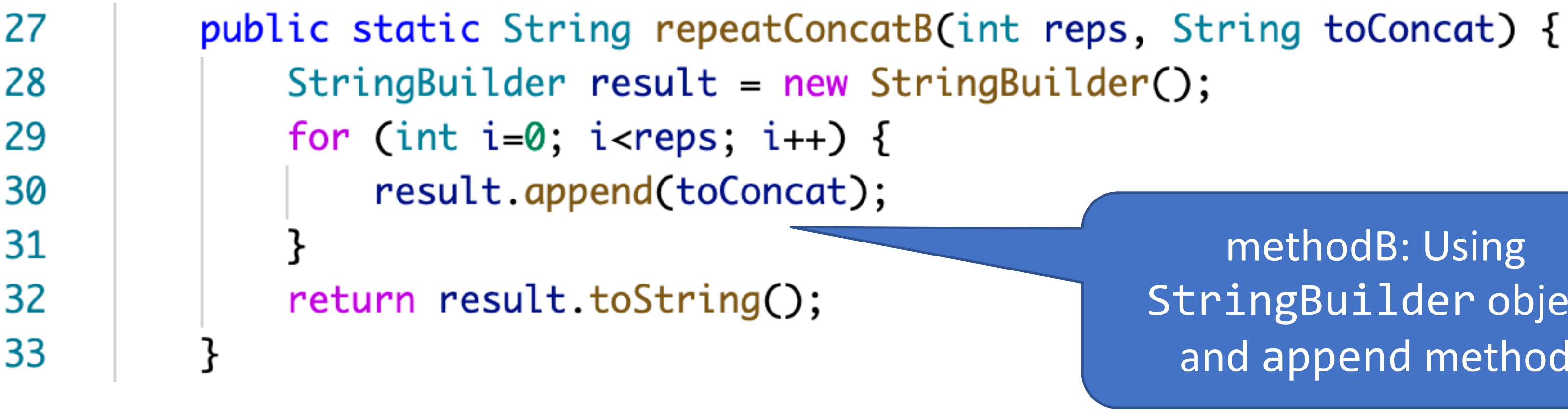
Runtime Efficiency, an Empirical Look at String Concatenation

Two methods for repeated concatenation



```java
public static String repeatConcatA(int reps, String toConcat) {
    String result = new String();
    for (int i=0; i<reps; i++) {
        result += toConcat;
    }
    return result;
}
```

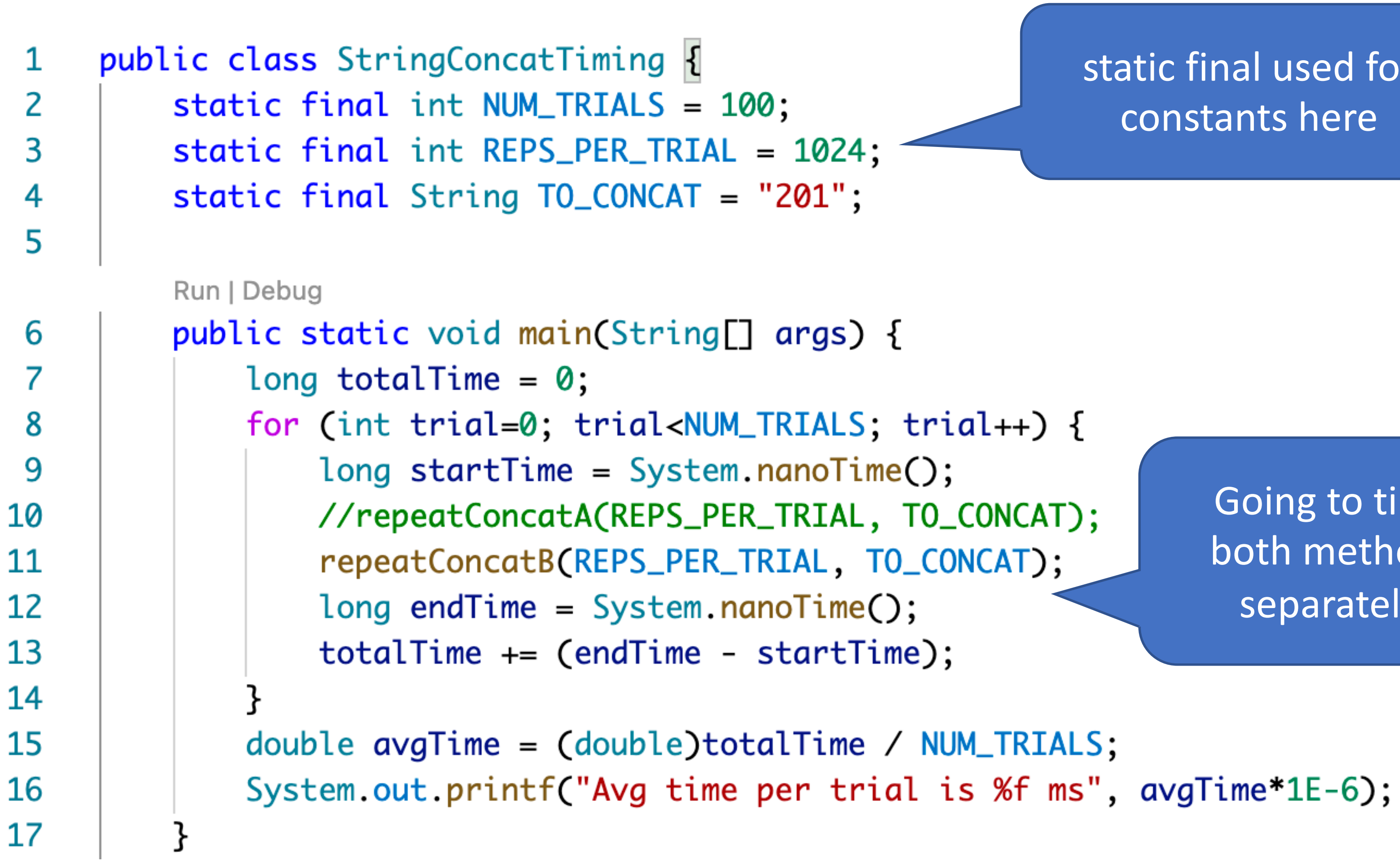
methodA: Using String object and basic + operator



```java
public static String repeatConcatB(int reps, String toConcat) {
    StringBuilder result = new StringBuilder();
    for (int i=0; i<reps; i++) {
        result.append(toConcat);
    }
    return result.toString();
}
```

methodB: Using StringBuilder object and append method

Empirical timing experiment



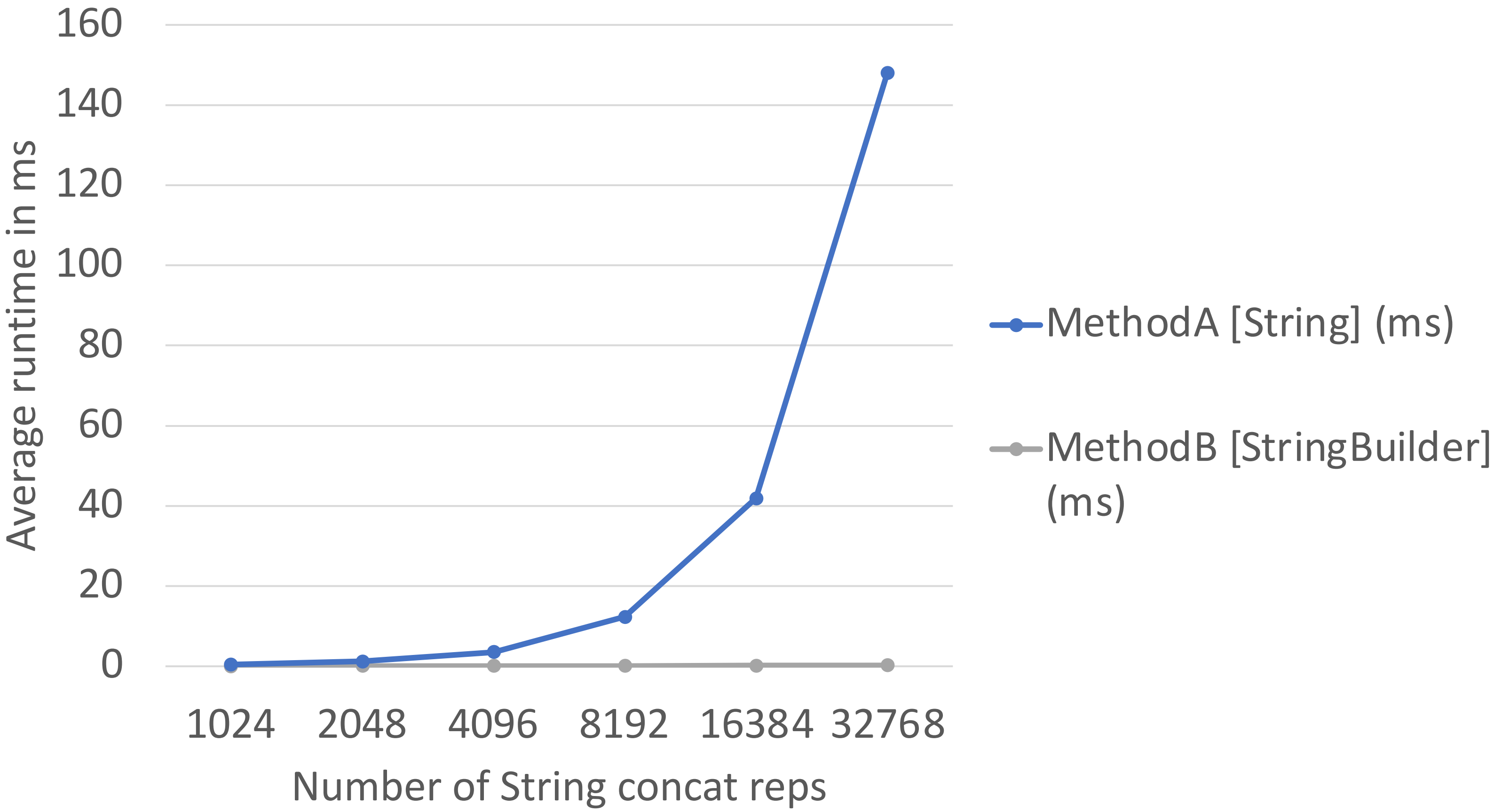
```java
public class StringConcatTiming {
    static final int NUM_TRIALS = 100;
    static final int REPS_PER_TRIAL = 1024;
    static final String TO_CONCAT = "201";

    Run | Debug
    public static void main(String[] args) {
        long totalTime = 0;
        for (int trial=0; trial<NUM_TRIALS; trial++) {
            long startTime = System.nanoTime();
            //repeatConcatA(REPS_PER_TRIAL, TO_CONCAT);
            repeatConcatB(REPS_PER_TRIAL, TO_CONCAT);
            long endTime = System.nanoTime();
            totalTime += (endTime - startTime);
        }
        double avgTime = (double)totalTime / NUM_TRIALS;
        System.out.printf("Avg time per trial is %f ms", avgTime*1E-6);
    }
```

static final used for constants here

Going to time both methods separately.

Empirical results



Empirical results in more detail

Reps	MethodA (ms)	MethodB (ms)
1024	0.384	0.050
2048	1.136	0.061
4096	3.443	0.077
8192	12.244	0.099
16384	41.754	0.143
32768	147.719	0.207

Multiply reps by 2 multiplies runtime by 4. Quadratic complexity.

Multiply reps by 2 multiplies runtime by ~2. Linear complexity.

Empirical results in more detail

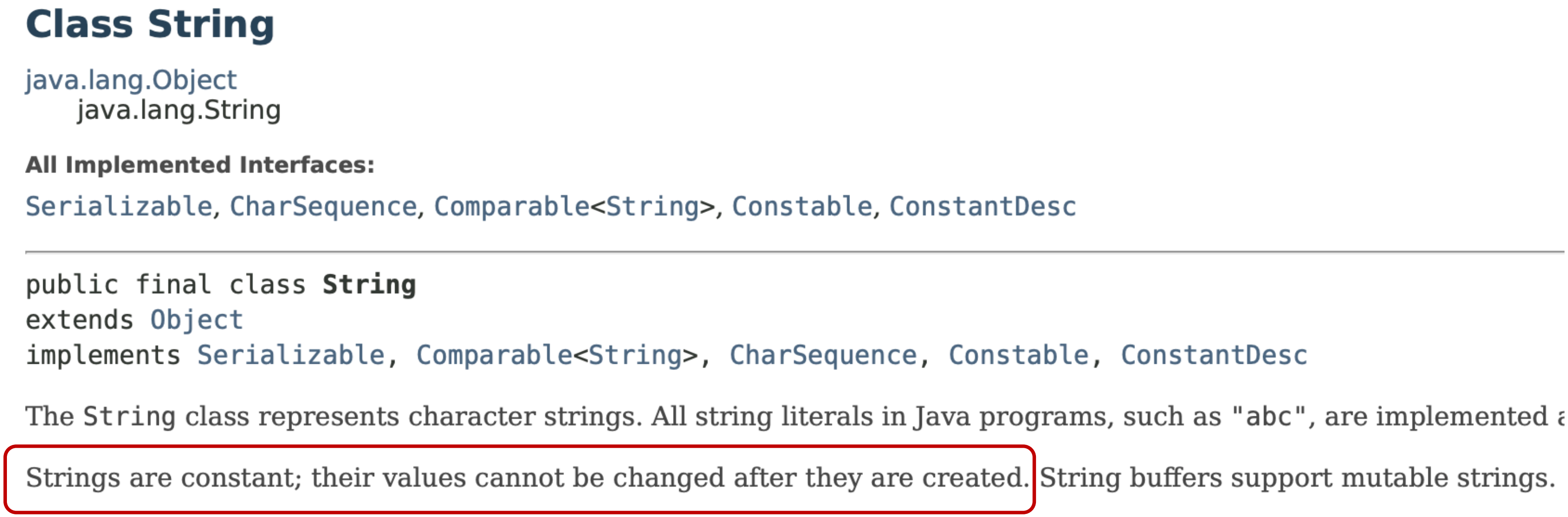
Reps	MethodA ns/rep	MethodB ns/rep
1024	0.375	0.048
2048	0.555	0.030
4096	0.841	0.019
8192	1.495	0.012
16384	2.548	0.009
32768	4.508	0.006

Runtime / rep increasing, *greater than linear* complexity.

Runtime / rep not increasing, at most linear complexity.

What’s going on? Documentation?

docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String



Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

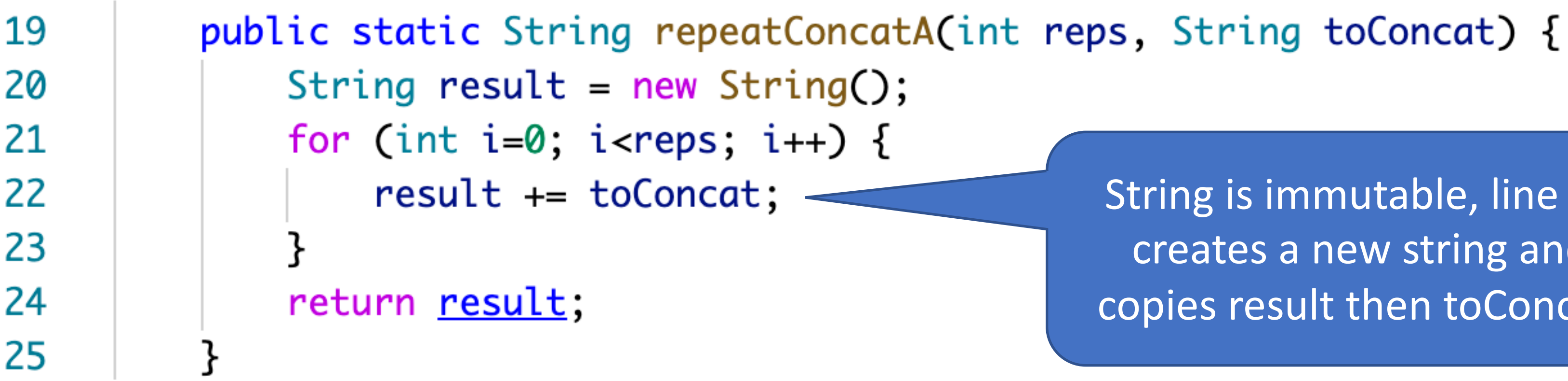
Serializable, CharSequence, Comparable<String>, Constable, ConstantDesc

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence, Constable, ConstantDesc
```

The String class represents character strings. All string literals in Java programs, such as "abc", are implemented

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings.

methodA revisited



```
19    public static String repeatConcatA(int reps, String toConcat) {
20        String result = new String();
21        for (int i=0; i<reps; i++) {
22            result += toConcat;
23        }
24        return result;
25    }
```

String is immutable, line 22 creates a new string and copies result then toConcat.

How many characters will be copied per iteration if `toConcat == “201”`?

- i=0: 3
- i=1: 6
- i=2: 9
- …
- On iteration i, need to copy 3*(i+1) characters!

How many total characters are copied? Algebra!

methodA: i goes from 0 to reps-1, copy 3*(i+1) characters per iteration.

$$\sum_{i=0}^{\mathrm{reps}-1} 3(\mathrm{i}+1) = 3(\mathrm{reps}) + 3\left(\sum_{i=0}^{\mathrm{reps}-1} \mathrm{i}\right)$$

$$= 3(\mathrm{reps}) + 3\left(\frac{\mathrm{reps}}{2}\right)(0 + \mathrm{reps} - 1)$$

$$\approx \frac{3}{2}\mathrm{reps}^2 + \mathrm{reps}$$

Arithmetic series formula:

$$\sum_{i=1}^{n} a_i = \left(\frac{n}{2}\right)(a_1 + a_n)$$

Abstracting, Intro to Big O Notation (Preview for next time)

- The 3/2 in $\frac{3}{2}\mathrm{reps}^2$ doesn’t tell us much about how the performance *scales with the size of reps*.

- Often, we use *asymptotic notation*, especially *Big O notation* to abstract away constants.

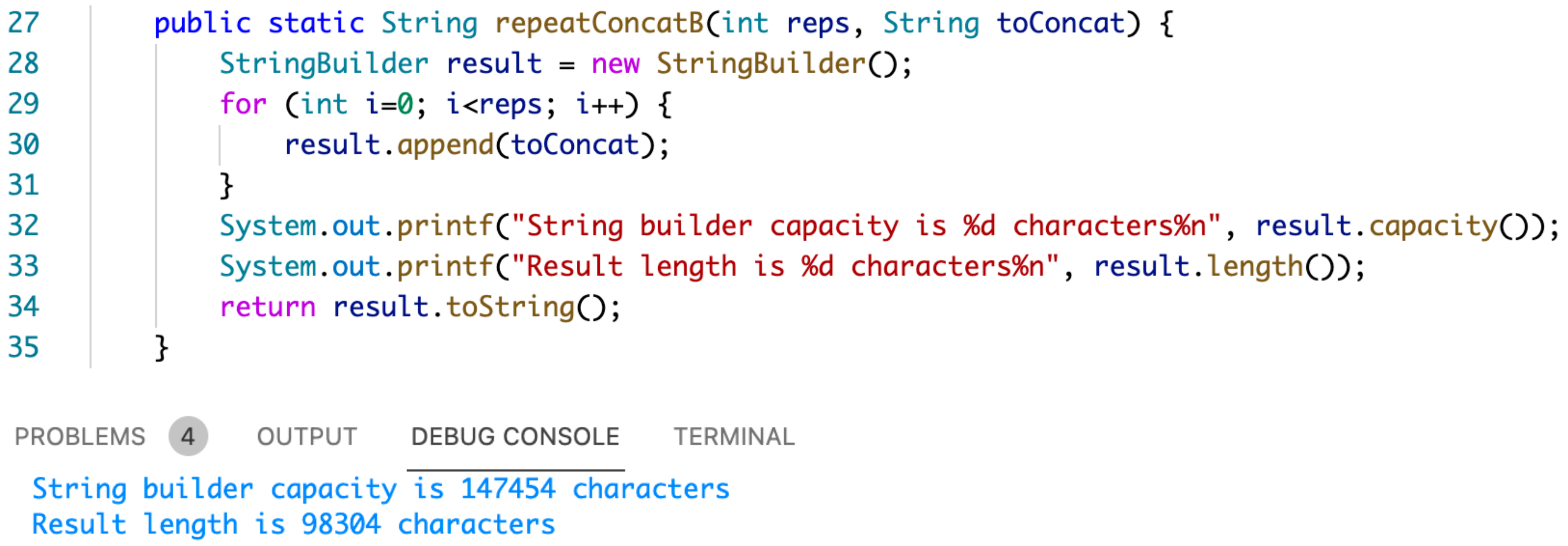
- For example: let N = reps, then we say that the asymptotic runtime complexity is O(N^2).
    - If you ~double N, you ~quadruple the runtime

What’s the real difference between methodA and methodB?

- methodA: Copies roughly $\frac{3}{2}\text{reps}^2$ characters.
- methodB: i goes from 0 to reps-1, copy 3 characters per iteration → copies roughly $3\times\text{reps}$ characters.

Reps	MethodA char copies	MethodB char copies
1024	1572864	3072
2048	6291456	6144
4096	25165824	12288
8192	100663296	24576
16384	402653184	49152
32768	1610612736	98304

Memory/Runtime Tradeoff



```java
    public static String repeatConcatB(int reps, String toConcat) {
        StringBuilder result = new StringBuilder();
        for (int i=0; i<reps; i++) {
            result.append(toConcat);
        }
        System.out.printf("String builder capacity is %d characters%n", result.capacity());
        System.out.printf("Result length is %d characters%n", result.length());
        return result.toString();
    }
```

```
String builder capacity is 147454 characters
Result length is 98304 characters
```

Final StringBuilder is using about 146k / 98k ~= 1.5 times as much memory as necessary. Very common tradeoff in data structures!

How does StringBuilder work?

"Every string builder has a capacity. As long as the length of the character sequence contained in the string builder does not exceed the capacity, it is not necessary to allocate a new internal buffer. If the internal buffer overflows, it is automatically made larger." - StringBuilder JDK 17 documentation.

- But how does it grow?
- Geometrically! Like ArrayList, HashMap, …
    - Still linear amortized complexity, for same reasons

WOTO

Go to duke.is/57dsn

Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!

Designing more efficient algorithms: Examples with HashMaps

CounterAttack APT

- CounterAttack APT
  - Count the number occurrences in `str` of each string in `words`.
  - Idea from discussion 3? Use `Collections.frequency()`

```
str = "one two one two one two vorpal blade"
words = {"snicker", "one", "blade", "runner"}
Returns {0,3,1,0}
```

Efficiency of current solution

- Suppose `String[] words` has N strings
- Suppose `str` has M Strings

Current algorithm:

- For each of the N strings in `words` :
    - count # occurrences in `str`: compare to M strings

$\mathrm{M}\times\mathrm{N}$ total comparisons, algorithm has O(MN) complexity. Can we decrease this?

Using a Map for M+N complexity

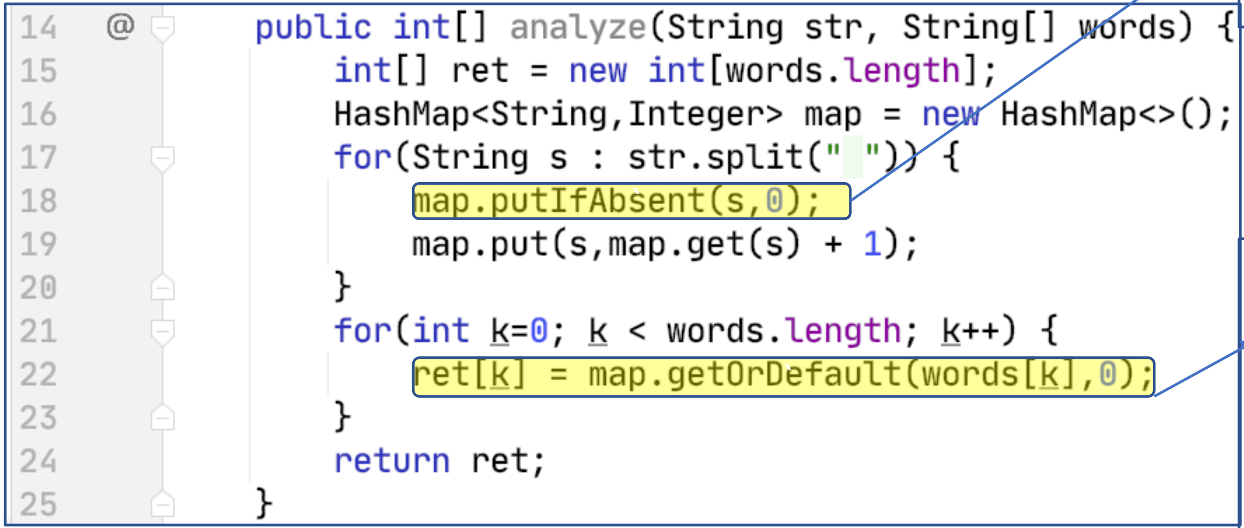
- Instead, use a Map to keep track, loop through words in **str** just once.

```
str = "one two one two one two vorpal blade"
words = {"snicker", "one", "blade", "runner"}
Returns {0,3,1,0}
```

Key	Value
one	3
two	3
vorpal	1
blade	1

Using a Map for M+N complexity

- **`HashMap<String,Integer> map`** stores counts
  - Avoid **`putIfAbsent/getOrDefault`**?
    - Guard with if statements



```java
public int[] analyze(String str, String[] words) {
    int[] ret = new int[words.length];
    HashMap<String,Integer> map = new HashMap<>();
    for(String s : str.split(" ")) {
        map.putIfAbsent(s,0);
        map.put(s,map.get(s) + 1);
    }
    for(int k=0; k < words.length; k++) {
        ret[k] = map.getOrDefault(words[k],0);
    }
    return ret;
}
```

```java
if (! m.containsKey(s)){
     m.put(s,0);
}
```

```java
if (m.containsKey(s)){
     ret[k] =
m.get(words[k])
}
else {
     ret[k] = 0;
}
```

NM vs .N+M Complexity

O(NM)

If we double N and double M?

- Runtime increases by a factor of 4.

What if N >> M and we double M?

- Doubles runtime, M still relevant

O(N+M)

If we double N and double M?

- Runtime increases by a factor of 2.

What if N >> M and we double M?

- Little difference in runtime, N dominates

Leetcode Isomorphic Strings

leetcode.com/problems/isomorphic-strings

<LiveCoding>