

10/17/22

Palindrome Review

Chapter 4.3

isPalindrome() by reversing

On the Palindrome exercise, this was the most common solution.
Compare the string to its reverse to see if it's a palindrome. It works!

```
public static String reverseString(String s) {  
    String result = "";  
    for (int i = s.length() - 1; i >= 0; i--) {  
        result += s.charAt(i);  
    }  
    return result;  
}  
  
public static boolean isPalindrome(String word) {  
    return word.equals(reverseString(word));  
}
```

isPalindrome() using StringBuilder

We saw the `StringBuilder` class last week, which can build up Strings using mutable operations. It has a built-in `reverse()` method, letting us do the check with very little code.

```
public static String reverseString(String s) {  
    return new StringBuilder(s).reverse().toString();  
}  
  
public static boolean isPalindrome(String word) {  
    return word.equals(reverseString(word));  
}
```

isPalindrome() using direct comparison

Reversing the string requires allocating memory. We can avoid this by directly comparing the characters in the original string.

```
public static boolean isPalindrome(String word) {  
    for (int i = 0, j = word.length() - 1; i < j; i++, j--) {  
        if (word.charAt(i) != word.charAt(j)) {  
            return false;  
        }  
    }  
    return true;  
}
```

Most string loops you've seen so far have a single counter.

This one starts two counters, one at each end of the string, and "races" them toward the middle until they meet, which ends the loop.

isPalindrome benchmark

Reversing the string through concatenation (isPalindromeReversed) generates many String instances which get discarded immediately (garbage objects).

isPalindromeReverseBuilder (StringBuilder) is ~4X faster than isPalindromeReversed

isPalindrome with no string reversal is ~8X faster than isPalindromeReversed

```
isPalindrome: 0.030893907 s
isPalindromeReversed: 0.236941624 s
isPalindromeReverseBuilder: 0.063064403 s
```

```
public static void main(String[] args) throws IOException {
    String[] words = readWordsFile();

    long startTime = System.nanoTime();
    for (int pass = 0; pass < NUM_PASSES; pass++) {
        for (int i = 0; i < words.length; i++) {
            isPalindrome(words[i]);
        }
    }
    long elapsed = System.nanoTime() - startTime;
    System.out.println("isPalindrome: " + (elapsed/1e9) + " s");

    startTime = System.nanoTime();
    for (int pass = 0; pass < NUM_PASSES; pass++) {
        for (int i = 0; i < words.length; i++) {
            isPalindromeReversed(words[i]);
        }
    }
    elapsed = System.nanoTime() - startTime;
    System.out.println("isPalindromeReversed: " + (elapsed/1e9) + " s");

    startTime = System.nanoTime();
    for (int pass = 0; pass < NUM_PASSES; pass++) {
        for (int i = 0; i < words.length; i++) {
            isPalindromeReverseBuilder(words[i]);
        }
    }
    elapsed = System.nanoTime() - startTime;
    System.out.println("isPalindromeReverseBuilder: " + (elapsed/1e9) + " s");
}
```

Benchmark nested loop

Computers are so fast that you may have to benchmark a method millions of times to get a good reading.

Here, we use a nested loop to run through the entire EOWL word list multiple times. (NUM_PASSES is set to 10, and words.length is 128985)

```
long startTime = System.nanoTime();
for (int pass = 0; pass < NUM_PASSES; pass++) {
    for (int i = 0; i < words.length; i++) {
        isPalindrome(words[i]);
    }
}
long elapsed = System.nanoTime() - startTime;
System.out.println("isPalindrome: " + (elapsed/1e9) + " s");
```

The computer was able to go through a dictionary 10X in 0.0295 seconds = 29.5 milliseconds!

Performance Optimization

Analyzing your code and making improvements to make it faster and/or use less memory is called **optimization**.

Not every piece of code needs to be optimized to the max...

There are trade-offs to evaluate:

- How much of my time do I want to spend making it faster or less memory-intensive?
- Is a more naive algorithm simpler to understand for other programmers?
- How will the code actually be used IRL?

However, sometimes you think your code will only have to deal with 1,000 records, and then the company gets some big customer and the same code is subjected to 1,000,000 records!

Java and other languages have **compiler optimizations** that make your code faster automatically... but these optimizations aren't able to change the fundamentals of your algorithms.

Nested Loops

Chapter 4.4

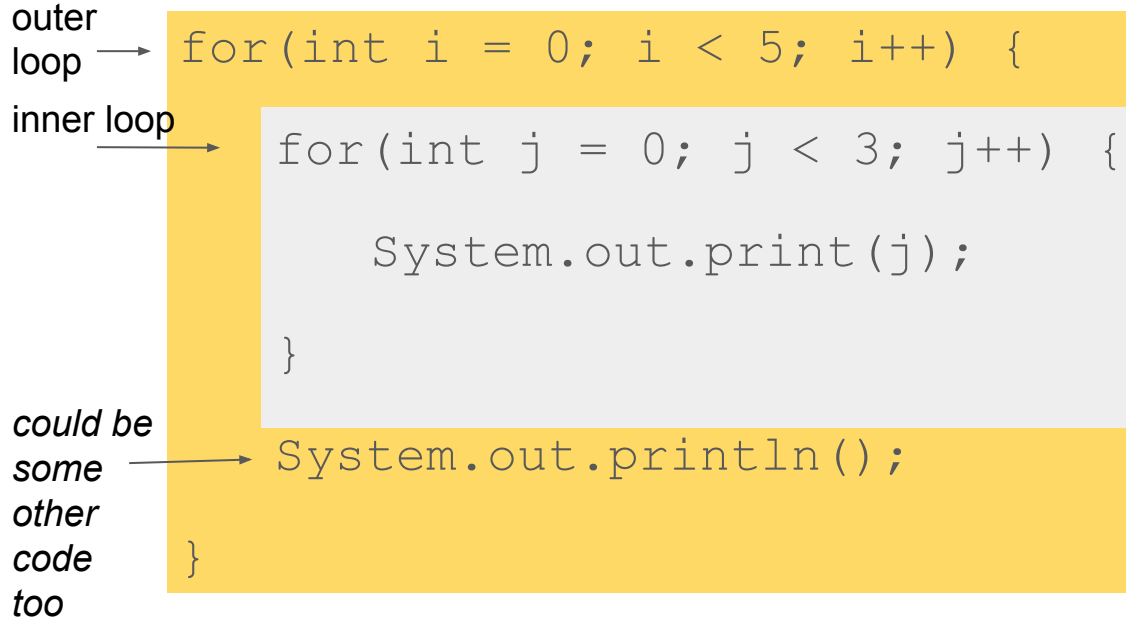
Nested Loops

A nested loop is a loop inside another loop, possibly >2 levels.

```
for(int i = 0; i < 5; i++) {  
    for(int j = 0; j < 3; j++) {  
        System.out.print(j);  
    }  
    System.out.println();  
}
```

Nested Loops

A nested loop is a loop inside another loop, possibly >2 levels.



The diagram illustrates nested loops with the following code structure:

```
for(int i = 0; i < 5; i++) {  
    for(int j = 0; j < 3; j++) {  
        System.out.print(j);  
    }  
    System.out.println();  
}
```

Annotations and arrows:

- outer loop** → points to the first `for` loop header.
- inner loop** → points to the second `for` loop header.
- could be some other code too** → points to the `System.out.println();` statement.

Rows and Columns

i = 0		j = 0		j = 1		j = 2	
i = 1		j = 0		j = 1		j = 2	
i = 2		j = 0		j = 1		j = 2	
i = 3		j = 0		j = 1		j = 2	
i = 4		j = 0		j = 1		j = 2	

```
for(int i = 0; i < 5; i++) {  
    for(int j = 0; j < 3; j++) {  
        System.out.print(j);  
    }  
    System.out.println();  
}
```

- The **outer loop** iterates through the rows
- The **inner loop** iterates through the columns
- The inner loop runs in its entirety on each iteration of the outer loop

Tracing Nested Loops

```
for(int i = 0; i < 5; i++) {  
    for(int j = 0; j < 3; j++) {  
        System.out.print(j);  
    }  
    System.out.println();  
}
```

i	j
0	0
0	1
0	2
1	0
1	1
1	2
2	0
2	1
2	2
3	0
3	1
3	2
4	0
4	1
4	2

```
class Main {  
    public static void main(String[] args) {  
        int n = 3;  
        int m = 3;  
        for (int i=0; i<n; i++) {  
            boolean topOrBottom = i == 0 || i == n-1;  
            for (int j=0; j<m; j++) {  
                char ch = ' ';  
                boolean leftOrRight = j == 0 || j == m-1;  
                if (topOrBottom && leftOrRight) {  
                    ch = '+';  
                } else if (topOrBottom) {  
                    ch = '-';  
                } else if (leftOrRight) {  
                    ch = '|';  
                }  
                System.out.print(ch);  
            }  
            System.out.println();  
        }  
    }  
}
```

Inner loop can depend on outer loop

```
class Main {  
    public static void main(String[] args) {  
        for (int i=1; i<=100; i++) {  
            boolean p = true;  
            for (int j=2; j<i; j++) {  
                if (i % j == 0) {  
                    p = false;  
                    break;  
                }  
            }  
            if (p) {  
                System.out.println(i);  
            }  
        }  
    }  
}
```

Finding Prime Numbers

This code does the same thing, less cryptically.

How many loop iterations does this code do?

Is this algorithm efficient?

```
class Main {  
    public static boolean isPrime(int x) {  
        for (int i=2; i<x; i++) {  
            if (x % i == 0) {  
                return false;  
            }  
        }  
        return true;  
    }  
  
    public static void main(String[] args) {  
        for (int i=1; i<=100; i++) {  
            if (isPrime(i)) {  
                System.out.println(i);  
            }  
        }  
    }  
}
```


Sieve of Eratosthenes

A more efficient algorithm was discovered... by a Greek mathematician in the 3rd century BC.

	2	3	4	5	6	7	8	9	10	Prime numbers
11	12	13	14	15	16	17	18	19	20	
21	22	23	24	25	26	27	28	29	30	
31	32	33	34	35	36	37	38	39	40	
41	42	43	44	45	46	47	48	49	50	
51	52	53	54	55	56	57	58	59	60	
61	62	63	64	65	66	67	68	69	70	
71	72	73	74	75	76	77	78	79	80	
81	82	83	84	85	86	87	88	89	90	
91	92	93	94	95	96	97	98	99	100	
101	102	103	104	105	106	107	108	109	110	
111	112	113	114	115	116	117	118	119	120	



No hammocks on the sculpture!

When the numbers are sufficiently large, no efficient [non-quantum integer factorization algorithm](#) is known.

Practice!

MultiplicationTable

Fibonacci

FibonacciSpiral

Multiplication Table

■ MultiplicationTable										
Multiplication Table										
	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

Multiplication Table

```
MultiplicationTable.css x +
1 ▼ table {
2   width: 80%;
3   font-family: Arial, Helvetica, sans-serif;
4   padding: 0px;
5 }
6
7 ▼ .topSide {
8   color: white;
9   font-weight: bold;
10  background-color: #d7bde2;
11 }
12
13 ▼ .leftSide {
14   color: white;
15   font-weight: bold;
16   background-color: #d4efdf;
17 }
18
19 ▼ .cell {
20   color: white;
21   background-color: #aed6f1;
22 }
23
24 ▼ td {
25   text-align: center;
26 }
```

```
<h2>Multiplication Table</h2>
<table>
  <tr>
    <td class="topSide"></td>
    <td class="topSide">1</td>
    <td class="topSide">2</td>
    <td class="topSide">3</td>
    <td class="topSide">4</td>
    <td class="topSide">5</td>
    <td class="topSide">6</td>
    <td class="topSide">7</td>
    <td class="topSide">8</td>
    <td class="topSide">9</td>
    <td class="topSide">10</td>
  </tr>
  <tr>
    <td class="leftSide">1</td>
    <td class="cell">1</td>
    <td class="cell">2</td>
    <td class="cell">3</td>
    <td class="cell">4</td>
    <td class="cell">5</td>
    <td class="cell">6</td>
    <td class="cell">7</td>
    <td class="cell">8</td>
    <td class="cell">9</td>
    <td class="cell">10</td>
  </tr>
```

Fibonacci Numbers

The Fibonacci numbers may be defined by the [recurrence relation](#)^[6]

$$F_0 = 0, \quad F_1 = 1,$$

and

$$F_n = F_{n-1} + F_{n-2}$$

for $n > 1$.

The first 20 Fibonacci numbers F_n are:^[1]

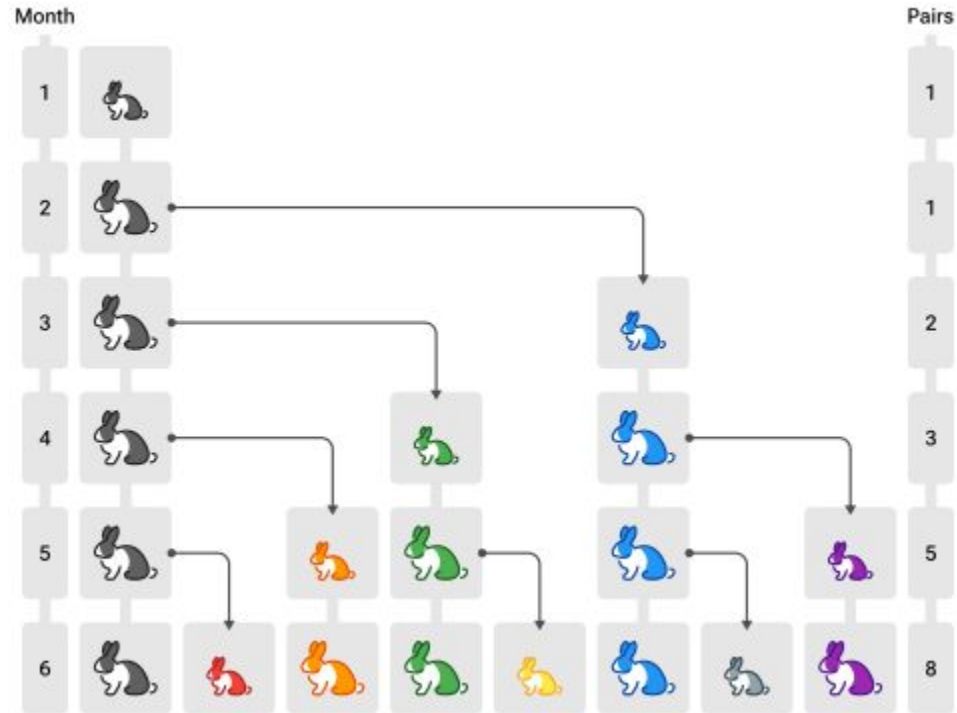
F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181



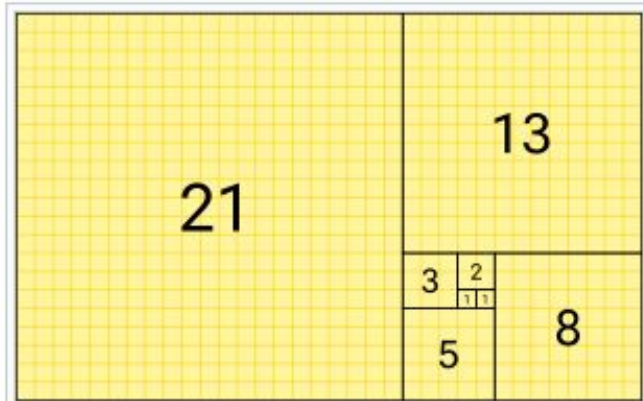
Fibonacci (c. 1170 – c. 1240–50) was an [Italian mathematician](#) from the [Republic of Pisa](#), considered to be "the most talented Western mathematician of the [Middle Ages](#)".

Applications of Fibonacci numbers include computer algorithms such as the [Fibonacci search technique](#) and the [Fibonacci heap](#) data structure, and graphs called [Fibonacci cubes](#) used for interconnecting parallel and distributed systems

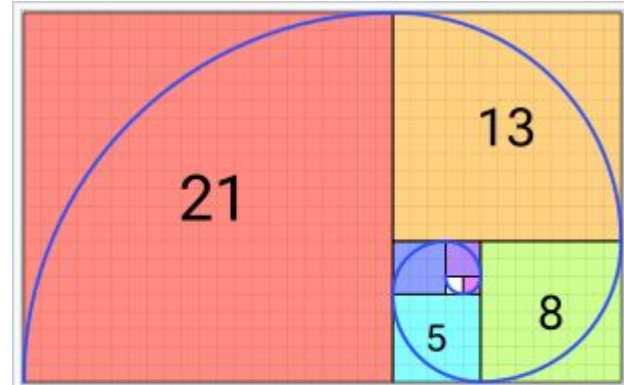
Fibonacci Numbers



Fibonacci Spiral

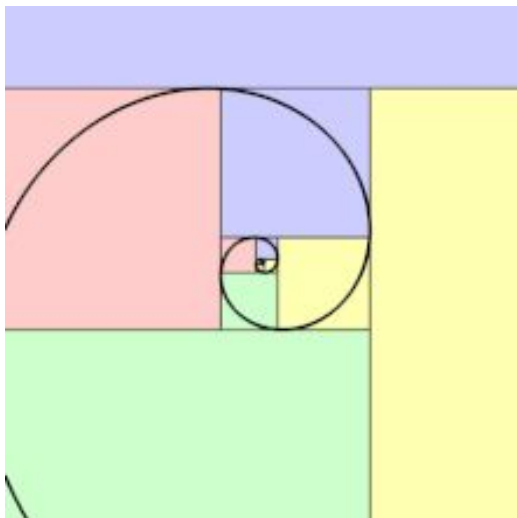


A tiling with squares whose side lengths are successive Fibonacci numbers: 1, 1, 2, 3, 5, 8, 13 and 21.

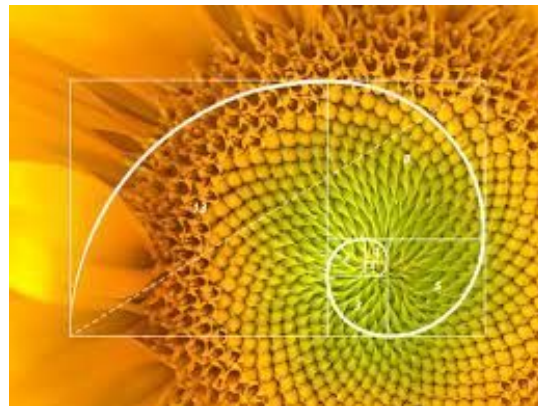


The Fibonacci spiral: an approximation of the **golden spiral** created by drawing **circular arcs** connecting the opposite corners of squares in the Fibonacci tiling; (see preceding image)

Fibonacci Spiral



$$\varphi = \frac{1 + \sqrt{5}}{2} = 1.618\ 033\ 988\ 749\ldots$$



Example

Write a nested for-loop that prints out a 3 by 3 grid of letters where the letters in the first column are A's, the letters in the second column are B's, and the letters in the third column are C's.

```
A B C  
A B C  
A B C
```

Example

```
for (int row = 0; row < 3; row++) {  
    for(int col = 0; col < 3; col++) {  
        if (col == 0) {  
            System.out.print("A");  
        } else if (col == 1) {  
            System.out.print("B");  
        } else {  
            System.out.println("C");  
        }  
    }  
}
```

Exercise 1

Write a program that asks the user for an integer N . Using nested for loops, print the numbers $\leq n$ for each n in $1, \dots, N$. For $N \geq 4$, your first few lines of output should be:

1

1 2

1 2 3

1 2 3 4

...

Exercise 2

Write a program that asks the user for a double R . Use nested for loops to write a program that counts the number of integer-valued points (e.g. $(1, 3)$) on or inside a circle of radius R .

Hint: If given a circle of radius R , how would you construct a rectangular “search area” that will always contain the entire circle? How can you check if a given integer-valued point is on or inside the circle?

Hint: $0 < R < 1$ should give you exactly 1 such point, e.g. $(0, 0)$. $R = 1$ should give you 5 points $[(0, 0), (0, 1), (1, 0), (-1, 0), (0, -1)]$ and $R = \sqrt{2}$ should give you 9 points.

How many points will $R = 3$ give you?

Exercise 3

Write a program that asks the user for a integer N. Print all of the **prime** factors of N.

Hint: First identify if each number is a factor of N. If it, use another for loop to determine if the factor is prime.

Why is this algorithm inefficient? At the end of your program, write a comment explaining how you might make it faster (concept only--no code required).

Exercise 1 Answer

```
public class Diagonal {  
    public static void main(String[] args){  
        // excluding scanner for simplicity  
        int N = 5;  
        for (int i = 1; i <= N; i++) {  
            for (int j = 1; j <= i; j++) {  
                System.out.print(j + " ");  
            }  
            System.out.println();  
        }  
    }  
}
```

Exercise 2 Answer

```
import java.lang.Math;

public class CountLatticePoints {

    public static void main(String[] args) {

        // excluding scanner for simplicity

        double r = Math.sqrt(2);

        int s = (int) r + 1;

        int count = 0;

        for (int i = -s; i <= s; i++)

            for (int j = -s; j <= s; j++)

                if (Math.sqrt(i*i + j*j) <= r)

                    count++;

        System.out.print(count);

    }

}
```


Warm Up!

Write a program that takes in a `String` and prints out the `String` without vowels.

Example:

“I love computer science!” -> “I lv cmptr scnc!”