Civilization advances by extending the number of operations we can perform without thinking about them.

-Alfred North Whitehead
Learn How To:

1. Write a method that takes in input
2. Write a method that gives back output
3. Trace method calls using stacks
Calling Methods

turnRight();

move();    readInt("Int please!");

println("hello world");

rect.setFilled(true);

drawRobotFace();

add(rect);

preventGlobalWarming();
Defining a Method

```java
private void turnRight() {
    turnLeft();
    turnLeft();
    turnLeft();
}
```

Big difference with Java methods:
Java methods can **take in data**, and can **return data**!
Toasters are Methods

For example: runToaster

- Thanks Mehran
Toasters are Methods

parameter
Toasters are Methods

parameter
Toasters are Methods
Toasters are Methods
Toasters are Methods

Piech, CS106A, Stanford University
Toasters are Methods
Toasters are Methods

* You don’t need a second toaster if you want to toast bagels. Use the same one.
Toasters are Methods
Toasters are Methods
Toasters are Methods
Toasters are Methods
Toasters are Methods
Methods are Like Toasters
Methods are Like Toasters
Methods are Like Toasters
Methods are Like Toasters
Methods are Like Toasters

parameter(s) ➔ return
Formally

\[
\text{visibility type nameOfMethod}(\text{parameters}) \begin{array}{l}
\text{statements} \\
\end{array} \\
\}
\]

- **visibility**: usually private or public
- **type**: type returned by method (e.g., int, double, etc.)
  - Can be **void** to indicate that nothing is returned
- **parameters**: information passed into method
Anatomy of a method

```java
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}

private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}

private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}

private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
Anatomy of a method

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}

private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```
Anatomy of a method

```java
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}

private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}

private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
**Anatomy of a method**

```
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}

private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;  // Ends the method and gives back a single value
}
```
Anatomy of a method

```java
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}
```

```java
private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```

This statement is necessary because `average` promised to return a double.
Anatomy of a method

```java
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}

private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
```

method “call”
Anatomy of a method

<table>
<thead>
<tr>
<th>Return Type</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public</strong> void run() {</td>
<td></td>
</tr>
<tr>
<td>double mid = average(5.0, 10.2);</td>
<td></td>
</tr>
<tr>
<td>println(mid);</td>
<td></td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

private double average(double a, double b) { 
  double sum = a + b; 
  return sum / 2; 
}
public void run() {
    double mid = average(5.0, 10.2);
    println(mid);
}

private double average(double a, double b) {
    double sum = a + b;
    return sum / 2;
}
Parameters

Parameters let you provide a method some information when you are calling it.
Learn by Example
private void printIntro() {
    println("Welcome to class");
    println("It's the best part of my day.");
}

public void run() {
    printIntro();
}
private double metersToCm(double meters) {
    return 100 * meters;
}

public void run() {
    double result = metersToCm(5.2);
    println(result);
}
private double metersToCm(double meters) {
    return 100 * meters;
}

public void run() {
    println(metersToCm(5.2));
    println(metersToCm(9.1));
}
private void printOpinion(int num) {
    if (num == 5) {
        println("I love 5!");
    } else {
        println("Whattever");
    }
}

public void run() {
    printOpinion(5);
}
Multiple Return Statements

* Note: typo in lecture. Used to have a void, should have an int

```java
private int max(int num1, int num2) {
    if(num1 >= num2) {
        return num1;
    }
    return num2;
}

public void run() {
    int larger = max(5, 1);
}
```

* Note: typo in lecture. Used to have a void, should have an int
Method for Weight on Moon

* Your weight on the moon is 16.5% your weight on the earth
Passing in Classes
public class FactorialExample extends ConsoleProgram {

    private static final int MAX_NUM = 4;

    public void run() {
        for (int i = 0; i < MAX_NUM; i++) {
            println(i + "! = " + factorial(i));
        }
    }

    private int factorial(int n) {
        int result = 1;
        for (int i = 1; i <= n; i++) {
            result *= i;
        }
        return result;
    }
}

A Full Program
public class FactorialExample extends ConsoleProgram {

    private static final int MAX_NUM = 4;

    public void run() {
        for(int i = 0; i < MAX_NUM; i++) {
            println(i + "! = " + factorial(i));
        }
    }

    private int factorial(int n) {
        int result = 1;
        for (int i = 1; i <= n; i++) {
            result *= i;
        }
        return result;
    }
}
Understand the Mechanism
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
public void run() {
    for (int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
public void run() {
    for (int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

i = 0
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

| n  | 0 | result | 1 | i  | 1 |
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```java
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

<table>
<thead>
<tr>
<th>n</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>result</td>
<td>1</td>
</tr>
<tr>
<td>i</td>
<td>1</td>
</tr>
</tbody>
</table>
```java
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        System.out.println(i + "! = " + factorial(i));
    }
}
```

```
$\begin{array}{c}
\text{i} \\
1 \\
0 \\
\end{array}$
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
public void run() {
    for(int i = 0; i < MAX NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
public void run() {
    for(int i = 0; i < MAX NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
```java
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

0! = 1
public void run() {
for(int i = 0; i < MAX_NUM; i++) {
    println(i + "! = " + factorial(i));
}
}

private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

0! = 1
```java
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

```
0! = 1
```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

0! = 1
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

0! = 1
```java
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

0! = 1
```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

0! = 1
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

0! = 1
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
1! = 1
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
1! = 1
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
1! = 1
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
1! = 1
```java
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

```plaintext
0! = 1
1! = 1
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
1! = 1
2! = 2
```java
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

```
0! = 1
1! = 1
2! = 2
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
1! = 1
2! = 2
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
1! = 1
2! = 2
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
1! = 1
2! = 2
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
1! = 1
2! = 2
public void run() {
    for (int i = 0; i < MAX_NUM; i++) {
        System.out.println(i + "! = " + factorial(i));
    }
}

0! = 1
1! = 1
2! = 2
3! = 6
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
1! = 1
2! = 2
3! = 6
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
1! = 1
2! = 2
3! = 6
Parameters

Every time a method is called, new memory is created for the call.
/ NOTE: This program is buggy!!

private void addFive(int x) {
    x += 5;
}

public void run() {
    int x = 3;
    addFive(x);
    println("x = " + x);
}
Good Times With Methods

// NOTE: This program is feeling just fine...

private int addFive(int x) {
    x += 5;
    return x;
}

public void run() {
    int x = 3;
    x = addFive(x);
    println("x = " + x);
}
For primitives: Variables are not passed when you use parameters. Values are passed.
Pass by “Value”
More Examples
private void run() {
    int num = 5;
    cow(num);
}

private void cow(int grass) {
    println(grass);
}
private void run() {
    int num = 5;
    cow();
    println(num);
}

private void cow() {
    int num = 10;
    println(num);
}
Methods Called on Objects

GRect rect = new GRect(20, 20);
rect.setColor(Color.Blue);

* We will talk about how to define these later in the class
private void run() {
    println("hello world");
    private void sayGoodbye() {
        println("goodbye!");
    }
}

Illegal modifier for parameter goodbye, only final is permitted

Huh?!!
private void run() {
    println("hello world");
    sayGoodbye();
}

private void sayGoodbye() {
    println("goodbye!");
}
Remember Booleans?
```java
boolean karelIsAwesome = true;
boolean myBool = 1 < 2;
```
Boolean Operations

```java
boolean a = true;

boolean b = false;

boolean and = a && b;

boolean or = a || b;

boolean not = !a;
```
private void run() {
    for(int i = 1; i <= 100; i++) {
        if(isDivisibleBy(i, 7)) {
            println(i);
        }
    }
}
private void run() {
    for(int i = 1; i <= 100; i++) {
        if(isDivisibleBy(i, 7)) {
            println(i);
        }
    }
}

private boolean isDivisibleBy(int a, int b) {
    if((a % b) == 0) {
        return true;
    } else {
        return false;
    }
}
Boolean Return

private void run() {
    for(int i = 1; i <= 100; i++) {
        if(isDivisibleBy(i, 7)) {
            println(i);
        }
    }
}

private boolean isDivisibleBy(int a, int b) {
    return a % b == 0;
}
Learn How To:

1. Write a method that takes in input
2. Write a method that gives back output
3. Trace method calls using stacks
Extra Exercise

• Greek mathematicians took a special interest in numbers that are equal to the sum of their proper divisors (a proper divisor of \( n \) is any divisor less than \( n \) itself). They called such numbers **perfect numbers**. For example, 6 is a perfect number because it is the sum of 1, 2, and 3, which are the integers less than 6 that divide evenly into 6. Similarly, 28 is a perfect number because it is the sum of 1, 2, 4, 7, and 14.

• Design and implement a Java program that finds all the perfect numbers between two limits. For example, if the limits are 1 and 10000, the output should look like this:

```
The perfect numbers between 1 and 10000 are:
6
28
496
8128
```