# Methods

# Methods

Methods are a **named set of instructions**

- calculating a person's age (given their birthday and today's date)
- determining the number of ingredients in a recipe (given a list of ingredients)

Allows us to just say what we want without describing how to calculate it

- when we ask a friend to determine someone's age, they know how to do the calculation

# Some Methods Seen Thus Far

**Scanner** [`nextLine()` `nextInt()`]: read in this type of value from the user

**DecimalFormat** [`format()`]: reformat this number according to some pattern

**Math** [`abs()` `pow()`]: perform this operation and give us the result

**String** [`substring()` `toCharArray()`]: manipulate the String and give us the result

# Example: `toCharArray()`

If we want to convert a String to an array, we write the code ourselves...

```
String exampleStr = "Hi!";

char[] arr = new char[exampleStr.length()];

for (int i = 0; i < arr.length; i++) {
    arr[i] = exampleStr.charAt(i);
}
```

# Example: `toCharArray()`

If we want to convert a String to an array, we write the code ourselves...

...or we can call a method (Java's named instruction for this task)

```
String exampleStr = "Hi!";

char[] arr = exampleStr.toCharArray();
```

# Method Basics

Methods allow us to define a **named set of instructions**

Components

- **method name** (a valid *identifier*)
- **parameters**: what info do we need to run the method
- **returned value**: what (if anything) will the method give us back (i.e., *return*)?

# Example: Calculating Age

Write out the code for calculating age (ignoring the method component)

- assume variables `bYear`, `bMonth`, `bDay`, `tYear`, `tMonth`, `tDay`
  - where the b = birthday and t = today

Then consider...

- what might you call this set of instructions?
- what info do you need from the user to run these instructions? what are the datatypes of this info?
- what value (if anything) do you expect to get back? what is the datatype of this value?

# Anatomy of a Method

Methods are defined **outside** of `main`, but **inside** the class

For now, we want to define the name, parameters, return type

```
public static int calculateAge(int bYear, int bMonth, int bDay, int tYear, int tMonth, int tDay) {

    int age = tYear - bYear;

    if(bMonth > tMonth || (bMonth == tMonth && bDay > tDay)) {
      age--;
    }

    return age;
}
```

# Example: Name (Identifier)

Follows same rules as other identifiers (e.g., variables)

Should follow same conventions as variable names

```
public              calculateAge









}
```

# Example: Parameters

Comma-separated list of datatype/variable name pairs

Scope is only the method they are in!

```
public                          (int bYear, int bMonth, int bDay, int tYear, int tMonth, int tDay) {




}
```

# Example: Return

Many (but not all!) methods will *return* a value

Method must a) define the return type, b) specify what value will be returned

```
public        int


    return age;
}
```

# Example: Method Signature

Methods define a name, parameters, and what is returned

All of these make up the *method signature* (also called *method header*)

```
public static int calculateAge(int bYear, int bMonth, int bDay, int tYear, int tMonth, int tDay)









}
```

# Method Execution

```
public static void main(String[] args) {

    int myAge = calculateAge(1997, 10, 19, 2017, 10, 31);

}
```

```
public static int calculateAge(int bYear, int bMonth, int bDay, int tYear, int tMonth, int tDay) {

    int age = tYear - bYear;

    if(bMonth > tMonth || (bMonth == tMonth && bDay > tDay)) {
      age--;
    }

    return age;
}
```

# Method Execution

```
public static void main(String[] args) {

   > int myAge = calculateAge(1997, 10, 19, 2017, 10, 31);

}
```

```
public




}
```

# Method Execution

```
public

   >            calculateAge(1997, 10, 19, 2017, 10, 31);

}
```

```
                                  1997,         10,       19,      2017,         10,       31
public static int calculateAge(int bYear, int bMonth, int bDay, int tYear, int tMonth, int tDay) {

    int age = tYear - bYear;

    if(bMonth > tMonth || (bMonth == tMonth && bDay > tDay)) {
      age--;
    }

    return age;
}
```

# Method Execution

```
public

   >            calculateAge(1997, 10, 19, 2017, 10, 31);

}
```

memory

| age (int) |
| --- |
| 20 |

```
                                    1997,         10,       19,      2017,         10,       31
public static int calculateAge(int bYear, int bMonth, int bDay, int tYear, int tMonth, int tDay) {

   > int age = tYear - bYear;

   > if(bMonth > tMonth || (bMonth == tMonth && bDay > tDay)) {
       age--;
     }

   > return age;
}
```

# Method Execution

```
public

  >             calculateAge(1997, 10, 19, 2017, 10, 31);

}
```

```
public









}
```

# Method Execution

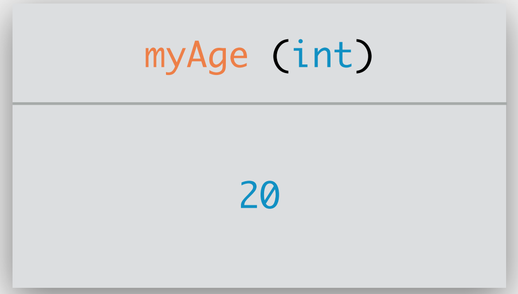
```
public

   >                calculateAge(1997, 12, 19, 2017, 10, 31);

}
```

```
public




}
```

# Method Execution

```
public static void main(String[] args) {

    int myAge = calculateAge(1997, 10, 19, 2017, 10, 31);
  >
}
```

memory




```
public




}
```

# Why Methods?

Allow us to write code once, reuse multiple places

What are the advantages of this?

# Why Methods?

Allow us to write code once, reuse multiple places

What are the advantages of this?

- need to change how a calculation is done? will only change in one place
- won't make mistakes when copying/pasting code
- concise
- code will read more English-like
  - e.g., when looping through an array, the method call clearly shows we are calculating the age of each student

# Parameters vs Arguments

*parameters* are the variables the method requires as input

- this is what goes in-between the parentheses in the method signature

*arguments* are the values given to the method

- they are *passed in*
- this is what goes in-between the parentheses in the method call
- these are what's assigned to the parameters

# Parameters vs Arguments

```
public static void main(String[] args) {

    int myAge = calculateAge(1997, 10, 19, 2017, 10, 31);

}
```

arguments

parameters

```
public static int calculateAge(int bYear, int bMonth, int bDay, int tYear, int tMonth, int tDay) {

    int age = tYear - bYear;

    if(bMonth > tMonth || (bMonth == tMonth && bDay > tDay)) {
      age--;
    }

    return age;
}
```

# Scope

Parameters only have scope in their method

Variables only have scope in the method they are declared in

- e.g., variables declared in other methods cannot be used in main

# Return

Methods with a return type must always have at least one return statement

- usually, but not always, at the bottom

Code after an executed return will never be executed

```
public static int evenOrOdd(int num) {
    if (num % 2 == 0) {
      return true;
    } else {
      return false;
    }
}
```

```
public static int evenOrOdd(int num) {
    if (num % 2 == 0) {
      return true;
    }

    return false;
}
```

# Void Methods

Some methods don't return anything

- we call these *void* methods
- return statements are unnecessary

Return type is replaced with the keyword `void`

Useful for printing

# Anatomy of a Method

We'll defer discussion of `public` and `static`

for now, always use them

```
public static int calculateAge(int bYear, int bMonth, int bDay, int tYear, int tMonth, int tDay) {

    int age = tYear - bYear;

    if(bMonth > tMonth || (bMonth == tMonth && bDay > tDay)) {
      age--;
    }

    return age;
}
```

# Are Arguments Modified?

Primitive type?

- modifications made in a method **do not** affect the original variable

Array or a class type?

- modifications made in a method **do** affect the original variable
- we'll see more of this with classes

# The `main` Method

A method...similar to all the other methods!

This is the method Java calls to start your program

`String[] args` is an array of String arguments

- options for how to run your program
- e.g., debugging, what to do with output
- can see this by running programs on the command line