Creating Methods in C#

Why do we need methods?

- Long sequences of lines of code are like a run on sentence in prose.
- Leads to spaghetti  code.
- Organized code is easier to debug.
- Organized code is easier to change.

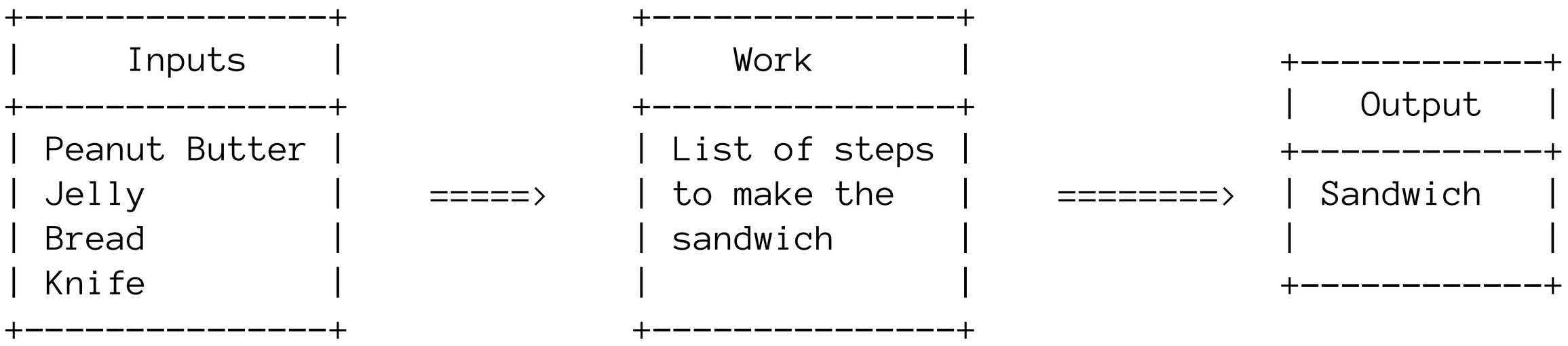
Don't Repeat Yourself

D

R

Y

Input - Work - Output


Inputs
Peanut Butter
Jelly
Bread
Knife
=====>
Work
List of steps
to make the
sandwich
========>
Output
Sandwich

Specifying a method

- The method name
- The **inputs** - known as `arguments`
- The **work** - or `body` of the method
- The **output** - or `return` type/value of the method

Let's create a method

```
using System;

namespace EmployeeDatabase
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("--------------------------------");
      Console.WriteLine("Welcome to Our Employee Database");
      Console.WriteLine("--------------------------------");
      Console.WriteLine();
      Console.WriteLine();

      Console.Write("What is your name? ");
      var name = Console.ReadLine();

      Console.Write("What is your department number? ");
      var department = int.Parse(Console.ReadLine());

      Console.Write("What is your yearly salary (in dollars)? ");
      var salary = int.Parse(Console.ReadLine());

      var salaryPerMonth = salary / 12;
      Console.WriteLine($"Hello, {name} you make {salaryPerMonth} a month.");
    }
  }
}
```

Many lines for a greeting

```
using System;

namespace EmployeeDatabase
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("--------------------------------");
      Console.WriteLine("Welcome to Our Employee Database");
      Console.WriteLine("--------------------------------");
      Console.WriteLine();
      Console.WriteLine();

      Console.Write("What is your name? ");
      var name = Console.ReadLine();

      Console.Write("What is your department number? ");
      var department = int.Parse(Console.ReadLine());

      Console.Write("What is your yearly salary (in dollars)? ");
      var salary = int.Parse(Console.ReadLine());

      var salaryPerMonth = salary / 12;
      Console.WriteLine($"You make {salaryPerMonth} a month.");
    }
  }
}
```

Ask for data three times

- Once for `strings`
- Twice for `ints`

```
using System;

namespace EmployeeDatabase
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("---------------------------------");
      Console.WriteLine("Welcome to Our Employee Database");
      Console.WriteLine("---------------------------------");
      Console.WriteLine();
      Console.WriteLine();

      Console.Write("What is your name? ");
      var name = Console.ReadLine();

      Console.Write("What is your department number? ");
      var department = int.Parse(Console.ReadLine());

      Console.Write("What is your yearly salary (in dollars)? ");
      var salary = int.Parse(Console.ReadLine());

      var salaryPerMonth = salary / 12;
      Console.WriteLine($"You make {salaryPerMonth} a month.");
    }
  }
}
```

Business Logic

- Computing monthly salary

```
using System;

namespace EmployeeDatabase
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("---------------------------------");
      Console.WriteLine("Welcome to Our Employee Database");
      Console.WriteLine("---------------------------------");
      Console.WriteLine();
      Console.WriteLine();

      Console.Write("What is your name? ");
      var name = Console.ReadLine();

      Console.Write("What is your department number? ");
      var department = int.Parse(Console.ReadLine());

      Console.Write("What is your yearly salary (in dollars)? ");
      var salary = int.Parse(Console.ReadLine());

      var salaryPerMonth = salary / 12;
      Console.WriteLine($"You make {salaryPerMonth} a month.");
    }
  }
}
```

Define our first method

Display the greeting

Step 1: Name the method

We need to come up with a good name for this method. Since it will *display the greeting* a good name is `DisplayGreeting` or `DisplayTheGreeting`

---

[1] There are two hard things in computer science:
1. Cache invalidation
2. Naming things
3. Off-by-one errors

Conventions

Notice we mashed all the words together and Capitalized Each Word.

This is known as: **Pascal Case**.

This is the convention for method names in C# and isn't a technical requirement.

Step 2: Do we need any input to do the work?

Not for this. We can display the greeting without any extra information.

Step 3: Does this method *return* anything to the code that calls it?

In this case, no. We **do** output to the console, but we don't send anything back.

Putting it all together

Name	`DisplayGreeting`
Input	`None`
Work	`Print greeting to the console`
Output	`None`

Signature

```
// static method (ignore this for the moment)
//  |
//  |  The return (output) type. Here there is none
//  |  since the method isn't giving anything back
//  |  to the code that called it.
//  |   |
//  |   |                The inputs, known as arguments. None in this case.
//  |   |                    |
//  |   |                    |
//  v   v                    v
static void DisplayGreeting()
```

The whole thing

```
// static method (ignore this for the moment)
//  |
//  |  The return (output) type. Here there is none
//  |  since the method isn't giving anything back
//  |  to the code that called it.
//  |   |
//  |   |                The inputs, known as arguments. None in this case.
//  |   |                    |
//  |   |                    |
//  v   v                    v
static void DisplayGreeting()
{
    //
    //    Body of the method
    //      |
    //      |
    //      v
    Console.WriteLine("--------------------------------");
    Console.WriteLine("Welcome to Our Employee Database");
    Console.WriteLine("--------------------------------");
    Console.WriteLine();
    Console.WriteLine();
}
```

Here is the method in our app

```
using System;

namespace EmployeeDatabase
{
  class Program
  {

    static void DisplayGreeting()
    {
      Console.WriteLine("--------------------------------");
      Console.WriteLine("Welcome to Our Employee Database");
      Console.WriteLine("--------------------------------");
      Console.WriteLine();
      Console.WriteLine();
    }

    static void Main(string[] args)
    {
      DisplayGreeting();

      Console.Write("What is your name? ");
      var name = Console.ReadLine();

      Console.Write("What is your department number? ");
      int department = int.Parse(Console.ReadLine());

      Console.Write("What is your yearly salary (in dollars)? ");
      int salary = int.Parse(Console.ReadLine());

      Console.WriteLine($"You make {salary / 12} dollars a month.");
    }
  }
}
```

Using (`calling`) the method

```
using System;

namespace EmployeeDatabase
{
  class Program
  {

    static void DisplayGreeting()
    {
      Console.WriteLine("--------------------------------");
      Console.WriteLine("Welcome to Our Employee Database");
      Console.WriteLine("--------------------------------");
      Console.WriteLine();
      Console.WriteLine();
    }

    static void Main(string[] args)
    {
      DisplayGreeting();

      Console.Write("What is your name? ");
      var name = Console.ReadLine();

      Console.Write("What is your department number? ");
      int department = int.Parse(Console.ReadLine());

      Console.Write("What is your yearly salary (in dollars)? ");
      int salary = int.Parse(Console.ReadLine());

      Console.WriteLine($"You make {salary / 12} dollars a month.");
    }
  }
}
```

Using the Method

When we call the method we have to include parenthesis even if, as in this case, there aren't any *arguments*.

The way you read the line is "Call the DisplayGreeting method, providing no arguments, and expecting no return."

```
//   Name of the method
//    |
//    |            Any input VALUES or arguments would go here
//    |             |
//    |             |
//    v             v
     DisplayGreeting();
```

Increased expressiveness

Just reading this code we've added to the *expressiveness* of our code.

As a reader of the code I may not care **how** the `DisplayGreeting` works, but I do know that it will show some form of greeting to the user.

In doing so we've **reduced the amount of code** the reader has to visually concern themselves with **while retaining the meaning** of the code.

Methods that take input and return output

Prompting for a string

- Name: `PromptForString`
- Input: The text of the prompt, as a `string`
- Work: Show the user the prompt - Wait for their response
- Output: The user's response, as a `string`.

Our method signature will be:

```
// static method (ignore this for the moment)
//   |
//   |  The return (output) type. This says that
//   |  we expect this method to return a single
//   |  string to the code that called it
//   |    |
//   |    |                  The inputs, known as arguments.
//   |    |                  In this case a single string
//   |    |                  in a variable known as `prompt`
//   |    |                         |
//   |    |                         |
//   v    v                         v
static string PromptForString(string prompt)
```

Calling a method that requires arguments and returns a value

- Still uses method name.
- Do something with the returned value. Store this in a variable.
- Supply value for the input(s).

```
// var declaration. Since C# knows this method returns a string
// our `answer` variable will be a string. (Type inference)
//  |
//  |  Name of the output variable
//  |   |
//  |   |         Name of the method
//  |   |           |
//  |   |           |             Argument value
//  |   |           |              |
//  |   |           |              |
//  v   v           v              v
   var answer = PromptForString("What is your name? ");
//        ^                         |
//        |                         |
//        |                         v
//        ^     PromptForString(string prompt)
//        ^                +--------------------+
//        |                |                    |
//        |                | ** WORK HAPPENS ** |
//        +--<< output ---|                    |
//                         +--------------------+
//
```

Writing the body of the method

Use the argument (what the caller sent us)

```
static string PromptForString(string prompt)
{
  // Use the argument, whatever the caller sent us.
  Console.Write(prompt);

  // Get some user input
  var userInput = Console.ReadLine();

  // RETURN that value as the output of this method.
  // The value in `userInput` will go wherever the
  // *CALLER* of the method has specified.
  return userInput;
}
```

Writing the body of the method

Get the user input.

```
static string PromptForString(string prompt)
{
  // Use the argument, whatever the caller sent us.
  Console.Write(prompt);

  // Get some user input
  var userInput = Console.ReadLine();

  // RETURN that value as the output of this method.
  // The value in `userInput` will go wherever the
  // *CALLER* of the method has specified.
  return userInput;
}
```

Writing the body of the method

Return the user's input

```
static string PromptForString(string prompt)
{
  // Use the argument, whatever the caller sent us.
  Console.Write(prompt);

  // Get some user input
  var userInput = Console.ReadLine();

  // RETURN that value as the output of this method.
  // The value in `userInput` will go wherever the
  // *CALLER* of the method has specified.
  return userInput;
}
```

```
static void DisplayGreeting()
{
  Console.WriteLine("----------------------------------------");
  Console.WriteLine("    Welcome to Our Employee Database    ");
  Console.WriteLine("----------------------------------------");
  Console.WriteLine();
  Console.WriteLine();
}

static string PromptForString(string prompt)
{
  Console.Write(prompt);
  var userInput = Console.ReadLine();

  return userInput;
}

static void Main(string[] args)
{
  DisplayGreeting();

  var name = PromptForString("What is your name? ");

  Console.Write("What is your department number? ");
  int department = int.Parse(Console.ReadLine());

  Console.Write("What is your yearly salary (in dollars)? ");
  int salary = int.Parse(Console.ReadLine());

  Console.WriteLine($"Hello, {name} you make {salary / 12} dollars per month.");
}
```

Revealing intent

Our main code is now shorter, and clearer. The line

```
var name = PromptForString("What is your name? ");
```

describes its entire **intent** without having to detail **how** it is done.

Do you think you could write the method to prompt for integers?

```
static void DisplayGreeting()
{
  Console.WriteLine("----------------------------------------");
  Console.WriteLine("    Welcome to Our Employee Database    ");
  Console.WriteLine("----------------------------------------");
  Console.WriteLine();
  Console.WriteLine();
}

static string PromptForString(string prompt)
{
  Console.Write(prompt);
  var userInput = Console.ReadLine();

  return userInput;
}

static int PromptForInteger(string prompt)
{
  Console.Write(prompt);
  var userInput = int.Parse(Console.ReadLine());

  return userInput;
}

static void Main(string[] args)
{
  DisplayGreeting();

  var name = PromptForString("What is your name? ");

  int department = PromptForInteger("What is your department number? ");

  int salary = PromptForInteger("What is your yearly salary (in dollars)? ");

  Console.WriteLine($"Hello, {name} you make {salary / 12} dollars per month.");
}
```

Encapsulation

We have been `DRY`ing up our code.

There is *ONE* place in our code where we prompt for strings, and *ONE* place where we prompt for integers.

Let's try to run this code and type something **other** than a number for a department or salary.

Different way to parse.

int.TryParse

This method behaves slightly different than `Int.Parse`. It *returns* a boolean value that indicates if the value was parsed, and we place the variable we are assigning as an *argument*. It looks like this:

```
int userInput;
var isThisGoodInput = int.TryParse(Console.ReadLine(), out userInput);
```

The code is a little more complex, but it allows us to do some checking.

Let's see how this might work in our method.

```
static int PromptForInteger(string prompt)
{
    Console.Write(prompt);

    int userInput;
    var isThisGoodInput = int.TryParse(Console.ReadLine(), out userInput);

    if (isThisGoodInput)
    {
        return userInput;
    }
    else
    {
        Console.WriteLine("Sorry, that isn't a valid input, I'm using 0 as your answer.");
        return 0;
    }
}
```

Conclusion

- Methods are a fundamental part of many programming languages
- Style and structure may vary You'll see when we get to JavaScript
- Increased organization == easier to understand + easier to improve