# Week 8: Computer Science 1

## Methods

# Methods

We have been using one method since the first week of class:

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

This is the method that is called when a Java program is run. We have been limited to this method so far, but we can create our own methods.

When you create a method, you are creating a block of code that can be called from other parts of your program. This allows you to reuse code and make your program more modular.

Methods are also referred to as functions. You have come into contact with functions in math class. For example, the function **f(x) = x*x** is a function that takes in a number and returns the square of that number. For example, **f(3) = 9**.

A function takes an input, performs some operations, and returns an output. That same concept applies to methods in programming.

If I wanted to create a method to perform the operation **f(x) = x*x** in Java, I would write the following code:

```
public static int add(int x) {
    return x * x;
}
```

This method takes in an integer `x` and returns the square of `x`.

Let's break down the syntax of this method.

# Method Syntax

```
public static returnType methodName(parameters) {
    // code
}
```

- `public` is an access modifier. It means that the method can be accessed from anywhere. Methods can also be `private` or have no access modifier. `private` means that the method can only be accessed from within the class.
- `static` means that the method belongs to the class, not an instance of the class. This will be explained in more detail later.
- `returnType` is the type of the value that the method returns. If the method does not return a value, the return type is `void`.
- `methodName` is the name of the method.
- `parameters` are the inputs to the method. If the method does not take any inputs, the parentheses are empty.

## Square Method Breakdown

```
public static int square(int x) {
    return x * x;
}
```

- `public` is an access modifier. It means that the method can be accessed from anywhere.
- `static` means that the method belongs to the class, not an instance of the class.
- `int` is the return type of the method. The method returns an integer. A method can only have one return type. Along with `int`, the return type can be any data type, or `void` if the method does not return a value.
- `square` is the name of the method.
- `int x` is the parameter of the method. The method takes in an integer `x`.
- `return x * x` is the body of the method. It returns the square of `x`. Since the return type is `int`, the method must return an integer.

## Access Modifiers

```
public static returnType methodName(parameters) {
    // code
}
```

- `public` - The method can be accessed from anywhere.
- `private` - The method can only be accessed from within the class.

When you create a method, you should think about who needs to access the method. If the method is only used within the class, it should be `private`. If the method is used outside of the class, it should be `public`.

So far, we have only been using one class, so we have not had to worry about access modifiers. As we start to create more classes, we will need to think about access modifiers.

## Static

```
public static returnType methodName(parameters) {
    // code
}
```

- `static` means that the method belongs to the class, not an instance of the class.

When you create an object from a class, you are creating an instance of the class. For example, if you have a class `Car`, you can create an instance of the class by writing `Car myCar = new Car();`. `myCar` is an instance of the class `Car`.

When you create a method, you can access the method from an instance of the class. For example, if you have a method `drive` in the `Car` class, you can call the method by writing `myCar.drive();`.

If the method is `static`, you can access the method from the class itself. For example, if you have a method `drive` in the `Car` class, you can call the method by writing `Car.drive();`. We will see more examples of this when we create classes.

## Return Type

```
public static returnType methodName(parameters) {
    // code
}
```

The return type of a method is the type of the value that the method returns. If the method does not return a value, the return type is `void`.

If the return type is not `void`, the method must return a value of the specified type. If the method does not return a value, the method will not compile.

The return type can be any data type, such as `int`, `double`, `String`.

## Method Name

```
public static returnType methodName(parameters) {
    // code
}
```

The method name is the name of the method. The method name can be any valid identifier. The method name should be descriptive of what the method does.

Method names should start with a lowercase letter and use camel case or snake case. For example, `calculateArea` is a good method name.

## Parameters

```
public static returnType methodName(parameters) {
    // code
}
```

Parameters are the inputs to the method. If the method does not take any inputs, the parentheses are empty. For example, `public static void printHello()` takes no parameters.

In the method `public static int square(int x)`, `int x` is the parameter. The type of the parameter is `int`, and the name of the parameter is `x`.

If the method takes multiple parameters, they are separated by commas. For example, `public static int add(int x, int y)` takes two parameters, `x` and `y`.

A method can also take parameters of different types. For example, `public static double calculateArea(int length, float width)` takes two parameters of different types.

## Return Statement

```
public static returnType methodName(parameters) {
    // code
    return value;
}
```

The `return` statement is used to return a value from the method. The return type of the method must match the type of the value being returned.

If the return type is `void`, the method does not return a value, and the `return` statement is not used.

The `return` statement can only be used once in a method. Once the `return` statement is executed, the method will stop executing.

## Method Example

```
public class Main {
    public static void main(String[] args) {
        int x = 5;
        int y = 10;
        int z = add(x, y);
        System.out.println(z);
    }

    public static int add(int x, int y) {
        return x + y;
    }
}
```

This program will output `15`. The method `add` takes in two integers and returns the sum of the two integers. The method is called from the `main` method, and the result is printed to the console.

Notice that the `static` method is included inside the `Main` class but outside the `main` method. This is because the `add` method is not part of the `main` method.

```
public class Main {
    public static void main(String[] args) {
        int x = 5;
        int y = 10;
        int z = add(x, y);
        System.out.println(z);
    }

    public static int add(int x, int y) {
        return x + y;
    }
}
```

We **call** the method `add` from the `main` method. When we **call** the method, we pass in the arguments `x` and `y`. The method returns the sum of `x` and `y`, and we store the result in the variable `z`.

```java
public class Main {
    public static void main(String[] args) {
        int x = 5;
        int y = 10;
        int z = add(x, y);
        System.out.println(z);
    }

    public static int add(int x, int y) {
        return x + y;
    }
}
```

What is the return type of the `add` method?

What are the parameters of the `add` method?

```java
public class Main {
    public static void main(String[] args) {
        int x = 5;
        int y = 10;
        int z = add(x, y);
        System.out.println(z);
    }

    public static int add(int x, int y) {
        return x + y;
    }
}
```

What is the return type of the `add` method? **int**

What are the parameters of the `add` method? **int**

```java
public class Main {
    public static void main(String[] args) {
        int x = 5;
        double y = 10;
        int z = add(x, y);
        System.out.println(z);
    }

    public static int add(int x, int y) {
        return x + y;
    }
}
```

**Be careful with the types of the parameters!** The types of the parameters must match the types of the arguments when the method is called. This will not compile because the type of `y` in the `main` method is `double`, but the type of `y` in the `add` method is `int`.

## Method with No Return Type

```
public class Main {
    public static void main(String[] args) {
        printHello();
    }

    public static void printHello() {
        System.out.println("Hello!");
    }
}
```

This program will output `Hello!`. The method `printHello` takes no parameters and returns no value. The method is called from the `main` method.

# Examples

Create a method that takes a number **n** and raises it to the power of 2. The method should return the result and except an integer or a double.

The function is **f(x) = x*x**.

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

        double num = square(4.2); // Pass in a double
        System.out.println(num);

    }

    public static double square(double x) { // Accept a double
        return x * x; // Return the square of x
    }
}
```

The method `square` takes in a double `x` and returns the square of `x`. Since the return type is `double`, the method must return a double. It can also accept an integer since an `int` can be cast to a `double`.

Create a method that takes three integers and returns the average of the three numbers. The method should return the result and except three integers. Print out the result in the `main` method.

The function is **f(x, y, z) = (x + y + z) / 3**.

What should the return type of the method be?

What are the parameters of the method?

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

        int x = 5;
        int y = 6;
        int z = 15;
        double avg = average(x, y, z);
        System.out.println(avg);

    }

    public static double average(int x, int y, int z) {
        return (x + y + z) / 3.0; // Return the average of x, y, and z
    }
}
```

The method `average` takes in three integers `x`, `y`, and `z` and returns the average of the three integers. Since the return type is `double`, the method must return a double.

Notice that we divide by `3.0` instead of `3`. This is because we want to return a `double` instead of an `int`.

Create a method that takes a number and returns `true` if the number is even and `false` if the number is odd. The method should return the result and except an integer.

What should the return type of the method be?

What are the parameters of the method?

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

        boolean result = isEven(5);
        System.out.println(result);

    }

    public static boolean isEven(int x) {
        if (x % 2 == 0) {
            return true;
        } else {
            return false;
        }
    }
}
```

Create a program that prompts the user for a positive integer and then prints all the numbers from 0 to the number. If the user enters a negative number, the method should print "Invalid input" and terminate.

What is the return type of the method?

What are the parameters of the method?

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a positive integer: ");
        int num = scanner.nextInt();
        printNumbers(num);

    }

    public static void printNumbers(int num) {
        if (num < 0) {
            System.out.println("Invalid input");
        } else{
            for (int i = 0; i <= num; i++) {
                System.out.println(i);
            }
        }
    }
}
```

Since we are not returning a value from the method, the return type is `void`. The method takes in an integer `num`.

Create a program that prompts the user for a `String` and an integer. The program should then print the `String` the number of times specified by the integer. If the user enters a zero or negative number, the method should print "Invalid input" and terminate.

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String str = scanner.nextLine();
        System.out.print("Enter a positive integer: ");
        int num = scanner.nextInt();
        printString(str, num);

    }

    public static void printString(String str, int num) {
        if (num <= 0) {
            System.out.println("Invalid input");
        } else {
            for (int i = 0; i < num; i++) {
                System.out.println(str);
            }
        }
    }
}
```

# Scope of a Variable in a Method

The scope of a variable is the region of the program in which the variable can be accessed. The scope of a variable is determined by where the variable is declared.

When a variable is declared inside a method, the variable can only be accessed within the method. This is called the local scope of the variable.

```
public class Main {
    public static void main(String[] args) {
        int x = 5;
        int y = 10;
        int z = add(x, y);
        System.out.println(z);
        System.out.println(sum); //Error!
    }

    public static int add(int x, int y) {
        int sum = x + y;
        return sum;
    }
}
```

The variable `sum` is declared inside the `add` method. The variable `sum` can only be accessed within the `add` method. If you try to access `sum` outside of the `add` method, the program will not compile.

# Method Overloading

**Method overloading** is a feature that allows a class to have more than one method having the same name, if their parameter lists are different.

So you can create multiple methods with the same name and the method that will be executed will depend on the arguments passed to the method.

```
public class Main {
    public static void main(String[] args) {
        int x = 5;
        int y = 10;
        int z = add(x, y);
        double a = 5.5;
        double b = 10.5;
        double c = add(a, b);
    }

    public static int add(int x, int y) {
        return x + y;
    }

    public static double add(double x, double y) {
        return x + y;
    }
}
```

There are two methods with the same name, but the parameters are different. The first method takes two integers, and the second method takes two doubles.

The version of the method that is called will depend on the arguments passed.