Week 8: Computer Science 1

Methods Continued

Methods Continued

Let's review the components and structure of a method.

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

- `public` is the access modifier
- `static` is the keyword that allows the method to be called without creating an object
- `returnType` is the data type of the value returned by the method
- `methodName` is the name of the method
- `parameters` is the list of parameters that the method accepts
- `return` is the keyword that returns a value from the method

Let's create a method that accepts two parameters or type `int` and `String` and returns both values concatenated together.

What return type should we use?

```
public static String concatenate(int number, String word) {
    return number + word;
}
```

- Remember that the return type should match the type of the value that is returned. When we concatenate an `int` and a `String`, the result is a `String`. Therefore, the return type should be `String`.

Now create a method (or methods) that accept two numbers and a word. The method should return the sum of the two numbers and print the word the number of times specified by the second number.

What if the number is an `int`? What if the number is a `double`?

```java
public static void printWord(int num1, int num2, String word) {
    for (int i = 0; i < num2; i++) {
        System.out.println(word);
    }
    return num1 + num2;
}

public static void printWord(double num1, double num2, String word) {
    for (int i = 0; i < num2; i++) {
        System.out.println(word);
    }
    return num1 + num2;
}
```

This is a good example of **method overloading**. We have two methods with the same name, but different parameter lists. One method accepts two `int` values and the other accepts two `double` values.

Method Overloading

We previously discussed the concept of method overloading. This is when you have two methods with the same name, but different parameter lists.

The parameter list can be different in the following ways:

- The number of parameters
- The type of parameters
- The order of parameters (watch out for this one!)

The type and number of parameters are what we are going to concentrate on. Stay away from changing the order of parameters. This can lead to confusion and errors.

Will the following code compile?

```
public static void printNumber(int x) {
    System.out.println(x);
}

public static void printNumber(int y) {
    System.out.println(y);
}
```

The code will not run. The methods have the same name and the same parameter list. The only difference is the name of the parameter. This will not work.

How about this code?

```
public static int printNumber(int x) {
    System.out.println(x);
}

public static void printNumber(int y) {
    System.out.println(y);
}
```

This will not run either. The methods have the same name and the same parameter list. The only difference is the return type. This will not work.

You can also create one method that accepts two `double` values and casts `int` values to `double` values.

```
public static void printWord(double num1, double num2, String word) {
    for (int i = 0; i < num2; i++) {
        System.out.println(word);
    }
    return num1 + num2;
}
```

If I call the method with two `int` values, the `int` values will be cast to `double` values and the method will execute.

```
printWord(5, 3, "Hello");
```

Method Calls

Methods are a great way to organize your code and make it more readable. They also allow you to reuse code and avoid redundancy.

We have called methods from within out `main` method, but you can also call methods from within other methods.

A method can call a method, which calls a method which,calls a method...
It's methods all the way down.

Whenever you are trying to solve a problem, you should always think about how you can break the problem down into smaller, more manageable pieces. This is a great way to approach programming problems.

For example, you might want to make a method that determines the area of a room. Then use this method to check the area of a house, then use this method to check the area of a neighborhood, etc.

```
public static double areaOfRoom(double length, double width) {
    return length * width;
}

public static double areaOfHouse(double length, double width, int numRooms) {
    return areaOfRoom(length, width) * numRooms;
}

public static double areaOfNeighborhood(double length, double width, int numHouses) {
    return areaOfHouse(length, width, numHouses) * numHouses;
}
```

Each one of these methods calls the method that is one level below it. This is a great way to organize your code and make it more readable.

It allows us to access each level of the problem separately and test each level separately.

- Easier it is to test and debug.
- Easier it is to understand and read.
- Easier it is to reuse.

Scope of Variables in Methods

Similar to the scope of variables in loops, the scope of variables in methods is limited to the method in which they are declared.

```
public static void main(String[] args) {
    int x = 5;
    System.out.println(x);
    printNumber();
    System.out.println(x);
}

public static void printNumber() {
    int x = 10;
    System.out.println(x);
}
```

In the above example, the variable `x` is declared in the `main` method and is accessible within the `main` method. The variable `x` is also declared in the `printNumber` method and is accessible within the `printNumber` method.

The same rules apply to variables that are declared within the method that also includes loops and conditional statements.

Let's look at an example of a method that accepts a number and prints all the numbers from 0 to that number.

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

What happens if we declare `int i = 0` outside of the loop?

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

The variable `i` is declared outside of the loop and is accessible within the entire method. This means that the value of `i` will be accessible after the loop has finished.

Notice that I have removed the `int` keyword from the declaration of `i` within the loop. This is because `i` has already been declared and initialized outside of the loop.

Designing Methods

When designing methods, you should think about the following:

- What is the purpose of the method?
- What are the inputs to the method?
- What is the output of the method?
- What is the name of the method?
- What is the return type of the method?
- What are the parameters of the method?
- What is the access modifier of the method?
- What is the scope of the method?

Built in Methods

We have been using methods to solve problems since you wrote your first `print` statement.

```
System.out.println("Hello, World!");
```

This is a method call. The method is `println` and it accepts a `String` parameter and returns nothing.

Since it returns nothing what is the return type?

The return type is `void`. This means that the method does not return a value.

We also used methods within the `Scanner` class.

```
Scanner input = new Scanner(System.in);
int x = input.nextInt();
```

The method is `nextInt` and it returns an `int` value.

We know the other methods of the `Scanner` class. What are they?

- `nextLine` - returns a `String` value
- `nextDouble` - returns a `double` value
- `nextBoolean` - returns a `boolean` value
- `next` - returns a `String` value

Math Class

There are other built in methods in Java that are very useful. One of the most useful classes is the `Math` class.

The `Math` class contains methods that perform basic mathematical operations.

- `abs` - returns the absolute value of a number
- `ceil` - returns the smallest integer that is greater than or equal to the number
- `floor` - returns the largest integer that is less than or equal to the number
- `max` - returns the largest of two numbers
- `min` - returns the smallest of two numbers
- `pow` - returns the value of the first argument raised to the power of the second argument
- `sqrt` - returns the square root of a number
- `random` - returns a random number between 0.0 and 1.0

Each of these methods is called using the class name `Math` followed by a period and the method name.

```
int x = Math.abs(-5); // x = 5
double y = Math.ceil(5.5); // y = 6.0
double z = Math.floor(5.5); // z = 5.0
int a = Math.max(5, 10); // a = 10
int b = Math.min(5, 10); // b = 5
double c = Math.pow(2, 3); // c = 8.0
double d = Math.sqrt(25); // d = 5.0
double e = Math.random(); // e = random number between 0.0 and 1.0
```

Let's work through an example of using the `Math` class and methods.

If you have a number that is a `double` and you want to find the distance to zero (the absolute value), you can use the `abs` method.

```
double x = -5.5;
double y = Math.abs(x); // y = 5.5
```

Create a method (or methods) that accept two `double` values, two `int` values, or a `double` and an `int`. The method should return the distance between the two values.

```
distance(5.5, 3.5) -> 2.0
distance(3, 5) -> 2.0
distance(5.5, 3) -> 2.5
distance(5, 3.5) -> 1.5
distance(-5,3) -> 8.0
```

```java
public static double distance(double x, double y) {
    return Math.abs(x - y);
}

public static double distance(double x, int y) {
    return Math.abs(x - y);
}

public static double distance(int x, double y) {
    return Math.abs(x - y);
}

public static double distance(int x, int y) {
    return Math.abs(x - y);
}
```