Java Classes, Objects and Methods

- *Class*: a blueprint for how a data structure should function
- *Constructor*: instructs the class to set up the initial state of an object
- *Object*: instance of a class that stores the state of a class
- *Method*: set of instructions that can be called on an object
- *Parameter*: values that can be specified when creating an object or calling a method
- *Return value*: specifies the data type that a method will return after it runs

Classes

One fundamental concept of object-oriented programming in Java is the *class*.

A *class* is a set of instructions that describe how a data structure should behave.

Java provides us with its own set of pre-defined classes, but we are also free to create our own custom classes:

```
class Dog { }
```

Java/C/C++/C#

Let's start by creating the starting state of our class. We can do this by adding a class *constructor* to it.

The Constructors

A constructor initializes an object when it is created. It has the same name as its class and is syntactically similar to a method. However, constructors have no explicit return type.

Typically, you will use a constructor to give initial values to the instance variables defined by the class, or to perform any other startup procedures required to create a fully formed object.

All classes have constructors, whether you define one or not, because Java automatically provides a default constructor that initializes all member variables to zero. However, once you define your own constructor, the default constructor is no longer used.

1. A class *constructor* will allow us to create `Dog` instances. With a class constructor, we can set some information about the `Dog`.
2. If we do not create a class constructor, Java provides one that does not allow you to set initial information.

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

Java/C/C++/C#

When we create a new class, we probably have specific details that we want the class to include. We save those specific details into instance variables.

Instance variables model real-world attributes, such as the age of a dog:

```
class Dog { int age; public Dog() { } }
```

Java/C/C++/C#

For the `Dog` class, we can specify the initial dog age by adding **parameters** to the class constructor.

Parameters allow data types to be created with specified attributes.

```
class Dog { int age; public Dog(int dogsAge) { age = dogsAge;
}
```

Java/C/C++/C#

You may have noticed a mysterious looking line of code in previous lessons that looks like this:

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

Java/C/C++/C#

This is Java's built-in `main` method. When Java runs your program, the code inside of the main method is executed.

Objects

To use the Dog class, we must create an instance of the Dog class. An

instance of a class is known as an object in Java. Here's how to create a new instance of Dog within the ***main*** method:

```
public static void main(String[] args) { Dog spike = new Dog(3); }
```

JavaScript

With this, we created a new Dog class object named *spike* and added the *age* parameter. We set the age to be 3.

Methods

A method is a pre-defined set of instructions. **Methods are declared within a class**. Java provides some pre-defined methods available to all classes, but we can create our own as well:

```
public void bark() { System.out.println("Woof!"); }
```

Java/C/C++/C#

Now the bark method is available to use on the spike object. We can do this by calling the method on our object. **This occurs in the main method.**

```
public static void main(String[] args) { Dog spike = new Dog(3); spike.bark(); }
```

Java/C/C++/C#

Similar to constructors, you can customize methods to accept parameters.

```
// When creating the method public void run(int feet) { System.out.println("Your dog ran " + feet + " feet!"); } // Calling the method on the object public static void main(String[] args) { Dog spike = new Dog(3); spike.bark(); spike.run(42); }
```

Java/C/C++/C#

The `void` keyword indicates that no value should be returned by the method after it executes all the logic in the method. If we do want the method to return a value after it finishes running, we can specify the *return type*.

1. The `void` keyword (which means "completely empty") indicates to the method that no value is returned after calling that method.

2. Alternatively, we can use data type keywords (such as `int`, `char`, etc.) to specify that a method should return a value of that type

```
// Creating the method public int getAge() { return age; } //
Calling method and setting return value to variable int
spikeAge = spike.getAge();
```

Java/C/C++/C#

Object and method types

- Private vs public - private objects/methods cannot be accessed from outside of the class
  - but if you call an object of a class in another class, you can "access" the private constructor variable, because they are ran from the other class
  - in a project, data should be private but operations should be public
- Static vs non-static method/variable - static method is called on the class, non-static on the object, at the same time, a static variable belongs to the class
  - i.e. when we want to save how many instances of a class we have, we create a static variable to count the number of objects - this is not used or changed by any of the objects, it is a property or our class

More on the constructor method

- this. variable type - it refers back to the variable created inside of the class
- when we change the value with the use of a method parameter with the same name, it can lead to problems, this. is used for clarification