Primitives

In Java the simplest type of data structure we can use are simple variables. These simple variables store a single piece of information such as a number, a character or a true/false boolean value. We call these **primitives** or **primitive types**. All data stored in Java is ultimately stored in primitives.

Java has eight primitive types: `boolean`, `byte`, `char`, `short`, `int`, `long`, `float` and `double`.

Definitions on Wikipedia

Primitives are stored in the stack, in the frame for the method they are created in.

Objects

Objects in Java tend to be a collection of data elements and methods. The data elements in an object may be primitives, collections of primitives (arrays), objects or collections of objects.

Java tries to hide how objects are stored in memory to make coding easier. It seems like objects are stored just like primitives but they are stored in a different area of memory, the **heap**, and are accessed differently.

Wrapper Classes

Java provides a set of classes that each store a single value of a primitive type, along with methods for operating on/with it. These are known as **wrapper classes** and there is a wrapper class for each primitive variable.

Primitive : `byte`, `short`, `int`, `long`, `float`, `double`, `boolean`, `char`
Wrapper : `Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`, `Boolean`, `Character`

The purpose of wrapper classes is to allow primitives to be treated as objects, allowing us to treat all data the same way in code.

Primitives Vs Objects

Primitives and objects are stored in different locations in memory. Most of the time we will only notice the difference between the two when we pass them to methods as parameters.

Primitives

When we pass a primitive type to a method a local version of that primitive is created in the method (in the method's frame) and the value of the primitive is copied in to the local version. In the code below the value in the primitive `number` is passed in to `increaseNumber` where it is stored in a new local variable called `input`. When `input` is incremented (`input++`) the value of `number` does not change. This is because the two primitives are stored in two seperate frames and there is no connection between the two.

```
int number = 1;
increaseNumber(number);

public int increaseNumber(int input)
{
```

```
    input++;
    return input;
}
```

If we want to change the value in `number` then we have to return the value from the method.

```
int number = 1;
*number=increaseNumber(number);*

public int increaseNumber(int input)
{
    input++;
    return input;
}
```

Objects

Objects are stored on the Heap. This is a separate area of memory from the stack. When we create an object it is placed in the stack and a link to the object is stored in the current frame. When we pass an object to a method what we actually pass to the method is the link to the object. This is why if we pass an object to a method and makke changes to it those changes are reflected through the code. The following example describes how this works.