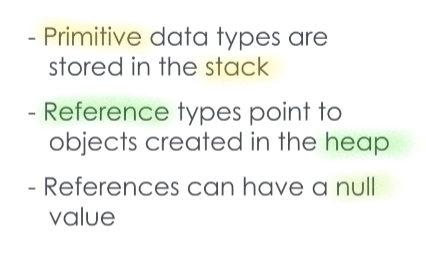
The distinction between a primitive and a reference data type is something I’ve mentioned before, but we haven’t really explored it in detail.

Well, let’s start by redefining primitive and reference types. You will recall the primitive types such as integer, double, character...

**these variable types store values directly**, and are **stored in the Stack** area of memory.[ <https://www.c-sharpcorner.com/article/stack-vs-heap-memory-c-sharp/> ]

On the other hand, a references type variable holds a **reference** – sort of like a memory address - of an Object that is instantiated from a class definition.

When I declare a Robot reference variable named ‘sam,’ this a reference that points to a Robot object that exists in the Heap memory.

**Reference variables can also have a null value**, which means they don’t point to anything.

In C#, just about everything can be an object.

This is because everything is implicitly **derived** from the System.Object class.

This means that there are certain methods that every class has inherited and can use, like the ToString() method.

**For the reference types this is intuitive, as we saw with the creation of a Robot object on the Heap.**

For the primitive types, the .NET Framework will map them to a class in the System library.

For example, when I declare the integer ‘i’, I can also instantiate a new integer just like a reference type being assigned to an object.

This is because .NET is mapping the primitive integer type to the System.Int32 class and therefore an Int32 object.

.NET also maps the other primitive types such as double, character, Boolean, and string to a corresponding class in the System library.

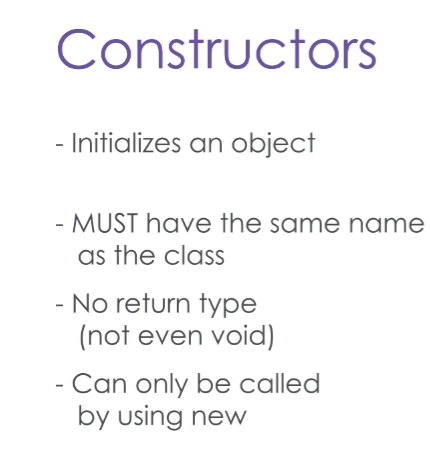
Now, we’ve seen how to create object instances already, by using the new keyword.

If I have the two lines, “Robot sam = new Robot()” and “Robot r2d2 = new Robot(),” I’ll have two different Robot objects, each with the same behaviors and the same variables, but different values stored therein.

For example, both robots have a maxSpeed integer as defined in their class, but sam might have a maxSpeed of 30 and r2d2 has a maxSpeed of 10.

But what exactly does that new keyword do?

You might have noticed that the Robot invocation with parenthesis and you’re not wrong looks similar to a method,.

This is an example of a special kind of method, called a constructor.

The constructor is responsible for initializing an object, and can be used to set variables to default or assigned values.

Constructors must have the same name as the class they are constructing.

They do not have a return type, and can only ever be invoked with the new keyword.

You can think of constructors as returning a reference to the new object that they create in an application’s heap memory.

They are also optional – a default constructor will be provided by .NET if – and only if - you don’t create one yourself. The default constructor takes in no parameters, and performs no functions.

Here is our Robot class, with a new constructor added.

This constructor will accept two parameters, maxSpeed and type.

We will use these passed-in values to set our Robot’s instance variables.

Note the keyword “this.” This is a special keyword in C# that refers to the specific object using it, and helps prevent naming conflicts.

Here we are saying, “set this object’s maxSpeed variable to the value passed in under the alias, ‘maxSpeed’.”

In our tester program, we now instantiate Robot by calling the constructor we just discussed, and passing in some values.

A class can have multiple constructors that are differentiated by the number and types of parameters that are passed-in

However, now that we’ve defined a constructor for Robot, we can’t instantiate it using the standard default constructor – because we provided one of our own, .NET won’t give us the free one.

Constructors are an important way to guarantee that your objects have the information they need to function properly.

You should also make sure that you understand the difference between a reference variable and the actual objects in memory that they point to.