# Constructors

**The goal for this exercise** is to understand what constructors are, how to define them, and how to call them, including ‘default’ constructors, and including the use of overloading to provide multiple constructors.

One of the advantages of having a clear separation between the public interface of an object, and private internal implementation of an object is that once you've got the data in the object, you can then ask the object to do things, such as saying "Hey, object – print yourself!" with a command like theDishwasher.Print();. This is all well and good, *as long as each object has valid data stored inside it*. But what if there isn't valid data stored inside it? (Perhaps a co-worker created the object, and forgot to properly initialize it) We can still tell the object to do things, but it won't be nearly as useful.

It would be great if the C# language (and the C# compiler) could be used to help us remember to initialize each object, as we create it. As it turns out, there is a language feature that does this, exactly. A **constructor** is a special method that will **always** be called when the object is created (it will be called by the **new** command, essentially). The method exists to construct (to initialize) each object, so as to ensure that each object has valid data assigned to it. A **default constructor** is a constructor that has no parameters, and is useful because it allows other people to create instances of your object without knowing much about it (for example, if you wanted to add a control (a software component/block) to Visual Studio, VS may create instances of the object using the default constructor). You can also **overload** your constructor, which is normally what's done (so that people have several options about how much info they need to provide, in order to properly initialize the new object). One neat trick is that you can use the this(…); statement, as the first line in one of your constructors, in order to call a different constructor.

**What you need to do for this exercise:**

Within the provided starter project, there is already a class named Constructors\_Exercise. You should create a class named Dishwasher below it, in the space indicated. Each Dishwasher object can hold a maximum number of glasses, and is currently holding between zero and that maximum number. For example, a small dishwasher might hold a maximum of 5 glasses, and currently be holding 2, while a larger dishwasher might hold a maximum of 20 glasses, and currently be holding 0 (none). Similarly, each Dishwasher object can hold a maximum number of plates, and a maximum number of bowls, and is currently holding some number of plates, and bowls.

Additionally, you should define a method named Print(), which takes no parameters, and returns no data. When called, Print() will print out how many of each item the Dishwasher object is currently holding, out of what maximum.

**What’s new for this exercise:**

1. Implement the Dishwasher class, as described above.
	1. Don’t forget to make the Dishwasher class public, so that the tests can access the class, like so:
	public class Dishwasher
	2. For this exercise, you should make all the instance variables private, like normal. However, you are not required to create any getter/setter methods. You are welcome to do so (if you want to), but it is not required for this exercise.
2. Make sure that your Dishwasher class has a default constructor, which initializes the all the ‘maximums’ to be 10, and the ‘currently holdings’ to be 0. So if you create a Dishwasher using the default constructor, and then printed it, it would print out the following:

Holding 0 of 10 glasses
Holding 0 of 10 plates
Holding 0 of 10 bowls
3. Overload your constructor by adding a second, non-default constructor, which takes 3 pairs of numbers as parameters – one parameter will for the maximum, and one for the current value of, each of the glasses, plates, and bowls fields. So running the following code (which demonstrates the use of this constructor):

Dishwasher d = new Dishwasher(0, 10, 1, 5, 3, 7);
d.Print();

will print:

Holding 0 of 10 glasses
Holding 1 of 5 plates
Holding 3 of 7 bowls
	1. As a side-note, you’ll notice that passing 6 integers into the constructor is getting a bit awkward. It’s hard to remember what they all mean, and even if you do remember that they’re pairs (0/10, 1/5, 3/7), it’s still tricky to remember which one goes first, second and third (Glasses, bowls, plates? Plates, bowls, glasses? Etc). While we won’t stop to explore better ways here, it’s worth noting that 6 parameters of the same type, in a row, is a bit awkward.
4. Make sure to call each version of the constructor at least once, so that you can verify that each one works.

Notice how the implementation of constructors helps guarantee that other programmers will properly initialize the objects that they create (or else get a compiler error).
5. You need to uncomment, and then run, the tests in the NUnit\_Tests\_Dishwasher class, and make sure that all the tests pass.