Introduction to Objects and Classes

In this section, you will get familiar with the basic building blocks of object-oriented programming: Objects and Classes.

WE'LL COVER THE FOLLOWING

Definitions

> A **class** can be thought of as a user-defined blueprint or prototype used for creating objects. In other words, a class specifies what properties an object should have and how it should behave.

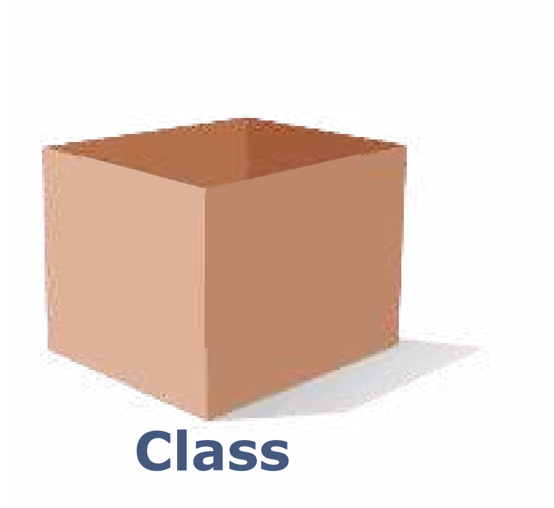
We referred to classes as the blueprints for creating objects. When looking at it from an object's perspective we can say that objects are the run time instances of classes. This may not make much sense right now, but we will look into it practically by creating our own objects soon.

From the above discussion we can infer that:

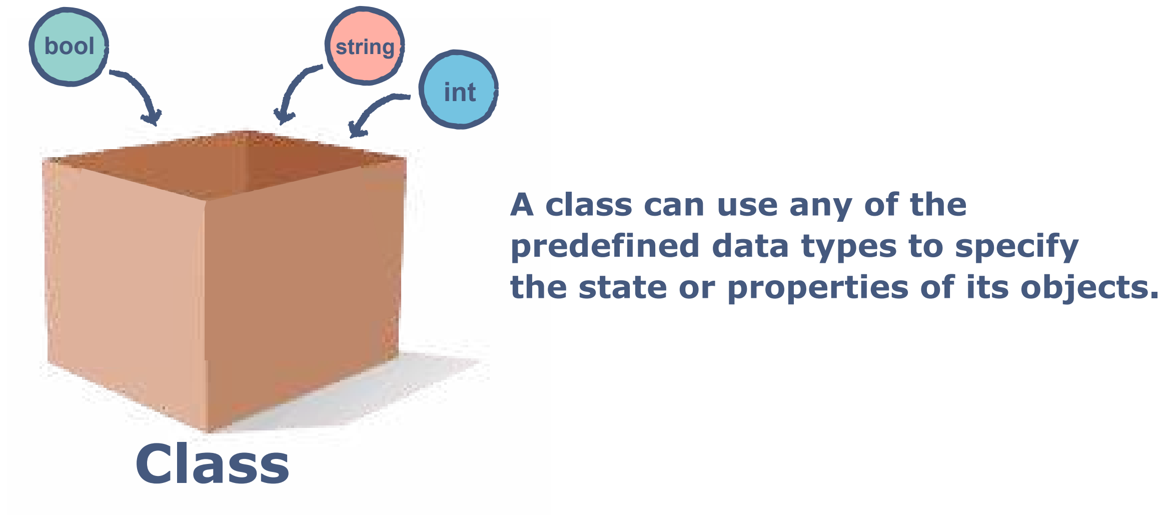
> An **object** is an instance of a class.

In C#, we have several different data types like `int,` `char,` `bool` etc. We can use any of these types in the program, but they provide very limited features to the developers. Well, object-oriented programming wouldn't make sense if we couldn't define our own data types. This is where **classes** come into play.

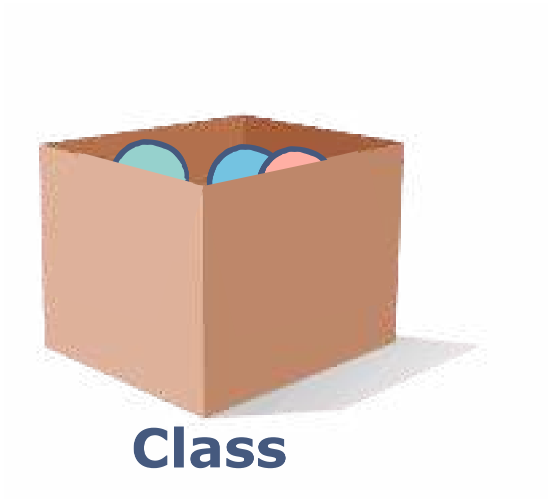
Classes are used to create *user-defined data types*. The predefined `string` data type in C# is also a class. We can use predefined data types to create our own classes. The cool part is that our classes can contain multiple variables and methods to manipulate these variables which would be available to us whenever a class object is created.

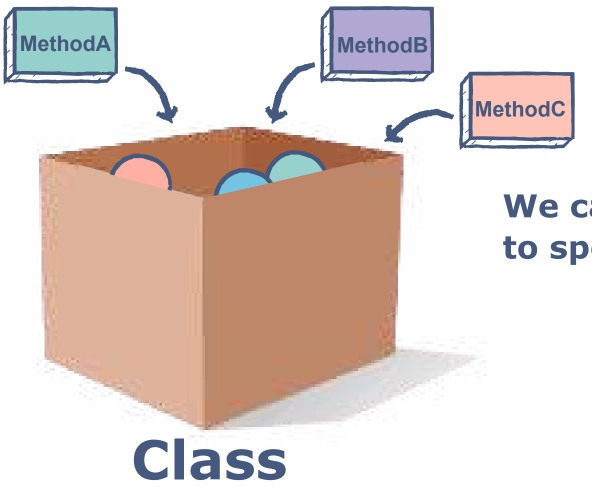


1 of 5

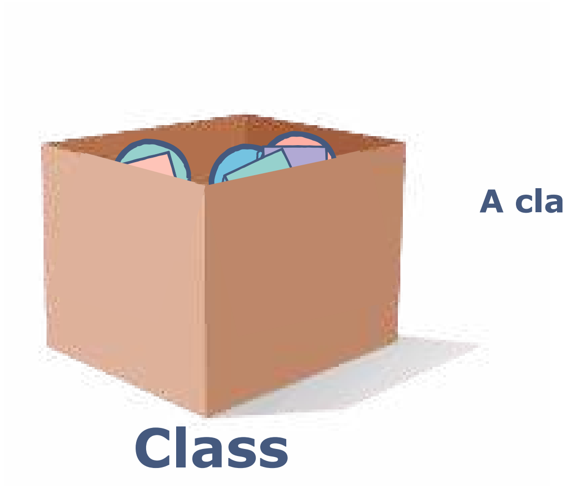

bool
string
int
A class can use any of the predefined data types to specify the state or properties of its objects.
Class

2 of 5





We can define methods in a class to specify the behavior of its objetcs.



A class contains fields and methods.

A Brief Encounter

In the real world, we can find many objects around us like cars, buildings, and even humans. All these objects have some state and behavior.

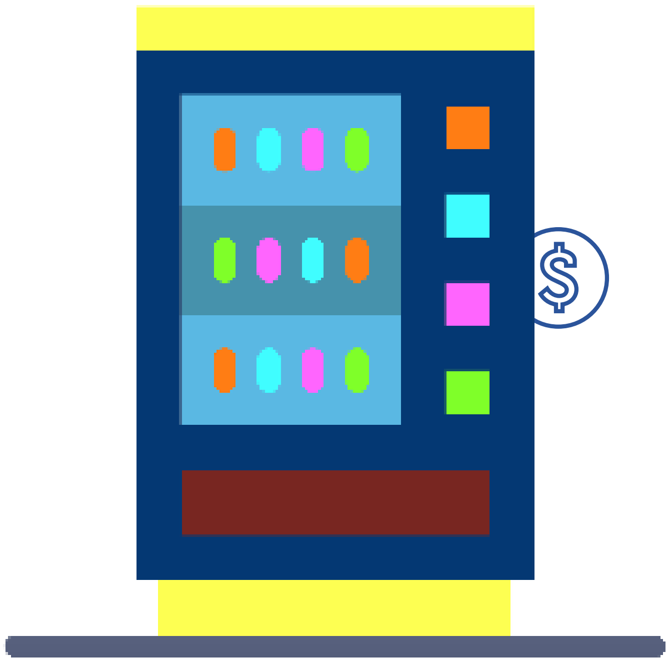
Let’s consider a very basic *vending machine*. This machine sells several products placed on its shelves.

On a macro level, we can consider this machine as an object. The *state* of the vending machine can be modeled using variables that include, but are not limited to, the following:

- A `count` that stores the count of products available in the machine.
- A `capacity` that stores the total number of products the machine can have.
- A `moneyCollected` to store the money it has collected.
- A `manufacturer` to store the name of the manufacturer of the machine.

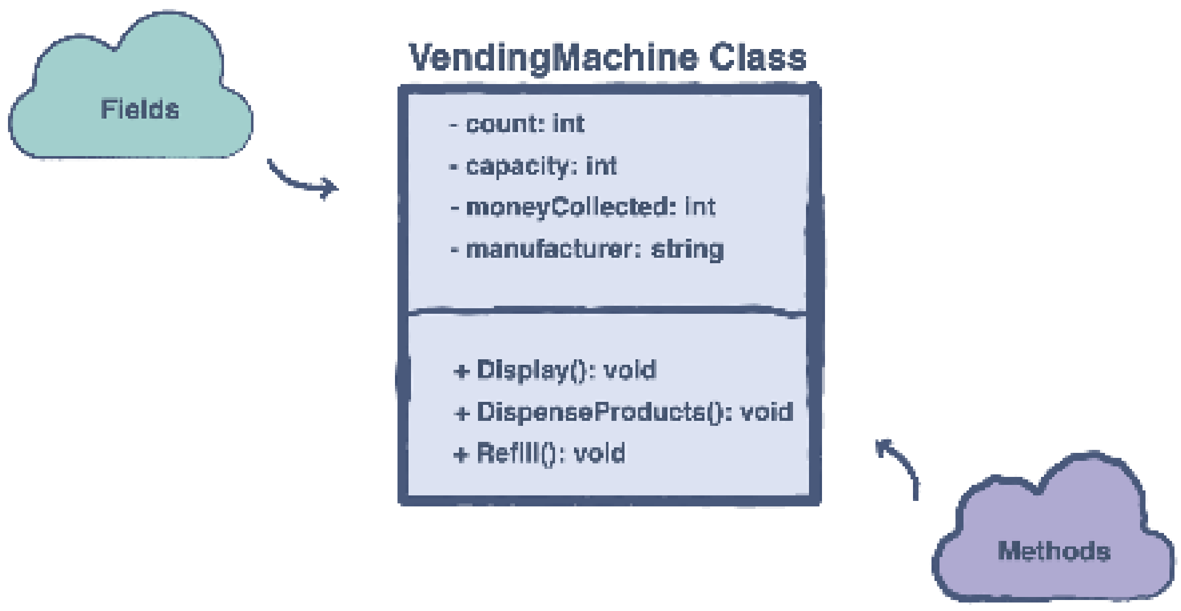
The *behavior* of the vending machine can be modeled by defining methods that include, but are not limited to, the following:

- `Display()` to display the available options to the buyer.
- `DispenseProducts()` to dispense the sold products to the buyers.
- `RefillProducts()` to refill the products having no stock left.


A food vending machine

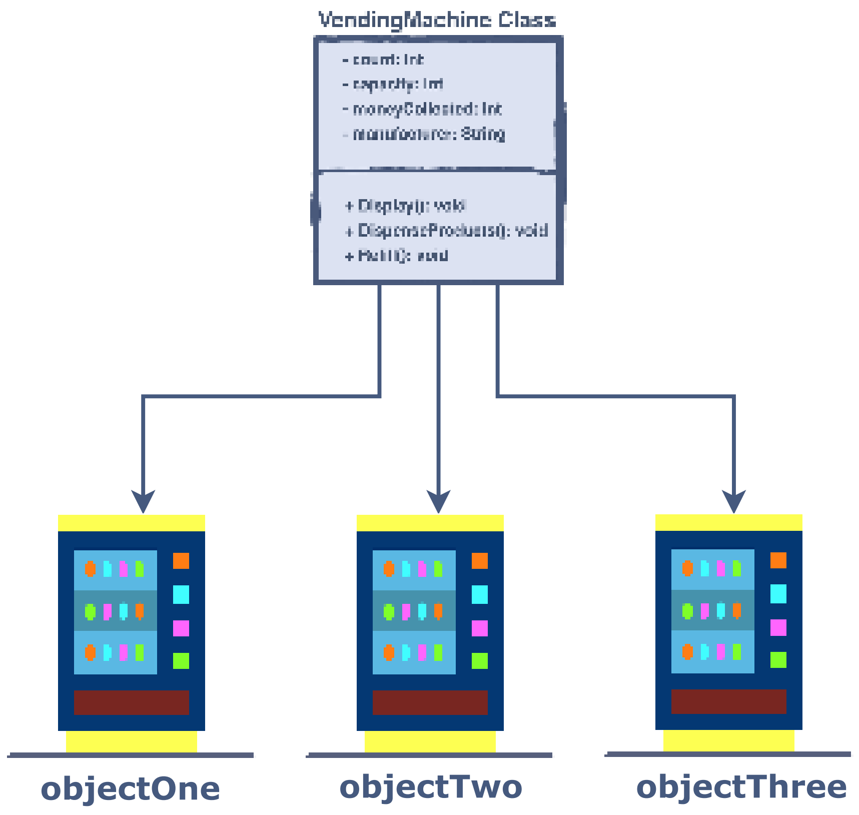
Example

Let's start with the example of a `VendingMachine` class. Below, we can see how the **state** of a vending machine object is modeled using variables and behavior is modeled by defining methods:



A basic vending machine class

1 of 2



Vending Machine Objects

Vending machine objects created using VendingMachine class

2 of 2

We can see two types of *members* in the `VendingMachine` class above. In general, these two categories are present in all classes.

> The variables and methods defined inside a class are also known as the members of that class.

Fields

> When variables are defined inside a class to store the state of an object, they are known as **fields**.

These are also known as the **member variables** of a class. This is because they store the information relevant to the object of the class. A vending machine object would have a capacity, a certain number of available products, and so many other pieces of data that we could store in variables.

Methods

The methods defined in a class enable its objects to perform operations on their fields. In the case of the `VendingMachine` class, whenever the `DispenseProducts()` method will be called, it will decrement the `count` field of the `vendingMachine` object as the machine's total stock will be now one fewer than before.

Benefits of Using Classes

The concept of classes allows us to create complex objects and applications in C#. This is why classes are the basic building blocks behind all of OOP's principles.

- Classes are also very useful in compartmentalizing the code of an application. Different components could become separate classes that would interact through interfaces. These ready-made components will also be available for use in future applications.

- Classes make it easier to maintain the different parts of an application since it is easier to make changes in classes, more on this later.

In the next lesson, we will start our journey into creating a class.