Inheritance is a core principle of Object-Oriented Programming, which allows you to write code that mimics the real world based on how we perceive it and how we categorize things into groups.

Implementing Inheritance can help you organize C# Classes and code into hierarchy.

Consider if you were writing code to simulate cars. You have lots of different kinds of car, like SUVs, sedans, coupes, and convertibles.

They all share the properties and behaviors of a car, but their implementations are slightly different.

Applying the concept of polymorphism, or many forms, with Inheritance allows a programmer to write common code for less specific classes, such as a generic “buckleUpPassengers” method in a Car class,

while specifying that the method in a Coupe class will only handle two passengers, but a sedan will handle 5.

Being able to organize code like this is advantageous when a program contains many classes and lines of code in several files.

Inheritance encourages reusing that code, and reducing code duplication also reduces maintenance.

When you have to update a block of code or add functionality, you only have to make the change in the parent class.

Before looking at some examples though, let’s clarify some terminology.

Inheritance starts with a Parent or Base Class, which is at the top of a class hierarchy and contains common class methods, and variables that are passed down.

We say the Child Class is derived from or extends a Parent Class, and retains the Parent class’ members while adding its own functionality or overriding the functionality inherited from the parent.

The keyword ‘virtual’ indicates that a method in a Parent Class can be overridden by a method in a Child Class to alter the functionality.

The keyword ‘override’ used in conjunction with keyword virtual, indicates a method in the Derived Class that is overriding a method in the Base Class.

The keyword ‘base’ is used in a Child Class to call a variable or method from the Parent Class and distinguish it from Child Class members with the same naming signature.

This is similar to how the “this” keyword works.

The keyword “sealed” is applied to a class declaration, to prevent that class from being inherited.

Here we have 2 classes: A Parent Class called Robot, and a Child Class called Hoverbot, which derives from the Robot Class using the colon operator and the Parent Class name.

C# does not support multiple inheritance – in other words, you could not have multiple parent classes for a single child class.

However, you could have multiple generations of parent and child classes.

In this example the Travel method in the Robot Class is declared with the ‘virtual’ keyword to allow the HoverBot Class to alter this method’s functionality using the ‘override’ keyword.

The method signatures for both Travel methods must be the same with the exception of the ‘virtual’ and ‘override’ keywords.

The HoverBot class inherits the maxSpeed and type variables, and also has its own properties not present in the parent class. The type is always set to “HoverBot” in the constructor.

When we create an instance of each, and invoke both object’s move methods, we get different results that showcase the inherited and new properties.