* Ex
* plain the meaning of Polymorphism.

**Polymorphism is the process of taking a method and changing its structure and behavior in order to accomplish the specifications of the class that needs to use it. That means that the “same” method can have different behaviors with the same logic.**

* Highlight a benefit of Polymorphism.

**You can have the same logic on the parent class and the whole code. But, since you can change it as you want, with the same code you can “call” the method with the behavior you want and require in that specific moment.**

* Provide an application of Polymorphism.

**For example, if you have a warehouse with different kinds of cars. Of course, all the cars have some similar characteristics, like color, brand, model, etc. But no all of them speed up to the same speed limit, so, you can have that “behavior” in common which is “GetMaxSpeed()” and, depending on what kind of car it is, you will get a different value, even when you are using the “same” method.**

* Use a code example of Polymorphism from the program you wrote. (You should copy and paste a few lines of code that demonstrate the use of the principle.)

**Class: Goal  
public abstract string GetStatus();**

**Class: SimpleGoal**

**public override string GetStatus()**

**{**

**string checkbox;**

**if (IsCompleted())**

**{**

**checkbox = "X";**

**}**

**else**

**{**

**checkbox = " ";**

**}**

**return $"[{checkbox}] {GetName()} ({GetDescription()})";**

**}**

**Class: GoalManager**

**if (input == "1") // SimpleGoal**

**{**

**\_goals.Add(new SimpleGoal(name, description, points));**

**}**

**Class: Program**

**else if (choice == "2")**

**{**

**manager.DisplayScore();**

**manager.ListGoals();**

**}**

* Thoroughly explain these concepts. (This likely cannot be done in less than 100 words.)

**First, we have the class Goal, this class has the method GetStatus() but it is set as *abstrack,* that means that the method can be overridden and change its result depending on the code that could be added. So, with that said, we have the class SimpleGoal, that also has the GetStatus() method, but it has its own behavior on this class, it will set an X if the goal is completed, if not, it will be empty. Then, the class GoalManager calls that method and apply it when add an object of the class SimpleGoal to the list, so, the next time that the GetStatus method id called with this obeject, it will take the values from the SimpleGoal-class behavior.**