Articulate Polymorphism

The word polymorphism, means “many forms”. As applied in programming, this means a method within a parent class that every child class uses but may implement differently. With an abstract method, every child class implements the method differently. With a virtual method, there is code in the parent class that most or all child classes use, and the method defined in the child class may still add unique code to it.

A benefit to polymorphism in programming is code reuse. A method is defined in the parent class, that all the child classes will use in one form or another, and the unique differences are defined in the child class, while all the similarities are coded only once in the parent class and used in every instance of the child class.

A problem encountered by programmers that polymorphism fixes was shown to us in the inheritance project. We had a method that each child class used, but implemented with small differences. Despite having the same function, we needed to code the entire method in each child class without polymorphism when it would have been much simpler and easier to use polymorphism.

Code example: ↓

Here is an example of code I used in my Goal program, in the Goal (base) class:

**public** **virtual** List<string> SaveInfo()

    {

        string rewardValue = \_rewardValue.ToString(); // converts int rewardValue to string

        return [\_name, \_description, rewardValue];

    }

We see the information returned by the base/parent class includes the name, description, and reward value. …

Here is an example of the method override in the Simple:Goal class

**public** **override** List<string> SaveInfo() => ["Simple Goal", ..base.SaveInfo(), \_isComplete.ToString()];

… Then in the child class, the method returns the type of goal “Simple Goal” and the completion status, in addition to the base method information.