Glen Kelley

11/22/2024

Prove: Articulate – Polymorphism

Polymorphism means that a set of code is able to is able to be altered for various scenarios. This means that a single function can be used repeatedly, but each iteration would be able to have different behaviors and results. In programming, this can be done using abstract classes, virtual functions and overriding. An abstract class or virtual function creates a form of blank template that is referenced by child classes. By setting a template in an abstract class or virtual function, the child classes can each create their own version of the function. However, this also means that the child classes function under similar circumstances. Like with inheritance where child classes could all be stored in a list of the same parent class, the same principle can be applied to the overrode functions.

The main benefit of polymorphism is simply that it allows for a core piece of code to be reused in many different circumstances. An example of this would be the goals project. In this case, multiple types of goal classes were used for the most recent project. In this project, each version of a goal has to be able to display the details. However, different types of goals have different requirement and details.

public abstract class Goal

{

    protected string \_title;

    protected string \_description;

    protected int \_points;

    protected bool \_isComplete = false;

    public abstract string Describe();

}

public class SimpleGoal: Goal

{

    public override string Describe()

    {

        if (!\_isComplete)

        {

            return $"[ ] {\_title} ({\_description})";

        }

        else

        {

            return $"[X] {\_title} ({\_description})";

        }

    }

}

public class ChecklistGoal: Goal

{

    private int \_targetGoal;

    private int \_timesCompleted;

    private int \_bonusPoints;

    public override string Describe()

    {

        if (!\_isComplete)

        {

            return $"[ ] {\_title} ({\_description}) -- Currently completed: {\_timesCompleted}/{\_targetGoal}";

        }

        else

        {

            return $"[X] {\_title} ({\_description}) -- Currently completed: {\_timesCompleted}/{\_targetGoal}";

        }

    }

}

It can be seen that each goal type has different requirements and uses, however, the same base class is able to accommodate these large changes because it uses polymorphism to act as a template.