Polymorphism, deriving from Greek, many (Poly) forms (morph). It is the ability of having the same code, behaving in different ways, enhancing code flexibility and reusability. The benefit of Polymorphism is that allows the code more extensible and modular, to be easier to maintain because we can add classes, without modifying existing code that uses polymorphic objects.

This concept, put together with the other principles of programming with classes, allow us to write many different types of programs. An example would be a GUI where a single “draw()” method can be defined in a base “Shape” class, and each derived class like “Circle”, “Rectangle”, or “Triangle” (as done during the practice activity, this week) can implement its own version of the “draw()” method.

Another example that shows the Polymorphism in action, can be taken from the program (Eternal Quest) requested for this week:

Base Class “Goal”  
public abstract class Goal

{

// Protected member variables that can be accessed by derived classes

protected string \_shortName;

protected string \_description;

protected int \_points;

// Constructor to initialize the goal

public Goal(string name, string description, int points)

{

\_shortName = name;

\_description = description;

\_points = points;

}

// Abstract methods to be overridden by derived classes

public abstract int RecordEvent();

public abstract bool IsComplete();

public abstract string GetStringRepresentation();

public abstract string GetDetailsString();

}

Derived Class “SimpleGoal”  
public class SimpleGoal : Goal

{

private bool \_isCompleted;

// Constructor to initialize a SimpleGoal

public SimpleGoal(string name, string description, int points)

: base(name, description, points)

{

\_isCompleted = false;

}

// Override RecordEvent to mark the goal as completed and return points

public override int RecordEvent()

{

if (!\_isCompleted)

{

\_isCompleted = true;

return \_points;

}

return 0;

}

// Override IsComplete to indicate if the goal is completed

public override bool IsComplete()

{

return \_isCompleted;

}

// Override GetStringRepresentation for saving the goal

public override string GetStringRepresentation()

{

return $"SimpleGoal|{\_shortName}|{\_description}|{\_points}|{\_isCompleted}";

}

// Override GetDetailsString for displaying the goal details

public override string GetDetailsString()

{

return $"[{(\_isCompleted ? "X" : " ")}] {\_shortName} ({\_description})";

}

}

Derived Class “ChecklistGoal”

public class ChecklistGoal : Goal

{

private int \_amountCompleted;

private int \_target;

private int \_bonus;

// Constructor to initialize a ChecklistGoal

public ChecklistGoal(string name, string description, int points, int target, int bonus)

: base(name, description, points)

{

\_amountCompleted = 0;

\_target = target;

\_bonus = bonus;

}

// Override RecordEvent to record progress and return points with bonus if completed

public override int RecordEvent()

{

\_amountCompleted++;

if (\_amountCompleted == \_target)

{

return \_points + \_bonus;

}

else if (\_amountCompleted > \_target)

{

return \_points;

}

return \_points;

}

// Override IsComplete to indicate if the goal is completed

public override bool IsComplete()

{

return \_amountCompleted >= \_target;

}

// Override GetStringRepresentation for saving the goal

public override string GetStringRepresentation()

{

return $"ChecklistGoal|{\_shortName}|{\_description}|{\_points}|{\_target}|{\_bonus}|{\_amountCompleted}";

}

// Override GetDetailsString for displaying the goal details

public override string GetDetailsString()

{

return $"[{(\_amountCompleted >= \_target ? "X" : " ")}] {\_shortName} ({\_description}) -- Currently completed: {\_amountCompleted}/{\_target}";

}

// Method to display fireworks when the goal is completed

public void DisplayFireworks()

{

Console.WriteLine("Fireworks display!");

// Fireworks display logic here

}

}

Usage in “GoalManager”

public class GoalManager

{

private List<Goal> \_goals;

private int \_score;

// Constructor to initialize the GoalManager

public GoalManager()

{

\_goals = new List<Goal>();

\_score = 0;

}

// Method to record an event for a selected goal

public void RecordEvent()

{

Console.WriteLine("\nSelect the goal you have completed:");

for (int i = 0; i < \_goals.Count; i++)

{

Console.WriteLine($"{i + 1}. {\_goals[i].GetDetailsString()}");

}

int choice = int.Parse(Console.ReadLine()) - 1;

if (choice >= 0 && choice < \_goals.Count)

{

// Using polymorphism to call the appropriate RecordEvent method

int points = \_goals[choice].RecordEvent();

\_score += points;

Console.WriteLine($"Congratulations! You have earned {points} points!");

// Checking if the goal is a ChecklistGoal and if it's complete, then display fireworks

if (\_goals[choice] is ChecklistGoal checklistGoal && checklistGoal.IsComplete())

{

checklistGoal.DisplayFireworks();

}

}

else

{

Console.WriteLine("Invalid choice.");

}

}

}

* The Goal class is the base class with abstract methods like RecordEvent, IsComplete, GetStringRepresentation, and GetDetailsString.
* The SimpleGoal and ChecklistGoal classes are derived classes that override these methods, providing specific implementations.