1 - Meaning of Polymorphism

Polymorphism is a key feature of object orientation, allowing certain methods to behave differently depending on whether an object is a base or derived class and which derived class it is.

In other words, a method with an identical signature, if called in the base class, behaves a certain way, if called in a child class may behave differently if that method has been adjusted because it was required to have a different behavior.

In other words, this principle is shown when one line of code can have different behavior depending on the context, so that the same interface can be used for a wide range of actions, which makes the code much more flexible.

2- Benefit of Polymorphism

Polymorphism provides more flexibility because it allows code written in the base class to be easily maintained and extended, since it allows different classes to share the same interface, while implementing specific behaviors. It also allows objects from different classes, but children of a base class, to be treated uniformly. It makes the code easier to maintain, since it is possible to create child objects of a base class, extending and adding new behaviors with little or no modification to the base class.

3- Provide an application of Polymorphism

Suppose a class that represents the employees of a company. It could be a salesperson, an hourly employee, a manager, among others. All of these characters are employees, but the way to calculate salaries is different depending on the type. So we can choose/create a class called Employee that will have the general attributes and behaviors common to any person and force the derived classes to implement their own way of calculating the salary using polymorphism.

In this way, the Salesperson class could have a method that calculates commissions on sales and includes them in the salary. A class for hourly employees could calculate by multiplying the number of hours worked in a period by the monetary value per hour. And another class representing managers could calculate the salary including a managerial bonus.

In other words, all classes would have the same signature, Get\_Salary(), for example, but each one calculates the salary according to its object type.

4- Code example of Abstraction

using *System*.*Numerics*;

public class CheckListGoal:Goal

{

private int \_valueCompleted = 0;

private int \_target;

private int \_bonus;

public CheckListGoal(string *name*, string *description*, int *points*, int *valueCompleted*, int *target*, int *bonus*):*base*(*name*, *description*, *points*)

{

*this*.\_valueCompleted = *valueCompleted*;

*this*.\_target = *target*;

*this*.\_bonus = *bonus*;

}

public override void RecordEvent()

{

if(*this*.\_valueCompleted < *this*.\_target)

{

*this*.\_valueCompleted++;

}

}

public override bool IsComplete()

{

return *this*.\_target == *this*.\_valueCompleted;

}

public override string getDetailsString()

{

string message = "";

if (!*this*.IsComplete())

{

message = $"[ ] {*base*.getDetailsString()} -- Currently completed: {*this*.\_valueCompleted}/{*this*.\_target}";

}

else

{

message = $"[x] {*base*.getDetailsString()} -- Currently completed: {*this*.\_valueCompleted}/{*this*.\_target}";

}

return message;

}

public override string getStringRepresantation()

{

return $"CheckList Goal:{*this*.getDetailsString()}, {*base*.GetPoints()}, {*this*.\_bonus}, {*this*.\_target}, {*this*.\_valueCompleted}";

}

public int GetTotalPoints()

{

int total = 0;

if (*this*.IsComplete())

{

total = *this*.\_valueCompleted \* *base*.GetPoints() + *this*.\_bonus;

}

else

{

total = *this*.\_valueCompleted \* *base*.GetPoints();

}

return total;

}

}