1 - Meaning of Inheritance

Inheritance in the context of object-oriented software development means that a class called Children or a Derived class inherits attributes and behaviors from a class called Parent or a Base class. This capability allows for code reuse and simplification, since code that is common to several related classes can be written in the base class and used by the derived class, allowing the latter to focus only on what is specific to it.

For example, a class called Person might contain attributes such as name, address, and age, while the Student class might inherit these attributes and include the student identification number (ID), which will uniquely identify the student among others in an educational institution.

2- Benefit of Inheritance

Inheritance in the context of object orientation offers several benefits, such as:

1- Code reuse, since a child class can make use of the attributes and methods of the base class.

2- Code extension, since new features can be added to the derived class that will give it new capabilities that the base class does not have.

3- Simpler maintenance, since any maintenance in the base class is immediately reflected in the derived classes, all subclasses will inherit the changes.

4- Inheritance organizes the code into a well-defined hierarchy that reflects natural relationships between various types of objects.

3- Provide an application of Inheritance.

Suppose a class that reflects the stakeholders in a college, teachers, students, library staff, among others. All stakeholders are people, so we can choose/create a class called Person that will have the attributes and general behaviors common to any person.

Person

-name: string;

-address: string;

-birthdayDate: Date;

Employer

-salary: double;

-position: string;

+ assignTasks():void;

But in a college, there are people who have attributes and behaviors that are more specific to their category. For example, employees are people, so they have the same attributes and methods as the People class. However, employees have more specific, specialized attributes and methods, such as salary and position.

In this way, we use inheritance to bring all the attributes and behaviors of the Person class to the Employee class and extend (specialize) it with new attributes and methods specific to an employee.

Using inheritance to make the code leaner, less repetitive, and easier to use.

4- Code example of Abstraction

Note the following. The ListingActivity class inherits from the Activity class, taking advantage of all its attributes and behaviors, but extends it by including a new behavior, namely the Run() method present in the following code.

public class ListingActivity:Activity

{

private PromptGenerator \_promptGenerator;

private List<string> \_prompts;

public ListingActivity(string *activityName*, string *activityDescription*, int *activityDuration*, List<string> *prompts*):*base*(*activityName*,*activityDescription*,*activityDuration*)

{

*this*.\_prompts = *prompts*;

//The PromptGenerator class is responsible for randomly choosing messages and displaying them without repeating them, that is, a used message will not be repeated.

*this*.\_promptGenerator = new PromptGenerator(*this*.\_prompts); //Initialize with prompts messages

}

public void Run()

{

Console.Clear();

string time = *base*.OpeningMessage();

if(int.TryParse(time, out int result))

{

*base*.SetActivityDuration(result); // After receiving duration from the user we set the activity duration in class

*base*.GetReady(3);

Console.Clear();

Console.WriteLine("List as many responses you can to the following prompt:");

var q = *this*.\_promptGenerator.getRandomQuestion();

Console.WriteLine(q.Item2);

Console.Write("You may begin in: ");

*base*.ShowCountDown(5);

Console.WriteLine("");

*DateTime* startTime = *DateTime*.Now;

*DateTime* futureTime = startTime.AddSeconds(*base*.GetActivityDuration());

*DateTime* currentTime;

currentTime = *DateTime*.Now;

bool flag = true;

while ((currentTime < futureTime) && flag)

{

Console.Write(">");

Console.ReadLine();

currentTime = *DateTime*.Now;

//If all prompts or questions have been used, end the activity.

if (q.Item1 == -1)

{

flag = false;

}

}

*base*.EndingMessage(3);

}

else

{

Console.Clear();

Console.Write("Invalid value, press any key to continue.");

Console.ReadLine();

}

}

}