**Object Oriented Programming in C#**

Object-oriented programming (OOP) is a programming paradigm that emphasizes the use of objects and classes to model real-world entities and their interactions. C# is a language that fully supports OOP, and it provides a wide range of features to make it easy to create and use objects.

Here are some of the key features of OOP in C#:

1. **Classes**: A class is a blueprint or template for creating objects. It defines the data and behavior of objects of that type. In C#, classes are declared using the class keyword.
2. **Objects**: An object is an instance of a class. It represents a real-world entity and has a specific state and behavior. Objects can interact with each other by sending messages or calling methods.
3. **Inheritance**: Inheritance is a mechanism that allows you to define a new class based on an existing class. The new class inherits the data and behavior of the existing class and can also add its own data and behavior. In C#, inheritance is implemented using the : operator.
4. **Polymorphism**: Polymorphism is the ability of objects of different classes to be used interchangeably. This is achieved through inheritance and method overriding. In C#, you can use the virtual and override keywords to implement polymorphism.
5. **Encapsulation**: Encapsulation is the practice of hiding the internal details of an object and exposing only the public interface. This helps to ensure that the object is used correctly and prevents accidental changes to its state. In C#, you can use access modifiers such as public, private, protected, and internal to control access to class members.
6. **Abstraction**: Abstraction is the process of reducing complex systems to their essential components. In C#, you can use abstract classes and interfaces to define abstract types that represent common behavior across multiple classes.

**Field and Property:**

Fields and properties are two types of class members in C# that can be used to define data in a class. A field is a variable that is defined directly in a class. It is used to store data and can be accessed using an instance of the class. Fields can be either public or private, and their value can be set directly or through a constructor.

A property, on the other hand, is a special type of method that is used to get or set the value of a private field. Properties provide a level of encapsulation, allowing the internal state of an object to be controlled while still providing a public interface for accessing it. Properties are declared using the get and set keywords, and can be either read-only, write-only, or read-write.

Syntax:

public class Sampleclass{

string samplefield;

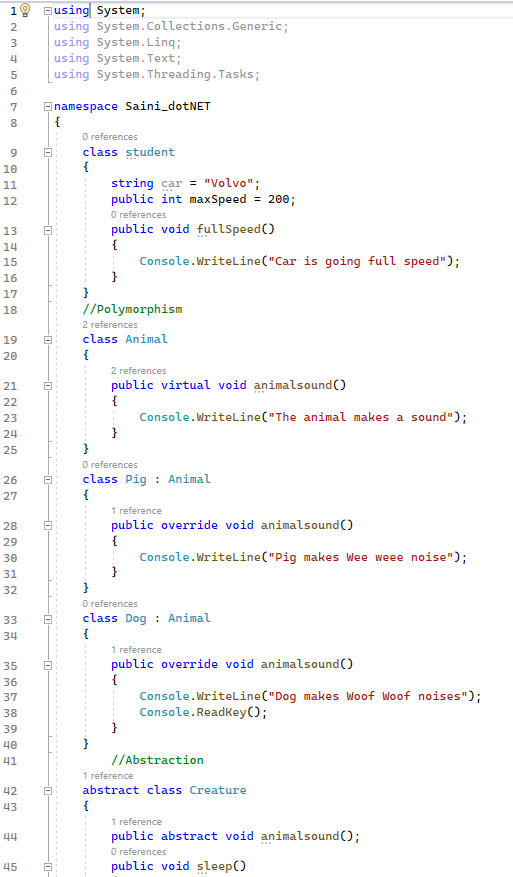
string field\_name;

public int property\_name{get; set;}

}

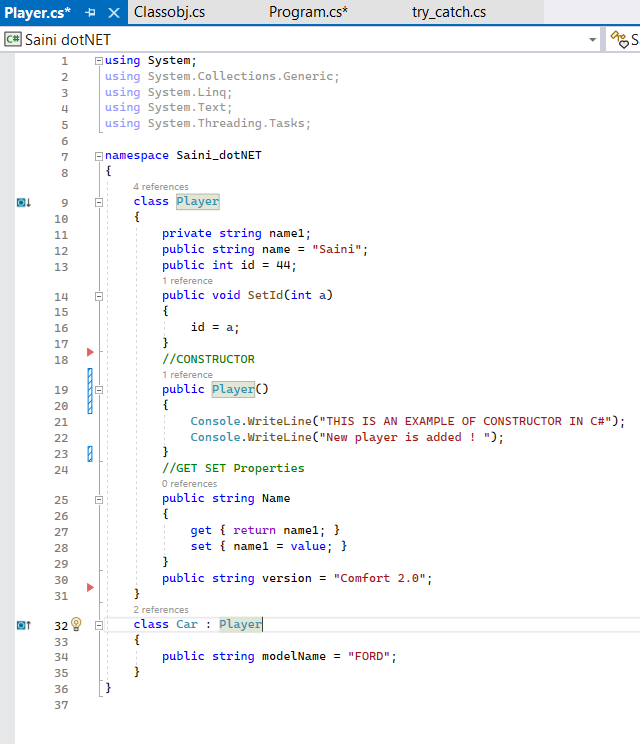
**C# program:**

class\_obj.cs

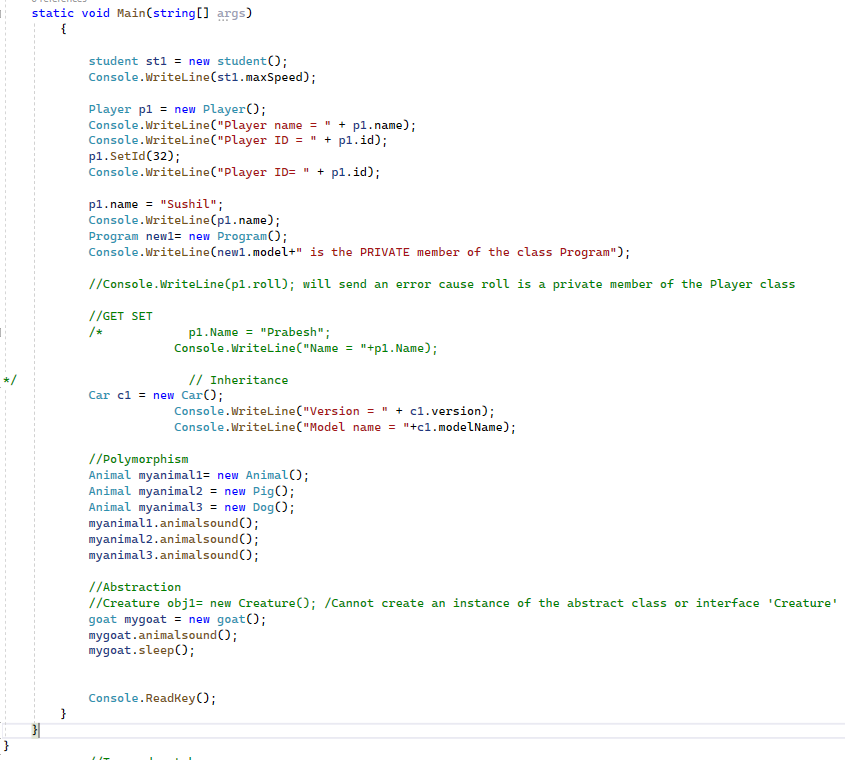




Player.cs



**Program.cs**



**Output:**

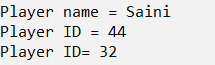
**Class and Object:**

 class Student from class\_obj.cs

**Constructor:**

 **(**constructor player from Player.cs)

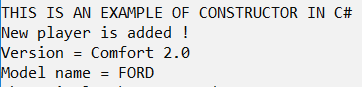
**Public Access Modifier and updating the value of the class instance:**

 (Player class from Player.cs)

**Private Access Modifier and using the instance value over default value of the class field:**

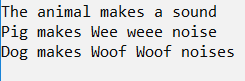


Class Player from Player.cs and class Program from Program.cs

**Inheritance:**

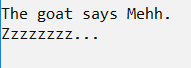
When the child class object is created it automatically calls the constructor of the parent class (i.e. Player in this case)

**Polymorphism:**



Function overriding, we are able to call the method with the same name from three different classes.

**Abstraction:**

Use of abstract in the parent’s class and override in child’s class