Scope and lifetime

* This determines the accessibility of the variable and how long the value of the variable last

Members

* Access modifiers

1. private: - the variable declared as private are only accessible inside the class

private int a = 10; ( only accessible inside the class)

1. protected: - the variable declared as protected are only accessible inside the class and all classes that derived from the class

protected int a = 10;

1. public: - can access anywhere in the project.

public int a = 10;

* Abstract Class

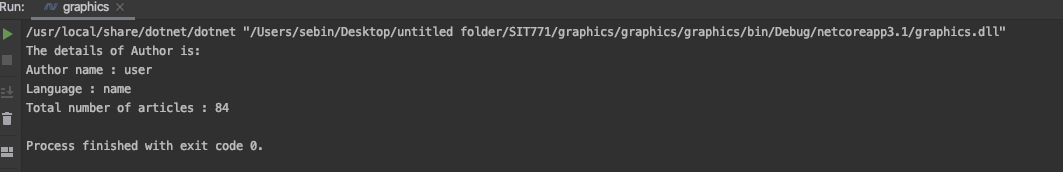
It’s inherited by the subclasses or overrides the method in its class, cannot create instance or object and all the methods in this class are abstract

public abstract class test  
{  
 *// abstract method 'show()'* public abstract void show();  
}  
  
*// class 'test' inherit   
// in child class 'child'*public class Child : test  
{  
 *// abstract method 'show()'   
 // declare here with   
 // 'override' keyword* public override void show()  
 {  
 Console.WriteLine("class child");  
 }  
}  
  
public class main\_method  
{  
 *// Main Method* public static void Main()  
 {  
 Child g;  
  
 *// instantiate class 'Child'* g = new Child();  
  
 *// call 'show()' of class 'Child'* g.show();  
 }  
}

* Static classes and class members

Purpose of static class to provide blueprints of its inherited classes. It can’t create instance to the static class because the static class members can be accessed directly with class name.

static class Author  
{  
 *// Static data members of Author* public static string first = "user";  
 public static string last = "name";  
 public static int no = 84;  
  
 *// Static method of Author* public static void details()  
 {  
 Console.WriteLine("The details of Author is:");  
 }  
}  
  
*// Driver Class*public class Program  
{  
 *// Main Method* static public void Main()  
 {  
 *// Calling static method of Author* Author.details();  
  
 *// Accessing the static data members of Author* Console.WriteLine("Author name : {0} ", Author.first);  
 Console.WriteLine("Language : {0} ", Author.last);  
 Console.WriteLine("Total number of articles : {0} ", Author.no);  
 }  
}



Inheritance

* inherit fields and methods from one class to another
* <access modifier> class NewClass:DerivedClass
* class Details {  
   *// data members* public string name;   
   public string subject;   
     
   *// public method of base class* public void readers(string name, string subject)   
   {   
   this.name = name;   
   this.subject = subject;   
   Console.WriteLine("Author name : " + name);   
   Console.WriteLine("My Favorite Author book is: " + subject);   
   }   
  }   
     
  *// inheriting the Details class using :*class Author : Details {   
     
   *// constructor of derived class* public Author()   
   {   
   Console.WriteLine("GeeksforGeeks");   
   }   
  }   
     
  *// Driver class*class Program {   
     
   *// Main Method* static void Main(string[] args)   
   {   
     
   *// creating object of derived class* Author g = new Author();   
     
   *// calling the method of base class   
   // using the derived class object* g.readers("Ruskin Bond", "C#");   
   }   
  }

inherits the methods base Details class to Author class

Polymorphism

* means having many forms, one interface and multiple functions. Provides overloading of methods.
* class baseClass   
     
  {   
   public void show()   
   {   
   Console.WriteLine("Base class");   
   }   
  }   
     
  *// derived class name 'derived'   
  // 'baseClass' inherit here*class derived : baseClass   
  {   
     
   *// overriding* new public void show()   
   {   
   Console.WriteLine("Derived class");   
   }   
  }  
    
  class Program  
  {  
    
   *// Main Method* public static void Main()  
   {  
    
   *// 'obj' is the object of   
   // class 'baseClass'* baseClass obj = new baseClass();  
    
    
   *// invokes the method 'show()'   
   // of class 'baseClass'* obj.show();  
    
   obj = new derived();  
    
   *// it also invokes the method   
   // 'show()' of class 'baseClass'* obj.show();  
    
   }  
  }

Properties

* read-only property provides a get accessor to fields not set accessor

public readonly string OriginalName;

* Setter and getter

1. get: property is used to return the value
2. set: property is used to assign value
3. public class Student {   
      
    *// Declare name field* private string name = "Student";   
      
    *// Declare name property* public string Name   
    {   
      
    get  
    {   
    return name;   
    }   
      
    set  
    {   
    name = value;   
    }   
    }   
   }  
     
   class TestStudent  
   {  
     
    *// Main Method* public static void Main(string[] args)  
    {  
    Student s = new Student();  
     
    *// calls set accessor of the property Name,   
    // and pass "Robin" as value of the   
    // standard field 'value'.* s.Name = "Robin";  
     
    *// displays name, Calls the get accessor   
    // of the property Name.* Console.WriteLine("Name: " + s.Name);  
    }  
   }

Output: Name: Robin

Constructor

* special method that runs when the class is instantiated
* same name of the class
* supports overloading

Control Flow

* If Statement : if the expression true then statement is executed

if (expression)

{

statement;

}

* Else statement: if keyword evaluates to false the statement in else will execute

if (expression)

{

statement;

}

else

{

statement;

}

* Else if:

if (expression)

{

statement;

}

else if

{

statement;

}

else

{

statement;

}

* switch statement
* while statement: statements are executed each time while the expression is evaluated as true

while (expression)

{

statement;

}

* Do-while statement