Properties are named members of classes, structures, and interfaces. Member variables or methods in a class or structures are called **Fields**. Properties are an extension of fields and are accessed using the same syntax. They use **accessors** through which the values of the private fields can be read, written or manipulated.

Properties do not name the storage locations. Instead, they have **accessors** that read, write, or compute their values.

For example, let us have a class named Student, with private fields for age, name, and code. We cannot directly access these fields from outside the class scope, but we can have properties for accessing these private fields.

Accessors

The **accessor** of a property contains the executable statements that helps in getting (reading or computing) or setting (writing) the property. The accessor declarations can contain a get accessor, a set accessor, or both. For example −

```
// Declare a Code property of type string:
public string Code {
   get {
      return code;
   }
   set {
      code = value;
   }
}

// Declare a Name property of type string:
public string Name {
   get {
      return name;
   }
   set {
      name = value;
   }
}

// Declare a Age property of type int:
public int Age {
   get {
      return age;
   }
   set {
      age = value;
   }
}
```

Example

The following example demonstrates use of properties −

```
using System;
namespace tutorialspoint {
   class Student {
      private string code = "N.A";
      private string name = "not known";
      private int age = 0;

      // Declare a Code property of type string:
      public string Code {
         get {
            return code;
         }
         set {
            code = value;
         }
      }

      // Declare a Name property of type string:
      public string Name {
         get {
            return name;
         }
         set {
            name = value;
         }
      }

      // Declare a Age property of type int:
      public int Age {
         get {
            return age;
         }
         set {
            age = value;
         }
      }
      public override string ToString() {
         return "Code = " + Code +", Name = " + Name + ", Age = " +
Age;
      }
   }

   class ExampleDemo {
      public static void Main() {
```

```
            // Create a new Student object:
            Student s = new Student();

            // Setting code, name and the age of the student
            s.Code = "001";
            s.Name = "Zara";
            s.Age = 9;
            Console.WriteLine("Student Info: {0}", s);

            //let us increase age
            s.Age += 1;
            Console.WriteLine("Student Info: {0}", s);
            Console.ReadKey();
        }
    }
}
```

When the above code is compiled and executed, it produces the following result −

```
Student Info: Code = 001, Name = Zara, Age = 9
Student Info: Code = 001, Name = Zara, Age = 10
```

Abstract Properties

An abstract class may have an abstract property, which should be implemented in the derived class. The following program illustrates this −

```
using System;

namespace tutorialspoint {
   public abstract class Person {
      public abstract string Name {
         get;
         set;
      }
      public abstract int Age {
         get;
         set;
      }
   }
   class Student : Person {
      private string code = "N.A";
      private string name = "N.A";
      private int age = 0;

      // Declare a Code property of type string:
      public string Code {
```

```
         get {
            return code;
         }
         set {
            code = value;
         }
      }

      // Declare a Name property of type string:
      public override string Name {
         get {
            return name;
         }
         set {
            name = value;
         }
      }

      // Declare a Age property of type int:
      public override int Age {
         get {
            return age;
         }
         set {
            age = value;
         }
      }
      public override string ToString() {
         return "Code = " + Code +", Name = " + Name + ", Age = " +
Age;
      }
   }

   class ExampleDemo {
      public static void Main() {
         // Create a new Student object:
         Student s = new Student();

         // Setting code, name and the age of the student
         s.Code = "001";
         s.Name = "Zara";
         s.Age = 9;
         Console.WriteLine("Student Info:- {0}", s);

         //let us increase age
         s.Age += 1;
         Console.WriteLine("Student Info:- {0}", s);
         Console.ReadKey();
      }
```

```
    }
}
```

When the above code is compiled and executed, it produces the following result −

```
Student Info: Code = 001, Name = Zara, Age = 9
Student Info: Code = 001, Name = Zara, Age = 10
```

Previous Page Print Page