Session: 9

Properties and Indexers

Objectives

- Define properties in C#
- Explain properties, fields, and methods
- Explain indexers

Applications of Properties

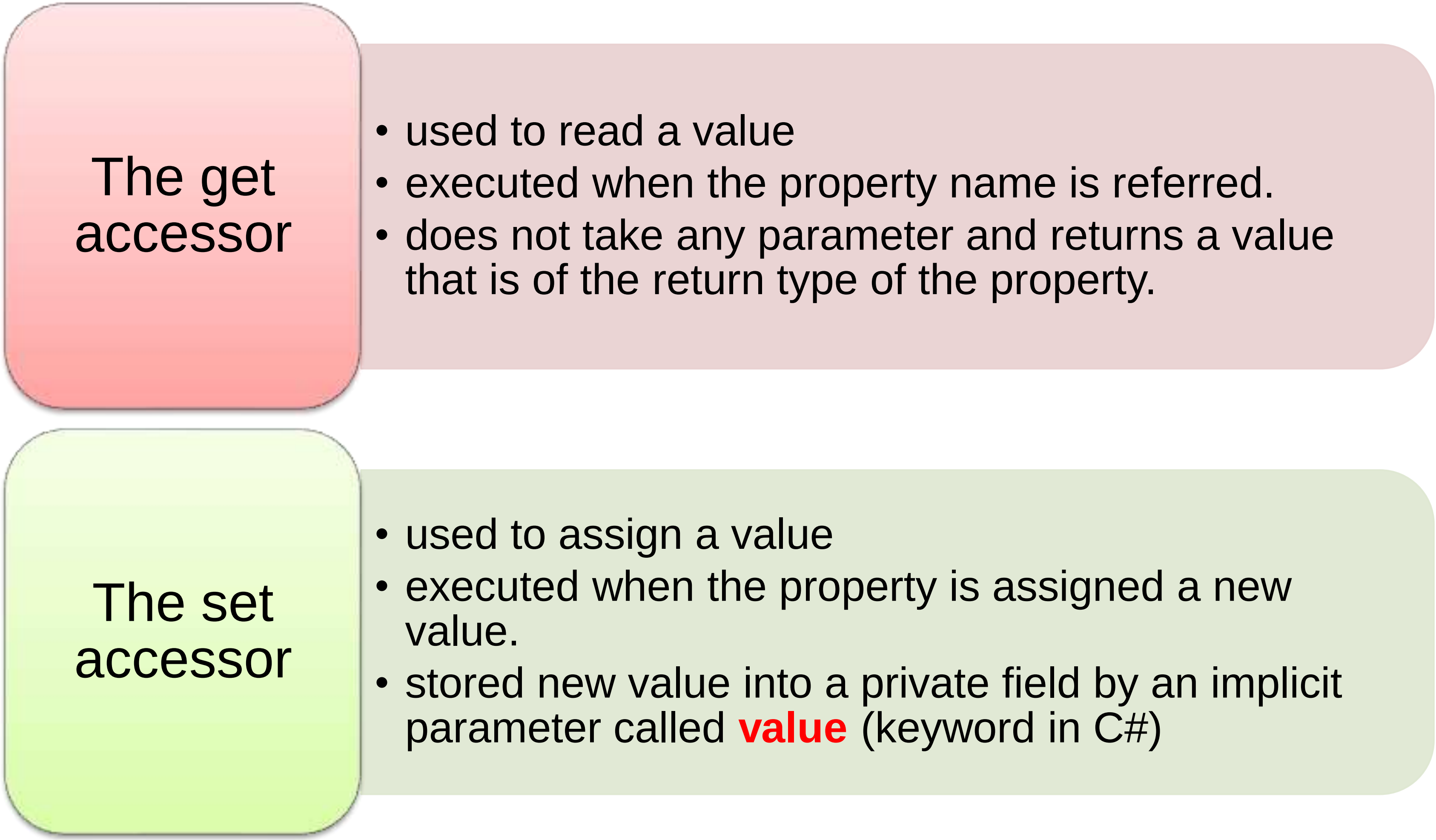
- Properties:
  - allow to access private fields and ensure security of them
  - can validate data before making changes to the protected fields and also perform specified actions on those changes.
  - support abstraction and encapsulation by exposing only necessary actions and hiding their implementation.

```
<access_modifier> <return_type> <PropertyName> {
   get  {
      // return value
   }
   set {
      // assign value
   }
}
```

where, **`return_type`**: the type of data the property will return.

get and set Accessors

- allow to read and assign a value to a field:



The get accessor

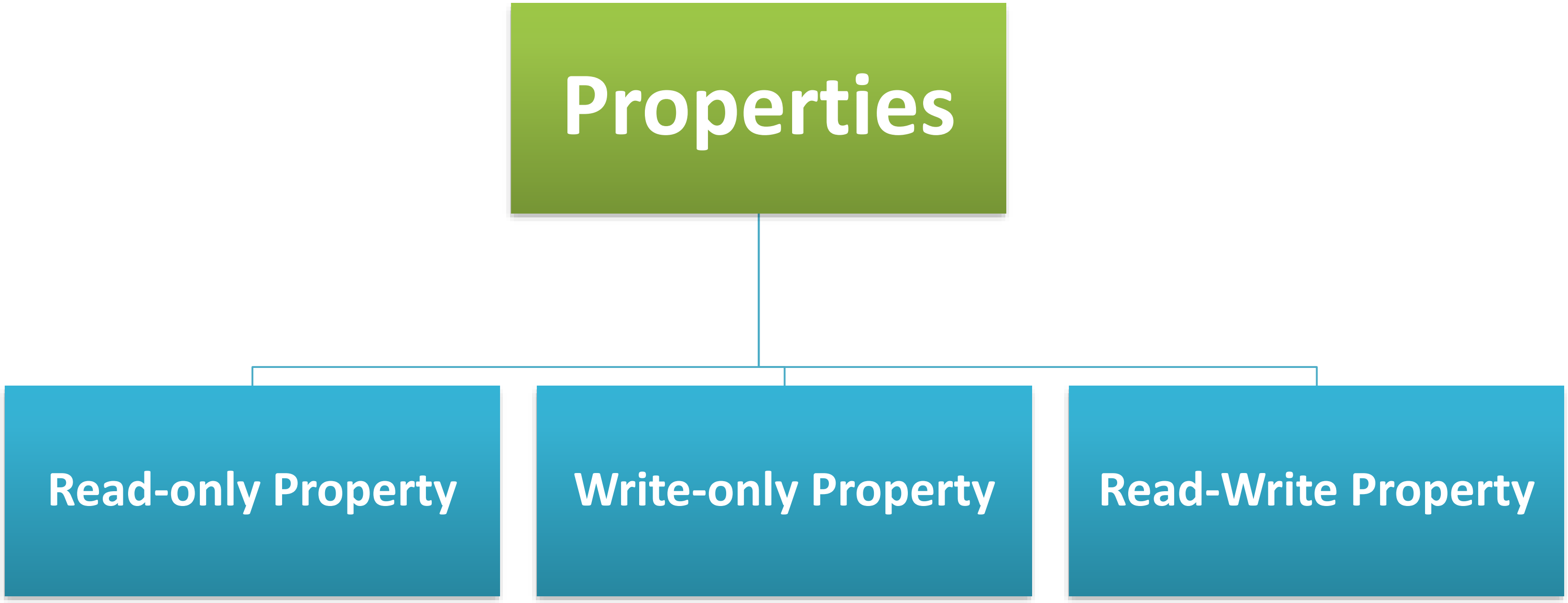
- used to read a value
- executed when the property name is referred.
- does not take any parameter and returns a value that is of the return type of the property.

The set accessor

- used to assign a value
- executed when the property is assigned a new value.
- stored new value into a private field by an implicit parameter called **value** (keyword in C#)

Categories of Properties

- Properties are broadly divided into three categories:



Static Properties

- The static property is:
  - declared by using the **static** keyword.
  - accessed using the class name.
  - used to access and manipulate static fields of a class in a safe manner.
- The following code demonstrates a class with a static property.

```
class University {
    private static string _department;
    private static string _universityName;
    public static string Department {
       get {
           return _department;
       }
       set {
           _department = value;
       }
    }
}
```

Abstract Properties

- Declared by using the **abstract** keyword.
- Contain only the declaration of the property without the body of the **get** and **set** accessors (can be implemented in the derived class).
- Are only allowed in an abstract class.
- Are used :
  - to secure data within multiple fields of the derived class of the abstract class.
  - to avoid redefining properties by reusing the existing properties.

```
public abstract class Figure {
   public abstract float DimensionOne {
        set;
   }
   public abstract float DimensionTwo {
       set;
   }
}
```

Auto-Implemented Properties

- Are properties **without** explicitly **providing** the **`get`** and **`set`** accessors.
- For an auto-implemented property, the compiler automatically creates:
  - a private field to store the property variable.
  - the corresponding **`get`** and **`set`** accessors.

- Syntax:

```
public string Name { get; set; }
```

- Example.

```
class Employee
{
      public string Name { get; set; }
      public int Age { get; set; }
}
```

Object Initializers

- The following code uses object initializers to initialize an **`Employee`** object.

```
class Employee {
    public string Name { get; set; }
    public int Age { get; set; }
    public string Designation { get; set; }
    static void Main (string [] args) {
        Employee emp1 = new Employee {
            Name = "John Doe",
            Age = 24,
            Designation = "Sales Person"
        };
        Console.WriteLine("Name: {0}, Age: {1}, Designation:
        {2}", emp1.Name, emp1.Age, emp1.Designation);
    }
}
```

Output

```
Name: John Doe, Age: 24, Designation: Sales Person
```

Properties vs Fields

Properties	Fields
are data members that can assign and retrieve values.	are data members that store values.
cannot be classified as variables and therefore, cannot use the ref and out keywords.	are variables that can use the ref and out keywords.
are defined as a series of executable statements.	can be defined in a single statement.
are defined with two accessors or methods, the get and set accessors.	are not defined with accessors.
can perform custom actions on change of the field’s value.	are not capable of performing any customized actions.

Properties vs Methods

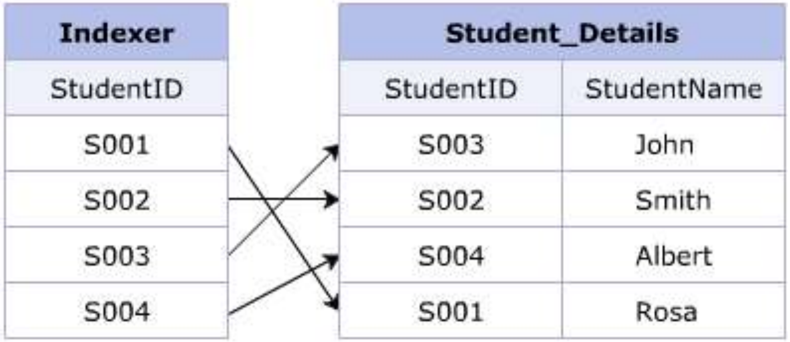
Properties	Methods
represent characteristics of an object.	represent the behavior of an object.
contain two methods which are automatically invoked without specifying their names.	are invoked by specifying method names along with the object of the class.
cannot have any parameters.	can include a list of parameters.
can be overridden but cannot be overloaded.	can be overridden as well as overloaded.

Indexers

- allow instances of a class or struct to be indexed like arrays.
- are syntactically similar to properties, but unlike properties, the accessors of indexers accept one or more parameters.

Example

- Consider a high school teacher who wants to go through the records of a particular student to check the student's progress.
- Calling the appropriate methods every time to set and get a particular record makes the task tedious.
- Creating an indexer for student ID:
  - makes the task of accessing the record much easier as indexers use index position of the student ID to locate the student record.



Indexer
StudentID
S001
S002
S003
S004

Student_Details	
StudentID	StudentName
S003	John
S002	Smith
S004	Albert
S001	Rosa

Declaration of Indexers

- An indexer can be defined by specifying the following:
  - An access modifier, which decides the scope of the indexer.
  - The return type of the indexer, which specifies the type of value an indexer will return.
  - The `this` keyword, which refers to the current instance of the current class.
  - The bracket notation ([ ]), which consists of the data type and the identifier of the index.
  - The open and close curly braces, which contain the declaration of the `set` and `get` accessors.

Syntax

```
<access_modifier> <return_type> this [<parameter>]
{
   get { // return value }
   set { // assign value }
}
```

Implementing Inheritance

- Indexers can be inherited like other members of the class.

Snippet

```
class Numbers {
   private int[] num = new int[3];
   public int this[int index] {
       get { return num [index]; }
       set { num [index] = value; }
   }
}
class EvenNumbers : Numbers {
    public static void Main() {
       EvenNumbers objEven = new EvenNumbers();
       objEven[0] = 0;
       objEven[1] = 2;
       objEven[2] = 4;
       for(int i=0; i<3; i++) {
            Console.WriteLine(objEven[i]);
       }
    }
}
```

Indexers in Interfaces

```
public interface Idetails {
    string this[int index] { get; set; }
}
class Students : Idetails {
    string [] studentName = new string[3];
    public string this[int index] {
       get { return studentName[index]; }
       set { studentName[index] = value; }
    }
    static void Main(string[] args) {
      Students objStudent = new Students();
      objStudent[0] = "James";
      objStudent[1] = "Wilson";
      objStudent[2] = "Patrick";
      Console.WriteLine("Student Names");
      Console.WriteLine();
      for (int i = 0; i< 3; i++) {
           Console.WriteLine(objStudent[i]);
      }
    }
}
```

Difference between Properties and Indexers

- Indexers are syntactically similar to properties.
- However, there are certain differences between them.

Properties	Indexers
are assigned a unique name	cannot be assigned a name and use the **this** keyword.
are invoked using the specified name.	are invoked through an index of the created instance.
can be declared as **`static`.**	can never be declared as **`static`.**
without parameters.	are declared with at least one parameter.
cannot be overloaded.	can be overloaded.
Overridden properties are accessed using the `syntax` **`base.Prop`**, where `Prop` is the name of the property.	are accessed using the syntax **`base[indExp]`,** where `indExp` is the list of parameters separated by commas.

Summary

- Properties protect the fields of the class while accessing them.
- Property accessors enable you to read and assign values to fields.
- A field is a data member that stores some information.
- Properties enable you to access the private fields of the class.
- Methods are data members that define a behavior performed by an object.
- Indexers treat an object like an array, thereby providing faster access to data within the object.
- Indexers are syntactically similar to properties, except that they are defined using the this keyword along with the bracket notation ([]).