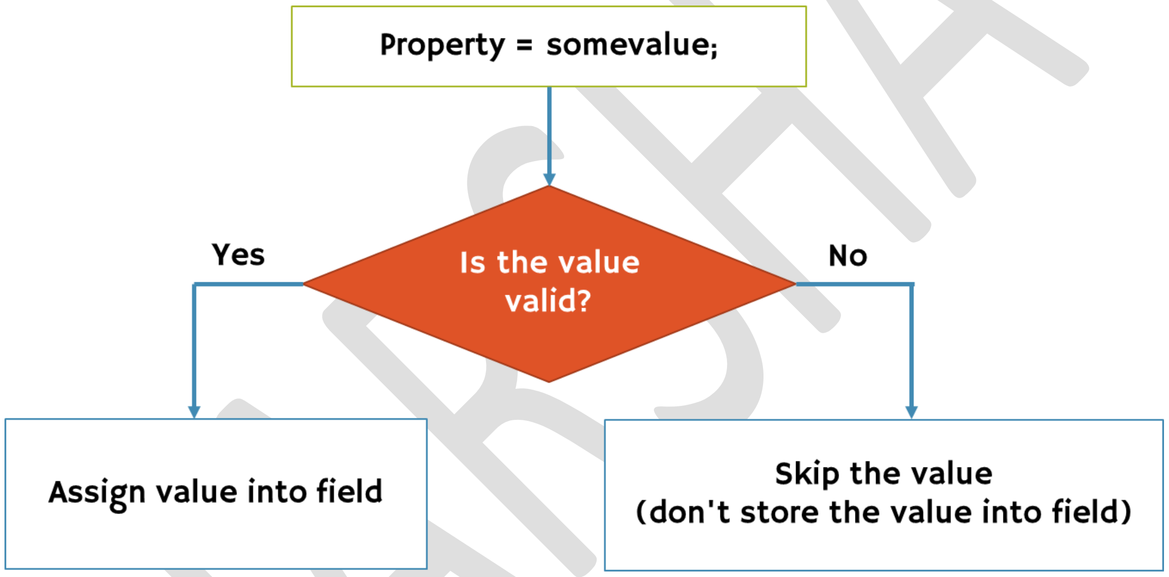
13

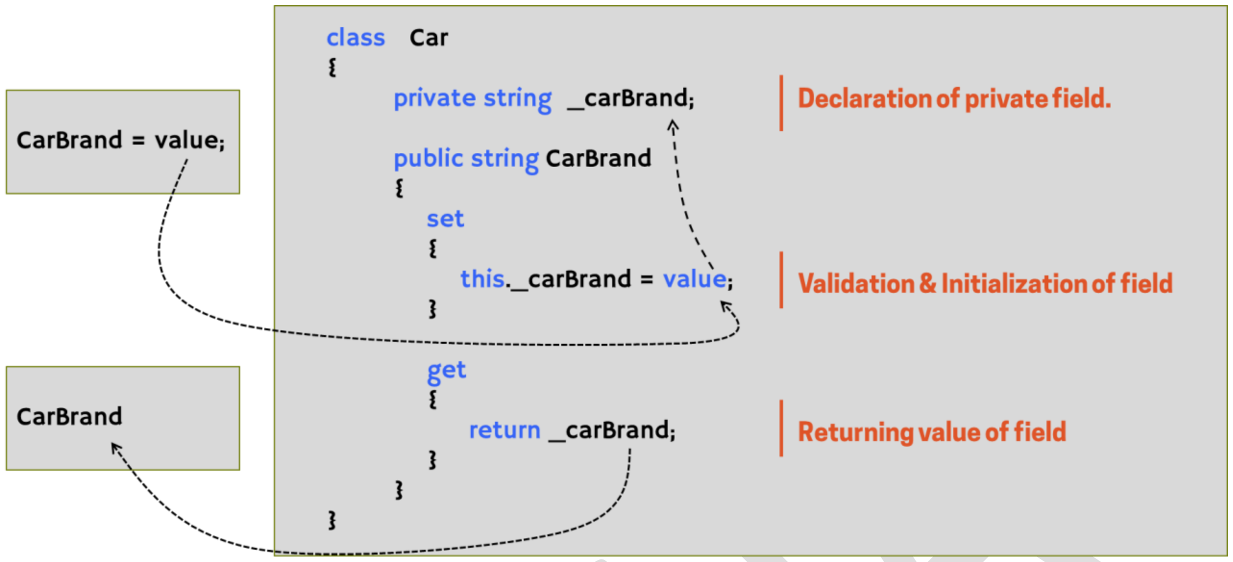
Properties

Introducing Properties

› Receive the incoming value; validate the value; assign value into field.

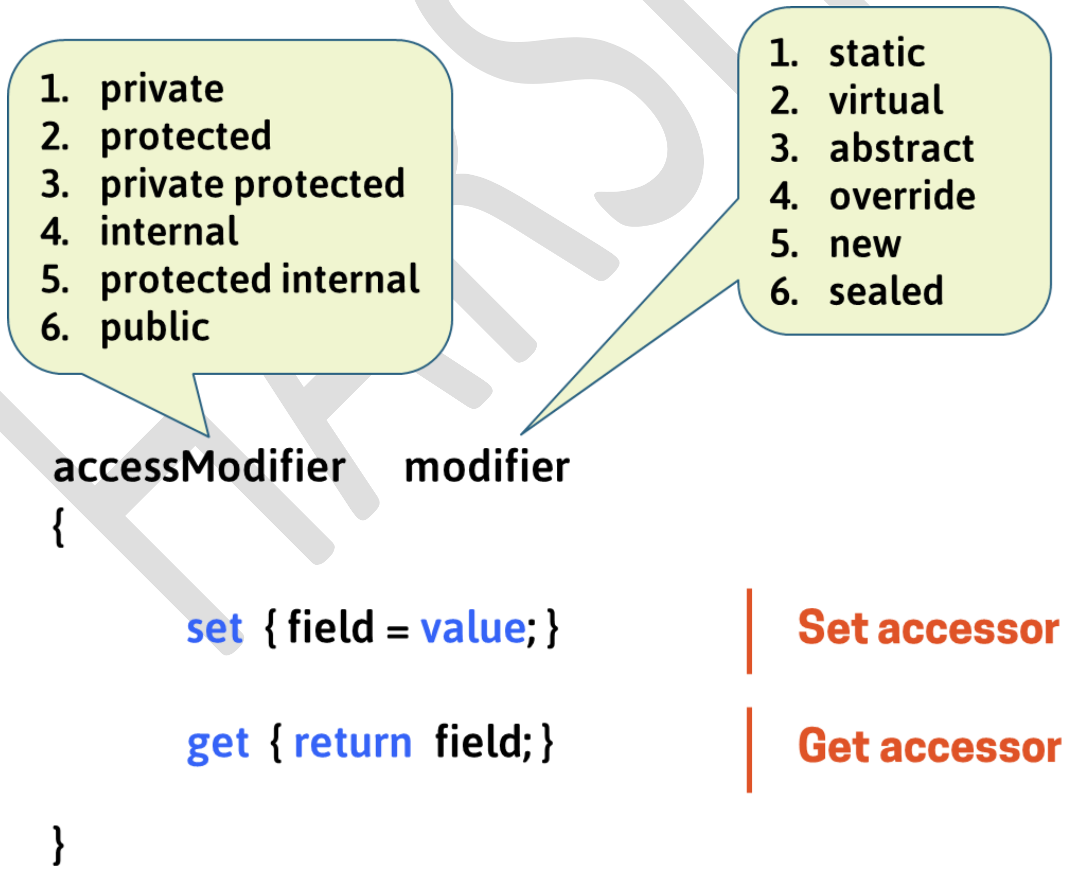


› Property is a collection of two accessors (get-accessor and set-accessor).



```
CarBrand = value;

CarBrand

class   Car
{
        private string  _carBrand;            Declaration of private field.

        public string CarBrand
        {
            set
            {
                this._carBrand = value;       Validation & Initialization of field
            }

            get
            {
                return _carBrand;             Returning value of field
            }
        }
}
```

Syntax of Property



1. private
2. protected
3. private protected
4. internal
5. protected internal
6. public

1. static
2. virtual
3. abstract
4. override
5. new
6. sealed

```
accessModifier    modifier
{

        set { field = value; }          Set accessor

        get { return  field; }          Get accessor

}
```

Set Accessor (vs) Get Accessor

Set Accessor

```
set
{
        field = value;
}
```

- Used to validate the incoming value and assign the same into field.
- Executes automatically when some value is assigned into the property.
- Has a default (implicit) parameter called "value", which represents current value i.e. assigned to the property.
- Can't have any additional parameters.
- But can't return any value.

Get Accessor

```
get
{
        return  field;
}
```

- Used to calculate value and return the same (or) return the value of field as-it-is.
- Executes automatically when the property is retrieved.
- Has no implicit parameters.
- Can't have parameters.

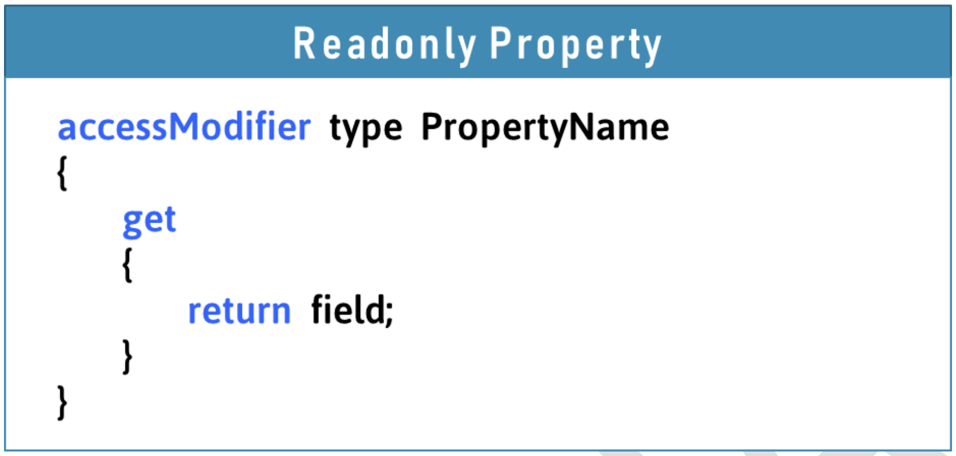
- Should return value of field.

Features and Advantages of Properties

- Properties create a protection layer around fields, preventing assignment of invalid values into properties & also do some calculation automatically when someone has invoked the property.
- No memory will be allocated for the property.
- Access modifier is applicable for the property, set accessor and get accessor individually.
  - But access modifiers of accessors must be more restrictive than access modifier of property.

```
internal   modifier   dataType   PropertyName
{
        private  set  { property = value; }

        protected  get  { return  property; }
}
```

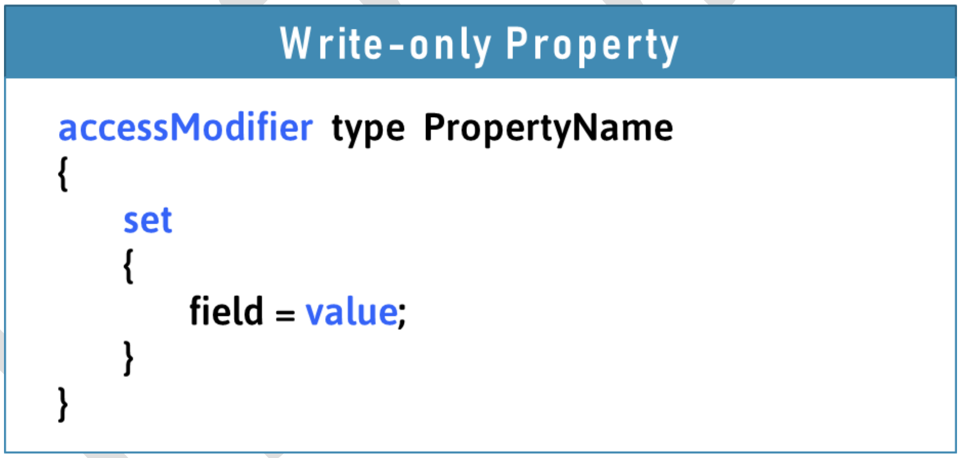
Readonly & Writeonly Property



Readonly Property

```
accessModifier type PropertyName
{
    get
    {
        return field;
    }
}
```

- Contains ONLY 'get' accessor.
- Reads & returns the value of field; but not modifies the value of field.



Write-only Property

```
accessModifier type PropertyName
{
    set
    {
        field = value;
    }
}
```

- Contains ONLY 'set' accessor.
- Validates & assign incoming value into field; but return the value.

Auto-Implemented Properties

- Property with no definition for set-accessor and get-accessor.
- Used to create property easily (with shorter syntax).

› Creates a private field (with name as _propertyName) automatically, while compilation-time.

› Auto-Implemented property can be 'Write-only Property (only set accessor)' or 'Read-only property (only set accessor).

› Useful only when you don't want to write any validation or calculation logic.

```
accessModifier   modifier type   propertyName
{

      accessModifier  set;

      accessModifer  get;

}
```

Auto-Implemented Property Initializer

› New feature in C# 6.0

› You can initialize value into auto-implemented property.

```
accessModifier   modifier   type   propertyName  { set;  get; } = value;
```

Points to Remember

› It is recommended to use Properties always in real-time projects.

  › You can also use 'Auto-implemented properties' to simplify the code.

› Properties doesn't occupy any memory (will not be stored).

› Properties forms a protection layer surrounding the private field that validates the incoming value before assigning into field.

› Read-only property has only 'get' accessor; Write-only property has only 'set' accessor.

› Properties can't have additional parameters.