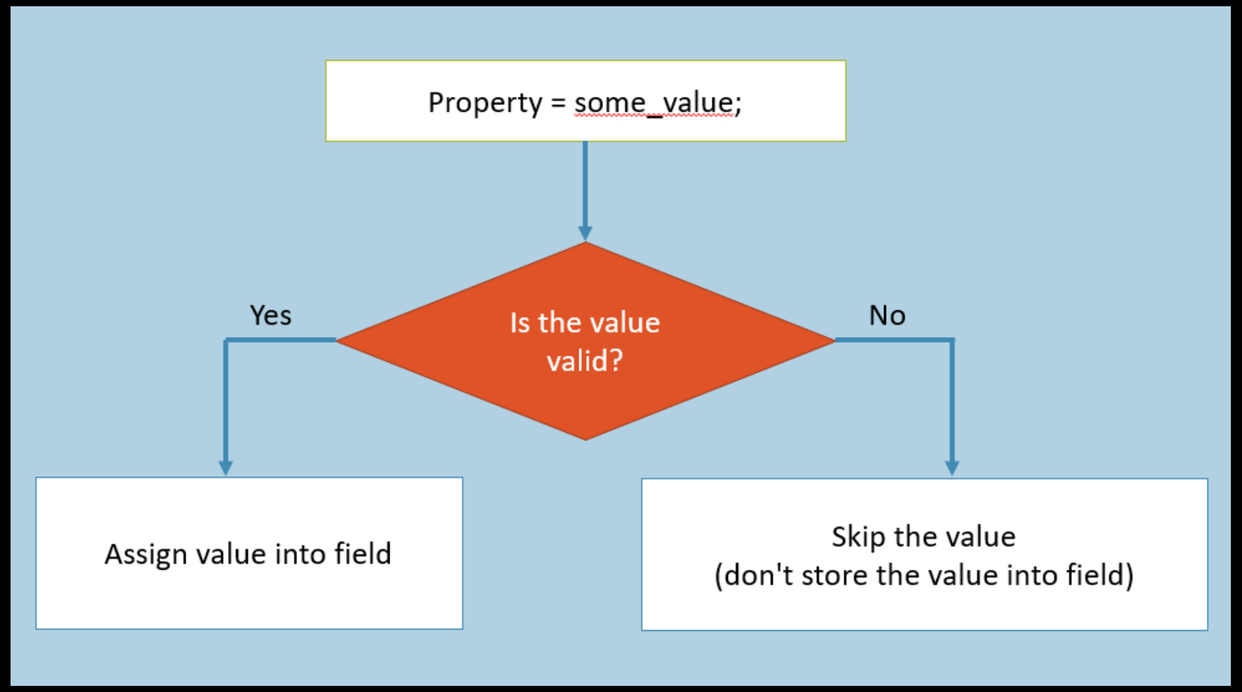
PROPERTIES

- Receive the incoming value; validate the value; assign a value into field.
- Property is a collection of two accessors (get-accessor and set-accessor).



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    type    PropertyName
{
        set  { field = value; }        Set accessor

        get  { return  field; }        Get accessor
}
```

PROPERTIES

Set Accessor [vs] Get Accessor

Set Accessor	Get Accessor
Used to validate the incoming value and assign the same into field.	Used to calculate value and return the same (or) return the value of field as-it-is.
Executes automatically when some value is assigned into the property.	Executes automatically when the property is retrieved.
Has a default (implicit) parameter called "value", which represents current value i.e. assigned to the property.	Has no implicit parameters.
Can't have any additional parameters.	Can't have parameters.
But can't return any value.	Should return value of field.

PROPERTIES

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 of accessors:

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 data_type PropertyName
{
        private set { property = value; }
        protected get { return property; }
}
```

PROPERTIES

Read-Only [vs] Write-Only Properties

Read-Only Properties	**Write-Only Properties**

accessModifier type PropertyName
{
get
{
return field;
}
}
	accessModifier type PropertyName
	{
	set
	{
	field = value;
	}
	}

Contains ONLY 'get' accessor	Contains ONLY 'set' accessor.
Reads & returns the value of field; but not modifies the value of field.	Validates & assigns incoming value into field; but doesn't return the value.

PROPERTIES

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 Read-only (only 'get' accessor) property; but it can't be Write-only (only 'set' accessor).

```
accessModifier modifier data_type PropertyName
{
    accessModifier set;
    accessModifier get;
}
```

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

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;
```

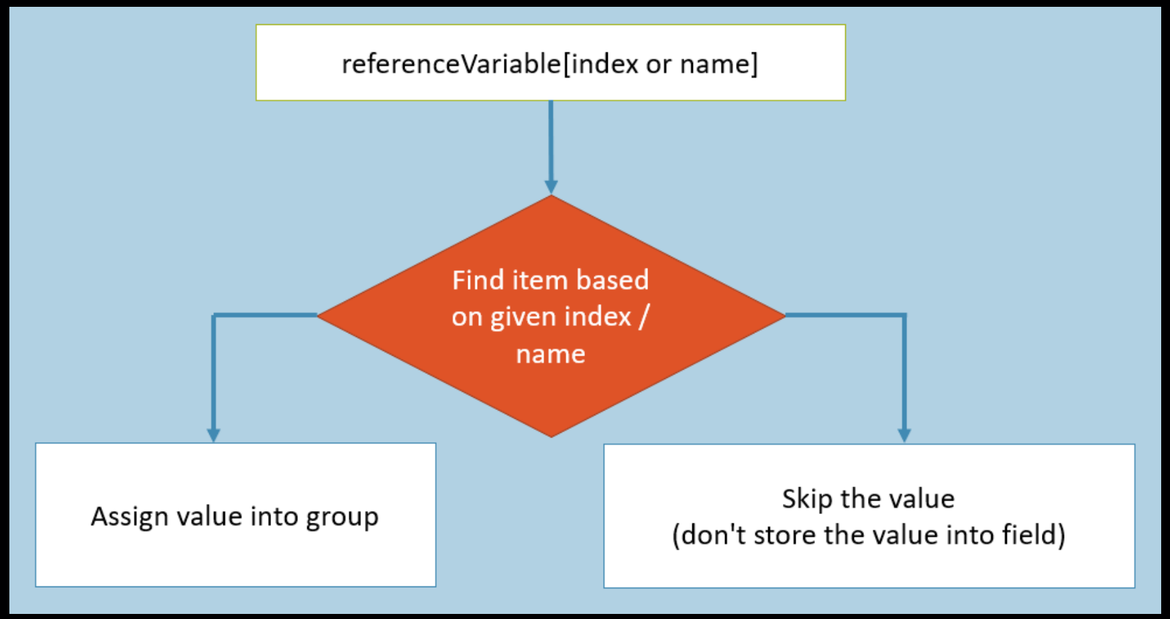
PROPERTIES

Key Points to Remember

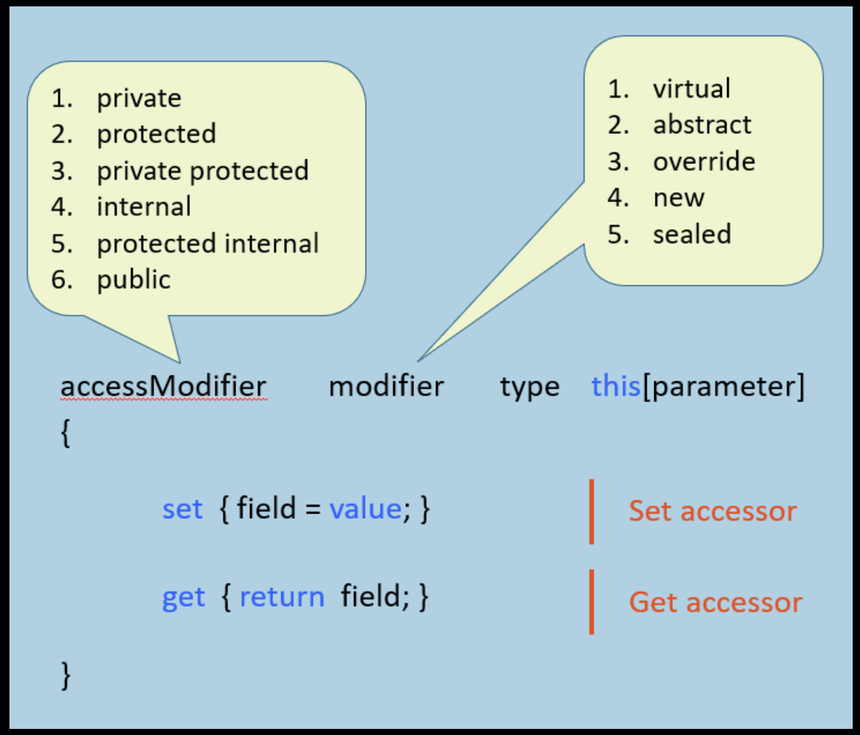
- It is recommended to use Properties always in real-time projects.
- You can also use 'Auto-implemented properties' to simplify the code.
- Properties doesn't occupy any memory (will not be stored).
- Properties forms a protection layer surrounding the private field that validates the incoming value before assigning into field.
- Read-only property has only 'get' accessor; Write-only property has only 'set' accessor.
- Properties can't have additional parameters.

INDEXERS

- Receive a number / string. Search for the particular item among a group of items; set or get value into the group of items.
- It provides shorter syntax to access a group of items
- Indexer is a special member of class, which contains set-accessor and get-accessor to access a group of items / elements.



Syntax:



INDEXERS

Key Points to Remember

- Indexers are always created with 'this' keyword.
- Indexers are generally used to access group of elements (items).
- Parameterized properties are called indexer.
- Indexers are implemented through get and set accessors along with the [ ] operator.
- Indexer must have one or more parameters.
- ref and out parameter modifiers are not permitted in indexer.
- Indexer can't be static.
- Indexer is identified by its signature (syntax of calling); where as a property is identified it's name.
- Indexer can be overloaded.