C# Indexers

An indexer is a special type of property that allows a class or a structure to be accessed like an array for its internal collection. C# allows us to define custom indexers, generic indexers, and also overload indexers.

An indexer can be defined the same way as property with `this` keyword and square brackets `[]`.

Syntax

```
<return type> this[<parameter type> index]
{
    get{
        // return the value from the specified index of an internal collection
    }
    set{
        // set values at the specified index in an internal collection
    }
}
```

The following example defines an indexer in the class.

Example: Indexer

```
class StringDataStore
{
    private string[] strArr = new string[10]; // internal data storage

    public string this[int index]
    {
        get
        {
            if (index < 0 || index >= strArr.Length)
                throw new IndexOutOfRangeException("Index out of range");

                return strArr[index];
        }

        set
        {
            if (index < 0 ||  index >= strArr.Length)
                throw new IndexOutOfRangeException("Index out of range");

            strArr[index] = value;
        }
    }
}
```

The above `StringDataStore` class defines an indexer for its private array `strArr`. So now, you can use the `StringDataStore` like an array to add and retrieve string values from `strArr`, as shown below.

Example: Indexer

```
StringDataStore strStore = new StringDataStore();
```

```
strStore[0] = "One";
strStore[1] = "Two";
strStore[2] = "Three";
strStore[3] = "Four";

for(int i = 0; i < 10 ; i++)
    Console.WriteLine(strStore[i]);
```

Output:

```
One
Two
Three
Four
```

You can use expression-bodied syntax for get and set from C# 7 onwards.

Example: Indexer

```
class StringDataStore
{
    private string[] strArr = new string[10]; // internal data storage

    public string this[int index]
    {
        get => strArr[index];

        set => strArr[index] = value;
    }
}
```

Generic Indexer

Indexer can also be generic. The following generic class includes generic indexer.

Example: Generic Indexer

```
class DataStore<T>
{
    private T[] store;

    public DataStore()
    {
        store = new T[10];
    }

    public DataStore(int length)
    {
        store = new T[length];
    }

    public T this[int index]
    {
        get
        {
            if (index < 0 &&  index >= store.Length)
                throw new IndexOutOfRangeException("Index out of range");
```

```
                return store[index];
        }

        set
        {
            if (index < 0 ||  index >= store.Length)
                throw new IndexOutOfRangeException("Index out of range");

            store[index] = value;
        }
    }

    public int Length
    {
        get
        {
            return store.Length;
        }
    }
}
```

The above generic indexer can be used with any data type. The following example demonstrates the use of generic indexer.

Example:

```
DataStore<int> grades = new DataStore<int>();
grades[0] = 100;
grades[1] = 25;
grades[2] = 34;
grades[3] = 42;
grades[4] = 12;
grades[5] = 18;
grades[6] = 2;
grades[7] = 95;
grades[8] = 75;
grades[9] = 53;

for (int i = 0; i < grades.Length;i++)
    Console.WriteLine(grades[i]);

DataStore<string> names = new DataStore<string>(5);
names[0] = "Steve";
names[1] = "Bill";
names[2] = "James";
names[3] = "Ram";
names[4] = "Andy";

for (int i = 0; i < names.Length;i++)
    Console.WriteLine(names[i]);
```

Overload Indexer

You can be overloaded with the different data types for index. The following example overloads an indexer with int type index as well as string type index.

Example: Override Indexer

```
class StringDataStore
{
    private string[] strArr = new string[10]; // internal data storage

    // int type indexer
    public string this[int index]
    {
        get
        {
            if (index < 0 || index >= strArr.Length)
                throw new IndexOutOfRangeException("Index out of range");

            return strArr[index];
        }

        set
        {
            if (index < 0 || index >= strArr.Length)
                throw new IndexOutOfRangeException("Index out of range");

            strArr[index] = value;
        }
    }

    // string type indexer
    public string this[string name]
    {
        get
        {
            foreach (string str in strArr){
                if(str.ToLower() == name.ToLower())
                    return str;
                }

            return null;
        }
    }
}

class Program
{
    static void Main(string[] args)
    {
        StringDataStore strStore = new StringDataStore();

        strStore[0] = "One";
        strStore[1] = "Two";
        strStore[2] = "Three";
        strStore[3] = "Four";

        Console.WriteLine(strStore["one"]);
        Console.WriteLine(strStore["two"]);
        Console.WriteLine(strStore["Three"]);
        Console.WriteLine(strStore["Four"]);
    }
}
```

Note:
Indexer does not allow ref and out parameters.