W2 Discussion

Eric Le

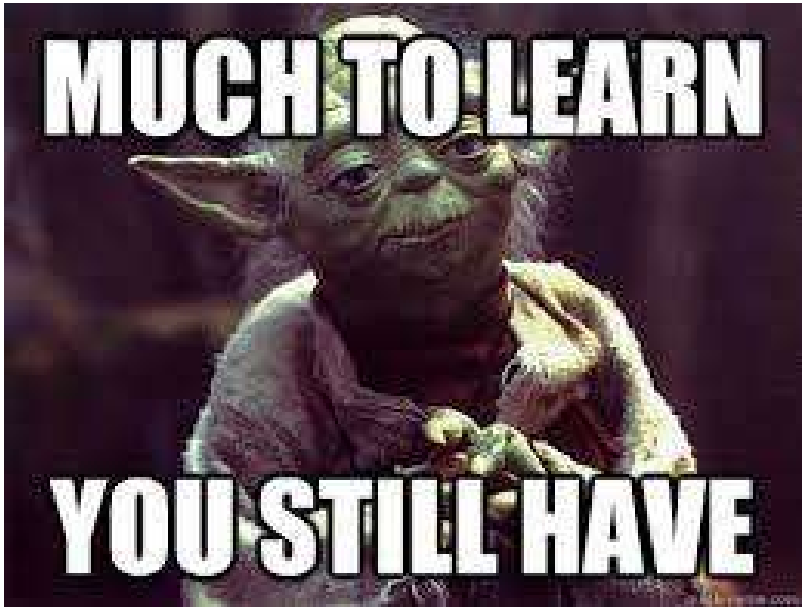
Last Week We Learn

```
...
public class semester
{
    private int duration;

    public semester()
    {
        duration = 144;
    }

    public void setDuration(int newDuration)
    {
        duration = newDuration;
    }
}
...
```

Source: [Constructors in C# | Beginners Guide to C# (wordpress.com)]

This Week Let’s learn

- Collections and Data Structures
- Unit Test
- UML Diagram
- More things on fields and property…



Data Structures

- A data structure…

Data Structures

- A data structure…

refers to a container designed specifically for arranging, manipulating, retrieving, and storing data.

Source: memechat.app

Data Structures

- Discussions:
  - Why do we need to know about Data Structures?
  - Can you compare some types of Data Structures?

C# collections

What are C# collections?

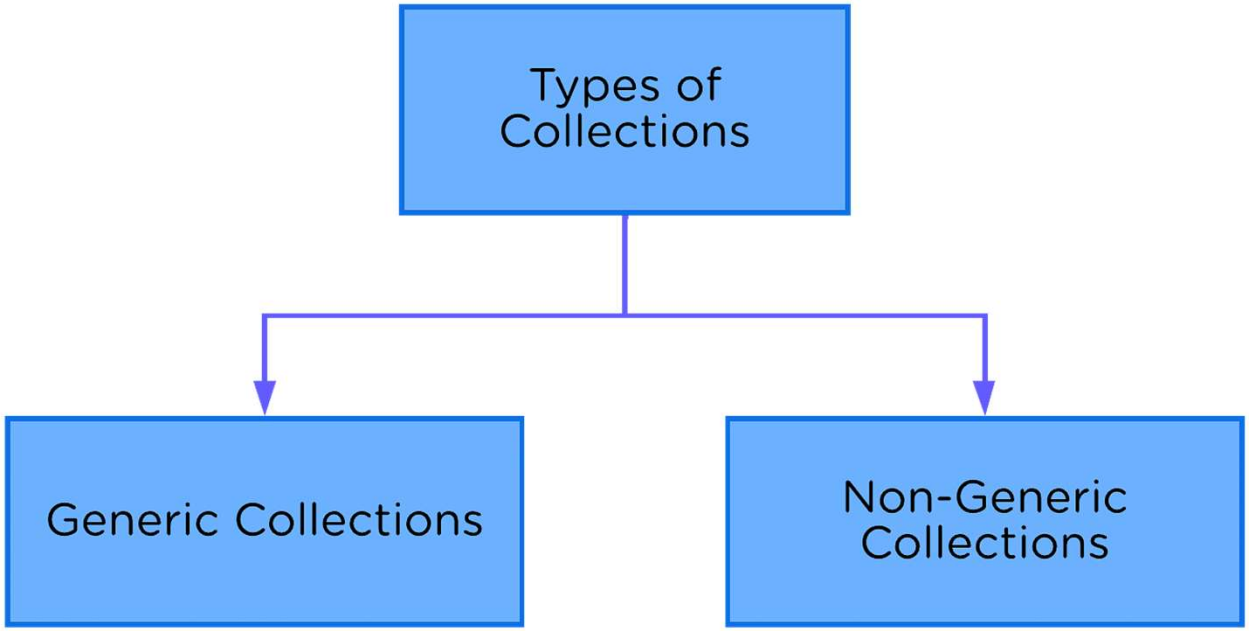
Collections provide a more flexible way to work with groups of objects.

You can organize objects (add, remove, sort…) and manage access to these objects with a Collection.

Two types of collections: Generic vs Non-generic

Discussion: What is the main differences between them?

C# collections



C# collections

System.Collections.Generic Classes

You can create a generic collection by using one of the classes in the System.Collections.Generic namespace. A generic collection is useful when every item in the collection has the same data type. A generic collection enforces strong typing by allowing only the desired data type to be added.

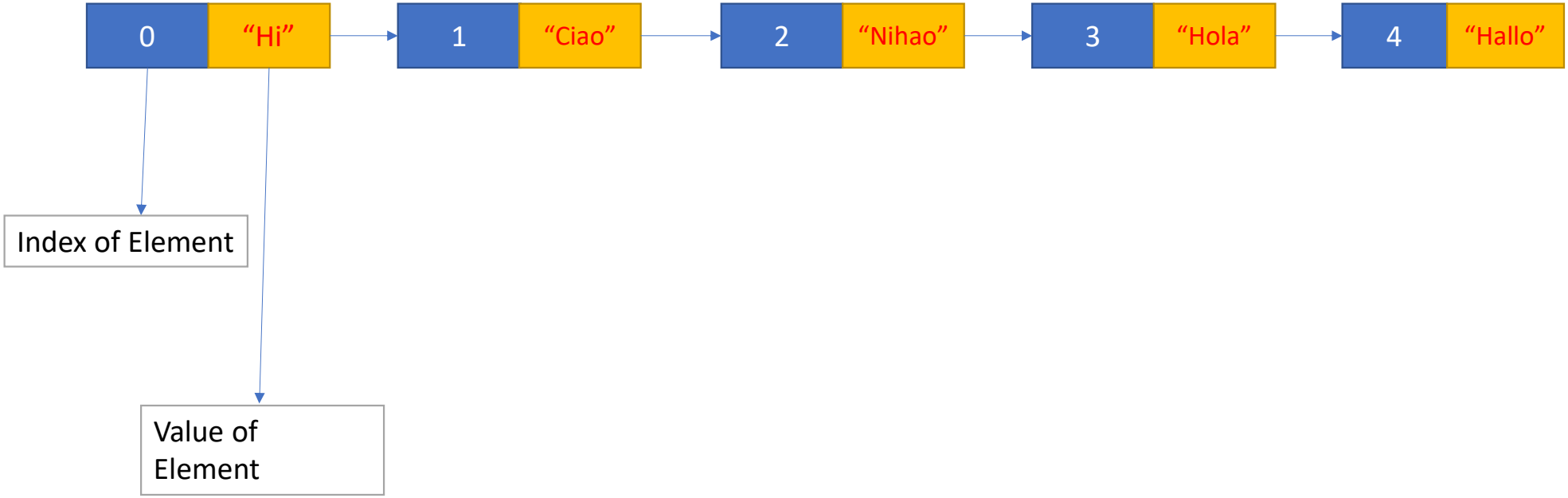
The following table lists some of the frequently used classes of the System.Collections.Generic namespace:

Class	Description
Dictionary<TKey,TValue>	Represents a collection of key/value pairs that are organized based on the key.
List<T>	Represents a list of objects that can be accessed by index. Provides methods to search, sort, and modify lists.
Queue<T>	Represents a first in, first out (FIFO) collection of objects.
SortedList<TKey,TValue>	Represents a collection of key/value pairs that are sorted by key based on the associated IComparer<T> implementation.
Stack<T>	Represents a last in, first out (LIFO) collection of objects.

What we will learn today!

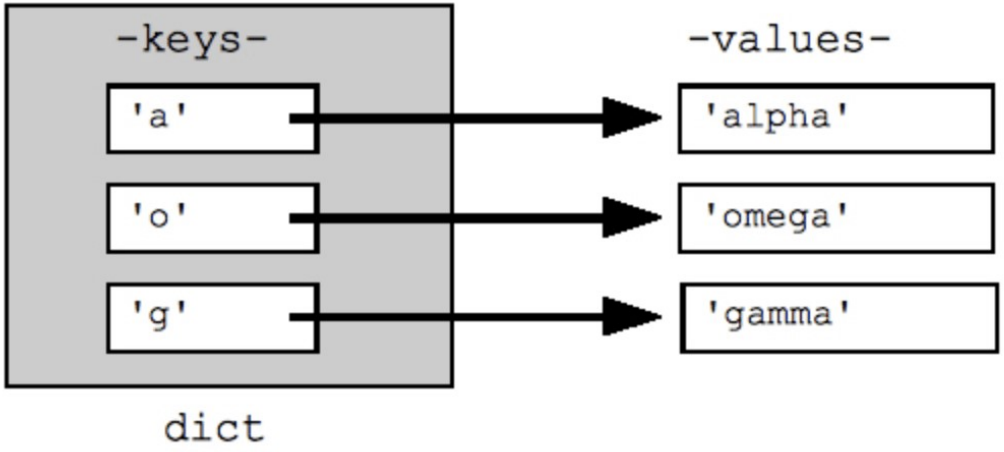
List vs Dictionary

- List stores data in an ordered and sequential manner.



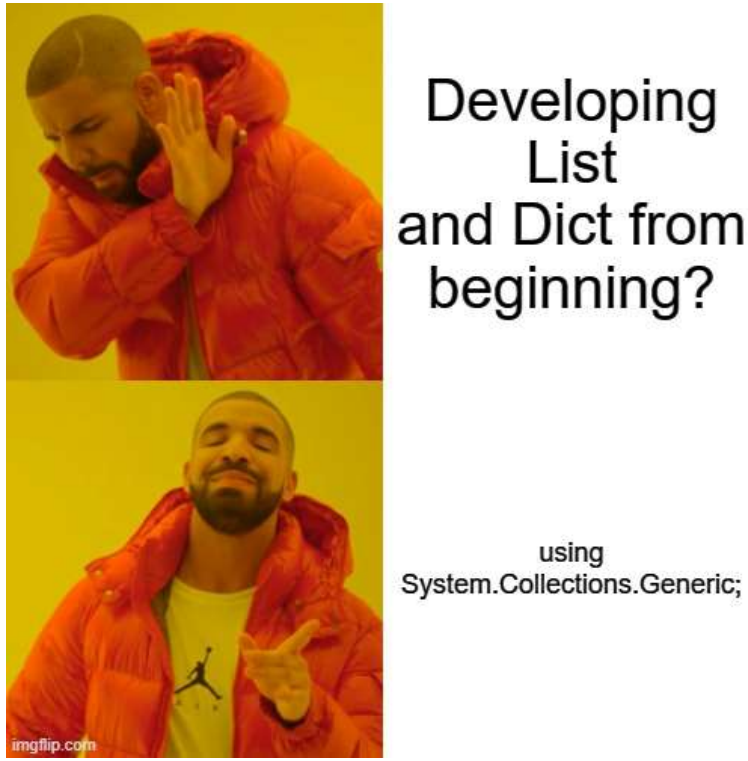
List vs Dictionary

- The Dictionary is a generic collection that stores key-value pairs in no particular order.



Source: Datacamp.com

List vs Dictionary: How to use



List vs Dictionary: How to use

- Creating a List

```
// Dynamic ArrayList with no size limit
List<int> numberList = new List<int>();
numberList.Add(32);
numberList.Add(21);
numberList.Add(45);
numberList.Add(11);
numberList.Add(89);
// List of string
List<string> authors = new List<string>(5);
authors.Add("Mahesh Chand");
authors.Add("Chris Love");
authors.Add("Allen O'neill");
authors.Add("Naveen Sharma");
authors.Add("Monica Rathbun");
authors.Add("David McCarter");
```

Source: [C# List Tutorial - Everything You Need To Learn About List In C# (c-sharpcorner.com)]

List vs Dictionary: How to use

- Creating a Dict

```
Dictionary<string, string> EmployeeList = new Dictionary<string, string>();
```

The following code snippet adds items to the Dictionary.

```
EmployeeList.Add("Mahesh Chand", "Programmer");
EmployeeList.Add("Praveen Kumar", "Project Manager");
EmployeeList.Add("Raj Kumar", "Architect");
EmployeeList.Add("Nipun Tomar", "Asst. Project Manager");
EmployeeList.Add("Dinesh Beniwal", "Manager");
```

Source: [Dictionary In C# (c-sharpcorner.com)]

Some methods for both…

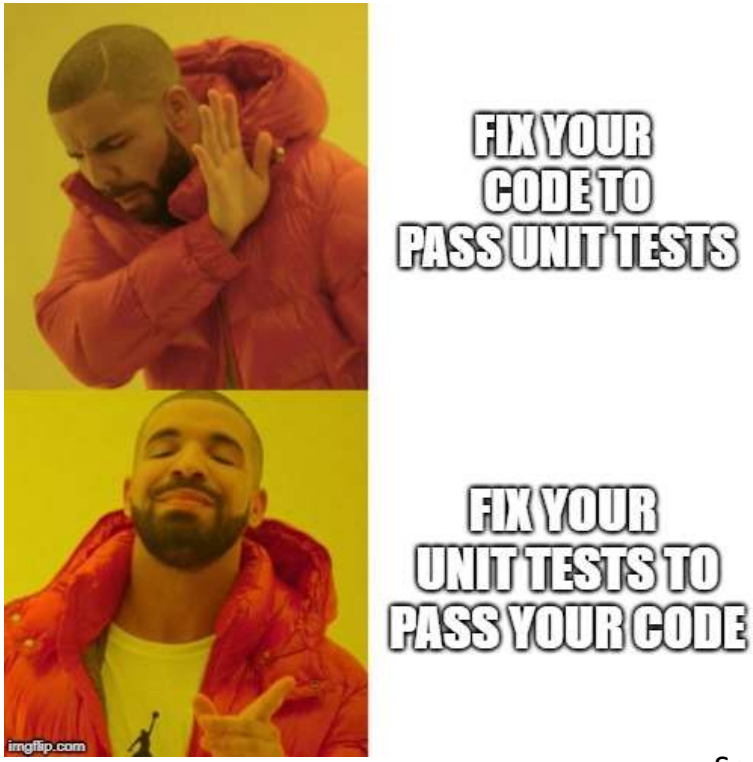
- Add()
- Remove()
- Clear()
- IndexOf()
- Contains()
- Insert()

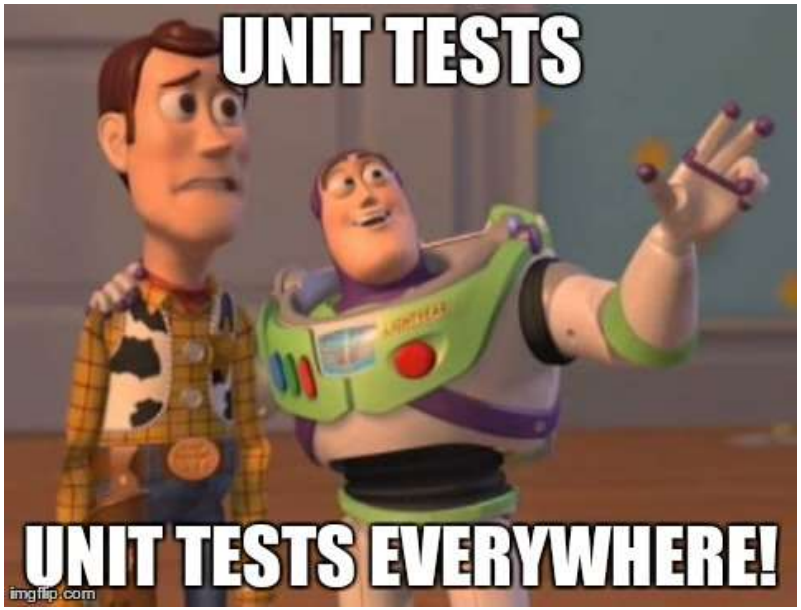
- Add()
- Remove()
- Clear()
- ContainsKey
- ContainsValue
- TryGetValue

Unit Test

- Discussion: What is unit test?
- Why do we need unit test?
- Do and Don’t in Writing unit test?

Unit Test





Source: ProgrammerHumor.io

Unit Test

Creating the first test

You write one failing test, make it pass, and then repeat the process. In the *PrimeService.Tests* directory, rename the *UnitTest1.cs* file to *PrimeService_IsPrimeShould.cs* and replace its entire contents with the following code:

C# Copy

```
using NUnit.Framework;
using Prime.Services;

namespace Prime.UnitTests.Services
{
    [TestFixture]
    public class PrimeService_IsPrimeShould
    {
        private PrimeService _primeService;

        [SetUp]
        public void SetUp()
        {
            _primeService = new PrimeService();
        }

        [Test]
        public void IsPrime_InputIs1_ReturnFalse()
        {
            var result = _primeService.IsPrime(1);

            Assert.IsFalse(result, "1 should not be prime");
        }
    }
}
```

:e: [Unit testing C# with NUnit and .NET Core - | Microsoft Learn]

NUnit

Source: Twilio.io

Nunit allows you to parameterize your test

```
public class CoordinateValidator
{
    public bool IsLatitudeValid(double latitude)
    {
        return latitude is >= -90 and <= 90;
    }

    public bool IsLongitudeValid(double longitude)
    {
        return longitude is >= -180 and <= 180;
    }
}
```

We can use the following test method:

```
    [TestCase(-90, true)]
    [TestCase(0, true)]
    [TestCase(10, true)]
    [TestCase(90, true)]
    [TestCase(-91, false)]
    [TestCase(91, false)]
    public void TestLatitude(double latitude, bool expected)
    {
        // ARRANGE
        var validator = new CoordinateValidator();

        // ACT
        bool valid = validator.IsLatitudeValid(latitude);

        // ASSERT
        Assert.AreEqual(expected, valid);
    }
```

[c# - Is it possible to parameterize a nunit test - Stack Overflow]

Property

- Sometimes, users or your mates may set a wrong value for a field…
- For example:

A student class who have ID, pass mark , name.Now in this example some problem with public field

ID should not be negative.

Name can not be set to null

Pass mark should be read only.

If student name is missing No Name should be return.

Property

- Sometimes, users or your mates may set a wrong value for a field…
- For example:

ID should not be negative.

Name can not be set to null

Pass mark should be read only.

If student name is missing No Name should be return.

How to prevent these things?

Property

- Sometimes, users or your mates may set a wrong value for a field…
- For example:

ID should not be negative.

Name can not be set to null

Pass mark should be read only.

If student name is missing No Name should be return.

How to prevent these things? PROPERTY

Property

Discussion:

Difference between properties and fields?

Any advantages of using property?

Property: How to use

```
public class Student
{
private int _id;
private string _name;
public int Id { get { return _id; } set { _id = value; } }
public string Name { get { return _name; } set { _name
= value; } }
}
```

Minding the name: get and set;

Property: How to use

```
public int Age
{
  get; // This is auto-implemented
  set  // This is not auto-implemented. Error!
  {
    ...
  }
}

/*-----------------------------------------------------------------------*/

public int Age { get; set; } // Getter and setter auto-implemented. Correct.

/*-----------------------------------------------------------------------*/

private int _age;
public int Age
{
  get             // This is custom, not auto-implemented
  {
    return _age;
  }
  set             // This is custom, not auto-implemented. Correct
  {
    _age = value;
  }
}
```

Reference: [Learn C# Properties: Getters and Setters at Intermediate C# Course (codeeasy.io)]