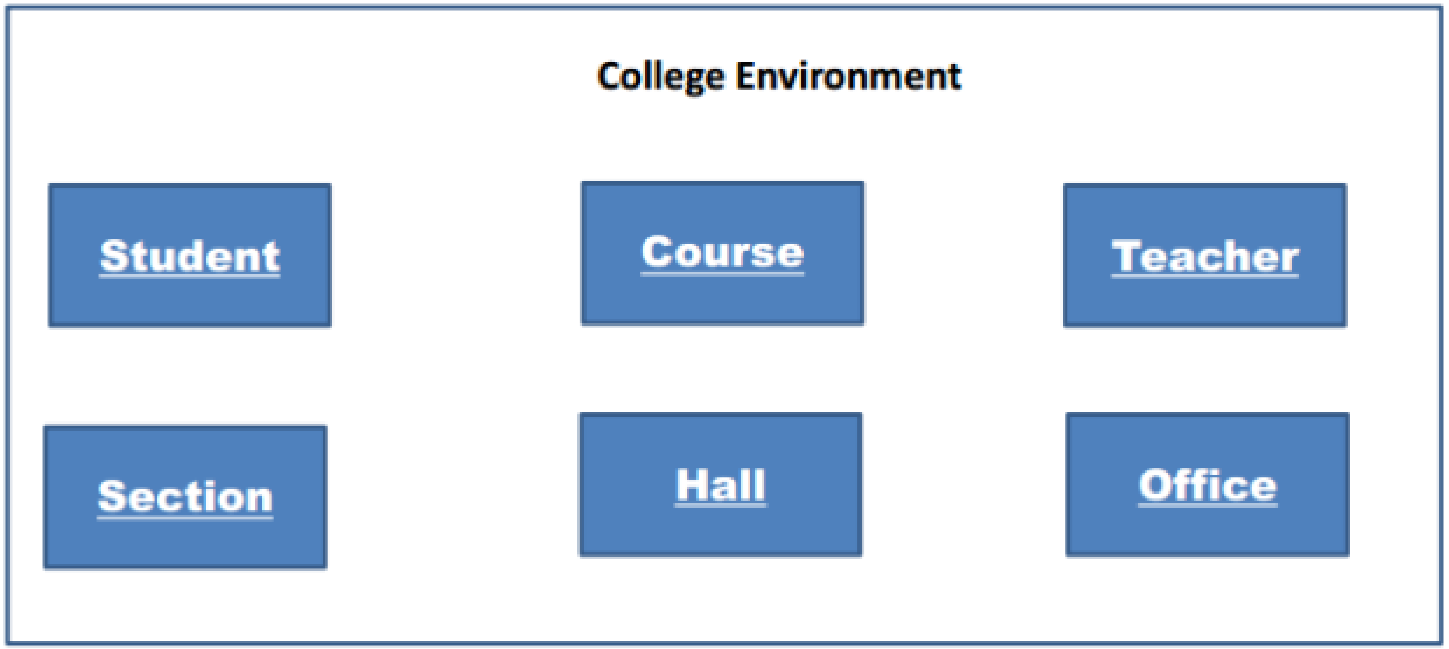
OBJECT-ORIENTED PROGRAMMING

LAB3 :OOP_CONCEPTS

OOP Concepts

Objects in College Management Program



OOP Concepts

Example1:

- ❑ Write a class named **Student** that has four fields: **name**, **age**, **dept**, and **level**.
- ❑ Overload the constructor .
- ❑The default constructor should print this message **"Welcome"** and set these default values to the class fields: **"Unknown"**, **20**, **"CS"**, **1**
- ❑And the parameterized constructor.
- ❑The Get_Information is method print all information about student.
- ❑Add new static field NumOfStudents.
- ❑The Destructor should print this message ”The object {this} is deleted”

OOP Concepts

Class Student

Data

1. student_name
2. student_ID
3. student_Age
4. student_Dept
5. student_Level

Operation()

1. Student()
2. Student(name,ID,age.dept,level)
3. Get_Information()

Student1

Data

1. student_name
2. student_ID
3. student_Age
4. student_Dept
5. student_Level

Operation()

1. Student()
2. Student(name,ID,age.dept,level)
3. Get_Information()

Student2

Data

1. student_name
2. student_ID
3. student_Age
4. student_Dept
5. student_Level

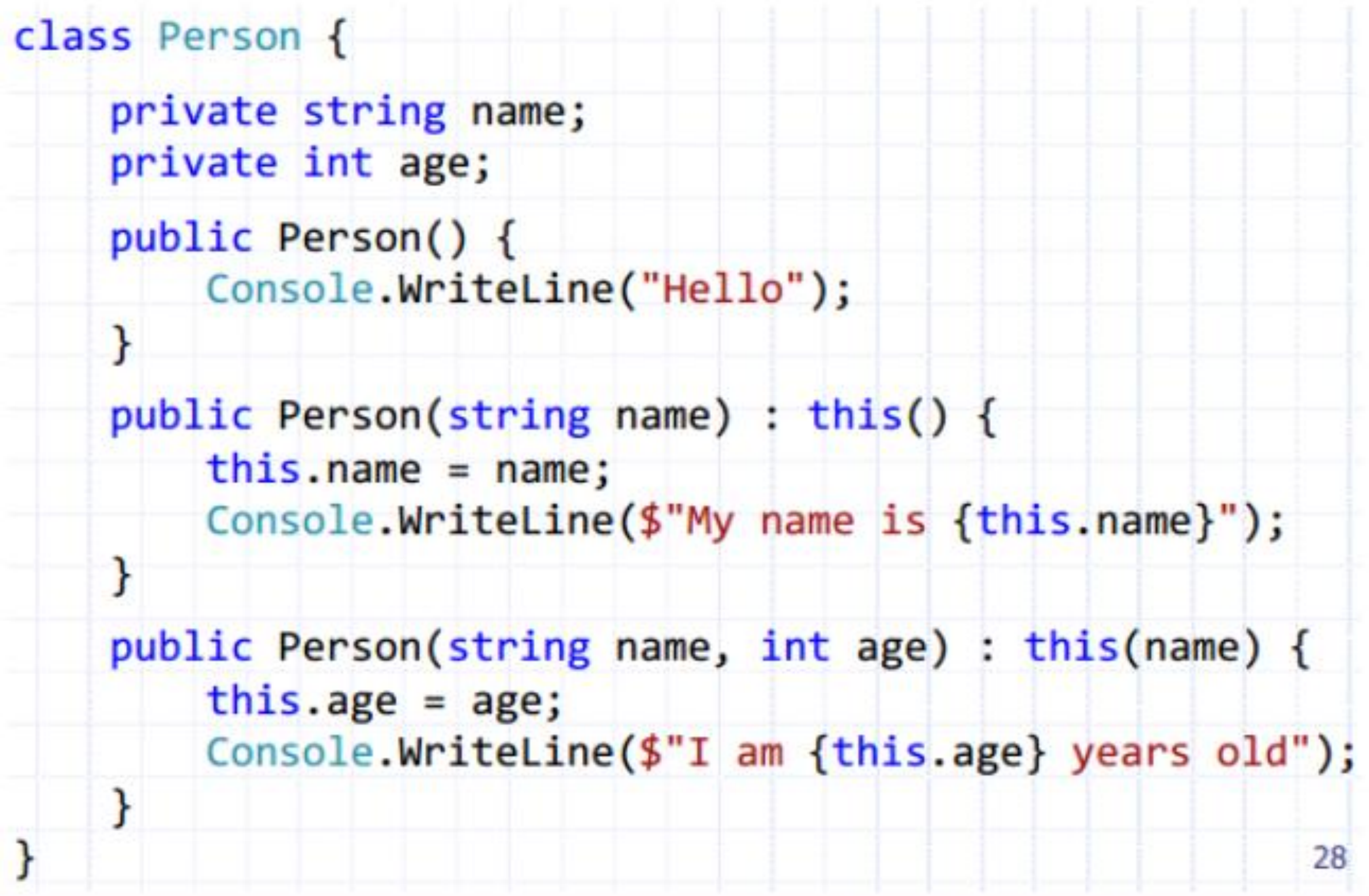
Operation()

1. Student()
2. Student(name,ID,age.dept,level)
3. Get_Information()

OOP Concepts

- في هذا المثال نحاول نطبق المفاهيم الأساسية بمعنى اول ننشأ الفصل من غير دالة بناءة ثم نعمل Defaultثم نضيف parameterizedثم نلغي الأثنين و ننشأ واحدة privateو ناكد ان المعالج لم يعطي لنا default
- نناقش الكلمة   this متى تكون مطلوبة و الى ماذا تشير
- استخدام   this في chaining technique
- نضيف حقل staticوهو عبارة عن عدد الطلاب الذين تم إنشاؤهم (نناقش عملية استخدامه)
- نحاول نطبق destructorsباستخدام ;()GC.collector

❑ Using Constructor Chaining



```
class Person {
    private string name;
    private int age;
    public Person() {
        Console.WriteLine("Hello");
    }
    public Person(string name) : this() {
        this.name = name;
        Console.WriteLine($"My name is {this.name}");
    }
    public Person(string name, int age) : this(name) {
        this.age = age;
        Console.WriteLine($"I am {this.age} years old");
    }
}
```

OOP Concepts

Exercise3.1

1. Write a class named **Employee** that has three fields: **name**, **salary**, and **address**.
2. The default constructor should print this message **"Welcome to our company"** and set these default values to the class fields: **"Unknown"**, **30000**, **"Mukalla”**

OOP Concepts

Exercise 3.2

1. Create a Console application;
2. Create a class named Client;
3. Add the following properties to your Client class : FirstName, LastName, Email, Address, PhoneNumber;
4. Create a parameter and a default constructor for the client class

OOP Concepts

```
Please enter the client first name:
Ahmed
Please enter the client last name:
Salem
Please enter his/her email:
test@test.com
Please enter his/her address:
56000 test rd.
finally enter the phone number:
888888888

Client information:
FirstName: Ahmed
LastName: Salem
Email: test@test.com
Address: 56000 test rd.
PhoneNumber: 8888888888
```

```
using System.Text.RegularExpressions;
namespace Exercise5
{
    public static class Validator
    {
        public static bool FieldIsEmpty(string Value)
        {
            if (String.IsNullOrEmpty(Value))
                return true;
            return false;
        }
        public static bool PhoneNumberIsValid(string PhoneNumber)
        {
            if (PhoneNumber.Length == 10)
                return true;
            return false;
        }

        public static bool EmailIsValid(string Email)
        {
            if (Regex.IsMatch(Email, "^[^@]+@[^@]+\\.[^@]+$"))
                return true;
            return false;
        }
    }
}
```

```
namespace Exercise5
{
   public class Client
   {
      public string FirstName { get; set; }
      public string LastName { get; set; }
      public string Email { get; set; }
      public string Address { get; set; }
      public string PhoneNumber { get; set; }

      public Client(string FirstName,
                    string LastName,
                    string Email,
                    string Address,
                    string PhoneNumber)
      {
         this.FirstName      = FirstName;
         this.LastName       = LastName;
         this.Email          = Email;
         this.Address        = Address;
         this.PhoneNumber    = PhoneNumber;
      }
       public Client()
      {
         FirstName   = "Alexandre";
         LastName    = "Sauve";
         Email       = "Test@hotmail.com";
         Address     = "450 Test St.";
         PhoneNumber = "2833939292";
      }

      public override string ToString()
      {
         return "FirstName: " + FirstName + "\n" +
             "LastName: " + LastName + "\n" +
             "Email: " + Email + "\n" +  "Address: " + Address + "\n" +
             "PhoneNumber: " + PhoneNumber;
      }
   }
}
```

```
using System;

namespace Exercise5
{
  class Program
  {
    static void Main(string[] args)
    {
      //Validate and store all client input values
      string FirstName = String.Empty;
      while(Validator.FieldIsEmpty(FirstName))
      {
        Console.WriteLine("Please enter the client first name: ");
        FirstName = Console.ReadLine();
      }

      string LastName = String.Empty;
      while (Validator.FieldIsEmpty(LastName))
      {
        Console.WriteLine("Please enter the client last name: ");
        LastName = Console.ReadLine();
      }

      string Email = String.Empty;
      while (Validator.FieldIsEmpty(Email) ||
          !Validator.EmailIsValid(Email))
      {
        Console.WriteLine("Please enter the client email: ");
        Email = Console.ReadLine();
      }

      string Address = String.Empty;
      while (Validator.FieldIsEmpty(Address))
      {
        Console.WriteLine("Please enter the client address: ");
        Address = Console.ReadLine();
      }

      string PhoneNumber = String.Empty;
      while (Validator.FieldIsEmpty(PhoneNumber) ||
         !Validator.PhoneNumberIsValid(PhoneNumber))
      {
        Console.WriteLine("Please enter the client phone number: ");
        PhoneNumber = Console.ReadLine();
      }

      //At this point we have all the required information, we can create our object!
      Client NewClient = new Client(FirstName,
                     LastName,
                     Email,
                     Address,
```

OBJECT-ORIENTED PROGRAMMING

LAB4 :OOP_CONCEPTS

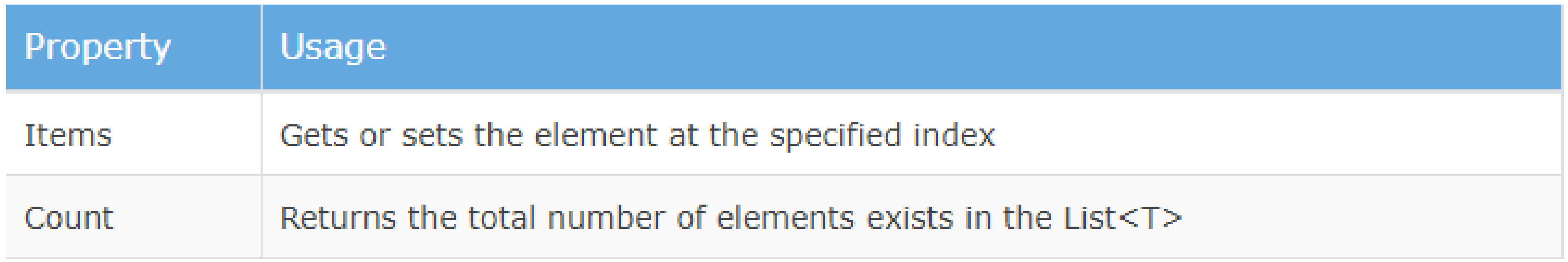
Generic List<T> Collection Class in C#

- This Generic List<T> Collection Class represents a strongly typed list of objects which can be accessed by using the integer index which is starting from 0.
- It also provides lots of methods that can be used for searching, sorting, and manipulating the list of items
- Syntax

```
List<string> countries = new List<string>();
```

Generic List<T> Collection Class in C#

List<T> Class Properties

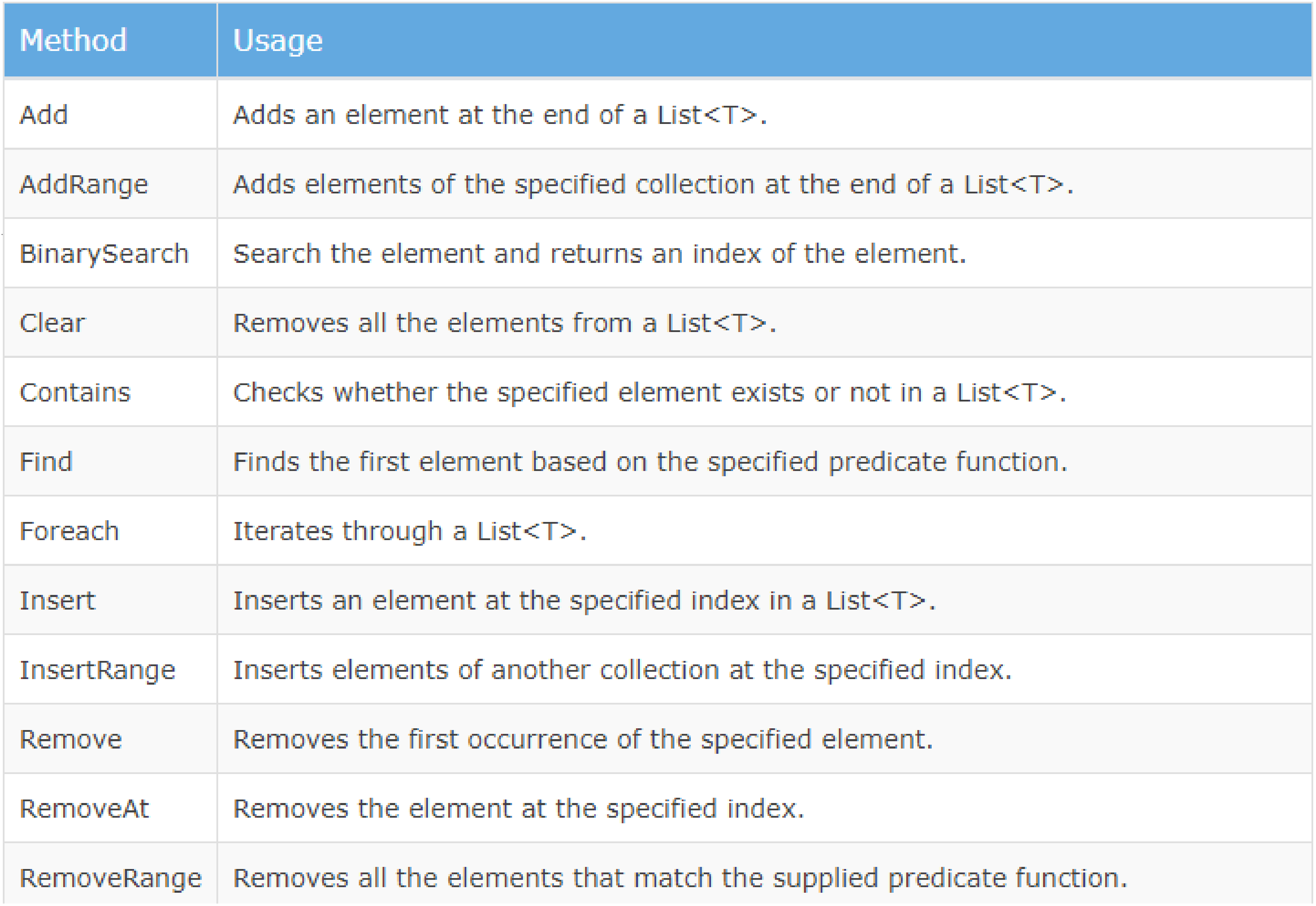


Property	Usage
Items	Gets or sets the element at the specified index
Count	Returns the total number of elements exists in the List<T>

Generic List<T> Collection Class in C#

List<T> Class Methods

- **Add(T)** This method is used to add an object at the end of the list. ...
- **Clear()** This method is used to remove all the elements from the list. ...
- **Insert(Int32, T)** This method is used to insert an element at the specified position in the list. ...
- RemoveAt**(Int32)** ...
- **Sort()**



Method	Usage
Add	Adds an element at the end of a List<T>.
AddRange	Adds elements of the specified collection at the end of a List<T>.
BinarySearch	Search the element and returns an index of the element.
Clear	Removes all the elements from a List<T>.
Contains	Checks whether the specified element exists or not in a List<T>.
Find	Finds the first element based on the specified predicate function.
Foreach	Iterates through a List<T>.
Insert	Inserts an element at the specified index in a List<T>.
InsertRange	Inserts elements of another collection at the specified index.
Remove	Removes the first occurrence of the specified element.
RemoveAt	Removes the element at the specified index.
RemoveRange	Removes all the elements that match the supplied predicate function.

OOP Concepts

Exercise 4.1

First, write a **class** called E**mployee** that has five fields: **ID** , **Name** , **Gender Salary**;

In implementation

- Create 5 employee objects
- Create a generic list collection of type employee and store all the employee objects in that collection.
- Retrieving All Employees of Generic List Collection using For Each loop and print out
- Inserting an Employee into the Index Position 1
- If you want to get the index position of a specific employee then use Indexof() method
- deletes a specific E**mployee** using id.

```
//If you want to get the index postion of a specific employee then use Indexof() method as follows

Console.WriteLine("\nIndex of emp3 object in the List = " + listEmployees.IndexOf(emp3));

Consc
}
```

https://unaura.com/cvisual-basic-exercise-5-all-about-classes/

http://unaura.com/cvisual-basic-exercise-5-all-about-classes/

OBJECT-ORIENTED PROGRAMMING

LAB5 :OOP_CONCEPTS

Exercise 5.1

Create a class file with the name SavingAccount

creating three private variables

Balance variable is used to hold the account balance of the user

PerDayWithdrawLimit is sued to restrict the user withdrawal i.e. we are setting 10000 per day

TodayWithdrawal variable is used to hold the current day withdrawal amount.

Exercise 5.1

DepositAmount: This method takes the amount to be deposited and the logic is very straightforward. Whatever amount we are getting, we are just adding the amount with the Balance private variable.

WithdrawAmount: In this method, we have written three pieces of logic. First, we are checking whether the withdrawal amount is less than the available balance or not. Second, we are checking whether the withdrawal amount exceeds the per-day withdrawal limit or not. In these two conditions are satisfied, then we are withdrawing the amount and returning true to the user.

CheckBalance: This method implementation is very straightforward; whatever value we have in the Balance variable we are simply returning.

The complete class code is given below.