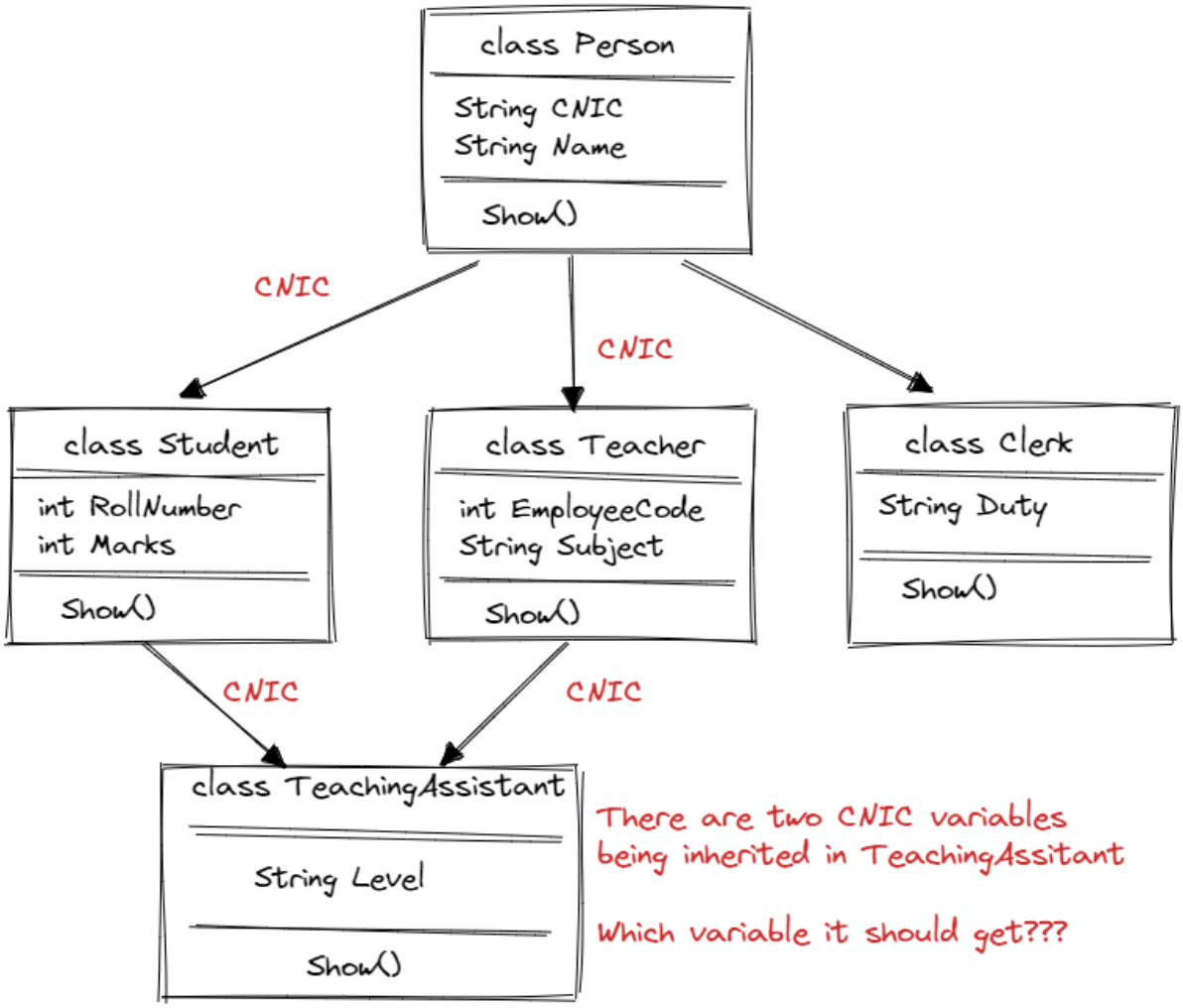
13 - Interfaces

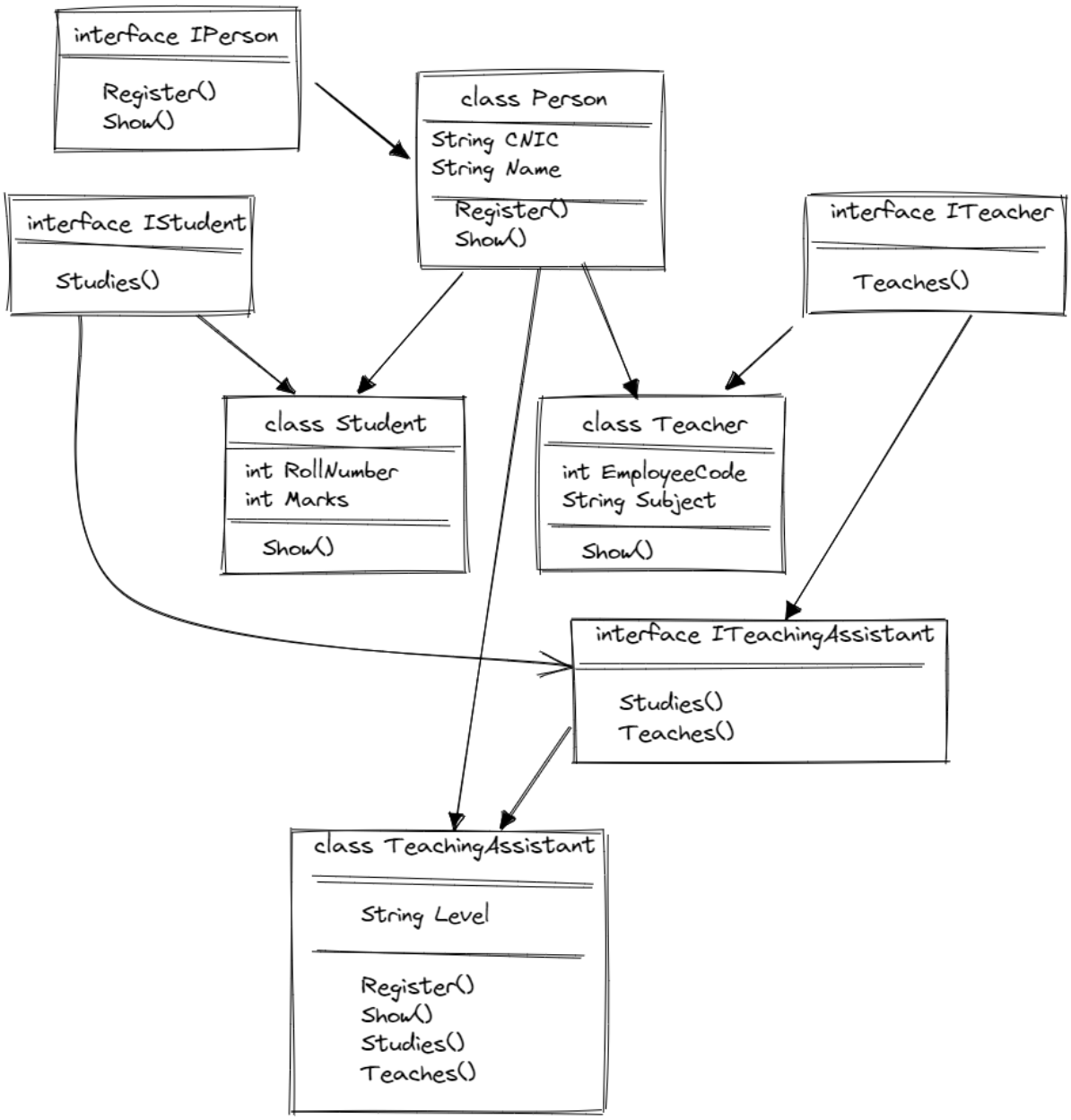
Suppose there is a TeachingAssistant who is Student as well as Teacher. So Teaching Assistant wants to inherit from Student and Teacher. Both Student and Teacher inherit from Person and both have separate copy of variable CNIC. When Teaching Assistant wants to inherit from Student and Teacher, it does not know which copy of variable CNIC it should take. It creates ambiguity. So C# does not allow multiple inheritance.



C# provides Interfaces to achieve multiple inheritance.

1. Interface is blue-print of a class and we only list down method signatures.
2. Any class can implement interface. The class that implement an interface, it should add implementation of all the functions that are written in the interface.
3. We can create hierarchy of interfaces

4. We cannot create objects of interfaces, so it is similar to abstract class
5. We can save object of a class in an interface that implements that interface




```
namespace _20220613_Interfaces.Interfaces
{
    public interface IPerson
    {
        void Register();
        void Show();
    }

    public interface IStudent : IPerson
    {
        void Studies();
    }
```

```
    public interface ITeacher : IPerson
    {
        void Teaches();
    }

    // Any class that implements this interface must write code for these functions
    // Register and Show from IPerson
    // Studies from IStudent
    // Teaches from ITeacher
    // MarksAssignments from this interface
    public interface ITeachingAssistant : IStudent, ITeacher
    {
        void MarksAssignments();
    }
}
```

Implementation classes will be like

```
using System;

namespace _20220613_Interfaces.Interfaces
{
    public class Person : IPerson
    {
        public String CNIC { get; set; }
        public String Name { get; set; }

        public Person(string cNIC, string name)
        {
            CNIC = cNIC;
            Name = name;
        }

        public virtual void Show()
        {
            Console.WriteLine("CNIC: \t\t" + CNIC);
            Console.WriteLine("Name: \t\t" + Name);
        }

        public void Register()
        {
            throw new NotImplementedException();
        }
    }

    public class Student : Person, IStudent
    {
        public int RollNumber { get; set; }
        public int Marks { get; set; }

        public Student(string cNIC, string name, int rollNumber, int marks) : base(cNIC, name)
        {
            RollNumber = rollNumber;
            Marks = marks;
        }

        public override void Show()
        {
            base.Show();
            Console.WriteLine("Roll Number: \t" + RollNumber);
            Console.WriteLine("Marks: \t\t" + Marks);
        }

        public void Studies()
        {
            throw new NotImplementedException();
        }
    }
```

```
    public class Teacher : Person, ITeacher
    {
        public int EmployeeCode { get; set; }
        public String Subject { get; set; }

        public Teacher(string cNIC, string name, int employeeCode, string subject) : base(cNIC, name)
        {
            EmployeeCode = employeeCode;
            Subject = subject;
        }

        public override void Show()
        {
            base.Show();
            Console.WriteLine("Employee Code: \t" + EmployeeCode);
            Console.WriteLine("Subject: \t" + Subject);
        }

        public void Teaches()
        {
            throw new NotImplementedException();
        }
    }

    public class TeachingAssistant : Person, ITeachingAssistant
    {
        public String Level { get; set; }

        public TeachingAssistant(string cNIC, string name, string level) : base(cNIC, name)
        {
            Level = level;
        }

        public override void Show()
        {
            Console.WriteLine("Teaching Assistant Data");
            base.Show();
            Console.WriteLine("Level: \t\t" + Level);
        }

        public void Studies()
        {
            throw new NotImplementedException();
        }

        public void Teaches()
        {
            throw new NotImplementedException();
        }

        public void MarksAssignments()
        {
            throw new NotImplementedException();
        }
    }
}
```

- TeachingAssistant object can be saved in its interface `ITeachingAssistant` or `TeachingAssistant`
- TeachingAssistant object can be saved in even generic interface `IPerson` or parent class `Person`

Best practice is save it in interface (`ITeachingAssistant`). It can be generic interface that is top in the hierarchy depending on requirement (`IPerson`)

```
using System;
```

```
namespace _20220613_Interfaces.Interfaces
{
    public class Example
    {
        public void Run()
        {
            Console.WriteLine("========================================");
            Console.WriteLine("============= Interfaces ===============");
            Console.WriteLine("========================================");

            // TeachingAssistant object can be saved in ITeachingAssistant or TeachingAssistant
            ITeachingAssistant person1 = new TeachingAssistant("CNIC-111", "Teaching Assistant 111", "Junior");
            person1.Show();

            TeachingAssistant person2 = new TeachingAssistant("CNIC-222", "Teaching Assistant 222", "Senior");
            person2.Show();

            // OR TeachingAssistant object can be saved in
            // even generic interface IPerson or parent class Person
            IPerson person3 = new TeachingAssistant("CNIC-333", "Teaching Assistant 333", "Junior");
            person3.Show();

            Person person4 = new TeachingAssistant("CNIC-444", "Teaching Assistant 444", "Senior");
            person4.Show();

            // Interface MAGIC is this
            Process(person1);
            Process(person3);
        }

        // Receive in child interface
        void Process(ITeachingAssistant person)
        {
        }

        // Receive in parent interface
        void Process(IPerson person)
        {
        }
    }
}
```

Real magic of Interfaces is that when we pass objects to a function call. Function is expecting an object of a class that is implementing `ITeachingAssistant` or `IPerson` inteface

Built-in Interfaces

Interface	Description	Classes that implement the interface
`IList`	Used by array-indexable collections.	`ArrayList`, `List<T>`, `SortedList`
`IEnumerable`	Enumerates through a collection using a **foreach** statement.	`List<T>` e.g. List<Student>
`IComparer`	Compares two objects held in a collection so that the collection can be sorted.	`SortedList` (CompareTo function)
`IDictionary`	For key/value-based collections such as Hashtable and SortedList.	`Dictionary`

Source Code:

https://github.com/eoccpk/CodingBasics/tree/main/20220613_Interfaces

References:

1. https://www.programiz.com/csharp-programming/interface
2. https://www.geeksforgeeks.org/c-sharp-interface/

Assignments

1. Design class diagram of Banking System **with interfaces**

   There are mainly two types of Accounts. Personal account and Business account. When an Account is opened. It is not active. You can deposit to or withdraw from an account. When you don't do any transaction in a 6 month, account becomes Dormant. You can close your account any time.

   Personal Account can be

   2. Current Account - No Profit No Loss
   3. Salaries Account, A special Current Account for Salaried persons
   4. Saving Account - Profit / Loss account
   5. Fix Deposit Account - Profit / Loss for a specific period of time e.g. 2 years / 5 years

2. Make a class diagram of Hospital System **with interface**

   Doctors as the persons who treat patients. Patient is also a person. Doctor can be Permanent or Visiting doctor. Permanent doctors stays 9-5 on daily basis and Visiting doctors comes for 2-3 hours a day. House Job are trainee doctors who don't have much experience. Specialist doctor have specialization in any field and has high fees. Surgeon are doctor who do patient operation.