set operations

Set Operations

- ✓ Set operations are used for those query operations that produce result set based on presence and absence of elements within same or separate data source.

Set Operations Examples

- ✓ Select distinct records from a data source (No duplicate records).
- ✓ Select all students from a school except class 1 students
- ✓ Select all the toppers from all the classes.
- ✓ Create a single class data source from all sections data sources.

Distinct

Removes all the duplicate datas

Distinct Operator

- ✓ Distinct operator is used to return distinct records from a data source.
- ✓ It has 2 overloaded methods.
- ✓ Distinct can be used with comparer also.

```
static void Main(string[] args)
{
    List<int> numbers = new List<int>() { 1,2,3,1,2,3,4,3,5,5,5};

    var ms = numbers.Distinct().ToList();

    var qs = (from num in numbers
              select num).Distinct().ToList();

    Console.ReadLine();
}
```

in object it will provide 1,2 object

```
static void Main(string[] args)
{

    List<Student> students = new List<Student>()
    {
        new Student(){ Id = 1, Name = "John"},
        new Student(){ Id = 2, Name = "Kim"},
        new Student(){ Id = 3, Name = "John"},
        new Student(){ Id = 4, Name = "Kim"},
    };

    var ms = students.Select(x => x.Name).Distinct().ToList();

    Console.ReadLine();
}
```

It will not work for the complete object so we have to use IEqualityComparer class to override it Distinct call use the reference check in contains method

```
class StudentComparer : IEqualityComparer<Student>
{
    0 references
    public bool Equals(Student x, Student y)
    {
        return x.Id.Equals(y.Id) && x.Name.Equals(y.Name);
    }

    0 references
    public int GetHashCode(Student obj)
    {
        int idHashCode = obj.Id.GetHashCode();
        int nameHashCode = obj.Name.GetHashCode();

        return idHashCode ^ nameHashCode;
    }
}
```

```
        new Student(){ Id = 4, Name = "Mark"},
    };

    var ms = students.Distinct(new StudentComparer()).ToList();

    Console.ReadLine();
}
```

Except

Returns the data source that does not exist int the second data source

Except Operator

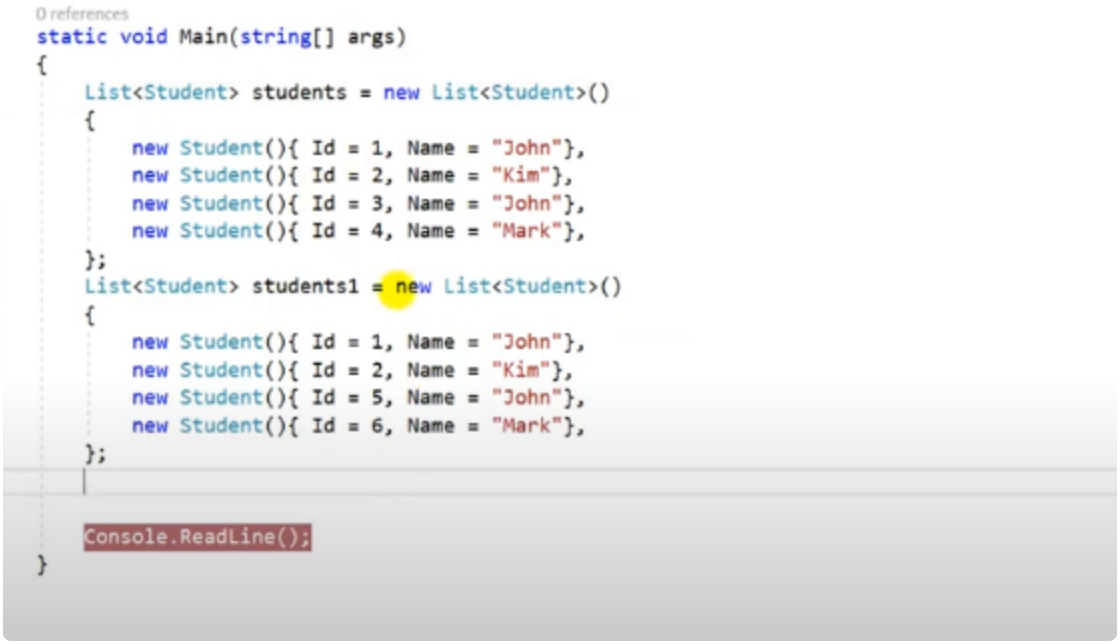
- ✓ Except operator is used to return all the elements from one data source that do not exist in second data source.
- ✓ It has 2 overloaded methods.
- ✓ Except can be used with comparer also.

```
0 references
static void Main(string[] args)
{
    List<string> datasource1 = new List<string>() { "A", "B", "C", "D" };
    List<string> datasource2 = new List<string>() { "C", "D", "E", "F" };

    var ms = datasource1.Except(datasource2).ToList();
    Console.F
}
```

ms Count = 2
[0] "A"
[1] "B"
Raw View

In the objects the output is mark1



```
0 references
static void Main(string[] args)
{
    List<Student> students = new List<Student>()
    {
        new Student(){ Id = 1, Name = "John"},
        new Student(){ Id = 2, Name = "Kim"},
        new Student(){ Id = 3, Name = "John"},
        new Student(){ Id = 4, Name = "Mark"},
    };
    List<Student> students1 = new List<Student>()
    {
        new Student(){ Id = 1, Name = "John"},
        new Student(){ Id = 2, Name = "Kim"},
        new Student(){ Id = 5, Name = "John"},
        new Student(){ Id = 6, Name = "Mark"},
    };

    Console.ReadLine();
}
```

```
0 references
static void Main(string[] args)
{
    List<Student> students = new List<Student>()
    {
        new Student(){ Id = 1, Name = "John"},
        new Student(){ Id = 2, Name = "Kim"},
        new Student(){ Id = 3, Name = "John"},
        new Student(){ Id = 4, Name = "Mark"},
        new Student(){ Id = 4, Name = "Mark1"},
    };
    List<Student> students1 = new List<Student>()
    {
        new Student(){ Id = 1, Name = "John"},
        new Student(){ Id = 2, Name = "Kim"},
        new Student(){ Id = 5, Name = "John"},
        new Student(){ Id = 6, Name = "Mark"},
    };

    var ms = students.Select(x => x.Name).Except(students1.Select(x => x.Name)).ToList();

    Console.ReadLine();
```

Remember there is another operation like separating each object separately and and comparing it

```
0 references
static void Main(string[] args)
{
    List<Student> students = new List<Student>()
    {
        new Student(){ Id = 1, Name = "John"},
        new Student(){ Id = 2, Name = "Kim"},
        new Student(){ Id = 3, Name = "John"},
        new Student(){ Id = 4, Name = "Mark"},
    };
    List<Student> students1 = new List<Student>()
    {
        new Student(){ Id = 1, Name = "John"},
        new Student(){ Id = 2, Name = "Kim"},
        new Student(){ Id = 5, Name = "John"},
        new Student(){ Id = 6, Name = "Mark"},
    };

    var ms = students.Select(x => new { x.Id, x.Name }).Except(students1.Select(x => new { x.Id, x.Name })).ToList();

    Console.ReadLine();
}
```

The second one is using the student comparer by using the class

```
0 references
class StudentComparer : IEqualityComparer<Student>
{
    0 references
    public bool Equals(Student x, Student y)
    {
        return x.Id.Equals(y.Id) && x.Name.Equals(y.Name);
    }

    0 references
    public int GetHashCode(Student obj)
    {
        int idHashCode = obj.Id.GetHashCode();
        int nameHashCode = obj.Name.GetHashCode();

        return idHashCode ^ nameHashCode;
    }
```

```
List<Student> students = new List<Student>()
{
    new Student(){ Id = 1, Name = "John"},
    new Student(){ Id = 2, Name = "Kim"},
    new Student(){ Id = 3, Name = "John"},
    new Student(){ Id = 4, Name = "Mark"},
};
List<Student> students1 = new List<Student>()
{
    new Student(){ Id = 1, Name = "John"},
    new Student(){ Id = 2, Name = "Kim"},
    new Student(){ Id = 5, Name = "John"},
    new Student(){ Id = 6, Name = "Mark"},
};


var ms = students.Except(students1, new StudentComparer()).ToList();

var qs = (from std in students
          select std).Except(students1, new StudentComparer()).ToList();

Console.ReadLine();
```

Intersect

Returns elements present in both the data set

```
0 references
static void Main(string[] args)
{

    List<string> dataSource1 = new List<string>() { "A", "B", "C", "D" };
    List<string> dataSource2 = new List<string>() { "C", "D", "E", "F" };

    var ms = dataSource1.Intersect(dataSource2).ToList();

    var mixs = (from ch in dataSource2
                select ch).Intersect(dataSource1).ToList();

    Console.ReadLine();
}
```

```
static void Main(string[] args)
{

    List<Student> students1 = new List<Student>()
    {
        new Student(){ Id = 1, Name = "John"},
        new Student(){ Id = 2, Name = "Kim"},
        new Student(){ Id = 3, Name = "John"},
        new Student(){ Id = 4, Name = "Mark"},
    };

    List<Student> students2 = new List<Student>()
    {
        new Student(){ Id = 1, Name = "John"},
        new Student(){ Id = 2, Name = "Kim"},
        new Student(){ Id = 5, Name = "John"},
        new Student(){ Id = 6, Name = "Mark"},
    };

    Console.ReadLine();
}
```

```
List<Student> students1 = new List<Student>()
{
    new Student(){ Id = 1, Name = "John"},
    new Student(){ Id = 2, Name = "Kim"},
    new Student(){ Id = 3, Name = "John"},
    new Student(){ Id = 4, Name = "Mark"},
};

List<Student> students2 = new List<Student>()
{
    new Student(){ Id = 1, Name = "John"},
    new Student(){ Id = 2, Name = "Kim"},
    new Student(){ Id = 5, Name = "John"},
    new Student(){ Id = 6, Name = "Mark"},
};

var ms = students1.Select(x => new { x.Id, x.Name }).Intersect(students2.Select(x => new { x.Id, x.Name })).ToList();

Console.ReadLine();
```

```
0 references
class StudentComparer : IEqualityComparer<Student>
{
    0 references
    public bool Equals(Student x, Student y)
    {
        return x.Id.Equals(y.Id) && x.Name.Equals(y.Name);
    }

    0 references
    public int GetHashCode(Student obj)
    {
        int idHashCode = obj.Id.GetHashCode();
        int nameHashCode = obj.Name.GetHashCode();

        return idHashCode ^ nameHashCode;
    }
}
```

```
static void Main(string[] args)
{

    List<Student> students1 = new List<Student>()
    {
        new Student(){ Id = 1, Name = "John"},
        new Student(){ Id = 2, Name = "Kim"},
        new Student(){ Id = 3, Name = "John"},
        new Student(){ Id = 4, Name = "Mark"},
    };
    List<Student> students2 = new List<Student>()
    {
        new Student(){ Id = 1, Name = "John"},
        new Student(){ Id = 2, Name = "Kim"},
        new Student(){ Id = 5, Name = "John"},
        new Student(){ Id = 6, Name = "Mark"},
    };

    var ms = students1.Intersect(students2, new StudentComparer()).ToList();

    Console.ReadLine();
}
```

Union

Return all the elements that appear in either of tow data source

```
{
    0 references
    static void Main(string[] args)
    {

        List<string> dataSource1 = new List<string>() { "A", "B", "C", "D" };
        List<string> dataSource2 = new List<string>() { "C", "D", "E", "F" };

        var ms = dataSource1.Union(dataSource2).ToList();

        Console.ReadLine();
    }
```

Refer the comparer above

```
        new Student(){ Id = 1, Name = "John"},
        new Student(){ Id = 2, Name = "Kim"},
        new Student(){ Id = 5, Name = "John"},
        new Student(){ Id = 6, Name = "Mark"},
    };

    var ms = students1.Union(students2, new StudentComparer()).ToList();
```