# 04 Generics

## Test your knowledge

1. Describe the problem generics address.

Generic allows you to use a class or method that can work with any data type. You need to define the data type while using class or method.

2. How would you create a list of strings, using the generic List class?

List<string> names = new List<string>();

3. How many generic type parameters does the Dictionary class have?

TKey: The type of the keys in the dictionary.

TValue: The type of the values in the dictionary.

4. True/False. When a generic class has multiple type parameters, they must all match.

True

5. What method is used to add items to a List object?

List.Add(): insert at the end of the array.

List.Insert(): add an item with an index.

6. Name two methods that cause items to be removed from a List.

List.Remove(T item)

List.RemoveAll()

List.RemoveAt(int index)

7. How do you indicate that a class has a generic type parameter?

Public class Class<T>

Use <T>

8. True/False. Generic classes can only have one generic type parameter.

False

9. True/False. Generic type constraints limit what can be used for the generic type.

True

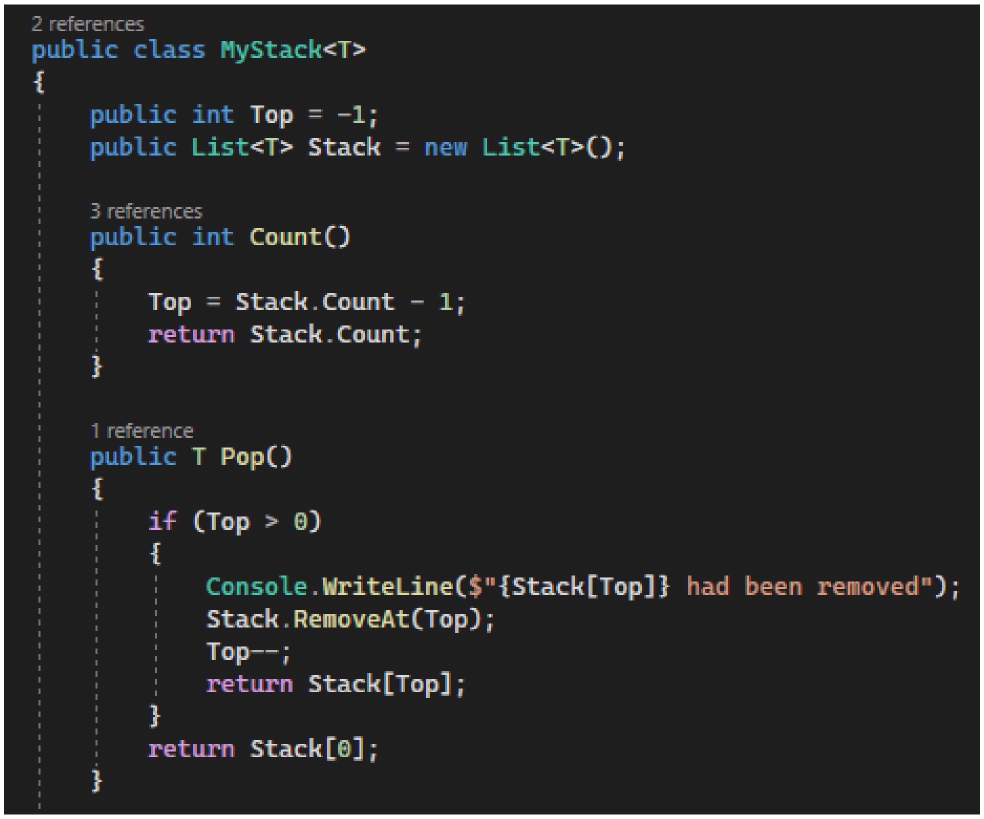
10. True/False. Constraints let you use the methods of the thing you are constraining to.

True

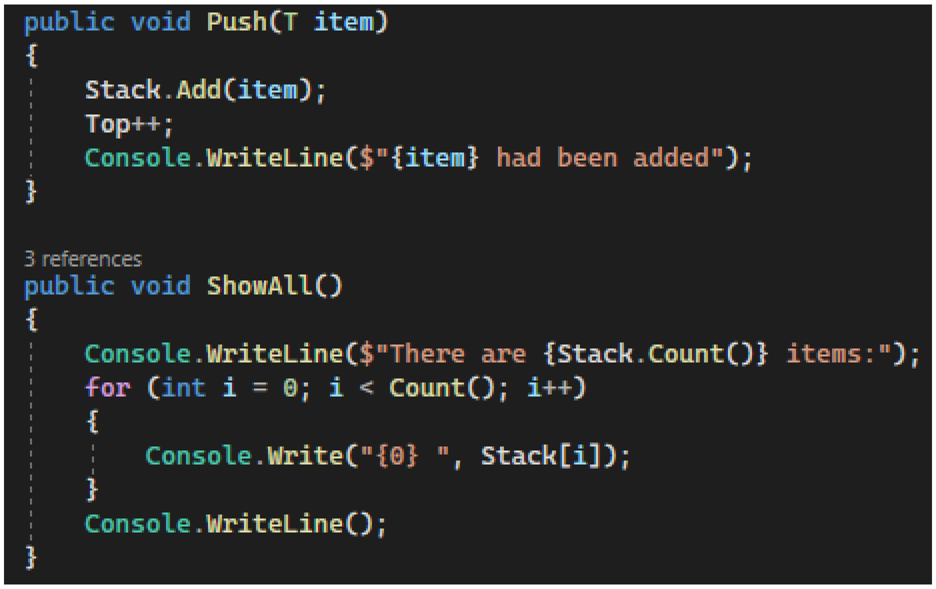
## Practice working with Generics

1. Create a custom Stack class MyStack<T> that can be used with any data type which has following methods

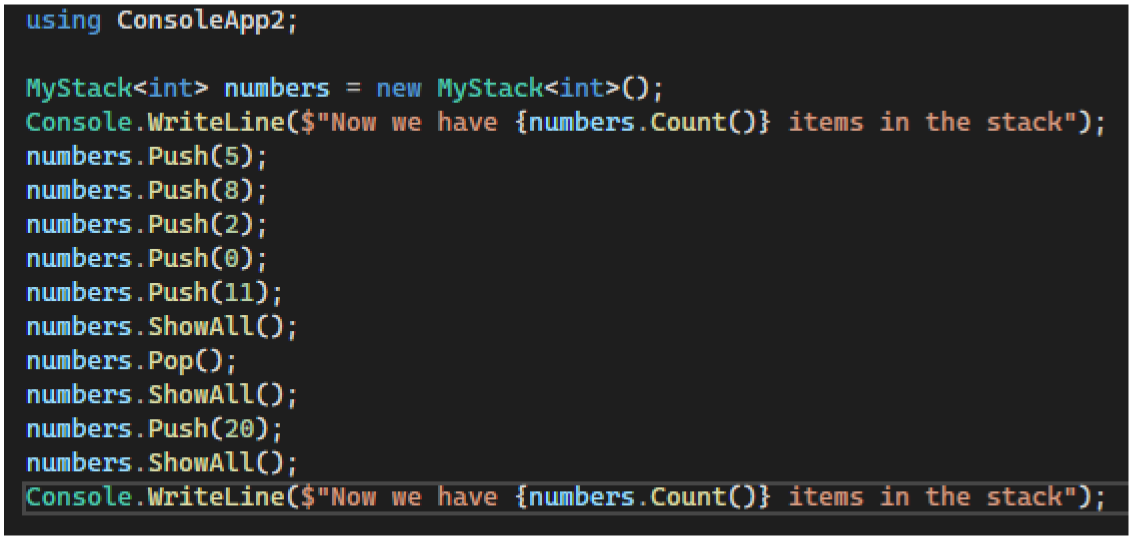
1. int Count()
2. T Pop()
3. Void Push()



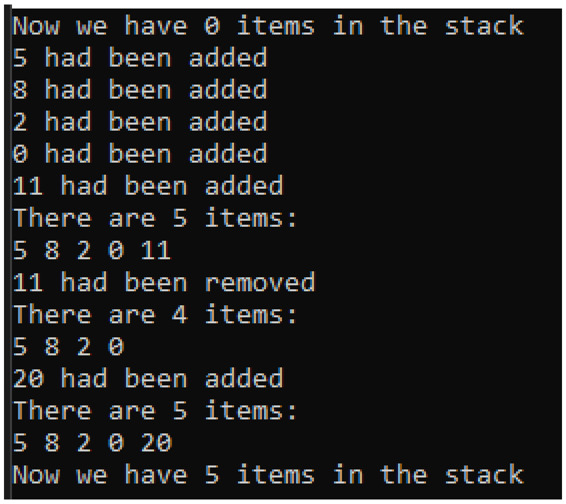
```
2 references
public class MyStack<T>
{
    public int Top = -1;
    public List<T> Stack = new List<T>();

    3 references
    public int Count()
    {
        Top = Stack.Count - 1;
        return Stack.Count;
    }

    1 reference
    public T Pop()
    {
        if (Top > 0)
        {
            Console.WriteLine($"{Stack[Top]} had been removed");
            Stack.RemoveAt(Top);
            Top--;
            return Stack[Top];
        }
        return Stack[0];
    }
```



```
public void Push(T item)
{
    Stack.Add(item);
    Top++;
    Console.WriteLine($"{item} had been added");
}

3 references
public void ShowAll()
{
    Console.WriteLine($"There are {Stack.Count()} items:");
    for (int i = 0; i < Count(); i++)
    {
        Console.Write("{0} ", Stack[i]);
    }
    Console.WriteLine();
}
```



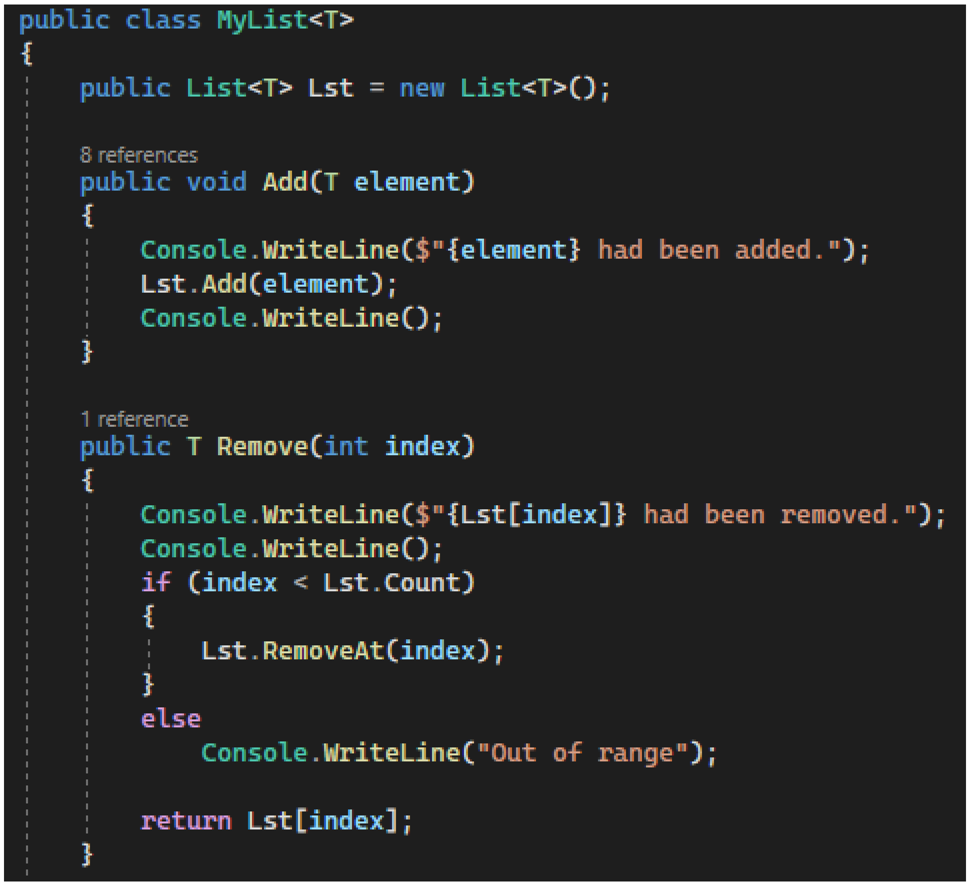
```
using ConsoleApp2;

MyStack<int> numbers = new MyStack<int>();
Console.WriteLine($"Now we have {numbers.Count()} items in the stack");
numbers.Push(5);
numbers.Push(8);
numbers.Push(2);
numbers.Push(0);
numbers.Push(11);
numbers.ShowAll();
numbers.Pop();
numbers.ShowAll();
numbers.Push(20);
numbers.ShowAll();
Console.WriteLine($"Now we have {numbers.Count()} items in the stack");
```



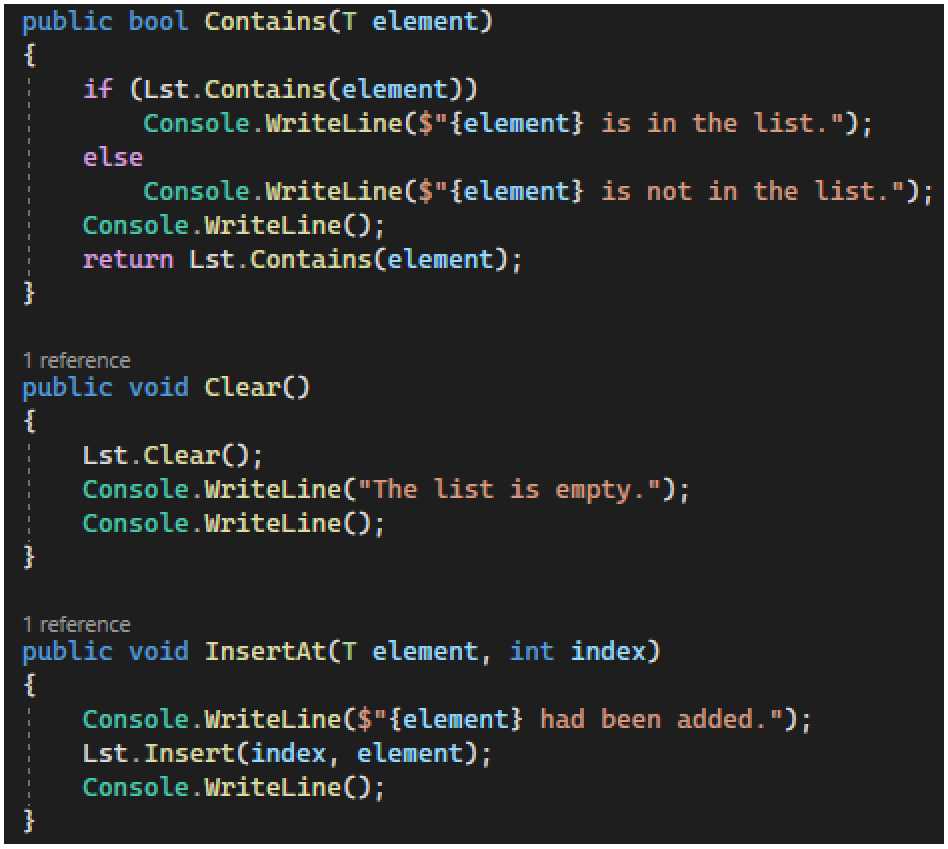
```
Now we have 0 items in the stack
5 had been added
8 had been added
2 had been added
0 had been added
11 had been added
There are 5 items:
5 8 2 0 11
11 had been removed
There are 4 items:
5 8 2 0
20 had been added
There are 5 items:
5 8 2 0 20
Now we have 5 items in the stack
```

2. Create a Generic List data structure MyList<T> that can store any data type. Implement the following methods.

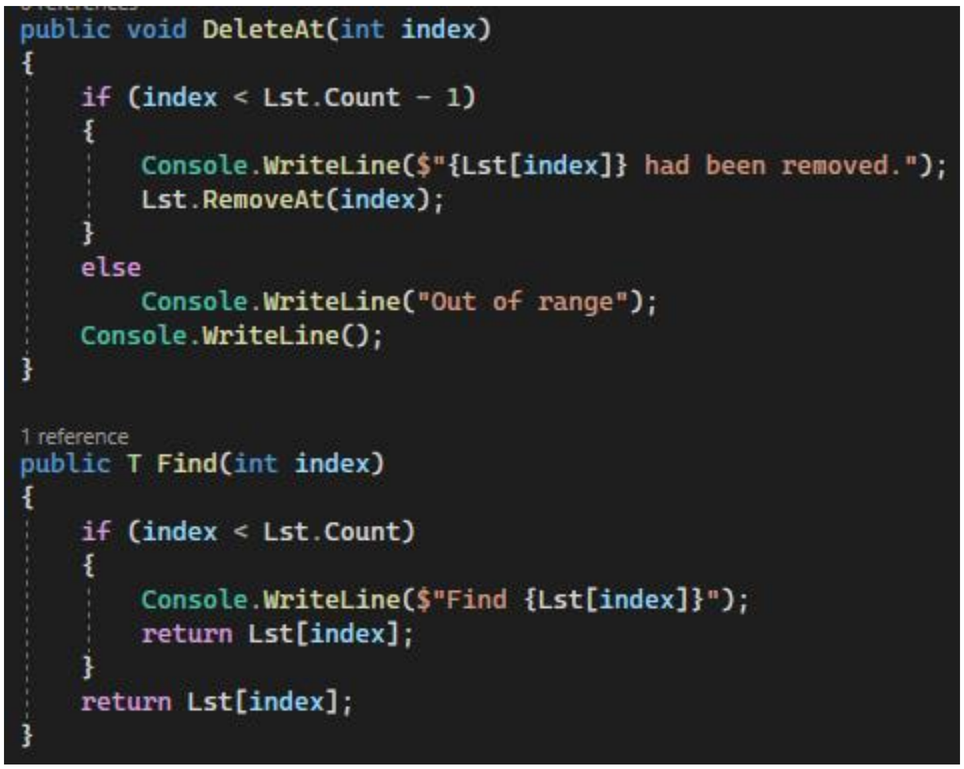
1. void Add (T element)
2. T Remove (int index)
3. bool Contains (T element)
4. void Clear ()
5. void InsertAt (T element, int index)
6. void DeleteAt (int index)
7. T  Find (int index)



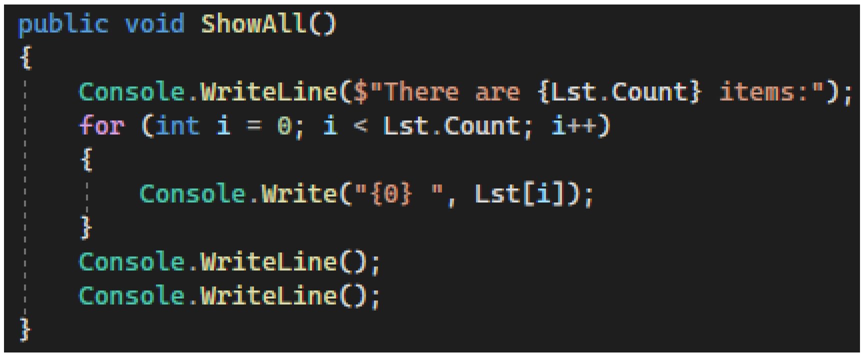
```
public class MyList<T>
{
    public List<T> Lst = new List<T>();

    8 references
    public void Add(T element)
    {
        Console.WriteLine($"{element} had been added.");
        Lst.Add(element);
        Console.WriteLine();
    }

    1 reference
    public T Remove(int index)
    {
        Console.WriteLine($"{Lst[index]} had been removed.");
        Console.WriteLine();
        if (index < Lst.Count)
        {
            Lst.RemoveAt(index);
        }
        else
            Console.WriteLine("Out of range");

        return Lst[index];
    }
```



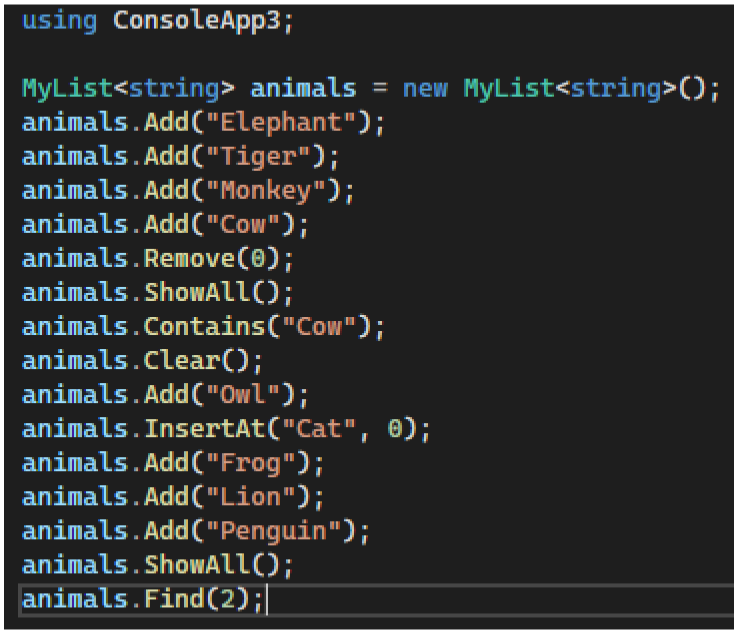
```
public bool Contains(T element)
{
    if (Lst.Contains(element))
        Console.WriteLine($"{element} is in the list.");
    else
        Console.WriteLine($"{element} is not in the list.");
    Console.WriteLine();
    return Lst.Contains(element);
}

1 reference
public void Clear()
{
    Lst.Clear();
    Console.WriteLine("The list is empty.");
    Console.WriteLine();
}

1 reference
public void InsertAt(T element, int index)
{
    Console.WriteLine($"{element} had been added.");
    Lst.Insert(index, element);
    Console.WriteLine();
}
```



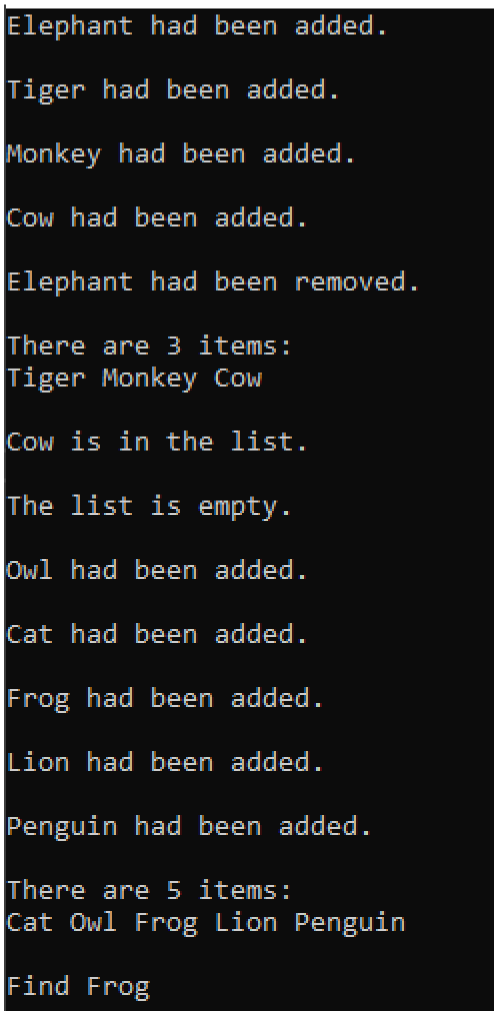
```
public void DeleteAt(int index)
{
    if (index < Lst.Count - 1)
    {
        Console.WriteLine($"{Lst[index]} had been removed.");
        Lst.RemoveAt(index);
    }
    else
        Console.WriteLine("Out of range");
    Console.WriteLine();
}

1 reference
public T Find(int index)
{
    if (index < Lst.Count)
    {
        Console.WriteLine($"Find {Lst[index]}");
        return Lst[index];
    }
    return Lst[index];
}
```



```csharp
public void ShowAll()
{
    Console.WriteLine($"There are {Lst.Count} items:");
    for (int i = 0; i < Lst.Count; i++)
    {
        Console.Write("{0} ", Lst[i]);
    }
    Console.WriteLine();
    Console.WriteLine();
}
```

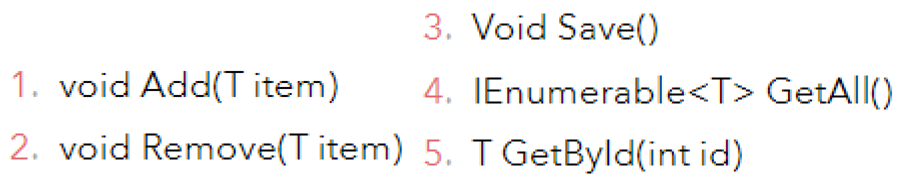


```csharp
using ConsoleApp3;

MyList<string> animals = new MyList<string>();
animals.Add("Elephant");
animals.Add("Tiger");
animals.Add("Monkey");
animals.Add("Cow");
animals.Remove(0);
animals.ShowAll();
animals.Contains("Cow");
animals.Clear();
animals.Add("Owl");
animals.InsertAt("Cat", 0);
animals.Add("Frog");
animals.Add("Lion");
animals.Add("Penguin");
animals.ShowAll();
animals.Find(2);
```



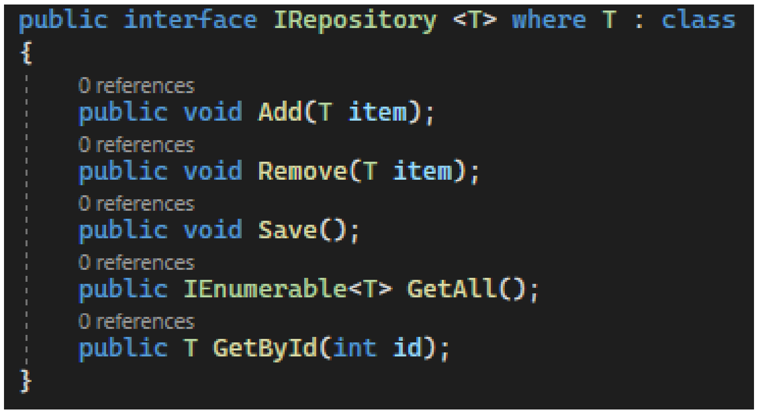
```
Elephant had been added.

Tiger had been added.

Monkey had been added.

Cow had been added.

Elephant had been removed.

There are 3 items:
Tiger Monkey Cow

Cow is in the list.

The list is empty.

Owl had been added.

Cat had been added.

Frog had been added.

Lion had been added.

Penguin had been added.

There are 5 items:
Cat Owl Frog Lion Penguin

Find Frog
```

3. Implement a GenericRepository<T> class that implements IRepository<T> interface that will have common /CRUD/ operations so that it can work with any data source such as SQL Server, Oracle, In-Memory Data etc. Make sure you have a type constraint on T were it should be of reference type and can be of type Entity which has one property called Id. IRepository<T> should have following methods



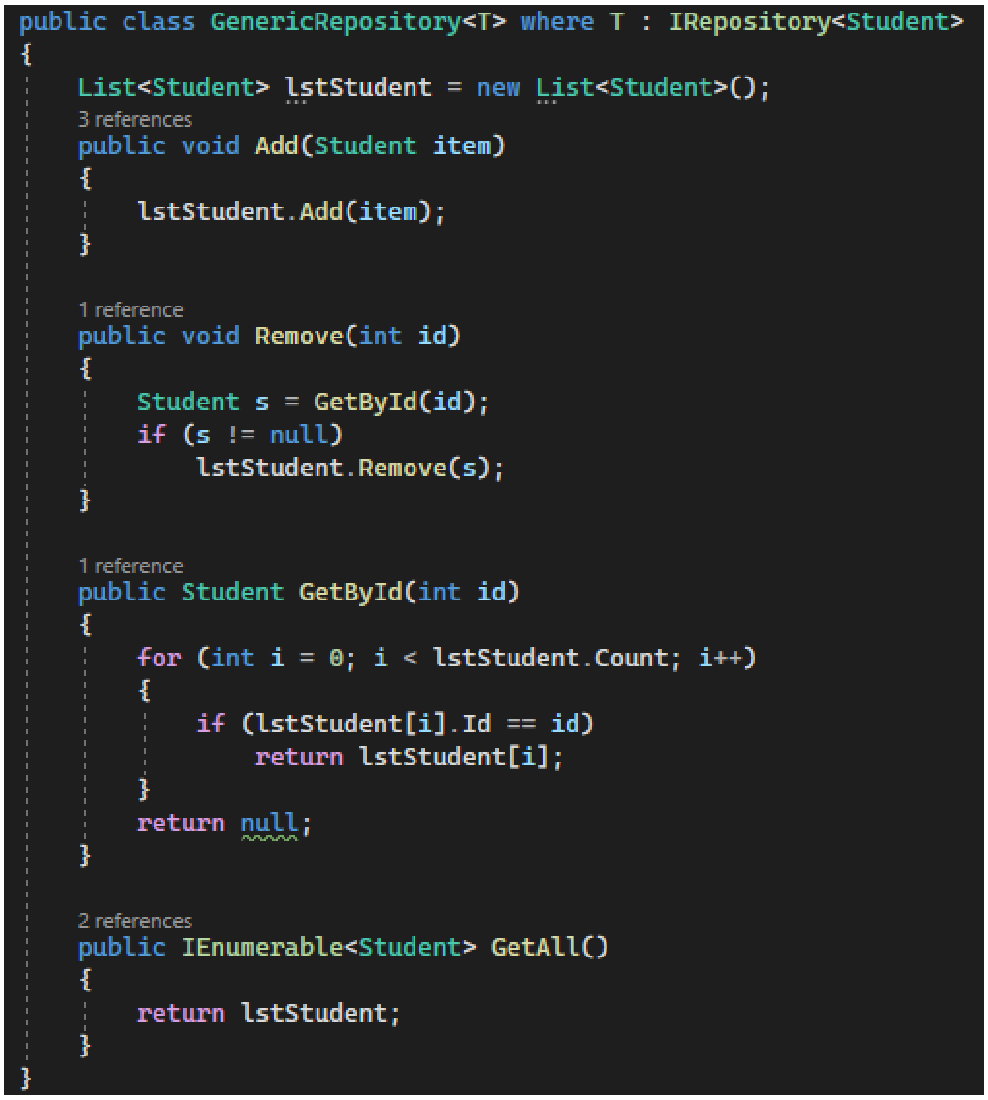
1. void Add(T item)
2. void Remove(T item)
3. Void Save()
4. IEnumerable<T> GetAll()
5. T GetById(int id)

IRepository.cs



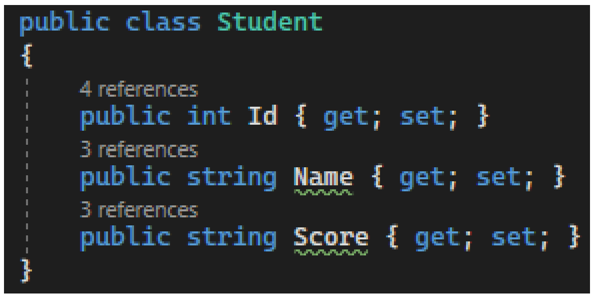
```
public interface IRepository <T> where T : class
{
    0 references
    public void Add(T item);
    0 references
    public void Remove(T item);
    0 references
    public void Save();
    0 references
    public IEnumerable<T> GetAll();
    0 references
    public T GetById(int id);
}
```

GenericRepository.cs



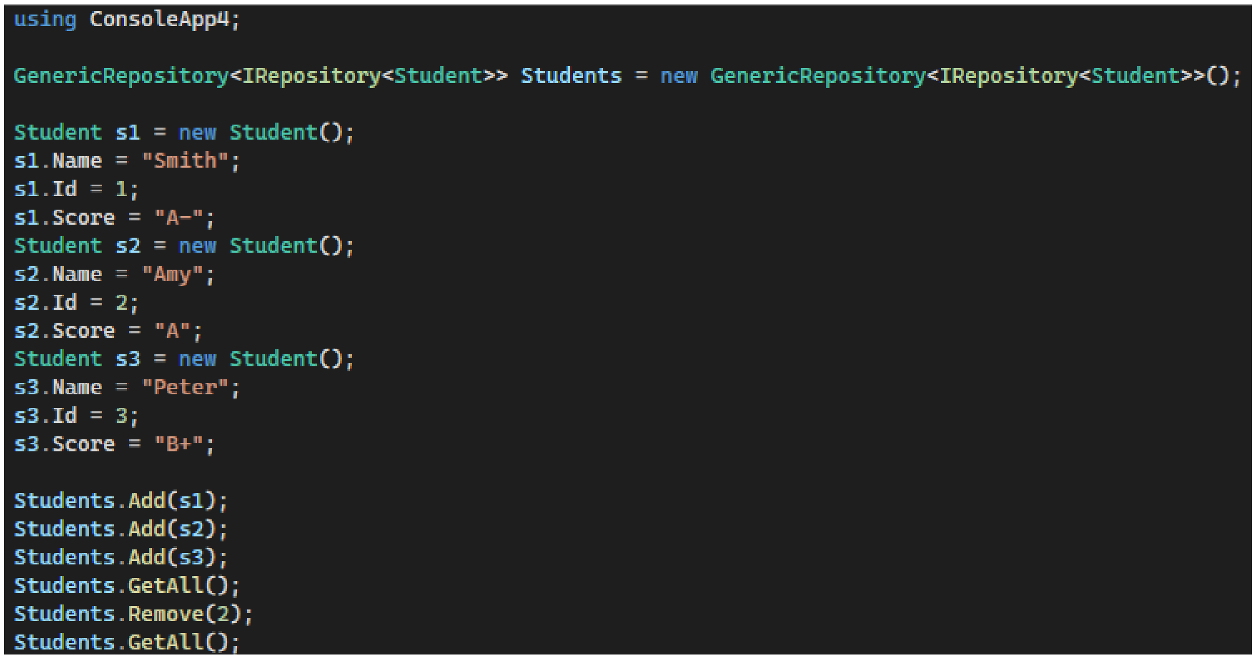
```
public class GenericRepository<T> where T : IRepository<Student>
{
    List<Student> lstStudent = new List<Student>();
    3 references
    public void Add(Student item)
    {
        lstStudent.Add(item);
    }

    1 reference
    public void Remove(int id)
    {
        Student s = GetById(id);
        if (s != null)
            lstStudent.Remove(s);
    }

    1 reference
    public Student GetById(int id)
    {
        for (int i = 0; i < lstStudent.Count; i++)
        {
            if (lstStudent[i].Id == id)
                return lstStudent[i];
        }
        return null;
    }

    2 references
    public IEnumerable<Student> GetAll()
    {
        return lstStudent;
    }
}
```

Student.cs



```
public class Student
{
    4 references
    public int Id { get; set; }
    3 references
    public string Name { get; set; }
    3 references
    public string Score { get; set; }
}
```

Program.cs



```
using ConsoleApp4;

GenericRepository<IRepository<Student>> Students = new GenericRepository<IRepository<Student>>();

Student s1 = new Student();
s1.Name = "Smith";
s1.Id = 1;
s1.Score = "A-";
Student s2 = new Student();
s2.Name = "Amy";
s2.Id = 2;
s2.Score = "A";
Student s3 = new Student();
s3.Name = "Peter";
s3.Id = 3;
s3.Score = "B+";

Students.Add(s1);
Students.Add(s2);
Students.Add(s3);
Students.GetAll();
Students.Remove(2);
Students.GetAll();
```