04 Generics

- Test your Knowledge

1. Describe the problem generics address.

   Generics allow us to design classes and methods but defer the specification of types until the class or method is declared and instantiated.

   Advantages of using generics:

   a. Code Reuse: By using Generics, one needs to write a method/ class/ interface only once and use for any type. Whereas in the non-generics, the code needs to be written again and again whenever needed.

   b. Type Safety: Generics make errors to appear compile time than at run time (It's always better to know problems in your code at compile time rather than making your code fail at run time).

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

```
//Declaration:
List<T> lst = new List<T>();

//Initialization:
List<string> lst = new List<string>();
```

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

   Two, because dictionary are key-value pairs.

```
Dictionary<TKey, TValue> dict = new Dictionary<TKey, TValue>();
```

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?

   Add(): Adds an object to the end of the List<T>.

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

Remove(): Removes the first occurrence of a specific object from the List<T>.

RemoveAt(): Removes the element at the specified index of the List<T>

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

   <T> should be given after class name

8. True/False. Generic classes can only have one generic type parameter — False, we can specify as many as we need. E.g.: Dictionary<TKey, TValue>

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, we can specify the generics type for the certain methods .Declaring those constraints means that we can use the methods calls of the constraining type,

- Practice working with Generics

1. MyStack<T>

```
public class MyStack<T>
{
    Stack<T> stack = new Stack<T> ();
    public int Count()
    {
        return stack.Count();
    }
    public T Pop()
    {
        return stack.Pop();
    }
    public void Push(T obj)
    {
        stack.Push(obj);
    }
}

//create a new instance demo:
MyStack<int> myStack = new MyStack<int>();
myStack.Push(1);
myStack.Push(2);
Console.WriteLine(myStack.Count());
```

2. MyList<T>

```
public class MyList<T>
{
    private List<T> list;
    private int Count;
    private static int MINLENGTH = 100;

    public MyList()
    {
        list = new List<T>();
    }


    public void Add(T element)
    {
        list.Add(element);
    }

    public bool Contains(T element)
    {
        foreach (var e in list)
        {
            if (e.Equals(element)) return true;
        }
        return false;
    }

    public void Clear()
    {
        while (list.Count > 0)
        {
            list.RemoveAt(0);
        }
    }

    public void InsertAt(T element, int index)
    {
        int i = 0;
        while(i <= index) {
            if (i < index)
            {
                i++;
            }
            else
            {
                Add(element);
            }
        }
    }

    public void DeleteAt(int index)
    {
        if (index < 0 || index > list.Count)
```

```
        {
            throw new ArgumentOutOfRangeException("index");
        }
        for (int i = 0; i < list.Count; i++)
        {
            if (i == index)
            {
                list.Remove(list[i]);
                break;
            }
        }
    }

    public T Find(int index)
    {
        for (int i = 0; i < list.Count; i++)
        {
            if (i == index)
            {
                return list[i];
            }
        }
        throw new IndexOutOfRangeException();
    }
}
```

3. Generic Interface

```
public interface IGenericRepository<T> where T : class
    {
        public void Add(T item);
        public void Remove(T item);
        public void Save();
        public IEnumerable<T> GetAll();
        public T GetById(int id);
    }
    public class GenericRepository<T> : IGenericRepository<T> where T: class
    {
        private List<T> list;
        public GenericRepository()
        {
            list = new List<T>();
        }
        public void Add(T item)
        {
            list.Add(item);
        }

        public IEnumerable<T> GetAll()
        {
            return list;
```

```
        }

        public T GetById(int id)
        {
            foreach (var item in list)
            {
                //there is a compiler error because we dont have entity class
                if (item.Id == id)
                {
                    return item;
                }
            }
            return null;
        }

        public void Remove(T item)
        {
            list.Remove(item);
        }

        public void Save()
        {

        }
    }
```