Assignment_04

04 Generics

Test your Knowledge

1. Describe the problem generics address.

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

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

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

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

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

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

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

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

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

1. Generics address the issue of code duplication and type-safety by allowing a class, interface, or method to operate on objects of various types while providing compile-time type-safety.
2. List<string> names = new List<string>();
3. The Dictionary class has two generic type parameters.
4. True. When a generic class has multiple type parameters, they must all match.
5. The `Add` method is used to add items to a List object.
6. Two methods that cause items to be removed from a List are `Remove` and `RemoveAt`.
7. public class MyClass<T>
8. False. A generic class can have multiple generic type parameters.

9. True. Generic type constraints limit what can be used for the generic type, such as class, interface, or specific value types.
10. False. Constraints do not let you use the methods of the thing you are constraining to. Instead, they allow you to use methods of the constraint as if they were part of the generic type.

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()

```
using System;
using System.Collections.Generic;

namespace StackExample
{
    public class MyStack<T>
    {
        private readonly List<T> _stack = new List<T>();

        public int Count()
        {
            return _stack.Count;
        }

        public T Pop()
        {
            if (_stack.Count == 0)
                throw new InvalidOperationException("Stack is empty");
            var item = _stack[_stack.Count - 1];
            _stack.RemoveAt(_stack.Count - 1);
            return item;
        }

        public void Push(T item)
        {
            _stack.Add(item);
        }
```

```
    }
}
```

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)

```
using System;
using System.Collections.Generic;

namespace DataStructures
{
    public class MyList<T>
    {
        private List<T> elements;

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

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

        public T Remove(int index)
        {
            T removedElement = elements[index];
            elements.RemoveAt(index);
            return removedElement;
        }

        public bool Contains(T element)
        {
```

```
            return elements.Contains(element);
        }

        public void Clear()
        {
            elements.Clear();
        }

        public void InsertAt(T element, int index)
        {
            elements.Insert(index, element);
        }

        public void DeleteAt(int index)
        {
            elements.RemoveAt(index);
        }

        public T Find(int index)
        {
            return elements[index];
        }
    }
}
```

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)

```
public interface IRepository<T> where T : class
{
    void Add(T item);
    void Remove(T item);
```

```
    void Save();
    IEnumerable<T> GetAll();
    T GetById(int id);
}

public class GenericRepository<T> : IRepository<T> where T : class, Entity
{
    private readonly List<T> _items = new List<T>();

    public void Add(T item)
    {
        _items.Add(item);
    }

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

    public void Save()
    {
        // Save changes to the data source (e.g. SQL Server, Oracle, In-Memory Data)
    }

    public IEnumerable<T> GetAll()
    {
        return _items;
    }

    public T GetById(int id)
    {
        return _items.FirstOrDefault(item => item.Id == id);
    }
}

public class Entity
{
    public int Id { get; set; }
}
```