Working with Collections and Generics in C#

Collections in C#

Collections are used to store, manage, and manipulate groups of objects. C# provides two main types of collections:

1. **Non-Generic Collections**: Found in the `System.Collections` namespace, these include classes like `ArrayList`, `Hashtable`, `SortedList`, `Stack`, and `Queue`. They can store any type of data but lack type safety and can lead to runtime errors due to type mismatches.
2. **Generic Collections**: Found in the `System.Collections.Generic` namespace, these include classes like `List<T>`, `Dictionary<TKey, TValue>`, `Queue<T>`, `Stack<T>`, and `HashSet<T>`. They provide type safety, better performance, and eliminate the need for boxing and unboxing[1][2].

Generics in C#

Generics allow you to define classes, methods, and interfaces with a placeholder for the type of data they store or use. This provides several benefits:

- **Type Safety**: Ensures that the data types are consistent.
- **Performance**: Eliminates the overhead of boxing and unboxing.
- **Code Reusability**: Allows the same code to be used with different data types[2].

Case Studies on `List<T>` Class

Case Study 1: Basic Usage of `List<T>`

The `List<T>` class represents a strongly typed list of objects that can be accessed by index. It provides methods to search, sort, and manipulate lists.

Example:

```
List<int> numbers = new List<int>();
numbers.Add(1);
numbers.Add(2);
numbers.Add(3);

foreach (int number in numbers)
{
    Console.WriteLine(number);
}
```

In this example, a list of integers is created, and elements are added to it. The list is then iterated over to print each element[3].

Case Study 2: Using `List<T>` with Custom Objects

You can also use `List<T>` with custom objects.

Example:

```
public class Student
{
    public string Name { get; set; }
    public int Age { get; set; }
}

List<Student> students = new List<Student>
{
    new Student { Name = "Alice", Age = 20 },
    new Student { Name = "Bob", Age = 22 }
};

foreach (Student student in students)
{
    Console.WriteLine($"Name: {student.Name}, Age: {student.Age}");
}
```

Here, a list of `Student` objects is created and populated. Each student's details are then printed3.

Case Study 3: Advanced Usage with LINQ

You can use LINQ (Language Integrated Query) to perform complex queries on `List<T>`.

Example:

```
List<int> numbers = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
var evenNumbers = numbers.Where(n => n % 2 == 0).ToList();

foreach (int number in evenNumbers)
{
    Console.WriteLine(number);
}
```

Example on List of Employee Objects

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

public class Employee
{
    public int Id { get; set; }
    public string Name { get; set; }
```

```
    public string Department { get; set; }
    public double Salary { get; set; }
}

public class Program
{
    public static void Main()
    {
        List<Employee> employees = new List<Employee>
        {
            new Employee { Id = 1, Name = "Alice", Department = "HR", Salary =
50000 },
            new Employee { Id = 2, Name = "Bob", Department = "IT", Salary = 60000
},
            new Employee { Id = 3, Name = "Charlie", Department = "Finance",
Salary = 55000 }
        };


    }
}
```

Example on Dictionary

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

public class Employee
{
    public int Id { get; set; }
    public string Name { get; set; }
    public double Salary { get; set; }
}

public class Program
{
    public static void Main()
    {
        Dictionary<int, Employee> employees = new Dictionary<int, Employee>
        {
            { 1, new Employee { Id = 1, Name = "Alice", Salary = 50000 } },
            { 2, new Employee { Id = 2, Name = "Bob", Salary = 60000 } },
            { 3, new Employee { Id = 3, Name = "Charlie", Salary = 55000 } }
        };

        // Access an element by key
        Employee employee = employees[2];
        Console.WriteLine($"Name: {employee.Name}, Salary: {employee.Salary}");

        // Iterate over the Dictionary
```

```
        foreach (var kvp in employees)
        {
            Console.WriteLine($"Key: {kvp.Key}, Name: {kvp.Value.Name}, Salary:
{kvp.Value.Salary}");
        }

        // Add a new employee
        employees.Add(4, new Employee { Id = 4, Name = "David", Salary = 70000 });

        // Update an existing employee's salary
        if (employees.ContainsKey(2))
        {
            employees[2].Salary = 65000;
        }

        // Remove an employee
        employees.Remove(3);

        // Display the updated Dictionary
        Console.WriteLine("Updated Dictionary:");
        foreach (var kvp in employees)
        {
            Console.WriteLine($"Key: {kvp.Key}, Name: {kvp.Value.Name}, Salary:
{kvp.Value.Salary}");
        }
    }
}
```

Example on SortedList

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

public class Employee
{
    public int Id { get; set; }
    public string Name { get; set; }
    public double Salary { get; set; }
}

public class Program
{
    public static void Main()
    {
        SortedList<int, Employee> employees = new SortedList<int, Employee>
        {
            { 3, new Employee { Id = 3, Name = "Charlie", Salary = 55000 } },
            { 1, new Employee { Id = 1, Name = "Alice", Salary = 50000 } },
            { 2, new Employee { Id = 2, Name = "Bob", Salary = 60000 } }
        };
```

```
        // Access by key
        Employee employeeByKey = employees[2];
        Console.WriteLine($"Access by key: Name: {employeeByKey.Name}, Salary:
{employeeByKey.Salary}");

        // Access by index
        Employee employeeByIndex = employees.Values[0];
        Console.WriteLine($"Access by index: Name: {employeeByIndex.Name}, Salary:
{employeeByIndex.Salary}");

        // Iterate over the SortedList
        foreach (var kvp in employees)
        {
            Console.WriteLine($"Key: {kvp.Key}, Name: {kvp.Value.Name}, Salary:
{kvp.Value.Salary}");
        }

        // Remove an element
        employees.Remove(1);
        Console.WriteLine("After removing key 1:");
        foreach (var kvp in employees)
        {
            Console.WriteLine($"Key: {kvp.Key}, Name: {kvp.Value.Name}, Salary:
{kvp.Value.Salary}");
        }
    }
}
```