C# Collections - Generic and Non-Generic

Introduction

Collections in C# are used to store, manage, and manipulate groups of related objects.
They provide a flexible way to work with groups of data. C# supports both generic and non-generic collections, each with its own use case.

Understanding collections helps in writing efficient, clean, and type-safe code.

Non-Generic Collections

Non-generic collections are part of the System.Collections namespace. These collections can store elements of any data type,
which offers flexibility but sacrifices type safety and performance. Non-generic collections store items as objects, leading to boxing/unboxing overhead when using value types.

```
using System.Collections;
ArrayList list = new ArrayList();
list.Add(1);
list.Add("Hello");
list.Add(3.14);

foreach (var item in list)
{
    Console.WriteLine(item);
}
```

Note: Because ArrayList can store any type of data, it is prone to runtime errors and performance issues due to boxing/unboxing of value types.

Generic Collections

Generic collections are part of the System.Collections.Generic namespace. They allow you to define the type of elements the collection will hold, which ensures type safety and improves performance.

Generics reduce runtime errors by catching type mismatches at compile time.

C# Collections - Generic and Non-Generic

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

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

Other common generic collections include:

- Dictionary<TKey, TValue>: Stores key-value pairs.
- Queue<T>: FIFO (First-In-First-Out) structure.
- Stack<T>: LIFO (Last-In-First-Out) structure.
- HashSet<T>: Stores unique elements.

Comparison

Comparison:

Non-Generic Collections:

- Can store any type of data (object-based).
- Not type-safe.
- Prone to boxing/unboxing (performance hit).
- Useful when dealing with mixed data types or legacy code.

Generic Collections:

- Type-safe and compile-time checking.
- Better performance (no boxing/unboxing).
- Easier to work with using IntelliSense and LINQ.
- Recommended for modern C# programming.

Summary

C# Collections - Generic and Non-Generic

Summary:

Use generic collections whenever possible for better type safety and performance. Non-generic collections should be used only when necessary, such as working with legacy systems or when you need to store mixed data types.