**Collections in C#**

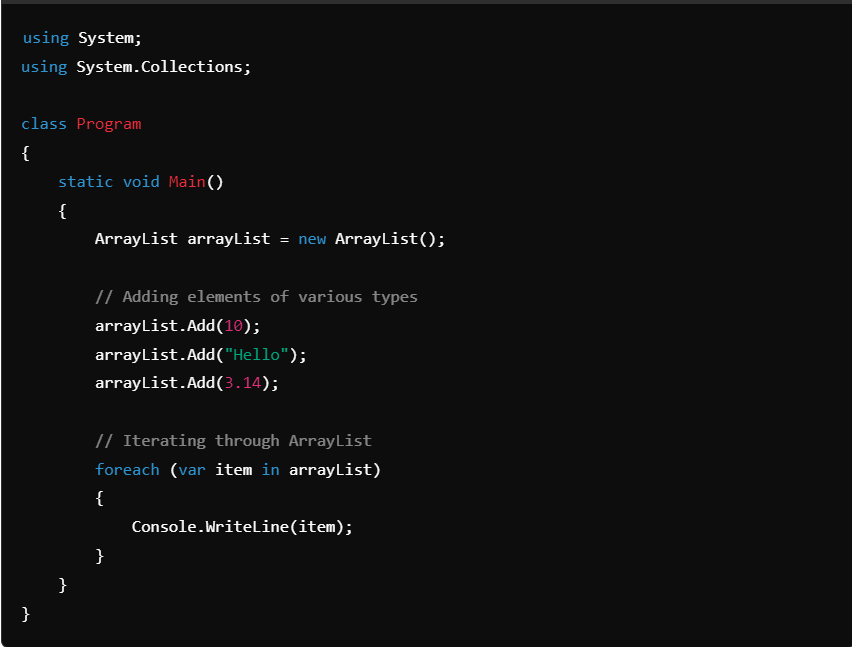
**Collections** in C# are used to store, manage, and manipulate groups of objects. They provide more flexible ways to work with groups of objects compared to arrays, as collections can dynamically grow and shrink in size and offer various utility functions for adding, removing, and searching for elements.

C# provides a variety of collection types under the System.Collections and System.Collections.Generic namespaces. Let's explore the most common ones with examples.

**1. ArrayList**

**ArrayList** is a non-generic collection that can store objects of any data type. It is dynamic in size and allows adding or removing elements at runtime. However, it's slower and less type-safe compared to generic collections like List<T>.

* **Namespace**: System.Collections
* **Key Features**:
  + Stores objects as Object.
  + Not type-safe.
  + Resizable.



**Output**:

10

Hello

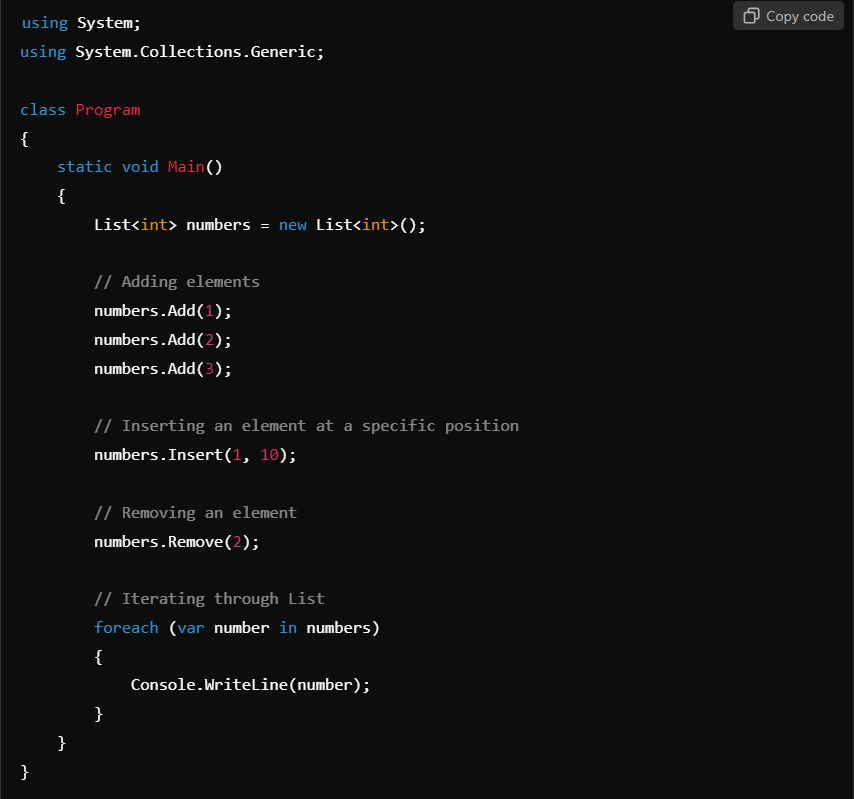
3.14

* **Limitations**:
  + Type casting is required when accessing elements.
    - In C#, type casting is the process of assigning a value of one data type to another. There are two types of type casting in C#:
    - Implicit casting - Automatically converts a smaller type to a larger type. For example, char -> int -> long -> float -> double.
    - Explicit casting-Manually converts a larger type to a smaller type. For example, double -> float -> long -> int -> char.
    - Here are some things to keep in mind about type casting in C#:
    - You can use the is operator to test for compatibility before performing a cast.
    - Type conversion is only possible if the data types are compatible. Otherwise, an InvalidCastException will occur.
    - Casting is required when information might be lost in the conversion, or if the conversion might not succeed for other reasons.
    - You can perform a cast by specifying the type you're casting to in parentheses in front of the value or variable to be converted.
  + Potential for runtime errors due to type mismatches.

**2. List<T> (Generic)**

**List<T>** is a generic collection that provides type safety and performance advantages over ArrayList. It allows you to store elements of a specific type, ensuring that type checking happens at compile-time.

* **Namespace**: System.Collections.Generic
* **Key Features**:
  + Type-safe.
  + Resizable.
  + Provides various utility methods like Add(), Remove(), Find(), etc.



**Output**:

1

10

3

* **Advantages**:
  + Type-safe: Elements must be of type T.
  + Performance: No boxing/unboxing (in case of value types).

**3. Dictionary<TKey, TValue>**

A **Dictionary** is a collection that stores key-value pairs. It is used when you want to associate a value with a unique key and retrieve values efficiently using the key.

* **Namespace**: System.Collections.Generic
* **Key Features**:
  + Key-based access.
  + Fast lookups.
  + Keys must be unique.

A computer screen with text and images

Description automatically generated

**Output**:

Student 102: Jane

ID: 101, Name: John

ID: 102, Name: Jane

ID: 103, Name: Alice

* **Advantages**:
  + Fast lookups based on keys.
  + Keys must be unique.
* **Use Cases**:
  + Associating student names with IDs.
  + Storing configuration settings.

**4. Queue<T>**

A **Queue** is a first-in, first-out (FIFO) collection where elements are added to the back and removed from the front. It's useful in scenarios like task scheduling or when you need to process elements in the order they arrive.

* **Namespace**: System.Collections.Generic
* **Key Features**:
  + FIFO (First-In-First-Out) behavior.
  + Enqueue() to add, Dequeue() to remove.

A computer screen shot of a program

Description automatically generated

**Output**:

Task 1

Task 2

* **Advantages**:
  + Processes elements in the order they were added.
* **Use Cases**:
  + Task scheduling.
  + Job processing.

**5. Stack<T>**

A **Stack** is a last-in, first-out (LIFO) collection where elements are added to and removed from the top. It is used in scenarios where you need to reverse items or undo operations.

* **Namespace**: System.Collections.Generic
* **Key Features**:
  + LIFO (Last-In-First-Out) behavior.
  + Push() to add, Pop() to remove.

A screen shot of a computer

Description automatically generated

**Output**:

Item 3

Item 2

* **Advantages**:
  + Reverses the order of elements.
* **Use Cases**:
  + Undo operations.
  + Reversing data.

**6. HashSet<T>**

A **HashSet** is a collection that contains unique elements, meaning duplicates are not allowed. It's often used when you need to maintain a collection of unique values.

* **Namespace**: System.Collections.Generic
* **Key Features**:
  + No duplicate elements.
  + Fast lookups for membership tests.

A screenshot of a computer program

Description automatically generated

**Output**:

Set contains 2

1

2

3

* **Advantages**:
  + Ensures uniqueness.
  + Fast lookups.
* **Use Cases**:
  + Storing unique IDs.
  + Checking for duplicates.

**7. LinkedList<T>**

A **LinkedList** is a doubly linked list, where each node contains a reference to the next and previous node. It allows for efficient insertions and deletions at any point in the list.

* **Namespace**: System.Collections.Generic
* **Key Features**:
  + Fast insertions and deletions.
  + Supports traversal in both directions (forward and backward).

A screen shot of a computer program

Description automatically generated

**Output**:

Node 0

Node 1

Node 2

* **Advantages**:
  + Efficient insertions and deletions at any position.
* **Use Cases**:
  + Implementing stacks, queues, and deques.
  + Manipulating elements in the middle of a list.

**Conclusion:**

C# provides a rich set of collection types, each suited for different scenarios:

* **ArrayList** for non-generic collections.
* **List<T>** for type-safe, resizable collections.
* **Dictionary<TKey, TValue>** for key-value mappings.
* **Queue<T>** and **Stack<T>** for FIFO and LIFO processing, respectively.
* **HashSet<T>** for maintaining unique elements.
* **LinkedList<T>** for efficient insertions and deletions.

Choosing the right collection depends on your specific use case, such as whether you need fast lookups, ordered elements, or unique values.