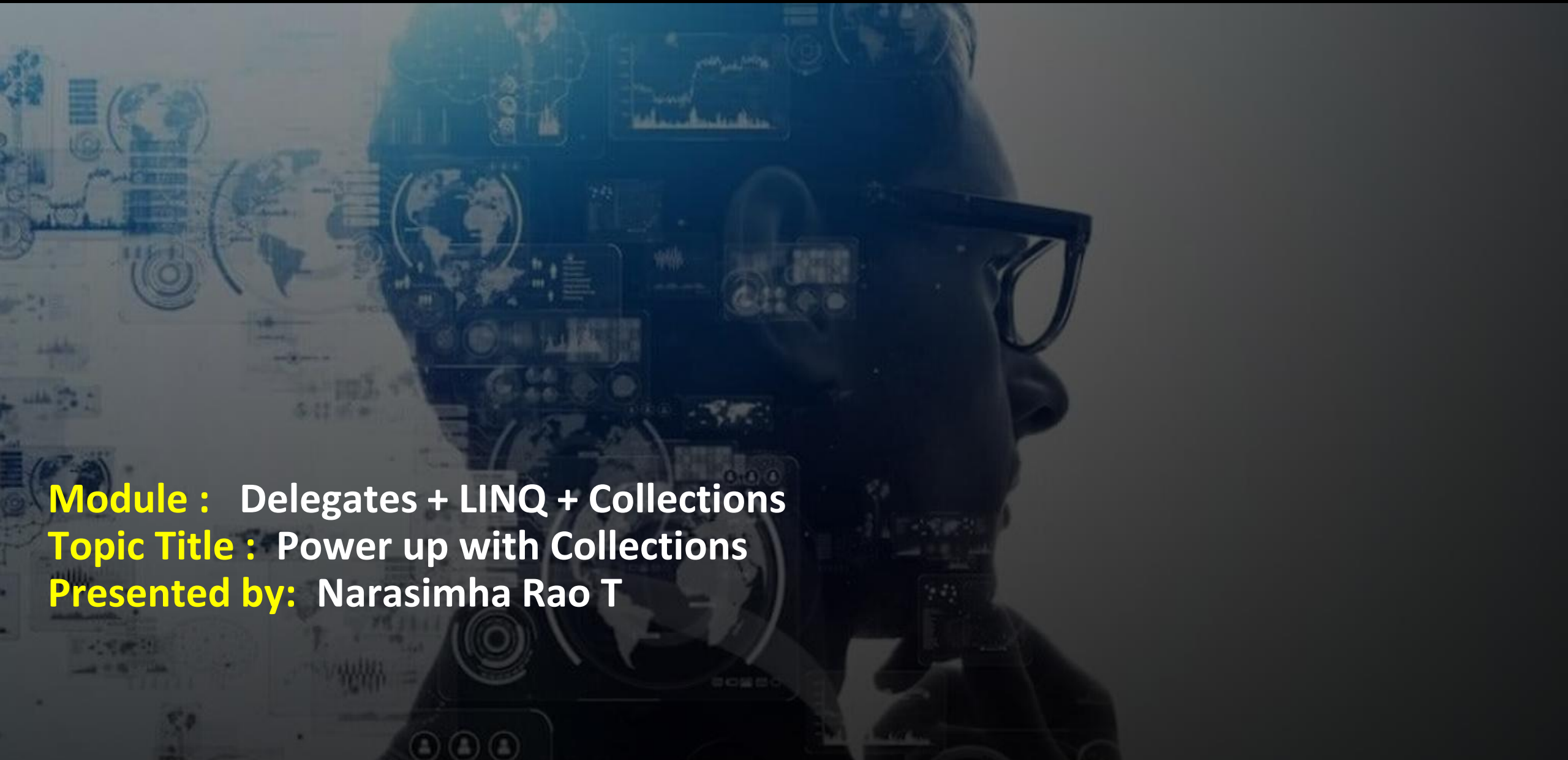

upGrad ENTERPRISE
.NET FSD
Bootcamp Training

upGrad ENTERPRISE



Module : Delegates + LINQ + Collections
Topic Title : Power up with Collections
Presented by: Narasimha Rao T

upGrad ENTERPRISE

Collections in C#

By
Narasimha Rao T
Microsoft.Net FSD Trainer
Professional Development Trainer
tnrao.trainer@gmail.com

upGrad ENTERPRISE

Index

1. Why use collections over arrays
2. Introduction to generic collections:
3. List<T>, Dictionary<TKey, TValue>
4. Queue<T>, Stack<T>
5. Common operations: Add, Remove, Contains, iteration
6. Hands-Ons
7. Q & A

Why Use Collections Over Arrays?

Arrays

- **Fixed size** once declared.
- Best when you know the exact number of elements.
- Offers fast access via index.

```
int[] numbers = new int[5];
numbers[0] = 10;
```

Limitations of Arrays:

- Size can't be changed at runtime.
- No built-in methods for dynamic operations like adding or removing.
- Not ideal for real-time or large-scale applications where size varies.

Collections Advantages:

- Dynamic sizing.
- Rich set of methods (Add, Remove, Contains, etc.).
- Easier data manipulation.
- More suitable for complex data structures.

Array vs ArrayList vs List<T>

Feature	Array	ArrayList	List<T>
Type Safety	(strongly typed)	(object type)	(generic, strongly typed)
Performance	Fast	Slower (boxing/unboxing)	Fast (type-safe)
Flexibility	Fixed size	Dynamic	Dynamic
Usage Example	`int[] arr = new int[3];`	`ArrayList list = new ArrayList();`	`List<int> list = new List<int>();`

Introduction to Generic Collections

What Are Generics?

- Allow classes and methods to operate on **any data type** while maintaining **type safety**.
- Avoid boxing/unboxing.
- Enable code reusability.

```
List<string> names = new List<string>();
```

Benefits:

- Compile-time type checking.
- Better performance.
- Code readability.

List<T>

Description:

- Represents a strongly typed list of objects.
- Resizable array.

```
List<int> numbers = new List<int>();
numbers.Add(10);
numbers.Add(20);
```

Dictionary<TKey, TValue>

Description:

- Stores key-value pairs.
- Fast lookups by key.

```
Dictionary<string, int> ages = new Dictionary<string, int>();
ages["Alice"] = 30;
ages["Bob"] = 25;
```

Queue<T>

Description:

- First-In-First-Out (FIFO) collection.

```
Queue<string> queue = new Queue<string>();
queue.Enqueue("Task1");
string next = queue.Dequeue();
```

Stack<T>

Description:

- Last-In-First-Out (LIFO) collection.

```
Stack<int> stack = new Stack<int>();
stack.Push(10);
int top = stack.Pop();
```

Common Operations on Collections

Add

```
list.Add("New Item");
```

Remove

```
list.Remove("Item");
```

Contains

```
if (list.Contains("Item")) {
    // Do something
}
```

Iteration

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

Applications of Generics in Real-Time

1. **Data Caching**
   Use `Dictionary<TKey, TValue>` for caching key-based data (e.g., user sessions).
2. **Queues in Job Scheduling**
   `Queue<T>` used in printer queues, background jobs, task pipelines.
3. **Undo Functionality**
   `Stack<T>` used in text editors for undo operations.
4. **Database Record Management**
   `List<T>` used to store and manipulate query results.
5. **API Response Handling**
   `List<T>` helps deserialize and manipulate JSON responses from APIs.

Summary

- Arrays are limited by fixed size.
- Collections offer flexibility, dynamic sizing, and richer features.
- Generics improve performance and safety.
- Choose the right collection (List, Dictionary, Queue, Stack) based on your use case.

upGrad ENTERPRISE

Q & A

Narasimha Rao T
tnrao.trainer@gmail.com