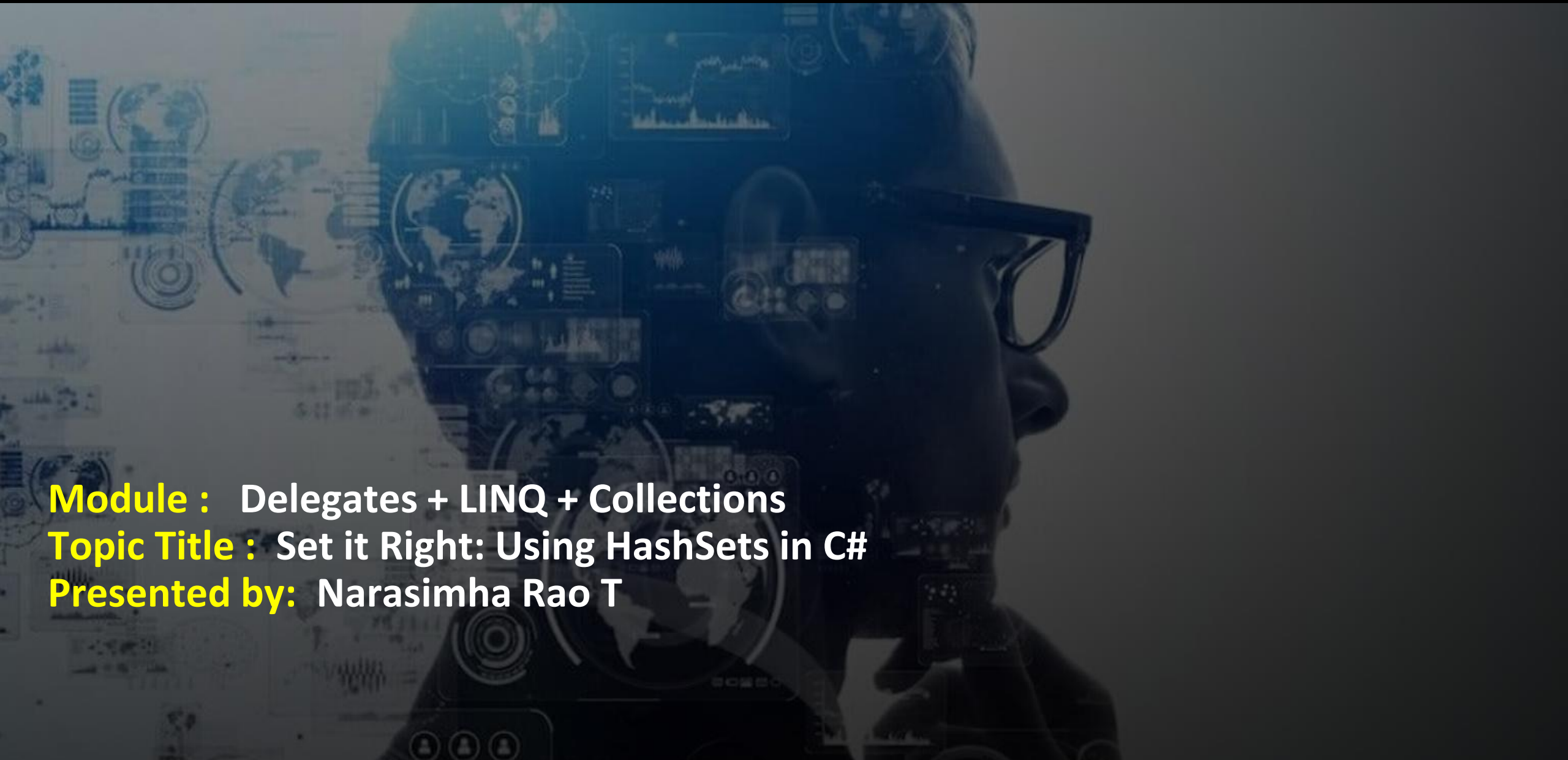

upGrad ENTERPRISE
.NET FSD
Bootcamp Training

upGrad ENTERPRISE



Module : Delegates + LINQ + Collections
Topic Title : Set it Right: Using HashSets in C#
Presented by: Narasimha Rao T

upGrad ENTERPRISE

Working with HashSets in C#

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

Day10

Index

1. What is `HashSet<T>`?

- A **collection of unique elements**.
- Found in `System.Collections.Generic`.
- Uses **hashing** for fast lookups.
- Unordered: does **not preserve insertion order**.
- Ideal when **duplicate values are not allowed**.

Example

```
HashSet<int> numbers = new HashSet<int>();
numbers.Add(1);
numbers.Add(2);
numbers.Add(2); // Ignored, already exists
```

2. HashSet and Uniqueness Enforcement

- `HashSet<T>` automatically **enforces uniqueness**.
- When you use `.Add()`, it returns:
  - `true` if the item was added.
  - `false` if it already existed.

```
HashSet<string> fruits = new HashSet<string>();
Console.WriteLine(fruits.Add("Apple")); // true
Console.WriteLine(fruits.Add("Apple")); // false
```

3. Set Operations

`HashSet<T>` provides mathematical set operations:

`UnionWith()`: Combines elements from another set.

```
var set1 = new HashSet<int> { 1, 2 };
var set2 = new HashSet<int> { 2, 3 };
set1.UnionWith(set2); // set1 = {1, 2, 3}
```

`IntersectWith()`: Keeps only common elements.

```
set1 = new HashSet<int> { 1, 2 };
set2 = new HashSet<int> { 2, 3 };
set1.IntersectWith(set2); // set1 = {2}
```

`ExceptWith()`: Removes elements that exist in another set.

```
set1 = new HashSet<int> { 1, 2, 3 };
set2 = new HashSet<int> { 2 };
set1.ExceptWith(set2); // set1 = {1, 3}
```

Practical Examples using HashSet

Practical Examples

Example 1: Removing Duplicates from List

```
List<string> names = new List<string> { "Tom", "Tom", "Jerry", "Anna" };

HashSet<string> uniqueNames = new HashSet<string>(names);
foreach (var name in uniqueNames)
    Console.WriteLine(name);
```

Example 2: Common Students in Two Courses

```
var courseA = new HashSet<string> { "Alice", "Bob", "Charlie" };
var courseB = new HashSet<string> { "Bob", "Daniel", "Charlie" };
courseA.IntersectWith(courseB); // Bob and Charlie
```

Example 3: Validate Unique Usernames

```
HashSet<string> usernames = new HashSet<string>();

bool AddUsername(string name)
{
    if (usernames.Add(name))
    {
        Console.WriteLine("Username added.");
        return true;
    }
    Console.WriteLine("Duplicate username.");
    return false;
}
```

4. Comparison with Other Collections

Feature	`HashSet<T>`	`List<T>`	`Dictionary<TKey,TValue>`
Allows Duplicates	❌ No	✅ Yes	❌ (keys)
Lookup Performance	⚡ Fast (O(1))	🚶 Slower (O(n))	⚡ Fast (O(1))
Maintains Order	❌ No	✅ Yes	✅ (by key)
Key-Value Pair Support	❌ No	❌ No	✅ Yes

5. Legacy Collections (Overview)

`ArrayList`

- Non-generic.
- Stores elements as `object`.
- Boxing/unboxing for value types.
- Slower and type-unsafe.

```
ArrayList list = new ArrayList();
list.Add(1);
list.Add("hello"); // Mixed types allowed
```

`Hashtable`

- Non-generic key-value pair collection.
- Keys and values stored as `object`.
- Superseded by `Dictionary<TKey,TValue>`.

```
Hashtable table = new Hashtable();
table["id"] = 101;
table[123] = "value";
```

Note : Prefer `List<T>`, `Dictionary<TKey, TValue>`, `HashSet<T>` for type safety and performance.

Quiz Time

Multiple Choice

1. What is the time complexity of `HashSet<T>.Contains()`?
    - A) O(n)
    - B) O(log n)
    - C) O(1)
    - D) O(n log n)

2. Which of these collections allows duplicates?

- A) HashSet<T>
- B) List<T>
- C) Dictionary<TKey, TValue> (keys)
- D) None of the above

3. What is the output?

```
HashSet<int> s = new HashSet<int> {1, 2, 3};
s.Add(2);
Console.WriteLine(s.Count);
```

- A) 2
- B) 3
- C) 4
- D) Compilation error

True/False

4. `HashSet<T>` maintains insertion order.
5. `HashSet<T>` is case-sensitive for strings by default.

Summary

- Use `HashSet<T>` for **unique, unordered data**.
- Ideal for **fast membership testing**.
- Supports efficient **set operations**.
- Prefer over legacy collections for **type safety** and **performance**.

upGrad ENTERPRISE

Q & A

Narasimha Rao T
tnrao.trainer@gmail.com