Collections and LINQ queries

Mirolim Majidov

1. List<T>

What is this?

List<T> is a dynamic array that can grow or shrink in size..

How/where can I use this?

Use List<T> when you need a dynamic, resizable array.

Example:

```
List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };
numbers.Add(6);
numbers.Remove(3);
```

2. Dictionary<Tkey, Tvalue>

What is this?

Dictionary<Tkey, Tvalue> is a collection of key-value pairs.

How/where can I use this?

Use Dictionary<Tkey, Tvalue> when you need to store data as key-value pairs.

Example:

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

3. ConcurrencyDictionary<Tkey, Tvalue>

What is this?

ConcurrencyDictionary<Tkey, Tvalue> is a thread-safe version of Dictionary<Tkey, Tvalue>.

How/where can I use this?

Use ConcurrencyDictionary<Tkey, Tvalue> in multithreaded scenarios to avoid race conditions.

Example:

```
ConcurrentDictionary<string, int> concurrentAgeMap = new ConcurrentDictionary<string, int>();
concurrentAgeMap["John"] = 25;
concurrentAgeMap["Alice"] = 30;
```

4. Span<T>

What is this?

Span<T> represents a contiguous region of arbitrary memory.

How/where can I use this?

Use Span<T> for efficient manipulation of contiguous memory regions.

Example:

```
int[] numbersArray = { 1, 2, 3, 4, 5 };
Span<int> numbersSpan = numbersArray.AsSpan();
```

5. Queue<T>

What is this?

Queue<T> is a collection that follows the First-In-First-Out (FIFO) principle.

How/where can I use this?

Use Queue<T> when you need to process elements in the order they were added.

Example:

```
Queue<string> tasks = new Queue<string>();
tasks.Enqueue("Task1");
tasks.Enqueue("Task2");
string nextTask = tasks.Dequeue();
```

6. Stack<T>

What is this?

Stack<T> is a collection that follows the Last-In-First-Out (LIFO) principle.

How/where can I use this?

Use Stack<T> when you need to process elements in reverse order.

Example:

```
Stack<int> numbersStack = new Stack<int>();
numbersStack.Push(1);
numbersStack.Push(2);
int lastAdded = numbersStack.Pop();
```

7. HashSet<T>

What is this?

HashSet<T> is a collection that does not allow duplicate elements.

How/where can I use this?

Use HashSet<T> when you need to ensure unique elements.

Example:

```
HashSet<string> uniqueNames = new HashSet<string>();
uniqueNames.Add("John");
uniqueNames.Add("Alice");
```

8. IEnumerable<T> and IQueryable<T>

What is this?

IEnumerable<T> represents a collection that can be enumerated.

IQueryable<T> represents a queryable collection that can be queried against a data source.

How/where can I use this?

Use IEnumerable<T> for in-memory collections, and IQueryable<T> for querying databases.

Example:

```
IEnumerable<int> numbers = new List<int> { 1, 2, 3, 4, 5 };
IQueryable<string> names = dbContext.Users.Select(u => u.Name);
```

9. LINQ Queries

What is this?

Language-Integrated Query (LINQ) provides a set of standard query operators.

How/where can I use this?

Use LINQ for querying collections in a concise and readable way.

Example:

```
6 references
public class User
{
    5 references
    public int Age { get; set; }
    5 references
    public string Name { get; set; }
}
```

```
var people = new List<User>() {
    new User{ Age = 10, Name ="Jake"},
    new User{ Age = 50, Name ="Jamshed"},
    new User{ Age = 35, Name ="Sanjar"},
    new User{ Age = 5, Name ="Alisa"},
    new User{ Age = 80, Name ="Sanjar"}
};

var adults = people.Where(p => p.Age > 30);
var jamshed = people.Single(p => p.Name == "Jamshed");
var sanjar = people.FirstOrDefault(p => p.Name == "Sanjar");
if (people.Any(p => p.Age > 50))
{
    //TODO
}
```