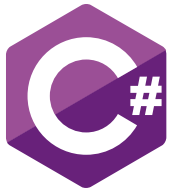
Other collections

Agenda

- Stacks
- Queues
- Hashsets
- Linked lists
- SortedList/SortedDictionary

Stack



Stack

Generic last-in-first-out collection

LIFO collection

Stack<T>

Add/Remove elements

Only at the top of the stack

`Push`/`Pop` items into/ from the stack

`Pop` removes and return the top element

Stack declaration and push

```
var stack = new Stack<string>();
stack.Push("First");
stack.Push("Second");
stack.Push("Third");
```

Initialization

No initialization

Use the **Push** method to add elements

Other members

`Count` number of elements in the stack

`Contains(T)` determines if an element is in the stack

`Peek()` Returns the object at the top of the stack

`TryPeek()` Indicates if there is an object at the top and gets it

Peek

```
var stack = new Stack<string>();
stack.Push("First");
stack.Push("Second");
stack.Push("Third");
Console.WriteLine(stack.Peek()); // Third
```

Null and duplicates

Accepts null reference

Accepts duplicates

Retrieve an element

Not possible retrieve an element directly

The `Pop` method returns the last added element

foreach

```
var stack = new Stack<string>();
stack.Push("First");
stack.Push("Second");
stack.Push("Third");
foreach (var element in stack)
{
    Console.WriteLine(element);
}
// Third
// Second
// First
```

Stack performance

Pushing/Poping an element is fast

To get an element in the stack, it must be enumerated

Stack performance

Push, Pop → O(1)

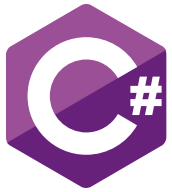
Contains → O(n)

Demo

Declare/Instantiate stacks

Use stack methods

Queue



Queue

Similar to Stack

Generic first-in-first-out collection

FIFO collection

`Queue<T>`

Add/Remove elements

Enqueue items at the end of the queue

Dequeue items at the beginning of the queue, removes and returns the element

Queue declaration and enqueue

```
var queue = new Queue<string>();
queue.Enqueue("First");
queue.Enqueue("Second");
queue.Enqueue("Third");
```

Initialization

No initialization

Use the enqueue method to add elements

Other members

`Count` number of elements in the queue

`Contains(T)` determines if an element is in the queue

`Peek()` Returns the object at the beginning of the queue

`TryPeek()` Indicates if there is an object at the top and gets it

Peek

```
var queue = new Queue<string>();
queue.Enqueue("First");
queue.Enqueue("Second");
queue.Enqueue("Third");
Console.WriteLine(queue.Peek()); // First
```

Null and duplicates

Accepts null reference

Accepts duplicates

Retrieve an element

Not possible retrieve an element directly

foreach

```
var queue = new Queue<string>();
queue.Enqueue("First");
queue.Enqueue("Second");
queue.Enqueue("Third");
foreach (var element in queue)
{
    Console.WriteLine(element);
}
// First
// Second
// Third
```

Queue performance

Enqueuing/Dequeuing an element is fast

To get an element in the queue, it must be enumerated

Queue performance

Enqueue, Dequeue → O(1)

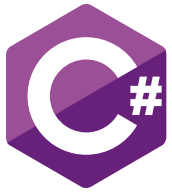
Contains → O(n)

Demo

Declare/Instantiate queues

Use queue methods

HashSet



HashSet

Represents a set of values without duplicates

Set of values

`HashSet<T>`

HashSet and dictionaries

HashSet is similar to dictionaries

There’s no pair of key values, but the value is the key

Initialization

No initialization

Use the `Add` method to add elements

HashSet

Elements are not ordered

Elements have no key

Elements have no index

No way to find a particular element

You must enumerate the collection to retrieve an element

Duplicates

Doesn’t accept duplicates because the value is used as the key

Doesn’t throw an exception like dictionaries, duplicates are ignored

Add/Remove elements

`Add` method adds an element to the set

`Remove(T)` removes the specified element

HashSet declaration and Add

```
var hashSet = new HashSet<int>();
hashSet.Add(1);
hashSet.Add(2);
```

Other members

`Count` number of elements in the set

`Contains(T)` determines if an element is in the set

`IntersectWith(IEnumerable<T>)` Returns the intersection of two sets

`UnionWith(IEnumerable<T>)` Returns the union of two sets

Retrieve an element

Not possible retrieve an element directly

Hashset performance

Add → O(1) or O(n)

Remove → O(1)

Contains → O(1)

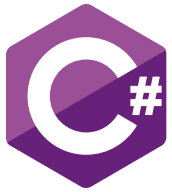
HashSet is great to check if an element is already contained in the set, to add or remove elements

Demo

Declare/Instantiate hashsets

Use hashset methods

Linked lists



LinkedList

Represents a doubly linked list

Doubly linked list

`LinkedList<T>`

Doubly linked ?

Each element is a `LinkedListNode`

Each node has a `Next` and `Previous` property

Initialization

No initializer

Use the Add... methods to add elements

Declaration instantiation

```
var train = new LinkedList<string>();
```

Add elements

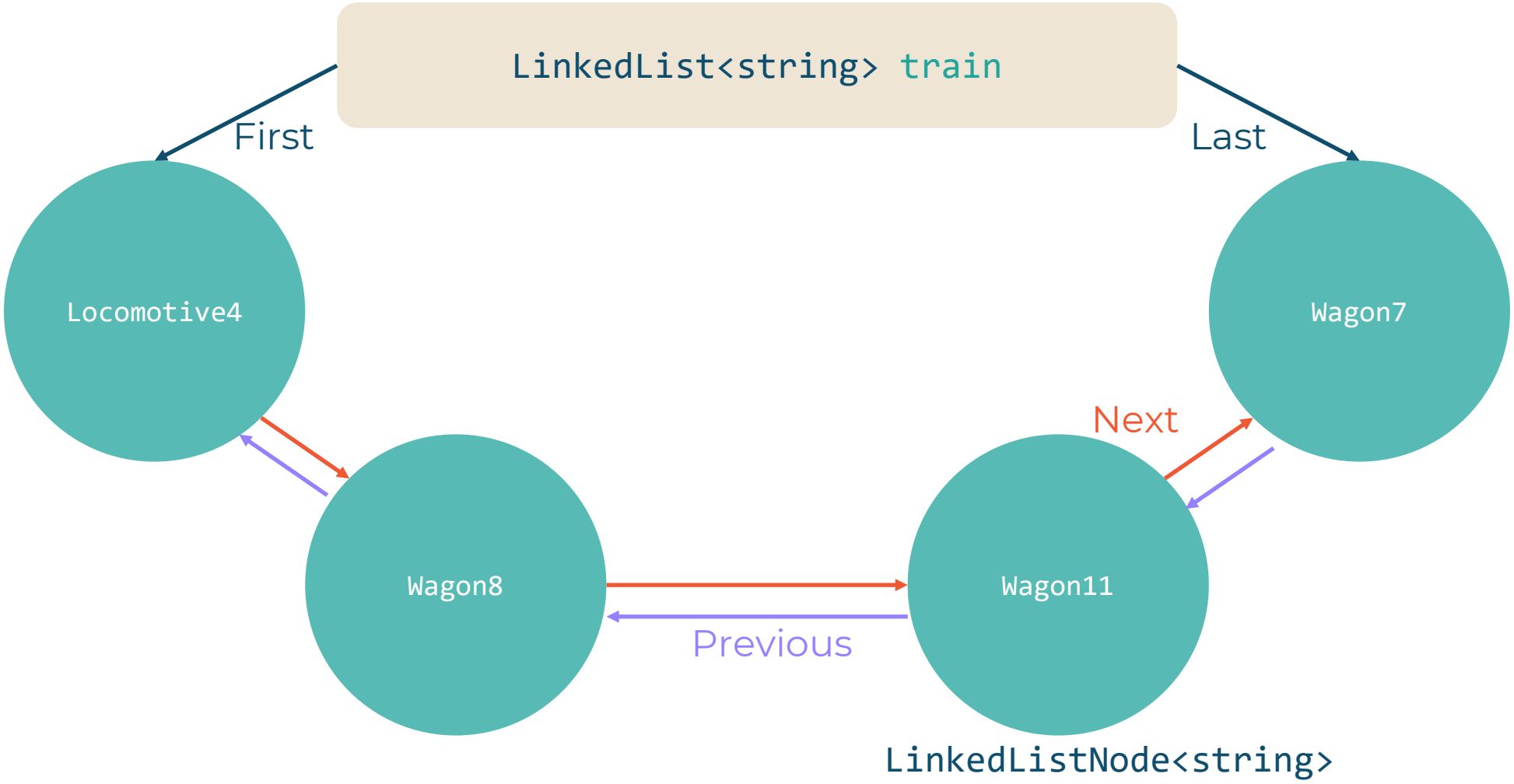
```
train.AddFirst("Locomotive4");
train.AddLast("Wagon8");
train.AddLast("Wagon11");
train.AddLast("Wagon7");
```

Under the hood

Elements are not stored in one single memory block

Elements can be stored anywhere on memory

Because of `Next` and `Previous` properties


LinkedList<string> train
First
Last
Locomotive4
Wagon7
Next
Wagon8
Wagon11
Previous
LinkedListNode<string>

Value property

An element is accessible through the `Value` property of the `LinkedListNode`

foreach

```
foreach (var trainNode in train)
{
    Console.Write($"-({trainNode})-");
}
// -(Locomotive4)--(Wagon8)--(Wagon11)--(Wagon7)-
```

Retrieve an element

Only by enumerating the list, the `Find` method enumerates the list

Find method

```
var element = train.Find("Wagon11");
Console.WriteLine(element.Value);
// Wagon11
```

Other members

`Count` Number of elements in the list

`First, Last` First and last nodes in the list

Other members

`AddAfter(LinkedListNode<T>, T)` Add a new node containing the specified value after the specified node

`AddFirst(T)` Add a new node containing the specified value at the start of the list

`AddLast(T)` Add a new node containing the specified value at the end of the list

`Find(T)` Finds the first node that contains the specified value

`Remove(T)` Removes the first occurrence of the specified value from the list

LinkedList performance

AddAfter → O(1)

AddFirst, AddLast → O(1)

Find → O(n)

Remove → O(n)

LinkedList has great performance for adding/Removing elements

LinkedListNode

Represents a node in a `LinkedList`

LinkedList node

`LinkedListNode<T>`

Properties

`List` the lists it belongs to

`Next` the next node

`Previous` the previous node

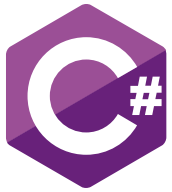
`Value` the value contained in the node

Demo

Declare/Instantiate linked list and nodes

Use linked list methods, add nodes to the list

SortedDictionary



SortedDictionary

Collection of key/value pairs sorted on the key

Sorted key/value pair collection

`SortedDictionary<Tkey,TValue>`

Keys

Keys are unique and cannot be `null`

Value

A value can be `null`

SortedDictionary declaration and instantiation

```
var sortedDictionary = new SortedDictionary<int, Movie>();
```

Initializer

```
var sortedDictionary = new SortedDictionary<int, Movie>()
{
    {3, new Movie {Id = 3, Title = "Title 3"}},
    {2, new Movie {Id = 2, Title = "Title 2"}},
    {1, new Movie {Id = 1, Title = "Title 1"}},
};
```

foreach

```
foreach (var element in sortedDictionary)
{
    Console.WriteLine(element.Value.Title);
}
// Title 1
// Title 2
// Title 3
```

Add/Remove elements

`Add(TKey,TValue)` adds an element

`Remove(TKey)` remove an element from its key

Order

SortedDictionary is sorted on the key

You can rely on the order of a `foreach`

Other members

`Count` number of elements

`Capacity` number of elements in the internal data structure that contains the elements can hold

`Item[TKey]` Value for specified key

`Keys` contains all the keys

`Values` contains all the values

Other members

`ContainsKey(TKey,TValue)` Determines if contains the specified key

`ContainsValue(TKey,TValue)` Determines if contains the specified value

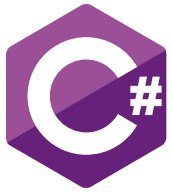
`TryGetValue(TKey,TValue)` Gets the value associated with the specified key

Retrieving, Adding or removing an element is fast for a SortedDictionary

Dictionary performance

```
Add → O(log n)
Remove → O(log n)
[] → O(log n)
```

SortedList



It’s a dictionary, not a list

SortedList

```
SortedList<Tkey,TValue>
```

SortedList

Same as `SortedDictionary`

Uses less memory

O(n) for modifications

Demo

Declare/Instantiate sorted dictionaries

Use sorted dictionary methods

Summary

A stack is a generic LIFO collection

A queue is a generic FIFO collection

Stacks and queues are performant for adding and removing elements

A hashset is a collection of unique elements performant for checking the existence of an element or adding and removing elements

A LinkedList is a doubly linked list and each element is wrapped in a LinkedListNode instance performant for adding and removing elements

A SortedList is a collection of key/value pairs sorted on the key great for retrieving, adding and removing elements