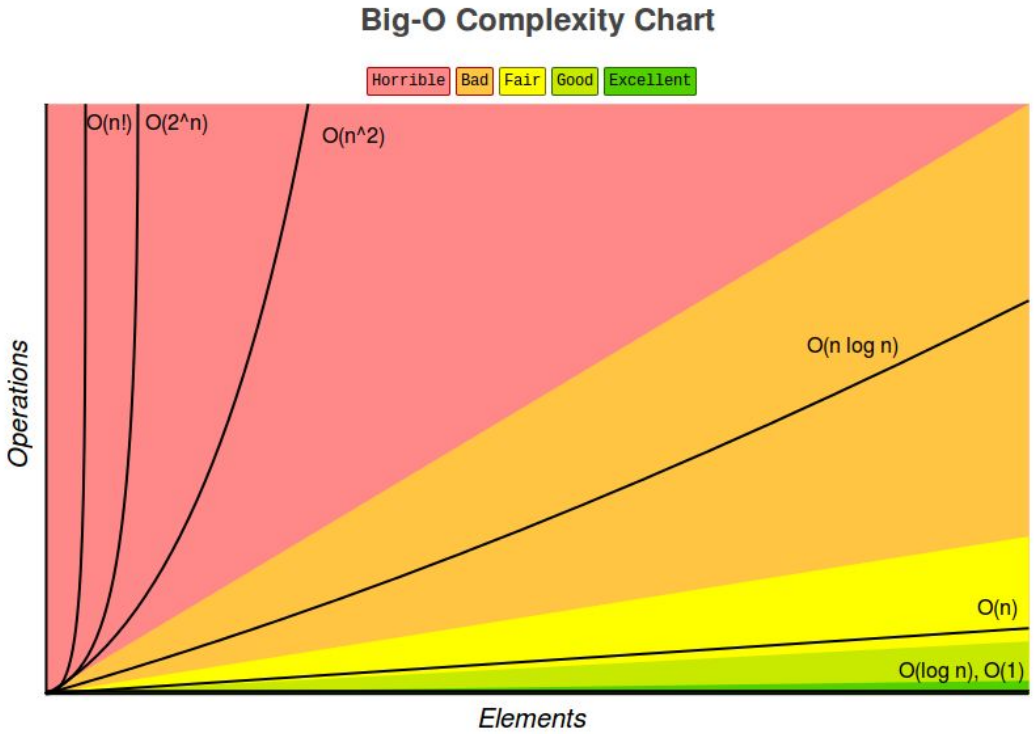
Strutture dati avanzate

Alcune strutture dati di comune utilità

Complessità

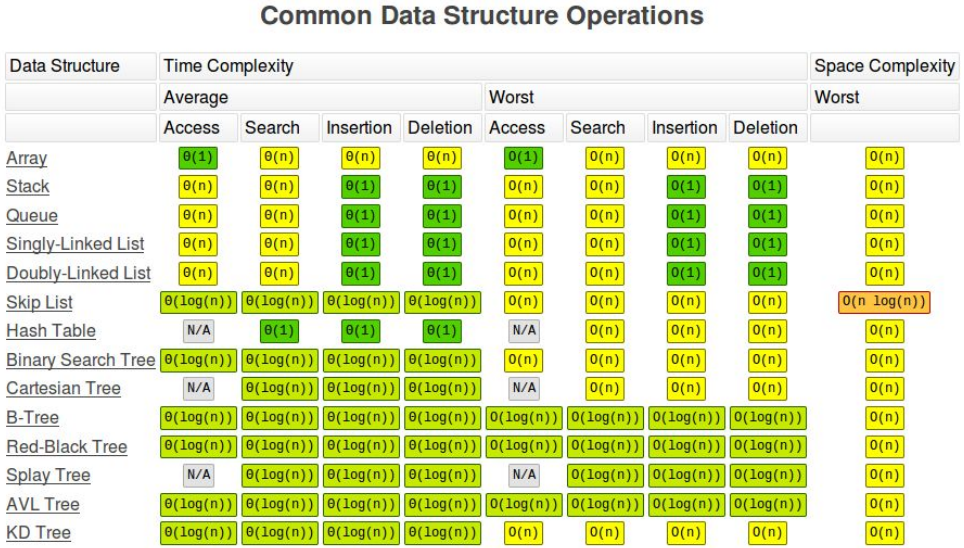
Perché è importante valutare la **complessità** di un algoritmo? Perché ci fa capire quanto il nostro software è **scalabile***.



* **Scalabilità**: In informatica, la caratteristica di un sistema software o hardware facilmente modificabile nel caso di variazioni notevoli della mole o della tipologia dei dati trattati.

Il framework .NET

Molti algoritmi e strutture dati sono già implementati nel framework .NET; percui è importante conoscere come lavorano e quali sono le loro caratteristiche in termini di performance e complessità di memoria.



Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Strutture dati in .NET

Nel framework .NET sono già **implementate** molte tipologie di strutture dati.

Possiamo suddividere le strutture dati in **semplici** e **generiche**.

Inizieremo a vedere le strutture dati **semplici**, cioè quelle strutture dati **non tipizzate** e che quindi possono ospitare qualsiasi tipo di dato.

Vedremo poi anche le strutture dati **generiche**, cioè quelle strutture dati tipizzate alla loro dichiarazione e quindi che possono memorizzare solo il tipo di dato scelto.

Le strutture dati semplici sono raggruppate insieme nel **namespace System.Collections**

Le strutture dati generiche sono raggruppate nel **namespace System.Collections.Generic**

***Namespace**: ambito che contiene un set di oggetti correlati così da organizzare il codice ed evitare collisione causate da elementi con lo stesso nome.

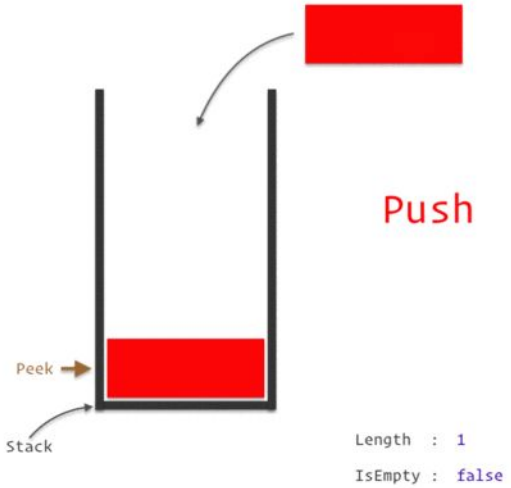
ArrayList (l'evoluzione dell'array)

- ricorda la **lista della spesa**
- gli elementi inseriti nella lista mantengono **l'ordine** con cui sono stati **aggiunti**
- la lista si **estende automaticamente** a seconda di quanti elementi vengono inseriti; a differenza dell'array che andava preventivamente dimensionato alla dichiarazione

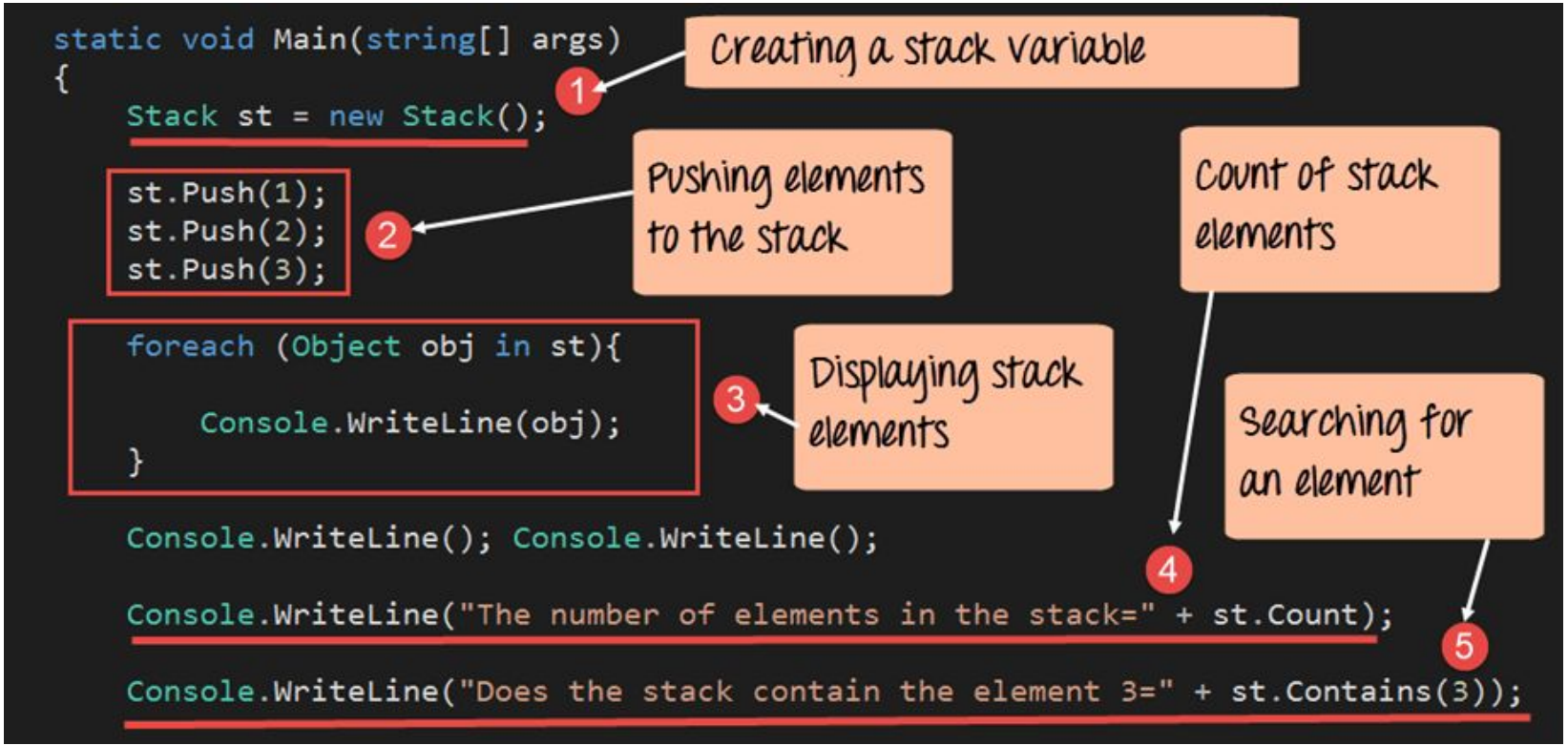
Esempio ArrayList

Stack (pila)

- ricorda una **scatola** o una pila di vestiti
- **LIFO: L**ast **I**n **F**irst **O**ut
- si auto estende al bisogno

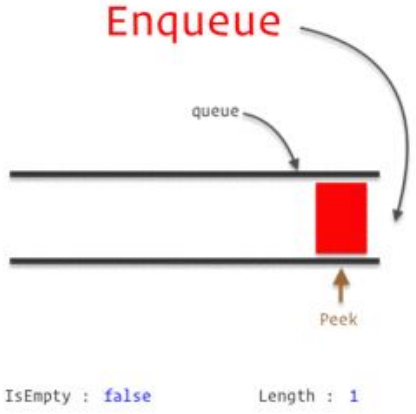


Esempio Stack

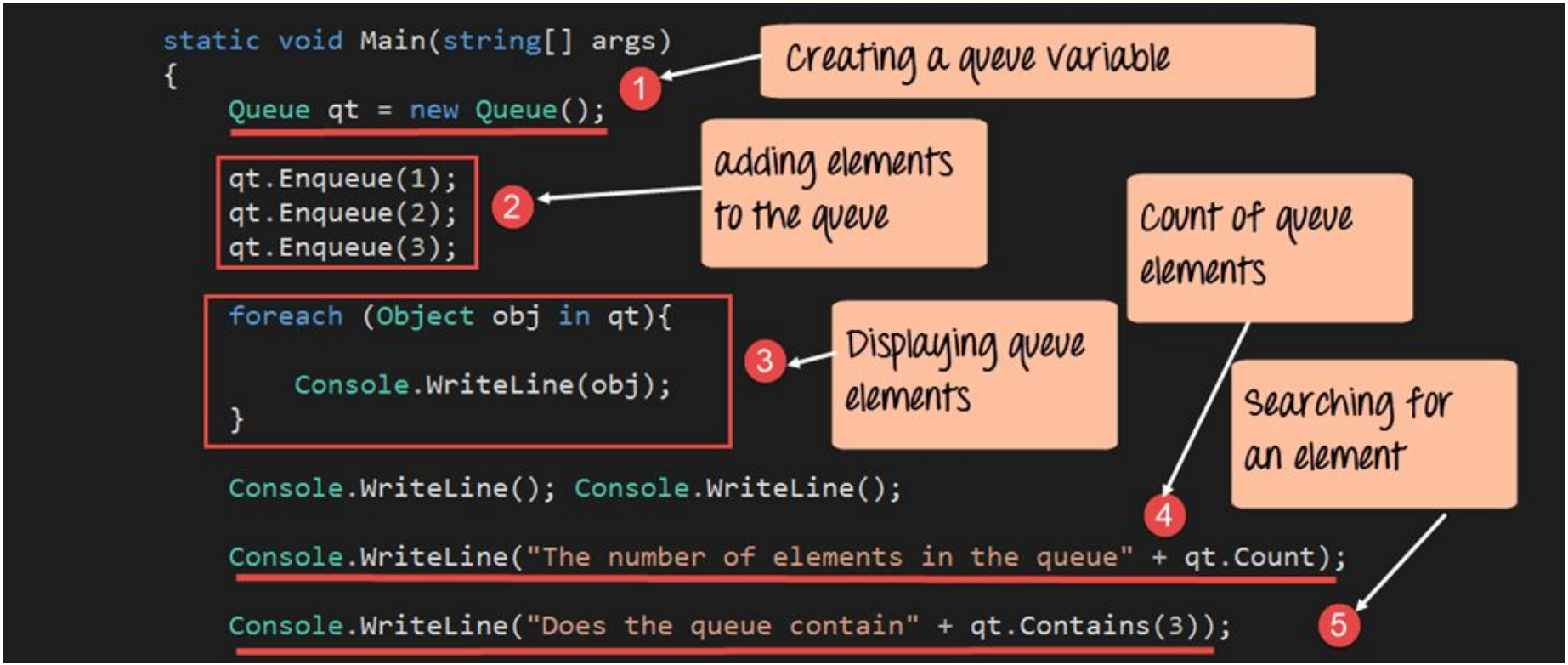


Queue (coda)

- ricorda la coda alla **cassa del supermercato**
- **FIFO**: **F**irst **I**n **F**irst **O**ut
- si auto estende al bisogno



Esempio Queue



Linked List (lista collegata)

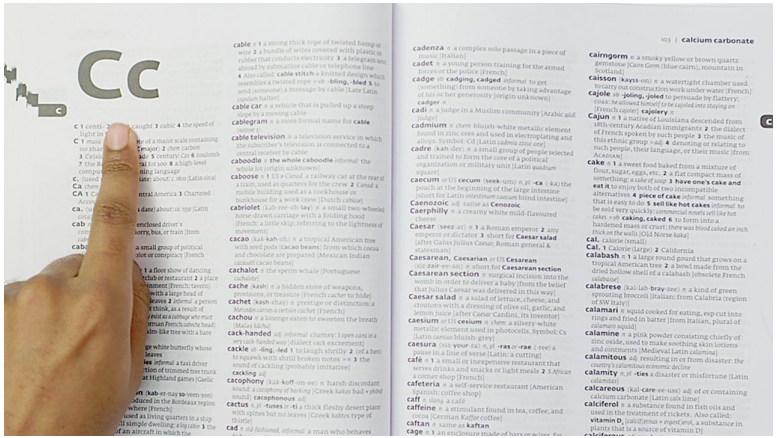
- ricorda un **treno** con i suoi **vagoni**
- ogni **nodo** è **collegato** al successivo
- la **testa** non ha un nodo precedente
- la **coda** non ha un nodo successivo
- si auto estende al bisogno
- è una struttura dati generica
- .NET namespace:
  System.Collections.Generic

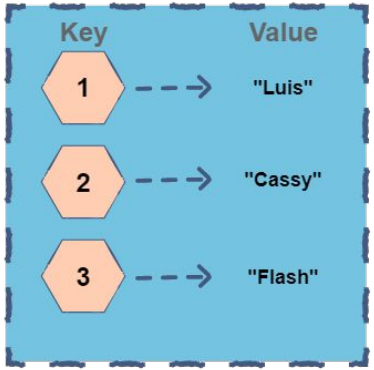
Esempio LinkedList

```
LinkedList<int> lk = new LinkedList<int>();
lk.AddFirst(1);
var node2 = new LinkedListNode<int>(2);
lk.AddLast(node2);
lk.AddLast(3);
lk.AddAfter(node2, 5);
foreach(int n in lk){
    Console.WriteLine(n);
}
```

Hashtable

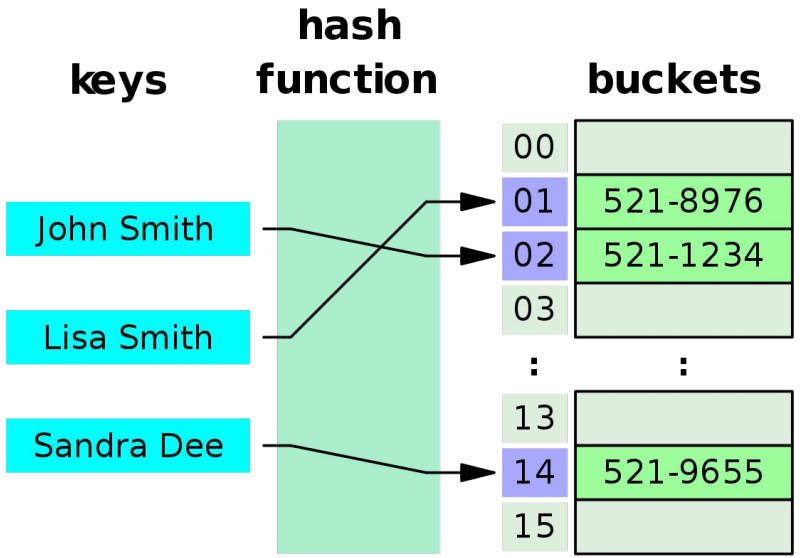
- ricorda un **dizionario**
- utilizza la tecnica di **hash** per memorizzare i dati
- questa tecnica rende la **ricerca** molto veloce: **O(1)**
- si auto estende al bisogno
- l'ordine degli elementi non è mantenuto



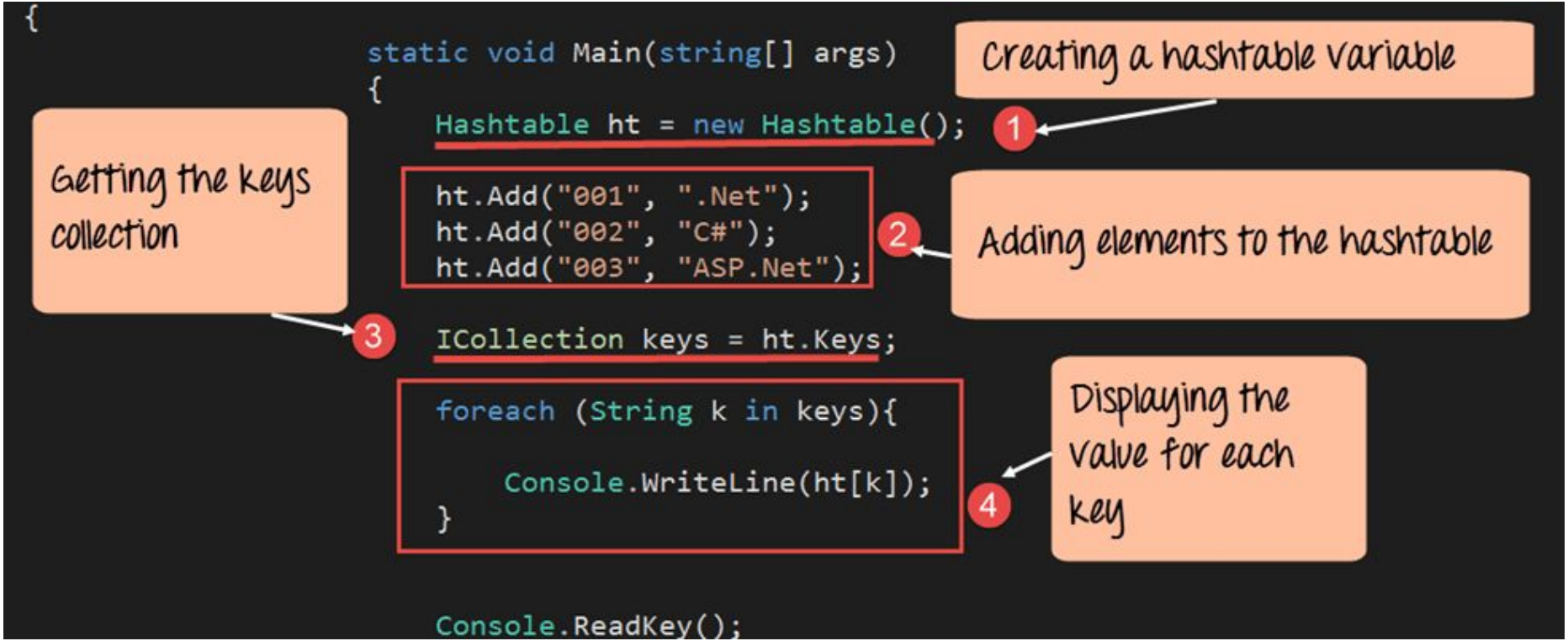


Tecnica di hashing

Tecnica secondo la quale una chiave viene convertita, per mezzo di una funzione matematica di hash, in un valore numerico univoco.



Esempio Hashtable



Algoritmi in .NET

- Ricerca binaria in array: metodo **Array.BinarySearch()**
- Ricerca sequenziale in Stack, Queue, LinkedList: metodo **Contains()**
- Ricerca sequenziale in Hashtable: metodo **ContainsKey()** e **ContainsValue()**
- Ricerca per chiave in Dictionary: **TryGetValue()** o accesso per chiave.