.NET Full Stack

Development Program

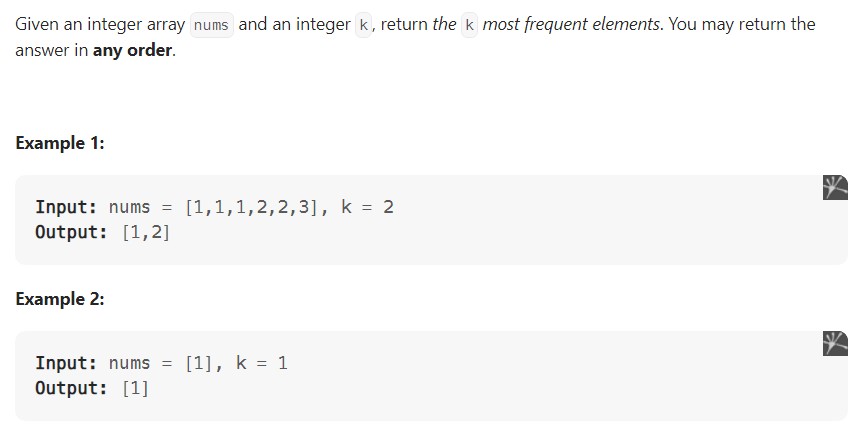
#### LINQ & Thread

Outline

* LINQ
  + What is LINQ?
  + LINQ to Objects
  + Standard Query Operators
  + LINQ to SQL
* Thread
  + Thread & Process
  + Thread Class
  + Thread Life Cycle
  + Thread Problem / Thread Safety
  + Thread Synchronization
  + Dead Lock

### Question

Top K Frequent Elements



## LINQ

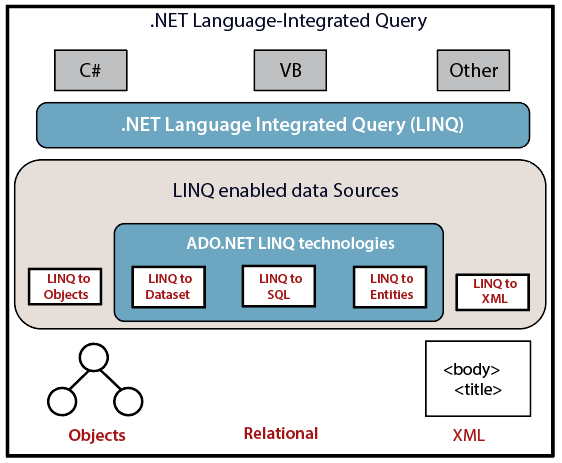
#### What is LINQ?

* LINQ(Language Integrated Query) provides us with common query syntax which allows us to query the data from various data sources.
* It was introduced by Microsoft with .NET Framework 3.5 and C# 3.0 and is

available in **System.Linq** namespace.

* It supports a consistent query experience(LINQ Provider)
* LINQ to objects
* LINQ to SQL
* LINQ to XML
* …

How does LINQ work?



LINQ Providers

* **LINQ to objects** (No provider needed): allows us to query in-memory objects from an array, collection and generics types.
* **LINQ to XML**(XLINQ): works with XML documents.
* **LINQ to SQL**(DLINQ): works with the SQL Server database.
* **LINQ to Datasets**: provides us the flexibility to query data cached in a Dataset.
* **LINQ to Entities**: is used to query any database(including SQL Server, Oracle, MySQL etc.)

LINQ to Objects

The term "LINQ to Objects" refers to the use of LINQ queries with any [IEnumerable](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ienumerable) or [IEnumerable<T>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.ienumerable-1) collection directly, without the use of an intermediate LINQ provider or API such as [LINQ to SQL](https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/linq/) or [LINQ to XML](https://docs.microsoft.com/en-us/dotnet/standard/linq/linq-xml-overview). You can use LINQ to query any enumerable collections such as [List<T>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1), [Array](https://docs.microsoft.com/en-us/dotnet/api/system.array), or [Dictionary<TKey,TValue>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2). The collection may be user-defined.

Advantages:

* + They are more concise and readable, especially when filtering multiple conditions.
  + They provide powerful filtering, ordering, and grouping capabilities with a minimum of application code.
  + They can be ported to other data sources with little or no modification.

#### IEnumerable

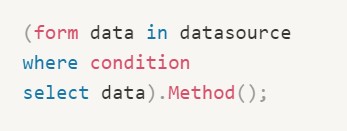
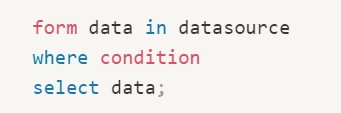
* IEnumerable is **an interface that defines one method, GetEnumerator() which returns an IEnumerator type**.
* The IEnumerable interface is a type of iteration design pattern. It means we

can **iterate on** the collection of the type IEnumerable.

* Basically, you can write LINQ queries for any type that supports [IEnumerable](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ienumerable) or the generic [IEnumerable<T>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.ienumerable-1) interface.

#### Different ways to write a LINQ

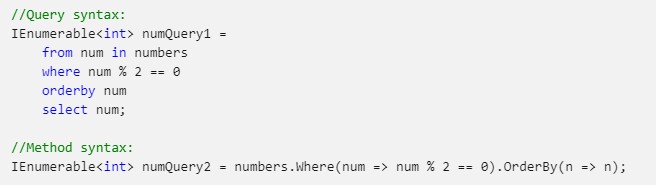
* Query Syntax
* Method Syntax
* Mixed Syntax (Query + Method)



### Query Syntax vs. Method Syntax

Most queries in the introductory Language Integrated Query (LINQ) documentation are written by using the LINQ declarative query syntax. However, the query syntax must be translated into method calls for the .NET common language runtime (CLR) when the code is compiled. These method calls invoke the standard query operators, which have names such as **Where, Select, GroupBy, Join, Max, and Average**. You can call them directly by using **method syntax** instead of query syntax.

There will be some queries that must be expressed as method calls. you must use a method call to express a query that retrieves the number of elements that match a specified condition. You also must use a method call for a query that retrieves the element that has the maximum value in a source sequence.



LINQ Queries

* + All LINQ query operations consist of three distinct actions:
    1. Obtain the data source
    2. Create the query
    3. Execute the query

c 1 as s Int roTo LIfTQ

static void Main()

// The 7hree Parts of a LIMQ Query:

// 1. Data sounce.

int[] numbers = nets int[7] ( 0, 1, 2, 3, 4, 5, 6 };

// 2. Query creation.

// numQuery is an IEnumerable<int> var numQueny =

from num in numbers l•/here ( nun R 2) == 8 select nun;

// 3. Query execution. foreach (int num in numQuery)

Console.Mrite( 1] " , num) ;

### Create the Query

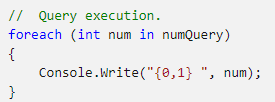
The query expression contains **three clauses**: from, where and select

The **from** clause specifies the data source, the **where** clause applies the filter, and the **select** clause specifies the type of the returned elements.

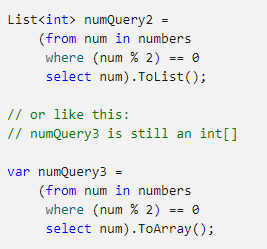
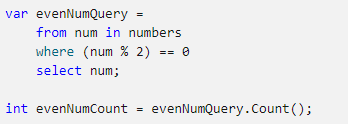


### Execute the Query

The actual execution of the query is **deferred** until you iterate over the query variable in a foreach statement.



### Manners Of Execution



LINQ operators are divided into 2 categories:

* Deferred Execution

Deferred execution means that the operation is not performed at the point in the code where the query is declared. The operation is performed only when the query variable is enumerated, for example by using a foreach statement.

Almost all the standard query operators whose return type is IEnumerable<T>

Ex: Select(), SelectMany(), Where(), etc.

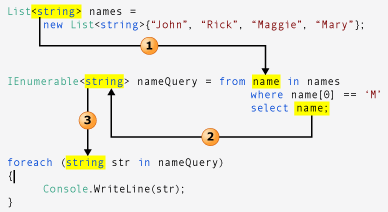
* Immediate Execution

Immediate execution means that the data source is read and the operation is performed at the point in the

code where the query is declared.

Ex: ToArray(), ToList(), Aggregate Methods: etc.

### Type Relationships in LINQ



Standard LINQ Operators

Standard Query Operators

* The ***standard query operators*** are the **methods** that form the LINQ pattern. Most of these methods operate on sequences, where a sequence is an object whose type implements the [IEnumerable<T>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.ienumerable-1) interface. The standard query operators provide query capabilities including **filtering, projection, aggregation, sorting and more.**
* Query Expression Syntax for Standard Query Operators:
  + Cast, GroupBy, GroupJoin, Join, OrderBy, OrderByDescending, Select, SelectMany,

ThenBy, ThenByDescending, Where, and more



Classification Standard Query Operators

Filtering SOAIO§

Grouping Jain PrajectiDn Aggregation Quantifiers Elements Set Partitioning

Concatenation Equal ity Generation Conversion

Where, OfType

OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse GraupBy, ToLaokup

Graup3ain, Join Select, SelectMany

Aggregate, Average, Count, LDN§CDUFlt, hay, Min, sum All, Any, CDntains

EIementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, single, singleOrDefault Distinct, Except, Intersect, U nian

Skip, skipWhile, Take, TakeWhile Concat

SequenceEqual

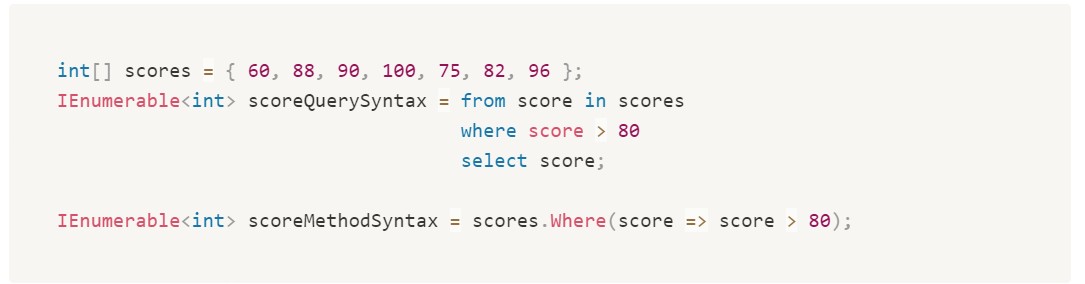
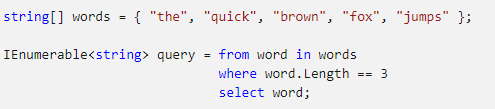
DefaultEmpty, Empty, Range, Repeat

AsEnumerable, AsQueryable, Cast, TaArray, TaDictianary, ToList



### Filtering Data

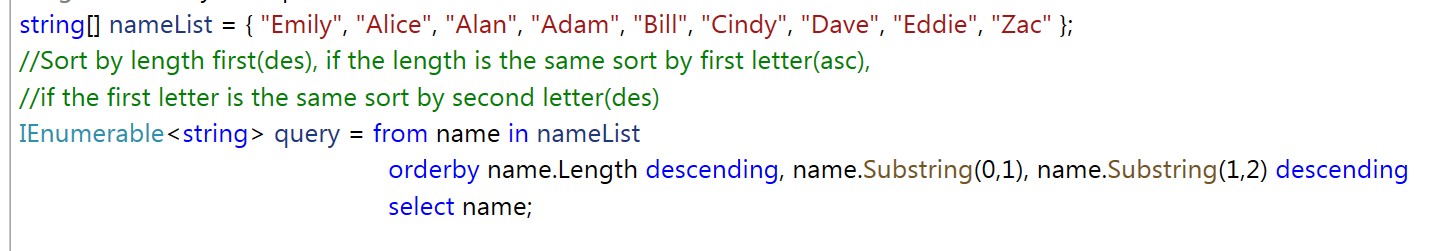
Filtering refers to the operation of restricting the result set to contain only those elements that satisfy a specified condition

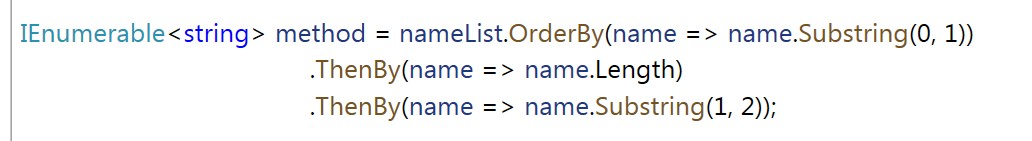


### Sorting Data

Sorting operations order the elements of a sequence based on one or more attributes:

OrderBy() | OrderByDescending() | ThenBy() | ThenByDescending()





### Set Operators

* Distinct 

Removes duplicate values from a collection.

* Except 

Returns the set difference, which means the elements of one collection that do not appear in a second collection.

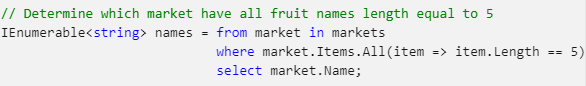
* Intersect 

Returns the set intersection, which means elements that appear in each of two collections

* Union 

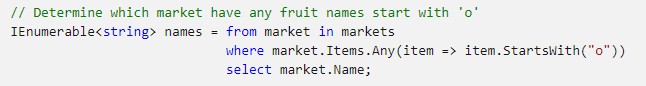
Returns the set union, which means unique elements that appear in either of two collections

### Quantifier Operation

Quantifier operations return a [Boolean](https://docs.microsoft.com/en-us/dotnet/api/system.boolean) value that indicates whether some or all of the elements in a sequence satisfy a condition

* All

Determines whether all the elements in a sequence satisfy a condition.



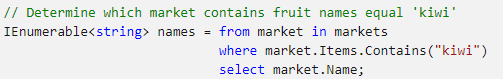
Any

Determines whether any elements in a sequence satisfy a condition

●

* Contains

Determines whether a sequence contains a specified element



### Projection Operation

Projection refers to the operation of transforming an object into a new form that often consists only of those properties that will be subsequently used.

* Select

Projects values that are based on a transform function

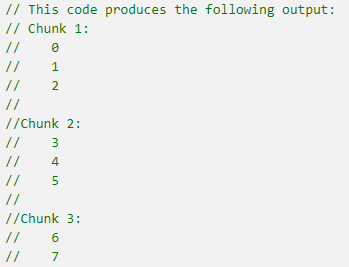
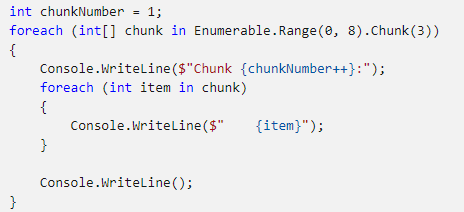
* SelectMany

Projects sequences of values that are based on a transform function and then flattens them into one sequence

### Partitioning Data

Partitioning in LINQ refers to the operation of dividing an input sequence into two sections, without rearranging the elements, and then returning one of the sections

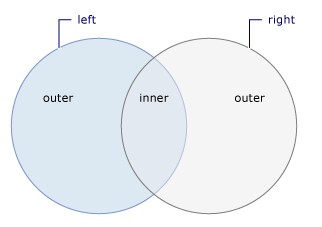
* Skip: Skips elements up to a specified position in a sequence
* SkipWhile: Skips elements based on a predicate function until an element does not satisfy the condition
* Take: Takes elements up to a specified position in a sequence
* TakeWhile: Takes elements based on a predicate function until an element does not satisfy the condition
* Chunk: Splits the elements of a sequence into chunks of a specified maximum size



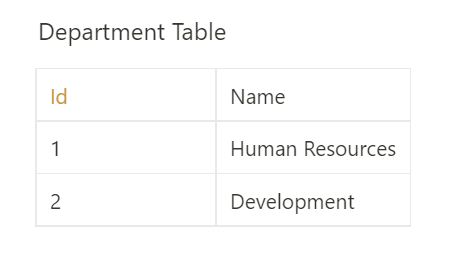
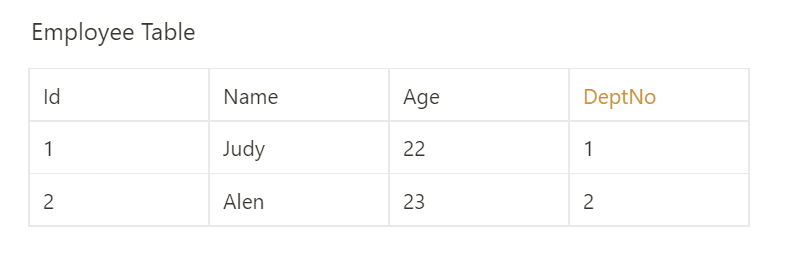
### Join Operation

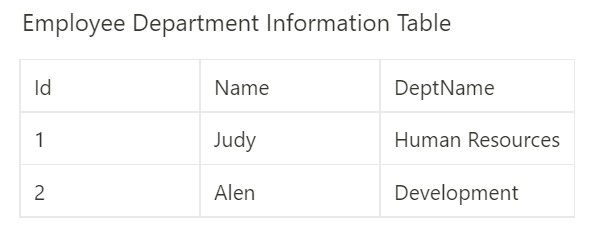
A ***join*** of two data sources is the association of objects in one data source with objects that share a

common attribute in another data source.

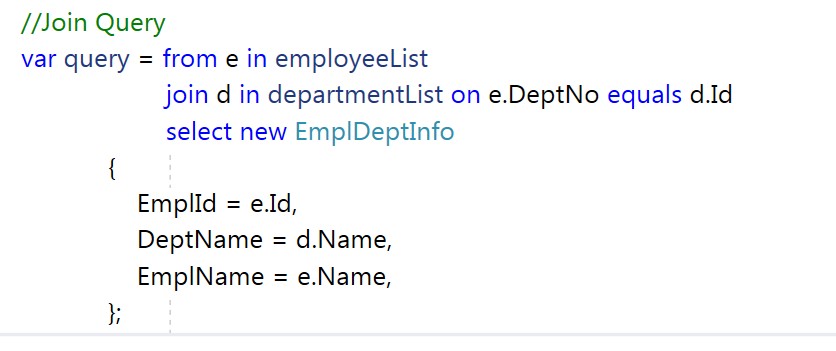


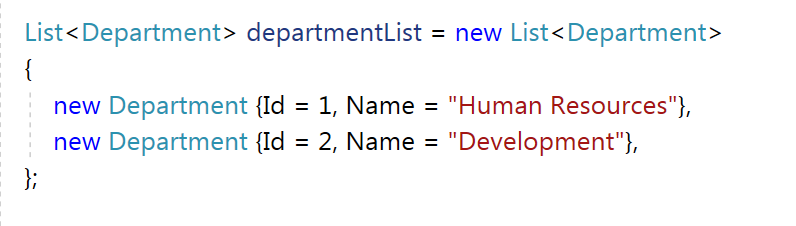
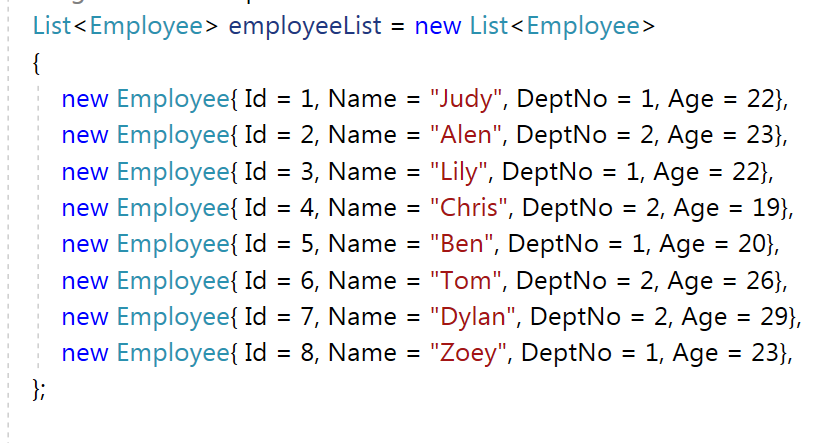
### Join Operator



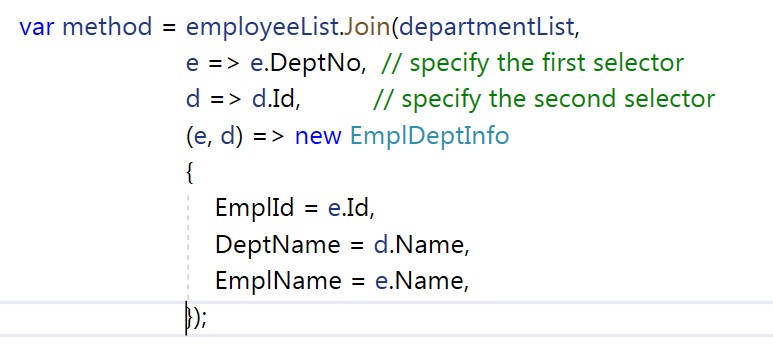


Join Operation





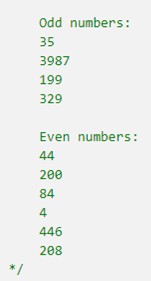
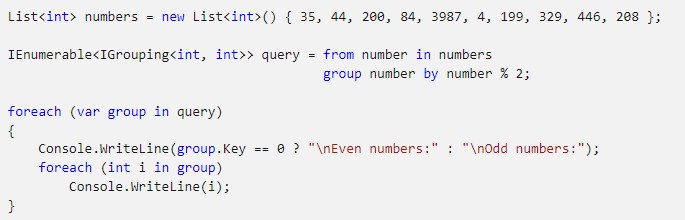
Join Operation



Grouping Data

* GroupBy()/group by: takes a flat sequence of elements and then organizes the elements into groups, based on a given key, it will return an **IEnumerable<IGrouping<TKey,**

**TSource>>** where **TKey** is nothing but the **Key** value on which the grouping has been formed

and **TSource** is the collection of elements that matches the grouping key value.

### Aggregate Methods

* [**Sum():**](https://dotnettutorials.net/lesson/linq-sum-method/) This method is used to calculate the total(sum) value of the collection.
* [**Max():**](https://dotnettutorials.net/lesson/linq-max-method/) This method is used to find the largest value in the collection.
* [**Min():**](https://dotnettutorials.net/lesson/linq-min-method/) This method is used to find the smallest value in the collection.
* [**Average():**](https://dotnettutorials.net/lesson/linq-average-method/) This method is used to calculate the average value(*double*) of the numeric type of the collection.
* **Count():** This method is used to count the number of elements present in the collection.



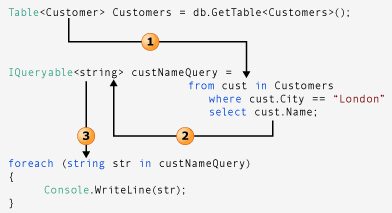
int[] arr = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

int sum = arr.Sum(); int max = arr.Max(); int min = arr.Min();

double avg = arr.Average(); int count = arr.Count();

## LINQ to SQL

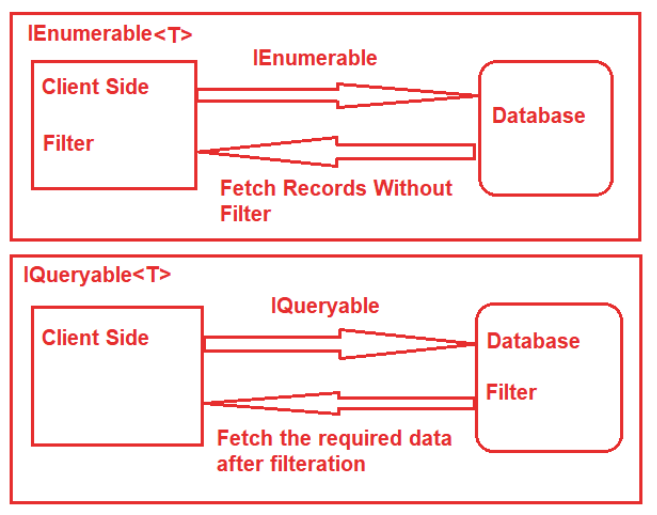
### LINQ to SQL



IQueryable

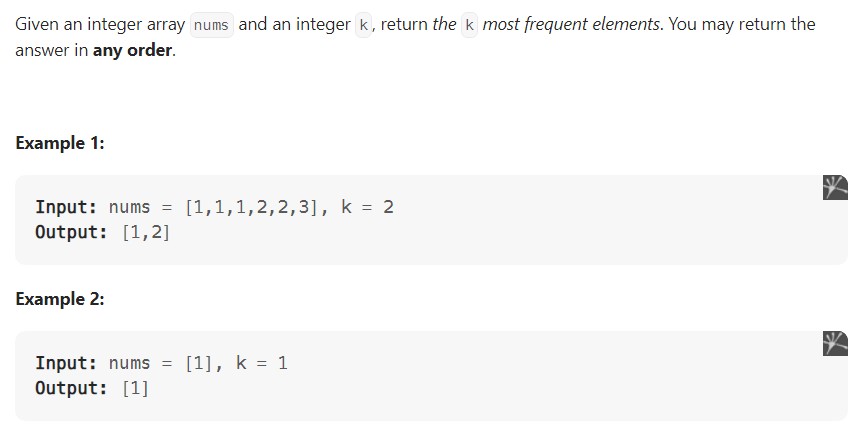
* The [IQueryable](https://learn.microsoft.com/en-us/dotnet/api/system.linq.iqueryable?view=net-7.0) interface inherits the [IEnumerable](https://learn.microsoft.com/en-us/dotnet/api/system.collections.ienumerable?view=net-7.0) interface so that if it represents a query, the results of that query can be enumerated.
* We can use AsQueryable() and AsEnumerable() to convert them.

### IEnumerable vs IQueryable

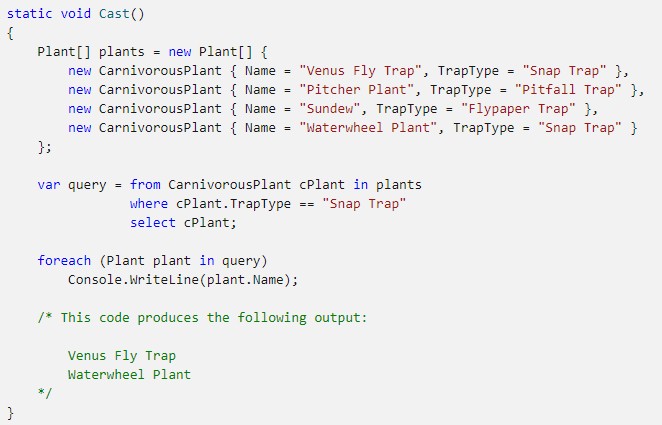
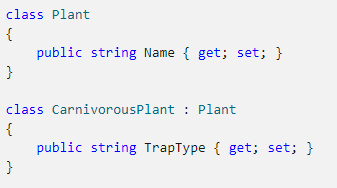


Question

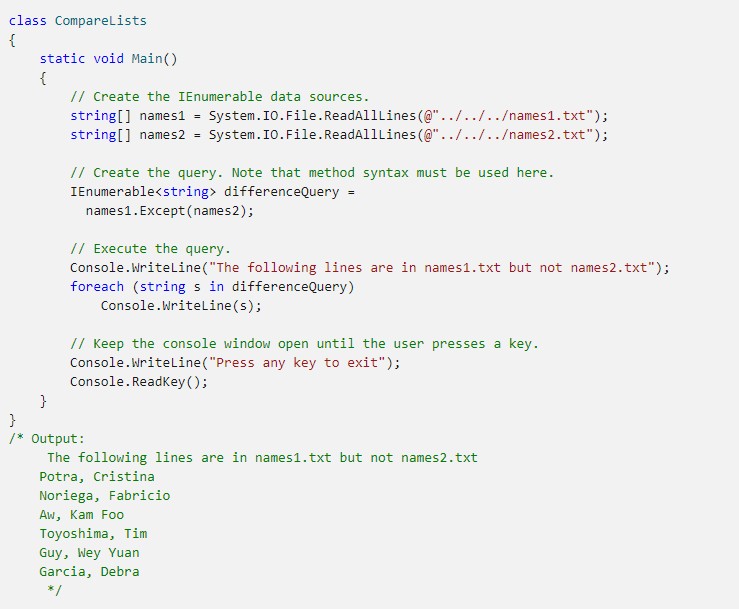
Top K Frequent Elements



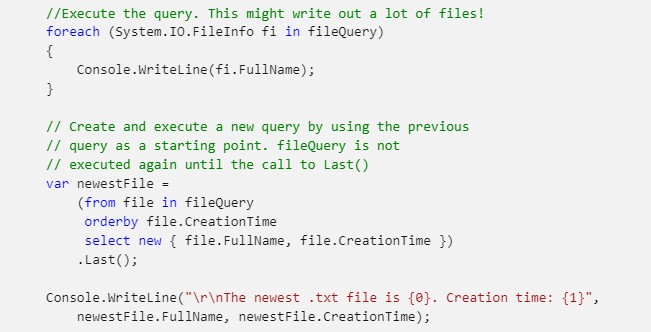
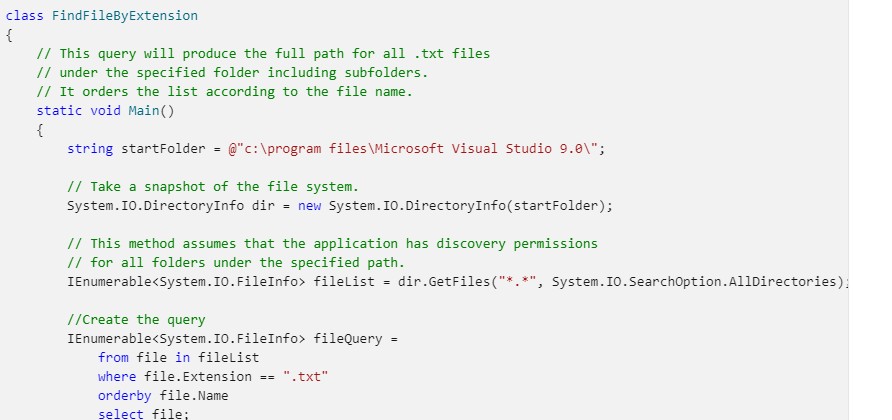
### Casting



Scenario: Find the set difference between two lists



Scenario: Query for files with a specified attribute



Thread

* Thread
  + Thread & Process
  + Thread Class
  + Thread Life Cycle
  + Thread Problem / Thread Safety
  + Thread Synchronization
  + Dead Lock

# Thread & Process

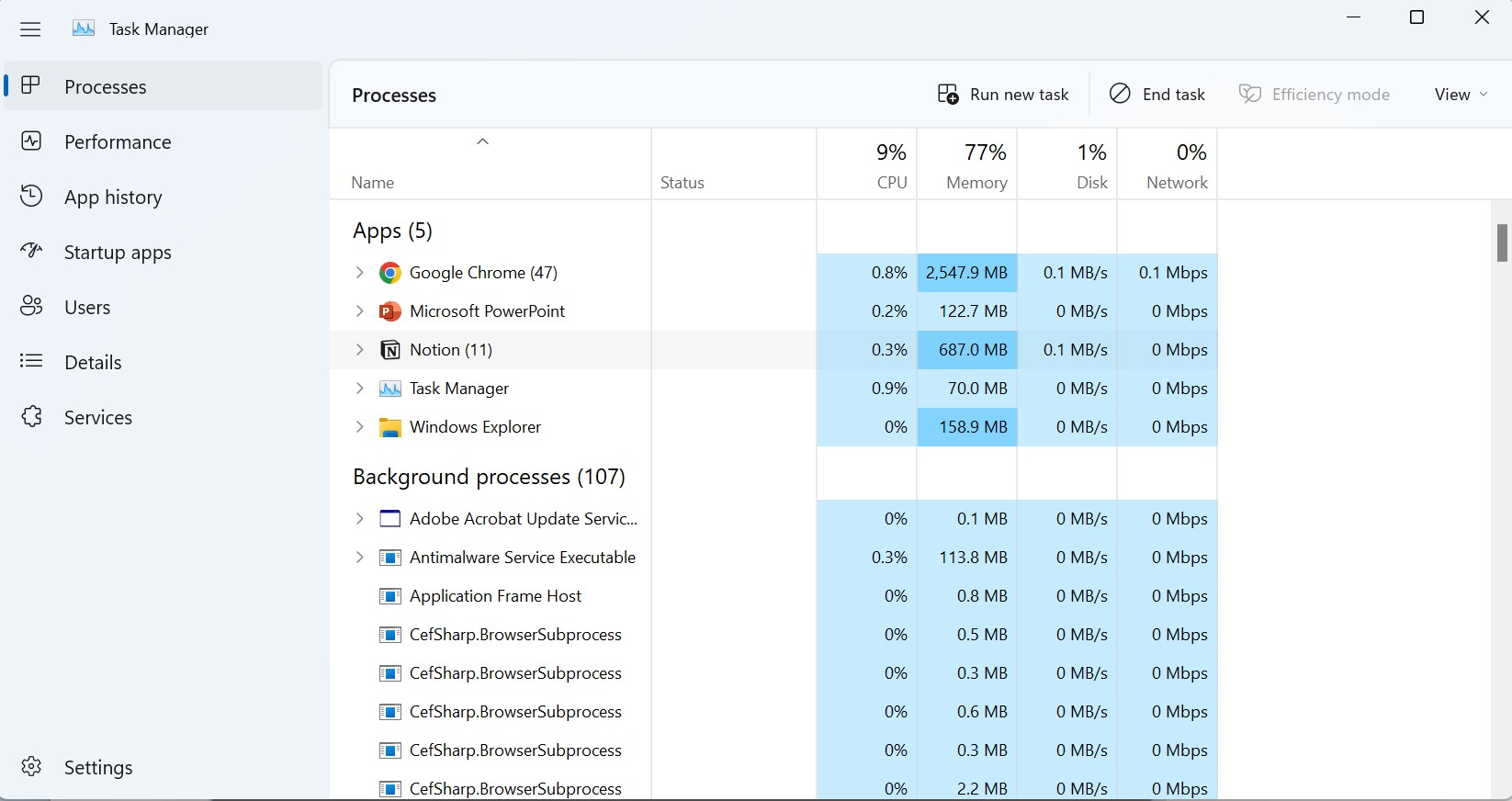
### Process

* A process has a self-contained execution environment.

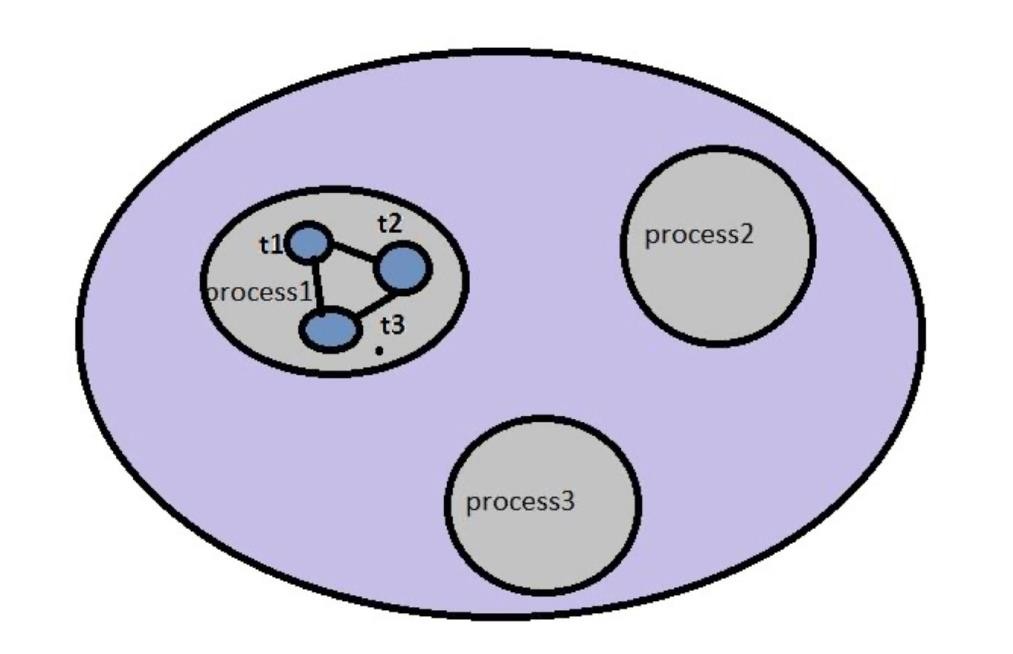
A process generally has a complete, private set of basic runtime resources; in particular, each process has its own memory space.

* Processes are often seen as synonymous with programs or applications However, what the user sees as a single application may in fact be a set of cooperating processes.
* Eg.To facilitate communication between processes, most operating systems support *Inter Process Communication* (IPC) resources, such as pipes and sockets

### Process



Thread vs. Process



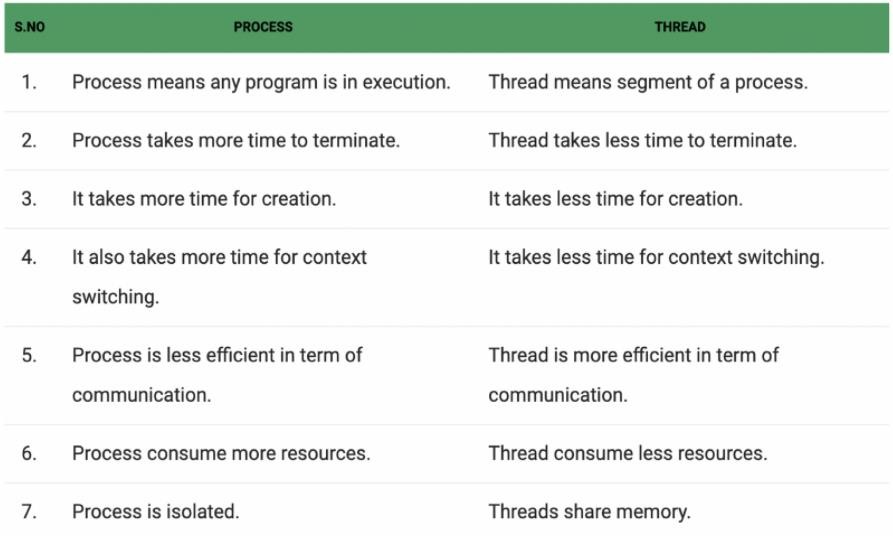
Thread

* A thread is a lightweight sub-process, the smallest unit of processing. It has a separate path of execution.
* Threads are independent, if an exception occurs in one thread, it doesn't

affect other threads.

* In other words, exceptions thrown in one thread cannot be handled by another thread.

### Thread vs. Process



Why Multi-Threading?

* To enhance parallel processing
* To reduce response time to the user
* Many servers use multithreads to achieve high performance
* To utilize the idle time of the CPU
* Unit testing uses threads to run test cases in parallel
* Prioritize your work depending on priority
* Computer games is a good example of multi-threading process(loading maps when you are working on other things)

# Thread Class

### Thread Class

* In C#, a multi-threading system is built upon the Thread class, which encapsulates the execution of threads.
* This class contains several methods and properties which helps in managing and creating threads and this class is defined under System.Threading namespace.
* The first thread to be executed in a process is called the **main** thread.
* When a C# program starts execution, the main thread is automatically created. The threads created using the **Thread** class are called the child threads of the main thread.
* Programmers can always take control of the main thread.

### Thread Class

* The thread class provides lots of properties. Some of the important properties are as follows:
* ***CurrentThread***: used to get the current running thread.
* ***Name***: used to get or set the name of the thread.
* ***Priority***: used to get or set the priority value(***Enum***) of the thread.
* ***ThreadState***: used to get the thread state value(***Enum***) of the thread.
* ***IsAlive***: returns a ***bool*** value representing whether or not this thread is alive
* ***IsBackground***: used to get or set the value(***bool***) indicating whether the thread is a background thread or not.

### Thread Class

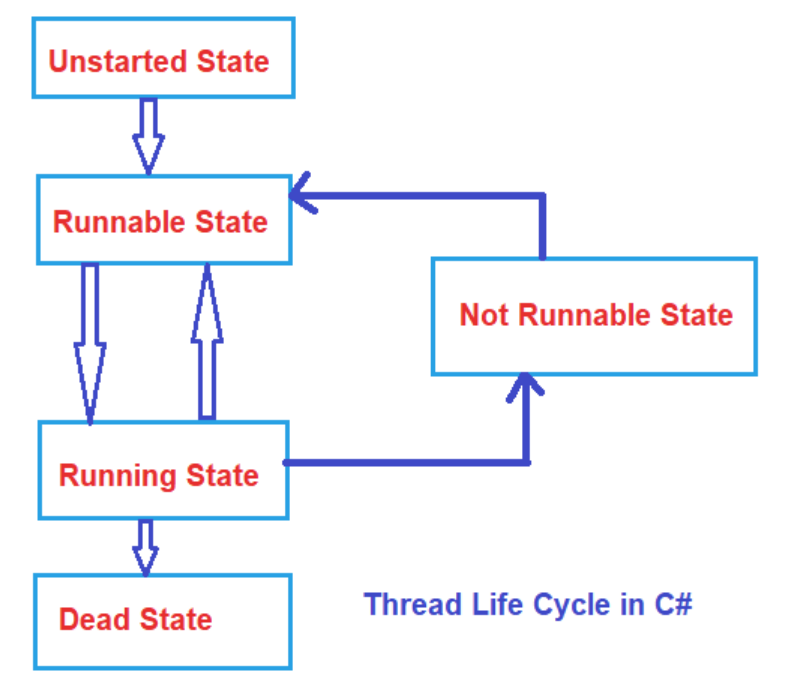
* Create a Thread

### Thread Life Cycle

The life cycle of a thread starts when an object of the Thread class is created and ends when the thread is terminated or completes execution.

* **The Unstarted State** − It is the situation when the instance of the thread is created but the Start method is not called.
* **The Runnable State** − It is the situation when the thread is ready to run(the Start method has been called).
* **The Running State** – The thread is running and not stopped yet.
* **The Not Runnable State** − A thread is not executable, when
  + Sleep method has been called
  + Join method has been called
  + Blocked by I/O operations
* **The Dead State** − It is the situation when the thread completes execution or is aborted by calling Abort() method.

### Thread Life Cycle



Not Runnable State

* Sleep() – Thread.Sleep() causes the current thread to suspend execution for a

specific milliseconds. Sleep() is a static method in Thread class.

* Join() – The Join() method blocks the calling thread until the thread represented by this instance terminates.
* Example: We create the Thread t in the main thread;
* Calling t.Join() will cause the main thread to wait for thread t.
* Suspend() – Suspend() method is called to suspend the thread.（deprecated）
* Resume() – Resume() method is called to resume the suspended thread.(deprecated)

# Types of Thread

### Types of Thread

* In C# we can create two types of threads in the application, they are:
* Foreground Thread
* Background Thread

### Foreground Thread vs. Background Thread

* Foreground threads are those threads that keep running even after the main application exits or quits. So, the foreground threads do not care whether the main thread is alive or not, it completes only when it finishes its assigned work. That means the life of a foreground thread does not depend upon the main thread. Foreground thread is the ***default type*** when a new thread is created.
* Background Threads are those threads that will quit if our main thread is

finished. The life of a background thread depends on the main thread.

* A thread can be changed to a background thread at any time by setting it’s ***IsBackground***

property to ***true***.

# Thread Safety

### Thread Safety

* Threads communicate primarily by sharing access to the same recourses such as fields or references to objects.
* This form of communication is extremely efficient, but it also makes some

problems possible: *thread interference* and *data inconsistency*.

### Thread Interference

* Consider a situation where two thread is operating on the same object at the same time.
* Interference happens when two operations, running in different threads, but acting on the same data, *interleave*. This means that the two operations consist of multiple steps, and the sequences of steps overlap.

### Thread Interference

The App will decompose the Increment

method into following steps:

* Retrieve the current Value.
* Increment the retrieved value by 1.
* Store the incremented value back in the Property.

What if thread A is calling increment and thread B is calling decrement? (What will be the result?)

This situation is also called the Race Condition

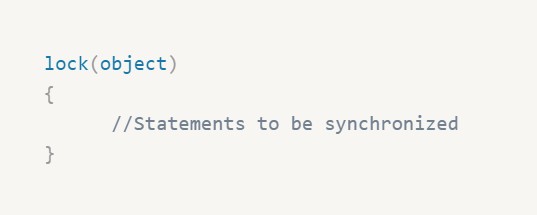
### Data Inconsistency

* Data Inconsistency when different threads have inconsistent views of what should be the same data
* Eg. SameCounter class as before—Thread A increases the counter by 1, and thread B tries to print the value of counter at the same time. Now the value can either be 1 or 0

### Thread Synchronization

* Synchronization can be achieved by using the ***lock*** keyword.
* It is used lock the object and only allow one thread to access the locked object, execute the task and then the lock will be released.
* It ensures that other thread does not interrupt the execution until the

execution finish.

* The lock can only apply on objects.
* Syntax:

### Thread-Safe Collections

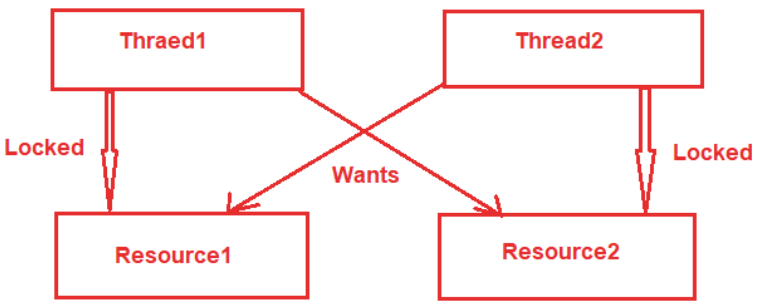
* The following table list some thread-safe collections under System.Collections.Concurrent namespace:



# Deadlock

### Deadlock

* *Deadlock* describes a situation where two or more threads are blocked forever, waiting for each other to release the lock.



### Recap

* LINQ
  + What is LINQ?
  + LINQ to Objects
  + Standard Query Operators
  + LINQ to SQL
* Thread
  + Thread & Process
  + Thread Class
  + Thread Life Cycle
  + Thread Problem / Thread Safety
  + Thread Synchronization
  + Dead Lock

# Any Questions?