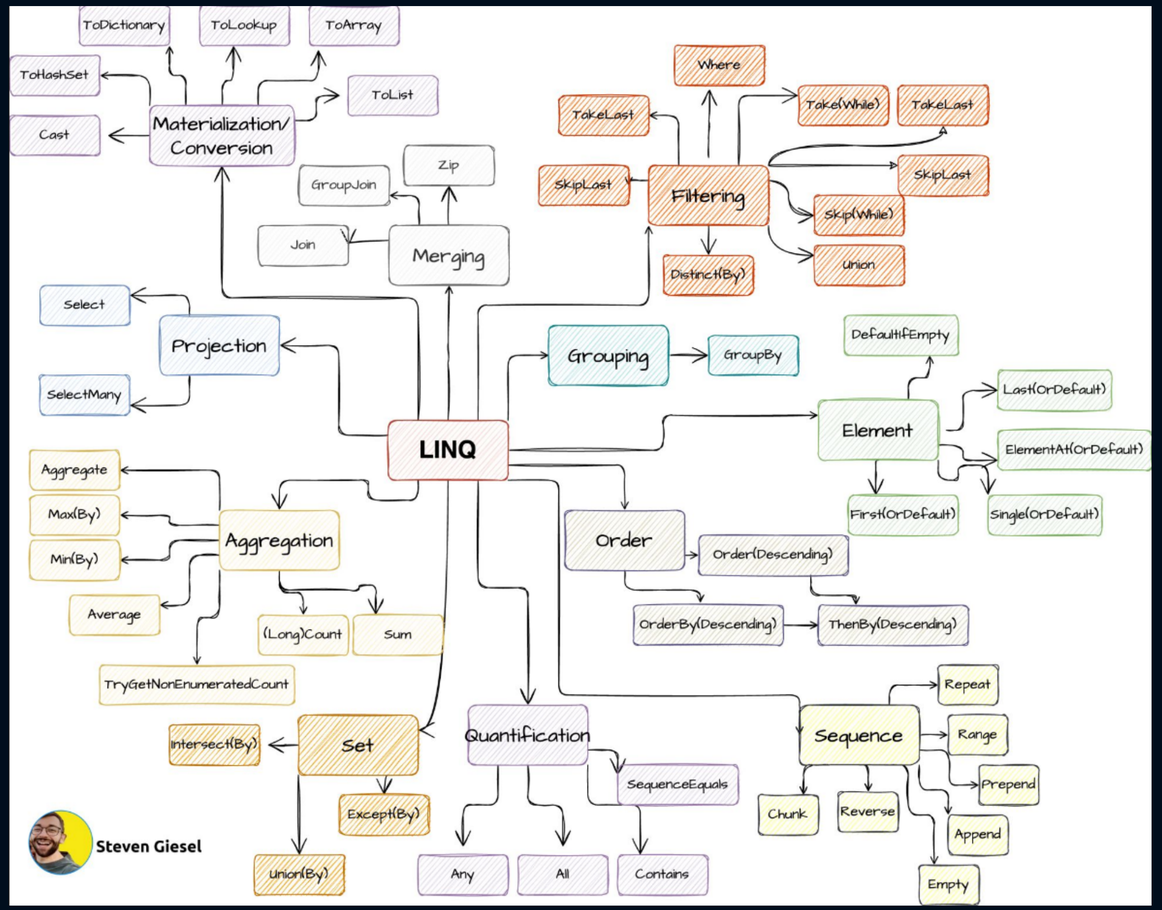
LINQ explained

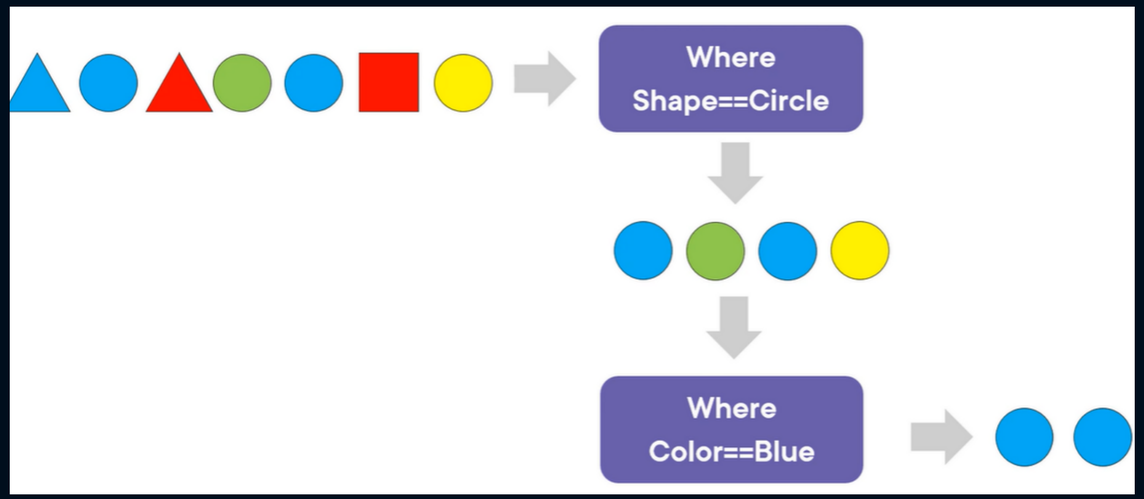
Lets dive in to more advanced concepts with LINQ.



LINQ Explained with sketches

Filtering

Filtering refers to the process of selecting specific elements from a collection based on certain conditions. It allows you to retrieve only the data that meets specific criteria, making it easier to work with the desired subset of items.



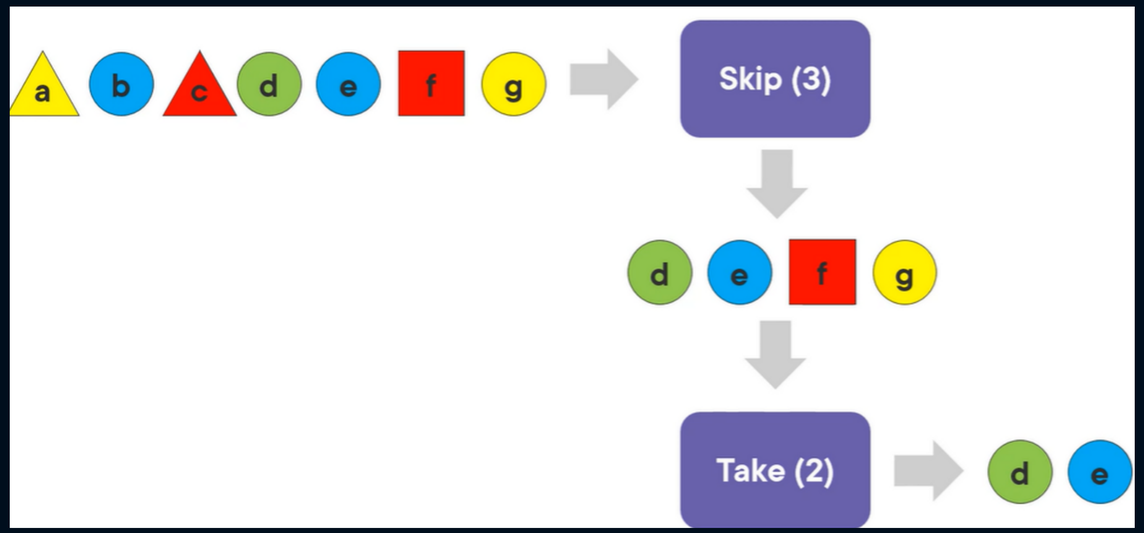
Filtering representation

```
private static List<Car> ProcessFile(string path)
{
    return
        File.ReadAllLines(path)
            .Skip(1)
            .Where(line => line.Length > 1)
            .Select(Car.ParseFromCsv)
            .ToList();
}
```

Filtering example

Partitioning

Partitioning refers to the process of dividing a collection into smaller parts or segments. It allows you to retrieve a specific portion of the data based on the desired size or index range.



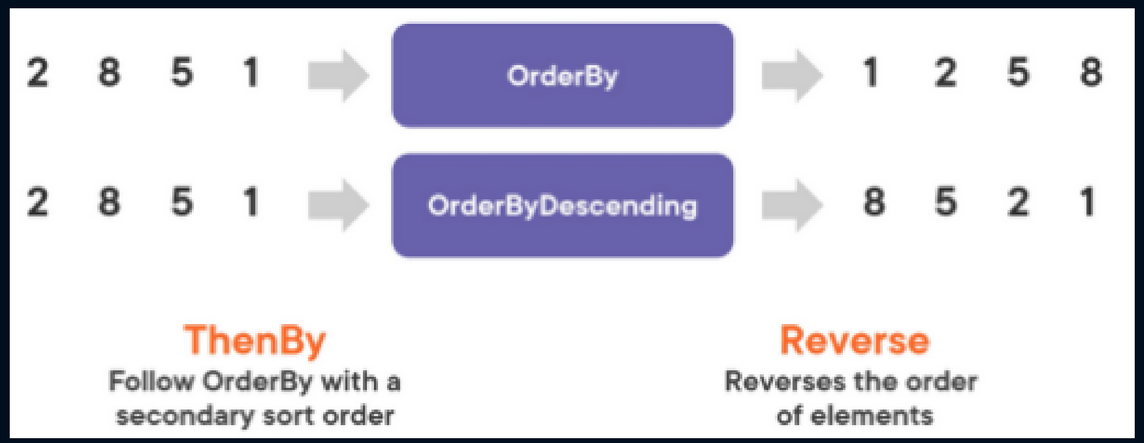
Partitioning

```
public List<Product> TakeRangeMethod() {
  List<Product> products = GetProducts();
  List<Product> list = new();

  // Write Query Syntax Here
  list = products.OrderBy(p => p.Name).Take(5..8).ToList();

  return list;
}
```

Using Take with range operator

Ordering

Ordering refers to the process of arranging the elements of a collection in a specific order. It allows you to sort the data based on one or more criteria.



Ordering

```
public List<Product> OrderByTwoFieldsQuery() {
  List<Product> products = GetProducts();
  List<Product> list = new();

  // Write Query Syntax Here
  list = (from prod in products
          orderby prod.Color descending, prod.Name ascending
          select prod).ToList();

  return list;
}
public List<Product> OrderByTwoFieldsMethod() {
  List<Product> products = GetProducts();
  List<Product> list = new();

  // Write Method Syntax Here
  list = products.OrderByDescending(prod => prod.Color)
                 .ThenBy(prod => prod.Name).ToList();

  return list;
}
```

Example

Quantification

Quantification refers to the process of determining whether all or any elements in a collection satisfy a specific condition.

```
IEnumerable<T>.All(predicate);


products.All(prod =>
    prod.ListPrice > prod.StandardCost);
```

- All() searches the entire collection
- Determines if *all* items match the condition

- Do all products' list price exceed their cost?

All

```
IEnumerable<T>.Any(predicate);


sales.Any(sale =>
    sale.LineTotal > 10000);
```

- Any() method searches entire collection
- Determines if *any* items in collection match the condition

- Do any sales have a line total greater than 10,000?

Any

```
public bool ContainsQuery()
{
  List<int> numbers = new() { 1, 2, 3, 4, 5 };
  bool value = false;

  // Write Query Syntax Here
  value = (from num in numbers select num).Contains(3);

  return value;
}
```

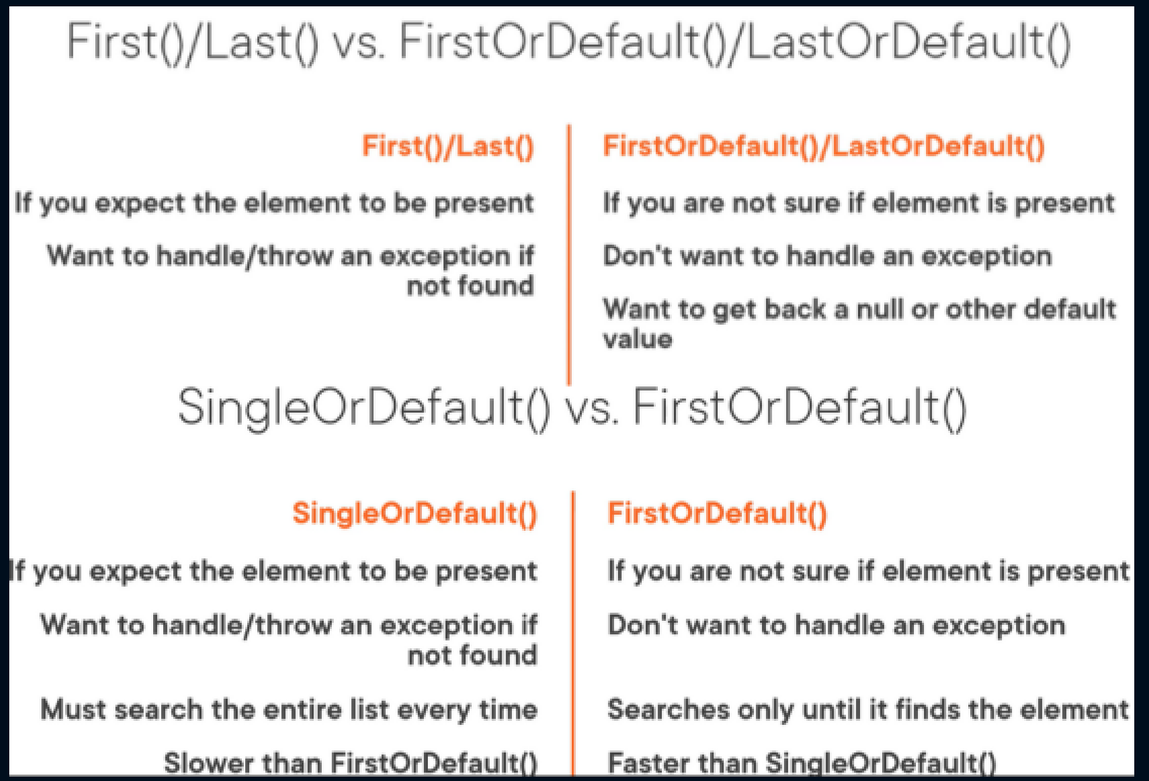
Contains

Get element

There are several methods available to retrieve specific elements from a collection. These methods allow you to get the first, last, or single element from a sequence based on certain conditions.

```
    products.Clear();
    value = (from prod in products
             select prod)
             .SingleOrDefault(prod => prod.ProductID == 706);


    // Test the exception handling for the list is empty and a default value is supplied

value = (from prod in products
         select prod)
         .SingleOrDefault(prod => prod.ProductID == 706,
         new Product { ProductID = -1, Name="NO PRODUCTS IN THE LIST"});
```

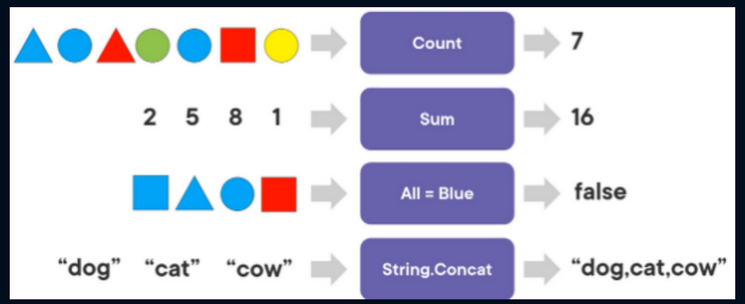
Using Single or Default



Comparison between get element methods

Aggregations

Aggregations refer to the process of computing a single value from a collection of elements. In LINQ we have several methods like Sum, Count, Average, Min, Max



How aggregates work?

```
// Write Query Syntax Here
value = (from prod in products
         select prod)
          .Aggregate(0M, (sum, prod) =>
                         sum += prod.ListPrice);
```

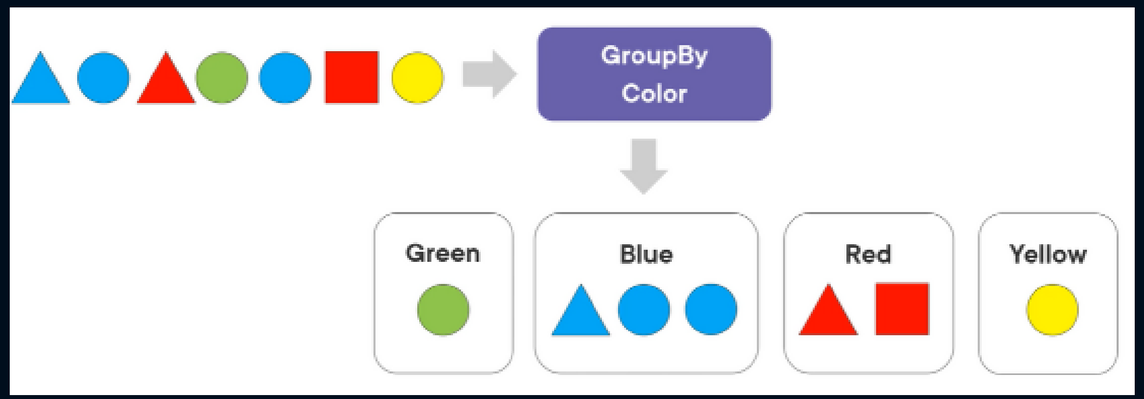
Use custom aggregates

```
select new ProductStats
{
  Size = sizeGroup.Key,
  TotalProducts = sizeGroup.Count(),
  MinListPrice =
    sizeGroup.Min(s => s.ListPrice),
  MaxListPrice =
    sizeGroup.Max(s => s.ListPrice),
  AverageListPrice =
   sizeGroup.Average(s => s.ListPrice)
}
```

Use multiple aggregates

Grouping

EventArgs class is used in the signature of many delegates and event handlers. When custom data needs to be passed the EventArgs class con be extended



Grouping explained

```
public List<IGrouping<string, Product>> GroupByWhereMethod()
{
  List<IGrouping<string, Product>> list;
  // Load all Product Data
  List<Product> products = ProductRepository.GetAll();

  // Write Method Syntax Here
  list = products.OrderBy(p => p.Size)
                 .GroupBy(prod => prod.Size)
                 .Where(sizes => sizes.Count() > 2)
                 .Select(sizes => sizes).ToList();

  return list;
}
```

Example

Joining

EventArgs class is used in the signature of many delegates and event handlers. When custom data needs to be passed the EventArgs class con be extended

```
var query =
    from car in cars
    join manufacturer in manufacturers
        on car.Manufacturer equals manufacturer.Name
    orderby car.Combined descending, car.Name ascending
    select new
    {
        manufacturer.Headquarters,
        car.Name,
        car.Combined
    };
```

Join using query syntax

```
var query2 =
    cars.Join(manufacturers,
                  c => c.Manufacturer,
                  m => m.Name, (c, m) => new
                  {
                      m.Headquarters,
                      c.Name,
                      c.Combined
                  })
        .OrderByDescending(c => c.Combined)
        .ThenBy(c => c.Name);
```

Join using method syntax

Demo

We will check the following:

Use LINQ

- Filtering
- Sorting
- Projecting
- Grouping
- Aggregations

You can check the following Repository(ies) for some examples:

Customer management system oop

You can check the following Repository for some examples:

C# fundamentals

Task

Quick Exercises

Using this repository(customer management) do the following:

Run the LINQ playground project, and select option 1 when running the first time(Generate data and save to file). For subsequent runs choose option 2.

Add the implementations for the solutions to the following exercises, and use the debugger data visualizer or print to console to see if the result is correct.

Exercise 1

Using the products generated data, retrieve the total current price of all products that start with an A, E and I

Exercise 3

Using the orders generated data, retrieve the highest 3 customers with the most orders and show the customer name, email, order id, order date and product id

Exercise 2

Using the order items generated data, retrieve all the orders that were made in this year and order them from latest to oldest. Split the records obtained in periods of months

Exercise 4

Using the order items generated data, retrieve the top 3 products with the most order items and show the product name, product description. Retrieve the List of customers related to the order items and the Address information.