### Variables

**Purpose**: Store data values for use throughout the program.

**Example**:

List<Car> cars holds all car objects.

bool debug controls extra output for troubleshooting.

string[] lines stores all lines read from the CSV file.

Local variables like **avgPrice**, **avgMileage**, and **cheapestCar** store calculated results.

### Loops

**Purpose**: Repeats actions until the loop conditions have been completed

**Example**:

The foreach (var line in lines) loop processes each line from the CSV file, converting it into a **Car** object.

### Conditional Statements

**Purpose**: Allows the program to perform an action based on the specified condition.

**Examples**:

if (**firstLine**) skips the header row in the CSV.

if (**debug**) controls whether debugging info is printed to the console.

### Collections

**Purpose**: Store and manage groups of related data.

**Examples**:

List<Car> is used to store all car objects for easy access and processing.

**LINQ** methods like **Select**, **Sum**, and **Aggregate** operate on these collections to calculate averages, totals, and find the cheapest car.

### Methods

**Purpose**: A method is a block of code that performs a specific task when it is called. It helps to organise code and make it reusable. Methods can accept inputs or return outputs as desired.

**Examples**:

Some methods I created are: **CalculateAverage**, **GetCheapestCar** and **GetStockSummary**. These methods are used to perform the core functionality of the program.

**GetCheapestCar** and **GetStockSummary** are ‘helper’ methods, for better human readability.   
The **CalculateAverage** method takes a list of Car objects and a lambda function as the ‘selector’ to grab the required properties. Next it uses the **Select** method to extract the chosen values from each car object, and finally the **Average** method adds all the values of the specified property and then divides by the number of properties in the list, to calculate the average.

### Objects

**Purpose**: An object is an instance of a class stored in the computer’s memory, allocated and configured according to the class blueprint. A variable can reference this object, allowing programs to access its data (via properties/fields) and behaviour (via methods) in order to model real-world entities and relationships.

**Examples**:

In the carManager9001 app, I create a Car class and then read all of data from the CSV file and instantiate it as a new **Car** object. Each car object has a *make*, *model*, *registration*, *mileage* and *price*.

### List

**Purpose:** A list is adynamic and strongly-typed collection that stores objects of a specified type (such as **Car**) and allows easy access, addition, removal, and manipulation of items using indexes.

**Examples**:

A list in C# is represented by the generic class List<T>, where T is the type of objects stored (such as **int**, **string**, or **Car**). Lists belong to the **System.Collections.Generic** namespace and unlike arrays, can be resized dynamically.

### Array

**Purpose**: An array is a fixed-size, strongly-typed collection used to store multiple elements of the same type, accessible via an index. Arrays are efficient for scenarios where the number of items will not change and have rapid access speeds.

**Examples**:

Arrays are not utilized in this application because lists are the more commonly used data structure in modern C# code and are generally regarded as providing greater consistency and readability, while also offering innate future-proofing and superior readability.

### Libraries/Namespaces

**Purpose**: *Namespaces* are logical groupings of related data types and methods that offer access to frequently used functionality (such as file operations via **System.IO**). *Libraries* are pre-compiled collections of code that contain implementations and are typically pre-compiled into a .dll format. I do not use any libraries in my code but I use the **System**, **System.IO**, **System.Collections.Generic** and **System.LINQ** *namespaces*.

**Examples**:

In this project I used the **System** namespace to give access to the console for read/write operations, the **Collections.Generic** namespace for access to the List<T> collection, the **System.IO** for further file operation support (including IO Exceptions) and the LINQ namespace for **Select**, **Aggregate** and **Average** methods.

### Path to CSV file

**Purpose**: Relative or absolute path to the CSV file to allow the C# program to find user data

**Examples**:

In this project, I use the string variable “relativePath” as a property for the relative path to the csv file.

// Path to the input CSV file

string relativePath = @"resources\\SD-TA-001-B\_DealershipStockList.csv";

**DealershipStockList.csv**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Registration | Make | Model | Mileage | Price |
| 231-D-3214 | Toyota | Corrolla | 8000 | 20000 |
| 161-MH-75251 | Opel | Astra | 108000 | 12000 |
| 09-G-99 | Nissan | Qashqai | 150000 | 7500 |
| 182-KE-4321 | Mercedes-Benz | E-Class | 12000 | 40000 |

### Constructor

**Purpose**: A constructor is a special method of a class in C# whose primary purpose is to initalise the values of a newly created object of the class. The constructor is automatically called when an instance of the class is first created, allowing us to set the initial values.

**Examples**:

No constructor was used in this C# program as the simple data class only stores data with public fields and no setup or initialisation is required.