C# Cheatsheet

Basic Structure and Execution

A `C#` program consists of namespaces, classes, and a `Main` method.

```
using System; // Import namespace

namespace HelloWorldApp {
    class Program {
        // Entry point
        static void Main(string[] args)
{
            Console.WriteLine("Hello,
World!");
        }
    }
}
```

- Execute with .NET CLI:

```
dotnet new console -o MyCsApp
cd MyCsApp
dotnet run
```

Variables, Types, and Operators

- Value Types: `int`, `double`, `char`, `bool`, `decimal`, `struct`, `enum`.
- Reference Types: `string`, `object`, `class`, `interface`, `delegate`, `array`.
- Declaration: `type variableName = value;` (e.g., `int age = 30;`)
- `var`: Compiler implicitly deduces the type. `var message = "Hello";`
- Constants: `const double PI = 3.14;`
- Nullable Types: `int? nullableInt = null;`

Control Flow

- `if-else`:

```
if (condition) { /* ... */ }
else { /* ... */ }
```

- `switch`:

```
switch (variable) {
    case 1:
        // ...
        break;
    case 2:
        // ...
        break;
    default:
        // ...
        break;
}
```

- `for` Loop: `for (int i = 0; i < 5; i++) { ... }`
- `foreach` Loop:

```
var numbers = new List<int> { 1, 2,
3 };
foreach (var number in numbers) {
    Console.WriteLine(number);
}
```

- `while` / `do-while` loops.

Classes and Objects

- Class Definition:

```
public class Car {
    // Properties
    public string Color { get; set; }
    public int Speed { get; private
set; }

    // Constructor
    public Car(string color) {
        Color = color;
        Speed = 0;
    }

    // Methods
    public void Accelerate(int amount)
{
        Speed += amount;
    }
}
```

- Object Creation: `Car myCar = new Car("blue");`
- Member Access: `myCar.Accelerate(10);`

Inheritance and Interfaces

- Inheritance: `public class ElectricCar : Car { ... }`
- Interface:

```
public interface IDrivable {
    void Drive();
}

public class MyCar : IDrivable {
    public void Drive() { /* ... */ }
}
```

Collections

Included in the `System.Collections.Generic` namespace.

List<T>

```
var names = new List<string>();
names.Add("Alice");
names.Add("Bob");
string first = names[0];
```

Dictionary<TKey, TValue>

```
var ages = new Dictionary<string,
int>();
ages["Alice"] = 30;
ages["Bob"] = 25;
int aliceAge = ages["Alice"];
```

LINQ (Language-Integrated Query)

Provides querying capabilities for data sources.

```
var scores = new List<int> { 97, 92, 81,
60 };

// Query Syntax
var query = from score in scores
            where score > 80
            select score;

// Method Syntax
var highScores = scores.Where(s => s >
80).ToList();
```

Exception Handling

- `try-catch-finally`:

```
try {
    // Potentially failing code
}
catch (FormatException e) {
    // Handle specific exception
}
catch (Exception e) {
    // Handle general exception
}
finally {
    // Always executed
}
```

Asynchronous Programming (async/await)

Perform long-running tasks without blocking the UI thread.

```
public async Task<string> GetDataAsync()
{
    using (var client = new
HttpClient()) {
        // Asynchronously download
webpage
        string result = await
client.GetStringAsync("https://www.
microsoft.com");
        return result;
    }
}

// Calling the method
string data = await GetDataAsync();
```

Google Style Guide

Naming Conventions

- Classes/Methods/Properties: `PascalCase`
- Interfaces: `IPascalCase` (I prefix)
- Local Variables/Parameters: `camelCase`
- Private Fields: `_camelCase` (underscore prefix)
- Files/Directories: `PascalCase.cs`
- Abbreviations: `MyRpc` (treated as a word, only the first letter capitalized)

Formatting

- Indentation: 2 spaces (no tabs)
- Line Length: Max 100 characters
- Braces: Starting brace `{` on the same line (Java style)
- Empty Blocks: Can be expressed concisely as `{}`
- Modifiers: Follow the order `public protected ... async`

Programming Practices

- Constants: Prefer `const` where possible, otherwise consider `readonly`.
- Collections: Recommend using the most restrictive types (`IEnumerable`, `IReadOnlyList`) for inputs.
- Struct vs Class: Default to `class`. Use `struct` only for very small, short-lived data.
- Delegates: Call safely as `SomeDelegate?.Invoke()`.
- Extension Methods: Use limitedly only when the original source cannot be modified.

C# Feature Usage

- var: Use only when the type is clear (`var apple = new Apple();`).
- LINQ: Prefer single-line writing. Consider replacing complex chains with imperative code.
- Attributes: Write on the line immediately above the field/method. Write multiple attributes separately.
- Lambda: Separate complex logic into named methods.
- Object Initializer: Recommended only for 'Plain old data' types.