C# 치트시트

기본 구조 및 실행

C# 프로그램은 네임스페이스, 클래스, 그리고 `Main` 메서드로 구성됩니다.

```
using System; // 네임스페이스 가져오기

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

- .NET CLI로 실행:

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

변수, 타입, 연산자

- 값 타입 (Value Types): `int`, `double`, `char`, `bool`, `decimal`, `struct`, `enum`.
- 참조 타입 (Reference Types): `string`, `object`, `class`, `interface`, `delegate`, `array`.
- 변수 선언: `type variableName = value;` (예: `int age = 30;`)
- `var`: 컴파일러가 타입을 자동으로 추론. `var message = "Hello";`
- 상수: `const double PI = 3.14;`
- Nullable 타입: `int? nullableInt = null;`

제어 흐름

- `if-else`:

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

- `switch`:

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

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

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

- `while` / `do-while` 루프.

클래스와 객체

- 클래스 정의:

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

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

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

- 객체 생성: `Car myCar = new Car("blue");`
- 멤버 접근: `myCar.Accelerate(10);`

상속과 인터페이스

- 상속: `public class ElectricCar : Car { ... }`
- 인터페이스:

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

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

컬렉션

`System.Collections.Generic` 네임스페이스에 포함.

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)

데이터 소스에 대한 쿼리 기능을 제공.

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

// 쿼리 구문
var query = from score in scores
            where score > 80
            select score;

// 메서드 구문
var highScores = scores.Where(s => s >
80).ToList();
```

예외 처리

- `try-catch-finally`:

```
try {
    // 예외 발생 가능 코드
}
catch (FormatException e) {
    // 특정 예외 처리
}
catch (Exception e) {
    // 일반 예외 처리
}
finally {
    // 항상 실행
}
```

비동기 프로그래밍 (async/await)

UI 스레드를 차단하지 않고 오래 걸리는 작업을 수행.

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

// 호출
string data = await GetDataAsync();
```

Google Style Guide

명명 규칙 (Naming)

- 클래스/메서드/속성: `PascalCase`
- 인터페이스: `IPascalCase` (I 접두사)
- 로컬 변수/매개변수: `camelCase`
- 비공개 필드: `_camelCase` (언더스코어 접두사)
- 파일/디렉토리: `PascalCase.cs`
- 약어: `MyRpc` (단어처럼 취급하여 첫 글자만 대문자)

포매팅 (Formatting)

- 들여쓰기: 공백 2개 (탭 금지)
- 줄 길이: 최대 100자
- 중괄호: 줄 바꿈 없이 시작 brace `{` 사용 (Java 스타일)
- 빈 블록: `{}` 처럼 간결하게 표현 가능
- Modifiers: `public protected ... async` 순서 준수

프로그래밍 관례

- Constants: 가능하면 항상 `const` 사용, 불가능하면 `readonly` 고려.
- Collections: 입력은 가장 제한적인 타입 (`IEnumerable`, `IReadOnlyList`) 사용 권장.
- Struct vs Class: 기본적으로 `class` 사용. 매우 작고 수명이 짧은 데이터에만 `struct` 고려.
- Delegates: `SomeDelegate?.Invoke()` 처럼 안전하게 호출.
- Extension Methods: 원본 소스를 수정할 수 없을 때만 제한적으로 사용.

C# 기능 활용

- var: 타입이 명확할 때만 사용 (`var apple = new Apple();`).
- LINQ: 가급적 한 줄로 작성. 복잡한 체인은 명령형 코드로 대체 고려.
- Attributes: 필드/메서드 바로 위 줄에 작성. 여러 속성은 각 줄에 별도로 작성.
- Lambda: 복잡한 로직은 명명된 메서드로 분리.
- Object Initializer: 'Plain old data' 타입에만 사용 권장.