OOP Concepts

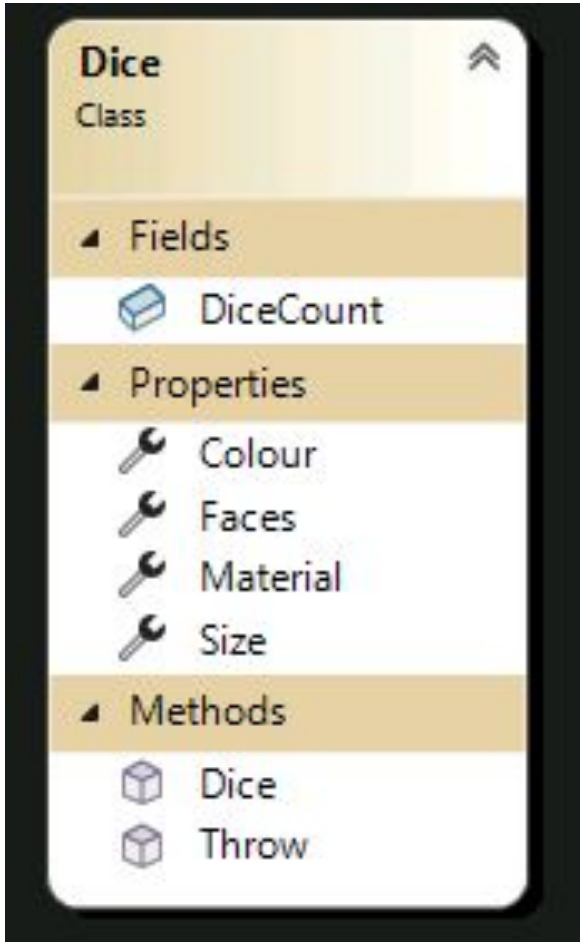
By Sergi Albalat Duran

Index

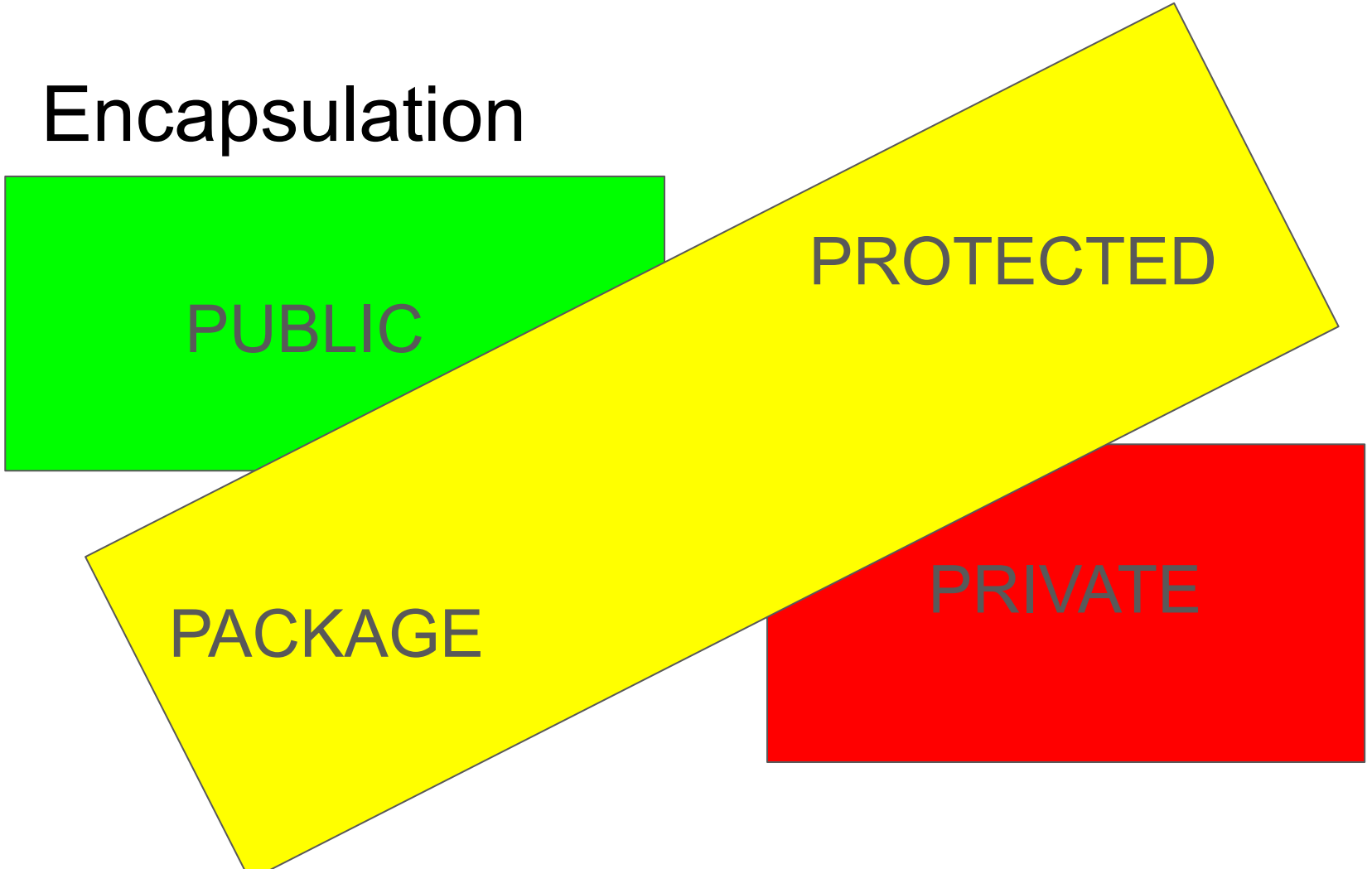
The Basics of OOP

Object ← Class

Instance → Constructor

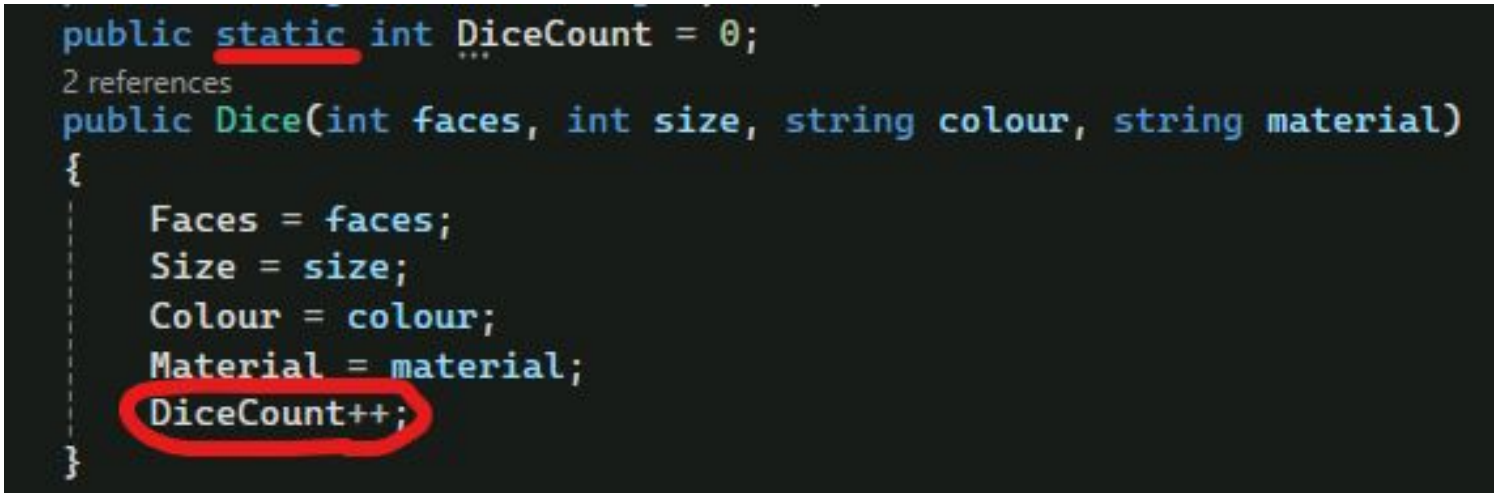


Encapsulation



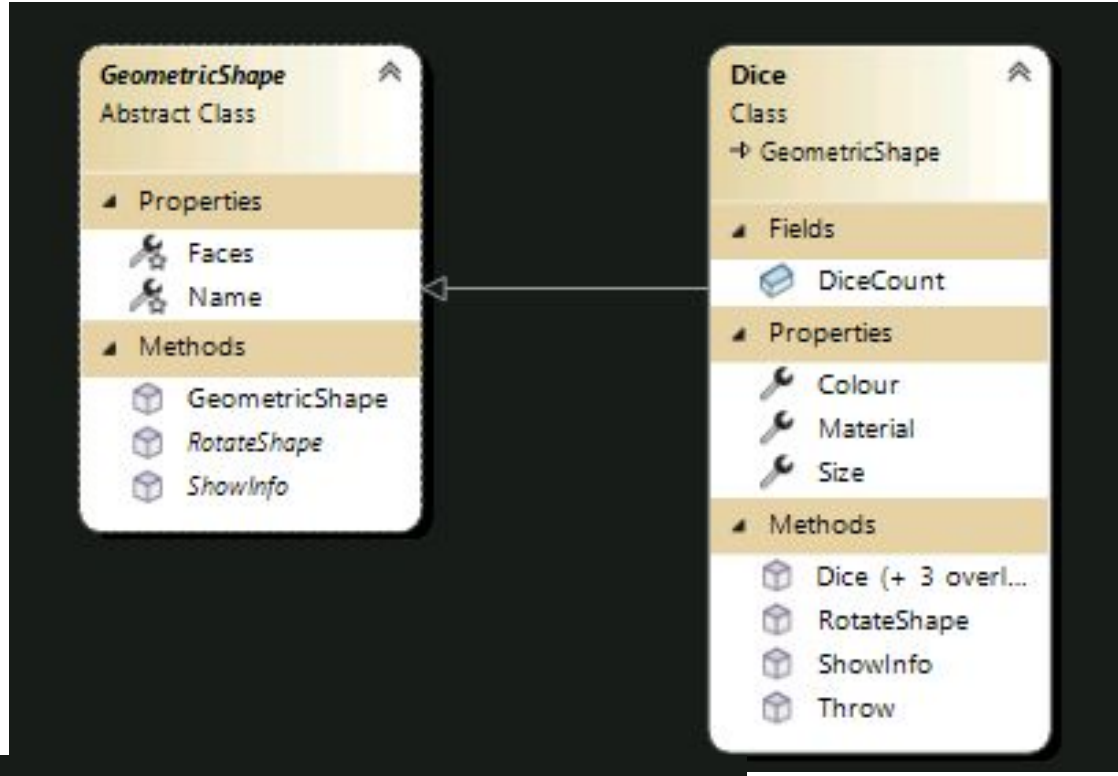
Encapsulation

STATIC ABSTRACT FINAL



```
public static int DiceCount = 0;
2 references
public Dice(int faces, int size, string colour, string material)
{
    Faces = faces;
    Size = size;
    Colour = colour;
    Material = material;
    DiceCount++;
}
```

Inheritance




```
11 references
public class Dice : GeometricShape
{
```

```
5 references
public Dice(string name, int faces, int size, string colour, string material) : base(name, faces)
{
    Size = size;
    Colour = colour;
    Material = material;
    DiceCount++;
}
```

Polymorphism

Compilation Time:

```
5 references
public Dice(string name, int faces, int size, string colour, string material) : base(name, faces)
{
    Size = size;
    Colour = colour;
    Material = material;
    DiceCount++;
}
0 references
public Dice() : this("Dice", 6, 3, "white", "plastic") { }
0 references
public Dice(int faces, string colour, string material) : this("Dice", faces, 3, colour, material) { }
0 references
public Dice(int faces) : this("Dice", faces, 3, "white", "plastic") { }
```

Polymorphism

Execution Time:

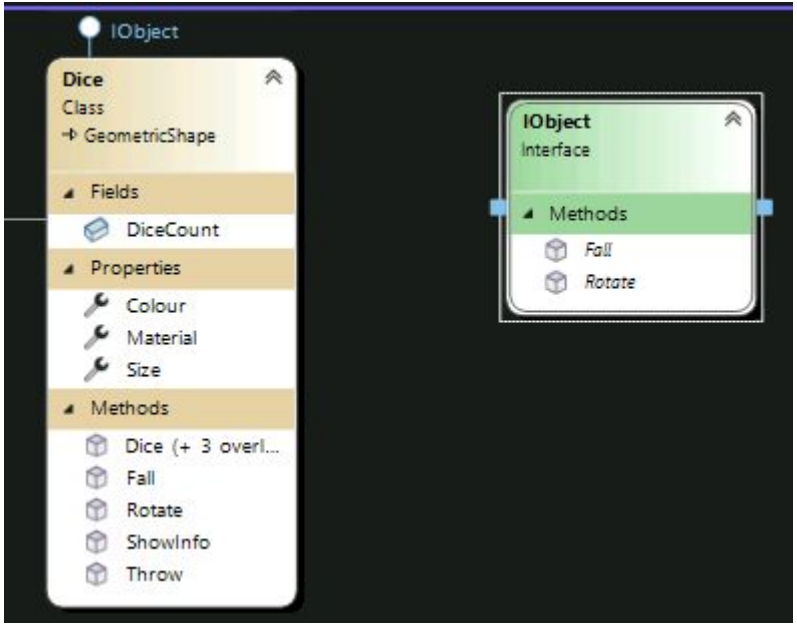
```
1 reference
public abstract void RotateShape();
3 references
public abstract void ShowInfo();
```

```
1 reference
public override void RotateShape()
{
    Console.WriteLine("Rotating");
}
3 references
public override void ShowInfo()
{
    Console.WriteLine("{0}, {1}, {2}, {3}, {4}", Name, Faces, Size, Colour, Material);
}
```

Helper Classes

```
11 references
public class Dice : GeometricShape, IObject
{
```

```
11 references
public class Dice : GeometricShape, IObject
{
```




```
using System;
namespace OopConcepts
{
    1 reference
    public interface IObject
    {
        1 reference
        void Rotate();
        1 reference
        void Fall();
    }
}
```

In the Class:

```
1 reference
public void Rotate()
{
    Console.WriteLine("Rotating Dice");
}
1 reference
public void Fall()
{
    Console.WriteLine("Dice Falling");
}
```

Relationship types

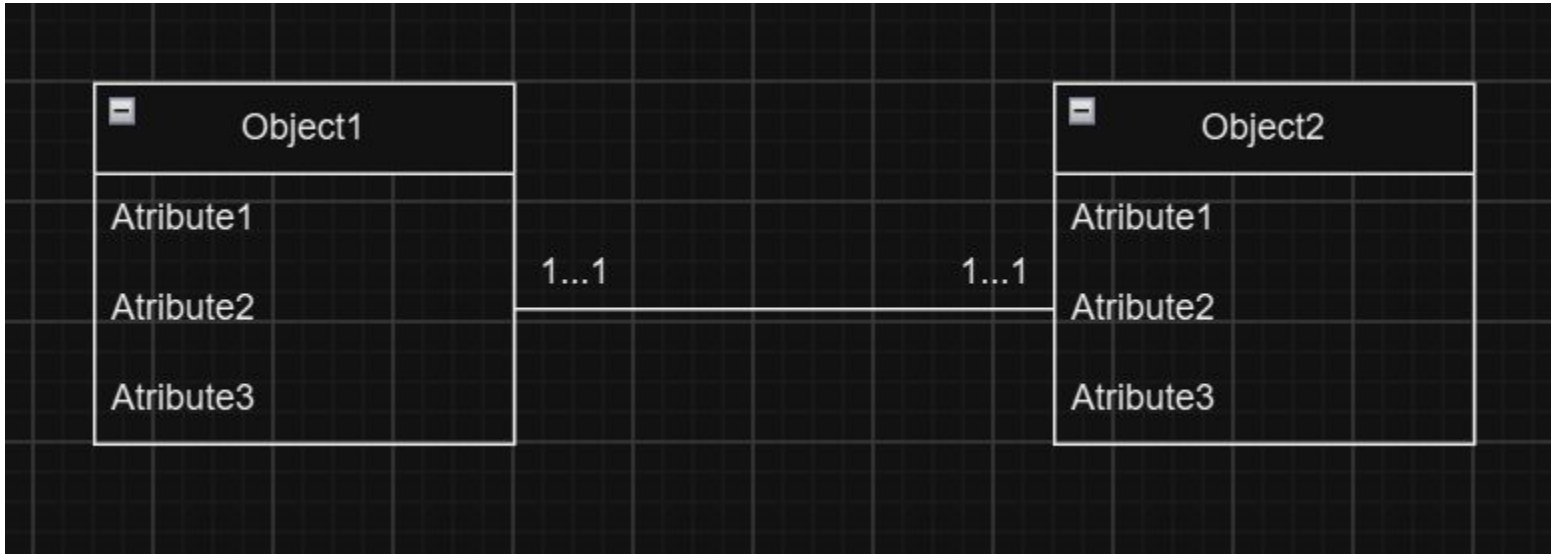
Association

Aggregation

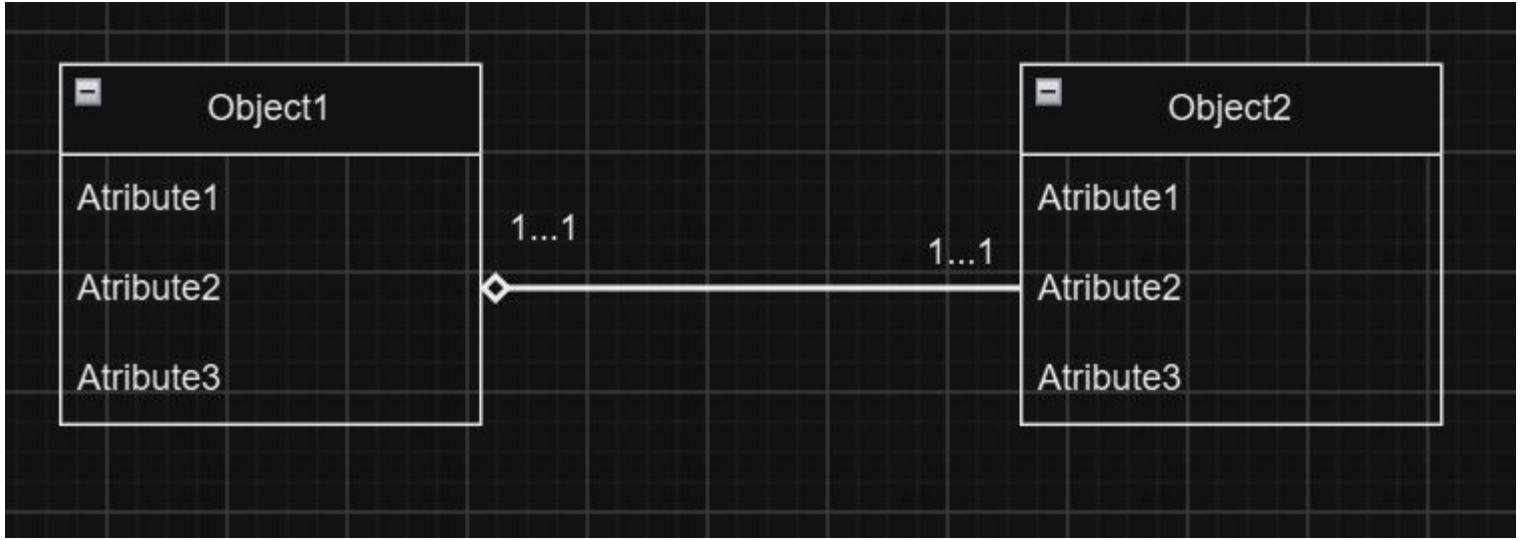
Composition

Inheritance

Association

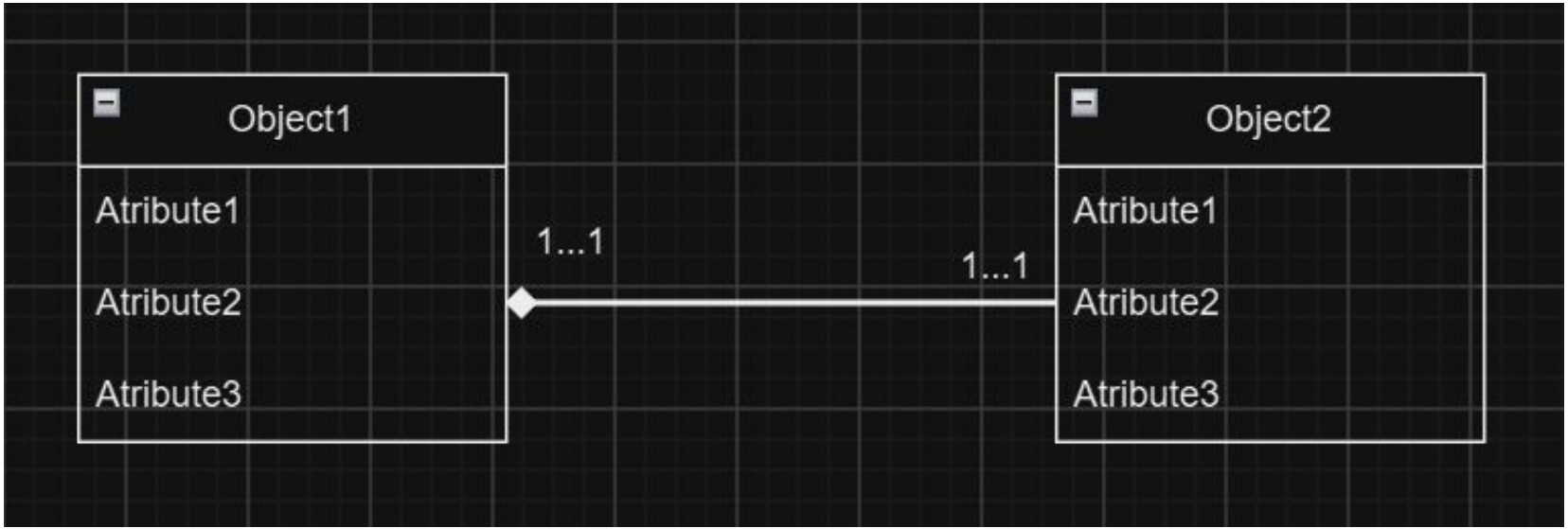


Aggregation



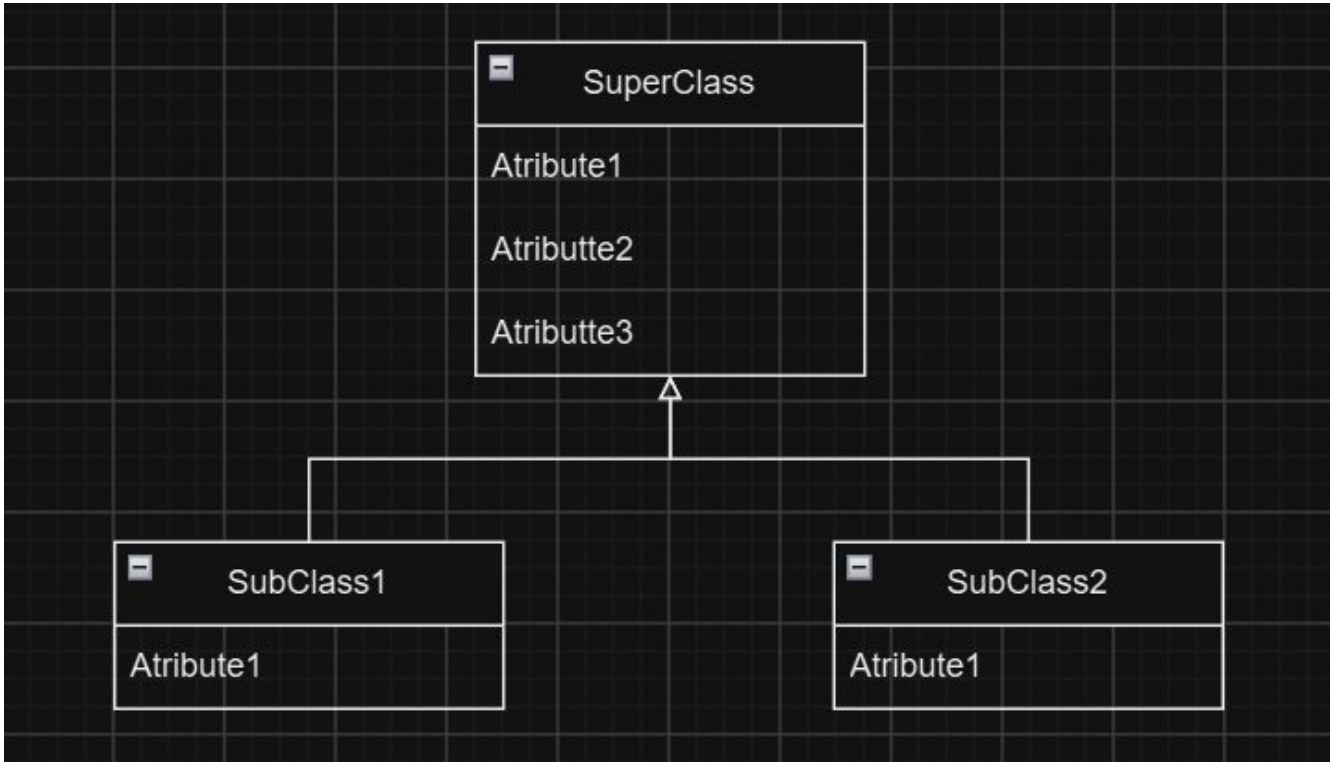
Components alone still exists

Composition



Components alone cannot exist

Inheritance



END