Object-oriented programming

Interfaces, inheritance and polymorphism

ST3ITS3

AARHUS
UNIVERSITY
SCHOOL OF ENGINEERING

Today's lecture

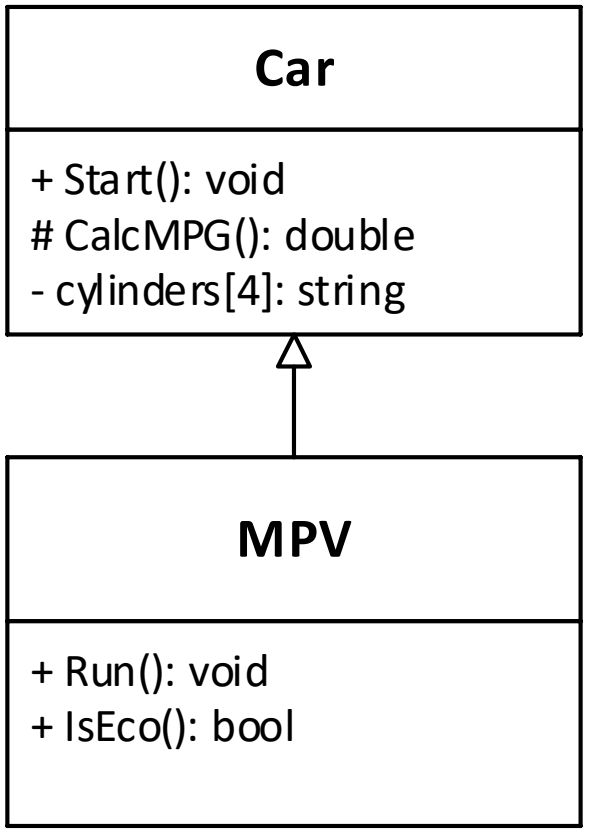
- Interfaces + the pillars object orientation

Inheritance

ST3ITS3

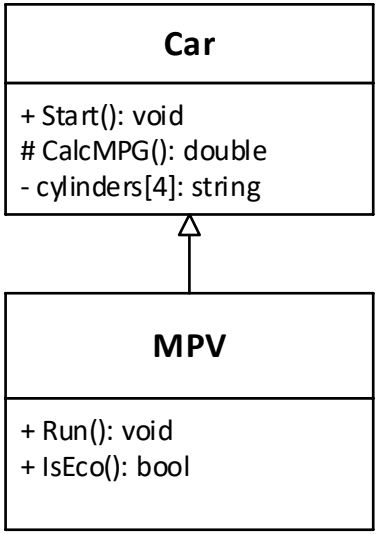
Inheritance

- Inheritance is often called an ”is-a”-relation. What does this mean?
- Inheritance of *implementation*: Subclasses can re-use methods etc. from superclass (under what circumstances?)
- Inheritance of *identity*: Subclasses may substitute s



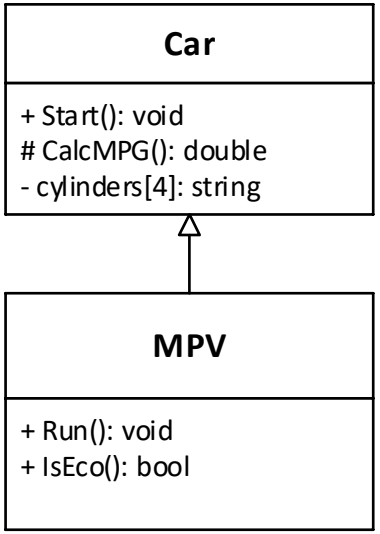
Inheritance of implementation

```
class MPV : Car
{
  void Run()
  {
    Start();  // OK – Car.Start() is public
    ...
  }

  bool IsEco()
  {
    if(CalcMPG() >= 20) // OK – CalcMPG() is protected
      return true;
    return false;
  }

  bool HasPart(string part)
  {
    foreach(var cyl in cylinders)  // ERROR – Car.cylinders[] is private
    {
      ...
    }
  }
}
```



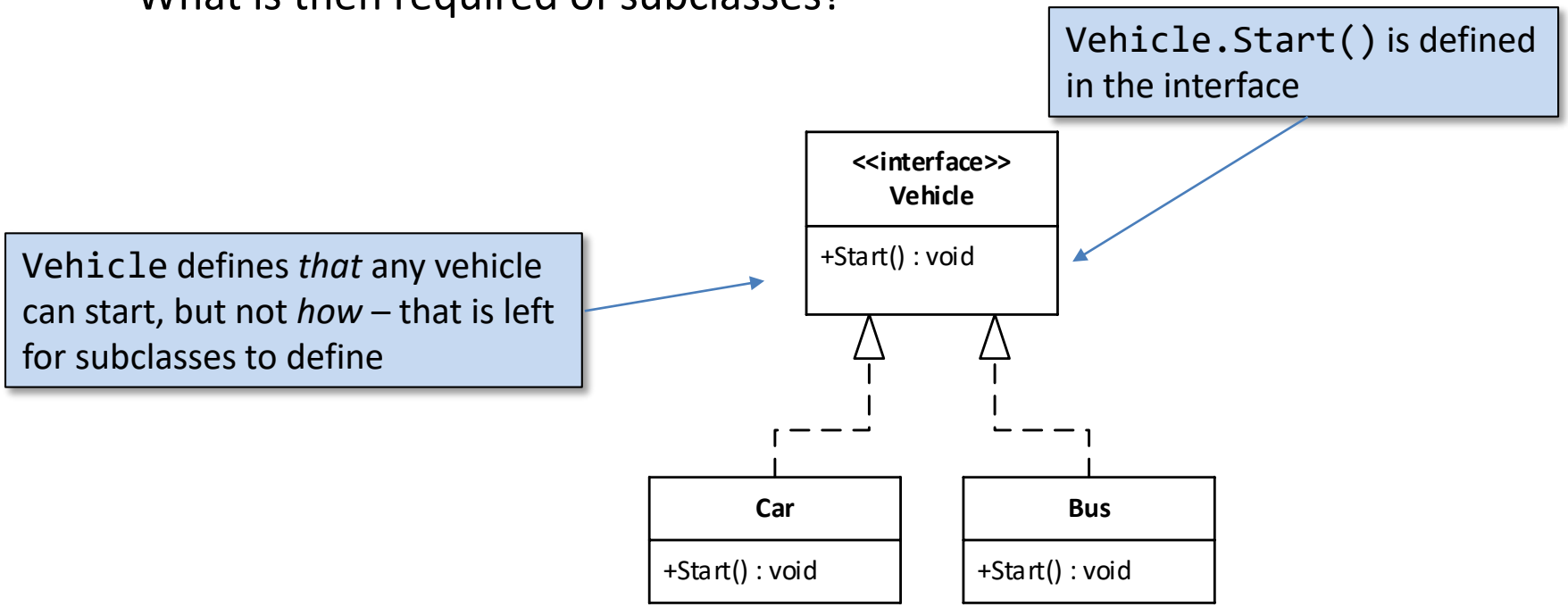
Inheritance of identity

```
void Main()
{
  Car _mpv = new MPV();  // OK – MPV is-a Car

  _mpv.Start();   // OK – Start() is member of Car, and _mpv is of type Car
  _mpv.Run();    // ERROR – Run() is member of MPV, but _mpv is of type Car

   MPV _mpv2 = new MPV();
  _mpv2.Start();  // OK – Start() is member of Car, and _mpv2 is-a Car
  _mpv2.Run();    // OK – Run() is member of MPV

  MPV _mpv3 = new Car();  // ERROR: MPV is-a Car, not vice versa
}
```



Inheritance of identity

- Inheritance is often used to define a common *identity*, not *implementation*. To do this, we can introduce an superclass or interface
  - What is then required of subclasses?



Interfaces

ST3ITS3

Interfaces are all around us!

Some properties of interfaces

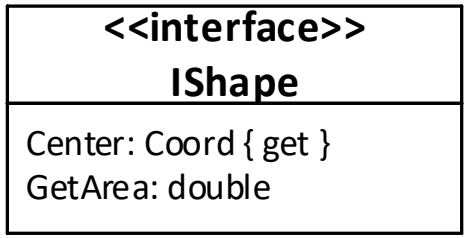
- An interface is a contract: If a class X implements the interface `IX`, then X
  - is required to implement all methods, properties, etc. in `IX`
  - is entitled to be considered of type `IX` by clients
- An interface cannot be instantiated. Interfaces do not contain implementation of methods.
- Classes can implement multiple interfaces.
- Interfaces can contain events, indexers, methods, and properties.

Interfaces in C#: Definition

- Interface definition in C#:

```
// IShape.cs
interface IShape
{
            Coord Center {get; }
            double GetArea();
}
```

- Interface definition in UML:



Interfaces in C#: Implementation

- Interface implementation in C#

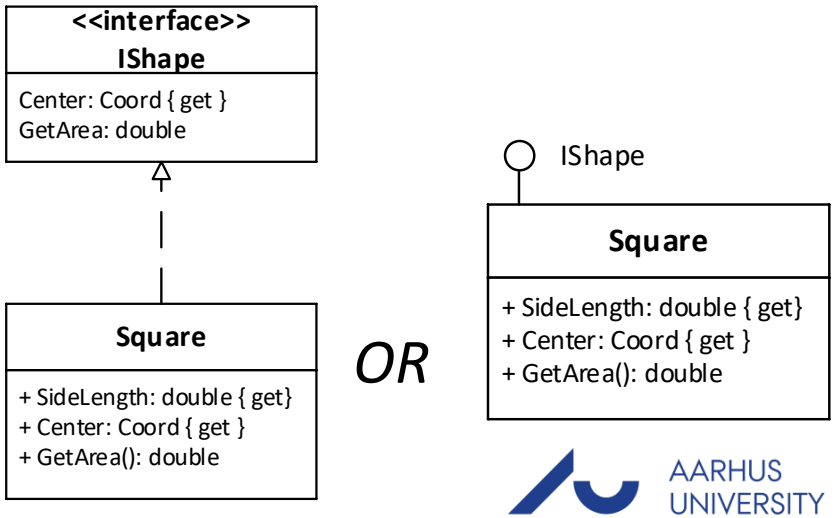
Square must implement the methods and properties defined in IShape

Square declares that it implements IShape

Square may of course implement methods etc. that are not defined in the interface

```
// Square.cs
class Square : IShape
{
        public Coord Center { get; private set;}
        public double GetArea(){…}

        public double SideLength { get; private set;}
}
```

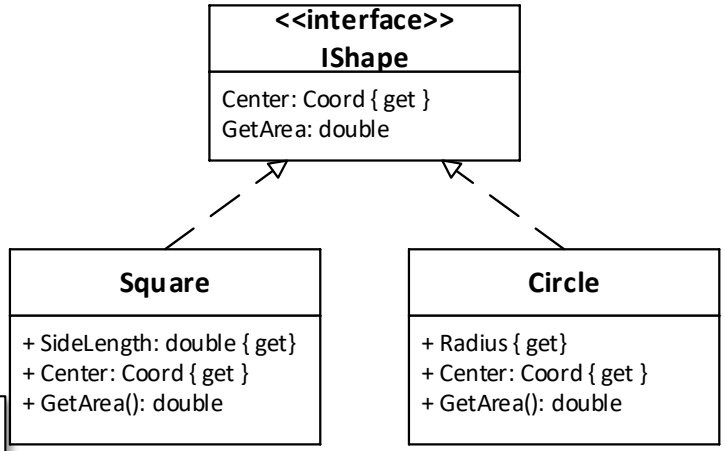
- Interface implementation in UML



Interfaces in C#: Implementation

- Any number of classes may implement the same interface

```
// IShape.cs
interface IShape
{
        Coord Center {get; };
        double GetArea();
}
```




```
// Square.cs
class Square : IShape
{
        private Coord _center;
}
```

```
// Circle.cs
class Circle : IShape
{
        private Coord _center;
        public double Radius { get; private set;}
        public Coord Center { get; private set;}
        public double GetArea(){…}
}
```

Interfaces in C# - Reference and use

Screen knows only of IShape – not Circle's or Square's

Any implementation of IShape may be added to _shapes

Screen can only call methods defined in IShape

```
class Screen
{
        private list<IShape> _shapes;
        public void AddShape(IShape s) { _shapes.Add(s);}

        public double GetTotalArea()
        {
                double res = 0;
                foreach (IShape s in _shapes)
                {
                  res += s.GetArea();
                }
                return res;
        }
}
```

Polymorphism

ST3ITS3

Polymorphism

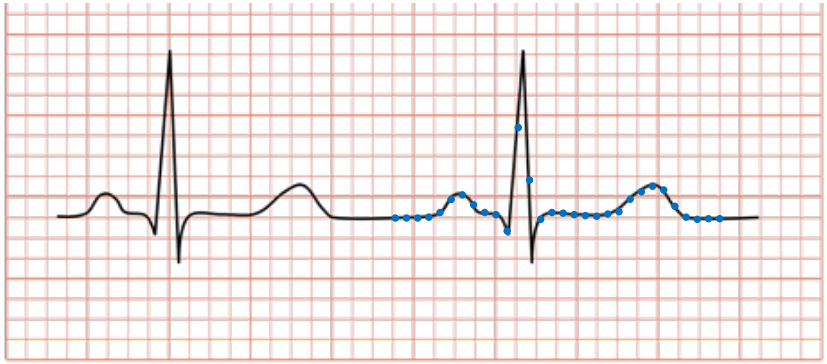
- “Polymorph” = “many forms”
- Polymorphism is used when we need different kinds of behavior from a class
- We create a superclass that defines the behavior and subtypes that implement it in different ways.
- Thus, the *behavior* of a given object varies depending on its *type.*

Polymorphism

- An interface (superclass) defines the common traits of a subclass through an interface (set of abstract methods)
- Implementations of the interfaces (subclasses of the superclass) implement the methods
  - Thus, the behavior – implemented in the methods – differ from subclass to subclass
- Clients see different behavior, but not the different subtypes.
- Polymorphism is best understood from the side of the *client*

Polymorphism

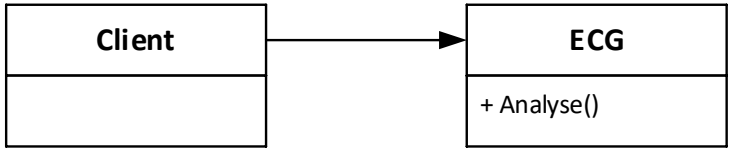
- Assume we have an ECG signal we wish to analyse in different ways
  - Extremes
  - Pulse
  - Arrhythmia
  - …



- How can we implement these different kinds of analyses of the signal?

ECG example – all in one class

- Implementation using switch/case in one (big) class



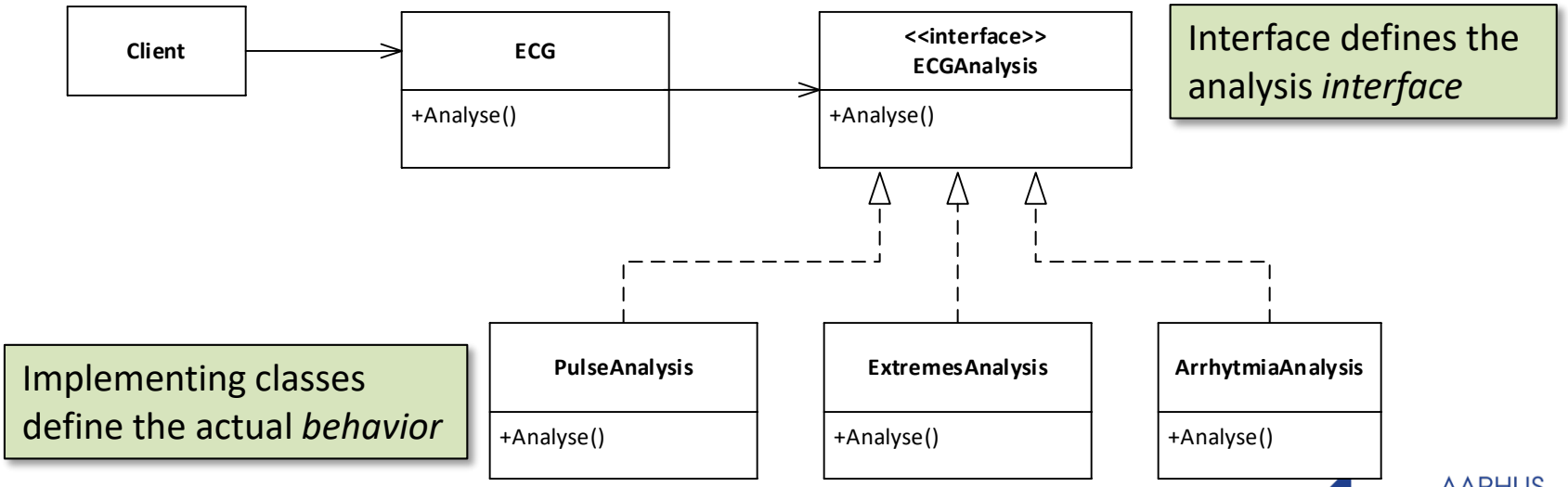

```
class ECG
{
  public enum ECGAnalyses
    { PULSE, EXTREMES, ARRHYTMIA } curAnalysisType;

  void Analyse(double[] samples)
  {
    switch(curAnalysisType)
    {
      case PULSE:
        // Do pulse analysis on samples

      case EXTREMES:
        // Find min/max values of the samples

      case ARRHYTMIA:
        // Analyse for arrhytmia
    }
  }
}
```

- Very fragile when changes or additions occur.
- All analyses are in one switch/case – becomes incomprehensible.
- Very hard to *test*
- Class ECG becomes very large

```
public void Main() // The client of the analysis
{
  ECG _ecg = new ECG();
  _ecg.curAnalysisType = ECG.PULSE;
  _ecg.Analyse(someBigDataSet);
}
```

ECG example – using polymorphism

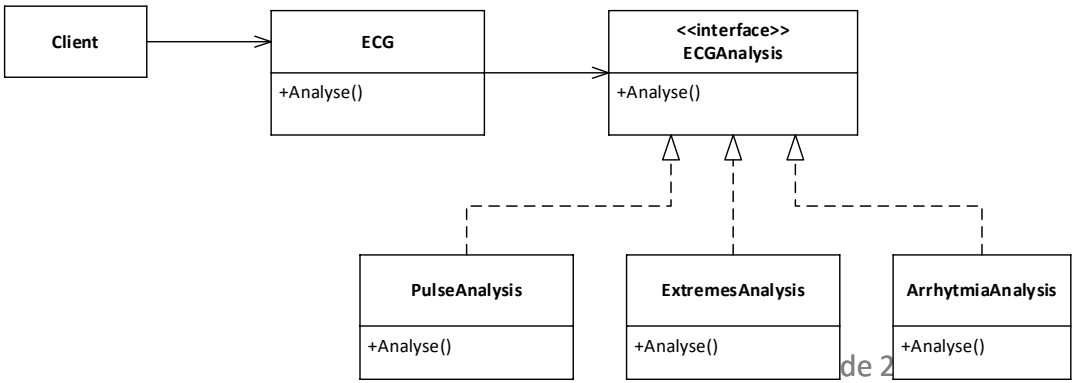
- Note how we need different *kinds* of analysis, i.e. different kinds of behavior.
- We will *isolate* this varying behavior and implement it using polymorphism (i.e. in super- and subclasses)



Polymorphism – ECG example

```
public void Main() // The client of the analysis
{
  ECG _ecg = new ECG();
  _ecg.curAnalysis = new PulseAnalysis()
  _ecg.Analyse(someBigDataSet);
}
```

```
class ECG
{
  public ECGAnalysis _curAnalysis;

  void Analyse(double[] samples)
  {
     _curAnalyses.Analyse(samples);
  }
}
```




```
interface ECGAnalysis
{
  void Analyse(double[] samples);
}
```

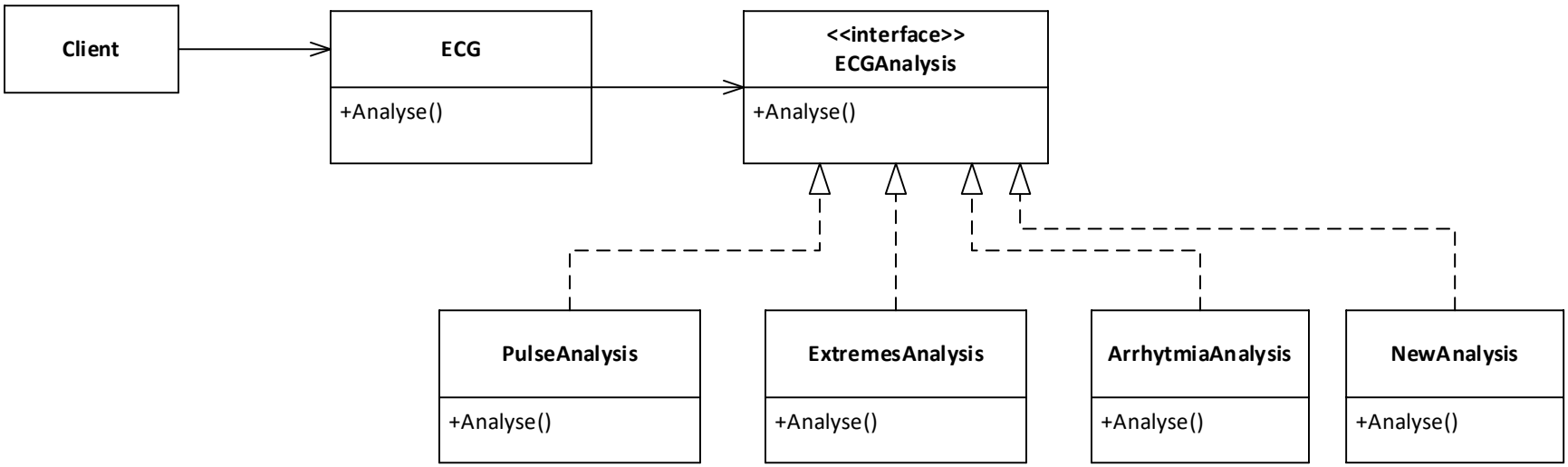
```
class ExtremesAnalyses : ECGAnalysis
{
  void Analyse(double[] samples)
  {
    // Find min/max values of the samples
  }
}
```

```
class PulseAnalyses : ECGAnalysis
{
  void Analyse(double[] samples)
  {
    // Do pulse analysis on samples
  }
}
```

```
class ArrhytmiaAnalyses : ECGAnalysis
{
  void Analyse(double[] samples)
  {
    // Analyse for arrhytmia
  }
}
```

ECG example – using polymorphism

- Note how new kinds of behavior is implemented by implementing new subclasses of `ECGAnalysis`
- *Zero* change is required to existing code!



Time for stunt code

image source: https://i.dailymail.co.uk/i/pix/2008/02_01/stuntmenMOS0202_800x618.jpg