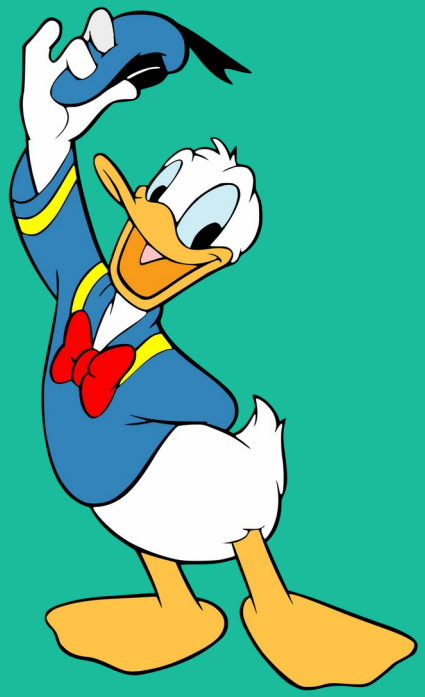
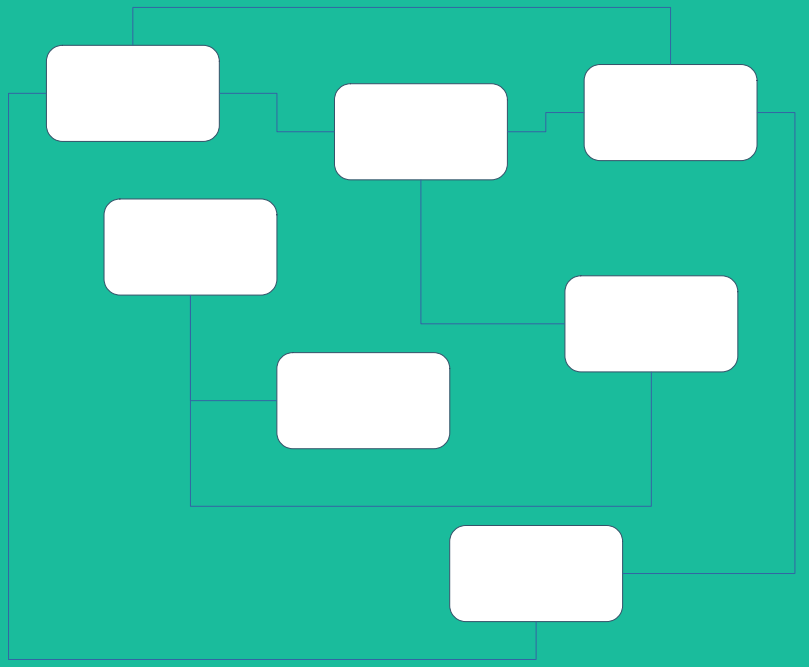
Strategy design pattern



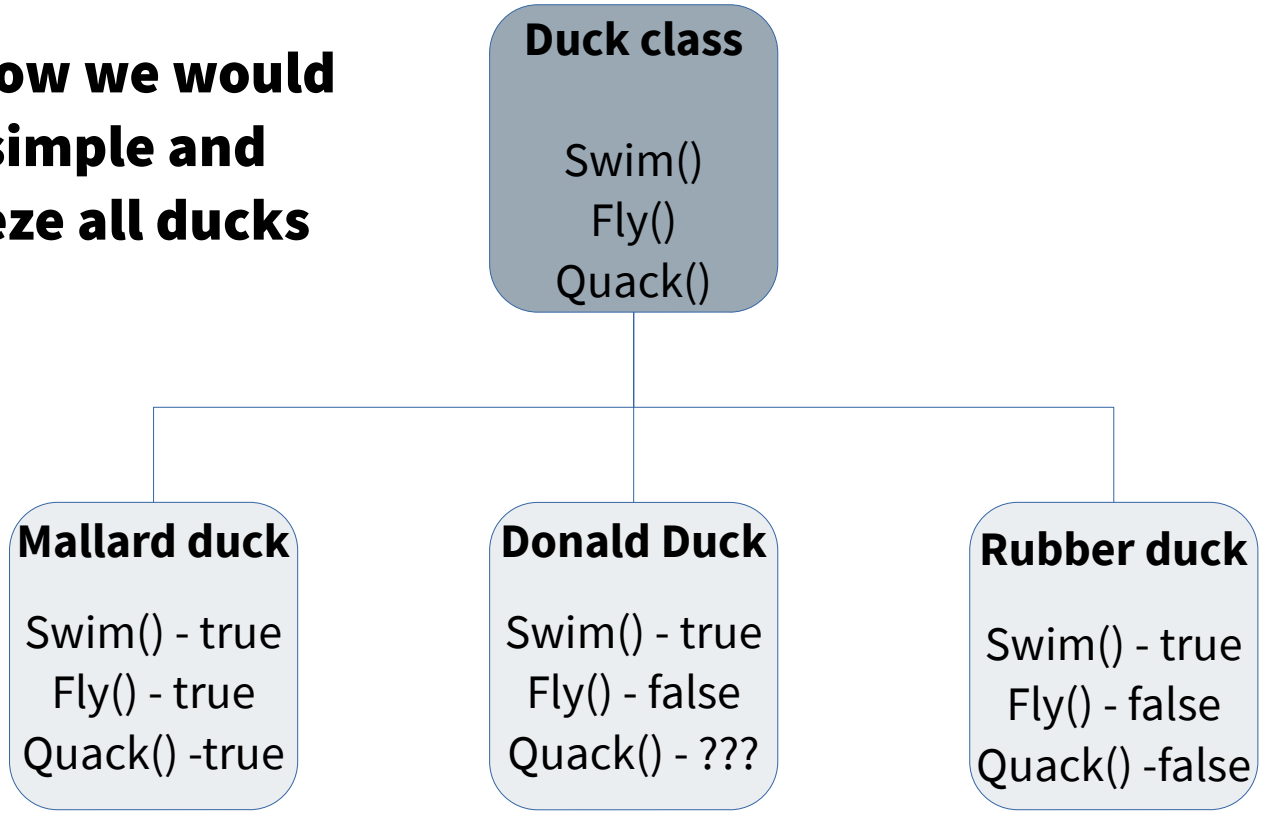


Strategy is a behavioral design pattern that lets you define a family of algorithms, put each of them into a separate class, and make their objects interchangeable.

If inheritance isn’t so straight forward then Strategy pattern helps to loose strict binding of methods to specific class

Ducks are more complicated as it seems

This is an obvious way how we would like to make Duck class simple and clean but we can't squeeze all ducks into one subclass.

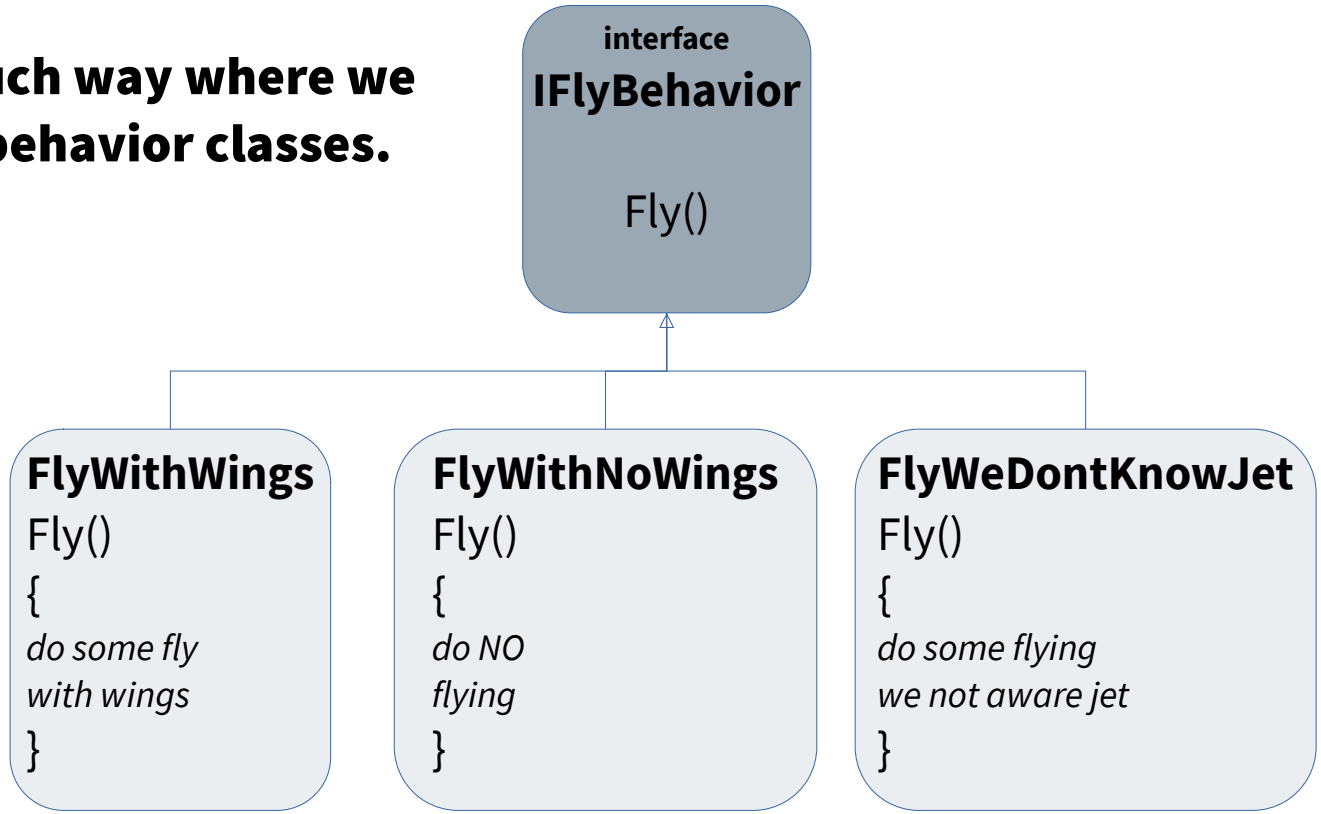


So we need different approach.

Strategy pattern

Strategy pattern shows us such way where we separate methods into new behavior classes.

And no crazy IF construction to distinct different behaviors



Strategy pattern - code

```
public interface IFlyBehaviour
{
    void Fly();
}
```

```
public class FlyWithWings : IFlyBehaviour
{
    public void Fly()
    {
        Console.WriteLine("I'm flying!");
    }
}
```

```
public class FlyWithNoWings : IFlyBehaviour
{
    public void Fly()
    {
        Console.WriteLine("I can't fly!");
    }
}
```

```
public class FlyWithAirplane : IFlyBehaviour
{
    public void Fly()
    {
        Console.WriteLine("I fly only on airplane!");
    }
}
```

```
public interface IQuackBehaviour
{
    void Quack();
}
```

```
public class Quacky : IQuackBehaviour
{
    public void Quack()
    {
        Console.WriteLine("Quack");
    }
}
```

```
public class MuteQuack : IQuackBehaviour
{
    public void Quack()
    {
        Console.WriteLine("Enjoy the silence");
    }
}
```

```
public class Speak : IQuackBehaviour
{
    public void Quack()
    {
        Console.WriteLine("I speak");
    }
}
```

Strategy pattern - code

```
public abstract class Duck
{
    private readonly IFlyBehaviour _flyBehaviour;
    private readonly IQuackBehaviour _quackBehaviour;
    protected Duck(IFlyBehaviour flyBehaviour, IQuackBehaviour quackBehaviour)
    {
        _flyBehaviour = flyBehaviour;
        _quackBehaviour = quackBehaviour;
    }
    public void PerformFly()
    {
        _flyBehaviour.Fly();
    }

    public void PerformQuack()
    {
        _quackBehaviour.Quack();
    }
    public abstract void Display();
}
```

```
public class MallardDuck : Duck
{
    public MallardDuck() : base(new FlyWithWings(), new Quacky()) { }

    public override void Display()
    {
        Console.WriteLine("I'm Mallard Duck");
    }
}
```

```
public class RubberDuck : Duck
{
    public RubberDuck() : base(new FlyWithNoWings(), new MuteQuack()) { }

    public override void Display()
    {
        Console.WriteLine("Take me to bath because i'm rubber Duck");
    }
}
```

```
public class DonaldDuck : Duck
{
    public DonaldDuck() : base(new FlyWithAirplane(), new Speak()) { }
    public override void Display()
    {
        Console.WriteLine("My name is Donald Duck");
    }
}
```

Strategy pattern – code & result

```
static void Main(string[] args)
{
    var ducks = new List<Duck>();
    ducks.Add(new MallardDuck());
    ducks.Add(new RubberDuck());
    ducks.Add(new DonaldDuck());

    foreach (var duck in ducks)
    {
        duck.Display();
        duck.PerformQuack();
        duck.PerformFly();
        Console.WriteLine("----------------------");
    }

    Console.ReadKey();
}
```

```
I'm Mallard Duck
Quack
I'm flying!
----------------------
Take me to bath because i'm rubber Duck
Enjoy the silence
I can't fly!
----------------------
My name is Donald Duck
I speak
I fly only on airplane!
----------------------
```