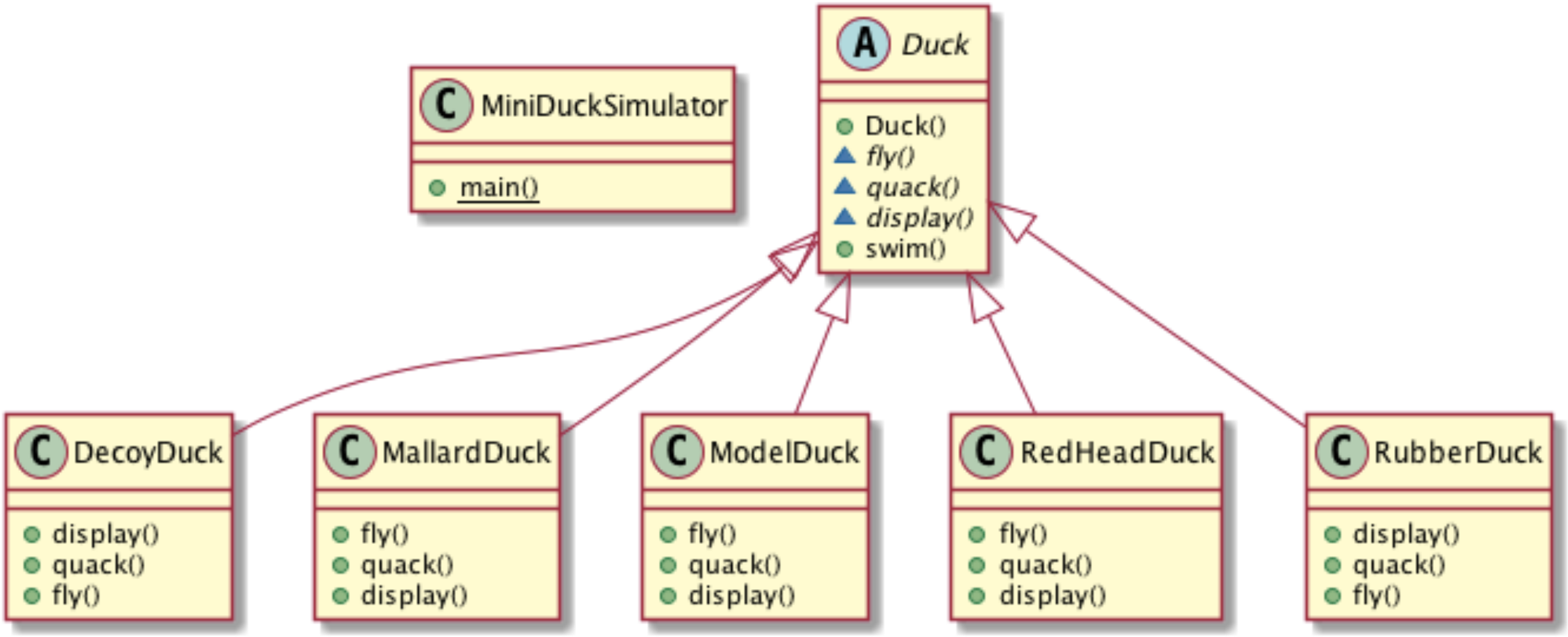
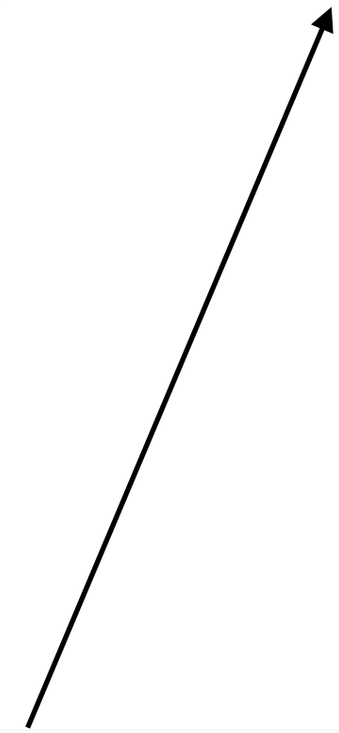
# Problema Strategy

Ànecs


MiniDuckSimulator
main()
Duck
Duck()
fly()
quack()
display()
swim()
DecoyDuck
display()
quack()
fly()
MallardDuck
fly()
quack()
display()
ModelDuck
fly()
quack()
display()
RedHeadDuck
fly()
quack()
display()
RubberDuck
display()
quack()
fly()

```java
public abstract class Duck {
        public Duck() {
        }
        abstract void fly();
        abstract void quack();
        abstract void display();

        public void swim() { System.out.println("All ducks float, even decoys!"); }
}
```

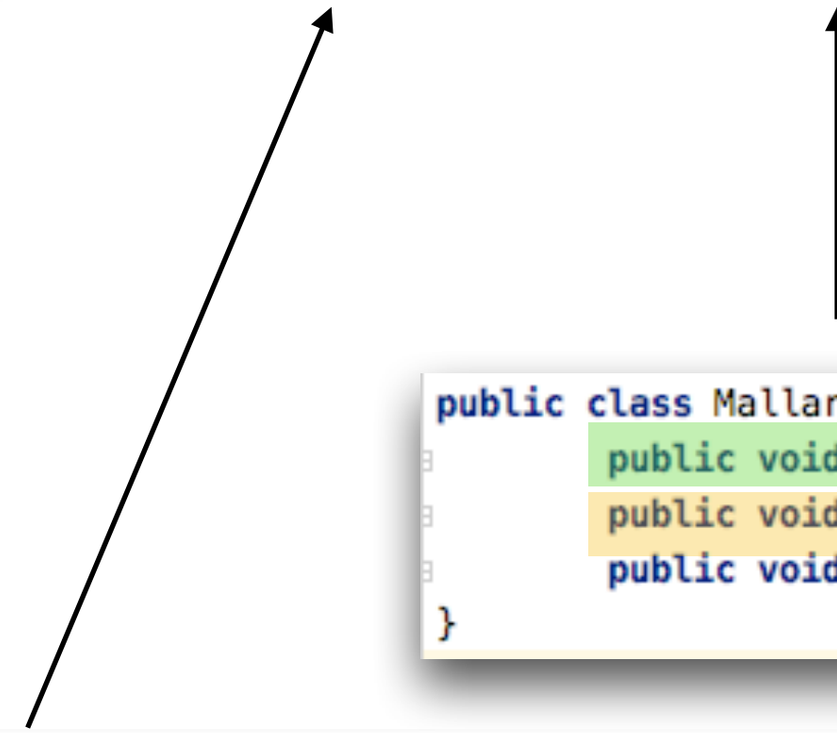


```java
public class MallardDuck extends Duck {
        public void fly() { System.out.println("I'm flying!!"); }
        public void quack() { System.out.println("Quack"); }
        public void display() { System.out.println("I'm a real Mallard duck");
}
```

```java
public class DecoyDuck extends Duck {
        public void display() { System.out.println("I'm a duck Decoy"); }
        public void quack() { System.out.println("<< Silence >>"); }
        public void fly() { System.out.println("I can't fly"); }
}
```

# 1. Principi que es vulnera?

```java
public abstract class Duck {
        public Duck() {
        }
        abstract void fly();
        abstract void quack();
        abstract void display();

        public void swim() { System.out.println("All ducks float, even decoys!"); }
}
```



```java
public class MallardDuck extends Duck {
        public void fly() { System.out.println("I'm flying!!"); }
        public void quack() { System.out.println("Quack"); }
        public void display() { System.out.println("I'm a real Mallard duck");
}
```

```java
public class DecoyDuck extends Duck {
        public void display() { System.out.println("I'm a duck Decoy"); }
        public void quack() { System.out.println("<< Silence >>"); }
        public void fly() { System.out.println("I can't fly"); }
}
```

# Principi que es vulnera?

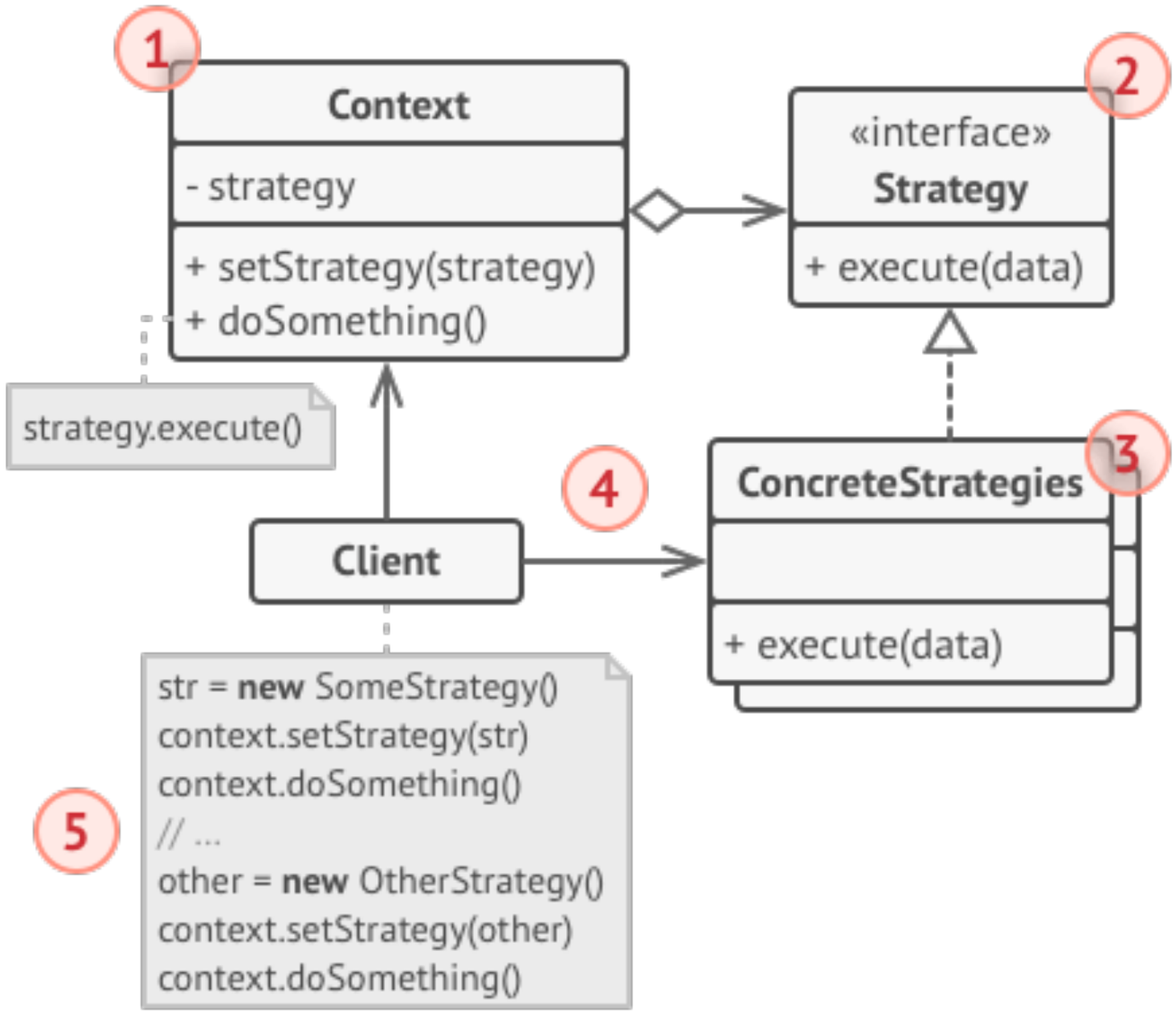
```java
public abstract class Duck {
        public Duck() {
        }
        abstract void fly();
        abstract void quack();
        abstract void display();

        public void swim() { System.out.println("All ducks float, even decoys!"); }
}
```

```java
public class MallardDuck extends Duck {
        public void fly() { System.out.println("I'm flying!!"); }
        public void quack() { System.out.println("Quack"); }
        public void display() { System.out.println("I'm a real Mallard duck");
}
```

```java
public class DecoyDuck extends Duck {
        public void display() { System.out.println("I'm a duck Decoy"); }
        public void quack() { System.out.println("<< Silence >>"); }
        public void fly() { System.out.println("I can't fly"); }
}
```

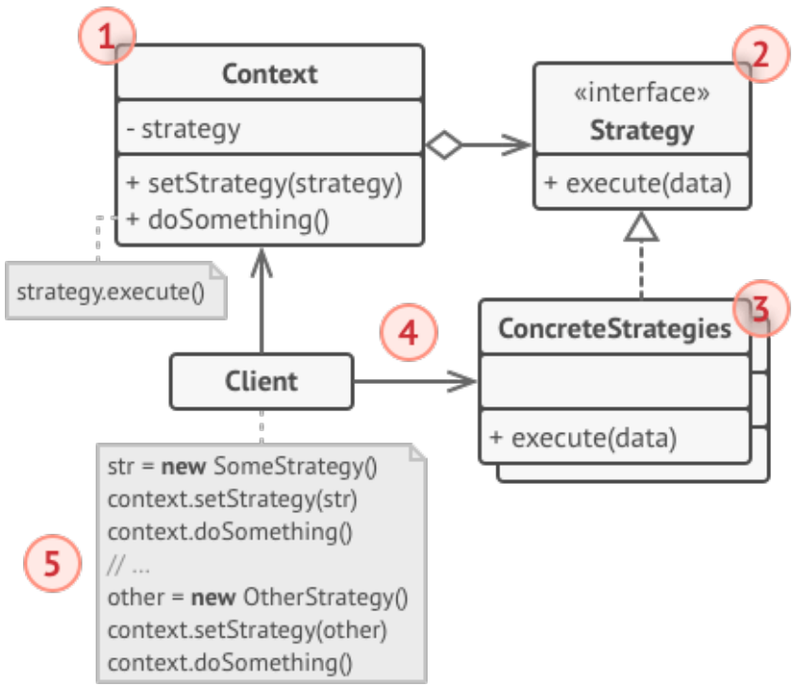
**Liskov!!**

# 2. Patró a aplicar: Strategy



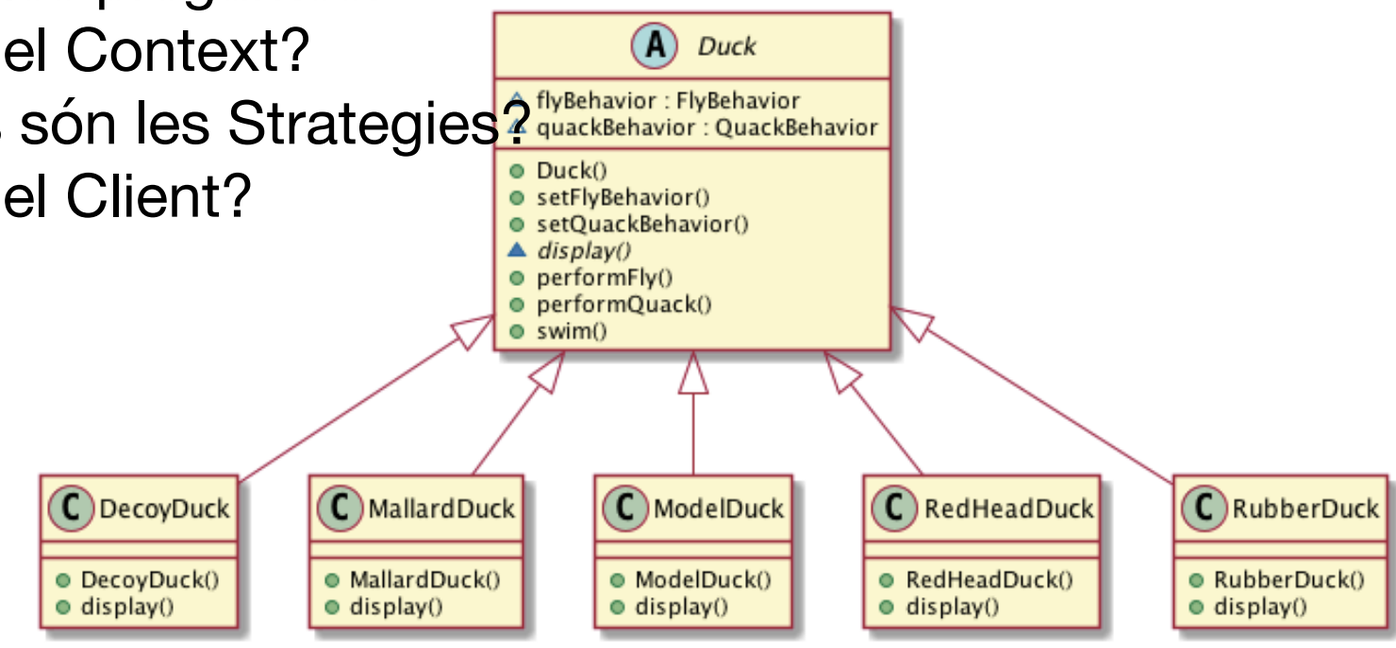

1. El **`Context`** no coneix res de les estratègies concretes, només les sap executar
2. La classe **`Strategy`** declara la interfície comuna a tots els algorismes
3. Les implementacions concretes de les estratègies estan a les classes **`ConcreteStrategies`**
4. El **`Client`** crea l'estratègis concreta que vol fer servir
5. El **`Client`** passa al **`Context`** l'estratègia (setStrategy) i delega en el **`Context`** que l'executi.

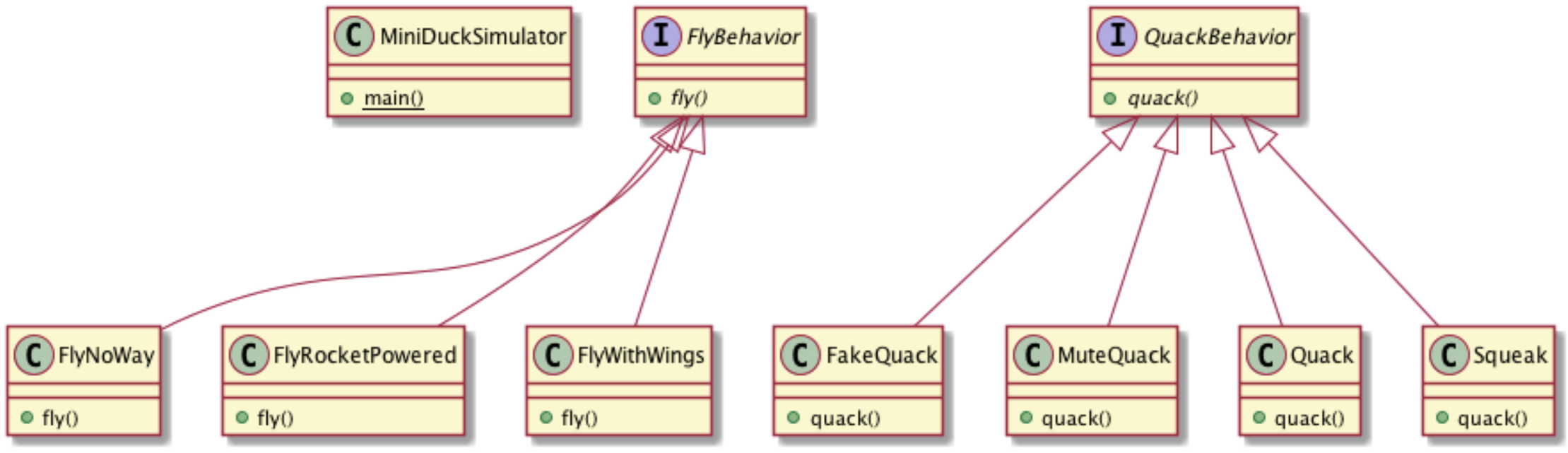
# 3. Aplicar el patró




Contestar les preguntes:

1. Qui és el Context?
2. Quines són les Strategies?
3. Qui és el Client?



# 4. Com funciona el main?

**Abans d'aplicar el patró:**

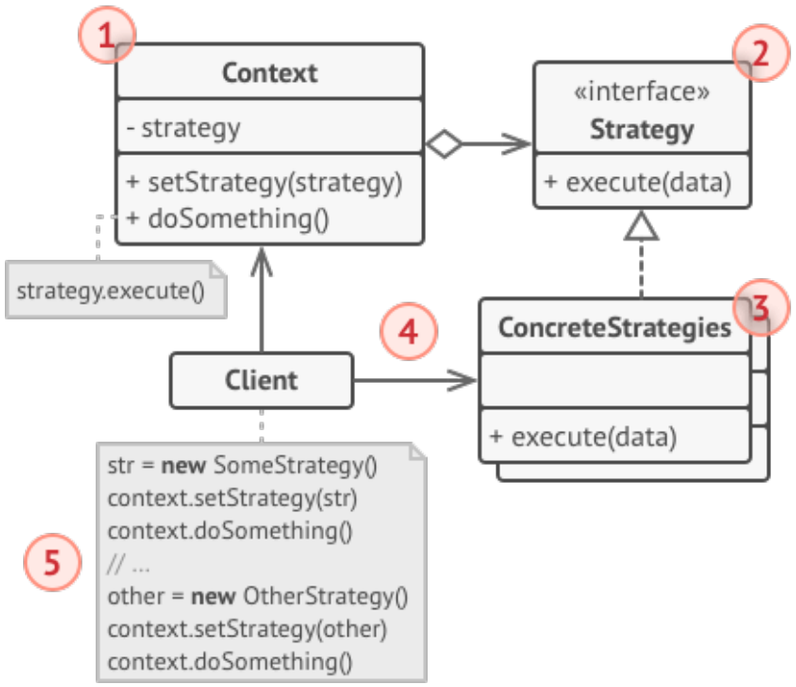
```
public static void main(String[] args) {

    MallardDuck mallard = new MallardDuck();
    RubberDuck  rubberDuckie = new RubberDuck();
    DecoyDuck   decoy = new DecoyDuck();

    Duck     model = new ModelDuck();

    mallard.display();
    mallard.fly();
    mallard.quack();

    rubberDuckie.display();
    rubberDuckie.fly();
    rubberDuckie.quack();

    decoy.display();
    decoy.fly();
    decoy.quack();

    model.display();
    model.fly();
    model.quack();
}
```

**Després d'aplicar el patró:**

```
public class MiniDuckSimulator {

    public static void main(String[] args) {

        MallardDuck mallard = new MallardDuck();
        RubberDuck  rubberDuckie = new RubberDuck();
        DecoyDuck   decoy = new DecoyDuck();
        Duck     model = new ModelDuck();

        mallard.performQuack();
        rubberDuckie.performQuack();
        decoy.performQuack();

        model.performFly();
        model.setFlyBehavior(new FlyRocketPowered());
        model.performFly();
    }
}
```

```
public class MallardDuck extends Duck {
    public MallardDuck() {

        quackBehavior = new Quack();
        flyBehavior = new FlyWithWings();

    }
    public void display() { System.out.println("I'm a real Mallard duck"); }
}
```

# 5. Com queden els principis?




**S: Single Responsability**

**O: Open Closed**

**L: Liskov**

**I: Interface Segregation**

**D: Dependency**