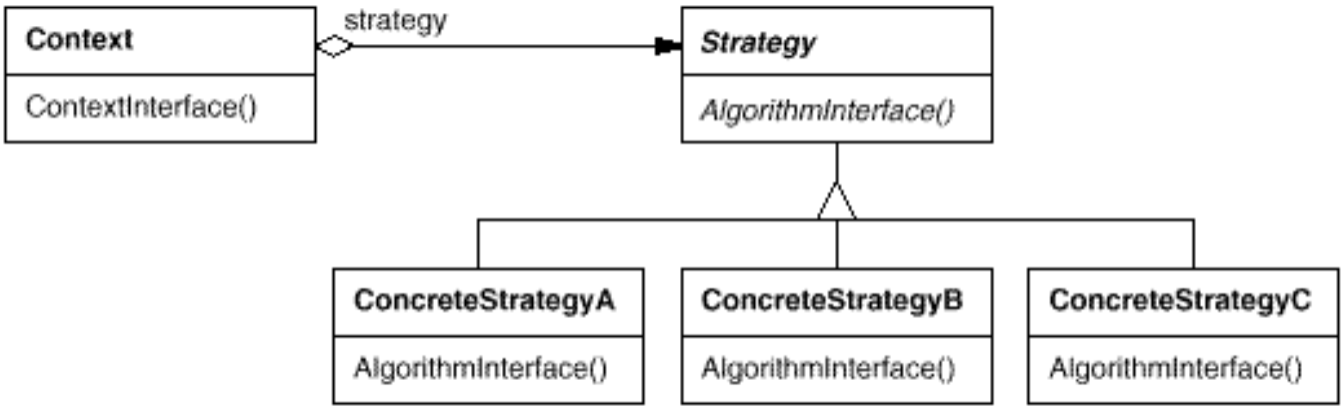
Strategy

↓ INTENT ↓

```
Ogni volta che si ha un algoritmo che ammette varianti si deve usare una
Strategy, in moda da riutilizzare sempre lo stesso metodo (utile per
aggiungere algoritmi in runtime).
```

↓ STRUCTURE ↓



↓ IMPLEMENTATION ↓

1. Definizione delle interfacce Strategy e Context.

Le interfacce Strategy e Context devono fornire a ConcreteStrategy l'accesso ai dati di cui ha bisogno.

Un approccio è fare in modo che Context passi i dati nei parametri a Strategyoperations.

In ogni caso, la Stategy può richiedere esattamente ciò di cui ha bisogno. Le esigenze del particolare algoritmo ed i suoi requisiti di dati determineranno la tecnica migliore.

2. Rendere opzionali gli oggetti Strategy.

La classe Context può essere semplificata se è significativo non avere un oggetto Strategy. Il Context controlla se ha uno Strategyobject prima di accedervi.

- Se ce n'è uno, il Context lo usa normalmente.
- Se non c'è una Strategy, il contesto esegue il comportamento predefinito.

Il vantaggio di questo approccio è che i client non devono occuparsi affatto degli oggetti Strategy a meno che non amino il comportamento predefinito.

↓ EXAMPLE ↓

Strategy ↓

```
public interface Strategy {
    public int doOperation(int n1, int n2);         <--
}
```

Context ↓

```
public class Context {
    private Strategy strategy;                      <--

    public Context(Strategy strategy) {
        this.strategy = strategy;
    }

    public int executeStrategy(int n1, int n2) {
        return strategy.doOperation(n1, n2);        <--
    }
}
```

OperationAdd ↓

```
public class OperationAdd implements Strategy {

    @Override
    public int doOperation(int n1, int n2) {
        return n1 + n2;
    }

}
```

OperationMult ↓

```
public class OperationMult implements Strategy {

    @Override
    public int doOperation(int n1, int n2) {
        return n1 * n2;
    }

}
```

Main ↓

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

    c = new Context(new OperationAdd());
    System.out.println("10 + 5 = " + c.executeStrategy(10, 5));

    c = new Context(new OperationMult());
    System.out.println("10 * 5 = " + c.executeStrategy(10, 5));
}
```