Strategy Pattern

PD2 – CRISAN ADRIAN GABRIEL, 30443

Definition

- It is a behavioral Design Pattern.
- Defines a family of algorithms, encapsulate each one, and make them interchangeable.
- Lets the algorithm vary independently from the clients that use it.
- Captures the abstraction in an interface, bury implementation details in derived classes.

Pros and Cons

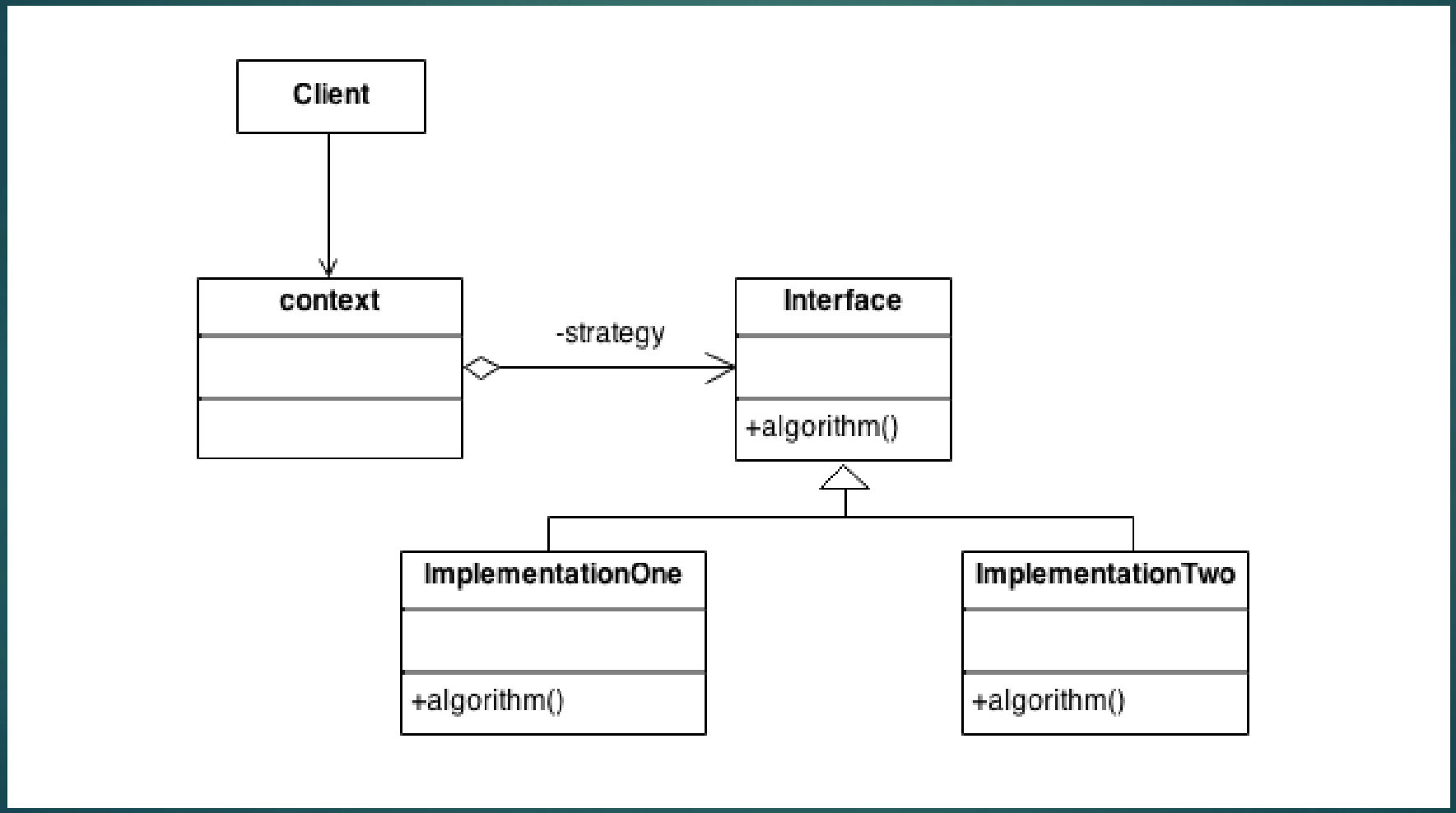
Pros

- Prevents the conditional statements. (switch, if, else…)
- The algorithms are loosely coupled with the context entity. They can be changed/replaced without changing the context entity.
- Easily extendable.

Cons

- Clients must know existence of different strategies and a client must understand how the Strategies differThe algorithms are loosely coupled with the context entity. They can be changed/replaced without changing the context entity.
- It increases the number of objects in the application.

Structure – Class Diagram



Example

```
// 1. Define the interface of the algorithm
interface Strategy {
    void solve();
}

// 2. Bury implementation
@SuppressWarnings("ALL")
abstract class StrategySolution implements Strategy {
    // 3. Template Method
    public void solve() {
        start();
        while (nextTry() && !isSolution()) {}
        stop();
    }

    abstract void start();
    abstract boolean nextTry();
    abstract boolean isSolution();
    abstract void stop();
}

class FOO extends StrategySolution {
    private int state = 1;

    protected void start() {
        System.out.print("Start  ");
    }

    protected void stop() {
        System.out.println("Stop");
    }

    protected boolean nextTry() {
        System.out.print("NextTry-" + state++ + "  ");
        return true;
    }

    protected boolean isSolution() {
        System.out.print("IsSolution-" + (state == 3) + "  ");
        return (state == 3);
    }
}
```

```
// 2. Bury implementation
abstract class StrategySearch implements Strategy {
    // 3. Template Method
    public void solve() {
        while (true) {
            preProcess();
            if (search()) {
                break;
            }
            postProcess();
        }
    }

    abstract void preProcess();
    abstract boolean search();
    abstract void postProcess();
}

@SuppressWarnings("ALL")
class BAR extends StrategySearch {
    private int state = 1;

    protected void preProcess()  {
        System.out.print("PreProcess  ");
    }

    protected void postProcess() {
        System.out.print("PostProcess  ");
    }

    protected boolean search() {
        System.out.print("Search-" + state++ + "  ");
        return state == 3 ? true : false;
    }
}
```

```
// 4. Clients couple strictly to the interface
public class StrategyDemo {
    // client code here
    private static void execute(Strategy strategy) {
        strategy.solve();
    }

    public static void main( String[] args ) {
        Strategy[] algorithms = {new FOO(), new BAR()};
        for (Strategy algorithm : algorithms) {
            execute(algorithm);
        }
    }
}
```