Chain of Responsibility Pattern

Michal Moravik, SD20w2

Name: Chain of Responsibility Pattern

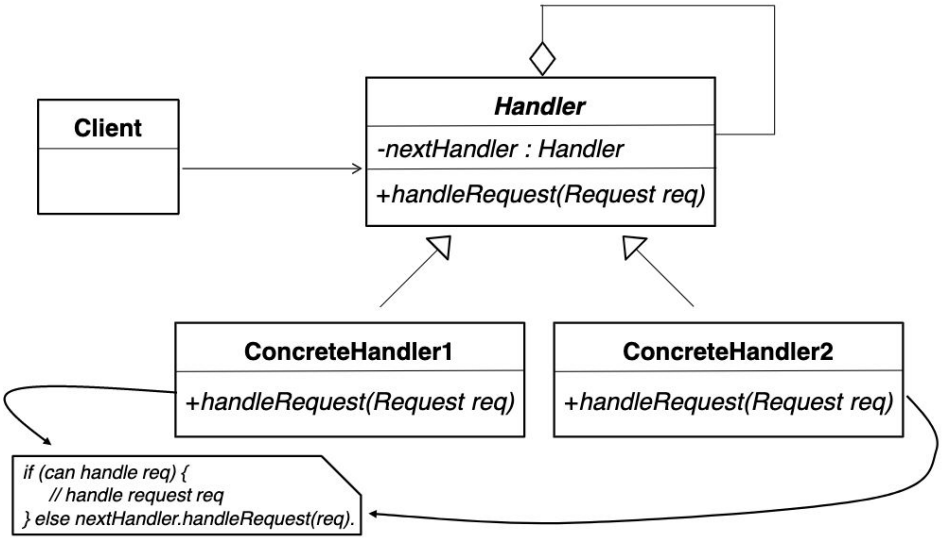
Category: Behavioral

Intent: The pattern lets you pass a request through a series of handlers (objects). Each handler then decides either to process the request or send it to the next handler in the chain.

Motivation: We want to use the pattern when our system is expected to process different kinds of requests in various ways, but the exact type of handler which should be chosen is not known in advance.

Based on the request, the right handler is chosen. So the intent of this pattern is to pick the one from the chain which suits well for handling the request. This way we can **avoid coupling** of the sender of a request and a specific and only one handler.

UML:



Implementation:

```
public enum CaseType {
    BOMB, MURDER, DRUGS
}
```

```java
public class Case {
    private CaseType caseType;

    public Case(CaseType caseType) {
        this.caseType = caseType;
    }

    public CaseType getCaseType() {
        return caseType;
    }
}
```

```java
// HANDLER
public abstract class PoliceUnit {
    private PoliceUnit nextPoliceUnit;

    public PoliceUnit(PoliceUnit nextPoliceUnit) {
        this.nextPoliceUnit = nextPoliceUnit;
    }

    public void handleCase(Case policeCase) {
        if (nextPoliceUnit != null) {
            nextPoliceUnit.handleCase(policeCase);
        }
    }

    public void printHandling(Case policeCase) {
        System.out.println(this.getClass().getSimpleName() +
                " handling a case of type: " + policeCase.getCaseType());
    }
}
```

```java
public class BombSquad extends PoliceUnit {

    public BombSquad(PoliceUnit nextPoliceUnit) {
        super(nextPoliceUnit);
    }

    @Override
    public void handleCase(Case policeCase) {
        if (policeCase.getCaseType().equals(CaseType.BOMB)) {
            printHandling(policeCase);
        } else {
            super.handleCase(policeCase);
        }
    }
}
```

```java
public class TacticalUnit extends PoliceUnit {

    public TacticalUnit(PoliceUnit nextPoliceUnit) { super(nextPoliceUnit); }

    @Override
    public void handleCase(Case policeCase) {
        if (policeCase.getCaseType().equals(CaseType.MURDER)) {
            printHandling(policeCase);
        } else {
            super.handleCase(policeCase);
        }
    }
}
```

```java
public class DEA extends PoliceUnit {

    public DEA(PoliceUnit nextPoliceUnit) { super(nextPoliceUnit); }

    @Override
    public void handleCase(Case policeCase) {
        if (policeCase.getCaseType().equals(CaseType.DRUGS)) {
            printHandling(policeCase);
        } else {
            super.handleCase(policeCase);
        }
    }
}
```

```java
// CHAIN
public class PoliceStation {
    // a chain from police units
    PoliceUnit chain;

    public PoliceStation() {
        buildChain();
    }

    private void buildChain() {
        chain = new BombSquad(new TacticalUnit(new DEA(null)));
    }

    public void makeCase(Case policeCase) {
        chain.handleCase(policeCase);
    }
}
```

```java
public class Client {
    public static void main(String[] args) {
        PoliceStation policeStation = new PoliceStation();
        policeStation.makeCase(new Case(CaseType.MURDER));
        policeStation.makeCase(new Case(CaseType.DRUGS));
        policeStation.makeCase(new Case(CaseType.BOMB));
    }
}
```

```
TacticalUnit handling a case of type: MURDER
DEA handling a case of type: DRUGS
BombSquad handling a case of type: BOMB
```

Consequences:

Pros:

- We can control the order of handlers
- We can let the system choose the proper handler for the request during the runtime
- Decoupling classes that invoke actions (clients) from the classes (handlers) that process this request - single responsibility principle
- We can add as many handlers as we want without breaking the existing client's code

Cons:

- The possibility that some requests end up unhandled if we do not cover all situations

Known uses:

- Servlet filters

Related patterns:

- Used with Composite
- The structure can remind Decorator's structure a bit