Chain of responsibility Pattern

각 요구를 처리할 수 있는지, 없는지를 순서대로 검색해가며 책임 떠넘기기. 복수의 객체가 연결되어 있는 내부의 어딘가에서 처리 수행. “어떤 처리를 어떤 객체가 수행해야 한다”는 정보를 중앙집권적으로 갖는게 아니라 각 객체 갖고 처리.

Architecture, Modeling, DesignPattern, Behavioral

Handler

```
public class Trouble {
    private int number;

    public Trouble(int number) {
        super();
        this.number = number;
    }

    public int getNumber() {
        return number;
    }

    @Override
    public String toString() {
        return "Trouble [number=" + number + "]";
    }
}
```

```
public abstract class Support {
    private String name;
    private Support next;
    public Support(String name) {
        this.name = name;
    }

    public Support setNext(Support next) {
        this.next = next;
        return next;
    }
    public final void support(Trouble trouble) {
        if(resolve(trouble))
            done(trouble);
        else if(next != null)
            next.support(trouble);
        else
```

```
            fail(trouble);
    }
    protected abstract boolean resolve(Trouble trouble);
    protected void done(Trouble trouble) {
        System.out.println(trouble + " is resolved by " + this);
    }
    protected void fail(Trouble trouble) {
        System.out.println(trouble + " cannot be resolved.");
    }

    @Override
    public String toString() {
        return "[" + name + "]";
    }
}
```

```
public class NoSupport extends Support {

    public NoSupport(String name) {
        super(name);
    }

    @Override
    protected boolean resolve(Trouble trouble) {
        return false;
    }
}
```

```
public class LimitSupport extends Support {

    private int limit;
    public LimitSupport(String name, int limit) {
        super(name);
        this.limit = limit;
    }

    @Override
    protected boolean resolve(Trouble trouble) {
        return trouble.getNumber() < limit;
    }
}
```

```

```

```
public class SpecialSupport extends Support {

    private int number;
    public SpecialSupport(String name, int number) {
        super(name);
        this.number = number;
    }

    @Override
    protected boolean resolve(Trouble trouble) {
        return trouble.getNumber() % 2 == number;
    }

}
```

```
public class OddSupport extends Support {

    public OddSupport(String name) {
        super(name);
    }

    @Override
    protected boolean resolve(Trouble trouble) {
        return trouble.getNumber() % 2 == 1;
    }

}
```

Client

```
public class Client {
    public static void main(String[] args) {
        Support alice = new NoSupport("Alice");
        Support bob = new LimitSupport("Bob", 100);
        Support charlie = new SpecialSupport("Charlie", 429);
        Support diana = new LimitSupport("Diana", 200);
        Support elmo = new OddSupport("Elmo");
        Support fred = new LimitSupport("Fred", 300);
        // Build the chain of responsibility
alice.setNext(bob).setNext(charlie).setNext(diana).setNext(elmo).setNext(fre
d);
        // Handled by Handlers
        for(int i=0 ; i<500 ; i+=33) {
            alice.support(new Trouble(i));
        }
```

```
    }
}
```