Design Patterns

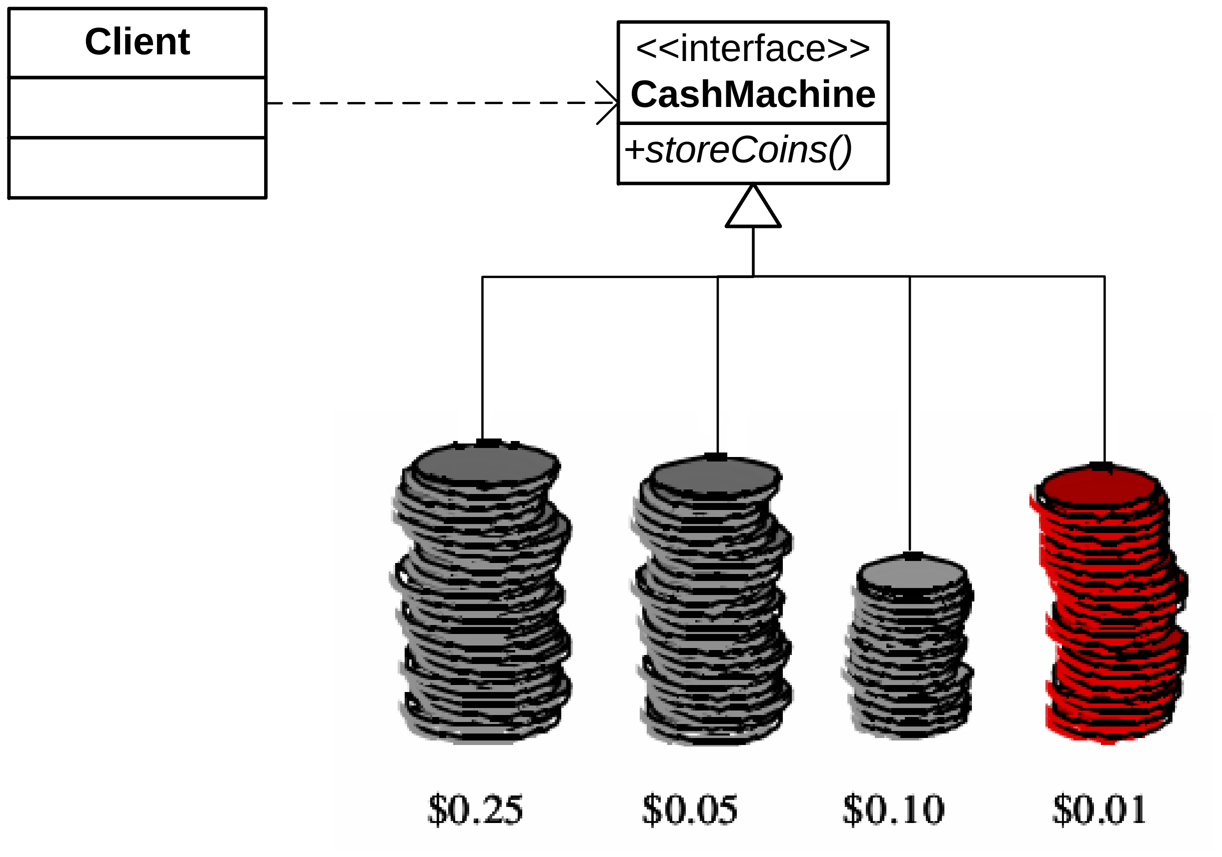
宋 杰
Song Jie
东北大学 软件学院
Software College, Northeastern University

16. Chain Of Responsibility Pattern

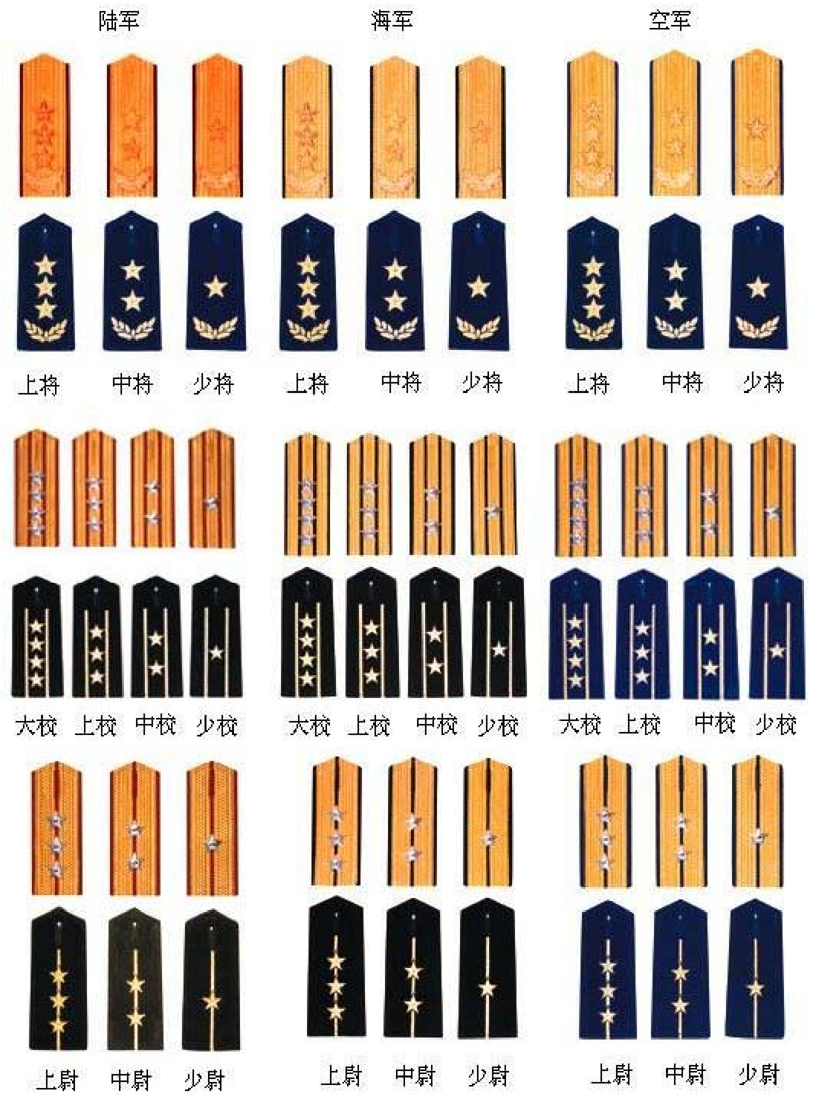
Intent

- Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.
- The chain of responsibility could be a line, a ring, a hierarchy or a graph.
- 在责任链模式里，很多对象由每一个对象对其下家的引用而连接起来形成一条链。请求在这个链上传递，直到链上的某一个对象决定处理此请求。发出这个请求的客户端并不知道链上的哪一个对象最终处理这个请求，这使得系统可以在不影响客户端的情况下动态地重新组织链和分配责任。

Example: Cash Machine



Example: Rank in military



```java
interface Coin {
    public int getParValue();
    public void setParValue(int parValue);
}

class RMBCoin implements Coin {
    private int parValue;
    public RMBCoin() {
    }
    public RMBCoin(int parValue) {
        this.parValue = parValue;
    }
    public int getParValue() {
        return parValue;
    }
    public void setParValue(int parValue) {
        this.parValue = parValue;
    }
}
```

```java
abstract class CoinCollector {
    private CoinCollector collector;
    public CoinCollector(CoinCollector next) {
        this.collector = next;
    }
    protected final CoinCollector next() {
        return collector;
    }
    public final void collect(Coin coin) {
        if (matched(coin)) {
            store(coin);
        } else {
            next().collect(coin);
        }
    }
    protected abstract boolean matched(Coin coin);
    protected abstract void store(Coin coin);
}
```

```java
class OneCoinCollector extends CoinCollector {
    private static final int PAR_VALUE = 1;
    public OneCoinCollector(CoinCollector next) {
        super(next);
    }
    protected boolean matched(Coin coin) {
        return coin.getParValue() == PAR_VALUE;
    }
    protected void store(Coin coin) {
        // store 1-valued Coin
    }
}
```

```java
class FiveCoinCollector extends CoinCollector {
    private static final int PAR_VALUE = 5;
    public FiveCoinCollector(CoinCollector next) {
        super(next);
    }
    protected boolean matched(Coin coin) {
        return coin.getParValue() == PAR_VALUE;
    }
    protected void store(Coin coin) {
        // store 5-valued Coin
    }
}
```

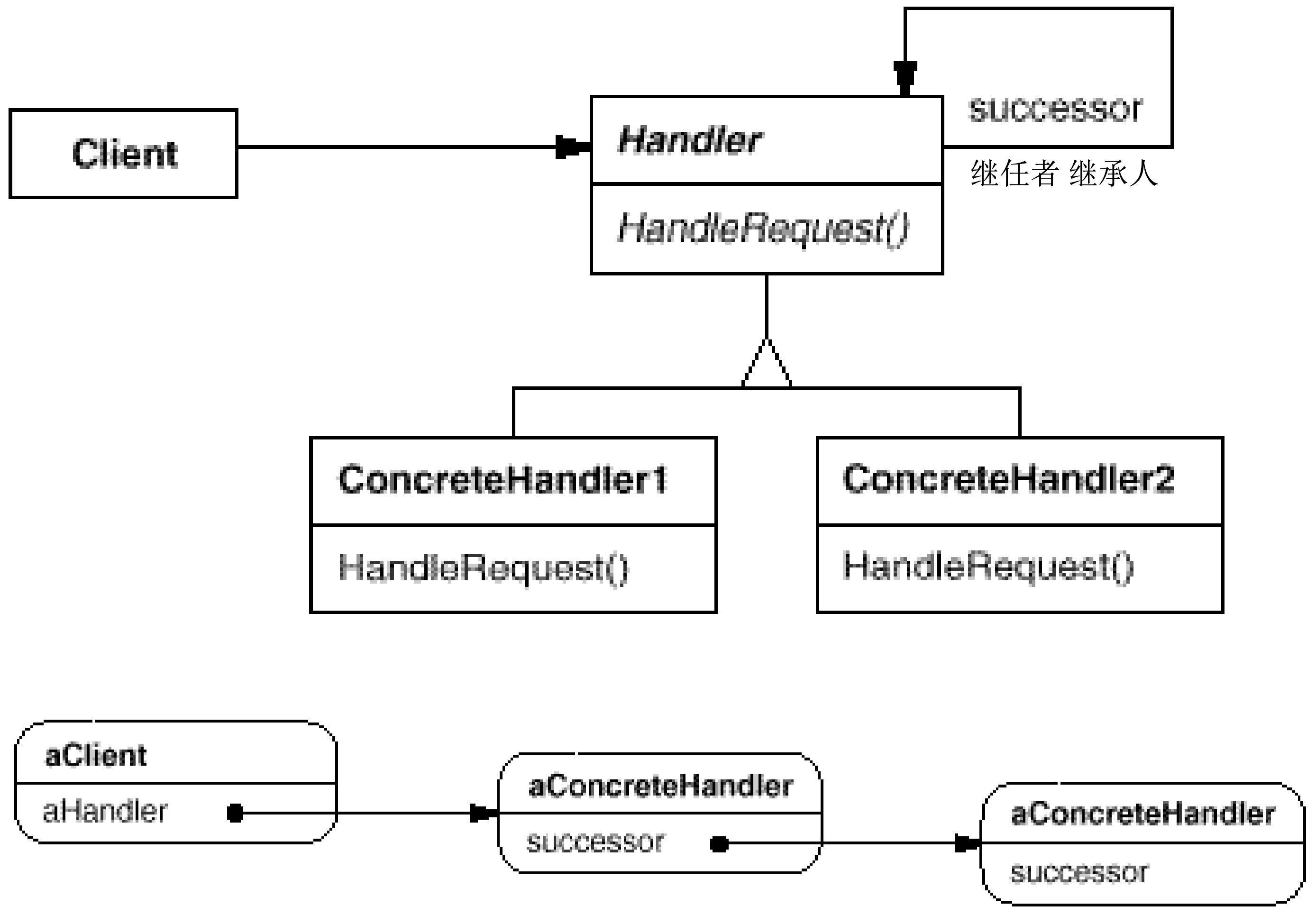
```java
class TenCoinCollector extends CoinCollector {
    private static final int PAR_VALUE = 10;
    public TenCoinCollector(CoinCollector next) {
        super(next);
    }
    protected boolean matched(Coin coin) {
        return coin.getParValue() == PAR_VALUE;
    }
    protected void store(Coin coin) {
        // store 10-valued Coin
    }
}
```

```java
class UnmatchedCoinCollector extends CoinCollector {
    public UnmatchedCoinCollector(CoinCollector next) {
        super(next);
    }
    protected boolean matched(Coin coin) {
        return true;
    }
    protected void store(Coin coin) {
        // reject unmatched Coin
    }
}
```

```java
class CoinClient {
    public void testCoinCollector() {
        CoinCollector coinCollector =
            new OneCoinCollector(
                new FiveCoinCollector(
                        new TenCoinCollector(
                                new UnmatchedCoinCollector(null))));

        Coin coin = new RMBCoin(7);
        coinCollector.collect(coin);
    }
}
```

Structure


Client
Handler
HandleRequest()
successor
继任者 继承人
ConcreteHandler1
HandleRequest()
ConcreteHandler2
HandleRequest()
aClient
aHandler
aConcreteHandler
successor
aConcreteHandler
successor

Participants

- Handler
  - Defines an interface for handling requests.
  - (Optional) Implements the successor link.
- ConcreteHandler
  - Handles requests it is responsible for.
  - If the ConcreteHandler can handle the request, it does so; otherwise it forwards the request to its successor.
- Client
  - Initiates the request to a ConcreteHandler object on the chain.

Collaborations

- When a client issues a request, the request propagates along the chain until a ConcreteHandler object takes responsibility for handling it.

Consequences -benefits

- Reduced coupling, (nothing need to know).
  - The client is free from knowing which object handles a request.
  - The client only has to know that a request will be handled "appropriately."
  - Both the receiver and the sender have no explicit knowledge of each other.
  - An handler in the chain doesn't have to know about the chain's structure.
- Added flexibility in assigning responsibilities to objects.
  - You can combine this with subclassing to specialize handlers statically.
  - You can add or change responsibilities for handling a request by adding to or otherwise changing the chain at run-time.

Consequences-drawbacks

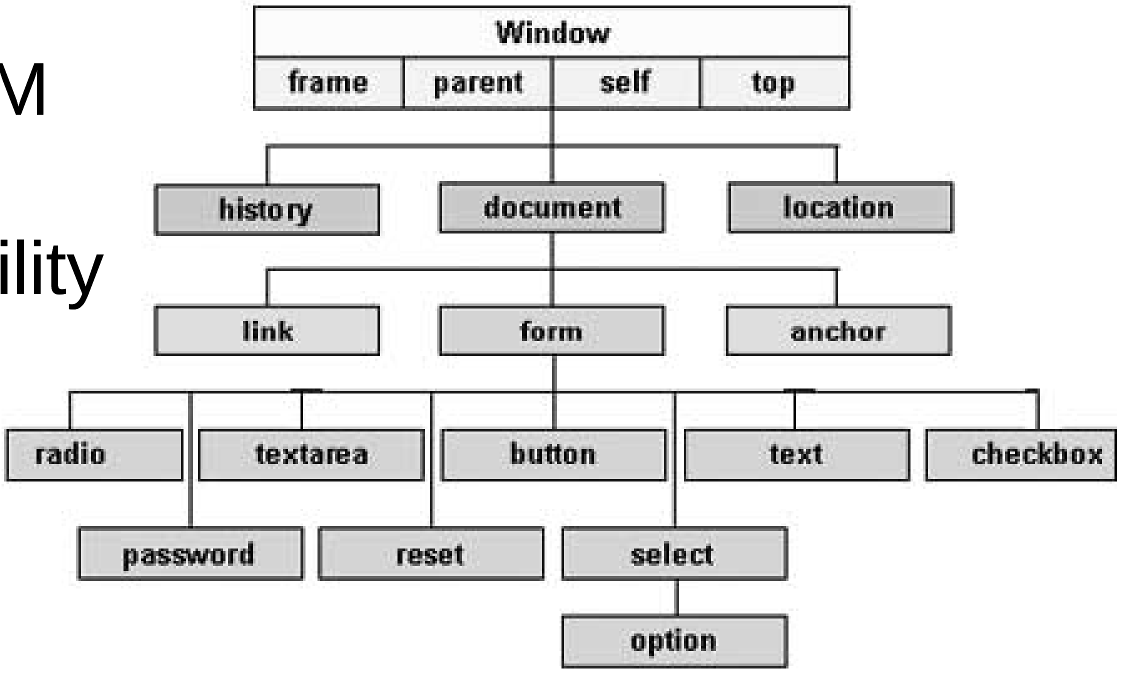
- Receipt isn't guaranteed. Since a request has no explicit receiver, there's no guarantee it'll be handled
  - The request can fall off the end of the chain without ever being handled.
  - A request can also go unhandled when the chain is not configured properly.
- The returned value is possible but not straightforward;
- All handler must confirm the same interface.

Applicability

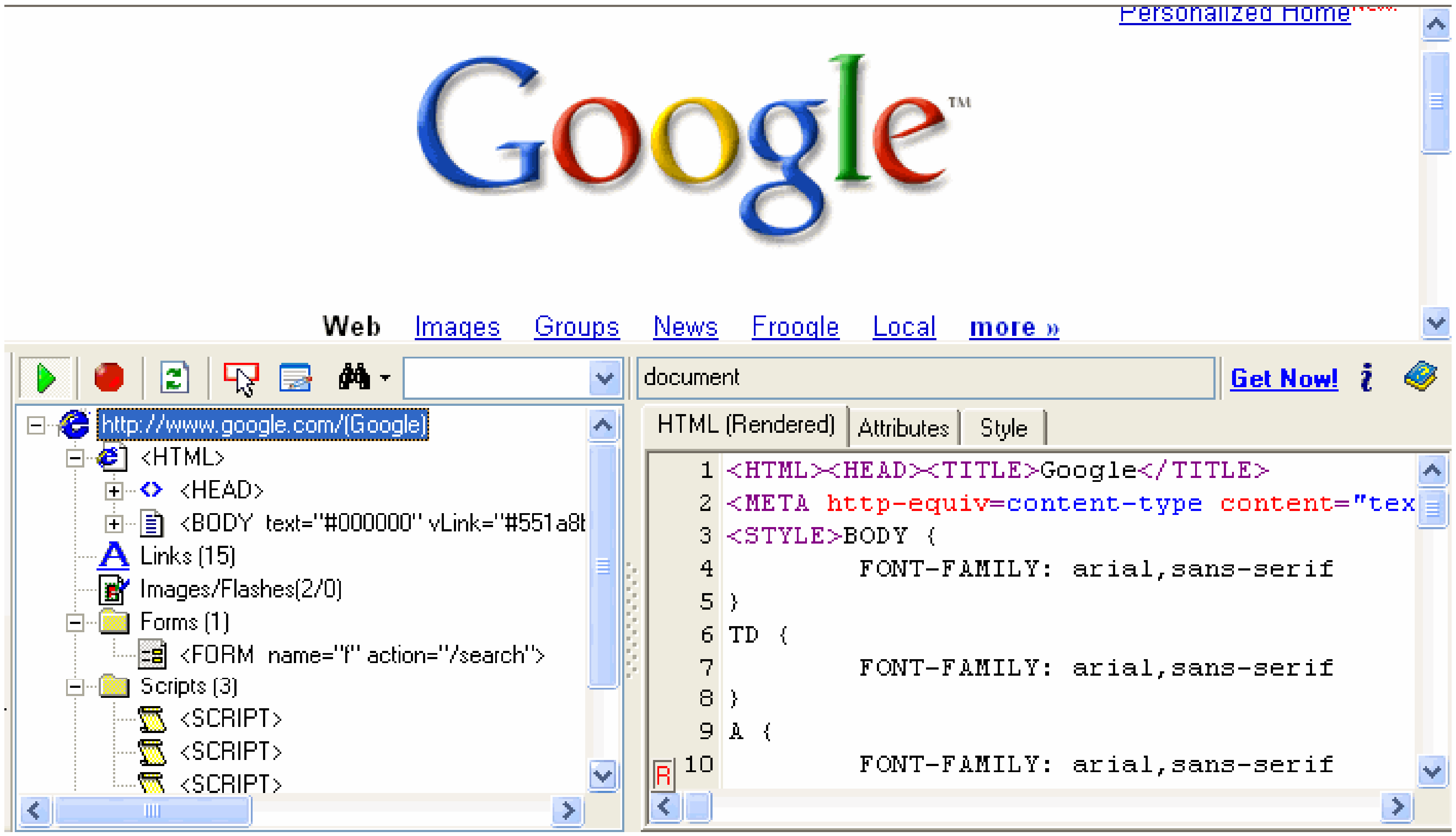
- More than one object may handle a request, and the handler isn't known *a priori*. The handler should be ascertained automatically.
- You want to issue a request to one of several objects without specifying the receiver explicitly.
- The set of objects that can handle a request should be specified dynamically.

Example: Event handler model in DOM (Document Object Model)

- DOM (Document Object Model)
- The event handle mechanism of DOM in Brower adopt chain of responsibility pattern
  - Netscape
  - Internet Explorer



DOM



```
http://www.google.com/(Google)
<HTML>
  <HEAD>
  <BODY text="#000000" vLink="#551a8b
Links (15)
Images/Flashes(2/0)
Forms (1)
  <FORM name="f" action="/search">
Scripts (3)
  <SCRIPT>
  <SCRIPT>
  <SCRIPT>
```

```
<HTML><HEAD><TITLE>Google</TITLE>
<META http-equiv=content-type content="tex
<STYLE>BODY {
        FONT-FAMILY: arial,sans-serif
}
TD {
        FONT-FAMILY: arial,sans-serif
}
A {
        FONT-FAMILY: arial,sans-serif
```

Example: Netscape

- Event handle mechanism in Netscape is named Event Capturing by invoking methods from top to bottom.
  - captureEvent()
  - releaseEvents()
- Event handle mechanism in InternetExplorer is named Event Bubbling by invoking method from bottom to top.
  - onXxx()
  - cancelBubble = true

Extension 1: Representing requests

- Invoking an operation directly
  - In the simplest form, the request is a hard-coded operation invocation.
  - This is convenient and safe;
  - But you can forward only the fixed set of requests that the Handler class defines.

Extension 1: Representing requests

- Using the encoded request code
  - Use a single handler function that takes a request code (e.g., an integer constant or a string) as parameter.
  - The sender and receiver agree on how the request should be encoded.
  - This approach is more flexible, but it requires conditional statements for dispatching the request based on its code.
- Moreover, there's no type-safe way to pass parameters, so they must be packed and unpacked manually.
- Obviously this is less safe than invoking an operation directly.

Extension 1: Representing requests

- Separate request objects
  - Using separate request objects that bundle request parameters.
  - A Request class can represent requests explicitly, and new kinds of requests can be defined by subclassing. Subclasses can define different parameters.
  - Handlers must know the kind of request (that is, which Request subclass they're using) to access these parameters.

Extension 2: Complete and incomplete CoR (chain of responsibility)

- Complete CoR
  - A ConcreteHandler can only choice to take the responsibility or forward the responsibility to the successor;
  - The request must be handled by one and only one handler.
- Incomplete CoR
  - A ConcreteHandle can only choice to take the responsibility partly, then forward the responsibility to the successor;
  - The request must be handled by many or nor handler.

Let's go to next…