Chain of Responsibility

CS356 Object-Oriented Design and Programming
http://cs356.yusun.io
November 17, 2014

Yu Sun, Ph.D.
http://yusun.io
yusun@csupomona.edu

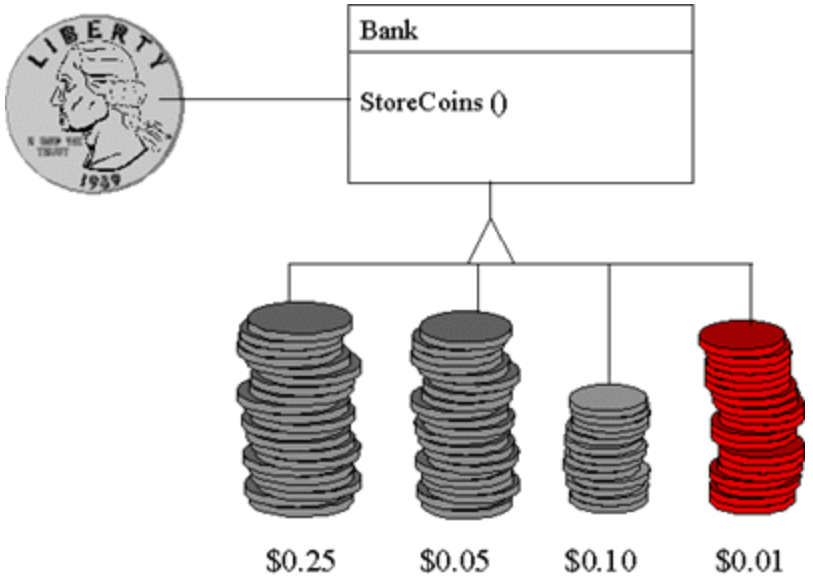
CAL POLY POMONA

Chain of Responsibility

- *Intent*
  - Decouple sender of a request from its receiver
    - By giving more than one object a chance to handle the request
  - Put receivers in a chain and pass the request along the chain
    - Until an object handles it

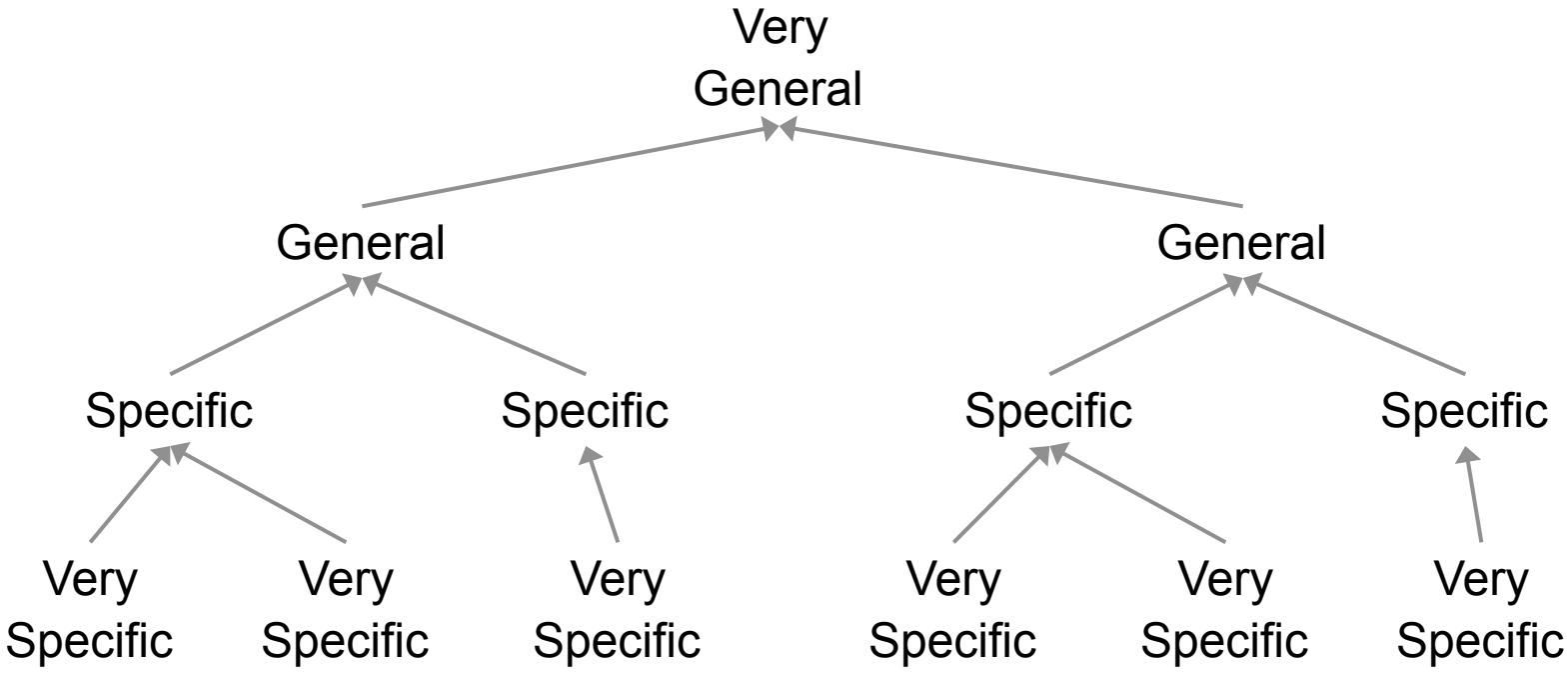
Non-software Example

- Rather than having a separate slot for each coin denomination coupled with receptacle for the denomination, a single slot is used
- When the coin is dropped, the coin is routed to the appropriate receptacle by the mechanical mechanisms within the bank



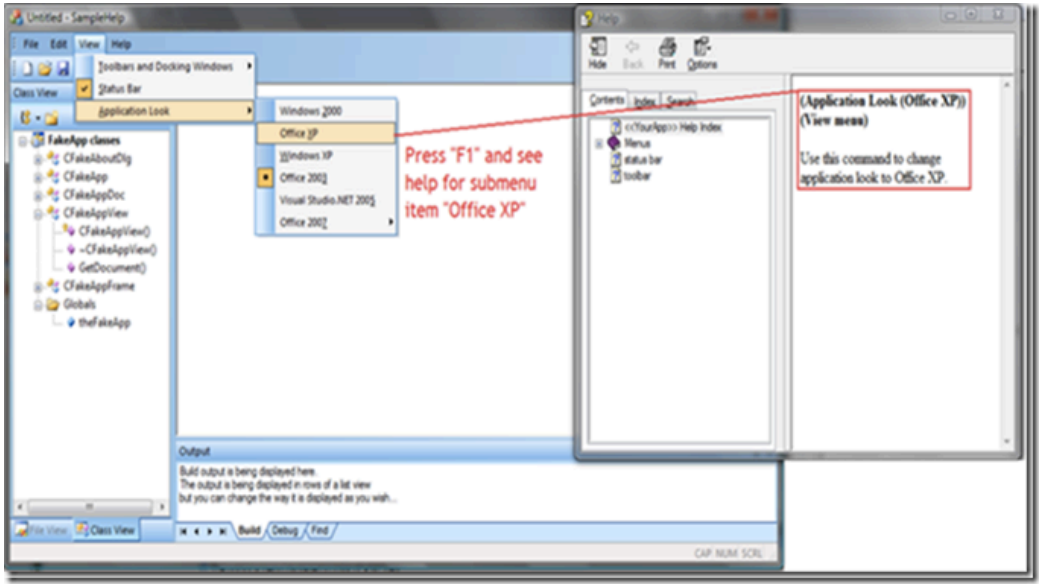
Chain of Commands

- In a military or business hierarchy
  - A request is made
  - It goes up the chain of command until someone has the authority to answer the request

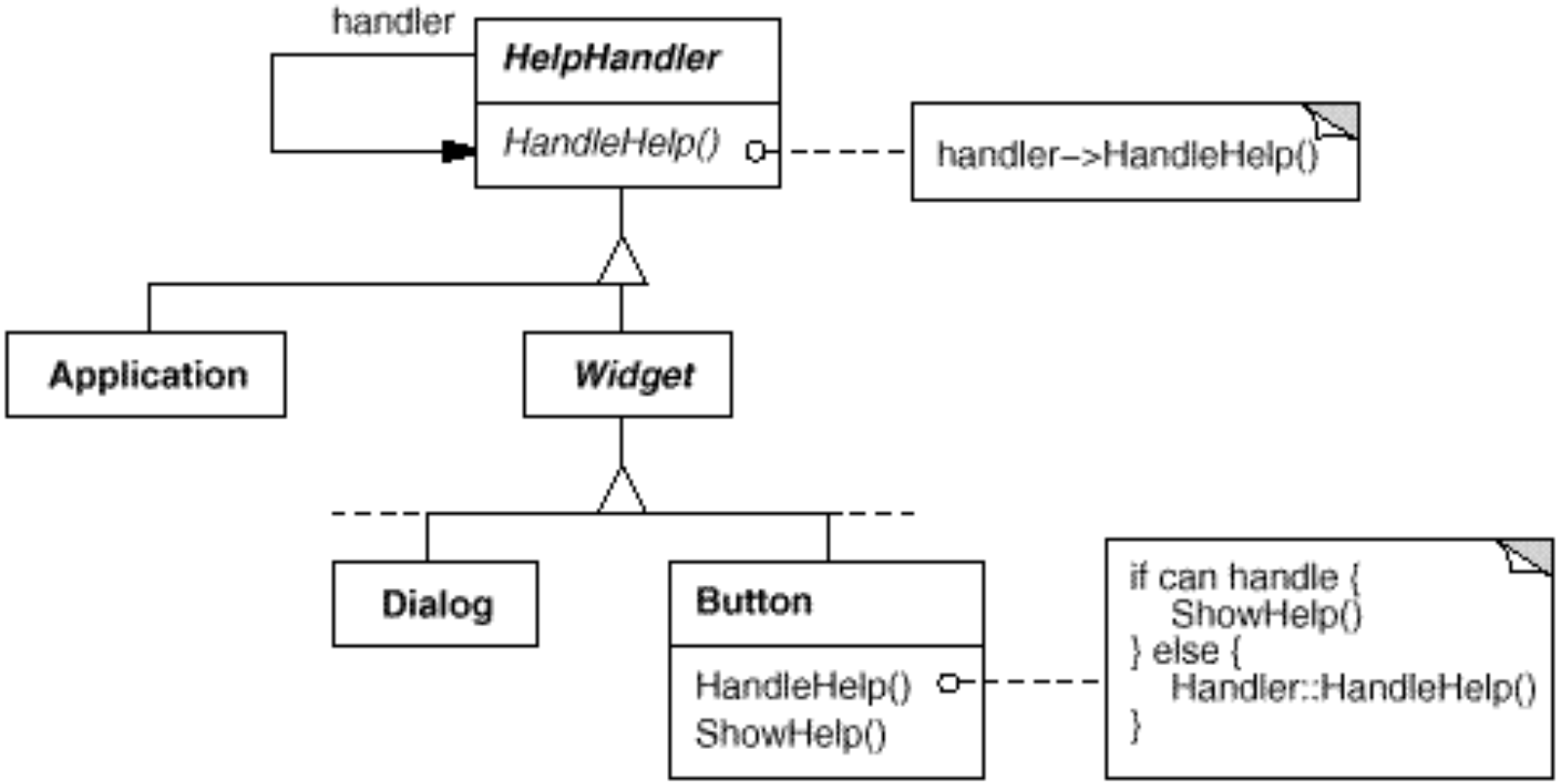


Motivation

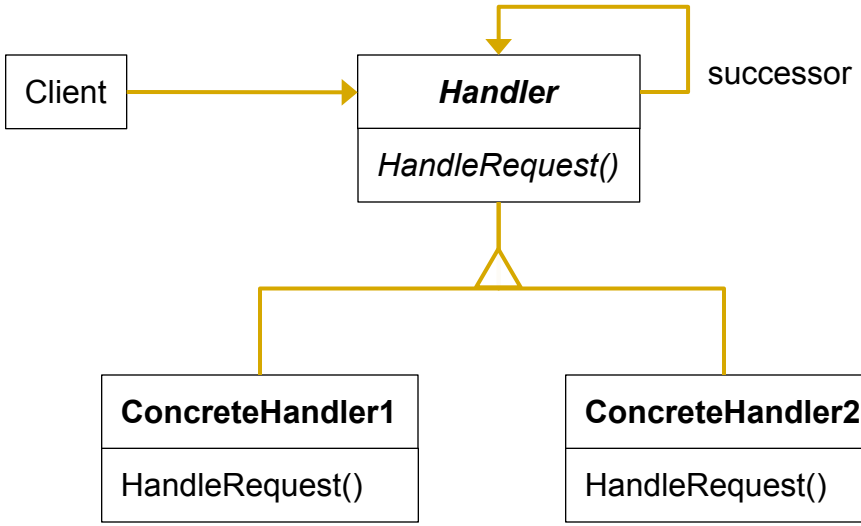
- Context-sensitive help
  - A help request is handled by one of several UI objects
- Which one?
  - Depends on the context
- The object that initiates the request does not know the object that will eventually provide the help

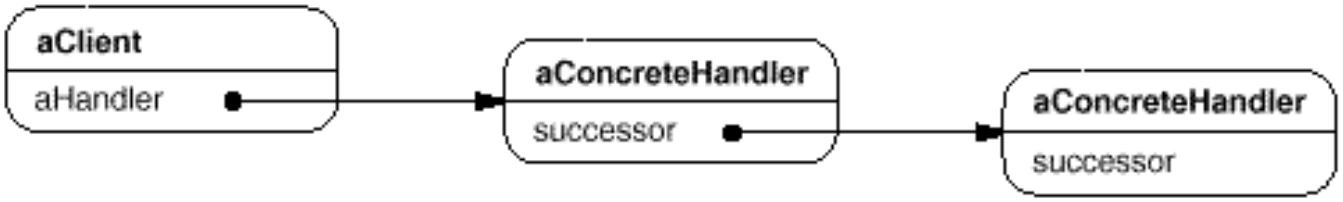


The Context-Help System



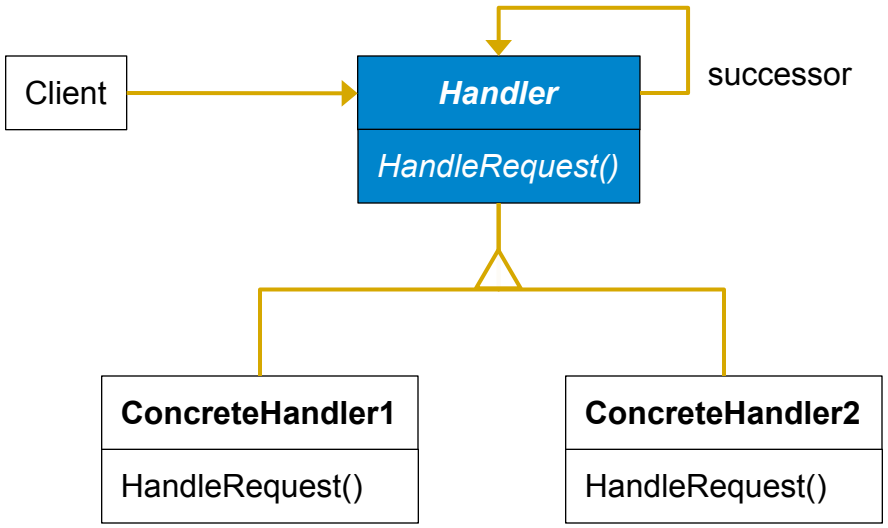
Structure


Client
Handler
HandleRequest()
successor
ConcreteHandler1
HandleRequest()
ConcreteHandler2
HandleRequest()


aClient
aHandler
aConcreteHandler
successor
aConcreteHandler
successor

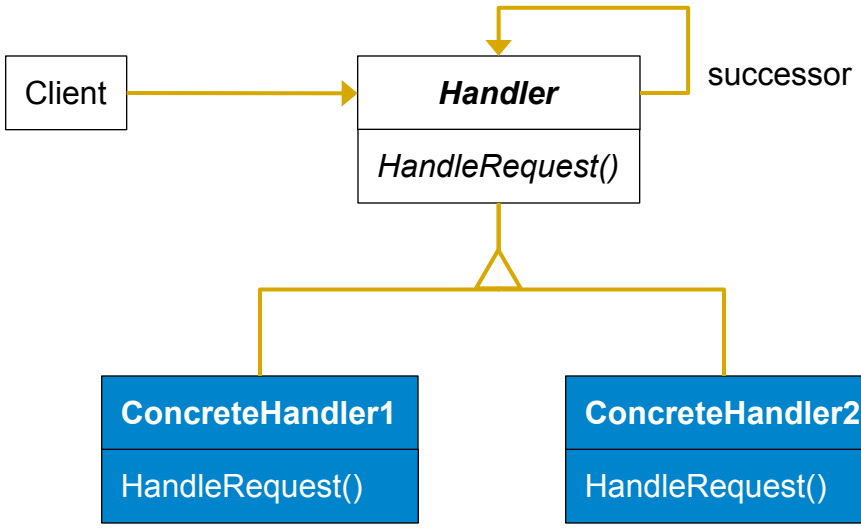
Handler

- Defines the interface for handling requests
- May implement the successor link



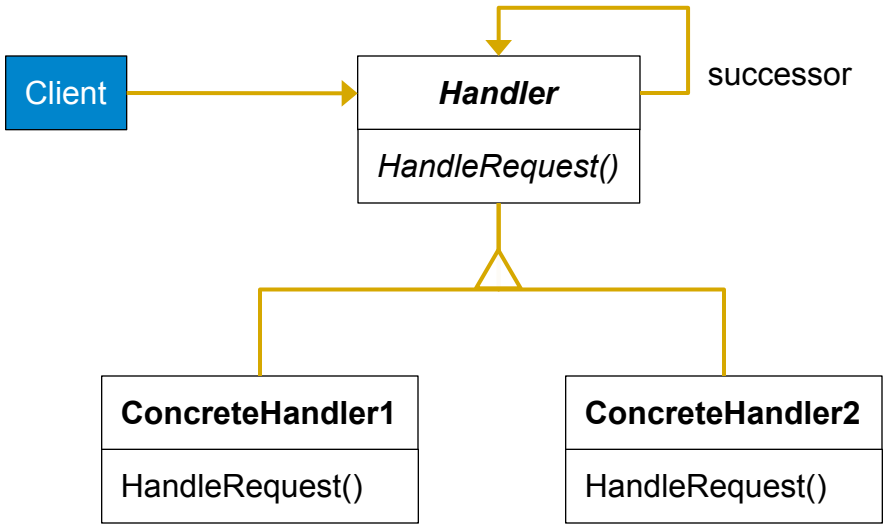
ConcreteHandler

- Either handles the request it is responsible for …
  - If possible
- … or otherwise it forwards the request to its successor



ConcreteHandler

- Initiates the request to a ConcreteHandler object in the chain



Applicability

- More than one object may handle a request
  - Handler isn't known a priori; *implicit* receiver
- Send a request to several objects without specifying the receiver
- Set of objects that can handle the request should be specified dynamically

Example – Coin Handler

```
public class Coin {
  private double weight;
  private double diameter;

  public Coin(double w, double d) {
    weight = w;
    diameter = d;
  }

  public double getWeight() {
    return weight;
  }

  public double getDiameter() {
    return diameter;
  }
}
```

CoinHandlerBase

```
public abstract class CoinHandlerBase {
  protected CoinHandlerBase _successor;

  public abstract void HandleCoin(Coin coin);

  public void SetSuccessor(CoinHandlerBase successor) {
    _successor = successor;
  }
}
```

CoinHandlerBase

```
public class FivePenceHandler extends CoinHandlerBase {

  public void HandleCoin(Coin coin) {
    if (Math.abs(coin.getWeight() - 3.25) < 0.02
```

```
public class TenPenceHandler extends CoinHandlerBase {

  public void HandleCoin(Coin coin) {
    if (Math.abs(coin.getWeight() - 6.5) < 0.03
```

```
public class TwentyPenceHandler extends CoinHandlerBase {

  public void HandleCoin(Coin coin) {
    if (Math.abs(coin.getWeight() - 5) < 0.01
```

```
public class FiftyPenceHandler extends CoinHandlerBase {

  public void HandleCoin(Coin coin) {
    if (Math.abs(coin.getWeight() - 8) < 0.02
```

```
public class OnePoundHandler extends CoinHandlerBase {

  public void HandleCoin(Coin coin) {
    if (Math.abs(coin.getWeight() - 9.5) < 0.02
        && Math.abs(coin.getDiameter() - 22.5) < 0.13) {
      System.out.println("Captured £1");
    } else if (_successor != null) {
      _successor.HandleCoin(coin);
    }
  }
}
```

TestChain

```
public class TestChain {

  public static void main(String[] args) {
    CoinHandlerBase h5 = new FivePenceHandler();
    CoinHandlerBase h10 = new TenPenceHandler();
    CoinHandlerBase h20 = new TwentyPenceHandler();
    CoinHandlerBase h50 = new FiftyPenceHandler();
    CoinHandlerBase h100 = new OnePoundHandler();

    h5.SetSuccessor(h10);
    h10.SetSuccessor(h20);
    h20.SetSuccessor(h50);
    h50.SetSuccessor(h100);

    Coin tenPence = new Coin(6.5, 24.49);
    Coin fiftyPence = new Coin(8.01, 27.31);
    Coin counterfeitPound = new Coin(9, 22.5);

    h5.HandleCoin(tenPence);
    h5.HandleCoin(fiftyPence);
    h5.HandleCoin(counterfeitPound);
  }
}
```

Output:

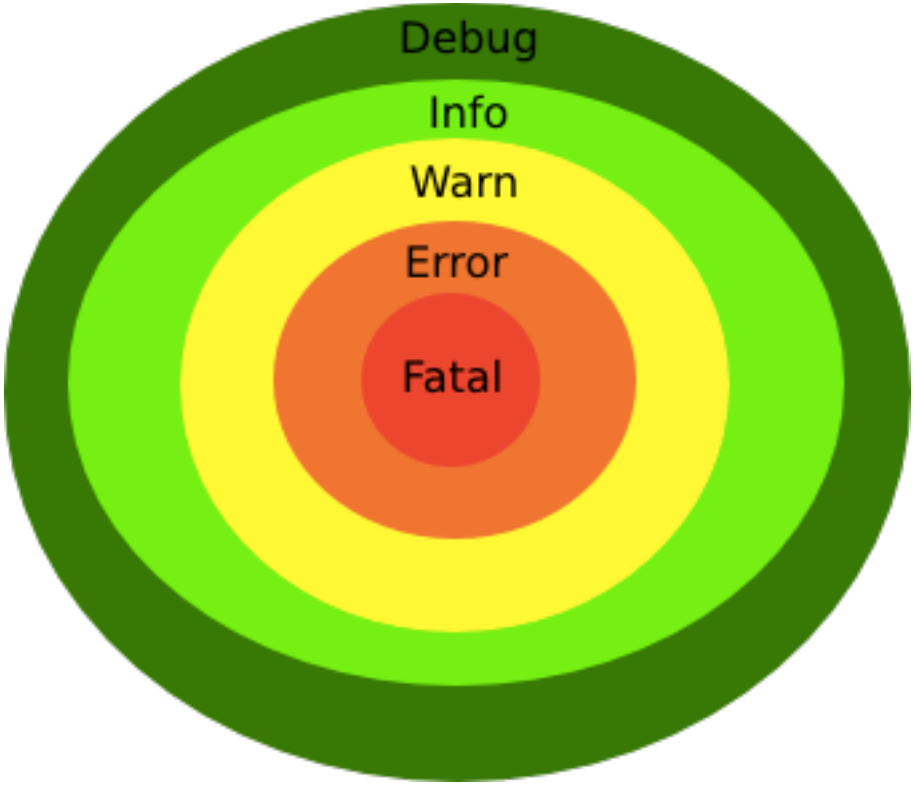
```
Captured 10p
Captured 50p
```

Example – Try-Catch

```
} catch (FileNotFoundException e) {
    logger.log(Level.SEVERE, "", e);
    throw e;
} catch (SQLException e) {
    logger.log(Level.SEVERE, "", e);
    throw e;
} catch (IOException e) {
    logger.log(Level.SEVERE, "", e);
    throw e;
}
```

Example - Logging

- Output logs to different targets based on the level



Consequences

- + Reduced Coupling
  - Frees client (sender) from knowing who will handle its request
  - Sender and receiver don't know each other
- + Flexibility in assigning responsibilities to objects
  - Responsibilities can be added or changed
- × Requests can go unhandled
  - Chain may be configured improperly