Design Patterns & Software Architecture

Chain of Responsbility

dr. Joost Schalken-Pinkster

University of Applied Science Utrecht

The Netherlands

Session overview

- Chain of Responsbility

Chain of Responsbility design pattern

Chain of Responsbility design pattern: Let’s start with an example

You have been asked to make a financial transaction system that needs approval for a financial transaction.

- a Financial Transaction has date and an amount and a source and target Account
- Each Transaction needs approval of the source Account’s bank manager
  - a Coordinator may approve transactions up to € 5.000
  - a Manager may approve transactions up to € 10.000
  - a President may approve all transactions

- Additional requirement:
  A new role has been created at the bank: Vice President
  - a Vice Presidents may approve transactions up to € 100.000

Let’s find a design pattern

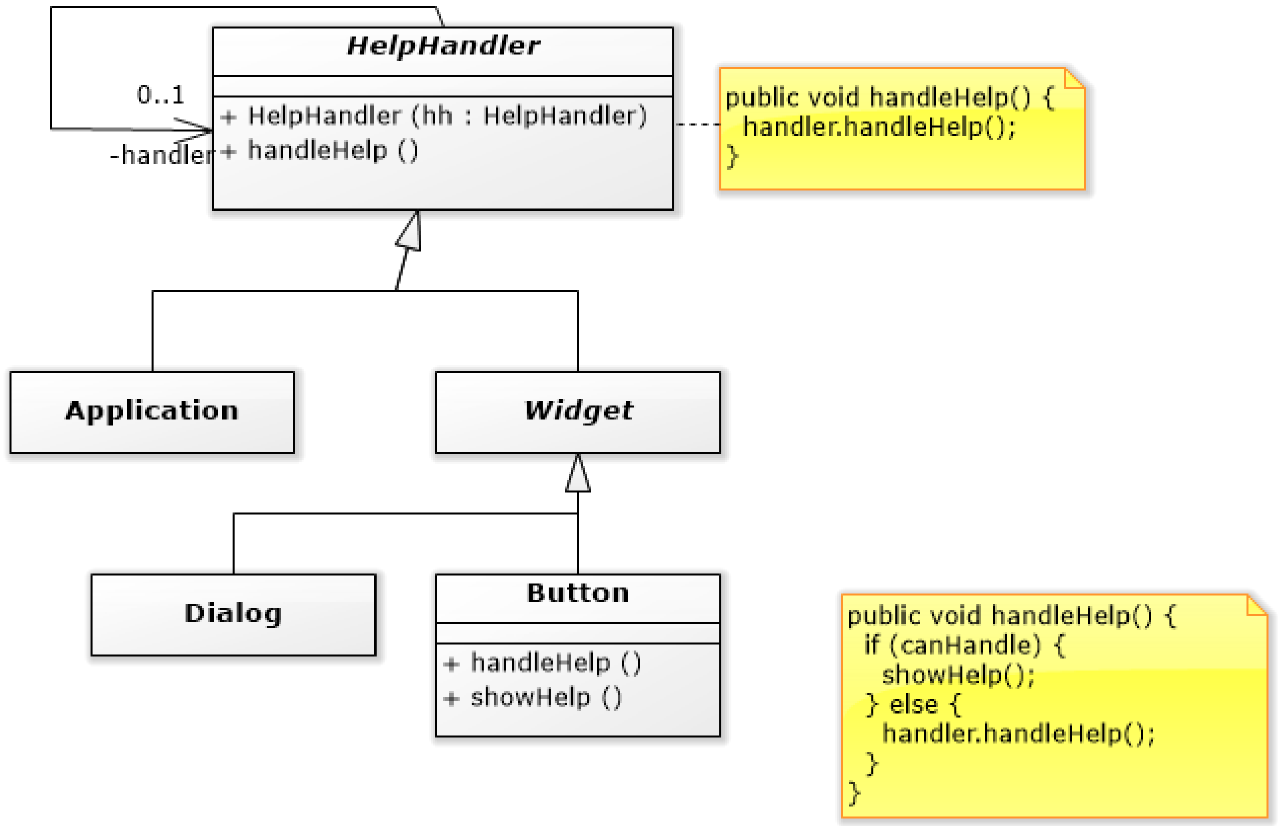
Will now present, on the board,
and using Eclipse,
a solution that utilises
the chain of responsbility design pattern…

Chain of Responsibility pattern definition

- **Intent**: Avoid coupling sender of request to its receiver by giving more than one object chance to handle request. Chain receiving objects and pass request along until an object handles it.

- **Motivation**: a context-sensitive help facility for a graphical user interface. The user can obtain help information on any part of the interface just by clicking on it. The help that's provided depends on the part of the interface that's selected and its context.

Motivation (cont.)

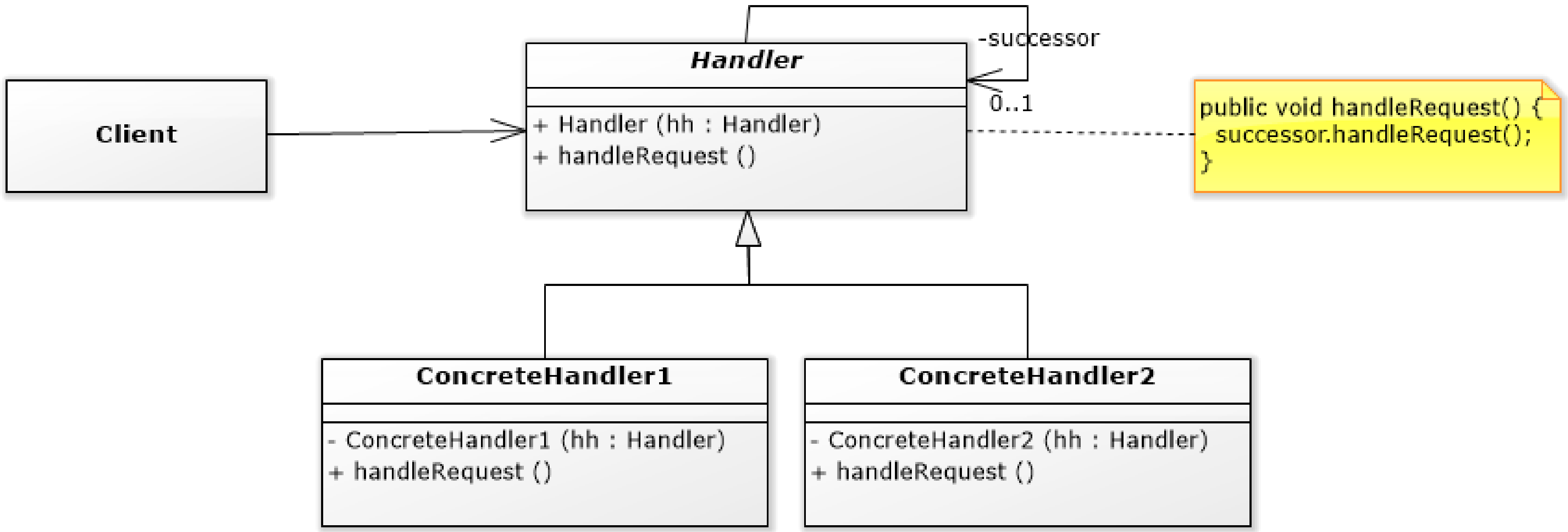


```
public void handleHelp() {
  handler.handleHelp();
}
```

```
public void handleHelp() {
  if (canHandle) {
    showHelp();
  } else {
    handler.handleHelp();
  }
}
```

- **Applicability**:
  Use the Chain of Responsibility pattern when:
  - more than one object may handle a request, and the handler isn't known *a priori*. The handler should be determined 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.

▪ Structure:



```
public void handleRequest() {
  successor.handleRequest();
}
```

- **Participants:**
  - **Handler** (HelpHandler)
    - defines an interface for handling requests.
    - (optional) implements the successor link.
  - **ConcreteHandler** (PrintButton, PrintDialog)
    - handles requests it is responsible for.
    - can access its successor.
    - 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.

- Consequences: The Chain of Responsibility pattern leads to:
  - *Reduced coupling.*
    The pattern frees an object from knowing which other object handles a request. An object only has to know that a request will be handled "appropriately." Both the receiver and the sender have no explicit knowledge of each other, and an object in the chain doesn't have to know about the chain's structure.
  - *Added flexibility in assigning responsibilities to objects.*
    Chain of Responsibility gives you added flexibility in distributing responsibilities among objects. You can add or change responsibilities for handling a request by adding to or otherwise changing the chain at run-time.
  - *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.

- **Implementation:**
  Some issues implementing a Chain of Responsibility:
  - *Implementing the successor chain.*
    There are two possible ways to implement the successor chain:
    - Define new links (usually in the Handler)
    - Use existing links
  - *Connecting successors.*
    If there are no preexisting references for defining a chain, then you'll have to introduce them yourself.
  - *Representing requests*
    - Object operations (methods)
    - Simple parameters to an operation
    - Request objects

Reading

For tomorrow please read:

- Chapter 9 (Well managed collections) of Head First Design Patterns.
- Chapter 13 (Patterns in the Real World) of Head First Design Patterns
- Chapter Chain of Responsibility of Design Patterns: Elements of Reusable Object-Oriented Software, pp 251-262.