# Chain of Responsibility Pattern


Bryan Hansen

twitter: bh5k | http://www.linkedin.com/in/hansenbryan

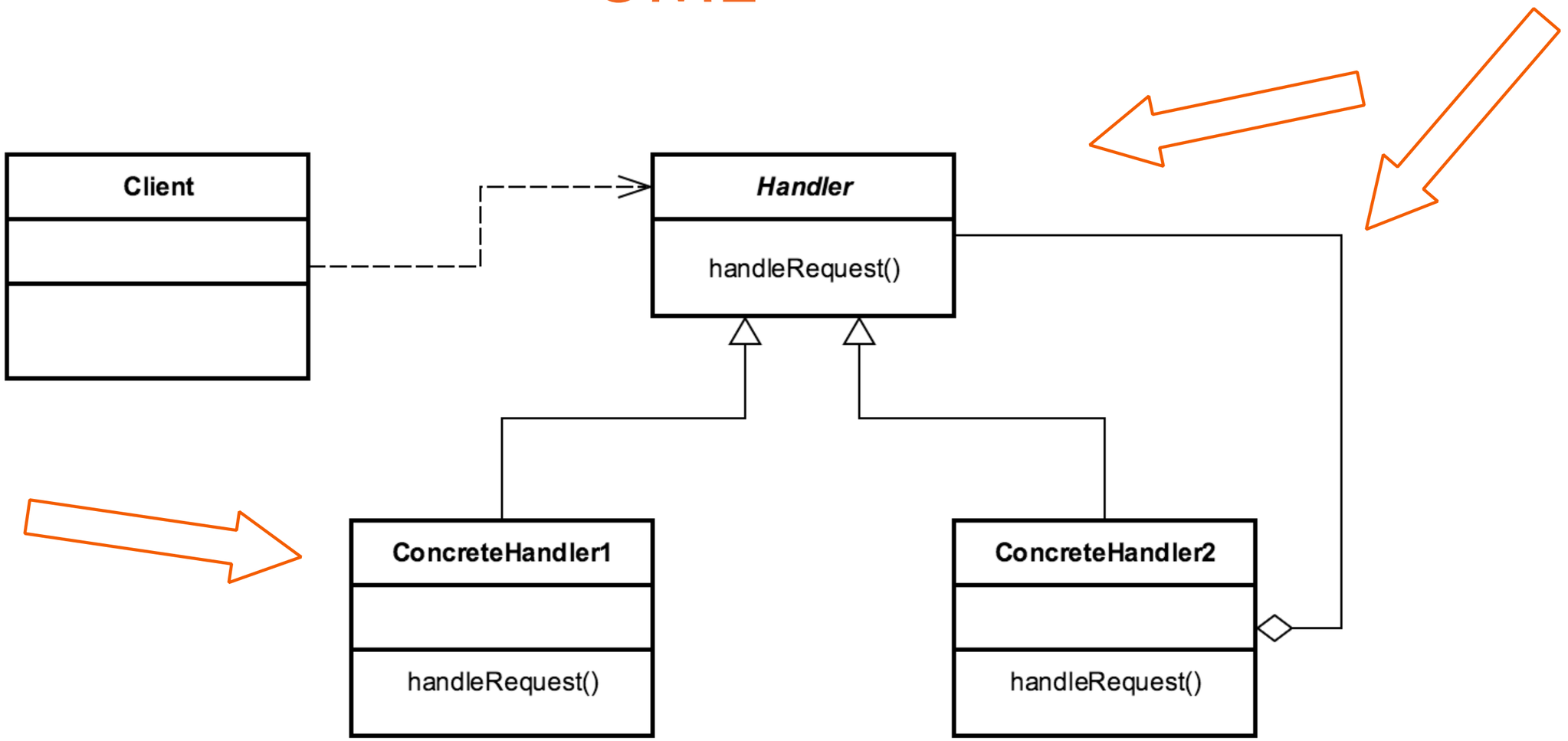
# Concepts

- Decoupling of sender and receiver
- Receiver contains reference to next receiver
- Promotes loose coupling
- No Handler - OK
- Examples:
  - java.util.logging.Logger#log()
  - javax.servlet.Filter#doFilter()
  - Spring Security Filter Chain

# Design

- Chain of receiver objects
- Handler is Interface based
- ConcreteHandler for each implementation
- Each Handler has a reference to the next
- Handler, ConcreteHandler

# UML


Client
Handler
handleRequest()
ConcreteHandler1
handleRequest()
ConcreteHandler2
handleRequest()

# Everyday Example - Logging

```
//level to log at
logger.setLevel(Level.FINER);

ConsoleHandler handler = new ConsoleHandler();
//level to publish at
handler.setLevel(Level.FINER);
logger.addHandler(handler);

logger.finest("Finest level of logging"); //this one won't print
logger.finer("Finer level, but not as fine as finest");
logger.fine("Fine, but not as fine as finer or finest");
```

# Exercise Chain of Responsibility

Handler, Successor, Request

Create Handler

Chain

# Pitfalls

- Handling/Handler guarantee
- Runtime configuration risk
- Chain length/performance issues

# Contrast

## Chain of Responsibility

- Handler is unique
- Successor
- Can utilize the Command

## Command

- Command also unique
- Encapsulates function
- Reversible or Trackable in nature

# Chain of Responsibility Summary

- Decouples sender and receiver
- Runtime configuration
- Hierarchical in nature
- Careful with large chains