Padrões de projeto comportamentais

Prof. Hugo de Paula

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Curso de Engenharia de Software

Sumário

Padrões de projeto comportamentais

Padrões Comportamentais

Organizam a comunicação entre objetos de forma clara e eficiente.

- Exemplos mais comuns: Chain of Responsibility, Observer (Model-View-Controller), iterator, visitor, template e strategy.
- Exemplos menos comuns: Interpreter e Mediator.

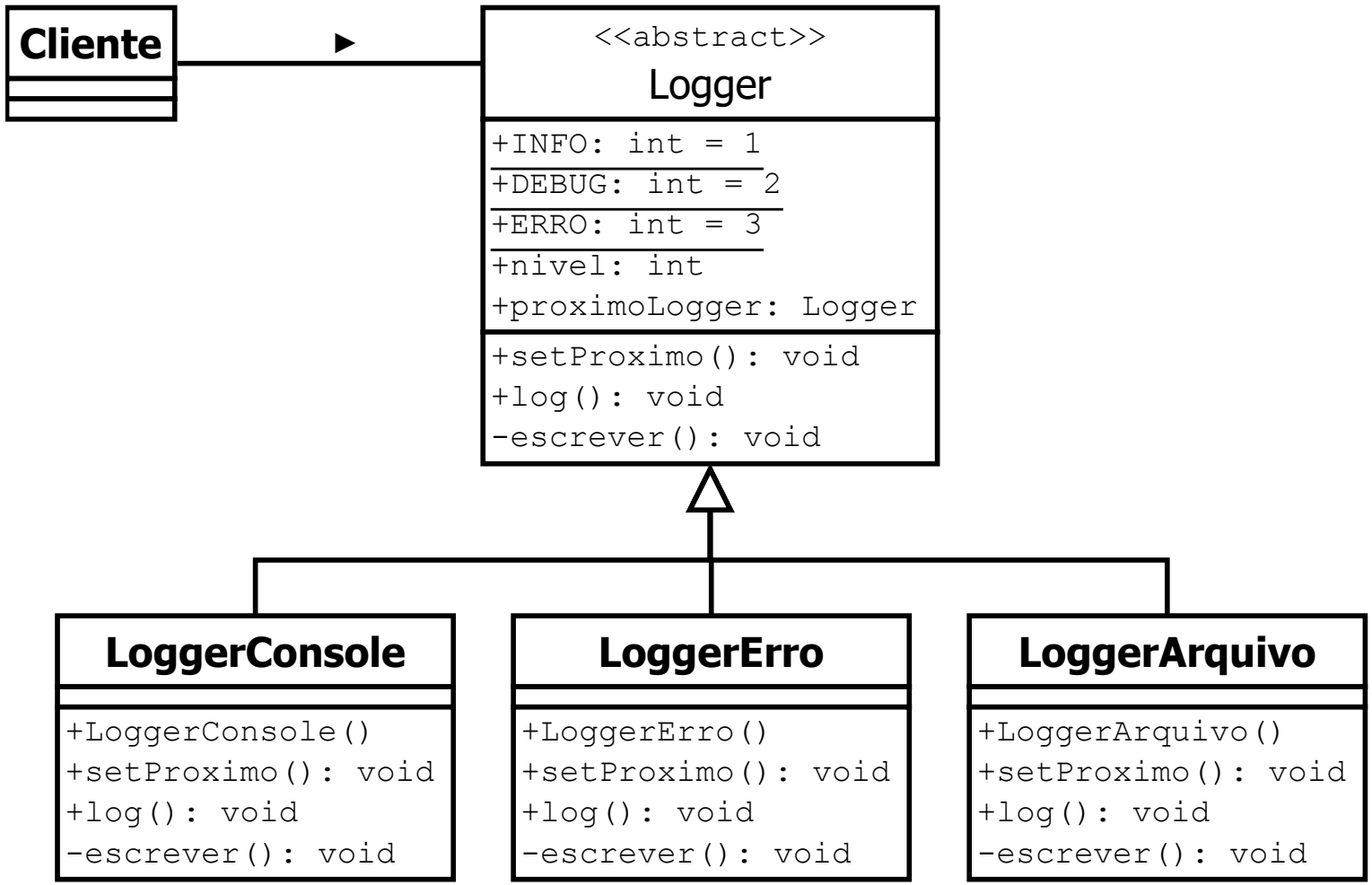
Chain of Responsibility

Chain of Responsibility

Descreve uma forma de organizar os objetos em cadeia (como na forma de uma corrente).

- Útil quando alguns comandos devem ser tratados por diferentes objetos, cada um deles passando o comando ao próximo objeto na forma de uma corrente.
- Cada receptor de uma requisição contém uma referência a outro receptor.

Chain of Responsibility: exemplo Logger



Chain of Responsibility: exemplo Logger

```
public abstract class Logger {
  public static int INFO = 1;
  public static int DEBUG = 2;
  public static int ERRO = 3;
  protected int nivel;
  // proximo elemento na cadeia de responsabilidades
  protected Logger proximoLogger;

  public void setProximo(Logger proximoLogger) {
    this.proximoLogger = proximoLogger;
  }

  public void log(int nivel, String mensagem) {
    if (this.nivel <= nivel) {
      escrever(mensagem);
    }
    if (proximoLogger != null) {
      proximoLogger.log(nivel, mensagem);
    }
  }
  abstract protected void escrever(String mensagem);
}
```

Chain of Responsibility: exemplo Logger

```
public class LoggerConsole extends Logger {
  public LoggerConsole(int nivel) {
    this.nivel = nivel;
  }
  @Override
  protected void escrever(String mensagem) {
    System.out.println("Console Padrao::Logger: " + mensagem);
  }
}

public class LoggerErro extends Logger {
  public LoggerErro(int nivel) {
    this.nivel = nivel;
  }
  @Override
  protected void escrever(String mensagem) {
    System.out.println("Console de Error::Logger: " + mensagem);
  }
}

public class LoggerArquivo extends Logger {
  public LoggerArquivo(int nivel) {
    this.nivel = nivel;
  }
  @Override
  protected void escrever(String mensagem) {
    System.out.println("Arquivo::Logger: " + mensagem);
  }
}
```

Chain of Responsibility: exemplo Logger

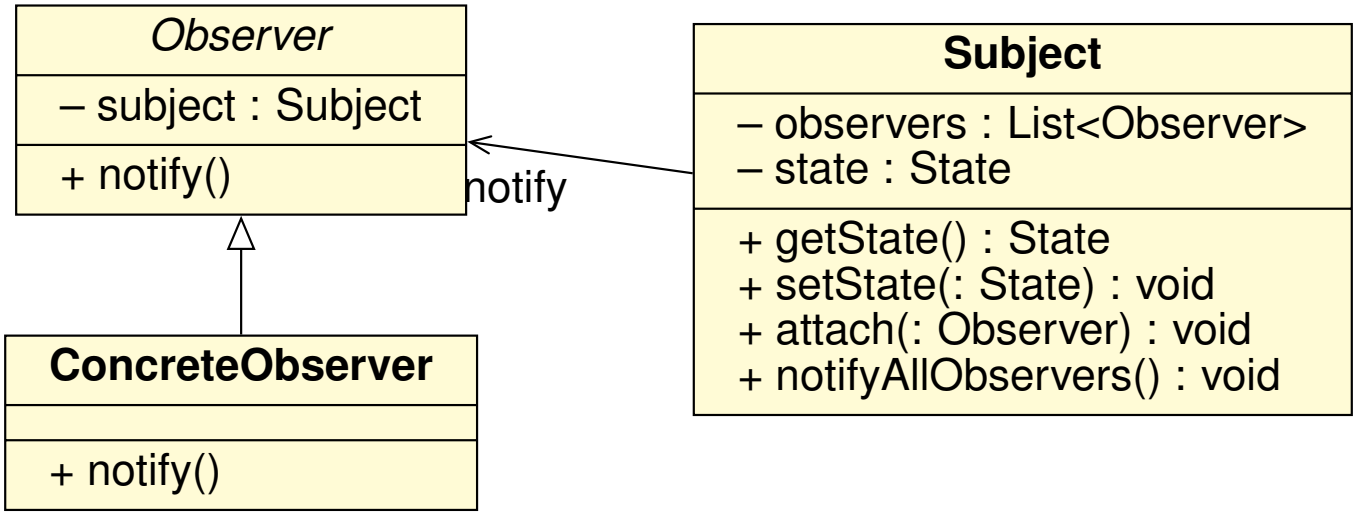
```
public class Cliente {

  private static Logger getCadeiaDeLoggers() {
    Logger errorLogger = new LoggerErro(Logger.ERRO);
    Logger fileLogger = new LoggerArquivo(Logger.DEBUG);
    Logger consoleLogger = new LoggerConsole(Logger.INFO);
    errorLogger.setProximo(fileLogger);
    fileLogger.setProximo(consoleLogger);
    return errorLogger;
  }

  public static void main(String[] args) {
    Logger loggerChain = getCadeiaDeLoggers();
    loggerChain.log(Logger.INFO, "Esta é uma informação de evento.");
    loggerChain.log(Logger.DEBUG, "Esta é uma informação de Debug.");
    loggerChain.log(Logger.ERRO, "Esta é uma mensagem de erro.");
  }
}
```

Observer

Observer

Relação de um-para-muitos entre objetos de tal forma que quando um objeto é modificado os objetos dependentes são notificados automaticamente.

Observer: solução geral



- **Subject**: permite a conexão de observadores e possui o estado que será observado.
- **Observer**: se conecta a um Subject e declara o método que é executado no momento da notificação.
- **ConcreteObserver**: implementa o método de notificação.

Exemplo: Observer