Centro Universitário UniBH
Ciência da Computação
Práticas de Programação
Professor: Lucas Schmidt

Aula Prática - 13 Padrões de Comportamento - Chain of Responsibility

Intenção

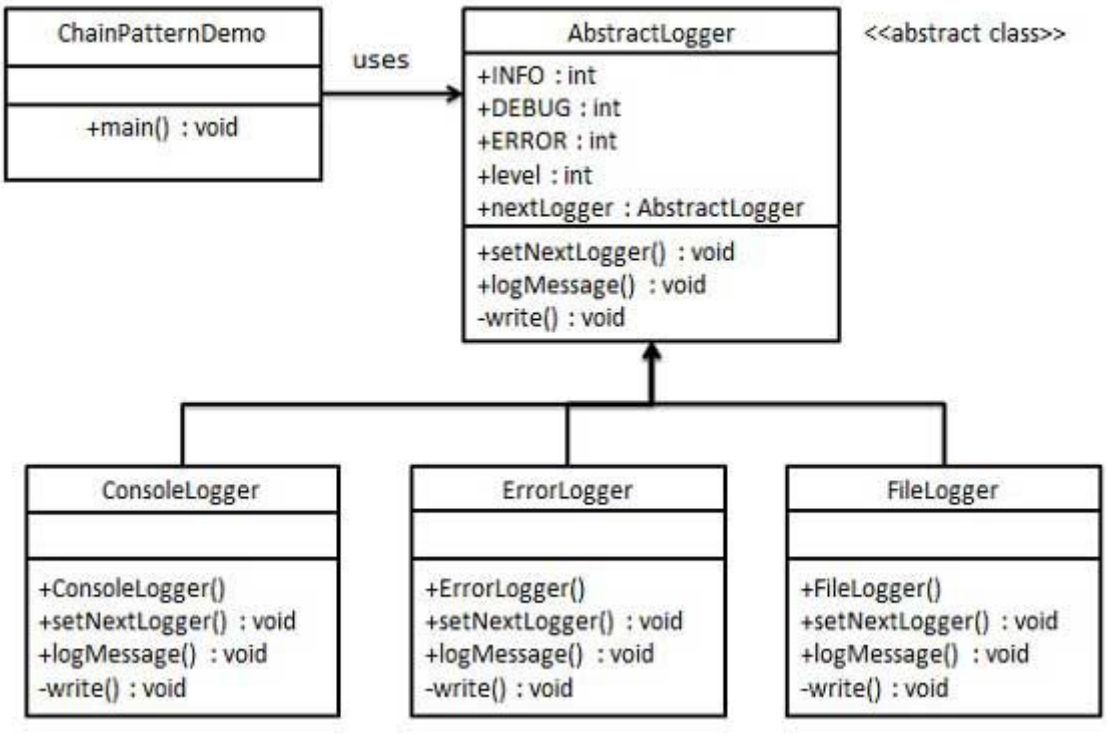
Formar uma cadeia de objetos receptores e passar uma requisição pela mesma, dando a chance a mais de um objeto a responder a requisição ou colaborar de alguma forma na resposta.

Usar este padrão quando…

- Mais de um objeto pode responder a uma requisição e:
  - não se sabe qual a priori;
  - não se quer especificar o receptor explicitamente;
  - estes objetos são especificados dinamicamente.

Vantagens e desvantagens

- Acoplamento reduzido:
  - Não se sabe a classe ou estrutura interna dos participantes. Pode usar Mediator para desacoplar ainda mais.
- Delegação de responsabilidade:
  - Flexível, em tempo de execução.
- Garantia de resposta:
  - Deve ser uma preocupação do desenvolvedor!
- Não utilizar identidade de objetos.



Passo 1
Crie uma classe abstrata de agente de log.

AbstractLogger.java

```
public abstract class AbstractLogger {
   public static int INFO = 1;
   public static int DEBUG = 2;
   public static int ERROR = 3;

   protected int level;

   //next element in chain or responsibility
   protected AbstractLogger nextLogger;

   public void setNextLogger(AbstractLogger nextLogger){
      this.nextLogger = nextLogger;
   }

   public void logMessage(int level, String message){
      if(this.level <= level){
         write(message);
      }
      if(nextLogger !=null){
         nextLogger.logMessage(level, message);
      }
   }

   abstract protected void write(String message);

}
```

Passo 2
Crie classes concretas estendendo o logger.

ConsoleLogger.java

```
public class ConsoleLogger extends AbstractLogger {

   public ConsoleLogger(int level){
      this.level = level;
   }

   @Override
   protected void write(String message) {
      System.out.println("Standard Console::Logger: " + message);
   }
}
```

ErrorLogger.java

```
public class ErrorLogger extends AbstractLogger {

   public ErrorLogger(int level){
      this.level = level;
   }

   @Override
   protected void write(String message) {
      System.out.println("Error Console::Logger: " + message);
   }
}
```

FileLogger.java

```
public class FileLogger extends AbstractLogger {

   public FileLogger(int level){
      this.level = level;
   }

   @Override
   protected void write(String message) {
      System.out.println("File::Logger: " + message);
   }
}
```

Passo 3
Crie diferentes tipos de registradores. Atribua os níveis de erro e defina o próximo registrador em cada registrador. O próximo registrador em cada registrador representa a parte da cadeia.

ChainPatternDemo.java

```
public class ChainPatternDemo {

   private static AbstractLogger getChainOfLoggers(){

      AbstractLogger errorLogger = new ErrorLogger(AbstractLogger.ERROR);
      AbstractLogger fileLogger = new FileLogger(AbstractLogger.DEBUG);
      AbstractLogger consoleLogger = new ConsoleLogger(AbstractLogger.INFO);

      errorLogger.setNextLogger(fileLogger);
      fileLogger.setNextLogger(consoleLogger);

      return errorLogger;
   }

   public static void main(String[] args) {
      AbstractLogger loggerChain = getChainOfLoggers();

      loggerChain.logMessage(AbstractLogger.INFO,
         "This is an information.");

      loggerChain.logMessage(AbstractLogger.DEBUG,
         "This is an debug level information.");

      loggerChain.logMessage(AbstractLogger.ERROR,
         "This is an error information.");
   }
}
```

Passo 4

Teste sua implementação!