آموزش الگوهای طراحی

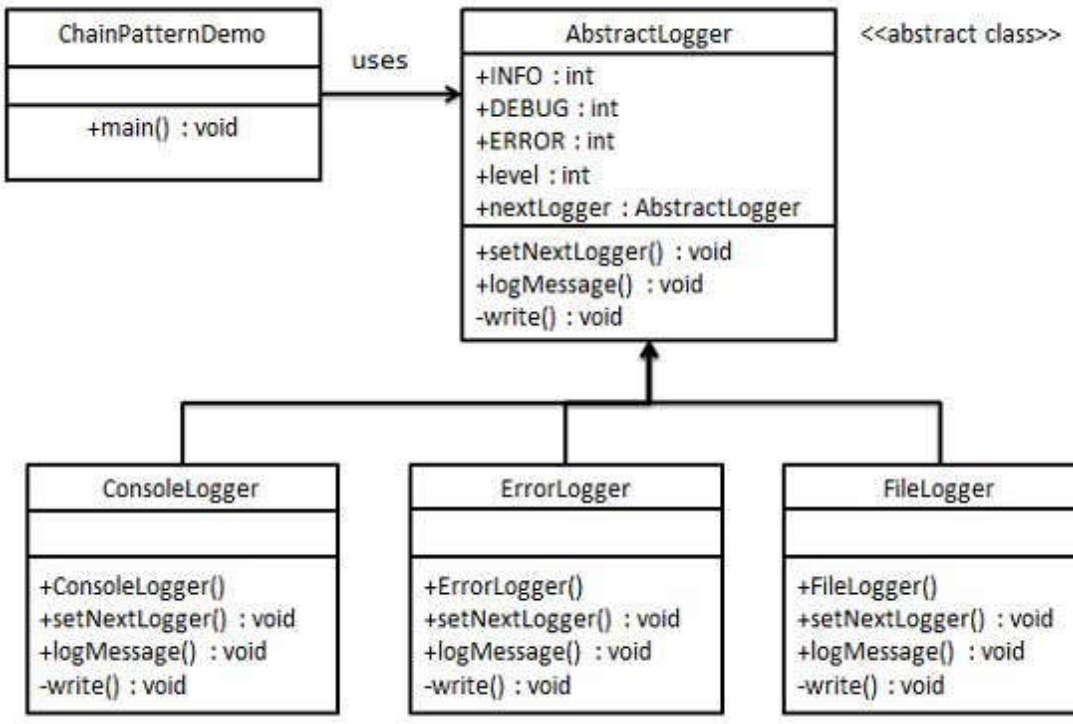
Design Pattern - الگوی Chain of Responsibility

همانطور که از نام الگو پیداست، الگوی chain of responsibility، زنجیره ای از اشیاءِ دریافت شده را برای یک درخواست، ایجاد می کنه. این الگو، بر اساس نوع درخواست، فرستنده و گیرنده ی درخواست را جدا می کنه. این نوع از الگوی طراحی زیرگروه الگوی ساختاری (structural) محسوب میشه.

در این الگو، بطور عادی هر دریافت کننده دارای ارجاعی به دریافت کننده ی دیگه ست. اگه یک شیء نتواند درخواست را اجرا کنه، درخواست را به دریافت کننده ی بعدی میده و همینطور الی آخر.

پیاده سازی

یک کلاسِ انتزاعی (abstract) به نام AbstractLogger با سطحِ ورود (logging)، ایجاد کرده ایم. سپس سه نوع وارد شونده (lgger) برای توسعه ی AbstractLoger ایجاد می کنیم. هر وارد شونده (logger)، سطح پیام را با سطحِ خودش مقایسه می کنه و نتیجه را چاپ می کنه (برمی گردانه)، در غیر اینصورت اگه چاپ نکنه (خروجی را برنگردانه)، پیام را به واردشونده ی (logger) بعدی پاس میده.



مرحله ی 1

ایجاد یک کلاسِ logger انتزاعی (abstract).

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);

}
```

مرحله ی 2

ایجاد کلاس های concreteای که logger را توسعه می دن.

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);
...}
}
```

FileLggor.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);
...}
}
```

مرحله ی 3

ایجاد انواع مختلفی از loggerها. تخصیص دادن سطوح خطا به آن ها و تنظیمِ logger بعدی در هر logger. logger بعدی نشان دهنده ی قسمتی از زنجیره ست.

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.");
...}
}
```

مرحله ی 4

بررسی خروجی

```
Standard.Console::Logger:.This.is.an.information.
File::Logger:.This.is.an.debug.level.information.
Standard.Console::Logger:.This.is.an.debug.level.information.
Error.Console::Logger:.This.is.an.error.information.
File::Logger:.This.is.an.error.information.
Standard.Console::Logger:.This.is.an.error.information.
```

در صورتی که سوال و یا نظری دارید، از بخش نظرات با ما در میان بگذارید.

برچسب ها: الگوهای طراحی جاوا الگوی Chain of Responsibility

الگوهای طراحی – الگوی Command درس بعدی درس قبلی الگوهای طراحی – الگوی Proxy