デザインパターン入門 レポート

平川 一樹

2018 年 3 月 30 日

概要

2018 年 3 月 30 日から作成開始。

デザインパターンの学習を目的にレポートを書いていく。3 日坊主にならないように気をつけよう^^

このレポートは「Java 言語で学ぶ　デザインパターン入門」（図 0.1 参考）を元に作っている。本を読んで理解した後、なるべく本を見ずに概要・例・私見を書いて理解を確認する形式で進めていこうと思う。

1 日の終わりには感想・疑問点をまとめておく。

図 0.1　参考文献

1　Iterator パターン

1.1　概要

あるコレクションの要素に便利にアクセスするためのデザインパターン。

イテレータクラスを作り、コレクションを持つクラスがイテレータクラスを返すようにする事で、後はイテレータクラスのみを用いてコレクションにアクセスできる。

イテレータクラスとイテレータを返すクラスはインターフェイスを用いて実装するべきメソッドを定めておく。

1.2　例

図 1.1 は Iterator パターンの一例だ。Aggregate、Iterator というインターフェイスを作り、それらを実装する事で Iterator パターンを実現している。

ConcreteIterator は ConcreteAggregate が持っているコレクションなどにアクセスする。Next はコレクションの要素を返して、参照を次に移動するメソッド、hasNext は次の参照を持っているかどうか[*1]を判断して返すメソッドとなっている。

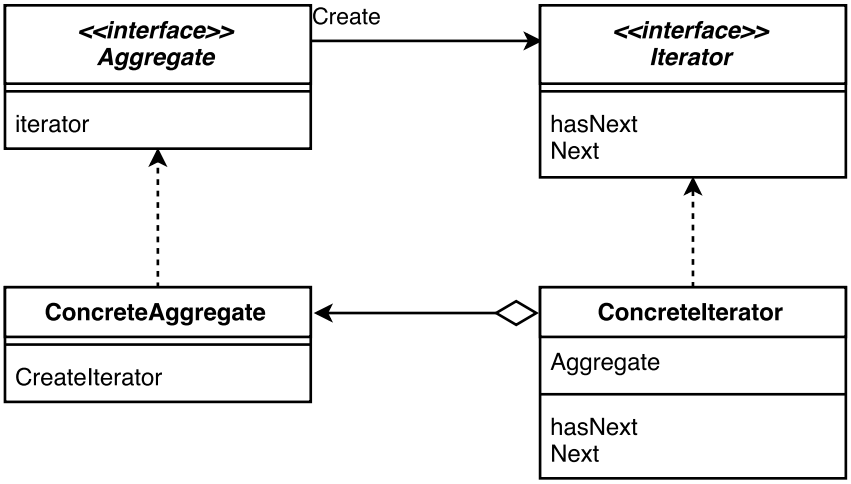

図 1.1　（例）Iterator パターン

1.3　私見

C++ の Iterator と同じようなものかと思ったが、それは今回のパターンとは少し実装の仕方が違っている気がした。特に C++ はインターフェイスを使っておらず、Iterator クラスの定義が、Aggregate クラスの中に書かれていた覚えがある。C#の IEnumerable と IEnumerator がこのパターンを使っているのだろうと感じた。

自分でこのパターンを使ったこともあったが、その時はインターフェイスを作らずに直接 Iterator クラスを作ってしまった。その結果、Iterator クラスの内容はかなり独特なものになってしまったと思う。インターフェイスを用いた方が、例でいう Concrete クラスの内容がどうであれインターフェイスの様式[*2]のみを知っていれば扱えるようになるのがかなりのメリットだと感じた。

2　2018 年 3 月 30 日時点のコメント

2.1　感想

レポートを書こうと思い立って 1 日目が終了。

TEX の環境を構築したり、クラス図を作るための手段を調べるのに時間がかかってしまい、今回は 1 つのデザインパターンしか書けなかったが今後はもう少し早いペースで書いていけると思う。

Iterator パターンも読み始める前はわかっているつもりだったが、読んでみるとなかなか有益だったと思う。

2.2　疑問点

現在のところ特になし。

[*1] コレクションの最後の参照かどうかとも言い換えられる

[*2] 今回の例では hasNext、next のメソッドを知っているだけで使える