Padrões de Projetos - Iterator

Adré Luis Velasques[1], Vinícius Marcondes Vieira [2]

[1]Bacharelado em Ciência da Computação – Universidade Tuiuti do Paraná (UTP)

andre.velasques@utp.edu.br, vinicius.vieira2@utp.edu.br

Abstract. *This meta-article discusses the design pattern known as Iterator. It will cover topics such as operation, diagrams, applications and implementation, it will also cover some of the history and creation of the pattern and how it can facilitate development when you need to deal with structured collections of data.*

Resumo. *Este meta-artigo discorre sobre o padrão de projeto conhecido como Iterator. Nele serão abordados tópicos como funcionamento, diagramas, aplicações e implementação, também será abordado parte da historia e criação do padrão como ele pode facilitar o desenvolvimento quando se tem necessidade de lidar com coleções estruturadas de dados.*

1. Padrões de projeto

Conforme a áre de desenvolvimento de software foi avançando ao longo fos anos diversos gupos de desenvolvedores foram adotando certos padões que os melor atendia, eventualmente esses padrões foram categorizados, formalizados e padronizados para que qualquer projeto que os aplicasse utilizasse das mesmas ferramentas e metodologias que os demais projetos que aplicasssem o mesmo padrâo.

Os primeiros padrões de projeto foram apresentados por volta de 1987, desde então diversos padrões foram modificados, excluidos ou criados. Dentro os padrões mais conhecidos atualmente estão: Factory Method, Abstract Factory, Builder, Adapter, Bridge, Composite, Chain of Responsibility, Command, Interpreter.

2. Padrão: Iterator

2.1. Problemática e funcionamento

Iterator é uma padrão de projeto do tipo comportamental, ele tem como principal função permitir que se percorra um conjunto de objetos de forma sequencial e sem expor sua natureza (lista, pilha, árvore, etc) ou lógica de implmentação.

O principal motivo para o Iterator ser desenvolvido foi por conta da alta complexibilidade que ação de percorrer as coleções de objetos pode atingir. Por termos diversos tipos de coleção, temos também diversas organizações, uma lista por exemplo é facilmente percorrida ao utilizar um array e o percorrer de forma sequencial, porém ao se percorrer uma árvore deve-se atentar ao tipo de arvore e ao fato que os elementos nao serão sequenciais e sim obedecer a regra de balanceamento da arvore, o que pode ser uma tarefa trabalhosa.

Ao se deparar com esse problema foi entao pensado uma forma de simplificar a ação de percorrer essas coleções de objetos e concentrar os algoritmos responsáveis por isso

atras de uma interface que fosse de fácil utilização para o desenvolvedor que precisar utiliza-la.

O objeto que implementa a interface do Iterator possui em si todas as informações necessárias para se percorrer a coleção, dentre elas: objeto inicial, objeto final, posição atual, elementos restantes. Possui também dois métodos principais, um para avançar para o próximo objeto (*next*()) e um para verificar se existe um próximo objeto (*hasNext*()).

2.2. Diagramas

O diagrama se baseia na interface Iterator (a princial) que é utilizada para declarar um objeto contendo as principais operações necessárias para percorrer a coleção. A classe *ConcreteIterator* tem objetivo de implementar os algoritmos necessários para percorrer as coleções e possui as váriaveis de controle para que se possa atraveçar a coleção de forma eficiente. A interface *IterableCollection* serve para declarar os métodos espeficios necessários para lidar com o tipo de coleção desejada. Por fim temos a classe *ConcreteCollection* implementa os métodos especificos para as coleções e intancia objetos *iterator* quando necessário.

A classe cliente demostra como o cliente não está acoplado ao funcionamento direto do iterator, assim ele pode lidar tanto com a coelção quanto com o iterator diretamente. Comumente iterators não são instanciados diretamente, ao invés disso são acessados através da instância da coleção.

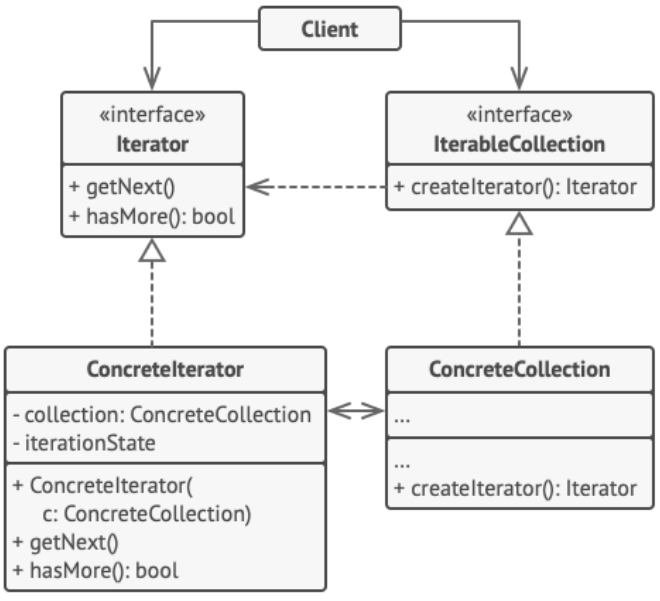

Figura 1. Diagrama de classe da implementação de um Iterador

2.2. Implementação em C++

Podemos ter como exemplo da implementação do Iterator em C++ da seguinte forma:

```cpp
template <typename T, typename U>
class Iterator {
 public:
  typedef typename std::vector<T>::iterator iter_type;
  Iterator(U *p_data, bool reverse = false) : m_p_data_(p_data) {
    m_it_ = m_p_data_->m_data_.begin();
  }

  void First() {
    m_it_ = m_p_data_->m_data_.begin();
  }

  void Next() {
    m_it_++;
  }

  bool IsDone() {
    return (m_it_ == m_p_data_->m_data_.end());
  }

  iter_type Current() {
    return m_it_;''
  }

 private:
  U *m_p_data_;
  iter_type m_it_;
};
```

Figura 2. Implementação da classe de um Iterador

```cpp
template <class T>
class Container {
  friend class Iterator<T, Container>;

 public:
  void Add(T a) {
    m_data_.push_back(a);
  }

  Iterator<T, Container> *CreateIterator() {
    return new Iterator<T, Container>(this);
  }

 private:
  std::vector<T> m_data_;
};

class Data {
 public:
  Data(int a = 0) : m_data_(a) {}

  void set_data(int a) {
    m_data_ = a;
  }

  int data() {
    return m_data_;
  }

 private:
  int m_data_;
};
```

Figura 3. Implementação de uma classe de coleção com suporte a iterator

Referencias

Refactoring Guru. “Iterator”, https://refactoring.guru/pt-br/design-patterns/iterator, Junho de 2022.

Chagas, Igor. “Design patterns: Breve introdução aos padrões de projeto”. https://www.alura.com.br/artigos/design-patterns-introducao-padroes-projeto, Junho de 2022

Davi. (2013) “Padrão de Projeto Iterator em Java - Conceitos, Funcionamento e Aplicação prática”. https://www.devmedia.com.br/padrao-de-projeto-iterator-em-java-conceitos-funcionamento-e-aplicacao-pratica/28748#, Junho de 2022