IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

Programação Avançada

Introdução aos Padrões de Software
Iterator Pattern

Programação Avançada – 2020-21

Bruno Silva, Patrícia Macedo

Sumário 

- Padrões de Desenho versus Padrões de Arquitetura
- Definição de Padrão de Desenho
- O Padrão Iterator

Introdução

Em Engenharia de Software, um padrão é **uma solução geral** para um problema que ocorre com frequência dentro de um determinado contexto do desenho de software.

- Existem varias classificações para os padrões, na uc de Programação Avançada vamos usar a seguinte classificação:
  - Padrões de Arquitetura (Architectural Patterns)
  - Padrões de Desenho (Design Patterns)

Padrões de Software

- Padrões de Arquitetura (Architectural Patterns)

  Resolvem um problema típico da arquitetura de software. Definem formas de organizar os vários componentes de um sistema de software
  MVC,MVP, DAO

- Padrões de Desenho (Design Patterns)
  - Resolvem problemas específicos e localizados (como criar uma variável global, como adaptar o comportamento de uma classe, como criar uma família de classes etc...)

Vantagens da Utilização de Padrões

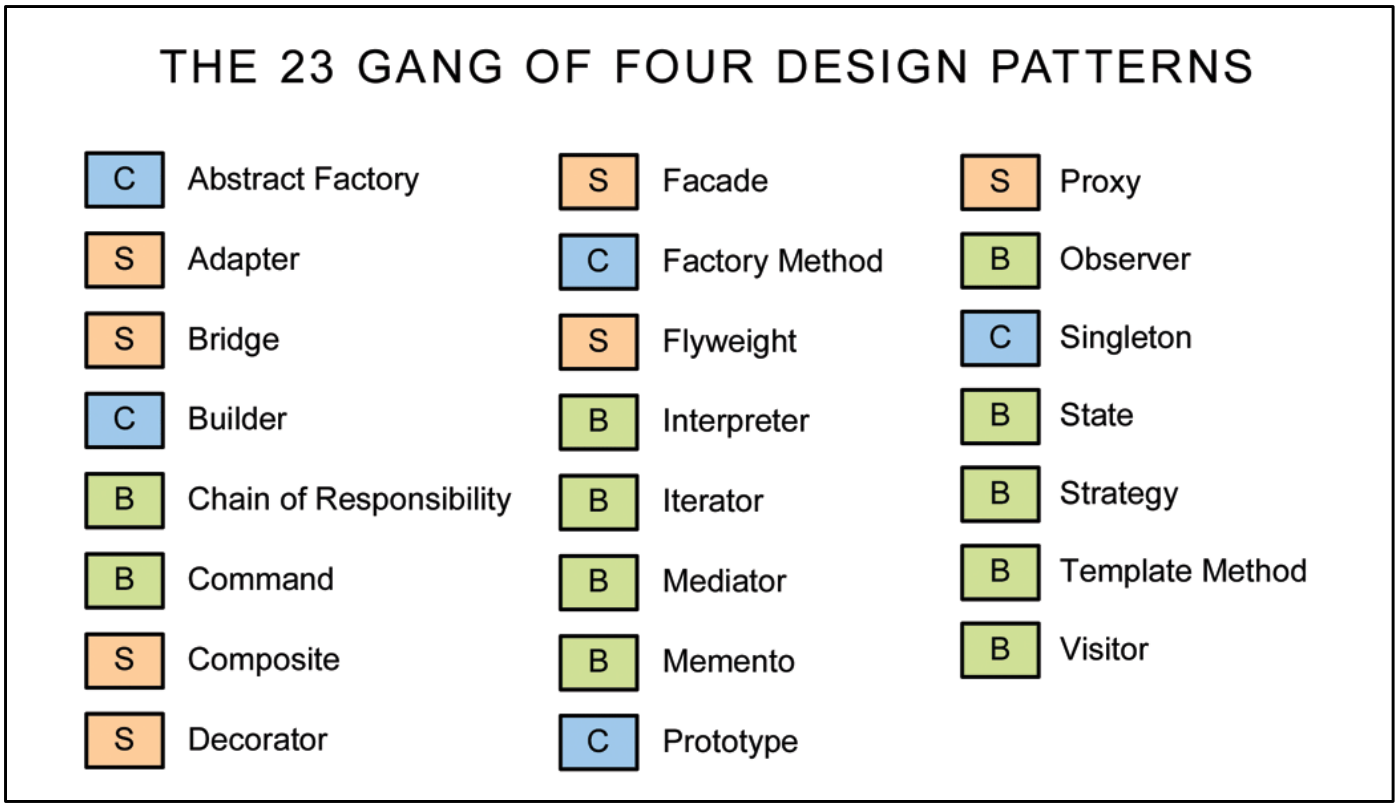
- Aprende-se com a experiência de outros;
- Utiliza-se soluções amplamente testadas;
- Permite utilizar uma linguagem comum entre os *designers* e programadores. Melhora-se assim a comunicação entre a equipa e a documentação dos sistemas;
- Leva ao uso de boas práticas no desenvolvimento de software orientado a objetos, obtendo-se assim software de melhor qualidade.

Especificação de um Padrão de Desenho

- Um padrão de desenho deverá ser especificado indicando:
  - O nome (cria um vocabulário de padrões)
  - O problema (diz quando aplicar o padrão)
  - A solução (descreve os elementos do design)
  - As consequências (de aplicar o padrão de desenho)

Padrões de desenho - GoF

"Design Patterns: Elements of Reusable Object-Oriented Software" é um livro de engenharia de software escrito por Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides **que descreve 23 padrões clássicos de desenho**.
Os 4 autores do livro ficaram conhecidos por GoF (Gang of Four).



O Padrão Iterator (Motivação)

- **Problema Genérico**
  - Uma coleção é um contentor de elementos, mas muitas vezes precisamos de percorre-los sequencialmente. **Como o Fazer?**

Quando estamos perante uma coleção do tipo List, normalmente percorremos os elementos em função do seu índice.

```
for (int i = 0; i < list.size(); i++)
    System.out.print(list.get(i) + " ");
```

- Mas se quisermos percorrer uma coleção do tipo Set, Map, Tree…então normalmente usamos um ciclo foreach

```
for (Integer i: set)
    System.out.print(i + " ");
```

- Os ciclos *foreach* só são possíveis de realizar se a coleção for *Iterable*
- Uma coleção que implementa a interface *Iterable,* é uma coleção que implementa o padrão *Iterator*

O Padrão Iterator (Motivação)

- **Problema Concreto**
  - Temos o nosso TAD Stack, que é uma coleção tipo STACK e queremos percorrer uma instancia de STACK sequencialmente do topo para a base, sem destruir o stack.

```
Stack<Integer> stack = new StackArray();

for (int i = 0; i < 20; i++)
    stack.push( 100 - i);

for (Integer i: stack)
    System.out.print(i + " ");
```

- **Solução**
  - Para percorrermos o stack, usando um ciclo foreach, temos que implementar o padrão Iterator.

O Padrão Iterator (Motivação)

- **Objectivo:** Tornar o classes do tipo Stack Iteráveis

Como ?

1. Estender a interface Stack de **Iterable**
2. Implementar o método **iterator** na classe de implementação do Stack
3. Criar uma inner classe na implementação do Stack que implemente a **interface Iterator**

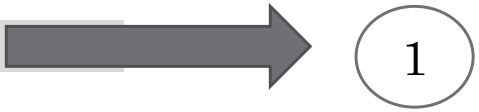
```
public interface Iterable<E>{
 Iterator<E> iterator();
}
```

```
public interface Iterator<E>{
 //devolve true se existe proximo elemento
   Boolean hasNext();
 //devolve o proximo element da sequência
   T next();
}
```

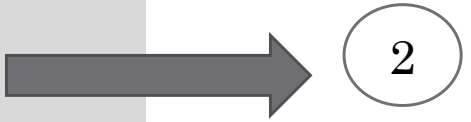
O Padrão Iterator (Motivação)

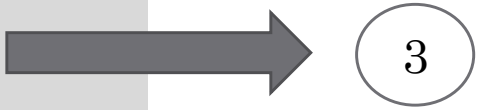
- Resolução

```
public interface Stack<E> extends Iterable<E>
```




```
public class StackArray<E> implements Stack<E> {

    private E[] storage;
    private int size;
    private final static int MAX = 500;
//other code from Stack Array
@Override
    public Iterator<E> iterator() {
        return new IteratorStack();
    }

    private class IteratorStack implements Iterator<E> {
        private int pos;

        public IteratorStack() {
            pos=size-1;
        }

        @Override
        public boolean hasNext() {
            return pos>=0;
        }

        @Override
        public E next() {
            E elem= storage[pos];
            pos--;
            return elem;
        }
    }
}
```





@

Iterator| Exercícios(1)

- Crie um projeto a partir do template:
https://github.com/pa-estsetubal-ips-pt/IteratorExamples_Template.git

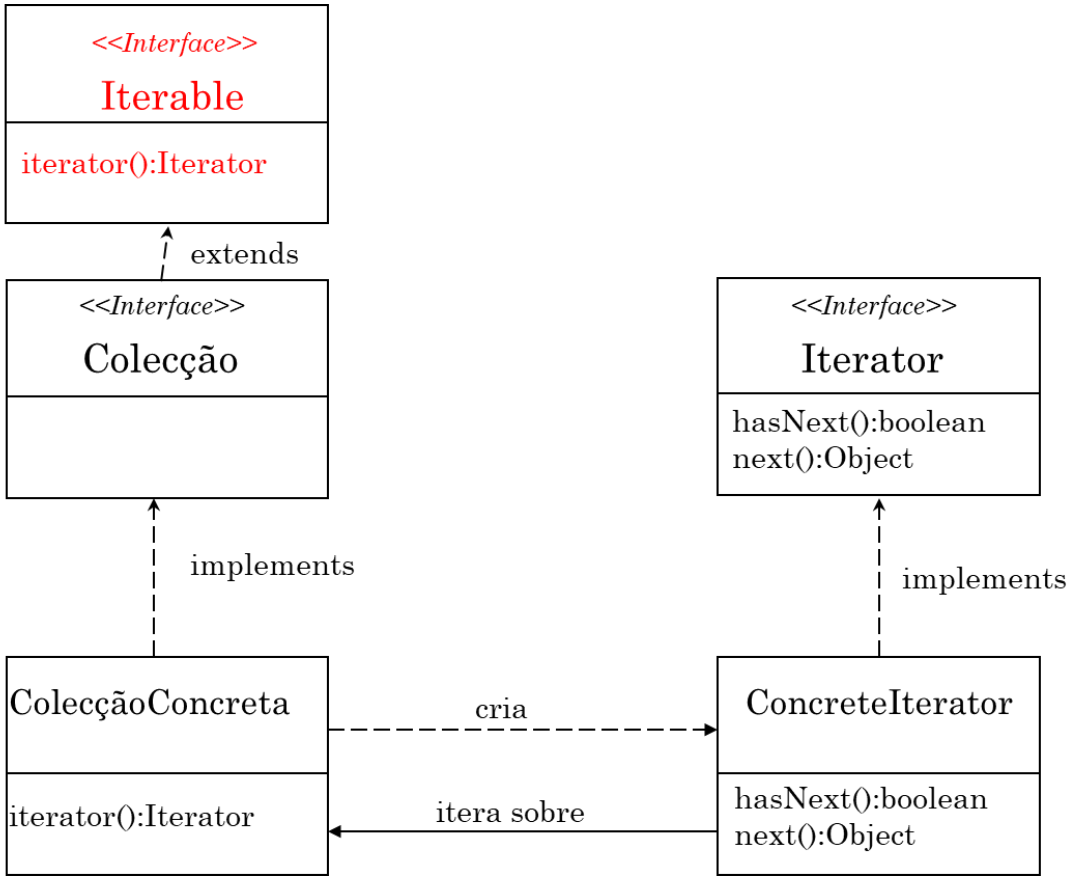
  - Reveja a implementação do Padrão Iterator para a classe StackArray

  - Implemente o método iterator na classe StackLinked, de forma a torna-la Iterável.

  - Torne a classe Tree Iterável, implementando o iterador de forma a disponibilizar o elementos da árvore na sequência pre-order.

Iterator Pattern (usando as classes e interfaces do JAVA)

- Um padrão é uma solução padronizada para um problema comum
- O padrão Iterator cria uma forma padronizada de iterar sequencialmente os elementos de um contentor



Participantes do Padrão

- Iterator
  - Define uma interface para aceder e percorrer os elementos
- ConcreteIterator
  - Implementa a interface Iterator
  - Mantém a informação da posição actual de iteração
- Coleção
  - Define uma interface para criar um objeto Iterator
- ColeçãoConcreta
  - Implementa a interface que cria o Iterator para retornar o IteradorConcreto apropriado

Revendo a receita -> solução padronizada

1. Torne a interface do tipo que pretende tornar iterável a estender de Iterable.

2. Na colecção que pretenda implementar o iterador, implemente o método Iterator<E>iterator(), que devolve uma instancia do iterador da classe ConcreteIterator.

3. Crie uma classe interna privada ConcreteIterator que implementa a interface Iterator.

4. Implemente os métodos next() e hasNext(), na classe ConcreteIterator.

Iterator| Exercícios Adicionais

Implemente para o TAD Stack um iterador que percorra os elementos da base para o topo.

- Defina na interface Stack o método Iterator<E> iteratorInverse()
- Implemente o método para o StackArray e StackLinked
- Teste este novo iterador no main, usando os métodos hasNext e next.

```
Iterator<Integer> it= stack.iteratorInverse();
 while(it.hasNext())
     System.out.print(it.next() + " ");
```

Estudar e Rever

Pagina 254-260

- https://www.tutorialspoint.com/design_pattern/iterator_pattern.htm
- https://refactoring.guru/design-patterns/iterator