Modelos, Métodos e Técnicas de Engenharia de Software
Visão e análise de projeto Padrões Prática 3 – Iterator (16)

Prof. Osmar de Oliveira Braz Junior

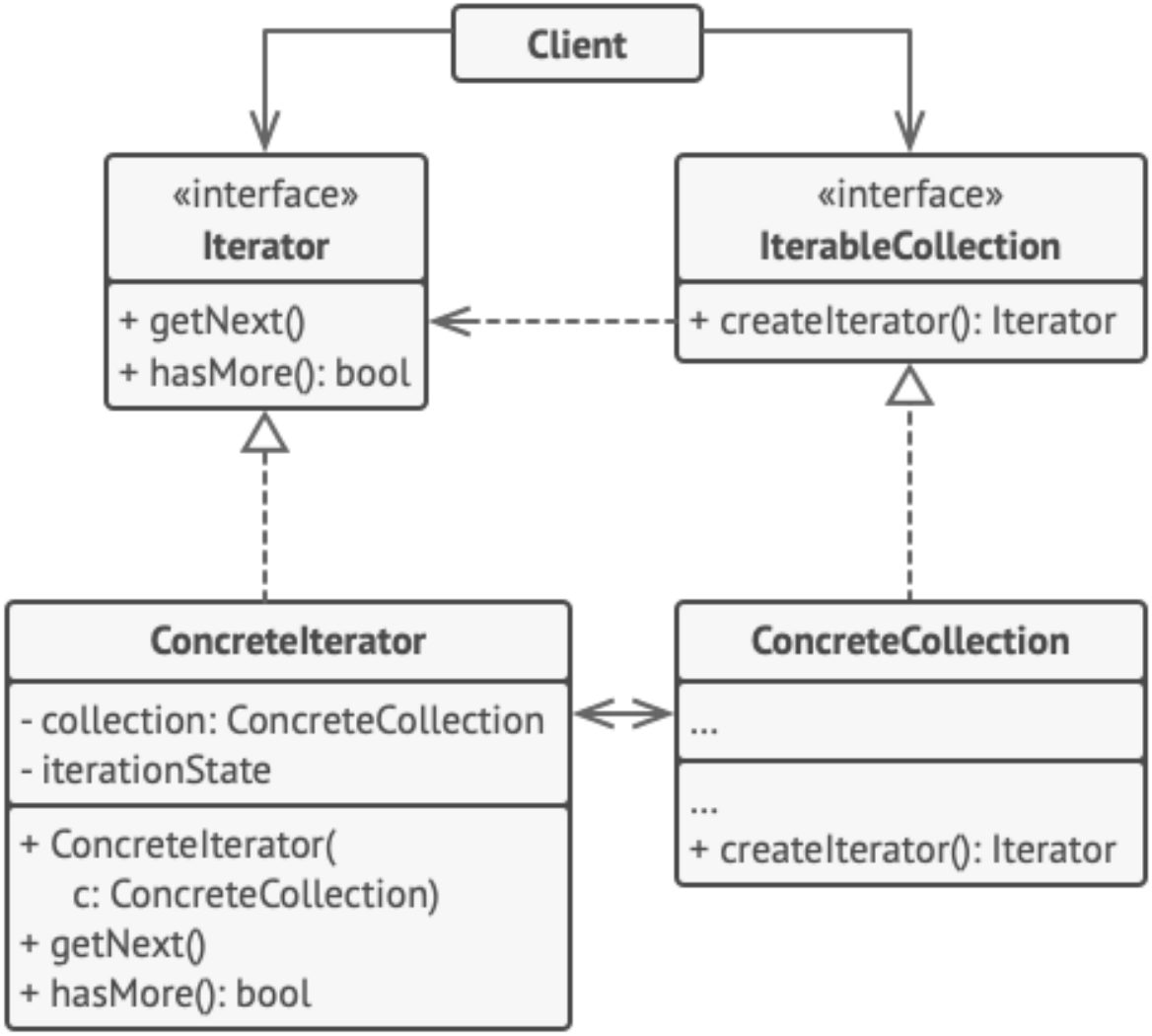
Prof. Richard Henrique de Souza

Objetivos

- Aplicar padrão comportamental ***Interpreter*** em situação problema.

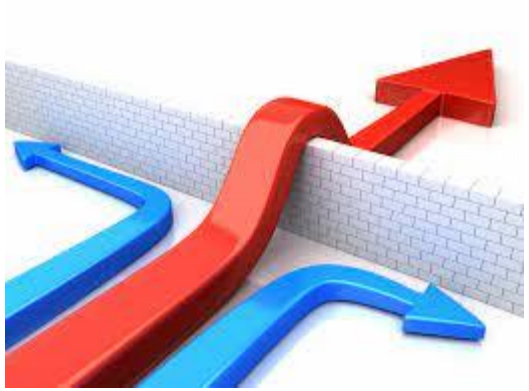
16. Iterator

Estrutura:



Importante

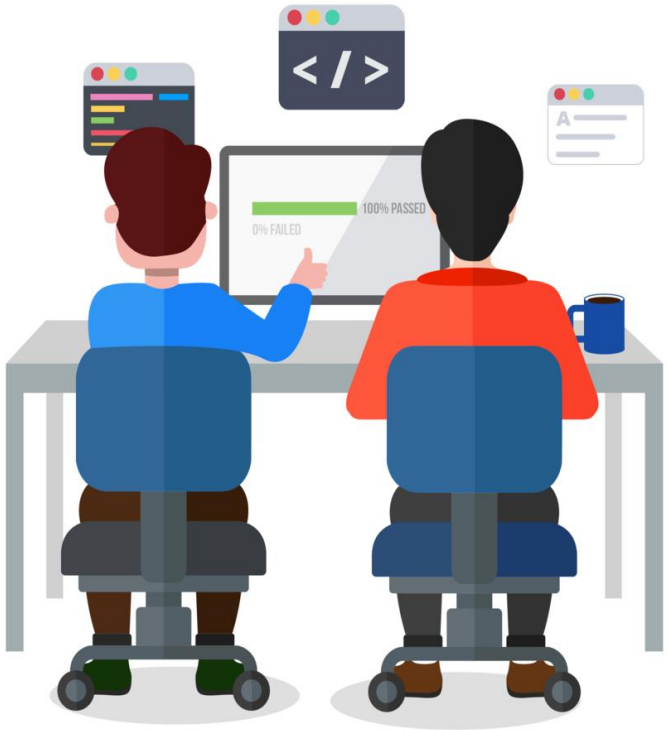
Siga os ROTEIROS !!!



Atividade em Grupo

Para esta atividade crie grupos de 2 alunos, para desenvolver a atividade segundo ***Pair Programming.***

Navegador



Piloto

Extreme Programming, metodologia ágil criada por Kent Beck em 1996.

Pair Programming

- Um é o **piloto**, responsável por escrever o código, o outro o navegador, acompanha a escrita de código e verificar se está de acordo com os **padrões do projeto** e de encontro à solução necessária.
- A intenção desta técnica é **evitar** erros de lógica, e ter um código mais confiável e melhor estruturado, utilizando-se para isso a máxima de que “**duas cabeças pensam melhor do que uma**”.

Preparação do ambiente

- Acesso a ferramenta **draw.io(https://app.diagrams.net/)** para realizar a modelagem.
- Escolha a sua linguagem de programação de preferência
- Escolha uma IDE ou o **git.dev**
- Crie um repositório no github(https://github.com/) para que todos os membros da equipe possam colaborar no desenvolvimento.

Atividade prática

1

16. Iterator
Intenção

- Prover uma forma de acessar os elementos de um conjunto em sequência sem expor a representação interna deste conjunto.
- Também conhecido como: Iterador ou Cursor.

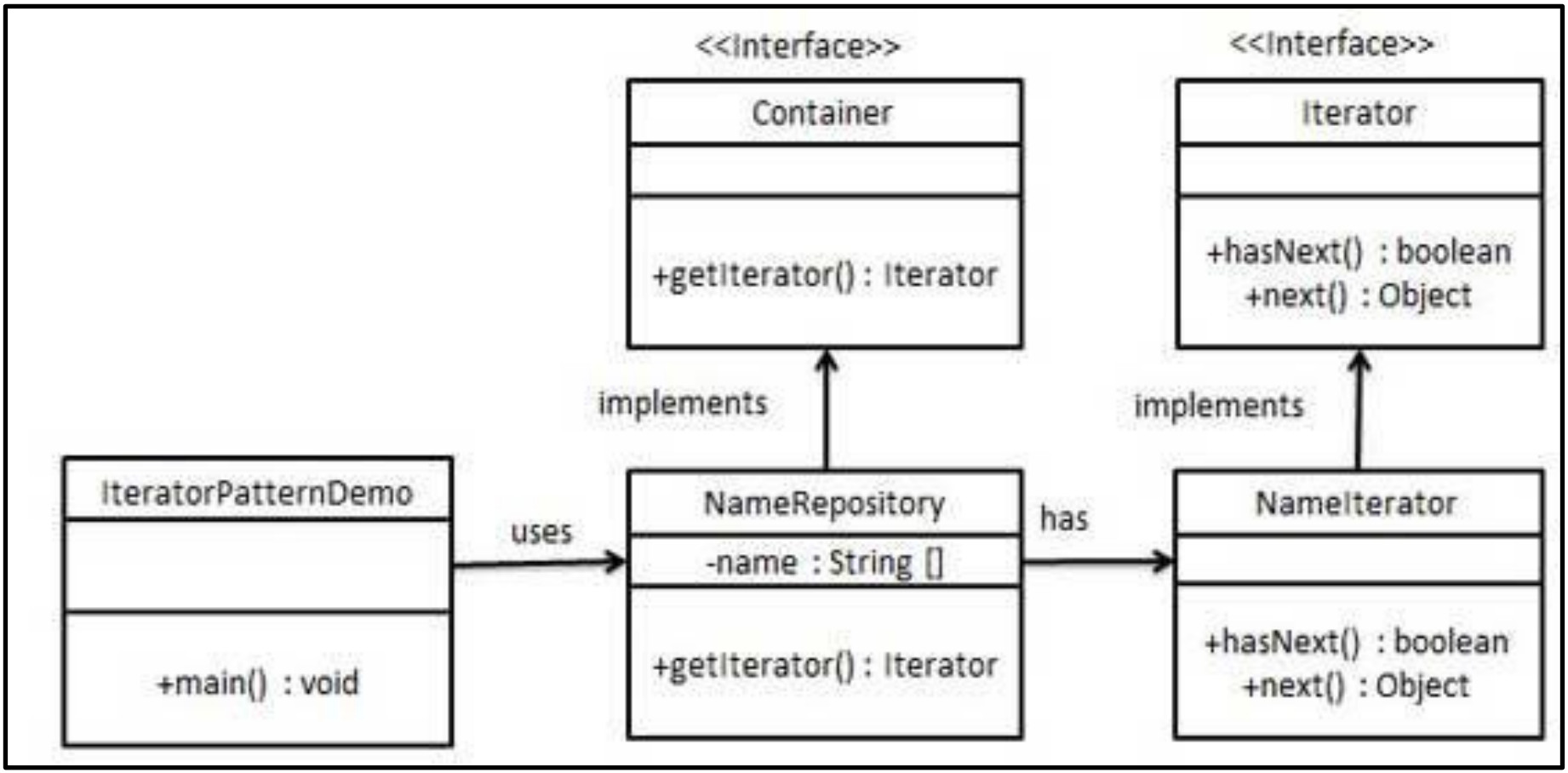
16. Iterator
Propósito

- Usar este padrão quando…
  - Quiser acessar objetos agregados (coleções) sem expor a estrutura interna;
  - Quiser prover diferentes meios de acessar tais objetos;
  - Quiser especificar uma interface única e uniforme para este acesso.

16. Iterator
Intenção

- Vantagens e desvantagens
  - Múltiplas formas de acesso:
    - Basta implementar um novo iterador com uma nova lógica de acesso.
  - Interface simplificada:
    - Acesso é simples e uniforme para todos os tipos de coleções.
  - Mais de um iterador:
    - É possível ter mais de um acesso à coleção em pontos diferentes.

16. Iterator


<<Interface>>
Container
+getIterator() : Iterator
<<Interface>>
Iterator
+hasNext() : boolean
+next() : Object
implements
implements
IteratorPatternDemo
+main() : void
uses
NameRepository
-name : String []
+getIterator() : Iterator
has
NameIterator
+hasNext() : boolean
+next() : Object

16. Iterator
Passo 1

Crie interfaces.

Iterator.java

```
public interface Iterator {
   public boolean hasNext();
   public Object next();
}
```

Container.java

```
public interface Container {
   public Iterator getIterator();
}
```

16. Iterator Passo 2

Crie uma classe concreta implementando a interface Container . Esta classe tem a classe interna NameIterator implementando a interface Iterator.

NameRepository.java

```
public class NameRepository implements Container {
   public String names[] = {"Robert" , "John" ,"Julie" , "Lora"};

   @Override
   public Iterator getIterator() {
      return new NameIterator();
   }

   private class NameIterator implements Iterator {

      int index;

      @Override
      public boolean hasNext() {

         if(index < names.length){
            return true;
         }
         return false;
      }

      @Override
      public Object next() {

         if(this.hasNext()){
            return names[index++];
         }
         return null;
      }
   }
}
```

16. Iterator
Passo 3

Use o NameRepository para obter o iterador e os nomes de impressão.

IteratorPatternDemo.java

```
public class IteratorPatternDemo {

   public static void main(String[] args) {
      NameRepository namesRepository = new NameRepository();

      for(Iterator iter = namesRepository.getIterator(); iter.hasNext();){
         String name = (String)iter.next();
         System.out.println("Name : " + name);
      }
   }
}
```

16. Iterator
Passo 6

- Terminamos
  - Teste sua implementação

Compile e **Mostre** o código para o professor
  - Pense, o que você fez aqui ?

Lembre de salvar no seu github

Conclusão

Os padrões comportamentais tem como principal função designar responsabilidades entre objetos.

Referências

- PRESSMAN, Roger; MAXIM, Bruce. Engenharia de software: uma abordagem profissional. 8.ed. Bookman, 2016. E-book. Disponível em: https://integrada.minhabiblioteca.com.br/books/9788580555349

- SOMMERVILLE, Ian. Engenharia de software. 9. ed. São Paulo: Pearson Prentice Hall, 2011. E-book. Disponível em: https://plataforma.bvirtual.com.br/Leitor/Publicacao/2613/epub/0

- LARMAN, Craig. Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e desenvolvimento iterativo. 3. ed Porto Alegre: Bookman, 2007. E-book. Disponível em: https://integrada.minhabiblioteca.com.br/books/9788577800476

Fim