Modelos, Métodos e Técnicas de Engenharia de Software
Visão e análise de projeto Padrões Prática 3 – Memento (18)

Prof. Osmar de Oliveira Braz Junior

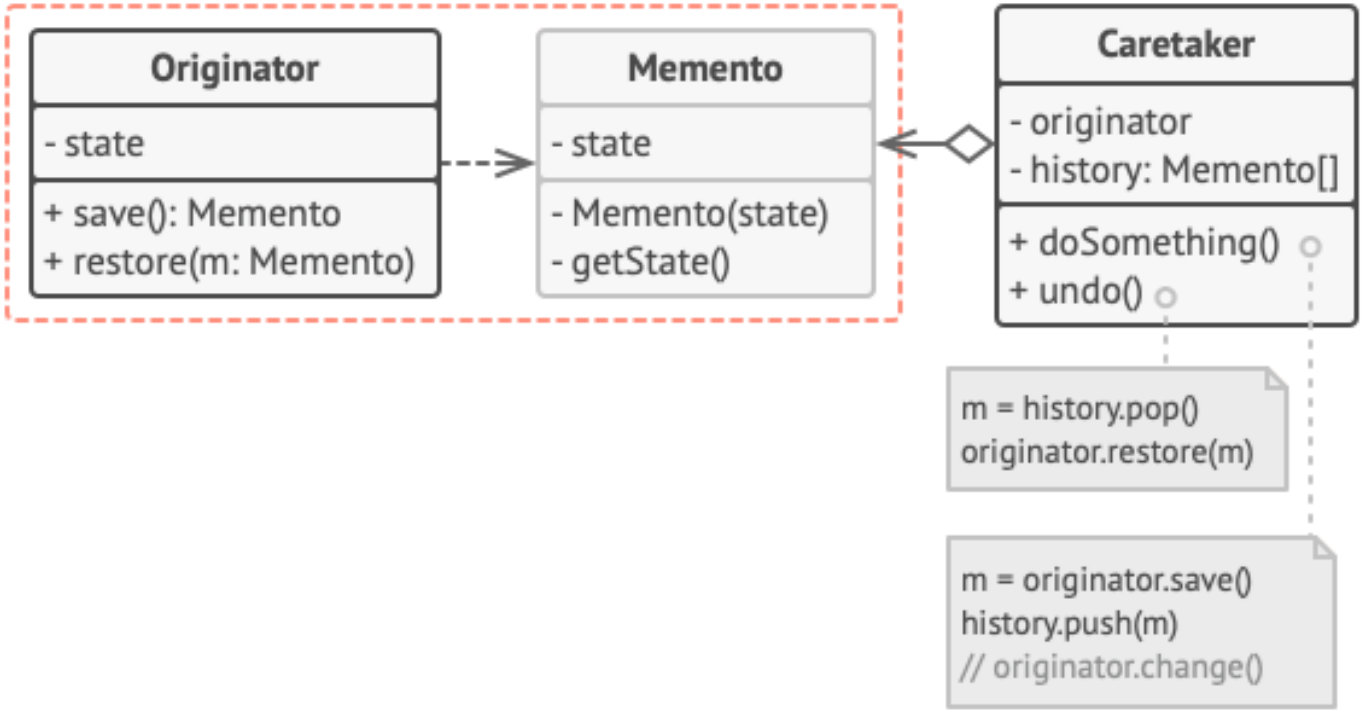
Prof. Richard Henrique de Souza

Objetivos

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

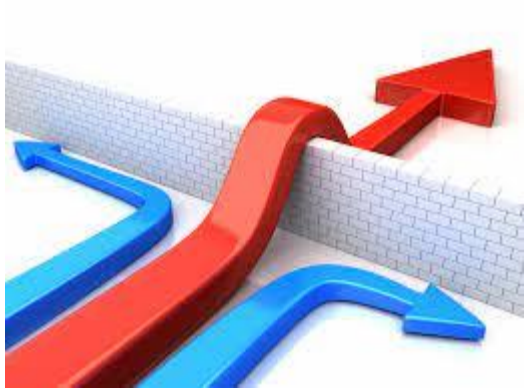
18. Memento

Estrutura: ***Implementação baseada em classes aninhadas***
A implementação clássica do padrão dependente do apoio para classes aninhadas, disponível em muitas linguagens de programação populares (tais como C++, C#, e Java).



Importante

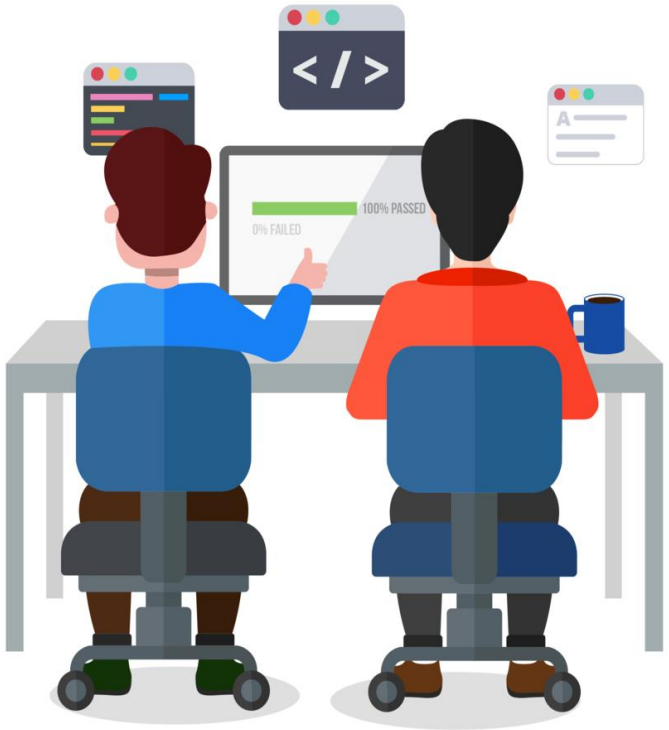
Siga os ROTEIROS !!!



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

18. Memento
Propósito

- Sem violar o encapsulamento, capturar e externalizar o estado interno de um objeto para que possa ser restaurado posteriormente.
- Também conhecido como: Lembrança, Retrato, Snapshot ou Token.

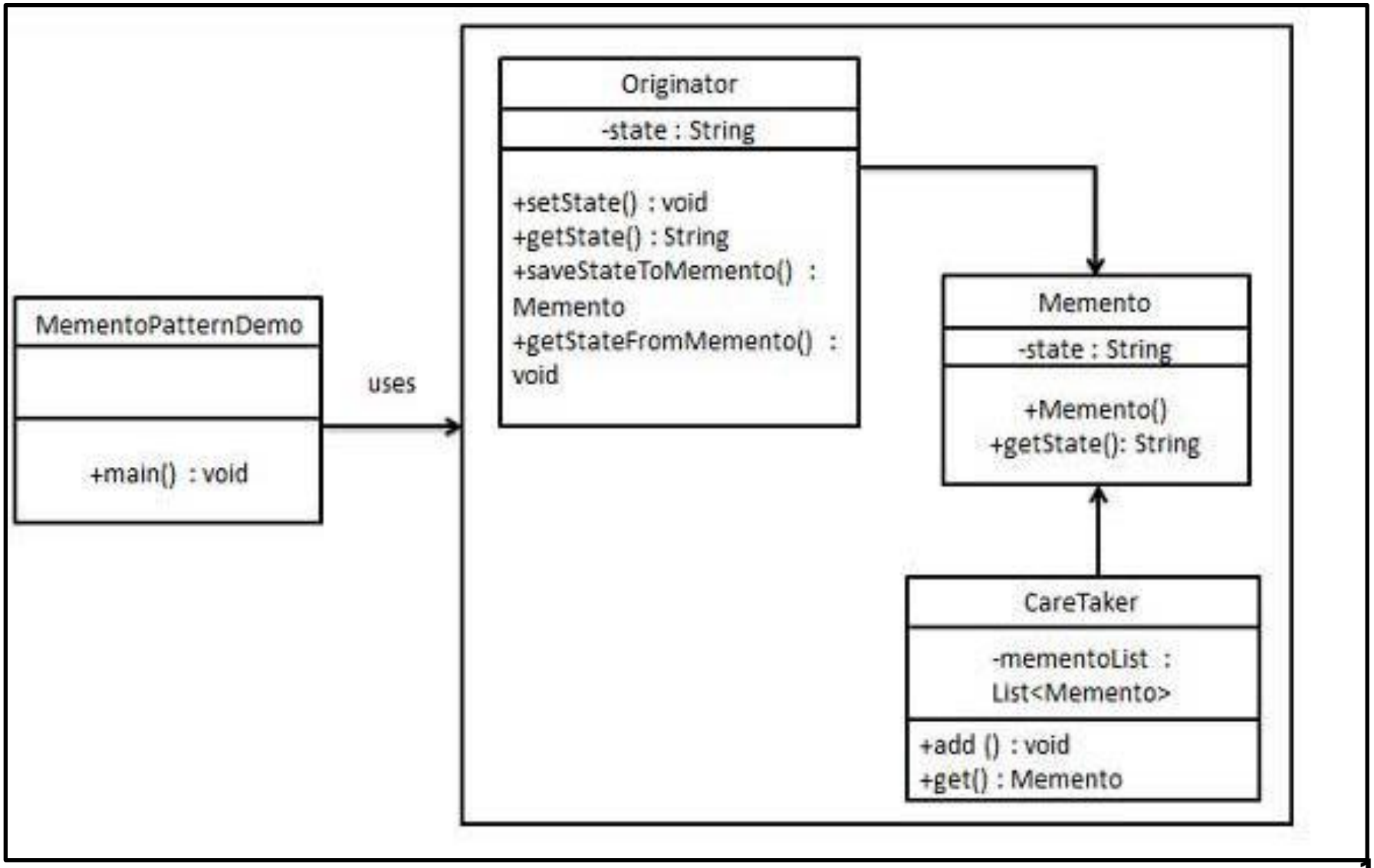
18. Memento

- Usar este padrão quando…
  - O estado do objeto (ou de parte dele) deve ser armazenado para ser recuperado no futuro;
  - Uma interface direta para obtenção de tal estado iria expor a implementação e quebrar o encapsulamento.

18. Memento

- Vantagens e desvantagens
  - Preserva o encapsulamento:
    - Retira do objeto original a tarefa de armazenar estados anteriores;
    - Caretaker não pode expor a estrutura interna do objeto, a qual tem acesso;
    - No entanto pode ser difícil esconder este estado em algumas linguagens.
  - Pode ser caro:
    - Dependendo da quantidade de estado a ser armazenado, pode custar caro.

18. Memento


Originator
-state : String
+setState() : void
+getState() : String
+saveStateToMemento() : Memento
+getStateFromMemento() : void
MementoPatternDemo
+main() : void
uses
Memento
-state : String
+Memento()
+getState(): String
CareTaker
-mementoList : List<Memento>
+add () : void
+get() : Memento

18. Memento
Passo 1

Crie a classe Memento.

Memento.java

```
public class Memento {
    private String state;

    public Memento(String state){
        this.state = state;
    }

    public String getState(){
        return state;
    }
}
```

18. Memento
Passo 2

Criar classe Originador

Originator.java

```
public class Originator {
   private String state;

   public void setState(String state){
      this.state = state;
   }

   public String getState(){
      return state;
   }

   public Memento saveStateToMemento(){
      return new Memento(state);
   }

   public void getStateFromMemento(Memento memento){
      state = memento.getState();
   }
}
```

18. Memento
Passo 3

Criar classe CareTaker

CareTaker.java

```
import java.util.ArrayList;
import java.util.List;

public class CareTaker {
   private List<Memento> mementoList = new ArrayList<Memento>();

   public void add(Memento state){
      mementoList.add(state);
   }

   public Memento get(int index){
      return mementoList.get(index);
   }
}
```

18. Memento
Passo 4

Use os objetos CareTaker e Originator

MementoPatternDemo.java

```
public class MementoPatternDemo {
   public static void main(String[] args) {

      Originator originator = new Originator();
      CareTaker careTaker = new CareTaker();

      originator.setState("State #1");
      originator.setState("State #2");
      careTaker.add(originator.saveStateToMemento());

      originator.setState("State #3");
      careTaker.add(originator.saveStateToMemento());

      originator.setState("State #4");
      System.out.println("Current State: " + originator.getState());

      originator.getStateFromMemento(careTaker.get(0));
      System.out.println("First saved State: " + originator.getState());
      originator.getStateFromMemento(careTaker.get(1));
      System.out.println("Second saved State: " + originator.getState());
   }
}
```

18. Memento
Passo 5

- Terminamos
  - Teste sua implementação

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

Lembre de salvar no seu github

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