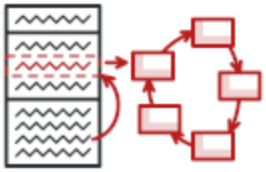


Modelos, Métodos e Técnicas de Engenharia de Software
Visão e análise de projeto Padrões Prática 3 – State (20)

Prof. Osmar de Oliveira Braz Junior

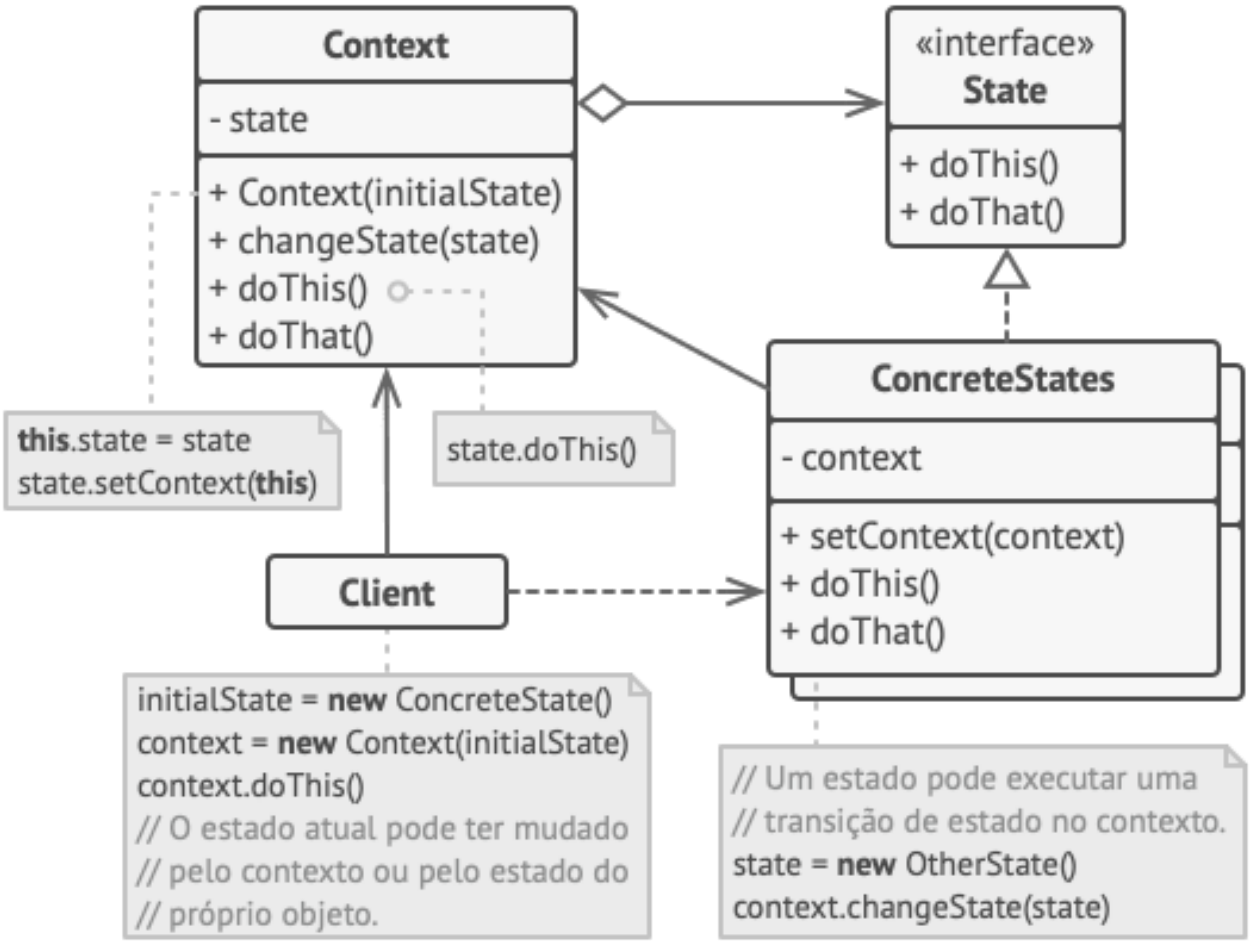
Prof. Richard Henrique de Souza

Objetivos

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

20. State

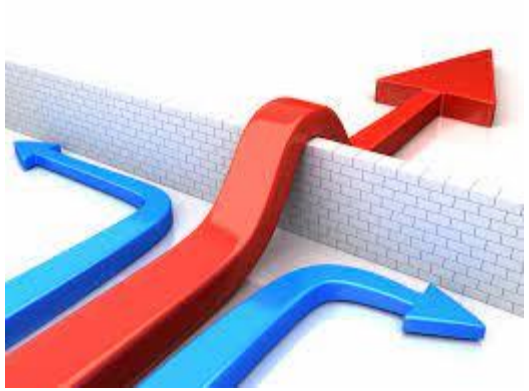
Estrutura:



https://refactoring.guru/pt-br/design-patterns/state

Importante

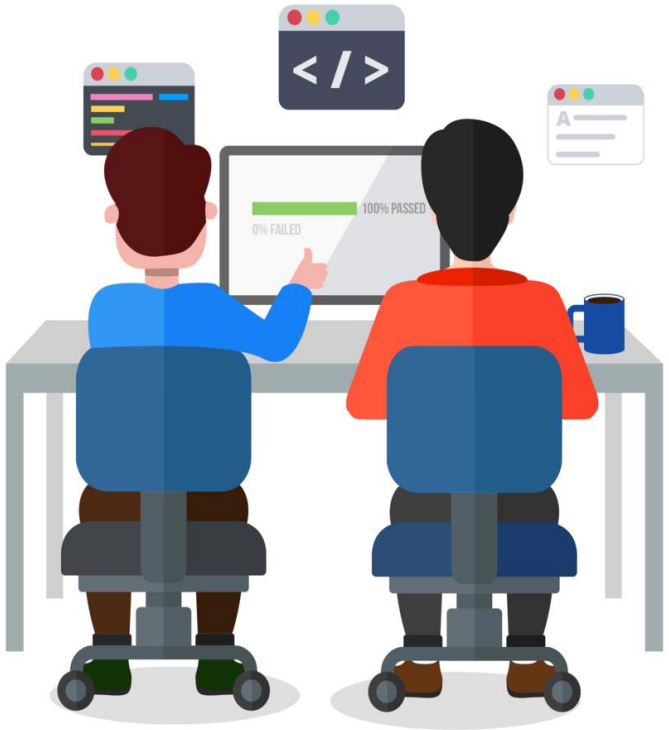
Siga os ROTEIROS !!!



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

20. State
Propósito

- Permitir que um objeto altere seu comportamento quando muda de estado interno.
- O objeto aparenta mudar de classe.
- Também conhecido como: States e Objects for States

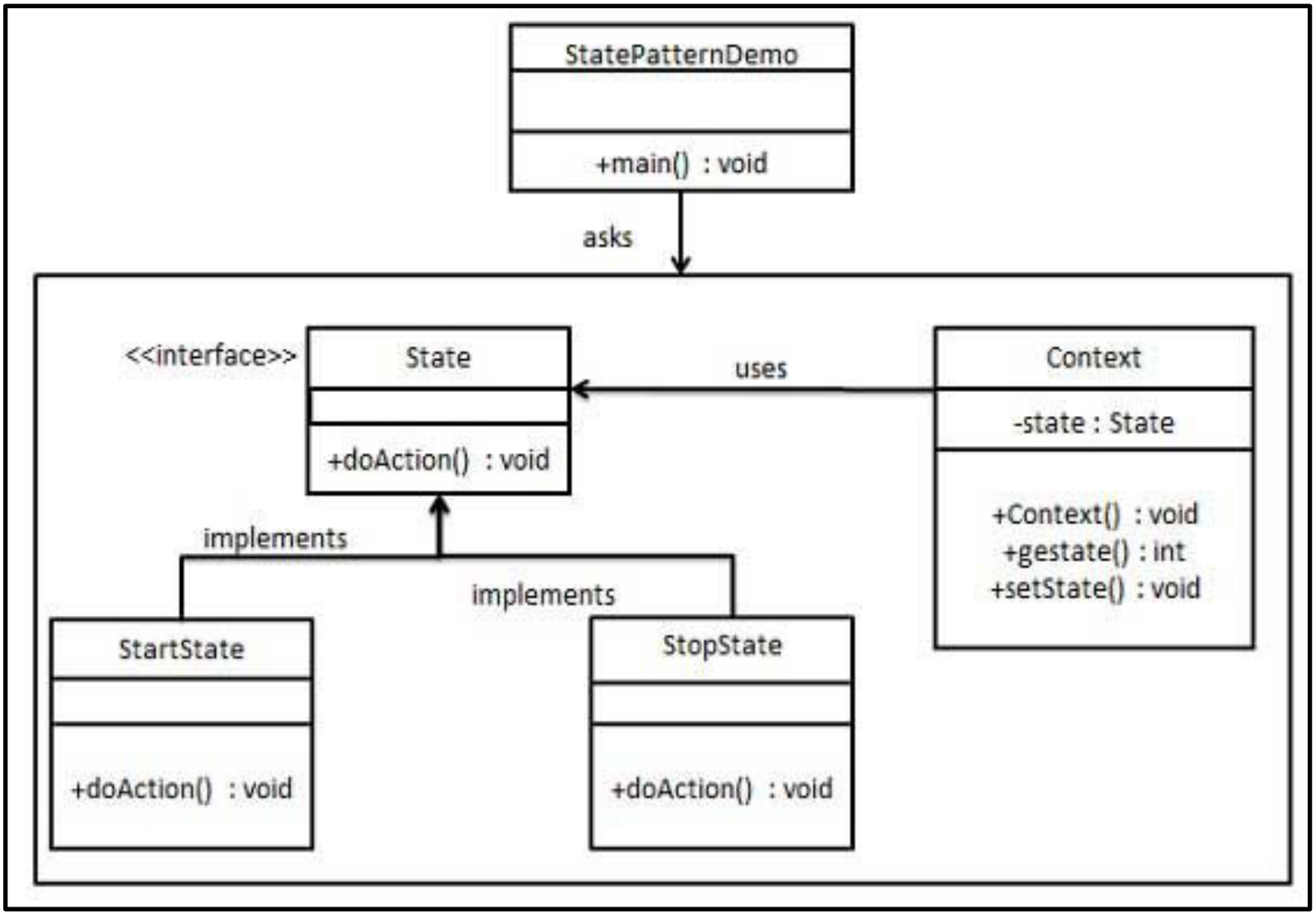
20. State

- Usar este padrão quando…
  - O comportamento de um objeto depende do seu estado, que é alterado em tempo de execução;
  - Operações de um objeto possuem condicionais grandes e com muitas partes (sintoma do caso anterior).

20. State

- Vantagens e desvantagens
  - Separa comportamento dependente de estado:
    - Novos estados/comportamentos podem ser facilmente adicionados.
  - Transição de estados é explícita:
    - Fica claro no diagrama de classes os estados possíveis de um objeto.
  - States podem ser compartilhados:
    - Somente se eles não armazenarem estado em atributos.

20. State


StatePatternDemo
+main() : void
asks
<<interface>>
State
+doAction() : void
uses
Context
-state : State
+Context() : void
+gestate() : int
+setState() : void
implements
implements
StartState
+doAction() : void
StopState
+doAction() : void

20. State
Passo 1

Crie uma interface.

State.java

```
public interface State {
   public void doAction(Context context);
}
```

20. State
Passo 2

Crie classes concretas implementando a mesma interface.

StartState.java

```
public class StartState implements State {

   public void doAction(Context context) {
      System.out.println("Player is in start state");
      context.setState(this);
   }

   public String toString(){
      return "Start State";
   }
}
```

20. State
Passo 2 - Continuação

StopState.java

```
public class StopState implements State {

   public void doAction(Context context) {
      System.out.println("Player is in stop state");
      context.setState(this);
   }

   public String toString(){
      return "Stop State";
   }
}
```

20. State
Passo 3

Criar classe de contexto.

Context.java

```
public class Context {
   private State state;

   public Context(){
      state = null;
   }

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

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

20. State
Passo 4

Use o contexto para ver a mudança no comportamento quando o estado muda.

StatePatternDemo.java

```
public class StatePatternDemo {
   public static void main(String[] args) {
      Context context = new Context();

      StartState startState = new StartState();
      startState.doAction(context);

      System.out.println(context.getState().toString());

      StopState stopState = new StopState();
      stopState.doAction(context);

      System.out.println(context.getState().toString());
   }
}
```

20. State
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

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