State

Instituto Federal do Espirito Santo – Campus Colatina
Disciplina: Projeto de Sistemas
Professor: Giovany Frossard Teixeira

Elementos Essenciais

- **Nome:** *State (Objects for States)*
- **Problema:** A troca de estados de um objeto força o programador a implementar um conjunto de condicionais grande para mapear cada situação (estado atual x comportamento diante de um certo evento).
- **Solução:** Permitir a um objeto alterar seu comportamento quando seu estado interno muda. O objeto parecerá ter mudado de classe .
- **Consequências:** O padrão elimina a necessidade de condicionais complexos e que frequentemente serão repetidos, além de permitir a inclusão de novos estados de forma mais simples.

Problema

Fonte: https://brizeno.wordpress.com/category/padroes-de-projeto/state/ (acessado em 16/03/2020)

- A troca de estados de um objeto é um problema bastante comum. Tome como exemplo o personagem de um jogo, como o Mario. Durante o jogo acontecem várias trocas de estado com o Mario, por exemplo, ao pegar uma flor de fogo o Mario pode crescer, se estiver pequeno, e ficar com a habilidade de soltar bolas de fogo. Seguem as situações possíveis:

Pegar Cogumelo:
Se Mario pequeno -> Mario grande
Se Mario grande -> 1000 pontos
Se Mario fogo -> 1000 pontos
Se Mario capa -> 1000 pontos

Pegar Flor:
Se Mario pequeno -> Mario grande e Mario fogo
Se Mario grande -> Mario fogo
Se Mario fogo -> 1000 pontos
Se Mario capa -> Mario fogo

Pegar Pena:
Se Mario pequeno -> Mario grande e Mario capa
Se Mario grande -> Mario capa
Se Mario fogo -> Mario capa
Se Mario capa -> 1000 pontos

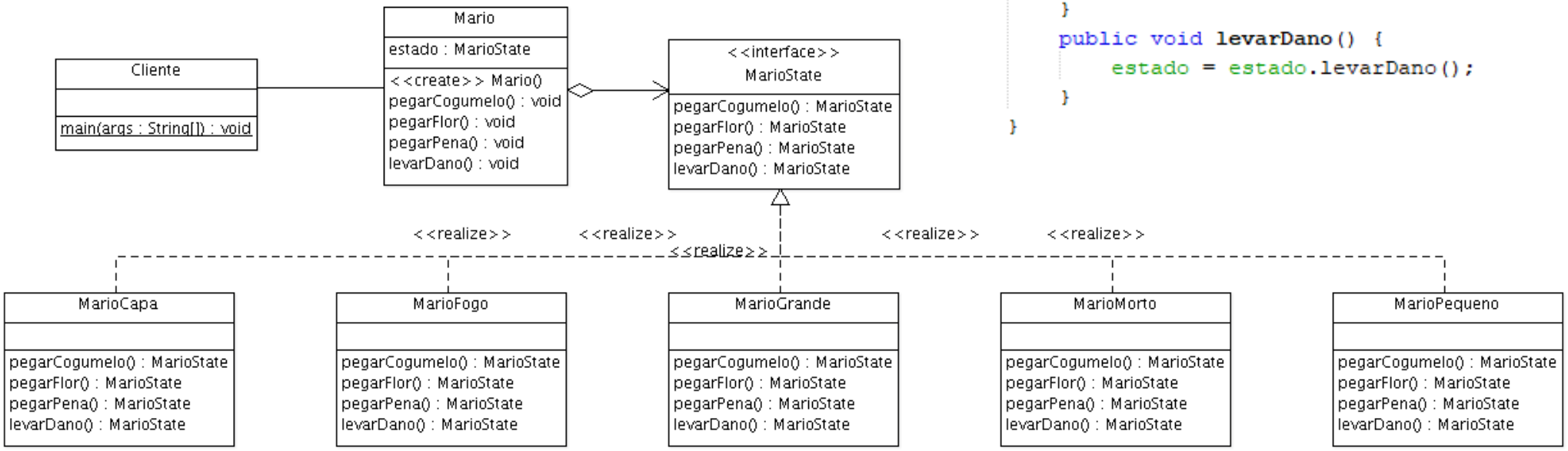
Levar Dano:
Se Mario pequeno -> Mario morto
Se Mario grande -> Mario pequeno
Se Mario fogo -> Mario pequeno
Se Mario capa -> Mario pequeno

Diagrama de classes

```
public interface MarioState {
    MarioState pegarCogumelo();
    MarioState pegarFlor();
    MarioState pegarPena();
    MarioState levarDano();
}
```

```
public class Mario {
    protected MarioState estado;

    public Mario() {
        estado = new MarioPequeno();
    }
    public void pegarCogumelo() {
        estado = estado.pegarCogumelo();
    }
    public void pegarFlor() {
        estado = estado.pegarFlor();
    }
    public void pegarPena() {
        estado = estado.pegarPena();
    }
    public void levarDano() {
        estado = estado.levarDano();
    }
}
```



main

```
public class State {
    public static void main(String[] args) {
        Mario mario = new Mario();

        mario.pegarCogumelo();
        mario.pegarPena();
        mario.levarDano();
        mario.pegarFlor();
        mario.pegarFlor();
        mario.pegarCogumelo();
        mario.levarDano();
        mario.pegarPena();
        mario.levarDano();
        mario.pegarCogumelo();
    }
}
```

```
run:
Mario grande
Mario com capa
Mario Pequeno
Mario grande com fogo
Mario ganhou 1000 pontos
Mario ganhou 1000 pontos
Mario Pequeno
Mario grande com capa
Mario Pequeno
Mario grande
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Estados do Mario

```java
public class MarioPequeno implements MarioState {
    @Override
    public MarioState pegarCogumelo() {
        System.out.println("Mario grande");
        return new MarioGrande();
    }

    @Override
    public MarioState pegarFlor() {
        System.out.println("Mario grande com fogo");
        return new MarioFogo();
    }

    @Override
    public MarioState pegarPena() {
        System.out.println("Mario grande com capa");
        return new MarioCapa();
    }

    @Override
    public MarioState levarDano() {
        System.out.println("Mario morto");
        return new MarioMorto();
    }
}
```

```java
public class MarioGrande implements MarioState {
    @Override
    public MarioState pegarCogumelo() {
        System.out.println("Mario ganhou 1000 pontos");
        return this;
    }

    @Override
    public MarioState pegarFlor() {
        System.out.println("Mario com fogo");
        return new MarioFogo();
    }

    @Override
    public MarioState pegarPena() {
        System.out.println("Mario com capa");
        return new MarioCapa();
    }

    @Override
    public MarioState levarDano() {
        System.out.println("Mario pequeno");
        return new MarioPequeno();
    }
}
```

Estados do Mario

```java
public class MarioCapa implements MarioState {
    @Override
    public MarioState pegarCogumelo() {
        System.out.println("Mario ganhou 1000 pontos");
        return this;
    }

    @Override
    public MarioState pegarFlor() {
        System.out.println("Mario com fogo");
        return new MarioFogo();
    }

    @Override
    public MarioState pegarPena() {
        System.out.println("Mario ganhou 1000 pontos");
        return this;
    }


    @Override
    public MarioState levarDano() {
        System.out.println("Mario Pequeno");
        return new MarioPequeno();
    }
}
```

```java
public class MarioFogo implements MarioState {
    @Override
    public MarioState pegarCogumelo() {
        System.out.println("Mario ganhou 1000 pontos");
        return this;
    }

    @Override
    public MarioState pegarFlor() {
        System.out.println("Mario ganhou 1000 pontos");
        return this;
    }

    @Override
    public MarioState pegarPena() {
        System.out.println("Mario com capa");
        return new MarioCapa();
    }

    @Override
    public MarioState levarDano() {
        System.out.println("Mario Pequeno");
        return new MarioPequeno();
    }
}
```

Estados do Mario

```
public class MarioMorto implements MarioState {
    @Override
    public MarioState pegarCogumelo() {
        return null;
    }

    @Override
    public MarioState pegarFlor() {
        return null;
    }

    @Override
    public MarioState pegarPena() {
        return null;
    }

    @Override
    public MarioState levarDano() {
        return null;
    }
}
```

- Notar que quando Mario chega a MarioMorto ele não volta mais aos estados anteriores.
- Uma opção interessante pode ser disparar uma exceção quando MarioMorto acionar qualquer uma das funcionalidades, uma vez que em MarioMorto não temos como pegarCogumelo(), pegarFlor(), pegarPena() ou levarDano().

Considerações finais

- O acréscimo de novos estados pode obrigar não só a redefinição da interface que define os eventos para migração entre estados (MarioState) mas também a própria implementação desses eventos nas classes concretas. Ou seja, posso ser obrigado a acrescentar um evento na interface e assim ter implementar esse evento em todas as classes concretas.
- Uma possibilidade para diminuir essa necessidade descrita no tópico anterior é o uso de métodos *default* (Interfaces em Java possuem esse recurso a partir do Java 8), dessa forma poder-se-ia definir um comportamento padrão (fazer nada, por exemplo) para a implementação nas classes concretas e só redefini-lo quando necessário.

Dúvidas?