



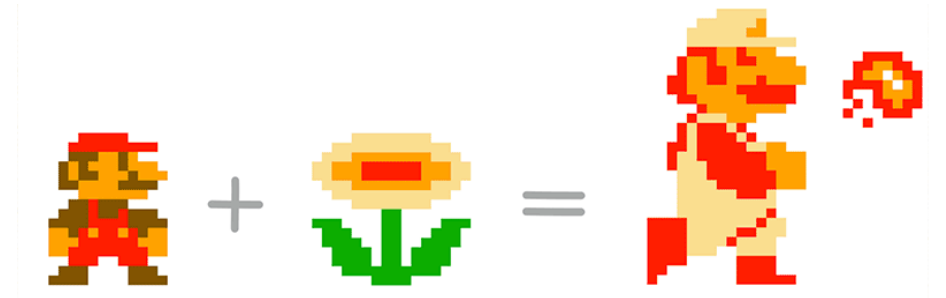
State

PADRÕES DE PROJETOS DE SOFTWARE

Introdução

O State é usado para realizar troca de estado de um objeto.

O padrão State propõe uma solução para quando um objeto requer alteração em seu comportamento de acordo com o estado interno que se encontra em um momento dado, como se ele tivesse mudado de classe.



Aplicação

Um exemplo de aplicação para o padrão State ocorre em um jogo, onde o seu personagem muda de habilidades no decorrer da aventura. Não importa se pegando itens ou ganhando poderes especiais.



Aplicação

Formas: Mario, Super Mario (quando pega um Super Mushroom), Mario Flor de Fogo (quando pega uma Fire Flower), Mario Invencível (quando pega um Starman) e Mario Morto.





Ações: Pegar Cogumelo, Pegar Starman, Pegar Fire Flower e Encostar em um inimigo.



Aplicação

Cada um dessas formas do Mario tem o mesmo conjunto de ações: seja um Mario, um Mario Flor de Fogo ou qualquer outra forma de Mario, todos têm que poder encostar em um inimigo, pegar um cogumelo, um starman ou uma fire flower. O que muda é o comportamento daquela forma de Mario perante aquela ação



	Pegar Cogumelo	Pegar Starman	Pegar Fire Flower	Encostar em um Inimigo
				
Mario	Super Mario	Mario Invencível	Mario Flor de Fogo	Mario Morto
Super Mario	Super Mario	Mario Invencível	Mario Flor de Fogo	Mario
Mario Flor de Fogo	Mario Flor de Fogo	Mario Invencível	Mario Flor de Fogo	Mario
Mario Invencível	Mario Invencível	Mario Invencível	Mario Invencível	Mario Invencível
Mario Morto	Mario Morto	Mario Morto	Mario Morto	Mario Morto

Na 1º coluna, temos os estados iniciais. A 1º linha tem as ações. O corpo da tabela é a forma que o Mario assume após o evento.

Esse mapeamento de troca de formas é o que chamamos de Máquina de Estados:

- ▶ Indica quais formas objeto tem (estado);
- ▶ Quais ações aquele estado pode realizar;
- ▶ O novo estado ele vai assumir após realizar tal ação.

4 ações x 5 formas = 20 métodos a serem implementados

O Problema

Precisamos saber qual ação foi executada e em qual forma o personagem estava. Para fazer isso, precisaríamos de um conjunto imenso de testes aninhados que fariam, para cada ação:

Se o jogador pegou um Cogumelo:

- está na forma Mario, troque para a forma Super Mario;

- está na forma Super Mario, mantenha-se como Super Mario;

- está na forma Fire Flower Mario, mantenha-se como Fire Flower Mario;

- está na forma Mario Invencível, mantenha-se como Mario Invencível;

- está na forma Mario Morto, mantenha-se como Mario Morto;

Desenvolvimento: Interface

```
• Mario.IMario
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Mario
{
    public interface IMario
    {
        IMario PegarCogumelo();
        IMario PegarEstrela();
        IMario PegarFlorDeFogo();
        IMario ReceberDano();
        String RetornarTipo();
    }
}
```

Para iniciar o State, comecemos encapsulando todos os métodos que identificamos como alteradores de estados e propagar esses métodos os estados.

- ▶ PegarCogumelo();
- ▶ PegarEstrela();
- ▶ PegarFlorDeFogo();
- ▶ ReceberDano();

Desenvolvimento : Estados

- ▶ Mario
- ▶ Super Mario
- ▶ Mario Flor de Fogo
- ▶ Mario Invencível
- ▶ Mario Morto

Cada um desses estados é uma classe que implementa a interface **IMario**.

Desenvolvimento : Classe TMario

```
public class TMario : IMario
{

    public TMario() { }
    public IMario PegarCogumelo()
    {
        Console.WriteLine("");
        Console.WriteLine("Mario Pega um GROW UP");
        Console.WriteLine("Status: SUPER MARIO");
        Console.WriteLine("");
        return new TSuperMario();
    }
    public IMario PegarEstrela()
    {
        Console.WriteLine("");
        Console.WriteLine("Mario Pega uma Starman");
        Console.WriteLine("Status: MARIO INVENCIVEL");
        Console.WriteLine("");
        return new TMarioInvencivel();
    }
}
```

Desenvolvimento : Classe TMario

```
}  
  
public IMario PegarFlorDeFogo()  
{  
    Console.WriteLine("");  
    Console.WriteLine("Mario Pega uma Fire Flower");  
    Console.WriteLine("Status: MARIO FIRE FLOWER");  
    Console.WriteLine("");  
    return new TMarioFlordeFogo();  
}  
public IMario ReceberDano()  
{  
    Console.WriteLine("");  
    Console.WriteLine("Mario foi atingido");  
    Console.WriteLine("Status: MARIO MORTO");  
    Console.WriteLine("");  
    return new TMarioMorto();  
}  
public String RetornarTipo()  
{  
    return "MARIO";  
}  
}
```

Desenvolvimento : Classe TSuperMario

```
public class TSuperMario : IMario
{
    public TSuperMario() { }
    public IMario PegarCogumelo()
    {
        Console.WriteLine("");
        Console.WriteLine("Mario Pega uma GROW UP");
        Console.WriteLine("Mario ganhou 1000 pontos");
        Console.WriteLine("Status: SUPER MARIO");
        Console.WriteLine("");
        return this;
    }
    public IMario PegarEstrela()
    {
        Console.WriteLine("");
        Console.WriteLine("Mario Pega uma Starman");
        Console.WriteLine("Status: MARIO INVENCIVEL");
        Console.WriteLine("");
        return new TMarioInvencivel();
    }
}
```

Desenvolvimento : Classe TSuperMario

```
public IMario PegarFlorDeFogo()  
{  
    Console.WriteLine("");  
    Console.WriteLine("Mario Pega uma Fire Flower");  
    Console.WriteLine("Status: MARIO FIRE FLOWER");  
    Console.WriteLine("");  
    return new TMarioFlordeFogo();  
}  
public IMario ReceberDano()  
{  
    Console.WriteLine("");  
    Console.WriteLine("Mario foi atingido");  
    Console.WriteLine("Status: MARIO");  
    Console.WriteLine("");  
    return new TMario();  
}  
public String RetornarTipo()  
{  
    return "SUPER MARIO";  
}  
}
```

Desenvolvimento: Main

Criar um objeto do tipo inicial (exemplo **TMario**) e, no decorrer do percurso, realizar as chamadas aos métodos.

```
static void Main(string[] args)
{
    Console.WriteLine("SUPER MARIO");

    IMario objPersonagem;
    objPersonagem = new TMario();
    objPersonagem = objPersonagem.PegarCogumelo();
    objPersonagem = objPersonagem.PegarFlorDeFogo();
    objPersonagem = objPersonagem.ReceberDano();
    objPersonagem = objPersonagem.PegarCogumelo();
    objPersonagem = objPersonagem.PegarFlorDeFogo();
    objPersonagem = objPersonagem.ReceberDano();
    objPersonagem = objPersonagem.ReceberDano();
    objPersonagem = objPersonagem.PegarCogumelo();
    objPersonagem = objPersonagem.PegarEstrela();
    objPersonagem = objPersonagem.ReceberDano();
    objPersonagem = objPersonagem.ReceberDano();

    Console.Read();
}
```

Vantagens

- ▶ O código fica muito mais simples de entender sem ifs e cases aninhados uns dentro dos outros.
- ▶ Simplifica a adição de um novo estado simplesmente fazendo com que ele implemente a interface base.
- ▶ Modifica o funcionamento do sistema sem que haja a necessidade de prévia programação. Trocar do estado inicial ou a chamada de um método faz com que toda a linha de funcionamento do programa possa ser modificada facilmente.