Mini EP 5 - Padrões e Contextos

Contexto 1:

Você está desenvolvendo um jogo chamado Mario. Nele, o protagonista pode receber vários itens que o fornecem melhorias, mudando assim o seu estado:

- Pequeno
    - Ao receber dano, morre.
    - Ao receber cogumelo vermelho, muda o seu estado para grande.
    - Ao receber a flor, muda seu estado para flamejante.
    - Ao receber a pena, muda o seu estado para voador.
- Grande
    - Ao receber dano, muda seu estado para pequeno.
    - Ao receber cogumelo vermelho, nada ocorre.
    - Ao receber a flor, muda seu estado para flamejante.
    - Ao receber a pena, muda o seu estado para voador.
- Flamejante
    - Ao receber dano, muda seu estado para grande.
    - Ao receber cogumelo vermelho, nada ocorre.
    - Ao receber a flor, nada ocorre.
    - Ao receber a pena, muda o seu estado para voador.
- Voador
    - Ao receber dano, muda seu estado para grande.
    - Ao receber cogumelo vermelho, nada ocorre.
    - Ao receber a flor, muda seu estado para flamejante.
    - Ao receber a pena, nada ocorre.

Problema:
Como escrever um código que represente bem os estados do Mario, permitindo que o código seja extensível para novos estados?

Solução: usar o padrão State, criando uma classe para cada estado possível. As classes devem implementar a classe abstrata EstadoMario, onde cada método indica uma ação possível. EstadoMario possui uma referência ao Mario e o Mario possui uma referência a um estado.

Classe Mario:
    estado: EstadoMario

```
    Método mudarEstado(novoEstado):
        estado = novoEstado

    Método receberDano():
        estado.receberDano()

    Método receberCogumeloVermelho():
        estado.receberCogumeloVermelho()

    Método receberFlor():
        estado.receberFlor()

    Método receberPena():
        estado.receberPena()
```

```
Classe abstrata EstadoMario:
    protagonista: Mario
    EstadoMario(mario):
        protagonista = mario

    Método receberDano()
    Método receberCogumeloVermelho()
    Método receberFlor()
    Método receberPena()
```

```
Classe PequenoState implementa EstadoMario:
    Método receberDano():
        Imprimir "Mario morreu!"
        // Realizar as ações necessárias

    Método receberCogumeloVermelho():
        Imprimir "Mario ficou grande!"
        protagonista.mudarEstado(new GrandeState())

    Método receberFlor():
        Imprimir "Mario ficou flamejante!"
        protagonista.mudarEstado(new FlamejanteState())

    Método receberPena():
        Imprimir "Mario ficou voador!"
        protagonista.mudarEstado(new VoadorState())
```

```
Classe GrandeState implementa EstadoMario:
    Método receberDano():
        Imprimir "Mario voltou a ser pequeno!"
        protagonista.mudarEstado(new PequenoState())

    Método receberCogumeloVermelho():
        // Nada ocorre

    Método receberFlor():
        Imprimir "Mario ficou flamejante!"
        protagonista.mudarEstado(new FlamejanteState())

    Método receberPena():
        Imprimir "Mario ficou voador!"
        protagonista.mudarEstado(new VoadorState())
```

```
Classe FlamejanteState implementa EstadoMario:
    Método receberDano():
        Imprimir "Mario voltou a ser grande!"
        protagonista.mudarEstado(new GrandeState())

    Método receberCogumeloVermelho():
        // Nada ocorre

    Método receberFlor():
        // Nada ocorre

    Método receberPena():
        Imprimir "Mario ficou voador!"
        protagonista.mudarEstado(new VoadorState())
```

```
Classe VoadorState implementa EstadoMario:
    Método receberDano():
        Imprimir "Mario voltou a ser grande!"
        protagonista.mudarEstado(new GrandeState())

    Método receberCogumeloVermelho():
        // Nada ocorre
```

```
    Método receberFlor():
        Imprimir "Mario ficou flamejante!"
        protagonista.mudarEstado(new FlamejanteState())

    Método receberPena():
        // Nada ocorre
```

Contexto 2:

Você está desenvolvendo um player versátil de som. Ele deve suportar diversos formatos: MP3, OGG, etc. A decodificação de cada formato usa uma estratégia diferente.

Problema: como escrever um código extensível que aceite diversos formatos?

Solução: usar o padrão strategy: criar uma interface que possui o método de decodificação. Criar uma classe que implementa essa interface para cada formato diferente. A classe PlayerSom deve ter uma referência para essa interface.

```
Interface FormatoAudio:
    Método decodificar(arquivo)
```

```
Classe Mp3Formato implementa FormatoAudio:
    Método decodificar(arquivo):
        Imprimir "Decodificando arquivo MP3: " + arquivo
        // Lógica de decodificação do arquivo MP3
```

```
Classe OggFormato implementa FormatoAudio:
    Método decodificar(arquivo):
        Imprimir "Decodificando arquivo OGG: " + arquivo
        // Lógica de decodificação do arquivo OGG
```

```
Classe PlayerSom:
    formato: FormatoAudio

    Método setFormatoAudio(formato):
        this.formato = formato
```

```
Método reproduzir(arquivo):
    Se formato != null:
        formato.decodificar(arquivo)
        // Lógica de reprodução do arquivo decodificado
    Senão:
        Imprimir "Formato de áudio não suportado."
```