Padrões de Projeto State

Universidade Federal de Uberlândia
Disciplina: POO2
Prof. Fabiano Dorça

Padrão State

- A intenção do padrão:
  - “Permite a um objeto alterar seu comportamento quando seu estado interno muda. O objeto parecerá ter mudado de classe.” [1]

Padrão State

- Atualização automática do estado de um objeto.
- Alteração de comportamento quando quando o estado interno muda.
- Torna o comportamento dependente do estado.

Padrão State

- Especificar
  - os diferentes estados que um objeto pode assumir
  - transições de estados,
  - definição de uma máquina de estados orientada a objetos.

Padrão State

- Adição de novos estados e comportamentos, através da adição de novas classes.
- Cria-se um objeto para cada estado possível.

Padrão State

- Objetivo do padrão:
  - Permitir que um objeto tenha seu comportamento alterado de acordo com o estado interno que se encontra em um momento dado.

Padrão State

- Participantes:
  - Context
    - Define uma interface com o cliente.
    - Mantém uma instância de um ConcreteState que define o seu estado atual.

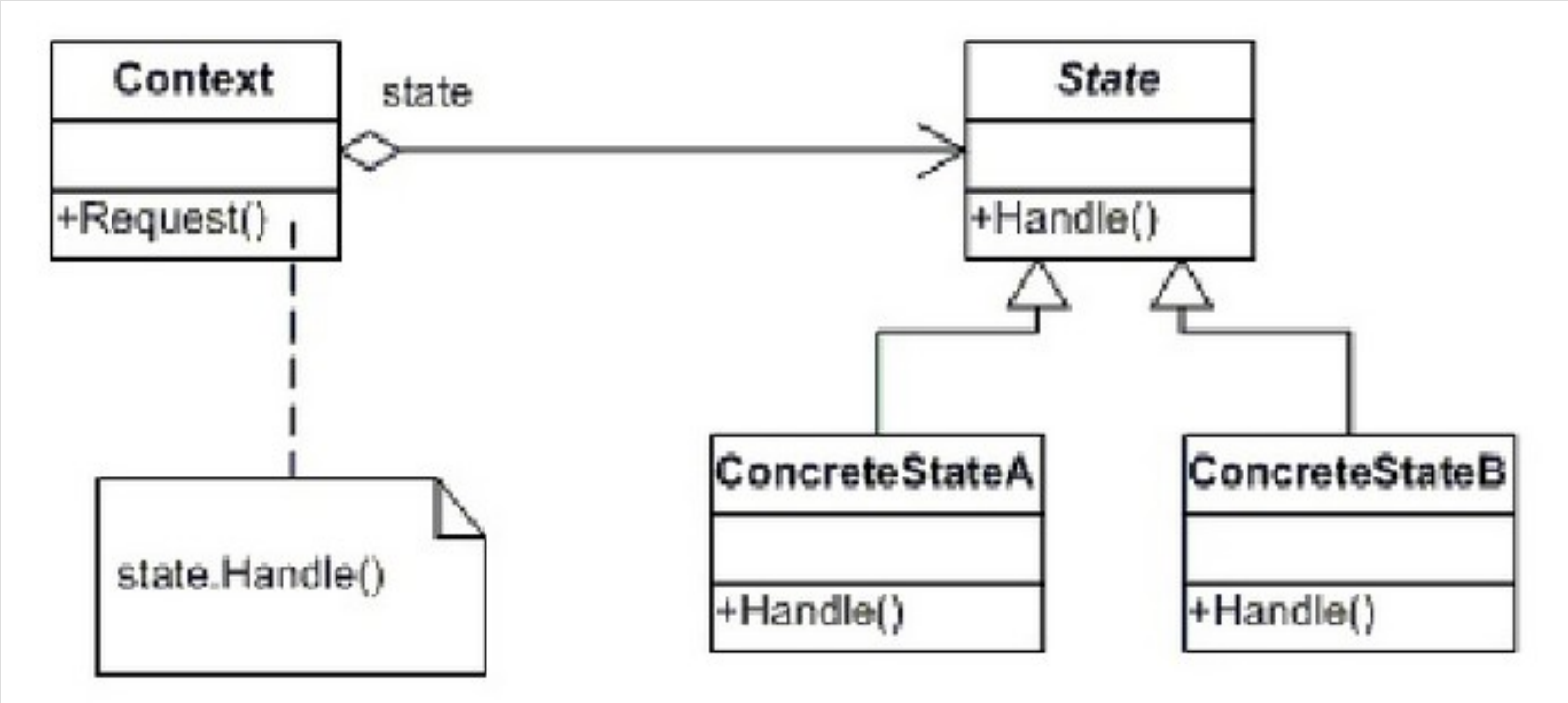
Padrão State

- State
  - Define uma interface para a criação de estados concretos.

- Concrete State
  - Implementa o comportamento associado a um estado particular de Context.

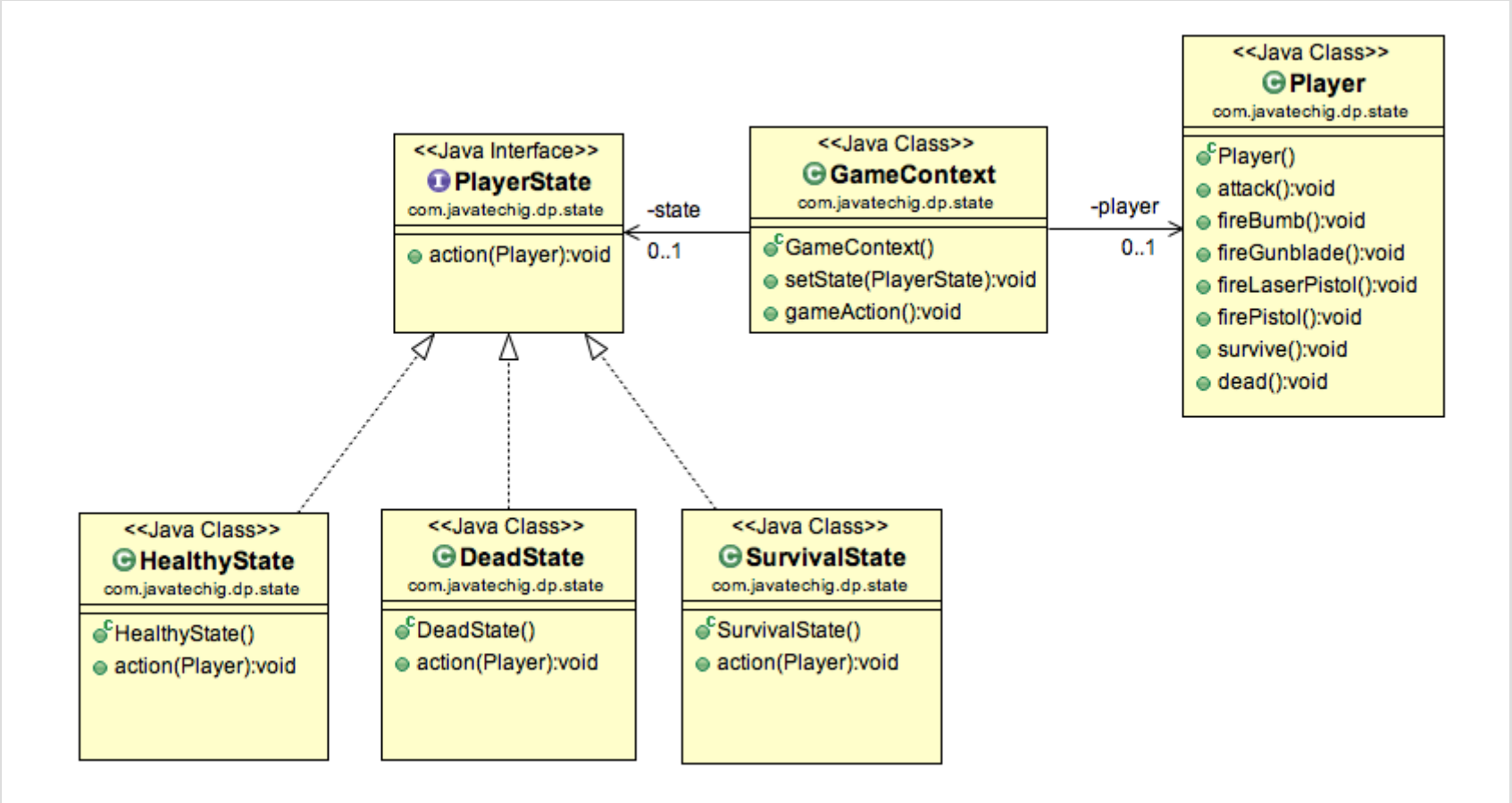
Padrão State

- Diagrama de classes



Padrão State

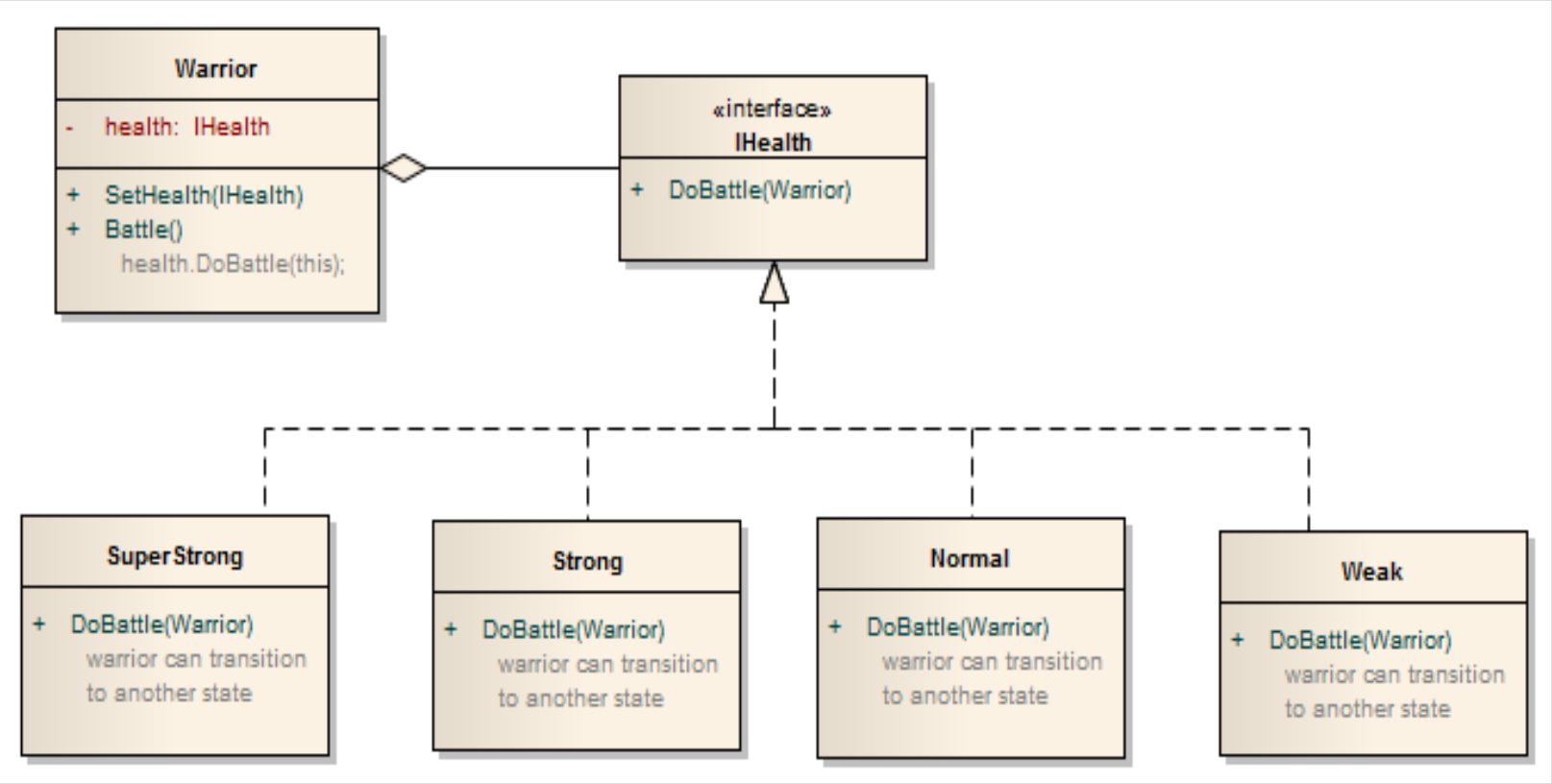
- Exemplo



Fonte: State Design Pattern in Java | Java Techig

Padrão State

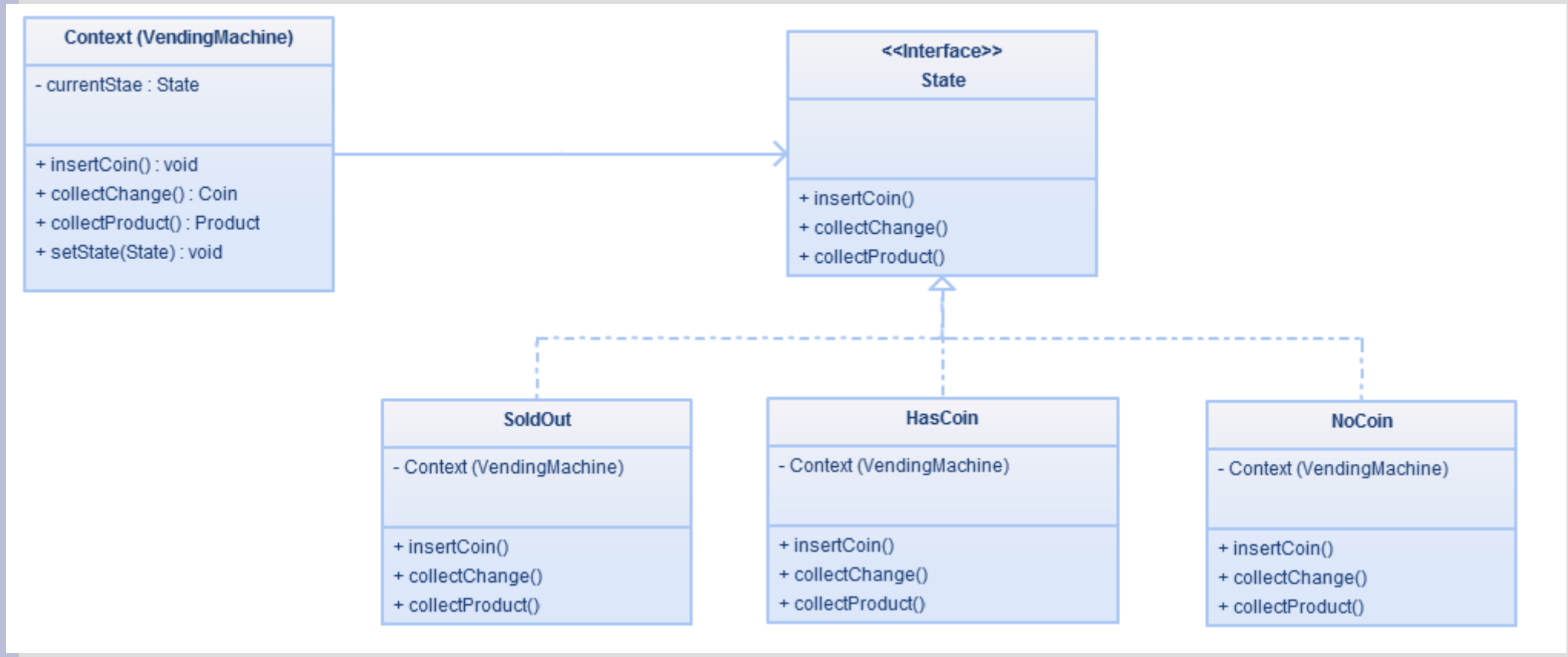
- Exemplo



Fonte: State Design Pattern - CodeProject
www.codeproject.com

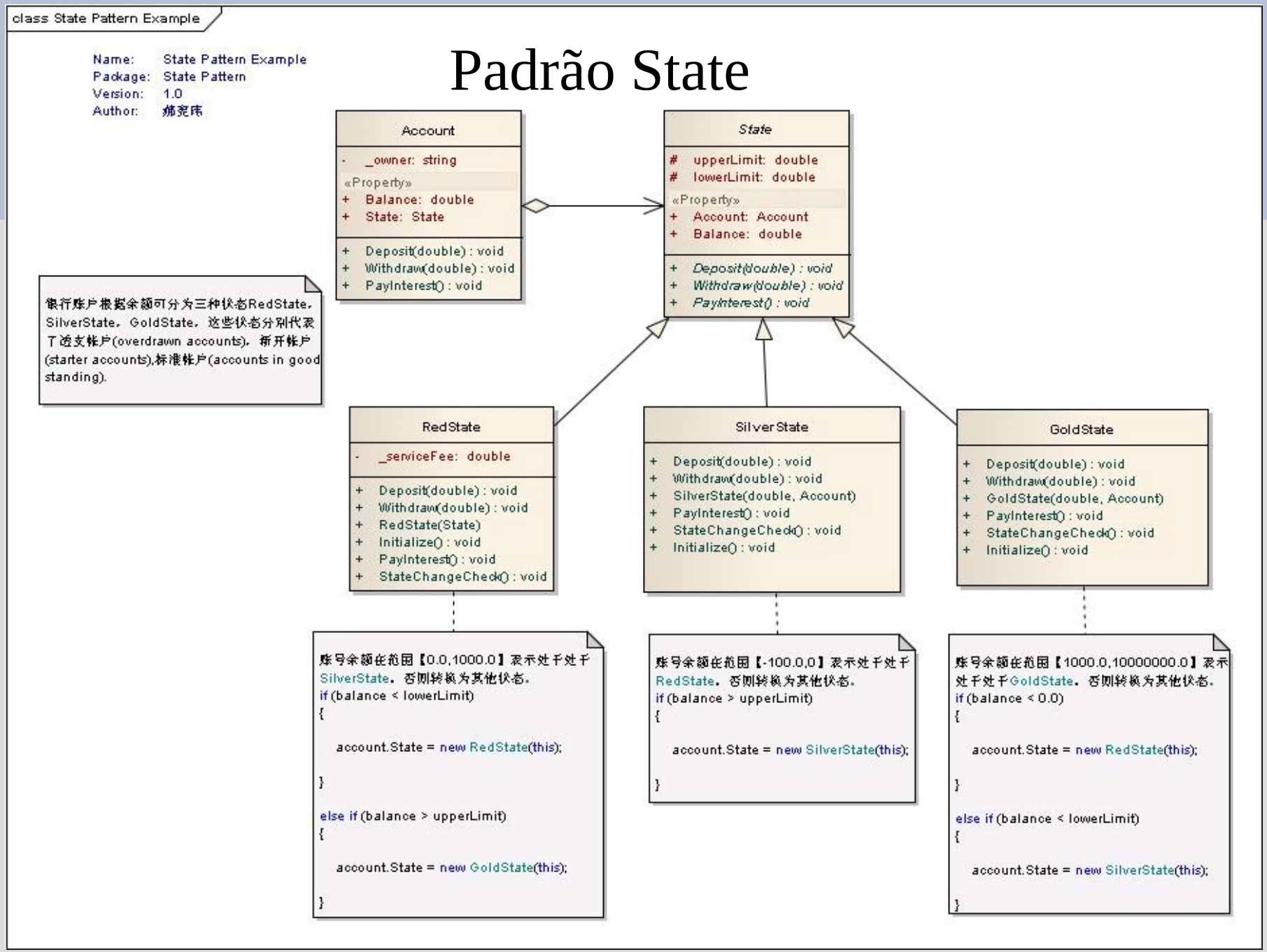
Padrão State

- Exemplo



Fonte: javarevisited.blogspot.com

Padrão State



Fonte:
http://www.cnblogs.com/Mayvar/archive/2011/09/08/wanghonghua_201109080323.html

Padrão State

Exemplo de implementação:

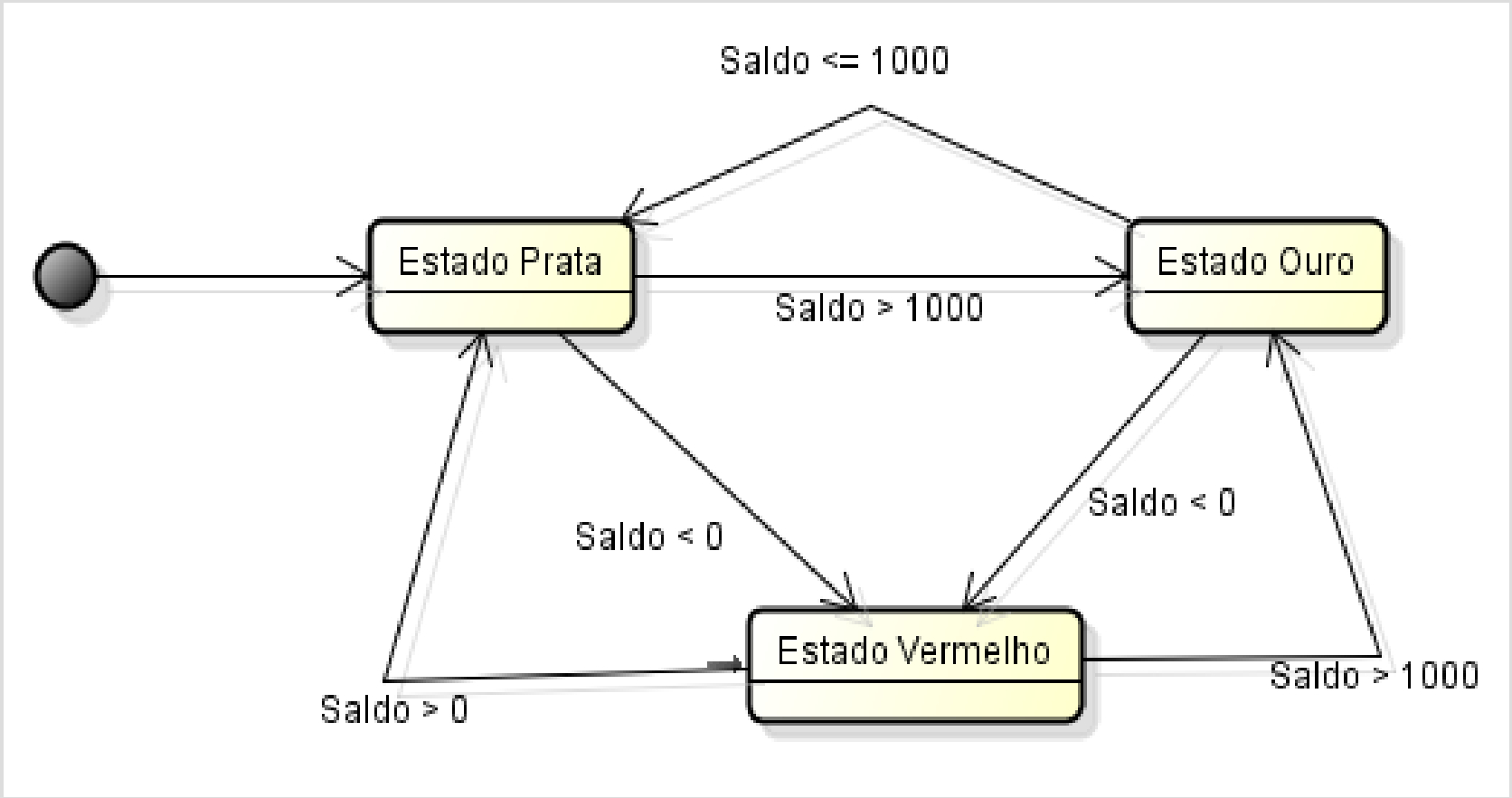
Controle de estado de contas bancárias:

- Se a conta estiver com saldo entre 0 e 1000 = estado prata.
  Neste caso, o cliente deve pagar uma taxa por cada saque que realizar.

- Se a conta estiver com saldo maior que 1000 = estado ouro.
  Neste caso, o cliente não paga taxa de saque e ainda ganha um rendimento imediado em cada depósito que realizar.

- Se a conta estiver com saldo negativo = estado vermelho. Neste caso, não é permitido a realização de saques. Apenas depósitos.

Padrão State



Padrão State

- Implementação

```
// "Context"
class Conta {
    private Estado estado;
    private String numero;
    private double saldo;

    // Construtor
    public Conta(String numero) {
        // As novas contas são por default 'Prata'
        this.numero = numero;
        this.estado = new EstadoPrata(this);
        this.saldo = 0.0;
     }
//getters & setters
```

Padrão State

```
    public void depositar(double quantia) {
        estado.depositar(quantia);
        System.out.println("Depósito--- " + quantia);
        System.out.println(" Saldo = " + this.getSaldo());
        System.out.println(" Estado = " + this.estado.getClass().getName());
    }

    public void sacar(double quantia) {
        estado.sacar(quantia);
        System.out.println("Saque--- " + quantia);
        System.out.println(" Saldo = " + this.getSaldo());
        System.out.println(" Estado = " + this.estado.getClass().getName());
     }
} //Fim da classe Conta
```

Padrão State

```
//State
public abstract class Estado {
    private Conta conta;
    private double limiteInferior;
    private double limiteSuperior;

    public Estado(Conta conta) {
        this.conta = conta;
        setLimites();
    }

    protected abstract void setLimites();

    //getters & setters
```

Padrão State

```
    public void depositar(double quantia) {
        this.conta.setSaldo(this.conta.getSaldo() + quantia);
        this.verificarAlteracaoEstado();
    }

    public void sacar(double quantia) {
        this.conta.setSaldo(this.conta.getSaldo() - quantia);
        this.verificarAlteracaoEstado();
    }

    protected abstract void verificarAlteracaoEstado();
} //Fim da classe Estado
```

Padrão State

```
// "ConcreteState"
class EstadoPrata extends Estado {
    public EstadoPrata(Conta conta) {
        super(conta);
    }

    //comportamento particular deste estado
    public void setLimites() {
        this.setLimiteInferior(0.0);
        this.setLimiteSuperior(1000.0);
    }
```

Padrão State

```
//comportamento particular deste estado: cliente paga taxa a cada saque
    realizado.
    public void sacar(double quantia) {
      this.getConta().setSaldo(this.getConta().getSaldo() - quantia -5.00);
      this.verificarAlteracaoEstado();
    }

    //depositar() herdado da superclasse e mantido
    public void verificarAlteracaoEstado() {
      if (this.getConta().getSaldo() < this.getLimiteInferior())
          this.getConta().setEstado(new EstadoVermelho(this.getConta()));
      else if (this.getConta().getSaldo() > this.getLimiteSuperior()) {
          this.getConta().setEstado(new EstadoOuro(this.getConta()));
      }
    }
} //Fim da classe EstadoPrata
```

Padrão State

```
// "ConcreteState"
class EstadoVermelho extends Estado {

        public EstadoVermelho(Conta conta) {
            super(conta);
        }

        //comportamento particular deste estado
        public void setLimites() {
            setLimiteInferior(-100.0);
            setLimiteSuperior(0.0);
        }
```

Padrão State

```
        //comportamento particular deste estado: não permite saque.
        public void sacar(double quantia) {
            System.out.println("Não existem fundos disponíveis para
              saque!");
            }

        public void verificarAlteracaoEstado() {
            if (this.getConta().getSaldo() > this.getLimiteSuperior()){
                this.getConta().setEstado(new
                  EstadoPrata(this.getConta()));
                this.getConta().getEstado().verificarAlteracaoEstado();
            }
        }
} //Fim da classe EstadoVermelho
```

Padrão State

```
// "ConcreteState"
class EstadoOuro extends Estado {

public EstadoOuro(Conta conta) {
        super(conta);
}

public void setLimites() {
        this.setLimiteInferior(1000.0);
        this.setLimiteSuperior(10000000.0);
}
```

Padrão State

```
//comportamento particular deste estado: ganha rendimento a cada
depósito.

public void depositar(double quantia) {
   this.getConta().setSaldo(this.getConta().getSaldo() + (quantia * 1.01));
}

public void verificarAlteracaoEstado() {
    if (this.getConta().getSaldo() < 0.0)
        this.getConta().setEstado(new EstadoVermelho(this.getConta()));
    else if (this.getConta().getSaldo() < this.getLimiteInferior())
        this.getConta().setEstado(new EstadoPrata(this.getConta()));

}
} //Fim da classe EstadoOuro
```

Padrão State

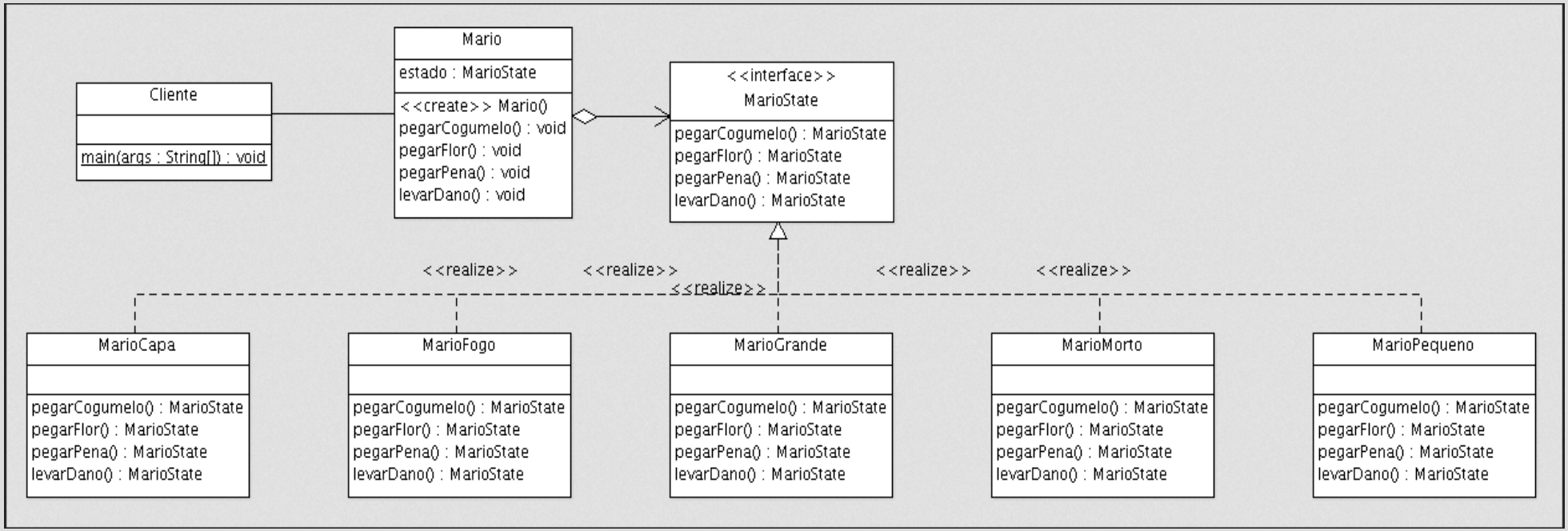
```
// Aplicação Cliente
public class Client {
    public static void main(String[] args) {
        //Abrir nova Conta
        Conta conta = new Conta("2923903");

        // Efetuar transações financeiras
        conta.depositar(500.0);
        conta.depositar(300.0);
        conta.depositar(550.0);
        conta.depositar(550.0);
        conta.sacar(2000.00);
        conta.sacar(1100.00);
    }
}
```

Padrão State

- Exemplo: Trocas de estado do Mario

  Fonte: https://**brizeno**.wordpress.com/category/padroes-de-projeto/state/



Padrão State

- 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

Padrão State

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

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

Padrão State

```
public interface MarioState {
    MarioState pegarCogumelo();

    MarioState pegarFlor();

    MarioState pegarPena();

    MarioState levarDano();
}
```

```
public class MarioPequeno implements MarioState {

    public MarioState pegarCogumelo() {
        System.out.println("Mario grande");
        return new MarioGrande();
    }

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

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

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

}
```

```
public class MarioCapa implements MarioState {

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

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

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

    public MarioState levarDano() {
        System.out.println("Mario grande");
        return new MarioGrande();
    }

}
```

```
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();
    }
}
```

Padrão State

```
public static void main() {
    Mario mario = new Mario();
    mario.pegarCogumelo();
    mario.pegarPena();
    mario.levarDano();
    mario.pegarFlor();
    mario.pegarFlor();
    mario.levarDano();
    mario.levarDano();
    mario.pegarPena();
    mario.levarDano();
    mario.levarDano();
    mario.levarDano();
}
```

Padrão State

- **Observações:**
  - Isola o comportamento de um objeto, que depende de seu estado interno.

  - A classe Context transfere aos estados uma responsabilidade.

  - A lógica de transição de estados é implementada pelos próprios estados, desobrigando o context de conhecê-la.

  - Esta lógica é dividida entre vários estados.

  - Cada estado encapsula parte desta lógica.

  - Elimina código complicado e extenso de decisão.

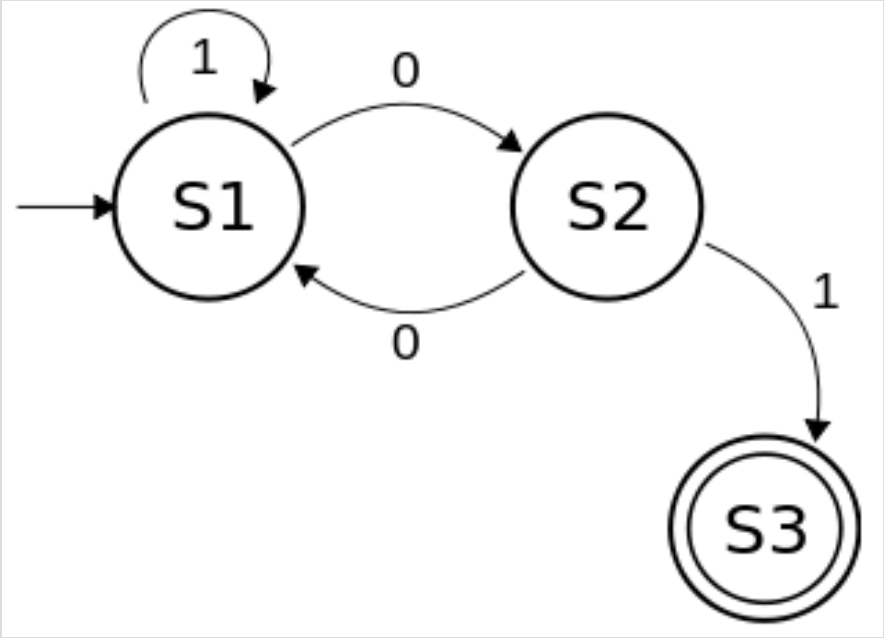
  - Com isto fica simples a inserção de novos estados e transições.

Padrão State

- O padrão elimina a necessidade de condicionais complexos e que frequentemente serão repetidos.
- Com o padrão cada “ramo” do condicional acaba se tornando um objeto,
- Trata-se cada estado como se fosse um objeto de verdade, distribuindo a complexidade dos condicionais.

Padrão State

- Desta forma, pode-se implementar facilmente qualquer tipo de máquina de estados, como os autômatos
- Exercício:



Padrão State

Aplicabilidade

Use o padrão State em um dos dois casos seguintes:

- o comportamento de um objeto depende do seu estado e ele pode mudar seu comportamento em tempo de execução, dependendo desse estado;
- operações têm comandos condicionais grandes, de várias alternativas, que dependem do estado do objeto. Esse estado é normalmente representado por uma ou mais constantes enumeradas. Freqüentemente, várias operações conterão essa mesma estrutura condicional. O padrão State coloca cada ramo do comando adicional em uma classe separada. Isto lhe permite tratar o estado do objeto como um objeto propriamente dito, que pode variar independentemente de outros objetos.

GAMMA, Erich. Padrões de projeto : soluções reutilizáveis de software orientado a objetos. Porto Alegre : Bookman, 2005.

Padrão State

- Fim
  - Go ahead!