

INF011 – Padrões de Projeto

23 – *State*

Sandro Santos Andrade
sandroandrade@ifba.edu.br

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas

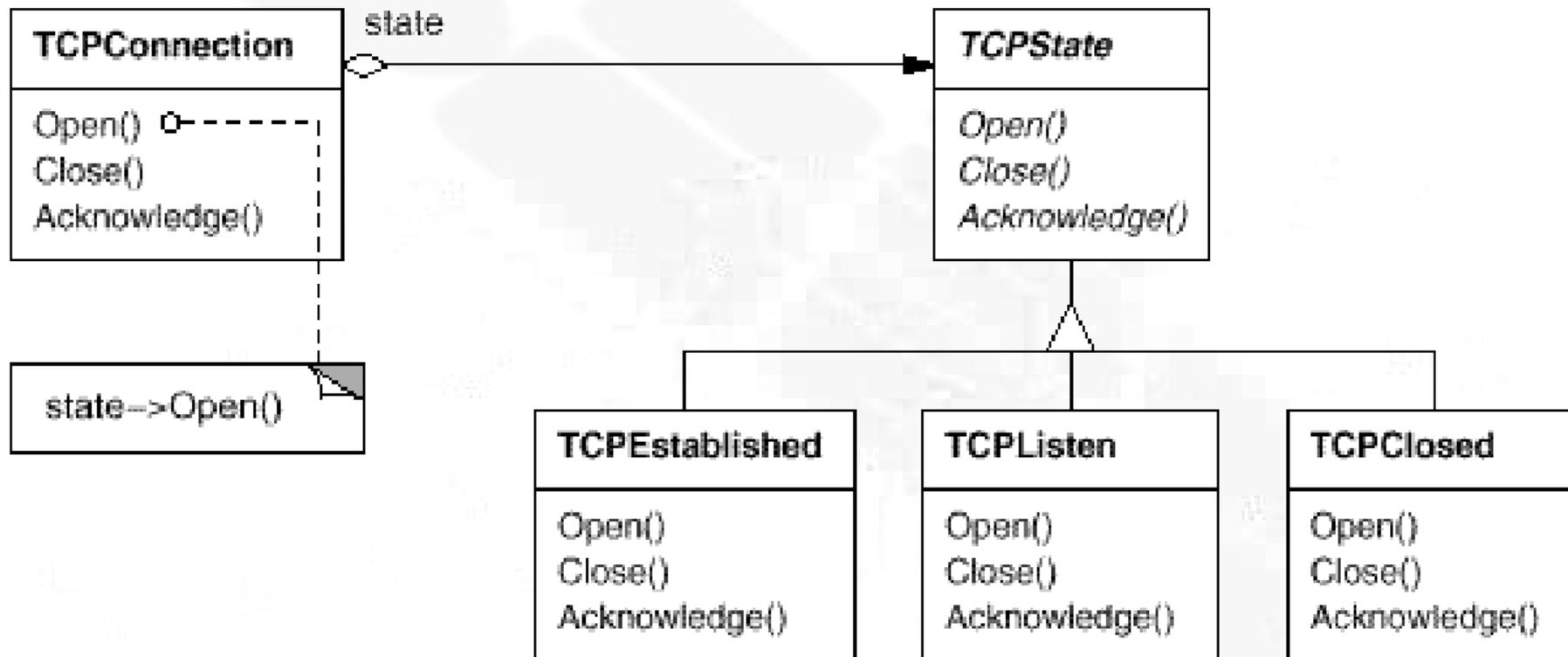


State

- Propósito:
 - Permitir que um objeto altere o seu comportamento como consequência de uma mudança no seu estado interno
- Também conhecido como: *Objects for States*
- Motivação:
 - Classe *TCPConnection* com os estados: *Established*, *Listening* e *Closed*
 - Objetos desta classe devem responder de forma diferente dependendo do seu estado atual
 - A ideia é introduzir uma classe abstrata *TCPState* para representar os estados da conexão

State

- Motivação:

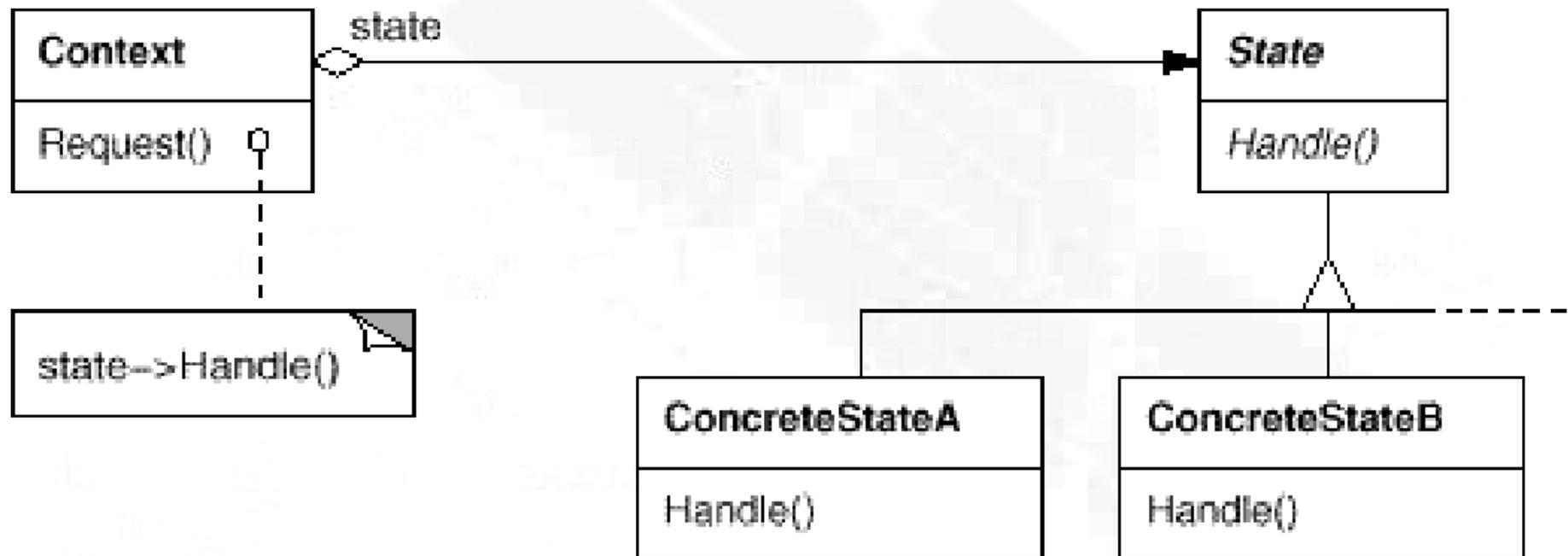


State

- Aplicabilidade:
 - Quando o comportamento do objeto depende do seu estado e este comportamento deve ser modificado em *run-time*
 - Quando os métodos possuem sentenças condicionais grandes e com várias opções. O padrão *State* coloca cada ramo da sentença condicional em uma classe separada

State

- Estrutura:



State

- Participantes:
 - *Context* (TCPConnection):
 - Define a interface de interesse dos clientes
 - Mantém uma instância do *ConcreteState* que define o estado atual do contexto
 - *State* (TCPState):
 - Define uma interface que encapsula o comportamento associado a um estado particular do contexto
 - *ConcreteState* (TCPEstablished, TCPListen, etc):
 - Cada sub-classe implementa o comportamento associado com o estado do contexto que ela representa

State

- Colaborações:
 - O *Context* delega requisições específicas de estado ao *ConcreteState* atual
 - O *Context* pode passar ele próprio como argumento para o objeto *State* atendendo a requisição
 - *Context* é a interface primária para clientes. Clientes configuram o *Context* com objetos *State*
 - Ou o *Context* ou as sub-classes *ConcreteState* podem decidir qual estado sucede outro e em quais circunstâncias

State

- Conseqüências:
 - Localiza comportamento específico de estado e particiona o comportamento de diferentes estados:
 - Todo o comportamento relacionado a um estado passa a estar em um único objeto. Pode-se facilmente adicionar novos estado e transições
 - Sem o *State* a adição de um novo estado requer a modificação de diversos métodos
 - O *State* pode aumentar o número de classes porém evita sentenças condicionais grandes ao modelar o conceito de estado de execução como uma entidade de primeira classe

State

- Conseqüências:
 - Faz com que as transições se tornem explícitas:
 - Quando o estado é representado somente por atributos internos as transições têm a forma de simples atribuições
 - Além de proteger o *Context* de estados internos inconsistentes (visto que suas transições são atômicas) o *State* torna as transições explícitas
 - Objetos *State* podem ser compartilhados:
 - Se eles não possuem variáveis de instância
 - Neste caso eles são implementados como *Flyweights*

State

- Implementação:
 - Quem define as transições entre estados ?
 - Se os critérios de transição são fixos eles podem ser implementados diretamente no *Context*
 - Entretanto, é mais flexível e apropriado deixar que as próprias sub-classes de *State* especifiquem seu estado sucessor e quando a transição ocorrerá
 - Isto requer uma interface no *Context* para que os *ConcreteState's* modifiquem seu estado explicitamente
 - Vantagem: torna fácil modificar ou estender a lógica através da definição de novos *ConcreteState's*
 - Desvantagem: um *ConcreteState* deve conhecer pelo menos outro *ConcreteState*

State

- Implementação:
 - Alternativa baseada em tabelas:
 - Utilizar uma tabela para mapear entradas em transições de estados
 - Para cada estado, a tabela mapeia toda entrada possível em um estado sucessor
 - Converte sentenças condicionais (ou ligações dinâmicas no caso do *State*) em uma tabela de *look-up*
 - Vantagem: regularidade. Pode-se mudar o critério de transição modificando os dados ao invés do código-fonte
 - Desvantagens:
 - Frequentemente menos eficiente que ligação dinâmica
 - Torna o critério de transição menos explícito
 - Não especifica as ações que devem acompanhar as transições

State

- Implementação:
 - Criando e destruindo objetos *State*
 - *Trade-off*:
 - 1) Criar o *State* somente quando necessário e destruí-lo após o uso; ou
 - 2) Criar todos os possíveis *States* antecipadamente e nunca destruí-los
 - 1) é preferível quando novos *States* puderem aparecer em *run-time* e os *Contexts* mudam de estado com pouca frequência
 - 2) é preferível quando as mudanças de estado ocorrem com muita frequência. Entretanto, é inconveniente por manter referências a todos os possíveis *States*

State

- Implementação:
 - Usando “herança dinâmica”
 - Pode-se modificar o comportamento de um objeto modificando, em *run-time*, a sua classe
 - A maioria das linguagens orientadas a objetos não suporta esta funcionalidade
 - Exceções: *Self* e outras linguagens baseadas em delegação. Estas suportam o padrão *State* nativamente

State

- Código exemplo:

```
class TCPOctetStream;
class TCPState;

class TCPConnection {
public:
    TCPConnection();

    void ActiveOpen();
    void PassiveOpen();
    void Close();
    void Send();
    void Acknowledge();
    void Synchronize();

    void ProcessOctet(TCPOctetStream*);
private:
    friend class TCPState;
    void ChangeState(TCPState*);
private:
    TCPState* _state;
};
```

State

- Código exemplo:

```
class TCPState {
public:
    virtual void Transmit(TCPConnection*, TCPOctetStream*);
    virtual void ActiveOpen(TCPConnection*);
    virtual void PassiveOpen(TCPConnection*);
    virtual void Close(TCPConnection*);
    virtual void Synchronize(TCPConnection*);
    virtual void Acknowledge(TCPConnection*);
    virtual void Send(TCPConnection*);
protected:
    void ChangeState(TCPConnection*, TCPState*);
};
```

State

- Código exemplo:

```
TCPCConnection::TCPCConnection () {
    _state = TCPClosed::Instance();
}

void TCPCConnection::ChangeState (TCPState* s) {
    _state = s;
}

void TCPCConnection::ActiveOpen () {
    _state->ActiveOpen(this);
}

void TCPCConnection::PassiveOpen () {
    _state->PassiveOpen(this);
}
```

```
void TCPCConnection::Close () {
    _state->Close(this);
}

void TCPCConnection::Acknowledge () {
    _state->Acknowledge(this);
}

void TCPCConnection::Synchronize () {
    _state->Synchronize(this);
}
```

State

- Código exemplo:

```
class TCPEstablished : public TCPState {  
public:  
    static TCPState* Instance();  
  
    virtual void Transmit(TCPConnection*, TCPOctetStream*);  
    virtual void Close(TCPConnection*);  
  
};
```

```
class TCPListen : public TCPState {  
public:  
    static TCPState* Instance();  
  
    virtual void Send(TCPConnection*);  
    // ...  
  
};
```

```
class TCPClosed : public TCPState {  
public:  
    static TCPState* Instance();  
  
    virtual void ActiveOpen(TCPConnection*);  
    virtual void PassiveOpen(TCPConnection*);  
    // ...  
  
};
```

State

- Código exemplo:

```
void TCPClosed::ActiveOpen (TCPConnection* t) {
    // send SYN, receive SYN, ACK, etc.
    ChangeState(t, TCPEstablished::Instance());
}

void TCPClosed::PassiveOpen (TCPConnection* t) {
    ChangeState(t, TCPListen::Instance());
}

void TCPEstablished::Close (TCPConnection* t) {
    // send FIN, receive ACK of FIN
    ChangeState(t, TCPListen::Instance());
}
```

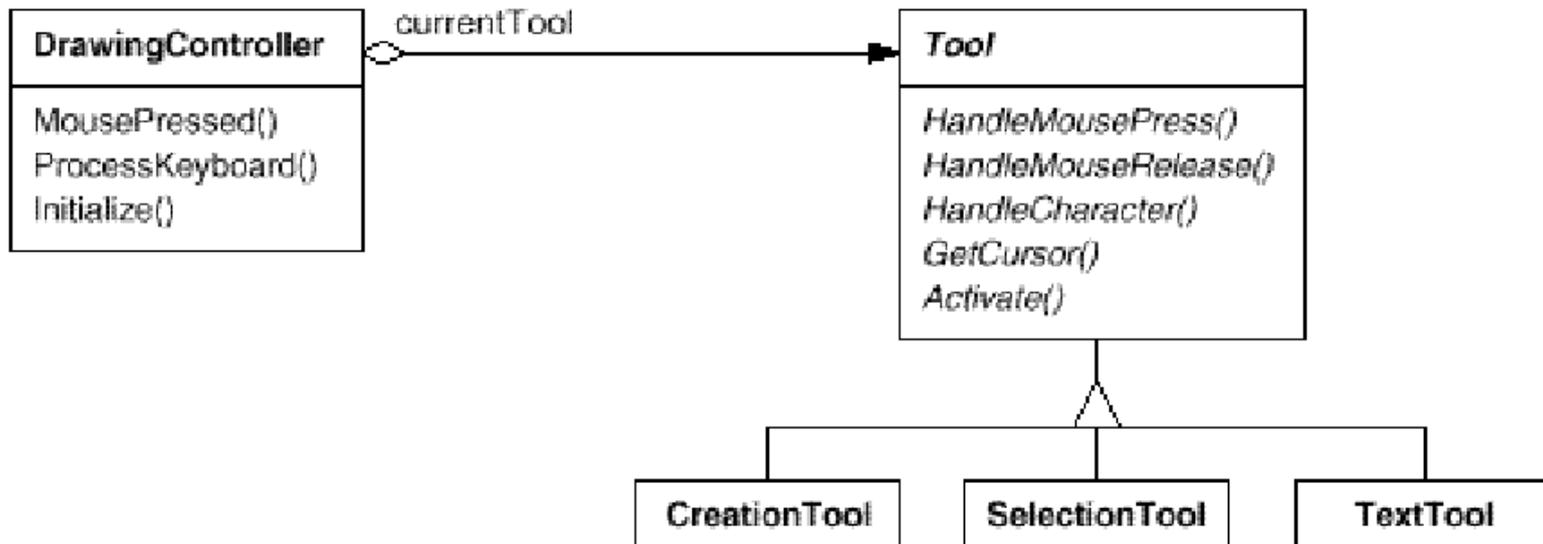
State

- Código exemplo:

```
void TCPEstablished::Transmit ( TCPConnection* t, TCPOctetStream* o ) {  
    t->ProcessOctet(o);  
}  
  
void TCPListen::Send (TCPConnection* t) {  
    // send SYN, receive SYN, ACK, etc.  
    ChangeState(t, TCPEstablished::Instance());  
}
```

State

- Usos conhecidos:
 - Protocolos de conexão TCP
 - Diferentes ferramentas para a realização de diferentes operações em editores gráficos:
 - *HotDraw*
 - *UniDraw*



State

- Padrões relacionados:
 - *States* compartilhados são geralmente implementados como *Flyweights*
 - Objetos *State* frequentemente são também *Singletons*

INF011 – Padrões de Projeto

23 – *State*

Sandro Santos Andrade
sandroandrade@ifba.edu.br

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas

