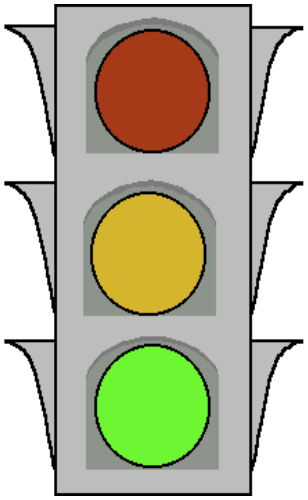
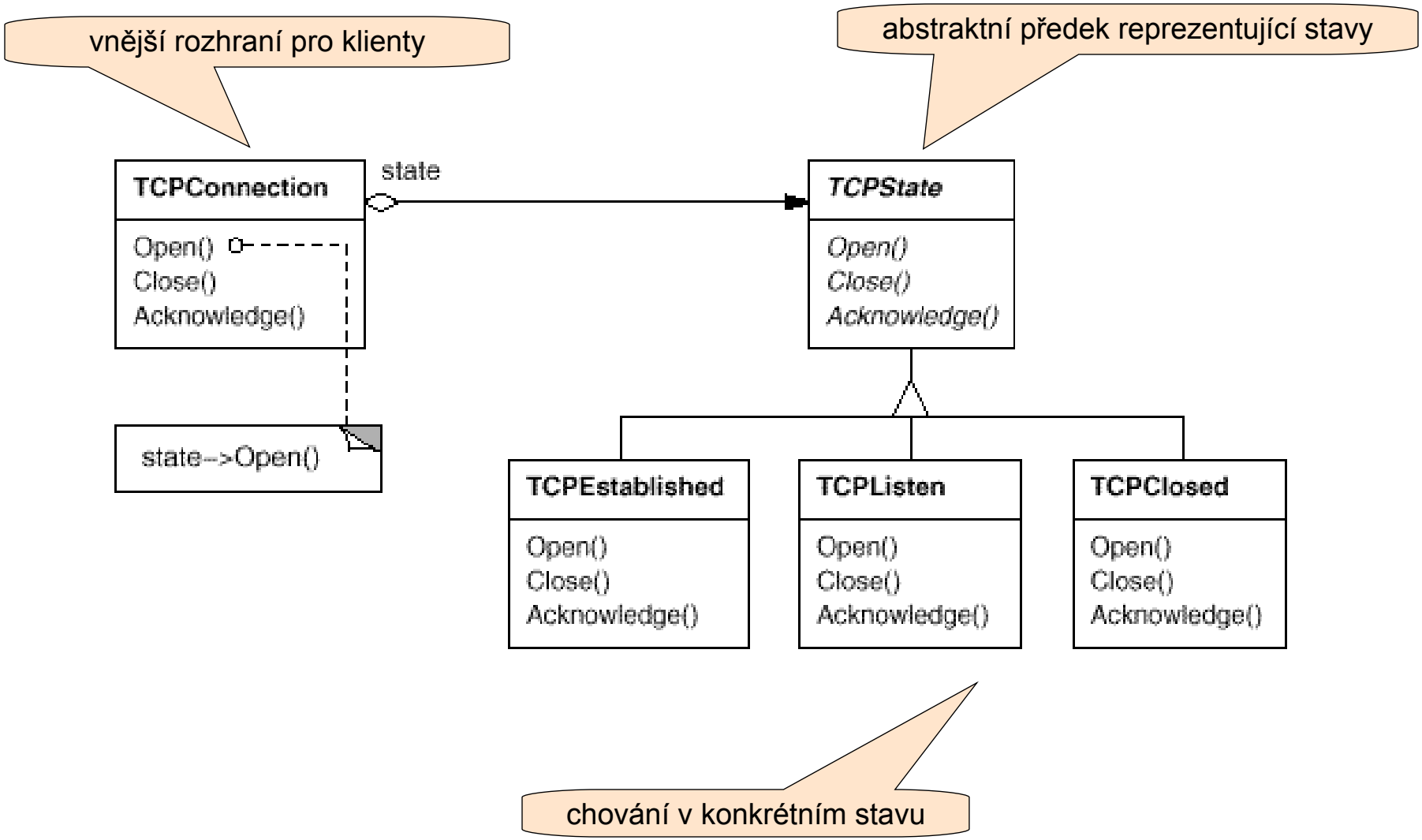
State



State

- **Známý jako**
  - Stav, Object for States
- **Účel**
  - umožňuje objektu měnit svoje chování v závislosti na stavu
  - objekt „mění svou třídu“
- **Použitelnost**
  - chování objektu závisí na jeho stavu, který se mění za běhu
  - metody vykonávající jednotlivé funkce objektu obsahují větvení v závislosti na nějaké sadě výčtových proměnných

State –TCPConnection – motivační příklad



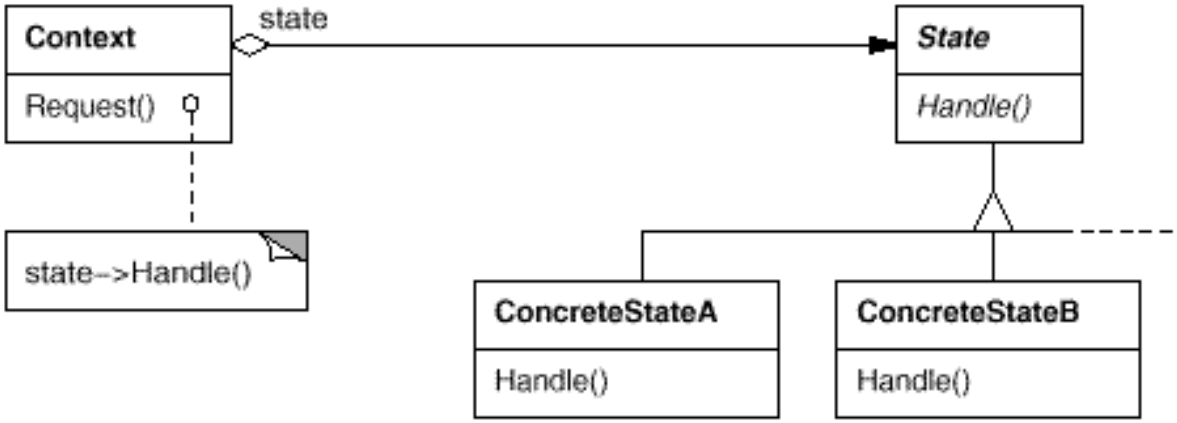
State – když ho neznám…

- **Stav = výčtový typ**
- **Stavy rozlišené podmínkou**
  - Málo stavů
    - if, else if, else
  - Více stavů
    - switch
- **Více stavově závislých metod**
  - Stačí jedna => Copy + Paste

- **Přidání stavu**
  - Upravit všechny metody =>

State – struktura

- Struktura



- Účastníci
  - Context (*TCPConnection)*
    - definuje rozhraní pro klienty
    - obsahuje instanci podtřídy ConcreteState – určuje současný stav
  - State (*TCPState*)
    - definuje rozhraní objektů reprezentujících jednotlivé stavy Contextu
  - ConcreteStateA, B, ... *(TCPEstablished, TCPListen, TCPClosed)*
    - implementuje konkrétní chování v daném stavu

State – použití

- **Použití**
  - Context deleguje požadavky na instanci stavu ConcreteState
  - Context definuje rozhraní pro klienty
    - ti mohou inicializovat Context konkrétním stavem
  - Context může předávat referenci na sebe sama při volání metod stavu
    - jednotlivé stavy mohou přistupovat přímo ke kontextu
  - Context mění svůj stav sám, nebo to dělají přímo objekty ConcreteState

State – důsledky

- **Chování asociované s konkrétním stavem se nachází v jednom objektu**
  - nové stavy se jednoduše přidávají definováním potomka State
  - místo větvení se přechody mezi stavy slučují do samostatných tříd
    - chování není roztroušeno po Contextu
  - může vznikat spousta stavů lišících se v drobnostech
- **Explicitní změna stavu**
  - vytváření samostatných objektů pro různé stavy (explicitní)
    - objekty State chrání Context před inkonzistencí interních stavů
      - přechod na jiný stav je atomický (z pohledu Contextu) – změna jedné proměnné
- **Sdílení objektů State**
  - instance objektů nemají proměnné, stav je reprezentovaný pouze jejich typem
  - v podstatě nemají žádný vlastní stav, jenom chování – Flyweight

State – implementace

- **Kdo mění stav?**
  - Context
    - pevná kritéria
  - ConcreteState
    - Context musí obsahovat interface pro změnu stavu
    - flexibilnější – stačí přidat novou třídu a navázat ji na konkrétních místech
    - vznik závislostí – jednotlivé stavy o sobě musejí vědět
- **Tabulkou řízený přístup**
  - (Vstup, Stav) -> (Funkce, Stav)
  - není třeba měnit kód, ale jen tabulku
  - obvykle pomalejší, obsahuje jen změny stavů bez dalších možností

State – implementace (pokračování)

- **Vytváření a rušení instancí stavu**
  - vytváření podle potřeby
    - předem neznámé a nepříliš časté změny stavu
  - vytváření předem
    - časté změny stavu
    - větší spotřeba paměti

State – konkrétní implementace

- **Context – TCPConnection**
  - veřejné rozhraní pro klienty
  - protected změna stavu, přátelská třída TCPState

```
class TCPOctetStream;
class TCPState;

class TCPConnection {
public:
    TCPConnection();
    TCPConnection(TCPState*);

    void ActiveOpen();
    void PassiveOpen();
    void Close();

    void Send();
    void Acknowledge();
    void Synchronize();

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

počáteční stav

interface pro změnu stavu

instance stavu

State – konkrétní implementace (pokračování)

- **TCPConnection**

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

TCPConnection::TCPConnection(TCPState* state) {
    _state = state;
}

void TCPConnection::ChangeState (TCPState* state) {
    _state = state;
}

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

// ...
```

Počáteční stav – bezparametrický konstruktor

Context deleguje konkrétní operace na aktuální instanci stavu

State – konkrétní implementace (pokračování)

- **State – TCPState**
  - abstraktní třída, předek všech stavů
  - interface pro chování

```
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* t, TCPState* s) {
        t->ChangeState(s);
    };
};
```

jednotlivé operace

společná metoda pro změnu stavu

- **TCPState**

```
void TCPState::Transmit (TCPConnection*, TCPOctetStream*) { }
void TCPState::ActiveOpen (TCPConnection*) { }
void TCPState::Close (TCPConnection*) { }

// ...
```

State – konkrétní implementace (pokračování)

- **TCPClosed – konkrétní stav**

```
class TCPClosed : public TCPState {
public:
    static TCPState* Instance();

    virtual void ActiveOpen(TCPConnection*);
    virtual void PassiveOpen(TCPConnection*);

    // ...
};

void TCPClosed::ActiveOpen (TCPConnection* t) {
    // send SYN, receive SYN, ACK, etc.

    ChangeState(t, TCPEstablished::Instance());
}

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

statická metoda Instance – stavy implementovány jako Singletony

State – souhrn, související NV, použití

- **Souhrn**
  - Měníme chování objektu v závislosti na vnitřním stavu
- **Související NV**
  - Flyweight
    - sdílené stavy bez vlastních vnitřních dat
  - Singleton
    - stavy implementovány jako Singletony
- **Použití návrhového vzoru State**
  - grafické editory
    - chování editoru se mění s aktivním nástrojem
      - kreslicí nástroj – kreslíme tvary
      - výběrový nástroj – vybíráme nakreslené tvary
    - můžeme definovat abstraktní třídu Tool – potomci implementují specifické chování
    - při změně nástroje se mění chování editoru