INF011 – Padrões de Projeto

16 – *Chain of Responsibility*

Sandro Santos Andrade
sandroandrade@ifba.edu.br

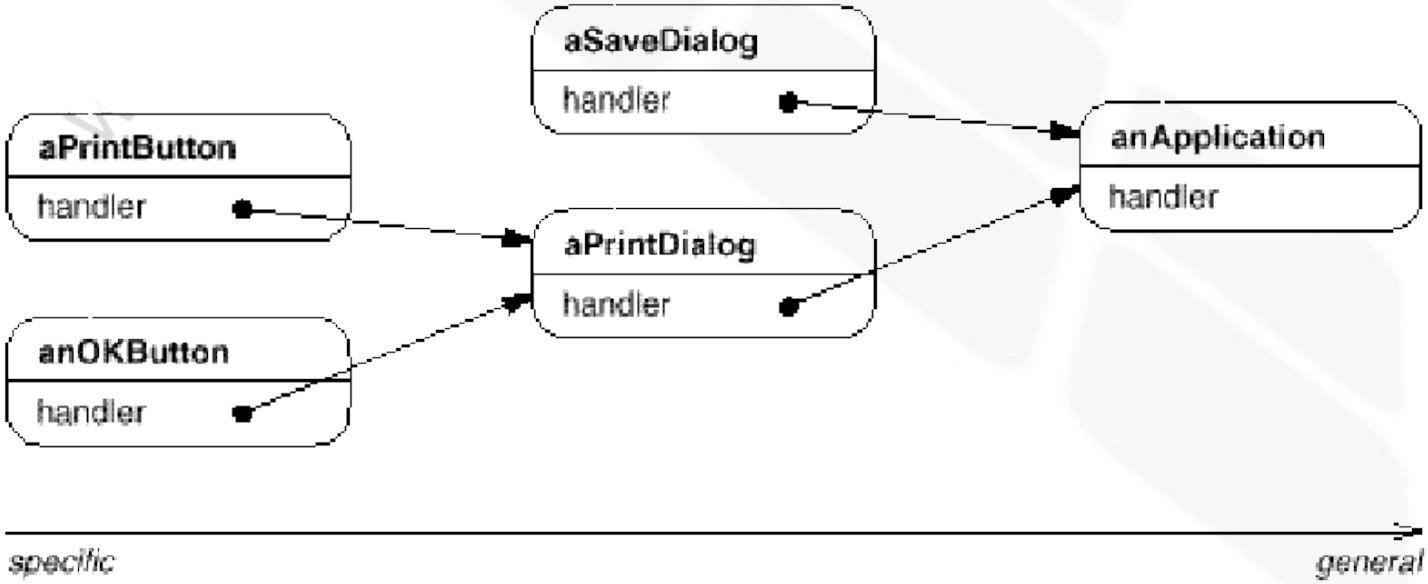
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas

Chain of Responsibility

- Propósito:
  - Evitar o acoplamento entre o emissor de uma requisição e seu receptor, dando a mais de um objeto a chance de processar a requisição
- Motivação:
  - *Help* sensível ao contexto em aplicações gráficas
  - A mensagem de *help* depende da parte da interface que foi clicada e do seu contexto
  - Se não existir mensagem de *help* para um botão, por exemplo, exibe-se a mensagem de help genérica do *dialog* que o contém

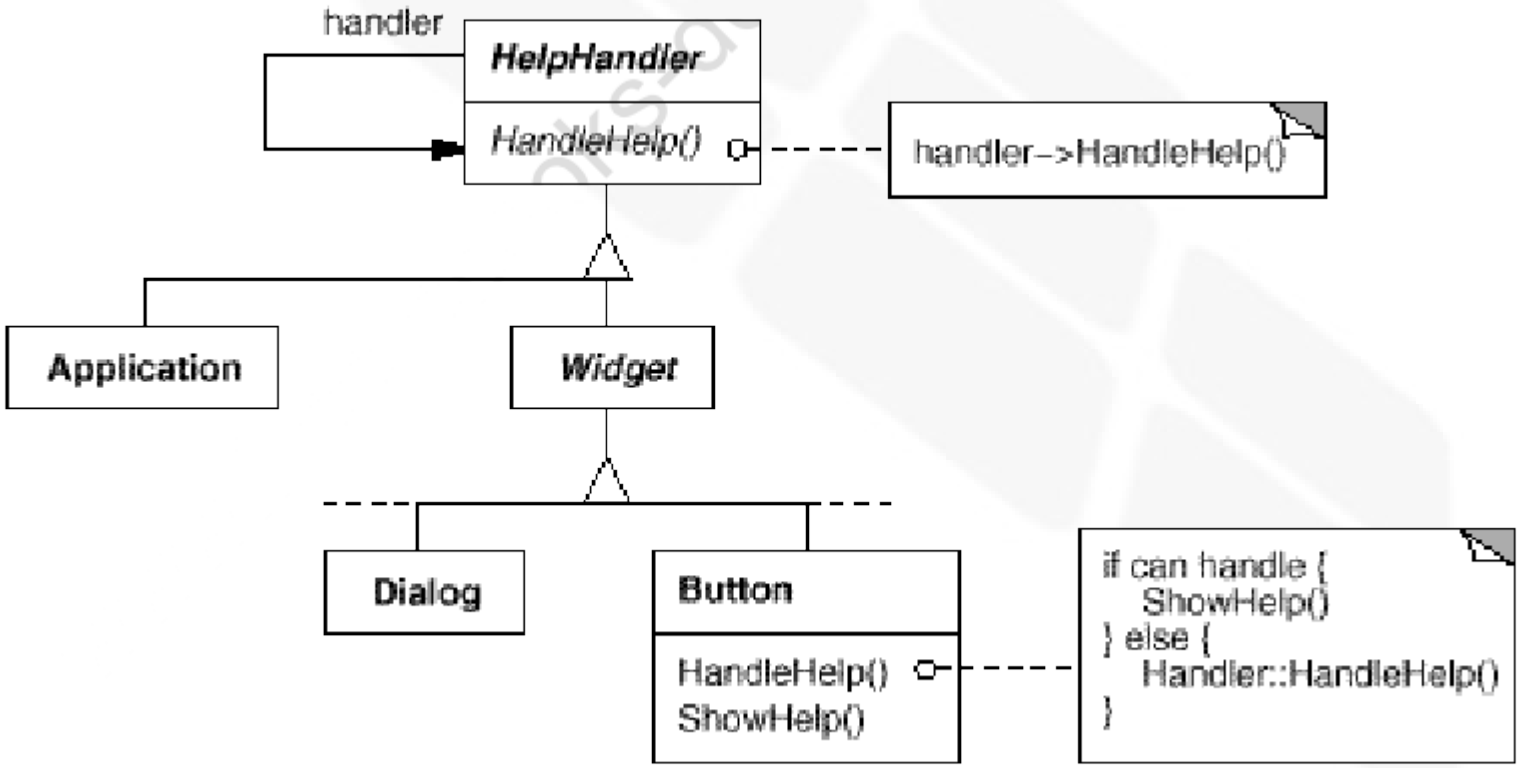
Chain of Responsibility

- Motivação:
  - O objeto que efetivamente apresenta o *help* não é explicitamente conhecido pelo objeto que solicitou ajuda
  - O cliente que fez a requisição não tem referência direta ao objeto que a processou



Chain of Responsibility

- Motivação:
  - Para garantir que os receptores sejam implícitos todos os objetos da cadeia compartilham uma mesma interface

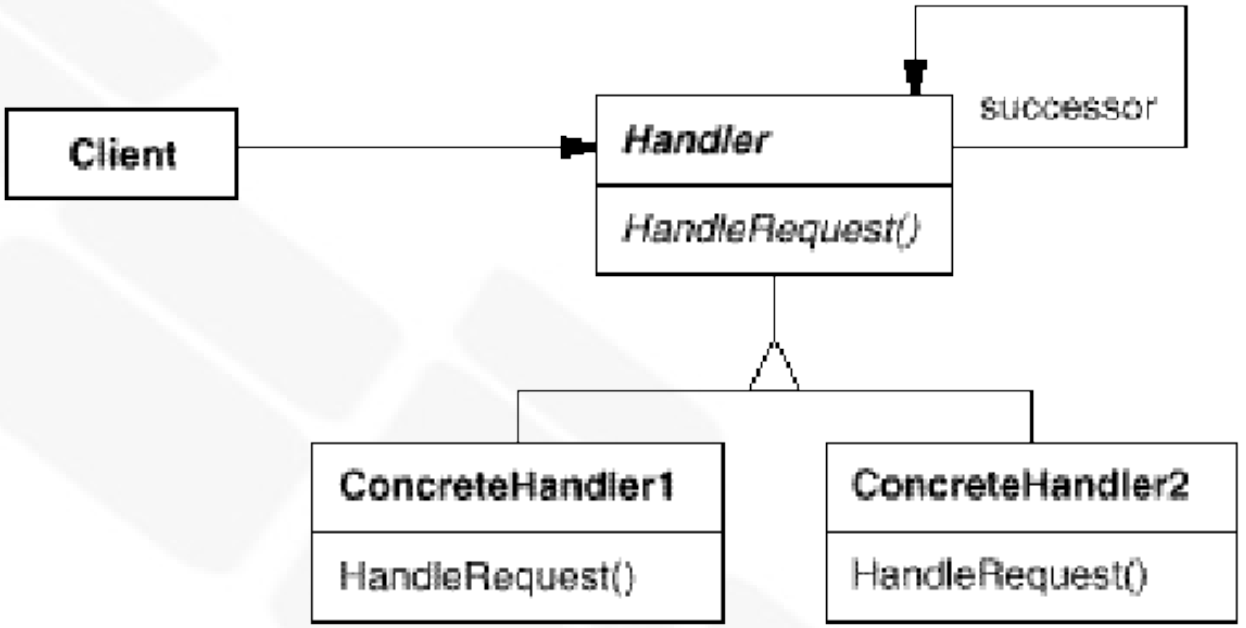


Chain of Responsibility

- Aplicabilidade:
  - Mais de um objeto pode atender uma requisição, porém este objeto atendente não é conhecido *a priori*
  - Deseja-se emitir uma requisição para um objeto dentre vários sem especificar o receptor explicitamente
  - O conjunto de objetos que podem atender à requisição deve ser especificado dinamicamente

Chain of Responsibility

- Estrutura:



Chain of Responsibility

- Participantes:
  - *Handler* (HelpHandler):
    - Define uma interface para atender as requisições
    - (opcional) Implementa a ligação ao sucessor
  - *ConcreteHandler* (PrintButton, PrintDialog):
    - Atende a requisição pela qual é responsável
    - Pode acessar o seu sucessor
    - Atende a requisição, quando possível, ou a repassa ao seu sucessor
  - *Client*:
    - Inicia a requisição a um *ConcreteHandler* da cadeia

Chain of Responsibility

- Colaborações:
  - Quando um cliente emite uma requisição ela é propagada ao longo da cadeia até que um objeto *ConcreteHandler* assume a responsabilidade do atendimento da requisição

Chain of Responsibility

- Consequências:
  - Reduz o acoplamento:
    - Um objeto não precisa conhecer qual outro objeto atende a requisição
    - O emissor e o receptor não se conhecem explicitamente
    - Um objeto na cadeia não possui conhecimento da estrutural atual da cadeia
    - Simplifica as inter-conexões entre objetos. Ao invés de manter referências para todos os receptores candidatos mantém-se uma única referência para o seu sucessor

Chain of Responsibility

- Consequências:
  - Maior flexibilidade na atribuição de responsibilidades a objetos:
    - Pode-se adicionar ou modificar as responsabilidades de atendimento de uma requisição através de modificações na cadeia em *run-time*
    - Pode-se utilizar *subclassing* para especializar os *handlers* estaticamente
  - O recebimento não é garantido:
    - A requisição pode ser repassada até o fim da cadeia sem receber nenhum tratamento específico
    - A requisição pode também não ser tratada se a cadeia estiver configurada de forma inapropriada

Chain of Responsibility

- Implementação:
  - Implementando a cadeia de sucessores:
    - Definir novos *links* ou utilizar aqueles possivelmente existentes ?
    - Referências ao *parent* em estruturas parte-todo, como em uma estrutura de *widgets*, possivelmente já implementam a cadeia de alguma forma (*Composite*)

Chain of Responsibility

- Implementação:
  - Conectando os sucessores:
    - Se não existem referências pré-definidas o *Handler* define a interface para requisições **e** armazena a referência para o sucessor
    - Pode-se, neste caso, ter uma implementação *default*

Chain of Responsibility

- Implementação:
  - Conectando os sucessores:

```
class HelpHandler {
public:
    HelpHandler(HelpHandler* s) : _successor(s) { }
    virtual void HandleHelp();
private:
    HelpHandler* _successor;
};

void HelpHandler::HandleHelp () {
    if (_successor) {
        _successor->HandleHelp();
    }
}
```

Chain of Responsibility

- Implementação:
  - Representando as requisições:
    1) Operação *hard-coded*: conveniente e seguro
    2) *Request Code* como parâmetro da função *handler*: parâmetros devem ser empacotados e desempacotados manualmente – inseguro
    3) Objetos *request* que encapsulam os parâmetros: classe *Request* e derivados

Chain of Responsibility

- Implementação:
  - Representando as requisições:

```
class ExtendedHandler : public Handler {
public:
    virtual void HandleRequest(Request* theRequest);
    // ...
};

void ExtendedHandler::HandleRequest (Request* theRequest) {
    switch (theRequest->GetKind()) {
    case Preview:
        // handle the Preview request
        break;

    default:
        // let Handler handle other requests
        Handler::HandleRequest(theRequest);
    }
}
```

```
void Handler::HandleRequest (Request* theRequest) {
    switch (theRequest->GetKind()) {
    case Help:
        // cast argument to appropriate type
        HandleHelp((HelpRequest*) theRequest);
        break;

    case Print:
        HandlePrint((PrintRequest*) theRequest);
        // ...
        break;

    default:
        // ...
        break;
    }
}
```

Chain of Responsibility

- Código exemplo:

```
typedef int Topic;
const Topic NO_HELP_TOPIC = -1;

class HelpHandler {
public:
    HelpHandler(HelpHandler* = 0, Topic = NO_HELP_TOPIC);
    virtual bool HasHelp();
    virtual void SetHandler(HelpHandler*, Topic);
    virtual void HandleHelp();
private:
    HelpHandler* _successor;
    Topic _topic;
};
```

```
HelpHandler::HelpHandler (
    HelpHandler* h, Topic t
) : _successor(h), _topic(t) { }

bool HelpHandler::HasHelp () {
    return _topic != NO_HELP_TOPIC;
}

void HelpHandler::HandleHelp () {
    if (_successor != 0) {
        _successor->HandleHelp();
    }
}
```

Chain of Responsibility

- Código exemplo:

```
class Widget : public HelpHandler {
protected:
    Widget(Widget* parent, Topic t = NO_HELP_TOPIC);
private:
    Widget* _parent;
};

Widget::Widget (Widget* w, Topic t) : HelpHandler(w, t) {
    _parent = w;
}
```

```
class Button : public Widget {
public:
    Button(Widget* d, Topic t = NO_HELP_TOPIC);

    virtual void HandleHelp();
    // Widget operations that Button overrides...
};
```

Chain of Responsibility

- Código exemplo:

```
Button::Button (Widget* h, Topic t) : Widget(h, t) { }

void Button::HandleHelp () {
    if (HasHelp()) {
        // offer help on the button
    } else {
        HelpHandler::HandleHelp();
    }
}
```

Chain of Responsibility

- Código exemplo:

```
class Dialog : public Widget {
public:
    Dialog(HelpHandler* h, Topic t = NO_HELP_TOPIC);
    virtual void HandleHelp();

    // Widget operations that Dialog overrides...
    // ...
};

Dialog::Dialog (HelpHandler* h,  Topic t) : Widget(0) {
    SetHandler(h, t);
}

void Dialog::HandleHelp () {
    if (HasHelp()) {
        // offer help on the dialog
    } else {
        HelpHandler::HandleHelp();
    }
}
```

Chain of Responsibility

- Código exemplo:

```
class Application : public HelpHandler {
public:
    Application(Topic t) : HelpHandler(0, t) { }

    virtual void HandleHelp();
    // application-specific operations...
};

void Application::HandleHelp () {
    // show a list of help topics
}
```

```
const Topic PRINT_TOPIC = 1;
const Topic PAPER_ORIENTATION_TOPIC = 2;
const Topic APPLICATION_TOPIC = 3;

Application* application = new Application(APPLICATION_TOPIC);
Dialog* dialog = new Dialog(application, PRINT_TOPIC);
Button* button = new Button(dialog, PAPER_ORIENTATION_TOPIC);
```

```
button->HandleHelp();
```

Chain of Responsibility

- Usos conhecidos:
  - Tratamento de eventos em GUIs: MacApp, ET++, TCL, NeXT, Qt4
  - *Unidraw*

Chain of Responsibility

- Padrões relacionados:
  - Frequentemente utilizado em conjunto com o *Composite*, onde o componente pai atua como sucessor

INF011 – Padrões de Projeto

16 – *Chain of Responsibility*

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