INF011 – Padrões de Projeto

13 – *Facade*

Sandro Santos Andrade
sandroandrade@ifba.edu.br

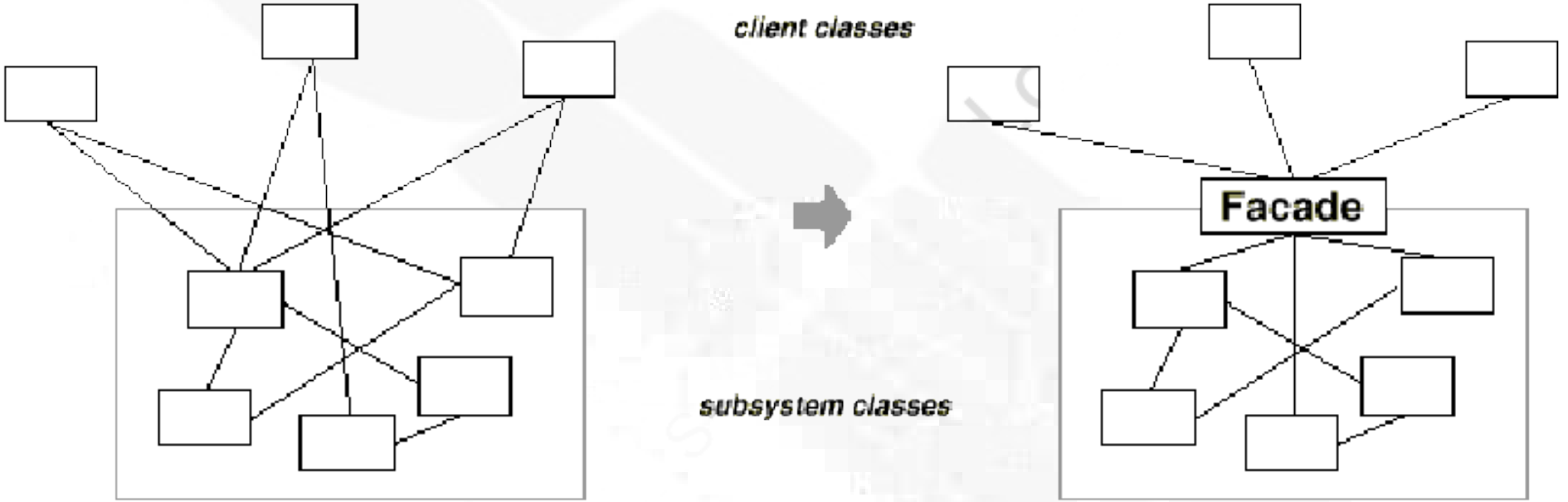
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas

Facade

- Propósito:
  - Disponibilizar uma interface unificada para um conjunto de interfaces de um sub-sistema
- Motivação:
  - Estruturar um sistema em sub-sistemas ajuda a reduzir complexidade
  - Geralmente deseja-se minimizar as comunicações e dependências entre sub-sistemas
  - O *Facade* pode ser utilizado para este objetivo

Facade

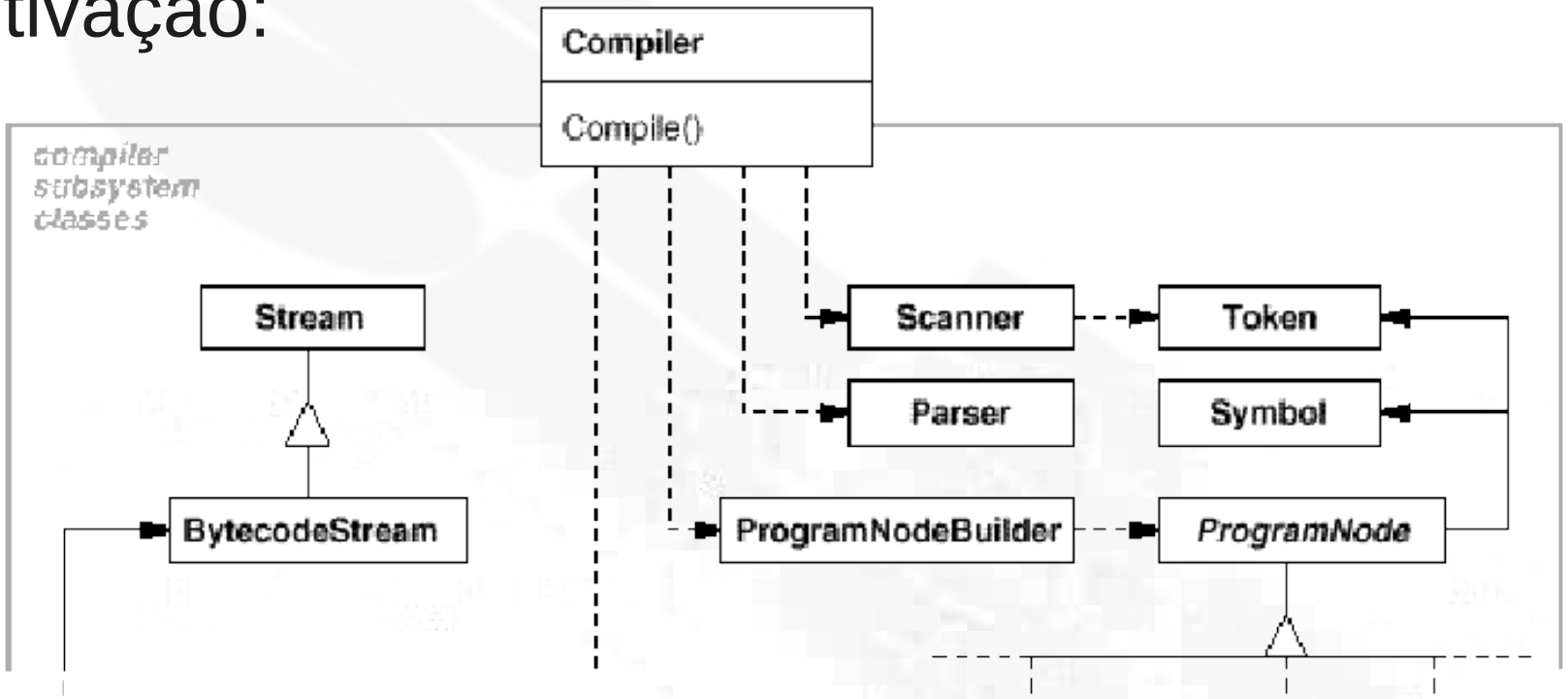
- Motivação:



  - Ex: classes de um sub-sistema de compilação: *Scanner*, *Parser*, *ProgramNode* etc
  - Geralmente os clientes desejam realizar todo o processo de compilação

Facade

- Motivação:



Facade

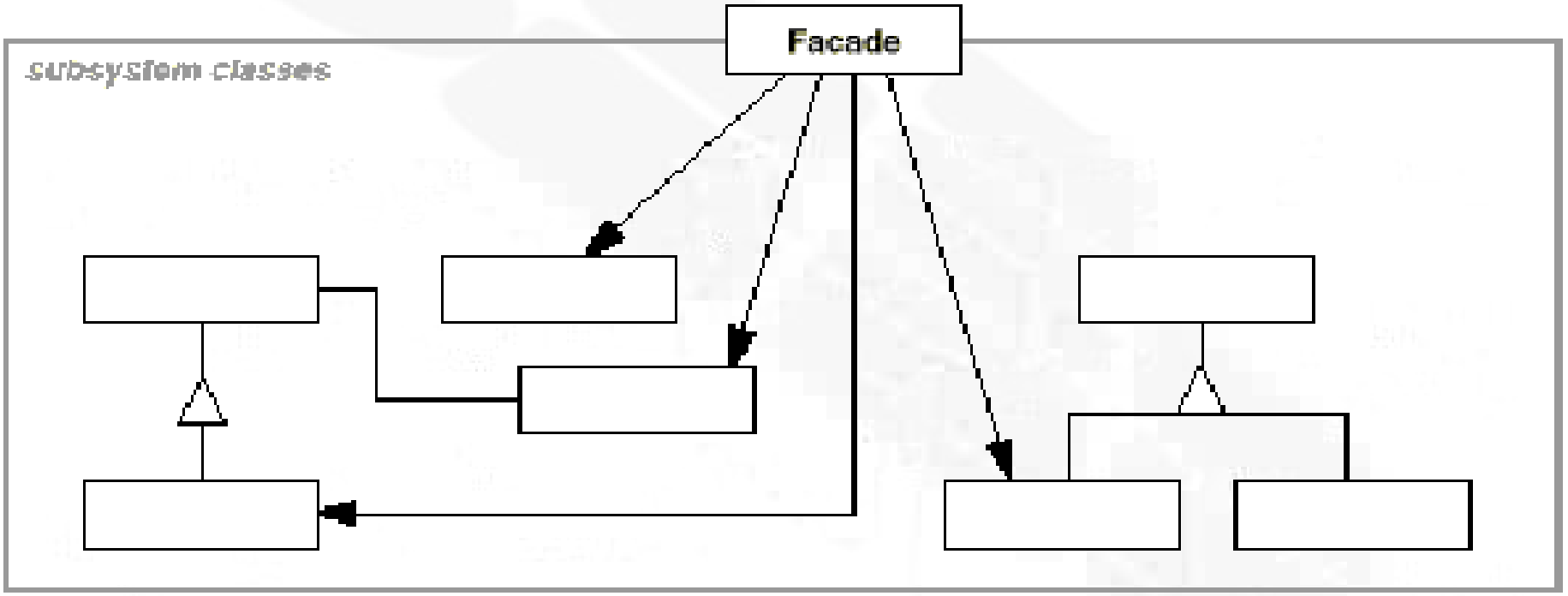
- Aplicabilidade:
  - Deseja-se disponibilizar uma forma de acesso (interface) simples a um sub-sistema complexo:
    - À medida em que evoluem e utilizam mais padrões de projeto, os sistemas passam a ser formados por um número maior de classes, geralmente pequenas
    - Isso torna o sistema mais reutilizável e fácil de configurar mas também o torna mais difícil de ser utilizado por clientes que não necessitam configurá-lo
    - O *Facade* disponibiliza uma visão *default* simples do sistema, suficiente para a maioria dos clientes
    - Somente aqueles clientes que precisam de uma maior capacidade de configuração irão acessar o sub-sistema sem utilizar o *Facade*

Facade

- Aplicabilidade:
  - Existe muitas dependências entre os clientes e as classes de implementação de uma abstração:
    - O *Facade* desacopla o sub-sistema dos clientes e também de outros sub-sistemas
    - Promove a independência e portabilidade do sub-sistema
  - Deseja-se definir um sistema em camadas:
    - O *Facade* define um *entry-point* em cada sub-sistema (nível)
    - Dependências entre sub-sistemas são simplificadas se eles se comunicarem somente através dos seus *Facades*

Facade

- Estrutura:



Facade

- Participantes:
  - *Facade* (Compiler):
    - Conhece quais classes do sub-sistema irão atender a requisição
    - Delega as requisições do cliente aos objetos apropriados no sub-sistema
  - *SubSystem Classes* (Scanner, Parser, ProgramNode):
    - Implementam as funcionalidades do sub-sistema
    - Realizam trabalhos solicitados pelo objeto *Facade*
    - Não conhecem o *Facade*, ou seja, não possuem referências para ele

Facade

- Colaborações:
  - Clientes se comunicam com o sub-sistema através de requisições enviadas ao *Facade*, que as repassa para os objetos do sub-sistema apropriados
  - Embora sejam os objetos do sub-sistema que realizem o trabalho o *Facade* pode precisar realizar algum processamento próprio para traduzir sua interface nas interfaces dos objetos do sub-sistema
  - Clientes que utilizam o *Facade* não têm acesso direto aos objetos do sub-sistema

Facade

- Consequências:
  - Isola os clientes dos componentes do sub-sistema, reduzindo o número de objetos com os quais o cliente precisa lidar e tornando o sub-sistema mais fácil de usar
  - Promove fraco acoplamento entre o sub-sistema e seus clientes:
    - Componentes de um sub-sistema geralmente são fortemente acoplados
    - Com o *Facade* pode-se variar os componentes do sub-sistema sem afetar seus clientes
    - Pode eliminar dependências complexas ou circulares
    - Minimiza recompilações quando as classes do sub-sistema mudarem
  - Não impede que aplicações utilizem diretamente as classes do sub-sistema se assim desejarem
    - Pode-se escolher entre facilidade de uso ou generalidade

Facade

- Implementação:
  - Reduzindo acoplamento entre o cliente e o sub-sistema:
    - Pode-se reduzir ainda mais o acoplamento se o *Facade* for uma classe abstrata com classes concretas para diferentes implementações do sub-sistema. Os clientes não conheceriam qual implementação do sistema estaria sendo utilizada
    - Uma alternativa à herança de interface é configurar o *Facade* com diferentes objetos do sub-sistema. Para mod tÀd en nda nãoj s F ç b dara

Facade

- Código exemplo:

```
class Scanner {
public:
    Scanner(istream&);
    virtual ~Scanner();

    virtual Token& Scan();
private:
    istream& _inputStream;
};
```

```
class Parser {
public:
    Parser();
    virtual ~Parser();

    virtual void Parse(Scanner&, ProgramNodeBuilder&);
};
```

Facade

- Código exemplo:

```
class ProgramNode {
public:
    // program node manipulation
    virtual void GetSourcePosition(int& line, int& index);
    // ...

    // child manipulation
    virtual void Add(ProgramNode*);
    virtual void Remove(ProgramNode*);
    // ...

    virtual void Traverse(CodeGenerator&);
protected:
    ProgramNode();
};
```

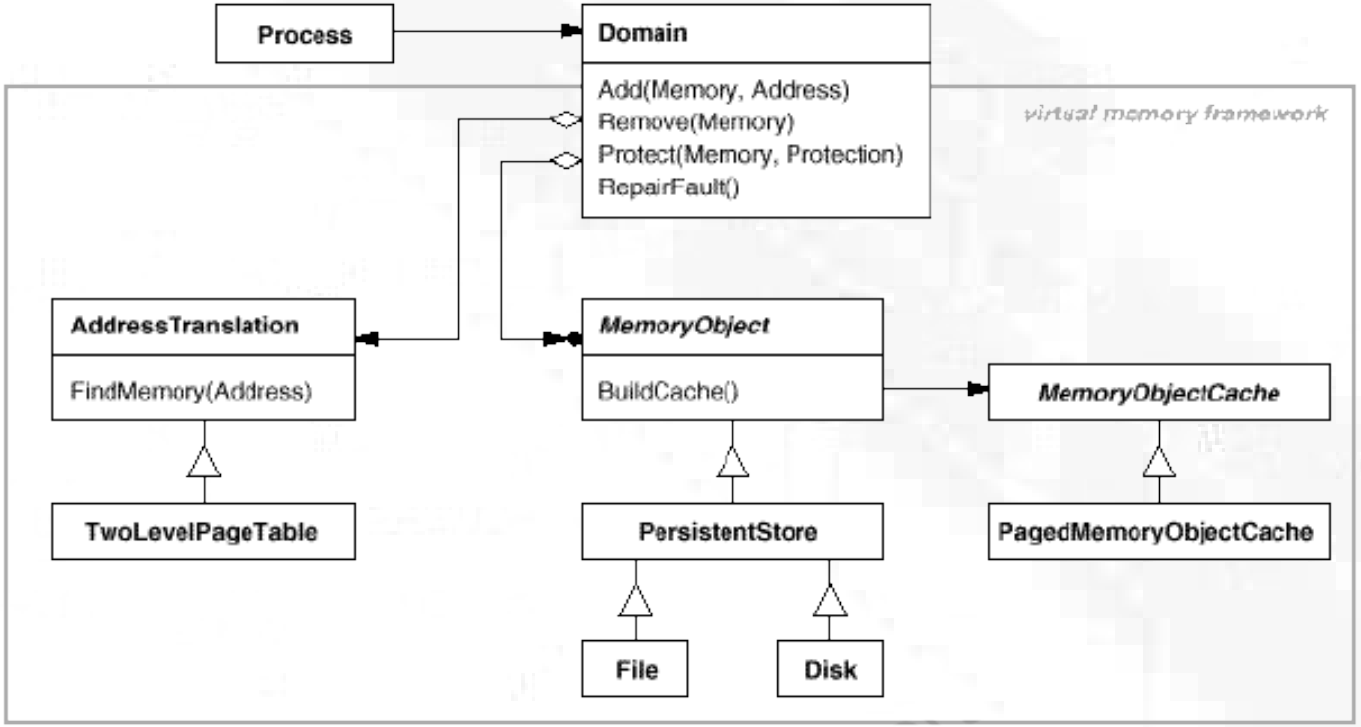
Facade

- Código exemplo:

```
class Compiler {
public:
    Compiler();

    virtual void Compile(istream&, BytecodeStream&);
};

void Compiler::Compile (
    istream& input, BytecodeStream& output
) {
    Scanner scanner(input);
    ProgramNodeBuilder builder;
    Parser parser;

    parser.Parse(scanner, builder);

    RISCCodeGenerator generator(output);
    ProgramNode* parseTree = builder.GetRootNode();
    parseTree->Traverse(generator);
}
```

Facade

- Usos conhecidos:
  - ET++
  - *Choices Operating System*



INF011 – Padrões de Projeto

13 – *Facade*

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