INF011 – Padrões de Projeto

03 – *Abstract Factory*

Sandro Santos Andrade
sandroandrade@ifba.edu.br

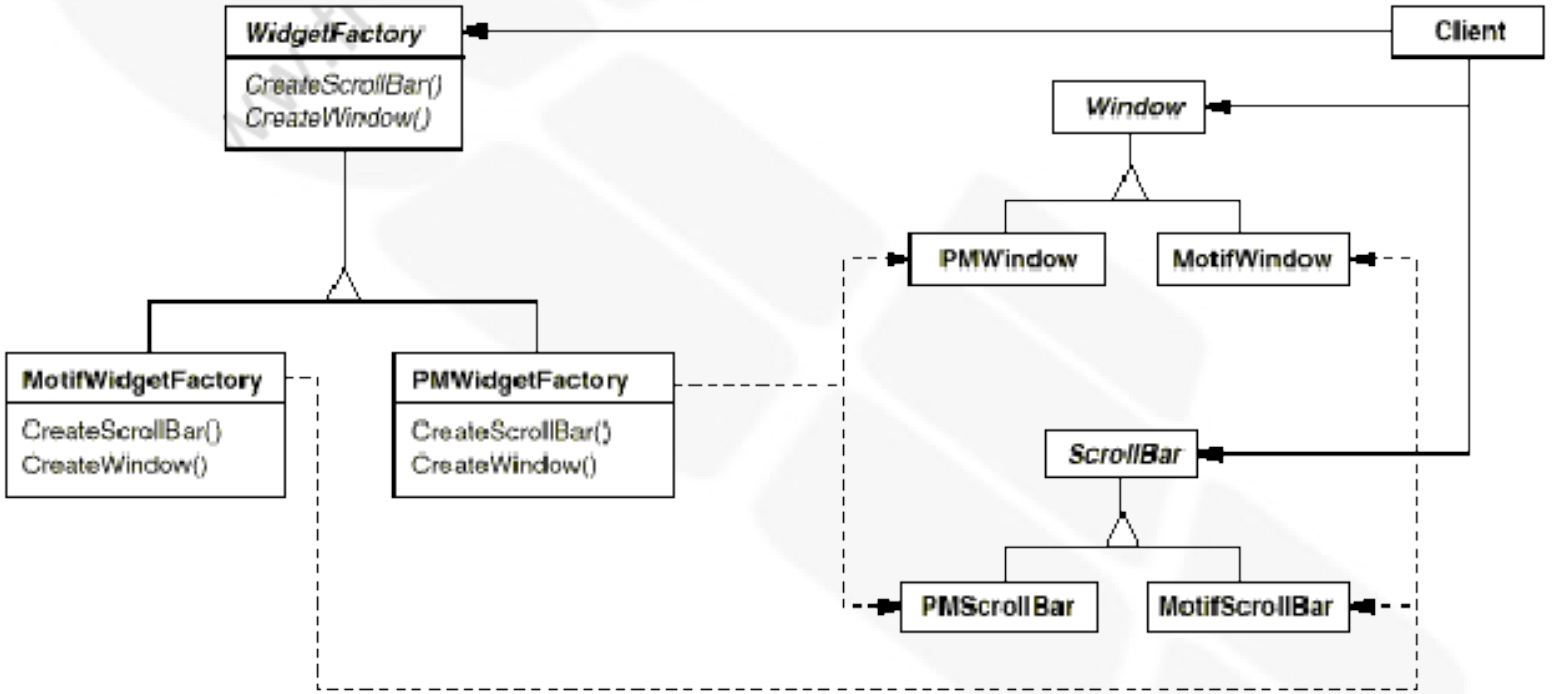
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas

Abstract Factory

- Propósito:
  - Disponibilizar uma interface para a criação de famílias de objetos dependentes ou relacionados sem especificar suas classes concretas
- Também conhecido como: *Kit*
- Motivação:
  - GUI com suporte a múltiplos *look-and-feel*. A aplicação não deve ter um *look-and-feel* particular *hard-coded*
  - O *Abstract Factory* define uma classe abstrata *WidgetFactory* com interfaces para a criação de cada tipo básico de *widget*
  - Define também uma classe abstrata para cada *widget*

Abstract Factory

- Motivação:



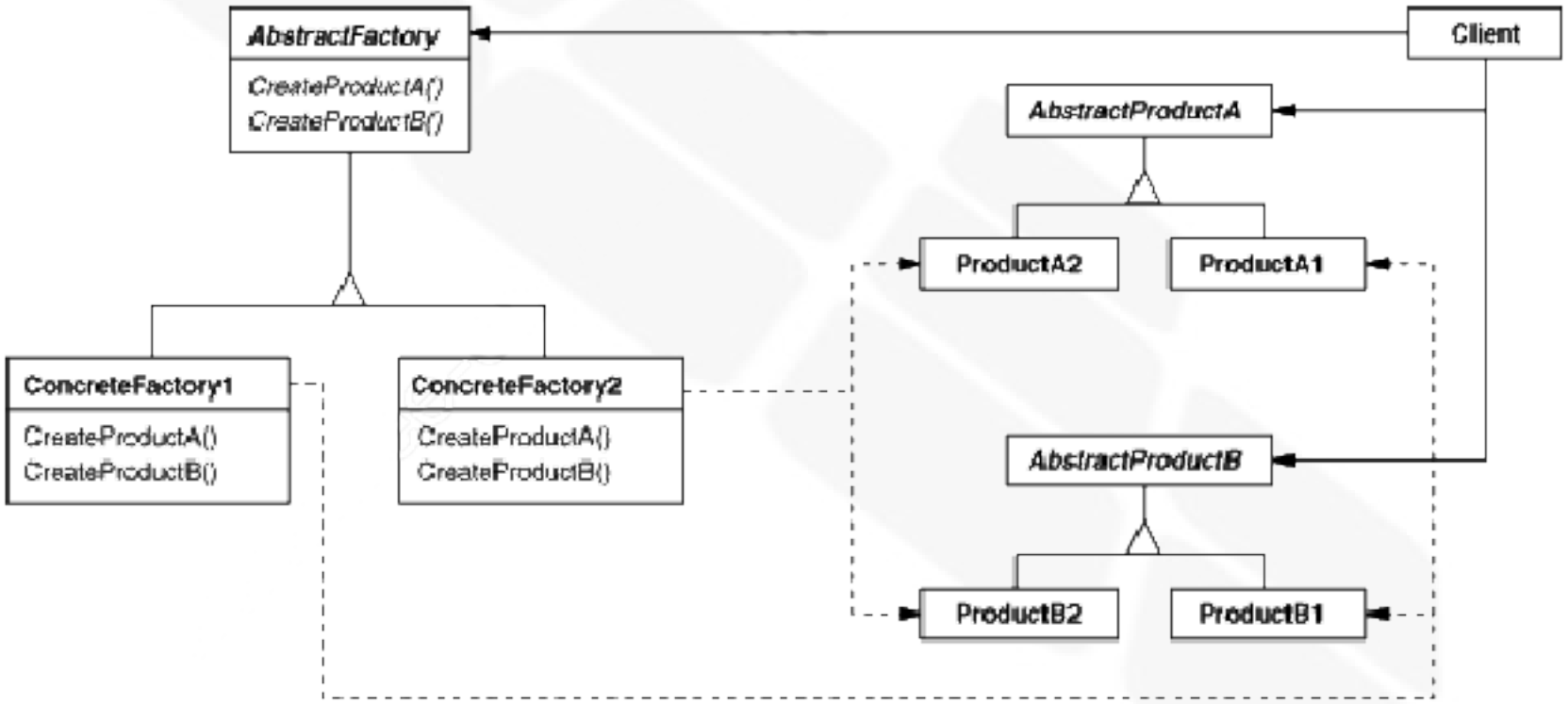
  - Uma sub-classe concreta para cada *look-and-feel*, implementando as operações de criação dos *widgets* apropriados

Abstract Factory

- Aplicabilidade:
  - O sistema precisa ser independente de como os produtos são criados, compostos e representados
  - O sistema deve ser configurado com uma de múltiplas famílias de produtos
  - Produtos de uma família devem ser sempre utilizados em conjunto e isto precisa ser garantido
  - Deseja-se disponibilizar uma biblioteca de classes de produtos mas revelar somente as suas interfaces, não suas implementações

Abstract Factory

- Estrutura:



Abstract Factory

- Participantes:
  - *AbstractFactory*: declara uma interface para operações que criam objetos produto abstratos
  - *ConcreteFactory*: implementa as operações para criar objetos produto concretos
  - *AbstractProduct*: declara uma interface para um tipo de objeto produto
  - *ConcreteProduct*: define o objeto produto a ser criado pela fábrica concreta correspondente e implementa a interface *AbstractProduct*

Abstract Factory

- Colaborações:
  - Normalmente uma única instância de fábrica concreta é criada em *run-time*. Esta fábrica cria produtos com uma implementação em particular. Para criar outros produtos deve-se utilizar uma fábrica diferente
  - As classes abstratas transferem a responsabilidade de criação dos objetos produto para as suas sub-classes

Abstract Factory

- Consequências:
  - Isola as classes concretas:
    - Ele controla as classes dos objetos que a aplicação cria. Clientes manipulam instâncias somente através de suas interfaces abstratas. Nomes de classes estão isolados nas fábricas concretas, eles não aparecem no código do cliente
  - Torna fácil a troca de famílias de produtos:
    - A classe da fábrica concreta sendo utilizada aparece somente uma vez na aplicação, facilitando modificações. A família de produtos mudaria toda de uma vez

Abstract Factory

- Consequências:
  - Promove consistência entre produtos:
    - Garante que os objetos utilizados são todos de uma mesma família, representada pela fábrica concreta sendo utilizada
  - Dificulta a inserção de novos tipos de produtos:
    - A interface da fábrica abstrata torna fixo o conjunto de produtos que podem ser criados
    - Suportar um novo produto exige a extensão da interface da fábrica abstrata e a modificação de todas as suas sub-classes

Abstract Factory

- Implementação:
  - Fábricas como *Singletons*:
    - Geralmente precisa-se de somente uma instância da fábrica concreta por aplicação
  - Criando os produtos:
    - A fábrica abstrata somente define uma *interface* para criação de produtos, geralmente através de um *Factory Method* para cada produto
    - As fábricas concretas implementam esses *Factory Methods* para instanciar os objetos
    - A implementação é simples, mas requer uma fábrica concreta para cada família de produtos, mesmo que elas sejam ligeiramente diferentes

Abstract Factory

- Implementação:
  - Criando os produtos:
    - Se estão previstas muitas famílias, a fábrica concreta pode ser implementada usando o padrão *Prototype*
    - Teríamos apenas uma fábrica concreta, inicializada com os protótipos dos produtos desejados
    - Uma outra variação é utilizar meta-classes, se disponíveis na linguagem

Abstract Factory

- Implementação:
  - Definindo fábricas extensíveis:
    - Na fábrica abstrata, os tipos de produtos fazem parte das assinaturas dos *Factory Methods*. Adicionar um novo produto requer mudar a interface da fábrica abstrata e de todas as classes dela dependentes
    - Uma solução mais flexível porém menos segura é adicionar um parâmetro ao *Factory Method* indicando o tipo de produto a ser criado
    - A fábrica abstrata (e concreta) teria somente um *Factory Method*
    - Entretanto, visto que os produtos deverão ter a mesma classe-base, o cliente os enxergará de uma única maneira, sem distinguir os tipos dos produtos

Abstract Factory

- Código exemplo:

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

    virtual Maze* MakeMaze() const
        { return new Maze; }
    virtual Wall* MakeWall() const
        { return new Wall; }
    virtual Room* MakeRoom(int n) const
        { return new Room(n); }
    virtual Door* MakeDoor(Room* r1, Room* r2) const
        { return new Door(r1, r2); }
};
```

Abstract Factory

- Código exemplo:

```
Maze* MazeGame::CreateMaze (MazeFactory& factory) {
    Maze* aMaze = factory.MakeMaze();
    Room* r1 = factory.MakeRoom(1);
    Room* r2 = factory.MakeRoom(2);
    Door* aDoor = factory.MakeDoor(r1, r2);

    aMaze->AddRoom(r1);
    aMaze->AddRoom(r2);
    r1->SetSide(North, factory.MakeWall());
    r1->SetSide(East, aDoor);
    r1->SetSide(South, factory.MakeWall());
    r1->SetSide(West, factory.MakeWall());

    r2->SetSide(North, factory.MakeWall());
    r2->SetSide(East, factory.MakeWall());
    r2->SetSide(South, factory.MakeWall());
    r2->SetSide(West, aDoor);

    return aMaze;
}
```

Abstract Factory

- Código exemplo:

```
class EnchantedMazeFactory : public MazeFactory {
public:
    EnchantedMazeFactory();

    virtual Room* MakeRoom(int n)  const
        { return new EnchantedRoom(n, CastSpell()); }

    virtual Door* MakeDoor(Room* r1, Room* r2)  const
        { return new DoorNeedingSpell(r1, r2); }

protected:
    Spell* CastSpell() const;
};
```

Abstract Factory

- Código exemplo – salas com bombas:

```
Wall* BombedMazeFactory::MakeWall () const {
    return new BombedWall;
}

Room* BombedMazeFactory::MakeRoom(int n) const {
    return new RoomWithABomb(n);
}
```

```
MazeGame game;
BombedMazeFactory factory;

game.CreateMaze(factory);
```

Abstract Factory

- Código exemplo – observações:
  - Note que a fábrica abstrata é uma coleção de *Factory Methods* e que não necessariamente precisa ser uma classe abstrata. Ela pode atuar como fábrica abstrata e concreta ao mesmo tempo
  - Se a fábrica *BombedMazeFactory* for utilizada, o labirinto será formado por objetos *RoomWithABomb* e *BombedWall*:
    - Objetos do tipo *RoomWithABomb* podem precisar acessar métodos específicos de *BombedWall*, portanto será necessário realizar *downcasting*
    - Este *downcast* é seguro desde que todas as paredes sejam construídas pela mesma fábrica, o que é garantido pelo padrão

Abstract Factory

- Usos conhecidos:
  - *InterViews*: *toolkit* para interfaces gráficas de usuário
  - ET++: utiliza *Abstract Factory* para garantir portabilidade entre diferentes sistemas de janelas (X Windows, Sun View, etc)

Abstract Factory

- Padrões relacionados:
  - O *Abstract Factory* é geralmente implementado com *Factory Methods*, mas também pode utilizar o *Prototype*
  - Uma fábrica concreta é frequentemente um *Singleton*

INF011 – Padrões de Projeto

03 – *Abstract Factory*

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