INF011 – Padrões de Projeto

05 – *Factory Method*

Sandro Santos Andrade
sandroandrade@ifba.edu.br

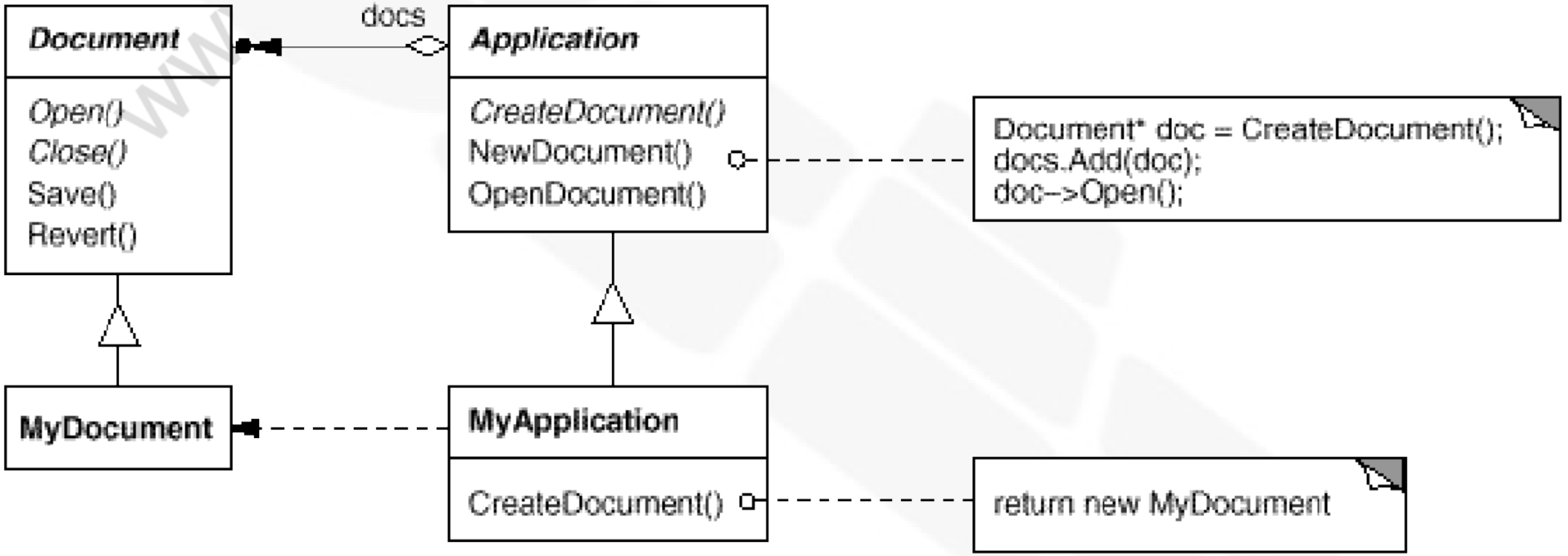
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas

Factory Method

- Propósito:
  - Definir uma interface para criação de um objeto porém deixando que sub-classes decidam de qual classe instanciá-lo
- Também conhecido como: *Virtual Constructor*
- Motivação:
  - Um *Application Framework* para construção de sistemas que apresentam múltiplos documentos ao usuário define as classes abstratas *Application* e *Document*
  - Um sistema de construção de diagramas, por exemplo, teria as classes *DrawingApplication* e *DrawingDocument*
  - O *framework* deve instanciar objetos mas só conhece classes abstratas

Factory Method

- Motivação:



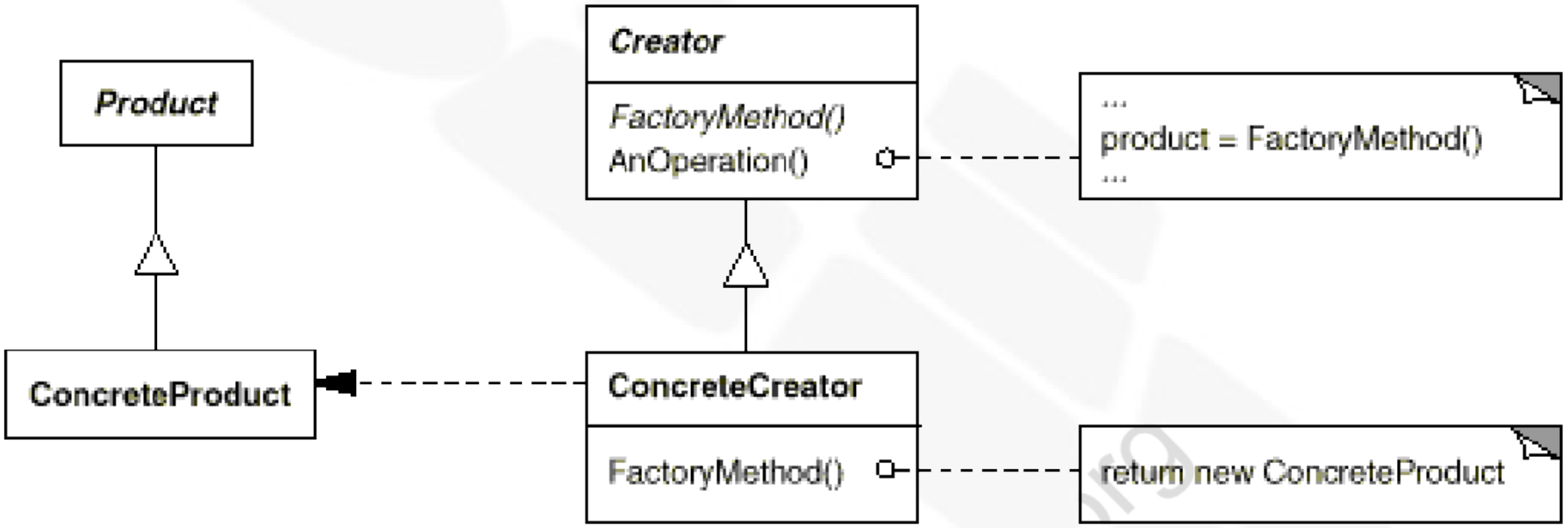
  - Sub-classes de *Application* implementam o método abstrato *CreateDocument* para retornar a sub-classe de *Document* apropriada

Factory Method

- Aplicabilidade:
  - Uma classe não conhece previamente as classes dos objetos que ela precisa instanciar
  - Uma classe quer que suas sub-classes especifiquem os objetos a serem criados
  - Classes delegam responsabilidade para uma entre várias sub-classes *helper* e deseja-se localizar o conhecimento de para qual sub-classe *helper* foi delegada a operação

Factory Method

- Estrutura:



Factory Method

- Participantes:
  - *Product* (Document): define a interface dos objetos que o *factory method* cria
  - *ConcreteProduct* (MyDocument): implementa a interface *Product*
  - *Creator* (Application):
    - Declara o *factory method*, retornando um objeto do tipo *Product.* Pode ser abstrato ou não
    - Geralmente invoca o *factory method*
  - *ConcreteCreator* (MyApplication): implementa (ou sobrepõe) o *factory method* para retornar uma implementação específica de *Product* (algum *ConcreteProduct*)

Factory Method

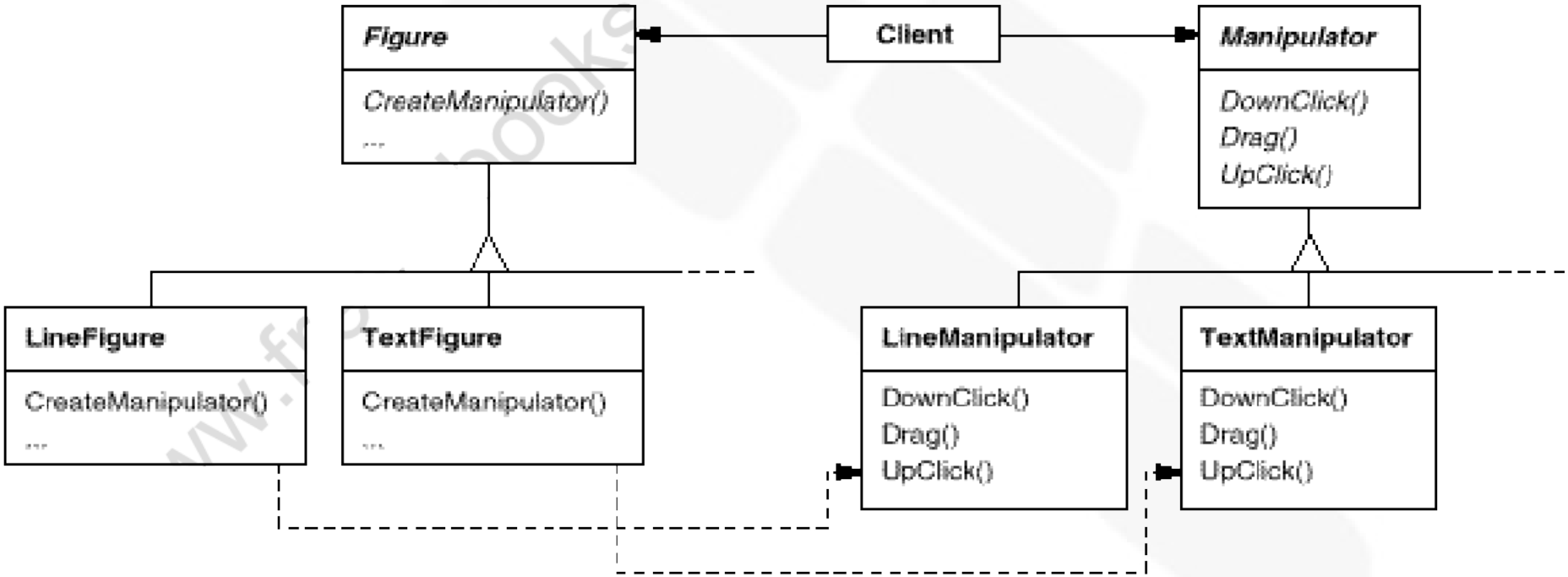
- Colaborações:
  - O *Creator* confia às suas sub-classes a definição do *factory method*, retornando uma instância apropriada de *ConcreteProduct*

Factory Method

- Consequências:
  - Elimina a necessidade de utilizar, no seu código, classes específicas de aplicação pois utiliza somente a interface *Product*
  - Exige a derivação de *Creator* para a implementação de um único método (provavelmente com uma linha)
  - Provê *hooks* às sub-classes: no sistema de documentos, por exemplo, pode-se definir um *factory method* chamado *createFileDialog()* com uma implementação *default* e permitir que sub-classes indiquem a utilização de um objeto alternativo

Factory Method

- Consequências:
  - Conecta hierarquias paralelas de classes:



Factory Method

- Implementação:
  - Duas variações:
    - *Creator* como classe abstrata:
      - Requer a existência de sub-classes que implementam o *factory method* e, portanto, se isolam totalmente de classes não anteriormente previstas
    - *Creator* com implementação *default* do *factory method*:
      - Não requer um *ConcreteCreator*. Define o *factory method* simplesmente para tornar o código mais flexível: sub-classes podem futuramente sobrescrever o *factory method*
  - *Factory methods* parametrizados:
    - O *factory method* recebe um parâmetro indicando o tipo de produto concreto a ser criado

Factory Method

- Implementação:
  - *Factory methods* parametrizados:

```
class Creator {
public:
    virtual Product* Create(ProductId);
};

Product* Creator::Create (ProductId id) {
    if (id == MINE)  return new MyProduct;
    if (id == YOURS) return new YourProduct;
    // repeat for remaining products...

    return 0;
}
```

```
Product* MyCreator::Create (ProductId id) {
    if (id == YOURS)  return new MyProduct;
    if (id == MINE)   return new YourProduct;
        // N.B.: switched YOURS and MINE

    if (id == THEIRS) return new TheirProduct;

    return Creator::Create(id); // called if all others fail
}
```

Factory Method

- Implementação:
  - Variações e problemas específicos de linguagem:
    - O *factory method* pode retornar a **classe** do objeto a ser criado e, portanto, um *ConcreteCreator* pode definir esta classe em *run-time*
    - A classe do objeto a ser criado pode ser também um atributo estático no *Creator*
    - No C++, o *factory method* do *ConcreteCreator* não pode ser chamado no construtor de *Creator*, pois ainda não está disponível. Em vez disso, faça *lazy initialization*
    - No C++, *templates* podem ser utilizados para evitar a criação de sub-classes com somente um método
    - Padronização para nomes dos *factory methods*:
      - *Class *doMakeClass()*, onde *Class* é a classe do produto

Factory Method

- Código exemplo:

```
class MazeGame {
public:
    Maze* CreateMaze();

// factory methods:

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

```
Maze* MazeGame::CreateMaze () {
    Maze* aMaze = MakeMaze();

    Room* r1 = MakeRoom(1);
    Room* r2 = MakeRoom(2);
    Door* theDoor = MakeDoor(r1, r2);

    aMaze->AddRoom(r1);
    aMaze->AddRoom(r2);

    r1->SetSide(North, MakeWall());
    r1->SetSide(East, theDoor);
    r1->SetSide(South, MakeWall());
    r1->SetSide(West, MakeWall());

    r2->SetSide(North, MakeWall());
    r2->SetSide(East, MakeWall());
    r2->SetSide(South, MakeWall());
    r2->SetSide(West, theDoor);

    return aMaze;
}
```

Factory Method

- Código exemplo:

```
class BombedMazeGame : public MazeGame {
public:
    BombedMazeGame();

    virtual Wall* MakeWall() const
        { return new BombedWall; }

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

```
class EnchantedMazeGame : public MazeGame {
public:
    EnchantedMazeGame();

    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;
};
```

Factory Method

- Usos conhecidos:
  - Altamente presente em *toolkits* e *frameworks*
  - MacApp (exemplo do documento) e ET++ (exemplo dos manipuladores)
  - *Framework* MVC do Smalltalk
  - Orbix ORB: *factory method* para gerar o tipo apropriado de *proxy* quando uma invocação remota é necessária

Factory Method

- Padrões relacionados:
  - *Abstract Factory* é frequentemente implementado com *Factory Methods*
  - *Factory methods* são geralmente chamados dentro de *Template methods*
  - Utilizando *Prototypes* a criação das sub-classes não é necessária, entretanto será necessário um método de inicialização do produto, a ser chamado pelo *Creator*. Com *factory methods* convencionais isso não é necessário

INF011 – Padrões de Projeto

05 – *Factory Method*

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