

# Նախագծման Ձևանմուշներ: FactoryMethod

Հրաչյա Թանդիլյան

2020

# FactoryMethod

## Նպատակը

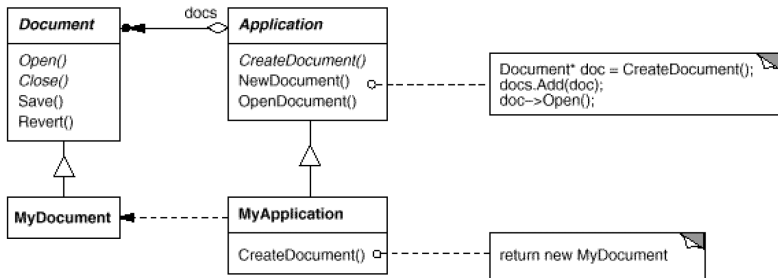
Ինտերֆեյս է սահմանում օբյեկտ ստեղծելու համար, սակայն ենթադասերին թույլ է տալիս որոշել ստեղծվելիք օբյեկտի տիպը:

Այսինքն այն թույլ է տալիս զիջել օբյեկտի ստեղծումը ենթադասերին:

Նաև հայտնի է որպես

- Virtual Constructor

# Մոտիվացիան



# Կիրառելիությունը

Այս Ն.Ձ. պետք է օգտագործել երբ.

- Դասը չի կարող կանխատեսել ստեղծվելիք օբյեկտի տիպը:

# Կիրառելիությունը

Այս Ն.Ձ. պետք է օգտագործել երբ.

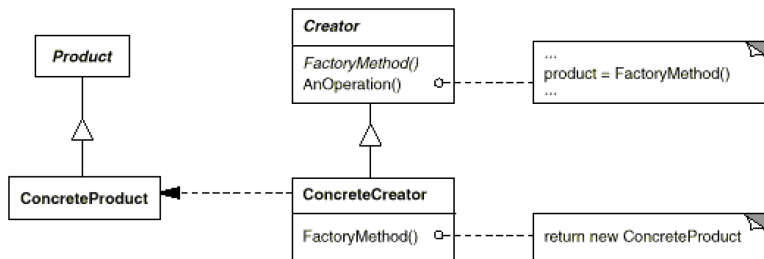
- Ա** Դասը չի կարող կանխատեսել ստեղծվելիք օբյեկտի տիպը:
- Բ** Անհրաժեշտ է, որ դասի ենթադասերը որոշեն ստեղծվելիք օբյեկտի տիպը:

# Կիրառելիությունը

Այս Ն.Ձ. պետք է օգտագործել երբ.

- Ա** Դասը չի կարող կանխատեսել ստեղծվելիք օբյեկտի տիպը:
- Բ** Անհրաժեշտ է, որ դասի ենթադասերը որոշեն ստեղծվելիք օբյեկտի տիպը:
- Գ** Դասը փոխանցում է պատասխանատվությունը մի քանի օգնական դասերի և անհրաժեշտ է լուրջաբանել, թե որ օգնական դասին է փոխանցվել պատասխանատվությունը:

# Կառուցվածքը



# Հետևանքները

Այս Ն.Ձ. ունի հետևյալ առավելություններն ու թերությունները.

- Վերացնում է կիրառությանը հատուկ դասերին հղվելու անհրաժեշտությունը կոդում:



# Հետևանքները

Այս Ն.Ձ. ունի հետևյալ առավելություններն ու թերությունները.

- Ա** Վերացնում է կիրառությանը հատուկ դասերին հղվելու անհրաժեշտությունը կոդում:
- Բ** Առաջացնում է հավելյալ ժառանգության վտանգ:

## Հետևանքները

Այս Ն.Ձ. ունի հետևյալ առավելություններն ու թերությունները.

- Ա** Վերացնում է կիրառությանը հատուկ դասերին հղվելու անհրաժեշտությունը կոդում:
- Բ** Առաջացնում է հավելյալ ժառանգության վտանգ:
- Գ** Ապահովում է ճկունություն ենթադասերի համար:

# Հետևանքները

Այս Ն.Ձ. ունի հետևյալ առավելություններն ու թերությունները.

- Ա** Վերացնում է կիրառությանը հատուկ դասերին հղվելու անհրաժեշտությունը կողում:
- Բ** Առաջացնում է հավելյալ ժառանգության վտանգ:
- Գ** Ապահովում է ճկունություն ենթադասերի համար:
- Դ** Կապ է ստեղծում դասերի գուգահեռ հիերարխիաների միջև:

# Իրականացումը

- Ա Երկու հիմնական տարբերակներ:
- Բ Պարամետրիզացված Factory Method:

# Պարամետրիզացված FactoryMethod

```
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;  
}
```

# Պարամետրիզացված FactoryMethod

```
Product* MyCreator::Create(ProductId id) {  
  
    if (id == YOURS) return new MyProduct;  
  
    if (id == MINE) return new YourProduct;  
  
    // switched YOURS and MINE  
  
    if (id == THEIRS) return new TheirProduct;  
  
    return Creator::Create(id); // called if all others fail  
}
```

# Իրականացումը

- Ա Երկու հիմնական տարբերակներ:
- Բ Պարամետրիզացված Factory Method:
- Գ Լեզվին հատուկ պրոբլեմներ:

## Լեզվին հատուկ պրոբլեմներ

```
class Creator {  
    public:  
        Product* GetProduct() {  
            if (_product == 0) {  
                _product = CreateProduct();  
            }  
            return product_;  
        }  
  
    protected:  
        virtual Product* CreateProduct();  
  
    private:  
        Product* _product;  
};
```



# Իրականացումը

- Ա Երկու հիմնական տարբերակներ:
- Բ Պարամետրիզացված Factory Method:
- Գ Լեզվին հատուկ պրոբլեմներ:
- Դ Օգտագործել template-ների ժառանգելուց խուսափելու համար:

# Template-ների ժառանգելուց խուսափելու համար

```
class Creator {
    public:
        virtual Product* CreateProduct() = 0;
};

template <class TheProduct>
class StandardCreator: public Creator {
    public:
        virtual Product* CreateProduct() () { return new TheProduct; }
};

class MyProduct : public Product {
    public:
        MyProduct();
};

StandardCreator<MyProduct> myCreator;
```

# Իրականացումը

- Ա Երկու հիմնական տարբերակներ:
- Բ Պարամետրիզացված Factory Method:
- Գ Լեզվին հատուկ պրոբլեմներ:
- Դ Օգտագործել template-ներ ժառանգելուց խուսափելու համար:
- Ե Անվանակոչում:

# Օրինակ

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

# Օրինակ

```
Maze* MazeGame::CreateMaze() {  
    Maze* aMaze = MakeMaze();  
  
    Room* r1 = MakeRoom(1); Room* r2 = MakeRoom(2);  
    aMaze->AddRoom(r1); aMaze->AddRoom(r2);  
  
    Door* aDoor = MakeDoor(r1, r2);  
  
    r1->SetSide(North, MakeWall()); r1->SetSide(East, aDoor);  
    r1->SetSide(South, MakeWall()); r1->SetSide(West, MakeWall());  
  
    r2->SetSide(North, MakeWall()); r2->SetSide(East, MakeWall());  
    r2->SetSide(South, MakeWall()); r2->SetSide(West, aDoor);  
  
    return aMaze;  
}
```

# Օրինակ

```
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;  
};
```

# Օրինակ

```
class BombedMazeGame : public MazeGame {  
    public:  
        BombedMazeGame();  
  
        virtual Room* MakeRoom(int n) const {  
            return new RoomWithABomb(n);  
        }  
  
        Wall* MakeWall () const {  
            return new BombedWall;  
        }  
};
```

```
BombedMazeGame game;  
game.CreateMaze();
```

# Առնչվող Նախագծման Ձևանմուշները

- Abstract Factory

- Template Methods