Factory Method

Linda Marshall

Department of Computer Science
University of Pretoria

26 August 2021

## Name and Classification:

Factory Method

(Class Creational)

Intent:

"Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses." GoF(107)

"Define an interface for creating an object, but let subclasses decide which class to

instantiate. Factory Method lets a class defer instantiation to subclasses."
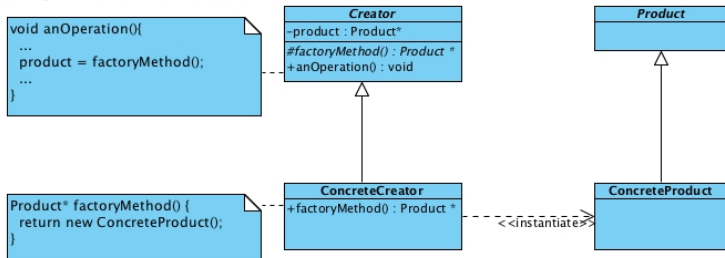
- A *Creator* creates product which the client uses.
- *Product* is always created by *Creator*.
- *ConcreteCreators* create specific concrete product.
- Makes use of the **Template Method** design pattern.

- Forces the creation of an object to occur in a common factory rather than scattered around the code
- A factory can be implemented by using a static factory member, or by making use of polymorphism

Product

- defines the product interface for the factory method to create

ConcreteProduct

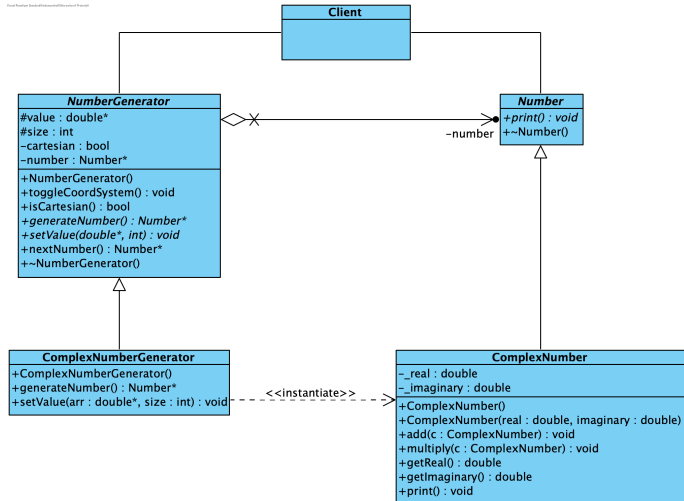- implements the interface for the product

Creator

- declares the factory method which returns a product object

- default factory method implementations may return a default concrete product

ConcreteCreator

- overrides the factory method to return an instance of the product

Related Patterns

- **Abstract Factory** (87): May be used to implement.

- **Template Method** (325): May be called by.

- **Prototype** (117): Used to initialise objects.

Identification
Structure
Participants
Related Patterns
Examples

Number generator
Chocolate factory

Identification
Structure
Participants
Related Patterns
Examples

Number generator
Chocolate factory

```cpp
class NumberGenerator {
  public:
    NumberGenerator();
    void toggleCoordSystem();
    bool isCartesian();
    virtual Number* generateNumber() = 0;
    virtual void setValue(double*, int) = 0;
    Number* nextNumber();
    virtual ~NumberGenerator();
  protected:
    double* value;
    int size;
  private:
    bool cartesian;
    Number* number;
};
```

Identification
Structure
Participants
Related Patterns
Examples

Number generator
Chocolate factory

```
NumberGenerator :: NumberGenerator()  {
      number = 0;   cartesian = true;
      value = 0;  size = 0;
}
void  NumberGenerator :: toggleCoordSystem() {
      cartesian = !cartesian;
}
bool  NumberGenerator :: isCartesian() {
      return  cartesian;
}
Number* NumberGenerator :: nextNumber() {
      number = generateNumber();
      return  number;
}
NumberGenerator :: ˜NumberGenerator()  {
      if (number != 0) {  number = 0;  }
      if (size != 0)   { delete [] value;  value = 0;  }
}
```

Identification
Structure
Participants
Related Patterns
Examples

Number generator
Chocolate factory

```cpp
class ComplexNumberGenerator : public NumberGenerator {
  public:
    ComplexNumberGenerator() : NumberGenerator() { };
    virtual Number* generateNumber() {
      if (size == 0) {
        value = new double [2];
        value[0] = 0;   value[1] = 0;   size = 2;         }
      if (isCartesian())
        return new ComplexNumber(value[0], value[1]);
      else
        return new ComplexNumber(value[0]*cos(value[1]),
                                  value[0]*sin(value[1]));
    };
    virtual void setValue(double* arr, int size) {
      if (this->size != 0) {
        delete [] value;
        this->size = 0;   }
      value = new double[size];
      value[0] = arr[0];   value[1] = arr[1];
      this->size = size;
    };
};
```

Identification
Structure
Participants
Related Patterns
Examples

Number generator
Chocolate factory

```cpp
class Number {
  public:
    virtual void print() = 0;
    virtual ~Number();
};

class ComplexNumber : public Number {
public:
        ComplexNumber();
        ComplexNumber(double real, double imaginary);
        void add(ComplexNumber c);
        void multiply(ComplexNumber c);
        double getReal();
        double getImaginary();
        void print();
private:
        double _real;
        double _imaginary;
};
```

Identification
Structure
Participants
Related Patterns
Examples

Number generator
Chocolate factory

```cpp
ComplexNumber::ComplexNumber() : Number() {
        _real = 0;
        _imaginary = 0;
}
ComplexNumber::ComplexNumber(double real, double imaginary) {
        _real = real;
        _imaginary = imaginary;
}
void ComplexNumber::add(ComplexNumber c) {
        _real = _real + c._real;
        _imaginary = _imaginary + c._imaginary;
}
void ComplexNumber::multiply(ComplexNumber c) {
        _real = (_real * c._real) - (_imaginary * c._imaginary);
        _imaginary = (_real * c._imaginary) + (_imaginary * c._rea
}
double ComplexNumber::getReal() { return _real; }
double ComplexNumber::getImaginary() { return _imaginary; }
void ComplexNumber::print() {
        std::cout << _real << " + " << _imaginary << " i" << std::endl;
}
```

Identification
Structure
Participants
Related Patterns
Examples

Number generator
Chocolate factory

Exercise

- Add a `Rational` Number class that inherits from `Number`.

    - Is it possible to abstract some operations out of the concrete products?
    - Challenge: Write the mathematical operations as C++ operators.

- Add a concrete factory class to create Rational numbers.

Identification
Structure
Participants
Related Patterns
Examples

Number generator
Chocolate factory