Factory Method in C++: Before and after

Back to **Factory Method** description

Before

The architect has done an admirable job of decoupling the client from Stooge concrete derived classes, and, exercising polymorphism. But there remains coupling where instances are actually created. If we design an "extra level of indirection" (a "factory method") and have clients use it (instead of "new"), then the last bit of coupling goes away. The "factory method" (aka "virtual constructor") can be defined in the Stooge base class, or, in a separate "factory" class. Note that `main()` is no longer dependent on Stooge derived classes.

```
class Stooge
{
  public:
    virtual void slap_stick() = 0;
};

class Larry: public Stooge
{
  public:
    void slap_stick()
    {
        cout << "Larry: poke eyes\n";
    }
};
class Moe: public Stooge
{
  public:
    void slap_stick()
    {
        cout << "Moe: slap head\n";
    }
};
class Curly: public Stooge
{
  public:
    void slap_stick()
    {
        cout << "Curly: suffer abuse\n";
    }
};

int main()
{
  vector<Stooge*> roles;
  int choice;

  while (true)
  {
    cout << "Larry(1) Moe(2) Curly(3) Go(0): ";
    cin >> choice;
    if (choice == 0)
      break;
    else if (choice == 1)
      roles.push_back(new Larry);
    else if (choice == 2)
      roles.push_back(new Moe);
    else
      roles.push_back(new Curly);
```

```
  }
  for (int i = 0; i < roles.size(); i++)
    roles[i]->slap_stick();
  for (int i = 0; i < roles.size(); i++)
    delete roles[i];
}
```

Output

```
Larry(1) Moe(2) Curly(3) Go(0): 2
Larry(1) Moe(2) Curly(3) Go(0): 1
Larry(1) Moe(2) Curly(3) Go(0): 3
Larry(1) Moe(2) Curly(3) Go(0): 0
Moe: slap head
Larry: poke eyes
Curly: suffer abuse
```

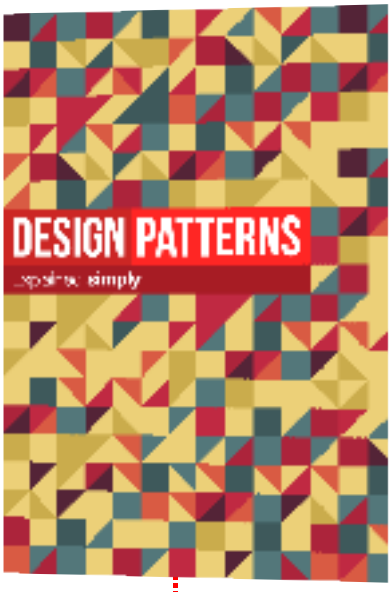
After

A factory method is a static method of a class that returns an object of that class' type. But unlike a constructor, the actual object it returns might be an instance of a subclass. Another advantage of a factory method is that it can return existing instances multiple times.

```
class Stooge
{
  public:
    // Factory Method
    static Stooge *make_stooge(int choice);
    virtual void slap_stick() = 0;
};

int main()
{
  vector<Stooge*> roles;
  int choice;
  while (true)
  {
    cout << "Larry(1) Moe(2) Curly(3) Go(0): ";
    cin >> choice;
    if (choice == 0)
      break;
    roles.push_back(Stooge::make_stooge(choice));
  }
  for (int i = 0; i < roles.size(); i++)
    roles[i]->slap_stick();
  for (int i = 0; i < roles.size(); i++)
    delete roles[i];
}

class Larry: public Stooge
{
  public:
    void slap_stick()
    {
        cout << "Larry: poke eyes\n";
    }
};
class Moe: public Stooge
{
  public:
    void slap_stick()
    {
        cout << "Moe: slap head\n";
    }
};
class Curly: public Stooge
{
  public:
    void slap_stick()
    {
        cout << "Curly: suffer abuse\n";
```

```
    }
};

Stooge *Stooge::make_stooge(int choice)
{
  if (choice == 1)
    return new Larry;
  else if (choice == 2)
    return new Moe;
  else
    return new Curly;
}
```

Read next



This article is taken from our book **Design Patterns Explained Simply**.

All of the design patterns are compiled there. The book is written in clear, simple language that makes it easy to read and understand (just like this article).

We distribute it in PDF & EPUB formats so you can get it onto your iPad, Kindle, or other portable device immediately after your purchase.

♥ Learn more

Code examples

Java	Factory Method in Java	
C++	Factory Method in C++: Before and after	Factory Method in C++
PHP	Factory Method in PHP	

Delphi	Factory Method in Delphi

Design Patterns
AntiPatterns
Refactoring
UML

My account
Forum
Contact us
About us