Abstract Factory in C++

Back to **Abstract Factory** description

Discussion. "Think of constructors as factories that churn out objects". Here we are allocating the constructor responsibility to a factory object, and then using inheritance and virtual member functions to provide a "virtual constructor" capability. So there are two dimensions of decoupling occurring. The client uses the factory object instead of "new" to request instances; and, the client "hard-wires" the family, or class, of that factory only once, and throughout the remainder of the application only relies on the abstract base class.

```
#include <iostream.h>

class Shape {
  public:
    Shape() {
      id_ = total_++;
    }
    virtual void draw() = 0;
  protected:
    int id_;
    static int total_;
};
int Shape::total_ = 0;

class Circle : public Shape {
  public:
    void draw() {
      cout << "circle " << id_ << ": draw" << endl;
    }
};
class Square : public Shape {
  public:
    void draw() {
      cout << "square " << id_ << ": draw" << endl;
    }
};
class Ellipse : public Shape {
  public:
    void draw() {
      cout << "ellipse " << id_ << ": draw" << endl;
    }
};
class Rectangle : public Shape {
  public:
    void draw() {
      cout << "rectangle " << id_ << ": draw" << endl;
    }
};

class Factory {
  public:
    virtual Shape* createCurvedInstance() = 0;
    virtual Shape* createStraightInstance() = 0;
};

class SimpleShapeFactory : public Factory {
  public:
    Shape* createCurvedInstance() {
      return new Circle;
    }
    Shape* createStraightInstance() {
      return new Square;
    }
};
class RobustShapeFactory : public Factory {
  public:
    Shape* createCurvedInstance()   {
      return new Ellipse;
    }
    Shape* createStraightInstance() {
      return new Rectangle;
    }
};

int main() {
#ifdef SIMPLE
  Factory* factory = new SimpleShapeFactory;
#elif ROBUST
  Factory* factory = new RobustShapeFactory;
#endif
  Shape* shapes[3];

  shapes[0] = factory->createCurvedInstance();   // shapes[0] = new Ellipse;
  shapes[1] = factory->createStraightInstance(); // shapes[1] = new Rectangle;
  shapes[2] = factory->createCurvedInstance();   // shapes[2] = new Ellipse;

  for (int i=0; i < 3; i++) {
    shapes[i]->draw();
  }
}
```

Output

```
ellipse 0: draw
rectangle 1: draw
ellipse 2: draw
```

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	Abstract Factory in Java	Abstract Factory in Java
C++	Abstract Factory in C++	Abstract Factory in C++: Before and after
PHP	Abstract Factory in PHP	Abstract Factory in PHP
Delphi	Abstract Factory in Delphi	

Design Patterns
AntiPatterns
Refactoring
UML

My account
Forum
Contact us
About us

© 2007-2018 SourceMaking.com
All rights reserved.

Terms / Privacy policy