Department of Computer Science
COS121 Lecture Notes

Chapter 7- Absract Factory design pattern

Copyright ©2015 by Linda Marshall and Vreda Pieterse. All rights reserved.

Contents

7.1 Introduction **2**

7.2 Abstract Factory Pattern **2**
7.2.1 Identification 2
7.2.2 Structure 2
7.2.3 Participants 2

7.3 Abstract FactoryPattern Explained **3**
7.3.1 Clarification 3
7.3.2 Common Misconceptions 3
7.3.3 Related Patterns 3

7.4 Example **4**

7.5 Exercises **6**

References **6**

7.1 Introduction

The Abstract Factory design pattern is a creational pattern used to produce product with a common theme [2]. The factories are grouped together under a single interface and linked to differentiated products. Each product hierarchy defines an interface.

7.2 Abstract Factory Pattern

7.2.1 Identification

Name	Classification	Strategy
Abstract Factory	Creational	Delegation (Object)
Intent		
Provide an interface for creating families of related or dependent objects without specifying the concrete classes. (Gamma et al. [1]:87)		

7.2.2 Structure

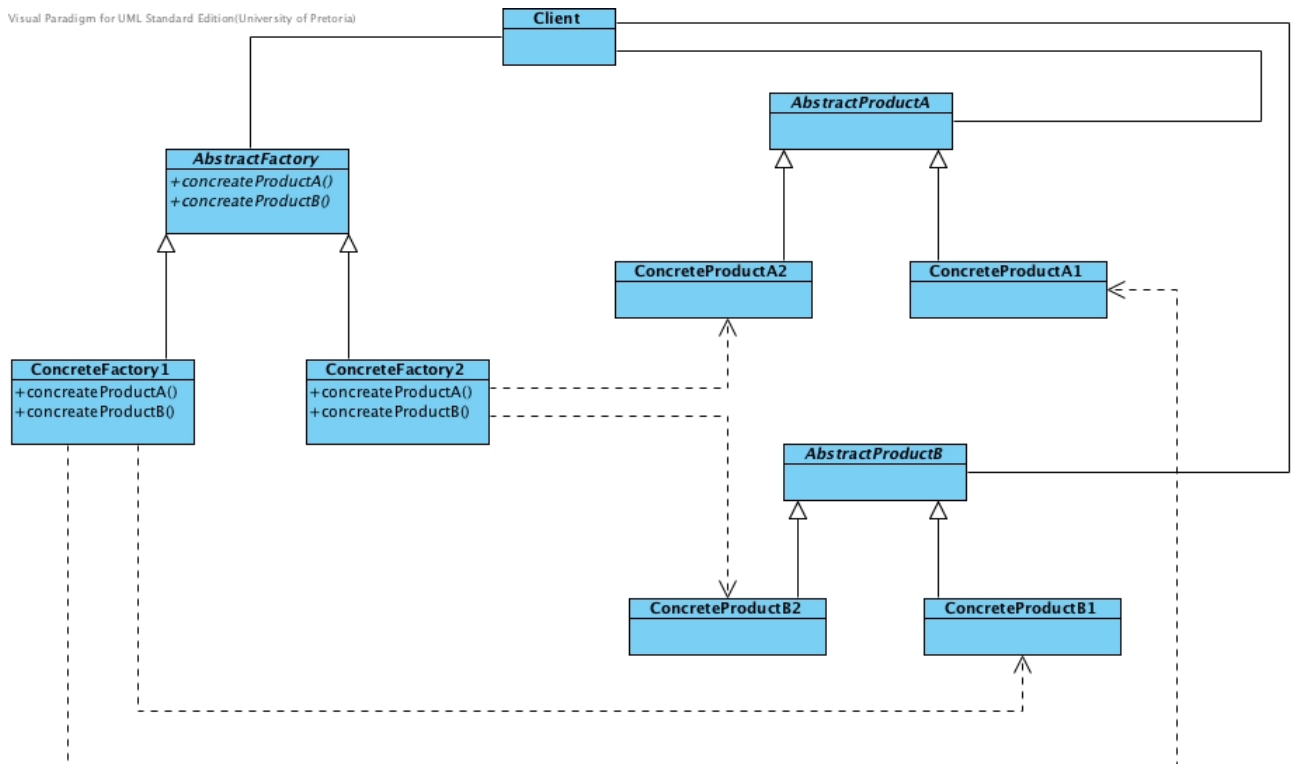

Figure 1: The structure of the Abstract Factory Pattern

7.2.3 Participants

AbstractFactory

- provides an interface to produce abstract product objects

ConcreteFactory

- implements the abstract operations to produce concrete product objects

AbstractProduct

- provides an interface for product objects

ConcreteProduct

- implements the abstract operations that produce product objects that are created by the corresponding ConcreteFactory

Client

- uses the interfaces defined by AbstractFactory and AbstractProduct

7.3 Abstract FactoryPattern Explained

7.3.1 Clarification

The abstract factory comprises of concrete factories. It is the concrete factories that creates product.

7.3.2 Common Misconceptions

Important to note the subtle differences between Factory Method and Abstract Factory. With Factory Method there is a one-to-one relationship between the factory and the product, Abstract Factories exhibit a one-to-many relationship.

7.3.3 Related Patterns

Factory Method or Protoype
The Abstract Factory makes use of the Factory Method or the Prototype for the creation of product. The choice of which route to follow is implementation dependent.

Template Method
May be used within the abstract factory and product hierarchies.

Singleton
Concrete factories may be implemented as Singletons..

7.4 Example

Consider a classification for two-dimensional shapes. 2D shapes are further classified as either polygons or not being a polygon. Figure 2 presents the hierarchy for polygons represented by the system.

Polygons are classified as either quadrilaterals or triangles. All these classes are abstract. The concrete classes are those that inherit from `Quadrilateral` and `Triangle`. The `Polygon` hierarchy forms that product hierarchy for polygons.

The product hierarchy for non-polygons is shown in figure 3. `NonPolygon` is an abstract class, while the `Ellipse` and `Circle` classes are both concrete classes.

As both these hierarchies represent two-dimensional shapes, another abstract class, the `Shape` class is introduced in order to ensure consistency in both hierarchies (refer to figure 4). This class does not form part of the Abstract Factory design pattern, but does not detract from it either. It merely defines the common aspects of all two-dimensional shapes.

The class `Shape` holds two state attributes that are of interest in this example. The first is `LoS`, or lines of symmetry, for each concrete shape the lines of symmetry is stored. The second interesting attribute is `RS` which represents the order of rotational symmetry of the shape. Some shapes have a RS of order 0, while other shapes such as a circle have a RS of infinity.

Till now, the example only comprises of a hierarchy of two-dimensional shapes. Notice that this hierarchy classifies the shapes in terms of their structural characteristics. The attributes of LoS and RS are just that, attributes, and do not form part of the classification. In order to *mesh* the symmetry classification with the structural classification, an abstract factory can be used to produce the shapes according to the symmetry characteristics. The class that represents the *Abstract Factory* participant of the design pattern is in figure 5.

The two concrete classes produce objects that are either line symmetric or rotational symmetric. The classification of polygon and non-polygon is also preserved as each of the classes produce their respective polygon types as well. The code showing how the `ConcreteFactory` classes are implemented is given to show how the classes are linked.

```
class LineSymmetricShapeFactory : public ShapeFactory {
  public:
    LineSymmetricShapeFactory() : ShapeFactory() {};
    LineSymmetricShapeFactory(int lines) {magnitude = lines;};
    Shape* createPolygonInstance() {
      switch (magnitude) {
        case 0 : return new RightAngledTriangle;
                      // or return new Parallelogram;
        case 1 : return new IsoscelesTriangle;
        case 2 : return new Rectangle;
                      // or return new Oblong;
        case 3 : return new EquilateralTriangle;
        case 4 : return new Square;
        default: return 0;
      }
    };
```

```
    Shape* createNonPolygonInstance() {
      if (magnitude == 2)
        return new Ellipse;
      return new Circle;
    }
};

class RotationalSymmetricShapeFactory : public ShapeFactory {
  public:
    RotationalSymmetricShapeFactory() : ShapeFactory() {};
    RotationalSymmetricShapeFactory(int order) {magnitude = order;};
    Shape* createPolygonInstance()   {
      switch (magnitude) {
        case 0 : return new IsoscelesTriangle;
                        // or return new RightAngledTriangle;
        case 2 : return new Parallelogram;
                        // or return new Rectangle;
                        // or return new Oblong;
        case 3 : return new EquilateralTriangle;
        case 4 : return new Square;
        default: return 0;
      }
    };

    Shape* createNonPolygonInstance() {
      if (magnitude == 2)
        return new Ellipse;
      return new Circle;
    }
};
```

To illustrate how the abstract factory is used to produce product, consider the following main program.

```
int main() {
  ShapeFactory** factory  = new ShapeFactory*[2];
  factory[0] = new LineSymmetricShapeFactory;
  factory[1] = new RotationalSymmetricShapeFactory(2);

  Shape* shapes[4];

  shapes[0] = factory[0]->createPolygonInstance();
  shapes[1] = factory[0]->createNonPolygonInstance();
  shapes[2] = factory[1]->createPolygonInstance();
  shapes[3] = factory[1]->createNonPolygonInstance();

  for (int i=0; i < 4; i++) {
    if (shapes[i] != 0)
      shapes[i]->setState();
```

```
  }

  for (int i=0; i < 4; i++) {
    if (shapes[i] != 0)
      cout << "Area = " << shapes[i]->area() << endl;
  }

  for (int i=0; i < 4; i++) {
    if (shapes[i] != 0)
      delete shapes[i];
  }

  for (int i=0; i < 2; i++) {
    delete factory[i];
  }
  delete [] factory;


  return 0;

}
```

7.5 Exercises

1. Merge the class diagrams given in figures 2, 3, 4 and 5. Make sure that all the delegation, specifically the dependencies, relationships between the concrete factories and the concrete products are included.
2. Consider the class diagram presented in figure 6 and identify the participants.

References

[1] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns : elements of reusable object-oriented software*. Addison-Wesley, Reading, Mass, 1995.

[2] Wikipedia. Abstract factory pattern — wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Abstract_factory_pattern`, 2011. [Online; accessed 12-August-2013].

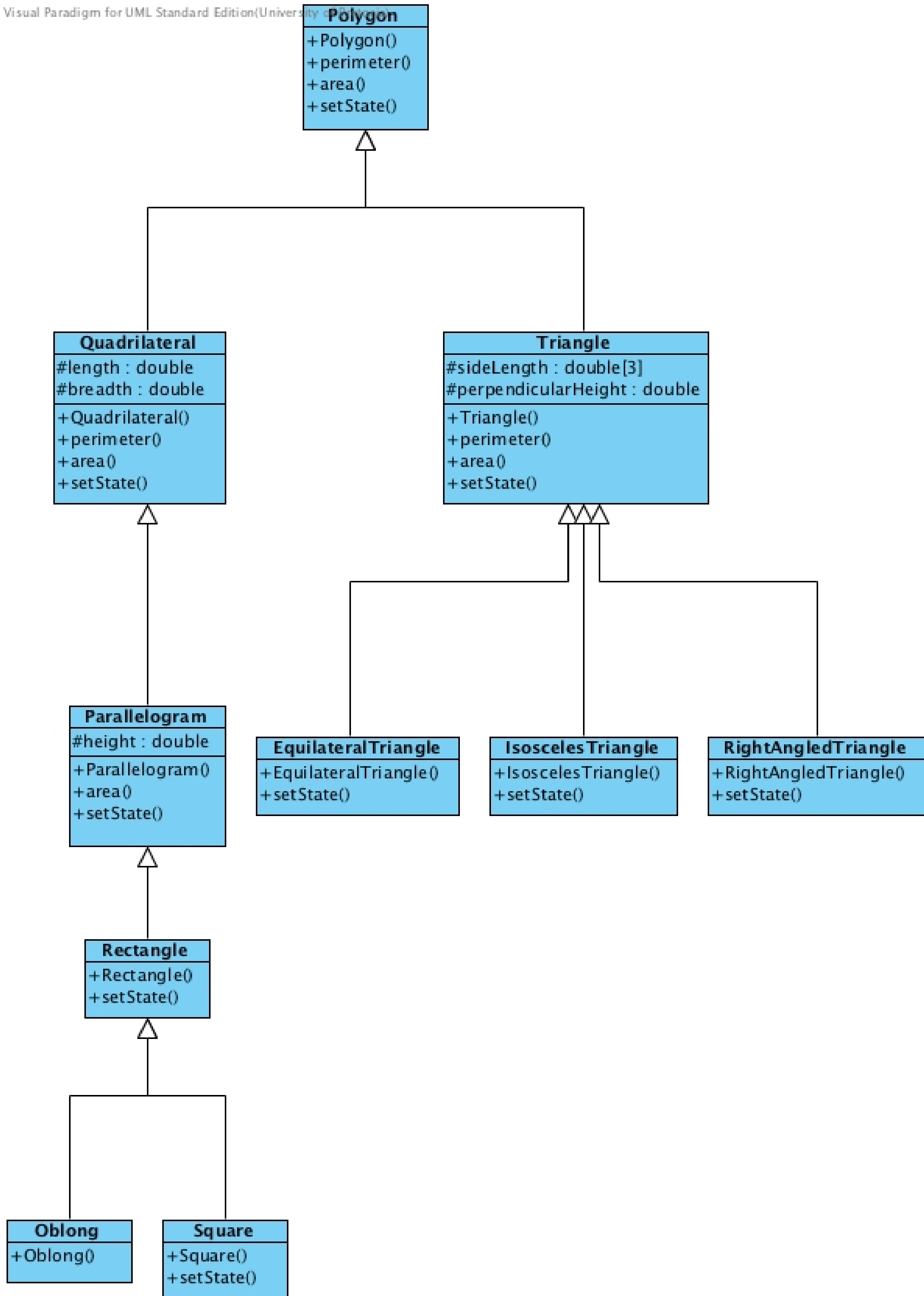

Figure 2: Polygon class hierarchy

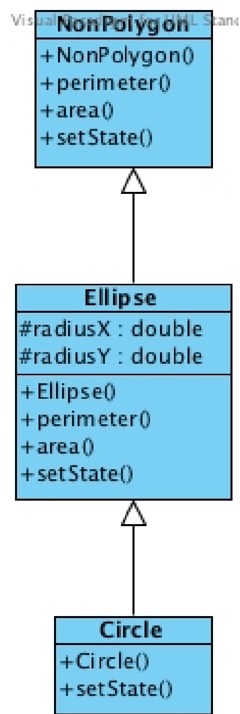

Figure 3: NonPolygon class hierarchy

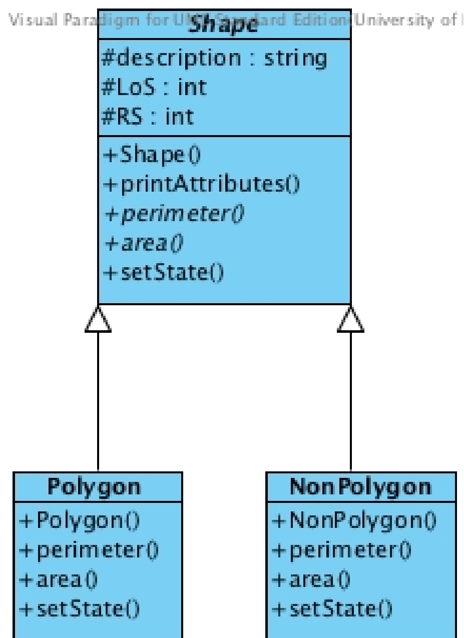

Figure 4: Overarching abstract `Shape` class

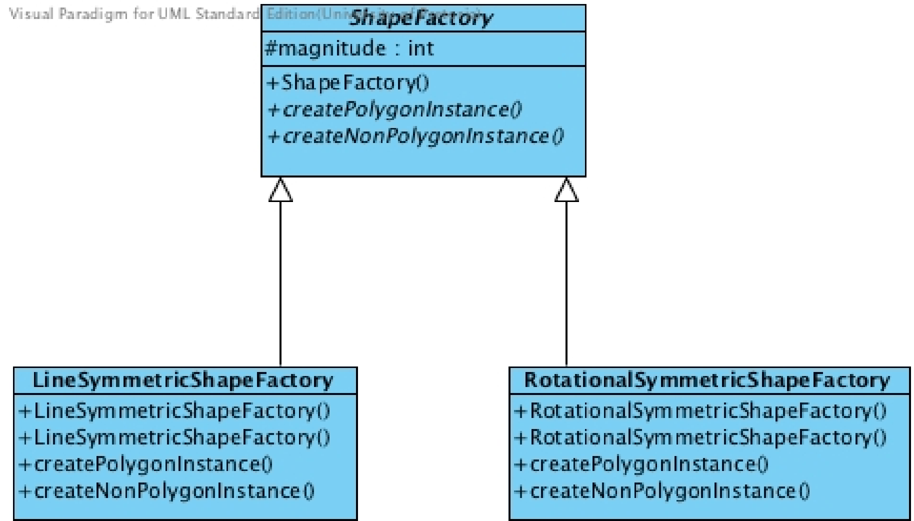

Figure 5: Abstract Factory class hierarchy diagram

Figure 6: Candy class hierarchy diagram