Object-Oriented Software Analysis and Design

School of Computer Science
University of Windsor

Gang of Four's Pattern Catalog

Creational	Structural	Behavioral
Abstract Factory	Adapter	Chain of Responsibility
Builder	Brideg	Command
Factory Method	Composite	Interpreter
Prototype	Decorator	Iterator
Singleton	Facade	Mediator
	Flyweight	Memento
	Proxy	Observer
		State
		Strategy
		Template Method
		Visitor

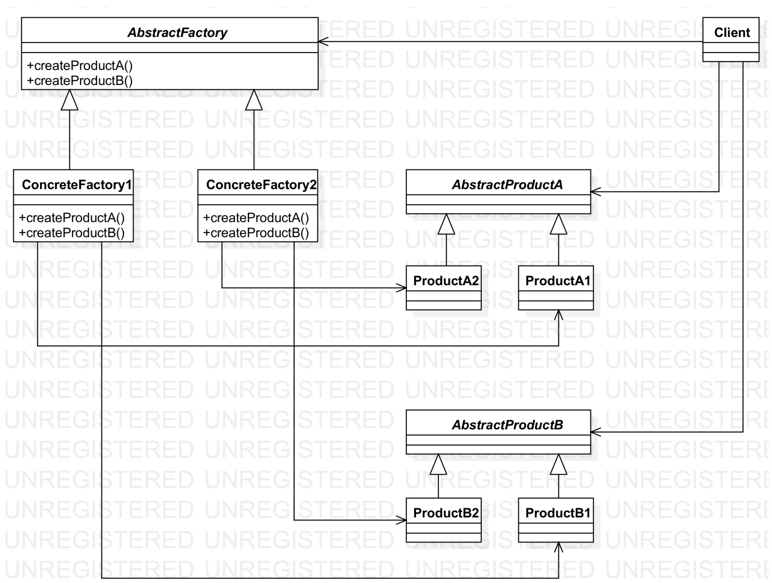
Abstract Factory: Intent

- Provide an interface for creating families of related or dependent objects without specifying their concrete classes.
- Also known as **Kit**

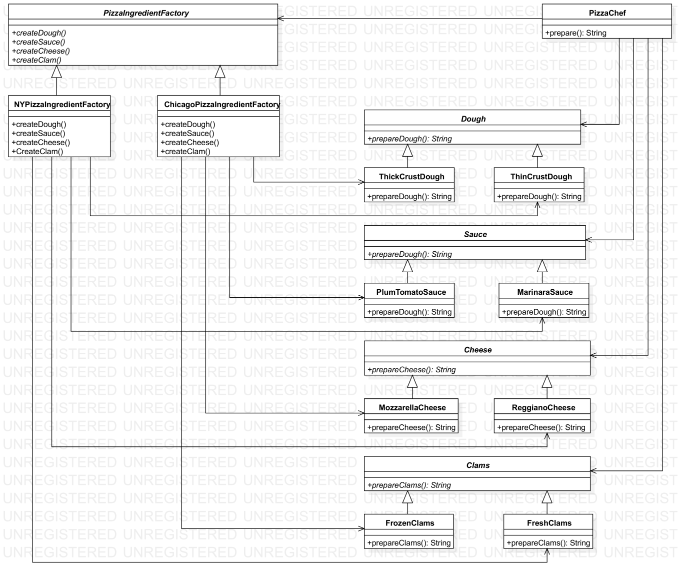
Abstract Factory: Applicability

- a system should be independent of how its products are created, composed, and represented.
- a system should be configured with one of multiple families of products.
- a family of related product objects is designed to be used together, and you need to enforce this constraint.
- you want to provide a class library of products, and you want to reveal just their interfaces, not their implementations.

Abstract Factory: Structure



Abstract Factory: Example



Abstract Factory: Example (contd.)

```
public abstract class PizzaIngredientFactory {
   public abstract Dough createDough();
   public abstract Sauce createSauce();
   public abstract Cheese createCheese();
   public abstract Clams createClams();
}
```

Abstract Factory: Example (contd.)

```
public class NYPizzaIngredientFactory extends
    PizzaIngredientFactory{

   public ThinCrustDough createDough(){
       return new ThinCrustDough();
   }

   public MarinaraSauce createSauce (){
       return new MarinaraSauce();
   }

   public ReggianoCheese createCheese(){
       return new ReggianoCheese();
   }

   public FreshClams createClams (){
       return new FreshClams();
   }
}
```

Abstract Factory: Example (contd.)

```
public class ChicagoPizzaIngredientFactory extends
    PizzaIngredientFactory{

   public ThickCrustDough createDough(){
      return new ThickCrustDough();
   }

   public PlumTomatoSauce createSauce (){
      return new PlumTomatoSauce();
   }

   public MozzarellaCheese createCheese(){
      return new MozzarellaCheese();
   }

   public FrozenClams createClams (){
      return new FrozenClams();
   }
}
```

Abstract Factory: Example (contd.)

```
public abstract class Dough {
    abstract String prepareDough();
}

public class ThickCrustDough extends Dough{

    public String prepareDough()
    {
        return "ThickCrust Dough";
    }
}

public class ThinCrustDough extends Dough{

    public String prepareDough()
    {
        return "ThinCrust Dough";
    }
}
```

Abstract Factory: Example (contd.)

```
public abstract class Sauce {
    abstract String prepareSauce();
}

public class PlumTomatoSauce extends Sauce{

    public String prepareSauce()
    {
        return "PlumTomato Sauce";
    }
}

public class MarinaraSauce extends Sauce{

    public String prepareSauce()
    {
        return "Marinara Sauce";
    }
}
```

Abstract Factory: Example (contd.)

```
public abstract class Cheese {
    abstract String prepareCheese();
}

public class MozzarellaCheese extends Cheese{

    public String prepareCheese()
    {
        return "Mozarella Cheese";
    }
}

public class ReggianoCheese extends Cheese{

    public String prepareCheese()
    {
        return "Reggiano Cheese";
    }
}
```

Abstract Factory: Example (contd.)

```
public abstract class Clams {
    abstract String prepareClams();
}

public class FrozenClams extends Clams{

    public String prepareClams()
    {
        return "Frozen Clams";
    }
}

public class FreshClams extends Clams{

    public String prepareClams()
    {
        return "Fresh Clams";
    }
}
```

Abstract Factory: Example (contd.)

```java
public class PizzaChef {

    private Dough doughType;
    private Sauce sauceType;
    private Cheese cheeseType;
    private Clams clamsType;

    public PizzaChef(PizzaIngredientFactory pizzaIngFac)
    {
        doughType = pizzaIngFac.createDough();
        sauceType = pizzaIngFac.createSauce();
        cheeseType = pizzaIngFac.createCheese();
        clamsType = pizzaIngFac.createClams();
    }
```

Abstract Factory: Example (contd.)

```
// PizzaChef (Contd.)
    public String prepare()
    {
        String myDough;
        String mySauce;
        String myCheese;
        String myClams;
        String outputPizza;

        myDough= doughType.prepareDough();
        mySauce = sauceType.prepareSauce();
        myCheese = cheeseType.prepareCheese();
        myClams = clamsType.prepareClams();

        outputPizza = myDough+", "+mySauce+", "+myCheese+",
            "+myClams;
        return outputPizza;
    }
}
```

Abstract Factory: Example (contd.)

```
import java.util.Scanner;

public class PizzaCustomer {

    private static PizzaChef myPizzaClient;
    private static PizzaIngredientFactory myPizza;

    public static void main(String a[]){

        System.out.println("What pizza you would like today?: ");
        Scanner in = new Scanner(System.in);
        String pizzaType = in.nextLine();
        String outputPizza;
```

Abstract Factory: Example (contd.)

```
//PizzaCustomer (contd.)

        if(pizzaType.equalsIgnoreCase("NY")){
            myPizza = new NYPizzaIngredientFactory();
        }
        else if(pizzaType.equalsIgnoreCase("Chicago")){
            myPizza = new ChicagoPizzaIngredientFactory();
        }
        else{
            System.out.println("Not a valid pizza type!");
            return;
        }
        myPizzaClient = new PizzaChef(myPizza);
        outputPizza = myPizzaClient.prepare();
        System.out.println(pizzaType+" is made with
             "+outputPizza);
    }
}
```

Abstract Factory

- Is that a Factory Method lurking inside the Abstract Factory?
  - The job of an Abstract Factory is to define an interface for creating a set of products.
  - Each method in that interface is responsible for creating a concrete product, and we implement a subclass of the Abstract Factory to supply those implementations.
  - So, factory methods are a natural way to implement your product methods in your abstract factories.