# 抽象工厂模式

## 一、基本概念 **1、定义：**

##### 定义一个大的工厂类，每个工厂类对应的实例不止一个，可以有多个，他可以根据参数的不同返回不同类的不同实例，被创建的实例通常都具有共同的父类。

##### 抽象工厂模式（Abstract Factory Pattern）是围绕一个超级工厂创建其他工厂。该超级工厂又称为其他工厂的工厂。这种类型的设计模式属于创建型模式，它提供了一种创建对象的最佳方式。

**抽象工厂模式的优势是可以对实例进行分类，大大减少新的工厂的创建，降低代码的复杂度。**

### 2、组成环节：

##### 1、AbstractFactory(抽象工厂):具体工厂的父类，负责实现创建所有产品的内部逻辑，提供工厂类的公共接口

##### 2、Product(抽象类产品)：具体产品类所有对象的父类，封装了产品对象的公共方法，描述了具体工厂的公开接口

##### 3、ProductA、ProductB…(具体产品)：抽象类产品的子类，可供具体工厂类进行调用，所有被创建的对象都是某个具体类的实例。它要实现抽象产品中声明的抽象方法。

**4、Factory A、Factory B…(具体工厂)：为核心部分，抽象工厂的子类，工厂类可以被外界直接调用，创建所需对象，实现工厂方法创建实例产品。在抽象工厂里面，一个工厂可以抽象成一个大类从而去对应多个实例。**

## 二、go语言中实现

##### 1、首先定义一个absractfactory类型

type AbstractFactory interface {  
 Get(s string) Product  
}

##### 2、在定义两个具体类型的factory

type AFactory struct {}  
type TFactory struct {}

##### 3、定义一个统一product的接口，并定义其实现的方法。

type Product interface {  
 Do()  
}

##### 4、定义product内的具体方法

type T1 struct {}  
type T2 struct {}

type A1 struct {}  
type A2 struct {}

##### 5、对具体的product进行do的实现。

func (t T1)Do() {  
 fmt.Printf("T1 do.\n")  
}  
  
func (t T2)Do() {  
 fmt.Printf("T2 do.\n")  
}  
  
  
func (a A1)Do() {  
 fmt.Printf("A1 do.\n")  
}  
  
func (a A2)Do() {  
 fmt.Printf("A2 do.\n")  
}

##### 6、对具体的工厂方法进行Get的实现。

func (a AFactory)Get(s string) Product {  
 switch s {  
 case *A1NAME*:  
 return A1{}  
 case *A2NAME*:  
 return A2{}  
 }  
 return nil  
}  
  
func (a TFactory)Get(s string) Product {  
 switch s {  
 case *T1NAME*:  
 return T1{}  
 case *T2NAME*:  
 return T2{}  
 }  
 return nil  
}

##### 7、开始进行方法的调用。

func main() {  
 var t TFactory  
 var a AFactory  
 t.Get(*T1NAME*).Do()  
 t.Get(*T2NAME*).Do()  
 a.Get(*A1NAME*).Do()  
 a.Get(*A2NAME*).Do()  
}

##### 8、结果的输出。

T1 do.

T2 do.

A1 do.

A2 do.