简单工厂：

简单工厂模式（simple factory pattern） 是指由一个工厂对象决定创建出哪一种产品类的实例。

属于创建型模式，但它不属于GOF 23中设计模式

Public class test {

Couse couse = new Couse(); couse.record();

ICouse couse = new CouseFactory().create(“java”); couse.record();

}

Public interface ICouse{void record();}

Public class JavaCouse extends ICouse{ void record(){System.out.println(“java ”)}}

Class CouseFactory{

public ICouse create(String name){ if(“java”.equals(name)){return new JavaCouse();} else if(“python”.equals(name)){ return PythonCouse();} }

If(null ==className || “”.equals(className)){

Return (ICouse)Class.forName(className).newInstance();

}

Public ICouse create(Class<? Extends ICouse>

clazz){

If(Null != clazz){

Return clazz.newINstance();

}

}

}

简单工厂适用场景：

工厂类负责创建的对象较少

客户端只需要传入工厂类的参数，对于如何创建对象的逻辑不需要关心。

优点：只需要传入一个正确的参数，就可以获取你所需要的对象无须知道其创建的细节。

缺点：工厂类的职责相对过重，增加新的产品时需要修改工厂类的判断逻辑违背开闭原则。不易于扩展过于复杂的产品结构。

工厂方法模式定义：

工厂方法模式（Fatory Method Pattern）是指定以一个创建对象的接口，但让实现这个接口的类来决定实例化哪个类，工厂方法让类的实例化推迟到子类中进行。

属于创建型设计模式。

工厂方法模式适用场景：创建对象需要大量重复的代码。

客户端（应用层）不依赖于产品类实例如何被创建、实现等细节一个类通过其自雷来指定创建哪个对象。

工厂方法模式的优点：用户只需关心所需产品对应的工厂，无须关心创建细节。加入新产品

符合开闭原则，提高了系统的可扩展性。

工厂方法模式的缺点：类的个数容易过多，增加了代码结构的复杂度。增加了系统的抽象性和理解难度。

抽象工厂模式：

是指提供一个创建一系列相关或相互依赖对象的接口，无须指定

抽象工厂模式的适用场景：客户端不依赖于产品类实例如何被创建、实现等细节。强调一系列相关的产品对象（属于同一产品族）一起使用创建对象需要大量重复的代码。提供一个产品类的库，所有的产品以同样的接口出现，从而使客户端不依赖于具体实现。

抽象工厂模式的缺点：规定了所有可能被创建的产品集合，产品族中扩展新的产品困难，需要修改抽象工厂的接口。增加了系统的抽象性和理解难度。 具体产品在应用层代码隔离，无须关系创建细节，将一个系列的产品族同一到一起创建

抽象工厂模式的缺点：规定了所有可能被创建的产品集合，产品族中扩展新的产品困难，需要修改抽象工厂的接口。增加了系统的抽象性和理解难度。

【简单工厂：产品的工厂

工厂方法：工厂的工厂

抽象工厂：复杂产品的工厂】