1. 工厂设计模式

在程序设计的时候，我们要自始至终本着一个原则：主方法是客户端，对于程序的修改不应该影响客户端。

所以我们应该尽可能地将解决问题的代码不要放在主方法中，而是分类分成不同的模块。

观察下面的代码：

```
// 定义一个接口，接口里只有一个抽象方法
interface IFruit {
    public void eat();
}

// 定义一个实现类
class Apple implements IFruit{

    @Override
    public void eat() {
        System.out.println("eat apple");
    }

}


public class Test {
    public static void main(String[] args) throws ClassNotFoundException, InstantiationException,
IllegalAccessException {
        // 调用实现类
        Apple apple = new Apple();
        apple.eat();
    }
}
```

如果在上面的代码中我们新添加了一个类 Banana，这时要创建它的对象，又得在主方法中再加：

```
    Banana banana = new Banana();
```

如果再添加新的类呢。这时主方法中也会变得十分臃肿，而且在使用主方法客户端的客户看来，他并不想使用这么多类以及记住这么多的类名，怎么解决这个问题呢。

new是整个开发过程中最大的耦合原因，而在开发之中想要进行解耦和的关键再与要引入一个第三方，这个第三方就可以使用一个Factory类来描述。

```
class Factory {
    // Factory是一个工具类，我们只是使用它的其中的方法，所以隐藏它的构造
    private Factory() {
    }

    // 通过传入的类名，来创建不同的类
    public static IFruit getInstance(String title) {
```

```
        if("banana".equals(title)) {
            return new Banana();
        } else if("apple".equals(title)) {
            return new Apple();
        } else {
            return null;
        }
    }
}

public class Test {
    public static void main(String[] args) throws ClassNotFoundException, InstantiationException,
IllegalAccessException {
        // 这里我们只需要传入相应的类名，就可以得到我们想要的类
        IFruit iFruit = Factory.getInstance("apple");
        iFruit.eat();
    }
}
```

这种设计模式称为工厂设计模式。

反射与工厂类结合

虽然上面的的工厂模式已经解决了很大的设计问题，但它还是存在一些瑕疵：

> 在增加了新的类之后，就必须在 getInstance() 中再增加返回该新增类实例化对象的语句，比较麻烦。

我们可以使用反射来解决这个问题。

即利用使用 Class.forName() 来取得类的Class对象，然后利用 Class 类的 newInstance()方法 实例化对象。

```
class Factory {
    // Factory是一个工具类，我们只是使用它的其中的方法，所以隐藏它的构造
    private Factory() {
    }

    public static IFruit getInstance(String title) throws InstantiationException,
IllegalAccessException, ClassNotFoundException {
        IFruit iFruit = null;

        iFruit = (IFruit)Class.forName(title).newInstance();
        return iFruit;
    }
}

public class Test {
    public static void main(String[] args) throws ClassNotFoundException, InstantiationException,
IllegalAccessException {
        // 通过类名的全名称进行创建类对象
        IFruit iFruit = Factory.getInstance("test.Apple");
        iFruit.eat();
    }
}
```

在工厂类中引入反射后，每当新增接口子类，无需去修改工厂类代码就可以很方便的进行接口子类扩容。
以上这种工厂类代码我们称之为简单工厂模式。