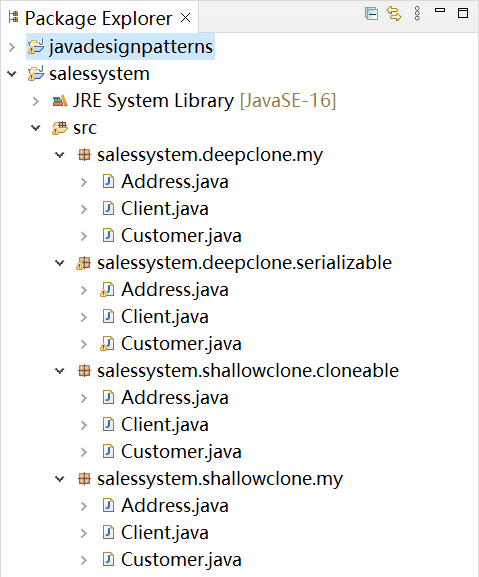
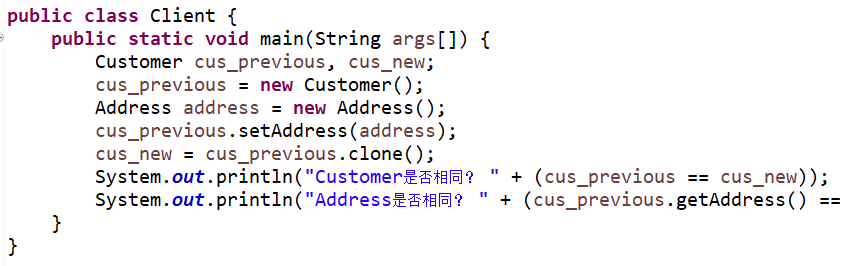
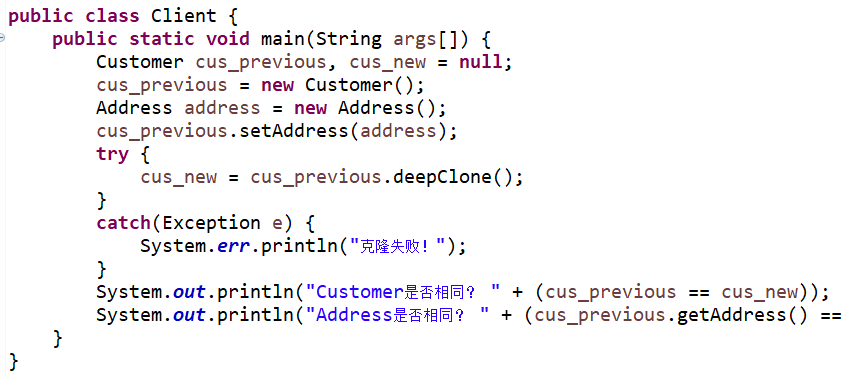
* shallowclone.cloneable和deepclone.serializable为使用Java中的Cloneable接口以及Serializable接口的实现，.my为不采用java中的接口的实现：



* 浅拷贝两种实现使用相同的客户端测试代码，为：

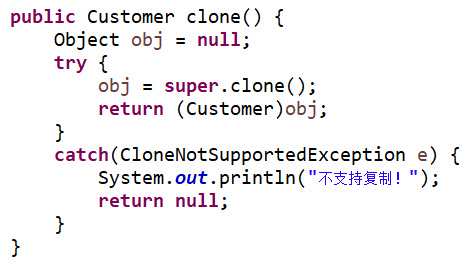


* 深拷贝两种实现使用相同的客户端测试代码，为：

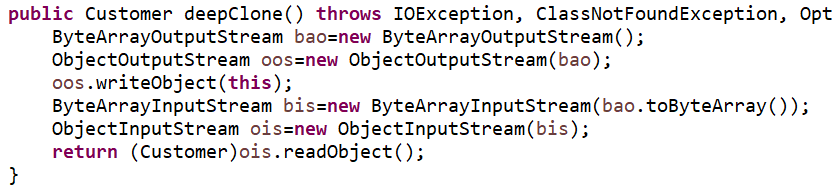


1. 使用Java中的Cloneable接口以及Serializable接口实现浅拷贝和深拷贝。

浅拷贝：仿照java design patterns中提供的示例为Customer提供clone()

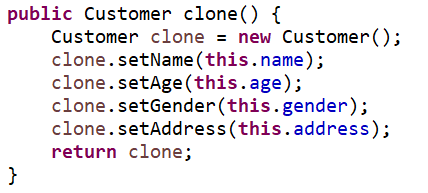


深拷贝：仿照java design patterns中示例为Customer提供deepClone()

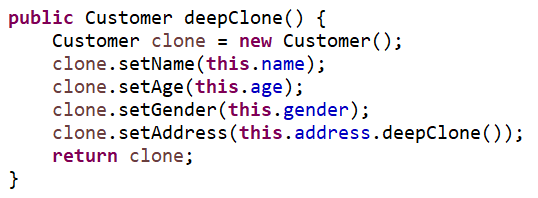


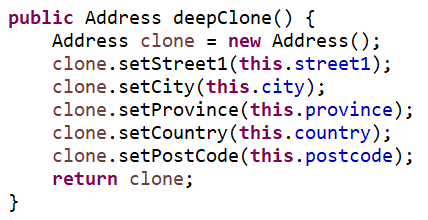
1. 不采用Java中的接口实现深拷贝以及浅拷贝的原型模式。

浅拷贝：new一个Customer对象并对成员逐个调用setter函数，将每个成员变量都set为this中对应项（此处setxx均用this.xx = xx实现）



深拷贝：其他同浅拷贝，setAddress的参数改为this.address.deepClone()，另外为Address添加deepClone函数，即需要手动对Address也进行深拷贝。





Address中的成员均按与浅拷贝相同的方式进行set即可。

* 运行结果：对两种实现的浅拷贝均有：

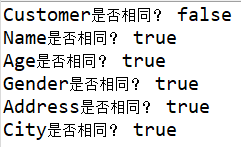


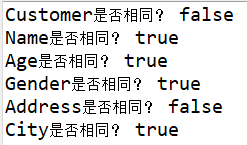
对两种实现的深拷贝均有：



* 思考：对于浅克隆（Shallow Clone），当原型对象被复制时，只复制它本身和其中包含的值类型的成员变量，而引用类型的成员变量并没有复制。即原型对象的成员变量不分值和地址类型，一律直接按照值的方式进行了复制。而对于深克隆（Deep Clone），除了对象本身被复制外，对象所包含的所有成员变量也将被复制。其中值直接复制，引用则递归地对引用对象进行复制。

那么，int是值自然不必说，String和Boolean是值还是引用？这些成员在进行深拷贝时还可以使用xx = this.xx和this.xx = xx的直接赋值方式吗？或者是像Address类一样需要另外提供deepClone()函数？观察使用Java接口实现的拷贝中Name（String）、Age（int）、Gender（Boolean）以及Address成员City，发现在深浅拷贝中它们均相同，只有Address在深拷贝中不同。





对不使用Java接口实现的拷贝进行输出，结果仍然相同。于是得出结论，在自己实现的深拷贝中继续对String和Boolean使用直接赋值是可以的。

Factory Method模式和Abstract Factory模式的区别在哪？

一般哪些情况下适合用前者，哪些情况下适合用后者？

区别：在Factory Method模式中，每个具体工厂，只能生产一种产品；而在Abstract Factory模式中，一个工厂可以生产一系列产品（一族产品）。

当一个工厂等级结构可以创建分属于不同产品等级结构的一个产品族中的所有对象时，Abstract Factory模式比Factory Method模式简单高效。总的来说，后者适用于系统中有多于一个产品族，每次只使用其中一个，且同一个产品族的产品总是一起使用的情况。此外，后者还要求产品等级结构稳定，即在设计完成之后不会向系统中增加新的或者删除已有的产品等级结构。如果系统中的产品不能划分成这样的产品族，或者产品等级结构经常变更的情况下，使用前者更适合。