OOAD Assignment5

抽象工厂模式（工厂模式 + 多个抽象产品）

提供一个创建一系列相关或相互依赖对象的接口，而无需指定他们具体的类

包含抽象产品以及抽象工厂

不同的工厂会对不同的产品有着不同的实现，通常是在运行时刻创建一个工厂的实例，由这个具体的工厂来创建具有特定实现的产品的对象。

抽象工厂模式的优缺点

优点

- 具体的工厂类只需要在初始化的时候出现一次，只需要改变具体工厂就可以使用不同的产品配置
- 具体的创建实例过程与客户端分离，客户端通过抽象接口操纵实例，产品的具体类名也被具体工厂的实现分离，不会在客户的代码当中

缺点

- 增加产品时非常不方便，需要针对每个工厂增加相应的产品，且需要增加相应的抽象类，以及改动每个工厂，增加对其的实现。

单例模式

保证一个类仅有一个实例，并提供一个访问它的全局访问点

让类自身负责保存它的唯一实例，保证没有其他实例可以被创建，并且它可以提供一个访问该实例的方法

单例模式的优缺点

优点

- 保证一个类只有唯一的一个实例，减少资源过度分配
- 可以严格地控制客户怎样访问一个类的实例以及何时进行访问
- 不适用于变化的对象，如果同一类型的对象总是要在不同的用例场景发生变化，单例就会引起数据的错误，不能保存彼此的状态。
- 一个类的职责过重，违背“单一职责原则”

单例+依赖注入+抽象工厂

```
public class DaoFactoryImpl implements DaoFactory {
    private static DaoFactoryImpl instance = null;
    private final Class<?> staffDaoClass;
    private final Class<?> computerDaoClass;

    public static synchronized DaoFactoryImpl getInstance(){
        if(instance == null){
```

```
            instance = new DaoFactoryImpl();
        }
        return instance;
    }

    private DaoFactoryImpl(){
        Properties properties =
PropertiesReader.readProperties("/Users/leo/Downloads/src/Singleton/resources.properties");
        try {
            staffDaoClass = Class.forName("dao." + properties.getProperty("StaffDao"));
            computerDaoClass = Class.forName("dao." +
properties.getProperty("ComputerDao"));
        } catch (ClassNotFoundException e) {
            throw new RuntimeException(e);
        }

    }
```