* 下午：3:30进行测试 4:00正式开始
* 网络环境
* 建议先把音量调到最大
* 调整屏幕布局
  + 客户端菜单 – 布局
* 调成比800\*600更大的分辨率
* 定期讲课
  + http://www.bjsxt.com
* 设计模式之工厂系列

**假设**

1. 初步具备面向对象的设计思维
   1. 网络课程第二课
   2. 视频第三章
2. 了解多态的概念
   1. 网络课程第二课
   2. 视频第三章
3. 了解JDK集合类（容器类）（非必须）
   1. 视频第七章
   2. 关于交通工具
4. 只给司机一辆车（单例、多例）
5. 任意定制交通工具的类型和生产过程
6. 系列产品（车、武器、食品补给……）
7. bean工厂
   1. 基于接口的类型任意定制
   2. 一定程度的生产过程定制（AOP）

* 作业
* 自己阅读观察者模式
  + <http://www.bjsxt.com/> 视频下载
* 自行生产坦克系列
  + 适合掌握了坦克项目的同学
* 自学JDOM
  + 入门即可

单例 懒汉式和饿汉式

懒汉式：

package com.test.designpattern.\_005;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class Singleton {  
 private static Singleton instance;  
 private Singleton(){}  
  
 private Singleton getInstance(){  
 if(instance==null){  
 synchronized (Singleton.class){  
 if(instance==null){  
 instance=new Singleton();  
 }  
 }  
 }  
  
 return instance;  
 }  
}

饿汉式：

package com.test.designpattern.\_005;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class Singleton2 {  
 private static Singleton2 instance=new Singleton2();  
 private Singleton2(){}  
  
 public Singleton2 getInstance(){  
 return instance;  
 }  
}

静态方法返回实例，称作静态工厂方法，因为只有一个实例所以又称作单例模式；

工厂方法可以写在具体的那个类中，也可以单独为一个类返回一些。

如果在一个类中维护了多个实例，然后在每次获取实例的时候从中取出一个，可以叫做多例，具体的例子就是jdbc的连接池。

简单工厂：

package com.test.designpattern.\_005.factory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public interface Movable { #类接口  
 void run();  
}

package com.test.designpattern.\_005.factory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public abstract class VehicleFactory { #工厂抽象类  
 abstract Movable create();  
}

package com.test.designpattern.\_005.factory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class Car implements Movable {  
 public void run(){  
 System.out.println("plane running ...");  
 }  
}

package com.test.designpattern.\_005.factory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class Plane implements Movable {  
 public void run(){  
 System.out.println("car flying ...");  
 }  
}

package com.test.designpattern.\_005.factory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class CarFacory extends VehicleFactory {  
 public Movable create(){  
 return new Car();  
 }  
}

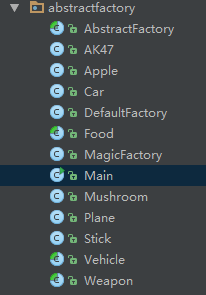
package com.test.designpattern.\_005.factory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class PlaneFactory extends VehicleFactory {  
 public Movable create(){  
 return new Plane();  
 }  
}

调用：

package com.test.designpattern.\_005.factory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class Main {  
 public static void main(String[] args){  
 VehicleFactory factory=new CarFacory(); #用哪个交通工具使用哪种工厂  
 Movable movable=factory.create();  
 movable.run();  
 }  
}

抽象工厂：

在编程的时候能够批量替换类的生成



package com.test.designpattern.\_005.abstractfactory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public abstract class AbstractFactory {  
 abstract public Vehicle createVehicle();  
 abstract public Food createFood();  
 abstract public Weapon createWeapon();  
}

package com.test.designpattern.\_005.abstractfactory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public abstract class Vehicle {  
 abstract public void run();  
}

package com.test.designpattern.\_005.abstractfactory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public abstract class Food {  
 abstract public void taste();  
}

package com.test.designpattern.\_005.abstractfactory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public abstract class Weapon {  
 abstract public void shoot();  
}

package com.test.designpattern.\_005.abstractfactory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class DefaultFactory extends AbstractFactory{  
 @Override  
 public Vehicle createVehicle() {  
 return new Car();  
 }  
  
 @Override  
 public Food createFood() {  
 return new Apple();  
 }  
  
 @Override  
 public Weapon createWeapon() {  
 return new AK47();  
 }  
}

package com.test.designpattern.\_005.abstractfactory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class MagicFactory extends AbstractFactory {  
 @Override  
 public Vehicle createVehicle() {  
 return new Plane();  
 }  
  
 @Override  
 public Food createFood() {  
 return new Mushroom();  
 }  
  
 @Override  
 public Weapon createWeapon() {  
 return new Stick();  
 }  
}

package com.test.designpattern.\_005.abstractfactory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class Car extends Vehicle {  
 @Override  
 public void run() {  
 System.out.println("car run");  
 }  
}

package com.test.designpattern.\_005.abstractfactory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class Plane extends Vehicle {  
 @Override  
 public void run() {  
 System.out.println("plane run");  
 }  
}

package com.test.designpattern.\_005.abstractfactory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class Apple extends Food {  
 @Override  
 public void taste() {  
 System.out.println("apple taste");  
 }  
}

package com.test.designpattern.\_005.abstractfactory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class Mushroom extends Food {  
 @Override  
 public void taste() {  
 System.out.println("mushroom taste");  
 }  
}

package com.test.designpattern.\_005.abstractfactory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class AK47 extends Weapon {  
 @Override  
 public void shoot() {  
 System.out.println("AK47 shoot");  
 }  
}

package com.test.designpattern.\_005.abstractfactory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class Stick extends Weapon {  
 @Override  
 public void shoot() {  
 System.out.println("stick shoot");  
 }  
}

调用：

package com.test.designpattern.\_005.abstractfactory;  
  
/\*\*  
 \* Created by DaiYan on 2017/9/14.  
 \*/  
public class Main {  
 public static void main(String[] args) {  
// AbstractFactory factory=new DefaultFactory();  
 AbstractFactory factory=new MagicFactory();  
  
 Vehicle vehicle=factory.createVehicle();  
 vehicle.run();  
 Food food=factory.createFood();  
 food.taste();  
 Weapon weapon=factory.createWeapon();  
 weapon.shoot();  
 }  
}

抽象工厂适用于产生产品系列的（如一次产生食物 交通工具 武器）

简单工厂能够产生单个的产品，而抽象工厂不能，简单工厂用于产生产品系列的时候会导致工厂泛滥

Spring的bean工厂

Xml配置文件->创建bean工厂

{

"name": "Attach to Process",

"type": "node",

"request": "attach",

"processId": "53426",

"port": 9229

}