|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | interface Car {// 车的父类      public void driver();// 开车  }  class Benz implements Car {// 奔驰车      public void driver() {          System.out.println("今天咱开奔驰！");      }  }  class Bike implements Car {// 自行车      public void driver() {          System.out.println("唉，现在经济危机，只能骑自行车了呀！");      }  }  class Bmw implements Car {// 宝马      public void driver() {          System.out.println("今天开宝马吧！");      }  }  class Driver {// 车的工厂      public static Car driverCar(String s) throws Exception {          if (s.equalsIgnoreCase("Benz")) {// 判断传入参数返回不同的实现类              return new Benz();          } else if (s.equalsIgnoreCase("Bmw")) {              return new Bmw();          } else if (s.equalsIgnoreCase("Bike")) {              return new Bike();          } else {              throw new Exception();// 抛出异常          }      }  }  public class TextSimpleFactory {// 操作工厂模式的类      public static void main(String[] args) {// java程序主入口处          try {              Car car = Driver.driverCar("Bike");// 调用方法返回车的实例              System.out.println("经理，今天开什么车呀？");              car.driver();// 调用方法开车          } catch (Exception e) {// 捕获异常              System.out.println("开车出现问题......");          } finally {// 代码总被执行              System.out.println("......");          }      }  } |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | package net.xsoftlab.baike;  interface fruit {      public abstract void eat();  }  class Apple implements fruit {      public void eat() {          System.out.println("Apple");      }  }  class Orange implements fruit {      public void eat() {          System.out.println("Orange");      }  }  class Factory {      public static fruit getInstance(String ClassName) {          fruit f = null;          try {              f = (fruit) Class.forName(ClassName).newInstance();          } catch (Exception e) {              e.printStackTrace();          }          return f;      }  } |

 \* 对于普通的工厂模式当我们在**添加一个子类**的时候，就需要对应的**修改工厂类**。 当我们添加很多的子类的时候，会很麻烦。

 \* Java 工厂模式可以参考

 \* http://baike.xsoftlab.net/view/java-factory-pattern

 \*

 \* 现在我们利用**反射机制实现工厂模式**，可以在不修改工厂类的情况下添加任意多个子类。

 \*

 \* 但是有一点仍然很麻烦，就是需要知道**完整的包名和类名**，这里可以使用properties配置文件来完成。

但和IOC不同，仍在类和类之间有依赖，不存在第三方容器管理它们的关系。