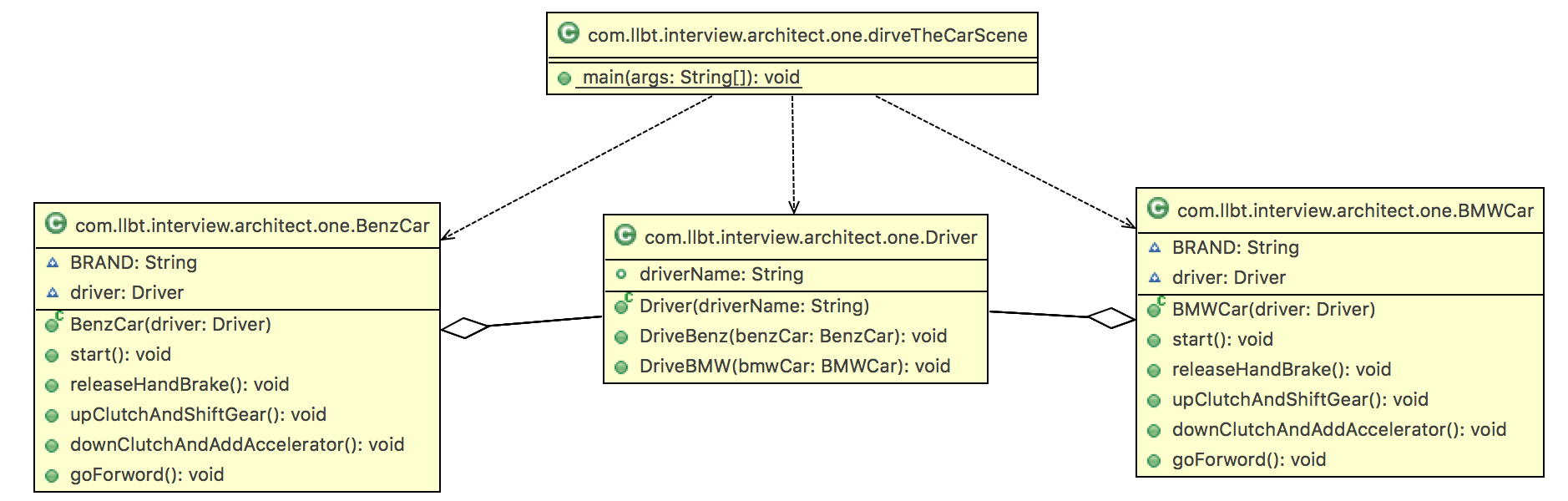
以下Java程序，模拟了小明同学到4S店试驾各种品牌汽车的场景，

UML结构为：



主类（ dirveTheCarScene.java ）运行结果为：



由于时间紧急，得知需求后立刻投入了开发，以至于没有太多架构思考。如果本‘试驾系统’由你来负责继续开发，你会如何重构。

1. 请说出你看到的问题？（任何你觉得有问题的地方）
2. 你会如何重构？（先给出对UML的设计修改方案）

package com.llbt.interview.architect.one;

/\*\*

 \* 模拟小明试驾各种汽车的场景

 \*/

public class dirveTheCarScene {

public static void main(String[] args) {

System.out.println("========== 模拟小明试驾奔驰汽车");

Driver xiaoMingDriver = new Driver("小明") ;

BenzCar benzCar = new BenzCar(xiaoMingDriver) ;

xiaoMingDriver.DriveBenz(benzCar);

System.out.println("========== 模拟小明试驾宝马汽车");

BMWCar bmwCar = new BMWCar(xiaoMingDriver) ;

xiaoMingDriver.DriveBMW(bmwCar);

}

}

/\*\*

 \* 模拟奔驰汽车

 \*/

public class BenzCar {

String BRAND = "奔驰" ;

Driver driver ;

public BenzCar( Driver driver){

System.out.println(driver.driverName + " 准备试驾 "+BRAND);

this.driver = driver ;

}

/\*\*

\* 启动汽车

\*/

public void start() {

System.out.println(this.driver.driverName +"：汽车启动了");

}

/\*\*

\* 放开手刹

\*/

public void releaseHandBrake() {

System.out.println(this.driver.driverName +"：放开了手刹");

}

/\*\*

\* 踩离合、挂档

\*/

public void upClutchAndShiftGear() {

System.out.println(this.driver.driverName +"：踩离合、挂档");

}

/\*\*

\* 放离合、给油

\*/

public void downClutchAndAddAccelerator() {

System.out.println(this.driver.driverName +"：放离合、给油");

}

/\*\*

\* 前进

\*/

public void goForword() {

System.out.println(this.driver.driverName +"：前进");

}

}

/\*\*

 \* 模拟宝马汽车

 \*/

public class BMWCar {

String BRAND = "宝马" ;

Driver driver ;

public BMWCar( Driver driver){

System.out.println(driver.driverName + " 准备试驾 "+BRAND );

this.driver = driver ;

}

/\*\*

\* 启动汽车

\*/

public void start() {

System.out.println(this.driver.driverName +"：汽车启动了");

}

/\*\*

\* 放开手刹

\*/

public void releaseHandBrake() {

System.out.println(this.driver.driverName +"：放开了手刹");

}

/\*\*

\* 踩离合、挂档

\*/

public void upClutchAndShiftGear() {

System.out.println(this.driver.driverName +"：踩离合、挂档");

}

/\*\*

\* 放离合、给油

\*/

public void downClutchAndAddAccelerator() {

System.out.println(this.driver.driverName +"：放离合、给油");

}

/\*\*

\* 前进

\*/

public void goForword() {

System.out.println(this.driver.driverName +"：前进");

}

}

/\*\*

 \* 模拟“司机”

 \*/

public class Driver {

/\*\* 司机名称 \*/

public String driverName = null ;

public Driver( String driverName ) {

this.driverName = driverName ;

}

/\*\*

\* 模拟司机开奔驰车

\* @param bmwCar

\*/

public void DriveBenz(BenzCar benzCar) {

// 启动汽车

benzCar.start();

// 放手刹

benzCar.releaseHandBrake();

// 踩离合、挂档

benzCar.upClutchAndShiftGear();

// 放离合、给油

benzCar.downClutchAndAddAccelerator();

// 汽车前进

benzCar.goForword();

}

/\*\*

\* 模拟司机开宝马车

\* @param bmwCar

\*/

public void DriveBMW( BMWCar bmwCar){

//启动汽车

bmwCar.start() ;

//放手刹

bmwCar.releaseHandBrake() ;

//踩离合、挂档

bmwCar.upClutchAndShiftGear() ;

//放离合、给油

bmwCar.downClutchAndAddAccelerator() ;

//汽车前进

bmwCar.goForword() ;

}

}

参考答案：

1. 类命名规范应该是驼峰规则；
2. 接口命名规范应该是lower驼峰规则；
3. Driver和Car应该设计为接口，在主类中通过接口编程；
4. Driver类提供一个统一的driver接口，用来开各种汽车（建议）；
5. Driver类实现抽象类，用模版方法包装开车的过程；
6. Car类中不应该有司机，可以由主类来完成司机＋汽车这样的组合逻辑，保证Car和Driver的是职责单一的；
7. 画出的UML中能体现接口、抽象、类的单一职责、模版模式、工程模式（考虑多种品牌的汽车）；