实验1 基础实验

1 实验目的

熟悉类、接口、继承、实现、依赖、关联、聚合、组合等要素的UML表达方式，学会用UML及工具对类图进行设计。熟悉JavaFX中AnimationTimer的用法。

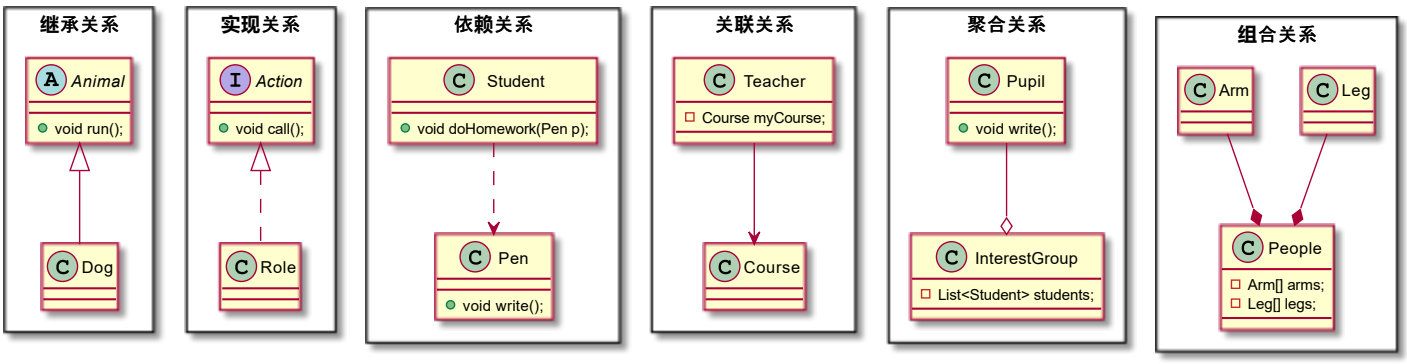
2 实验环境

开发环境：JDK 8.0（或更高版本，高版本要下载独立JavaFX）
开发工具：Eclipse
设计工具：StarUML（或PlantUML）

3 基础知识

3.1 类图中的常见关系

- **泛化关系**（Generalization）或称继承关系，表示一般与特殊的关系，指定了子类如何特化父类的所有特征和行为。
- **实现关系**（Implementation）是一种类与接口的关系, 表示类是接口所有特征和行为的实现。
- **依赖关系**（Dependency）表示一个类中会用到另一个类，类之间相对独立。
- **关联关系**（Association）是一种拥有的关系（has-a），它使一个类知道另一个类的属性和方法。
- **聚合关系**（Aggregation）是整体与部分的关系，且部分可以离开整体而单独存在。（whole-part, part can be shared）
- **组合关系**（Composition）是整体与部分的关系，但部分不能离开整体而单独存在。（whole-part, part can not be shared）



3.2 计时器的基本用法

JavaFX提供计时器类`AnimationTimer`实现动画功能，每帧会执行`handle(long now)`方法，`start()`和`stop()`用于控制计时器的活动状态。如下代码，实现"Hello"文本在(100,100)至(100,200)之间移动的动画。

```
public class TimerTest extends Application {
        @Override
        public void start(Stage primaryStage) throws Exception {
                AnchorPane p = new AnchorPane();
                // 创建一个画布用于绘画
                Canvas c = new Canvas();
                p.getChildren().add(c);
                c.heightProperty().bind(p.heightProperty());
                c.widthProperty().bind(p.widthProperty());

                // 获取用于调用绘画功能的图形上下文对象
                GraphicsContext gc = c.getGraphicsContext2D();

                // 创建计时器匿名类对象，在handle方法中定义每帧的行为
                AnimationTimer at = new AnimationTimer() {
                        int x = 100;
                        int y = 100;
                        int add = 1;
                        @Override
                        public void handle(long now) {
                                // 用白色清除上一帧的画布
                                gc.setFill(Color.WHITE);
                                gc.fillRect(0,0, c.getWidth(),c.getHeight());
                                gc.setFill(Color.BLACK);
                                // 设置字体，绘制一个文本
                                gc.setFont(new Font(22));
                                gc.fillText("Hello", x, y);
                                // 更新位置信息
                                x += add;
                                if(x == 200) {
                                        add = -1;
                                }else if(x == 100) {
                                        add = 1;
                                }
                        }
                };
                at.start();

                primaryStage.setScene(new Scene(p));
                primaryStage.show();
        }
        public static void main(String[] args) {
                launch(args);
        }
}
```

4 实验内容

实验1 类图分析

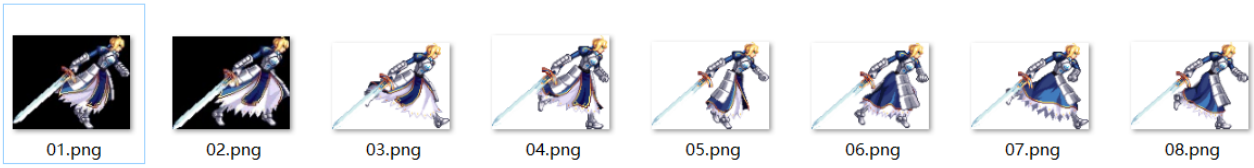
> 问题描述：假设一个计算机测试部门根据电脑类型分为多个产品测试线，每个测试员可能加入多条测试线。测试过程中主要针对电脑的中央处理器、内存、硬盘三个组件进行测试，现在部门主要测试A和B两种型号的电脑。试分析上述场景并画出类图。

对象	命名
部门	Department
测试线	TestLine
测试员	Tester
电脑	Computer
处理器	CPU
内存	Memery
硬盘	HardDesk

提示：这里部门和测试线之间是组合关系，测试员和测试线是聚合关系，测试员需要实现`test(Computer c)`功能。例如：

```
public boolean test(Computer c) {
        CPU cpu = c.getCPU();
        Memery m = c.getMemery();
        HardDesk hd = c.getHD();
        boolean status = true;
        switch(cpu.getVersion()) {
        // 分类型运行测试CPU，若运行有错误则将status设置为false
        }
        switch(m.getVersion()) {
        // 分类型运行测试内存，若运行有错误则将status设置为false
        }
        switch(hd.getVersion()) {
        // 分类型运行测试硬盘，若运行有错误则将status设置为false
        }
        return status;
}
```

实验2 角色动画

> 问题描述：给出人物奔跑的8张静态图片，编写程序实现奔跑动画。
>
> 
>

提示：`GraphicsContext`类提供`drawImage()`用于绘制图片，解题的要点在于需要考虑每一帧绘制哪一张图片。

5 实验要求

5.2 实验评价

1、完成实验内容（60%）
2、对实验思路进行阐述（20%）
3、对实验过程进行总结（20%）

5.1 实验报告

1、实验1完成UML图设计，截取清晰的设计图到报告中；

2、实验2完成程序实现，达到实验效果，附上源码和解题思路。