作业 3

本作业的目的是练习使用诸如事件、动画和实例变量。你将首先实现两个小练习作为热身问题。然后，你的主要任务将是实现经典的街机游戏——突破。下面更详细地描述了动画游戏。

**练习 #1: 写一个方法判断一个整数是否为素数**

你需要写一个方法实现这个功能，你输入一个比1大的正整数给方法，方法返回true或者false 来告诉别人这个正整数是不是素数。

（素数：除了1和它本身以外，不能被其它整数整除的整数。）

类：PrimeChecker.java

程序开始执行方法 run();

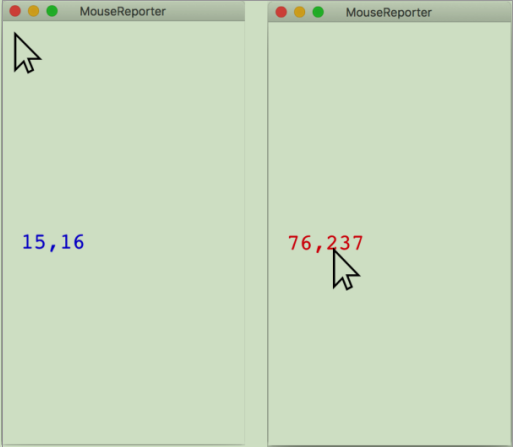
要实现的方法名称：isPrime(int number)

**练习 #2: 鼠标位置报告器**

写一个报告鼠标位置坐标的小程序。

类：MouseReporter.java

在屏幕左侧中间位置创建一个GLable对象，用来显示鼠标当前的坐标, 鼠标移动时更新GLable上的坐标。当鼠标移动到GLable对象上时，坐标文字颜色为红色，其它情况为绿色。



1. 写一个类MouseReporter;
2. 创建一个GLable对象，将它添加到窗口中；
3. 调用GLable对象的setLable()方法，显示鼠标的坐标位置；
4. 在指定的坐标位置获取图形化对象（如：GLable），需要使用到GraphicsProgram类的方法：

public GObject getElementAt(double x, double y)

**练习#3 Breakout**

这是一个比较大的作业，但只要您遵循以下几条建议，就完全可以实现：

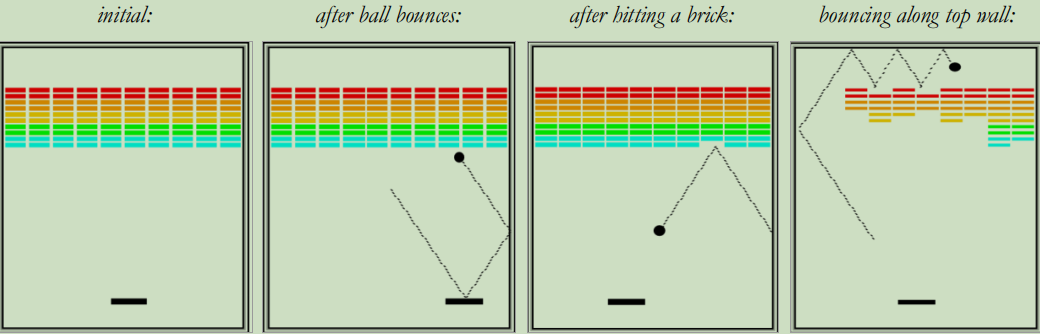
• 尽早开始，这个需要在一个周内完成，不要拖到最后一刻才做。

• 按照讲义把整个开发过程划分成若干阶段，一个阶段一个阶段地做，最后完成整个项目。

• 每写完一个功能的代码，就测试一下，测试通过后，再继续下一个功能开发。

**游戏机制**

在游戏中，玩家控制一个平放着的木板，通过鼠标可以左右移动木板。一个在窗口中移动的球，窗口上方有10排砖块，每排10块砖。球打中砖块时，打中的砖块消失，游戏的目标是清除所有的砖块。移动的木板用来防止球调入底部。

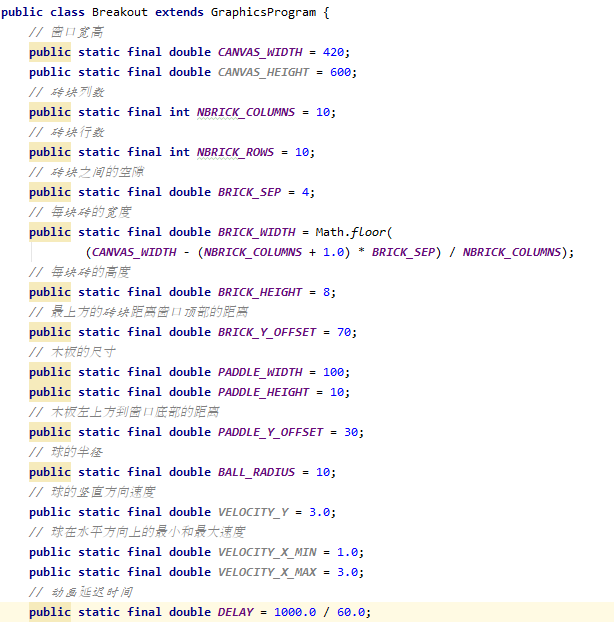


规则：

1. 黑球开始时出现在窗口的中间位置，点击屏幕后，球以一个随机的角度向下落；
2. 底部黑色木板初始位置为底部水平中间位置，玩家通过鼠标控制木板左右移动；
3. 黑球碰到木板会被弹起；
4. 黑球碰到窗口底部地面，游戏挑战失败一次；
5. 黑球碰到左右及顶部窗口墙壁，则会被反弹；
6. 黑球碰到带色砖块，碰到的砖块消失，球被反弹；
7. 带色砖块全部被清楚，游戏胜利；
8. 黑球碰到窗口底部墙壁3次，游戏失败。

这项作业的成功完成将取决于将问题分解成若干可管理的部分，并在进行下一项任务之前让之前每个部分正确运行。接下来的几节描述了一个合理的阶段性方法来解决这个问题。

先来看一下我们开始项目中为你提供的Breakout.java 类：



**方法：**

由于这是一个相对复杂的程序，我们强烈建议您分几个阶段开发和测试它，始终确保您有一个在每个阶段之后正确编译和运行的程序。在这里我们建议的几个开发阶段，在讲义中对每个阶段的开发有更详细地讨论：

1) 砖块

2) 木板

3) 球和反弹

4) 碰撞

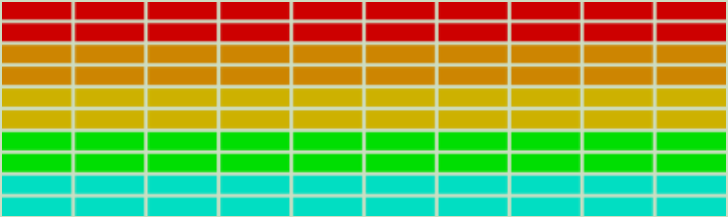
5) Turns, End of Game, and Final Touches

**开发阶段 1: 构建砖块**

Before you start playing the game, you have to set up the various pieces. Thus, it probably makes

sense to implement the **run** method as two method calls: one that sets up the game and one that plays it. An important part of the setup, and our first suggested task, consists of creating the rows of bricks at the top of the game, which look like this:

在你开始玩游戏之前，你必须设置好游戏中各种各样的元素（组件）。因此，将run方法实现为两个方法调用可能是有意义的：一个方法调用设置游戏，另一个方法调用开始游戏。设置的一个重要部分，也是我们的第一个建议任务是在游戏顶部创建一排排的砖块，如下所示：



砖块的数量、尺寸和间距通过起始文件中的命名常量指定，从窗口顶部到第一行砖块的距离也是如此。您需要计算的唯一值是第一列的X坐标，计算好X坐标后，将砖块按照设置好的大小一块一块画出来，剩余的空间在左右两侧均分。

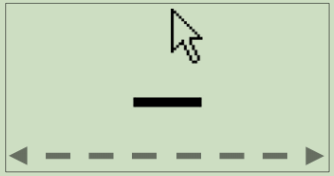
每两行颜色相同，颜色如同所示：**Color.RED, ORANGE, YELLOW, GREEN, CYAN**.

用到的常量: NBRICK\_COLUMNS, NBRICK\_ROWS, BRICK\_SEP, BRICK\_WIDTH,

BRICK\_HEIGHT, BRICK\_Y\_OFFSET

**开发阶段 2: 构建木板**

我们下一个任务是创建一个木板。这个比较容易，我们通过GRect类来构建一个长方形来表示木板。设置它的大小为：PADDLE\_WIDTH, PADDLE\_HEIGHT。平板底部距离窗口底部的距离为PADDLE\_Y\_OFFSET.

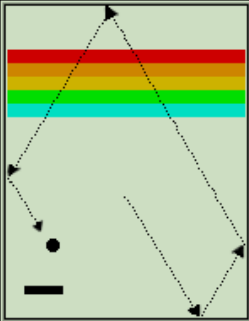


当鼠标在屏幕上移动时，木板水平地跟随鼠标移动；具体来说，必须使拨片的水平中心跟随鼠标。鼠标跟踪使用教科书第10章和7/10讲座中讨论的事件。您只需注意鼠标的X坐标，因为木板的Y坐标是固定的。另外，不要让木板的任何部分离开屏幕边缘。检查鼠标的X坐标是否超出屏幕边界，并确保在窗口中可以看到整个挡板。

*用到的常量:* PADDLE\_WIDTH, PADDLE\_HEIGHT, PADDLE\_Y\_OFFSET

**开发阶段 3: 球 + 弹跳**

现在让我们构建一个球，并让它在窗口中来回弹跳（先忽略球与砖块、木板的碰撞，或者从底部掉下来）。



使用GOval 创建一个球对象，并将其显示在窗口中央。当你这样创建球时，请记住，球的坐标并不是球中心点坐标，而是指定球的左上角。程序需要跟踪球的速度，它由两个实例变量表示，其声明如下：

**private double vx, vy;**

把他们生命为实例变量，是因为它们将在整个程序中使用，并且是有用的“游戏状态”。速度分量表示在每个时间步发生的位置变化。最初，球应该向下，你可能会尝试一个速度为+3的VY（注意，当你在屏幕上向下移动时，Java中的Y值会增加）。如果每个球都走同一个路线，游戏会很无聊，所以你应该随机生成VX的值。

根据我们本周学习的关于生成随机数的方法，你可以创建一个随机数生成器：

1. 声明一个实例变量 rgen, 它将帮助我们生成一个随机数:

**private RandomGenerator rgen = RandomGenerator.getInstance();**

2. 初始化变量 **vx** :

**vx = rgen.nextDouble(1.0, 3.0);**

**if (rgen.nextBoolean(0.5)) vx = -vx;**

这段代码会生成一个范围在1.0到3.0之间的double类型的值，并把该值赋给vx。然后再让它有50%的概率变为负值。

一旦你做到了这一点，你的下一个挑战就是让球在全窗口范围内弹跳，完全忽略木板和砖块。为此，您需要检查球的坐标是否超出了边界，同时考虑到球的大小非零。因此，要查看球是否从右墙反弹，需要查看球右边缘的坐标是否大于窗口的宽度；其他三个方向的处理方式类似。计算反弹后发生的情况非常简单。如果球从墙顶或墙底反弹，你所要做的就是反转vy的符号。对应地，从侧壁反弹，只需反转VX的符号。

开发中使用到的常量: BALL\_RADIUS, VELOCITY\_X\_MIN, VELOCITY\_X\_MAX, VELOCITY\_Y, DELAY

**开发阶段 4: 处理碰撞**

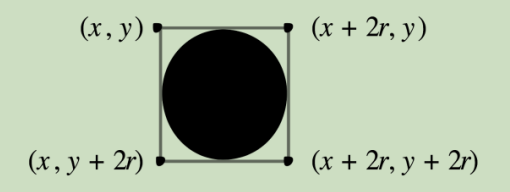
有趣的部分来了。为了使Breakout成为一个真正的游戏，你必须能够知道球是否与窗口中的另一个物体碰撞。就像科学家经常做的那样，先做一个简单的假设，然后再对这个假设扩展，这很有帮助。假设球是一个单点而不是一个圆。在这种情况下，你怎么知道它是否与另一个物体相撞？当你调用方法：

**getElementAt(x, y)**

时，如果x,y 刚好是球的坐标，会怎样? 如果坐标 (x, y) 刚好在一个物体的边界上，此方法就会返回与球碰撞的物体。如果在坐标点 (x, y) 上没有物体，方法就会返回一个空对象null。

到目前为止，一切都很好。但不幸的是，球不是单点的。它占据了物理区域，因此可能会与屏幕上的某个物体发生碰撞，即使它的中心没有。实际上是在真正的电脑游戏中所做的简化假设的典型特征，就是在球的外侧检查一些精心选择的点，看看这些点是否与任何东西发生了碰撞。一旦你在其中一个点上发现什么东西，你就可以宣布球与那个物体相撞了。

在您的实现代码中，最简单的事情就是检查球所在正方形上的四个角点。请记住，GOval是根据其边界矩形定义的，因此，如果球的左上角位于点（x，y），则其他角将位于图中所示的位置：



这些点的优势是在球的外面，这意味着getElementAt()不会返回球本身，但仍然足够近，使它看起来发生了碰撞。因此，对于这四点中的每一点，您需要：

1. 通过调用getElementAt() 判断当前点上是否有一个点上有物体；
2. 如果返回的值不为null, 说明球与物体发生了碰撞；
3. 如果返回值为null，检测下一个点，如果4个点都返回null，说明没有发生碰撞。你可以单独写个方法来获取碰撞对象：

**private GObject getCollidingObject()**

From here, the only remaining thing you need to do is decide what to do when a collision occurs.

There are only two possibilities. First, the object you get back might be the paddle, which you can

test by checking

从现在开始，你唯一需要做的就是决定当碰撞发生时该怎么做。其实，碰撞发生时，只有两种可能，一种可能，返回的对象是木板，另一种可能是砖块。所以就可以这样判断：

**if (collider == paddle) {**

**.......**

**} else {**

**......**

**}**

如果是木板，你需要弹起球，使它开始向上运动。如果不是木板的话，那就只可能是一块砖，因为那是窗口中除了木板上唯一的物体了。此时，你除了需要在垂直方向上引起反弹，还需要把砖块拿走。通过remove方法实现移除砖块。

**开发阶段 5: 重新开始游戏的控制, 游戏结束控制, and Final Touches**

恭喜！我们已经到了开发的最后一个阶段了! 不过，还有一些重要的细节需要你实现：

* 你需要给玩家3次重新开始游戏的机会。每次球碰到窗口底部，玩家挑战失败一次。每次玩家挑战失败后，球回到窗口中心点，等待下次开始，知道3次机会用完。 你可以通过调用球的**setLocation(x, y)** 方法将球移动的窗口中央。记住要给球一个随机的水平方向速度vx。
* 建议在每次挑战之前，通过调用waitForClick方法等待用户单击鼠标开始游戏。
* 当球撞到墙底时，你必须要小心。在你的设计原型中，球会像碰到其它物体一样弹回，这可能这使得比赛很难输掉。一旦遇到这种情况，您必须修改您的循环结构，以便游戏正常运行。
* 你得检查一下另外一个终止条件，那就是最后一块砖。怎么判断最后一个砖块被清除了？有很多方式可以判断，但最简单的方法之一是让你的程序跟踪剩余砖块的数量。每次你打一个，从那个计数器中减去一个。当计数为零时，游戏胜利，并结束。就任务的要求而言，你可以在那一点上停下来，但最好给玩家一些反馈，至少表明游戏是赢还是输。
* 你还需对控制程序速度的设置进行实验。你应该测试在更新球的位置的循环中暂停多长时间合适？是否需要更改速度值以获得更好的游戏体验？
* 你必须测试一下你的程序，看看它是否有效。玩一段时间，确保尽可能多的部分都能正常工作。要测试的一个特殊情况是：在球要通过木板之前，快速移动木板，使其从侧面滑过球。一切还都正常吗？你的球是不是粘在木板上了？为什么会发生这样的错误？你将如何修复这个问题？