作业6
计算机程序设计简介：作业6

截止日期为2018年12月3日晚7点

目标

BallDropper程序将为您提供使用字段和构造函数定义对象类的经验。这是一个非常简单的物理模拟，是许多涉及某种模拟世界的计算机游戏所需的编程类型的第一次介绍。

CircuitDrawer程序将为您提供使用简单事件驱动输入（鼠标和按钮）构建程序的体验。这是一个允许用户构建图像或图表的大量绘图应用程序的简单示例。 CircuitDrawer用于绘制简单的电子电路图，但相同的原理几乎适用于任何类型的图表。注意：在赋值10中，我们将升级CircuitDrawer以使用户能够更轻松地编辑图表。

资源和链接

下载zip文件并解压缩。
提交你的答案。
标记和反馈

摘要

球滴管：
编写一个程序来模拟掉球。完成定义DroppingBall对象的DroppingBall类。
CircuitDrawer计划：
完成CircuitDrawer程序，允许用户使用鼠标在图形窗格上绘制简单的Circuit Diagrams。
反射。
写下你对这项任务的反思。

上交

您应该在截止日期之前提交DroppingBall.java，BallDropper.java，CircuitDrawer.java和Reflection.txt的版本。

制备

下载任务6的zip文件并将其解压缩到主文件夹中的Assig6文件夹。它应包含要完成的Java程序的模板以及图像文件。仔细阅读整个作业，看看你需要做什么。
球滴管

BallDropper程序是一个简单的球形动画，从窗口的左侧开始，然后向右下方移动。

该程序的结构非常类似于CartoonStrip程序和赋值2中的“新对象”练习：有一个类（DroppingBall）代表单个球，另一个类（BallDropper）创建两个DroppingBall对象，并且有一个循环让他们迈出了一步，直到他们到了地面。对于CartoonStrip程序，单个对象的类（CartoonCharacter）是为您编写的，您必须编写控制对象的类。对于这个程序，它是相反的：“控制”类是为你编写的，但你必须为对象编写DroppingBall类。

阅读BallDropper程序，了解它如何制作两个新的DroppingBall对象，然后调用方法。每次围绕主循环（即每个“时间滴答”），程序将
将两个球移动一步。
清除图形窗格
重新划分地面和球在新的位置
检查一个球是否已经击中地面，在这种情况下，它会用一个新球替换球。
makeNewBall方法在窗口的左边缘创建一个新球，具有随机高度，随机速度和随机颜色。

DroppingBall类表示向右下方的球。这是您必须完成的课程。丢球的状态包括
它的水平位置，
它在地板上方的高度（从DroppingBall的地板到底部），
它的水平速度（每一步向右移动多远），
它的垂直速度（每步中向下（或向上）移动的距离），和
它的颜色。

每个新DroppingBall都有不同的起始位置和不同的水平速度。因此，DroppingBall的构造函数必须具有初始高度，速度和颜色的几个参数。

DroppingBall至少应该有三种方法：
draw（）方法应该将球拉到当前位置（存储在字段中）。球是直径为SIZE的圆。
step（）方法应该使球向右移动一步。
getHeight（）方法应该返回球在地板上方的高度。
核心
在BallDropper类中，通过添加参数来完成makeNewBall方法以创建新的DroppingBall。
在DroppingBall类中，完成字段的定义，构造函数，绘制方法，getHeight方法以及仅将球向下和向右移动的步骤方法的简单版本。对于核心，它可以以恒定速度（即，以直线）移动。应将垂直速度初始化为随机值。

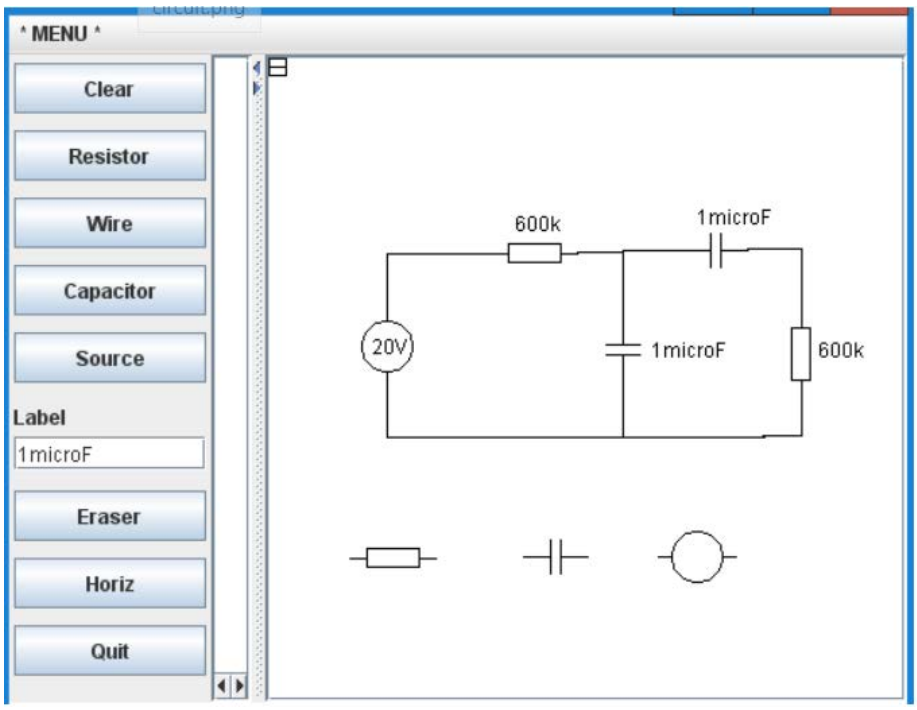
完成
改进步进方法，使球落下时更快更快 - 就像重力一样。为了实现重力，每步的垂直速度必须增加固定量（例如0.1）。初始垂直速度可以设置为0。

挑战

扩展程序以允许球反弹到屏幕的末尾。球在抛物线弹跳中移动应该看起来很逼真。一旦球移出右侧的屏幕，就会创建一个新球。

请注意，这些更改需要更改BallDropper类。
CircuitDrawer计划

对于这个问题，您必须实现一个名为CircuitDrawer的简单应用程序，它允许用户在屏幕上绘制简单的电路图。 这是该程序的屏幕截图，显示了一个简单的电路图
一个“源”（左侧标有20V的圆圈）
两个电阻（标记为600k的矩形）
两个电容器（标记为1microF）
连接组件的电线。
电阻器，电容器和底部没有电线的电源。



CircuitDrawer允许用户选择不同的“工具”来绘制不同的组件（电阻器，电容器和电源），电线和标签。用户可以使用鼠标向图表添加组件，电线或标签。组件可以是垂直的（连接线位于顶部和底部）或水平的（连接线位于侧面）。有一个按钮让用户在水平和垂直模式之间切换。用户可以使用鼠标绘制导线以连接图上的两个点。用户还可以选择橡皮擦工具，然后用鼠标擦除图表中的位。

CircuitDrawer类有
用于存储当前工具名称的字段
存储模式的字段（垂直或水平）
用于存储当前标签的字段
用于存储鼠标按下位置的字段
一个构造函数，它将调用
用于设置GUI的setupGUI方法
响应按钮的方法，
一种响应鼠标的方法，它将调用
绘制组件的方法

CircuitDrawer应该有以下按钮：
清除：清除图形窗格。

电阻：用户选择电阻后，用户可以使用鼠标在图表上的某个位置放置一个电阻（每端有一根短线的小矩形）。电阻将为水平或垂直，具体取决于程序是处于水平模式还是垂直模式。

电容：用户选择电容后，用户可以使用鼠标在图表上的某个位置放置一个电容器（两条平行线，每侧有一根短线）。根据模式，电容器可以是水平或垂直的。

来源：用户选择Source后，用户可以使用鼠标在图表上的某个点放置一个源（每侧有一根短线的圆）。根据模式，导线可以是水平的或垂直的。

Wire：用户选择Wire后，用户可以通过将鼠标从一个点拖动到另一个点来绘制图表上的电线。理想情况下，导线总是由水平线和垂直线组成。如果导线从（x，y）变为（u，v），则水平线从（x，y）变为（u，y），垂直线从（u，y）变为（u， v）。
虽然理想的程序会在用户拖动鼠标时显示正在绘制的导线，但CircuitDrawer更简单，并且只有在用户释放鼠标后才会显示导线。

标签：用户在“标签”框中输入一些文本后，用户可以使用鼠标将标签放在图表上的某个位置。

橡皮擦：用户选择橡皮擦后，用户可以通过在屏幕上的某个位置释放鼠标按钮来擦除小的圆形区域。

Horiz / Vert：改变水平分量和垂直分量之间的模式。单击该按钮应使模式与其当前值相反。

请注意，选择时，没有任何按钮（Clear除外）实际上会更改图形窗格上的任何内容。所有实际绘图都是为了响应在图形窗格上拖动和释放鼠标而发生的。按钮的作用是更改程序的内部状态，以便下一个鼠标操作将绘制不同的形状。

核心

实现CircuitDrawer以便它
有适当的字段。
正确设置用户界面
（setupGUI方法）
正确选择电阻，电容，源，线和擦除工具。
（doSetResistor等，方法）
如果当前工具是电阻器，则在释放鼠标的位置绘制一个水平电阻。
（doMouse和drawResistor方法）。
如果当前工具是电容器，则在释放鼠标的位置绘制一个水平电容。
（doMouse和drawCapacitor方法）
如果当前工具是Source，则在释放鼠标的中心每一侧绘制一条带有短导线的圆。
（doMouse和drawSource方法）
如果选择了“导线”工具，则从按下的点到释放点绘制导线。
对于核心，线可以是两点之间的直线。
（doMouse和drawWire方法）
如果选择了橡皮擦工具，则擦除释放鼠标的圆形区域。
（doMouse和doErase方法）

提示1：要绘制电阻器或电容器，您可以使用提供的图像，也可以绘制它们：
电阻：绘制一条线，然后擦除中间的矩形区域，然后绘制一个矩形的轮廓。
电容：绘制一条线，然后绘制一个矩形，然后在中间擦除一个较窄但　　较长的矩形以形成平行线。

提示2：最常见的困难是确切地确定一切应该发生的时间和地点。请特别注意，当您单击按钮（“清除”按钮除外）时，程序不会绘制任何内容 - 它只是记住选择了哪个按钮。只有在您释放鼠标时才会实际绘制电线或组件。它绘制的组件取决于它从早期的按钮按下时所记得的内容。另请注意，按下鼠标时不会绘制任何内容;只有当鼠标被释放时。

完成

将以下功能添加到您的程序：
使电线更好地绘制：电线应由水平线和垂直线组成。但如果线很短，它只会画一个小点。
使标签工具正常工作，以便用户可以在图表上放置文本标签。
使“水平/垂直”按钮工作，以便更改模式。
使drawResistor，drawCapacitor和drawSource方法绘制水平或垂直分量，具体取决于模式。

挑战
有很多方法可以使这个程序更好用:(注意，有些方法需要UI来响应MouseMotion。)
使Horiz / Vert按钮以某种方式显示当前模式（通过修改按钮或在图表的角上绘制图标）。提示：查看addButton方法返回的内容。
使橡皮擦工具擦除拖动鼠标的所有区域，而不仅仅是释放鼠标的位置。提示：您将需要UI.setMouseMotionListener方法。
使线工具“橡皮筋” - 当用户拖动鼠标时，程序应显示被拖出的电线。但是，它不应该删除拖动它的图表的位。提示：使用invertLine方法。
使工具按钮显示哪个工具处于活动状态（通过修改按钮或在图表的角上绘制图标）。

反射
回答Reflection.txt文件中的以下问题，并确保提交。

你的DroppingBall是否直线下降，重力下降，还是反弹？
你有没有弄清楚如何让球以更陡的角度下降以考虑重力？你是怎么做到的？
如果你让你的DroppingBall弹跳，你怎么知道什么时候你需要制造一个新球？
为什么我们需要DroppingBall类？这些代码都不能成为BallDropper类的一部分吗？将它分开有什么好处。
你为什么需要CircuitDrawer程序中的字段？为什么不能在方法中使用局部变量？

Assignment 6

Introduction to Computer Program Design: Assignment 6

- Due 03 Dec 2018 7 pm

Goals

The **`BallDropper`** program will give you experience in defining classes for objects with fields and constructors. It is a very simple physics simulation, a first introduction to the sort of programming required in many computer games that involve some kind of simulated world.

The **`CircuitDrawer`** program will give you experience building programs with simple event-driven input (mouse and buttons). It is a simple example of the huge number of drawing applications that allow users to construct images or diagrams. **`CircuitDrawer`** is for drawing simple electronic circuit diagrams, but the same principles would apply to almost any kind of diagram. Note: in assignment 10, we will upgrade **`CircuitDrawer`** to enable to user to edit the diagram much more easily.

Resources and links

- Download zip file and unzip it.
- Submit your answers.
- Marks and Feedback

Summary

- Ball Dropper:
  ➡ Write a program to simulate dropping balls. Complete the `DroppingBall` class that defines DroppingBall objects.
- CircuitDrawer Program:
  ➡ Complete the `CircuitDrawer` program that allows the user to draw simple Circuit Diagrams on the graphics pane with the mouse.
- Reflection.
  ➡ Write up your reflections on this assignment.

To Hand in

You should submit your versions of `DroppingBall.java`, `BallDropper.java`, `CircuitDrawer.java` and `Reflection.txt` by the due date.

Preparation

Download the zip file for assignment 6 and extract it to the `Assig6` folder in your home folder. It should contain templates for the Java programs you are to complete, along with image files. Read through the whole assignment to see what you need to do.

Ball Dropper

The `BallDropper` program is a simple animation of balls that start on the left side of a window, and drop across to the right.

The structure of the program is very similar to the `CartoonStrip` program and the "new objects" exercises in assignment 2: there is one class (`DroppingBall`) that represents individual balls, and another class (`BallDropper`) that creates two DroppingBall objects, and has a loop that makes them take a step until they have got to the ground. For the `CartoonStrip` program, the class for the individual objects (`CartoonCharacter`) was written for you, and you had to write the class that controlled the objects. For this program, it is the opposite: the "controlling" class is written for you, but you have to write the `DroppingBall` class for the objects.

Read through the `BallDropper` program to see how it makes the two new DroppingBall objects, and calls methods on then. Each time round the main loop (ie, each "time tick"), the program will

- move both balls by one step.
- clear the graphics pane
- redraw the ground and the balls in their new position
- check whether a ball has hit the ground, in which case it replaces the ball by a new one.

The `makeNewBall` method creates a new ball at the left edge of the window with a random height, a random speed and a random colour.

The `DroppingBall` class represents a ball that is falling to the right. This is the class that you must complete. The state of a dropping ball consists of

- its horizontal position,
- its height above the floor (from the floor to the bottom of the DroppingBall),
- its horizontal speed (how far it will move to the right in each step),
- its vertical speed (how far it will move down (or up) in each step), and
- its colour.

Each new `DroppingBall` will have a different starting position and a different horizontal speed. Therefore, the constructor for `DroppingBall` must have several parameters for the initial height, speed and colour.

A `DroppingBall` should have at least three methods:

- The `draw()` method should draw the ball at its current position (as stored in the fields). The ball is a circle of diameter `SIZE`.
- The `step()` method should make the ball move one step to the right.
- The `getHeight()` method should return the height of the ball above the floor.

Core

- In the `BallDropper` class, complete the `makeNewBall` method by adding the arguments to create a new DroppingBall.

- In the `DroppingBall` class, complete the definitions of the fields, the constructor, the `draw` method, the `getHeight` method, and a simple version of the `step` method which just moves the ball down and to the right. For the core, it can move at a constant speed (ie, in a straight line). The vertical speed should be initialised to a random value.

Completion

- Improve the `step` method to make the ball drop faster and faster as it falls - like with gravity. To implement gravity, the vertical speed must be increased by a fixed amount (eg 0.1) each step. The initial vertical speed can be set to 0.

Challenge

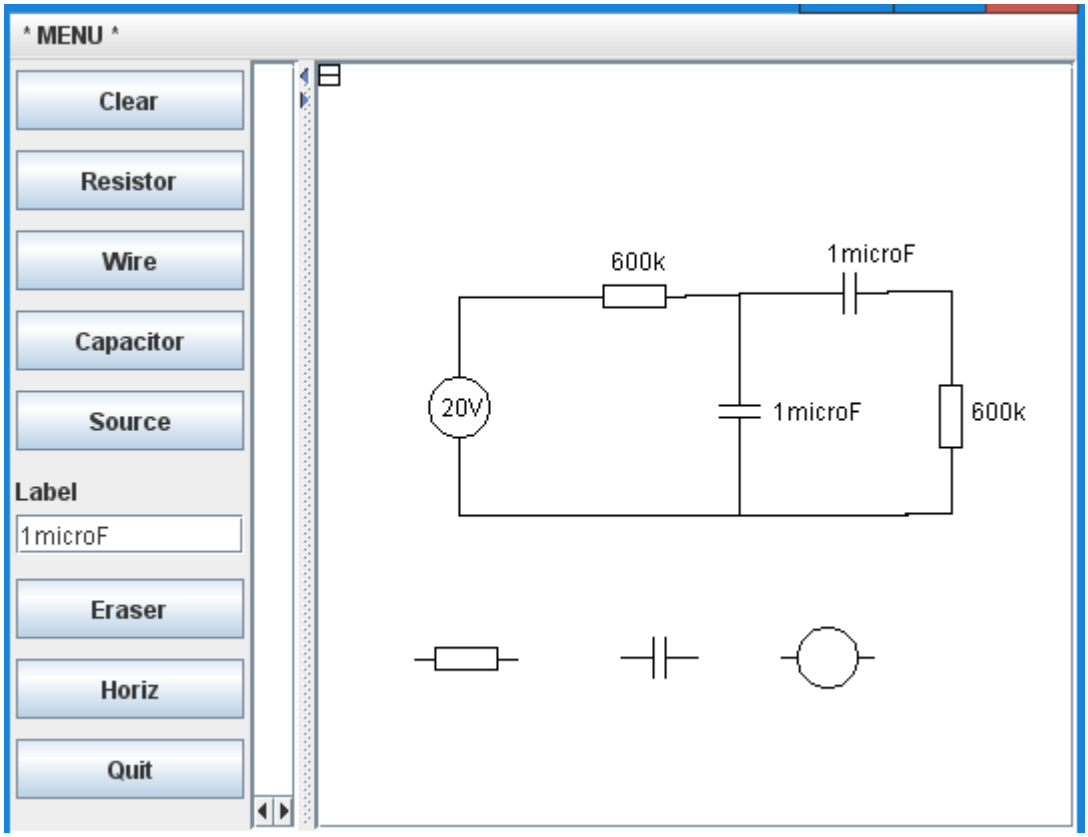
Extend your program to allow the ball to bounce to the end of the screen. It should look realistic with the ball moving in parabolic bounces. Once the ball moves out of the screen on the right, a new ball is created.

Note, these changes require changing the `BallDropper` class.

CircuitDrawer Program

For this question, you must implement a simple application called `CircuitDrawer` that lets a user draw simple circuit diagrams on the screen. Here is screenshot of the program showing a simple circuit diagram with

- one "source" (the circle labeled 20V on the left)
- two resistors (the rectangles labeled 600k)
- two capacitors (labeled 1microF)
- wires connecting the components.
- a resistor, a capacitor, and a source at the bottom that have no wires yet.



`CircuitDrawer` lets the user select different "tools" to draw different components (resistors, capacitors, and sources), wires, and labels. The user can use the mouse to add components, wires, or labels to the diagram. The components can be vertical (the connecting wires are at the top and bottom) or horizontal (the connecting wires are at the sides). There is a button to let the user switch between horizontal and vertical mode. The user can use the mouse to draw wires to connect two points on the diagram. The user can also select the eraser tool and then erase bits of the diagram with the mouse.

The `CircuitDrawer` class has

- a field to store the name of the current tool
- a field to store the mode (vertical or horizontal)
- a field to store the current label
- fields to store the position the mouse was pressed
- a constructor, which will call
- a setupGUI method for setting up the GUI
- methods to respond to the buttons,
- a method to respond to the mouse, which will call
- methods for drawing the components

`CircuitDrawer` should have the following buttons:

- **Clear**: Clears the graphics pane.

- **Resistor**: After the user has selected **Resistor**, the user can place a resistor (a small rectangle with a short wire on each end) at a point on the diagram using the mouse. The resistor will either be horizontal or vertical,

depending on whether the program is in horizontal mode or vertical mode.

- **Capacitor**: After the user has selected **Capacitor**, the user can place a capacitor (two parallel lines with a short wire on each side) at a point on the diagram using the mouse. The capacitor will either be horizontal or vertical, depending on the mode.

- **Source**: After the user has selected **Source**, the user can place a source (a circle with a short wire on each side) at a point on the diagram using the mouse. The wires will either be horizontal or vertical, depending on the mode.

- **Wire**: After the user has selected **Wire**, the user can draw wires on the diagram by dragging the mouse from one point to another. Ideally, a wire is always made of a horizontal line and a vertical line. If the wire goes from (x,y) to (u,v), then the horizontal line goes from (x, y) to (u, y), and the vertical line goes from (u, y) to (u, v). Although an ideal program would show the wire being drawn while the user drags the mouse, `CircuitDrawer` is simpler, and only shows the wire once the user has released the mouse.

- **Label**: After the user enters some text in the **Label** box, the user can place the label at a point on the diagram using the mouse.

- **Eraser**: After the user has selected **Eraser**, the user can erase a small circular region by releasing the mouse button at some point on the screen.

- **Horiz/Vert**: Changes the mode between horizontal components and vertical components. Clicking the button should make the mode be the opposite of its current value.

Note that none of the buttons (except **Clear**) actually change anything on the graphics pane when they are selected. All actual drawing happens in response to dragging and releasing the mouse on the graphics pane. The effect of a button is to change the internal state of the program, so that the next mouse action will draw a different shape.

Core

Implement `CircuitDrawer` so that it will

- Have the appropriate fields.
- Set up the user interface correctly
  (The `setupGUI` method)
- Select the **Resistor**, **Capacitor**, **Source**, **Wire** and **Erase** tools correctly.
  (The `doSetResistor` etc, methods)
- Draw a horizontal resistor centered where the mouse is released if the current tool is **Resistor**.
  (The `doMouse` and `drawResistor` methods).
- Draw a horizontal capacitor centered where the mouse is released if the current tool is **Capacitor**.
  (The `doMouse` and `drawCapacitor` methods)

- Draw a circle with short wires on each side centered where the mouse is released if the current tool is **Source**. (The `doMouse` and `drawSource` methods)
- Draw a wire from the pressed point to the released point if the **Wire** tool is selected.
  For the core, the wire can be a straight line between the two points.
  (The `doMouse` and `drawWire` methods)
- Erase a circular region where the mouse is released point if the **Eraser** tool is selected.
  (The `doMouse` and `doErase` methods)

Hint 1: To draw the resistors or capacitors, you can either use the images provided, or you can draw them:

- Resistor: draw a line, then erase a rectangle region in the middle, then draw the outline of a rectangle.
- Capacitor: draw a line, then draw a rectangle, then erase a narrower but longer rectangle in the middle to make the parallel lines.

Hint 2: The most common difficulty is working out exactly when and where everything should happen. Notice especially that when you click on the buttons (except the **Clear** button), the program *does not draw anything* - all it does is to remember which button was chosen. It is only when you release the mouse that it actually draws a wire or a component. What component it draws depends on what it remembered from an earlier button press. Note also that it doesn't draw anything when the mouse is pressed; only when the mouse is released.

Completion

Add the following features to your program:

- Make the wires draw better: The wire should consist of a horizontal line and a vertical line. But if the line is very short it only draws a small dot.
- Make the **Label** tool work so that the user can place text labels on the diagram.
- Make the **Horizontal/Vertical** button work so that it changes the mode.
- Make the `drawResistor`, `drawCapacitor` and `drawSource` methods draw a horizontal or vertical component, depending on the mode.

Challenge

There are lots of ways in which this program could be made nicer to use: (Note, some will require the UI to respond to MouseMotion.)

- Make the **Horiz/Vert** button show the current mode in some way (either by modifying the button, or drawing an icon in the corner of the diagram). Hint: look at what is returned by the `addButton` method.
- Make the eraser tool erase the region everywhere the mouse is dragged, not just where the mouse is released. Hint: you will need the `UI.setMouseMotionListener` method.
- Make the wire tool "rubber-band" - as the user drags the mouse, the program should show the wire being dragged out. However, it should not erase the bits of the diagram that it is dragged over. Hint: use the `invertLine` method.
- Make the tool buttons show which tool is active, (either by modifying the button, or drawing an icon in the corner

of the diagram).

Reflection

Answer the following questions in the `Reflection.txt` file, and make sure you submit it.

1. Did your DroppingBall go straight, fall nicely by gravity, or did it bounce?
2. Did you work out how to make the ball go down at a steeper angle to take gravity into account? How did you do it?
3. If you made your DroppingBall bounce, how did you tell when you needed to make a new ball?
4. Why do we need a `DroppingBall` class? Couldn't the code all be part of the `BallDropper` class? What is the advantages of having it separate.
5. Why do you need fields in the `CircuitDrawer` program? Why couldn't you use local variables in the methods?

Print version | Backlinks |