Lab 05: Keyboard Events

In prior labs, we've implemented spheres which bounce around the screen and are affected by gravity. However, every program has been "fire and forget" — that is, we run the program and then watch it, without the ability to have any input other than closing the window and terminating the program. In this lab, we'll add the ability to affect what's going on *during runtime* by pressing keys on the keyboard.

First, download `KeyboardSpheres.java`.

This is a slightly modified version of `OOBouncingSpheres.java` from a previous lab.
The key difference is that now the main class `implements KeyListener`. (We've also reimplemented the `Pair` class after it was totally actually stolen by a nefarious hacker.) This lets our program detect what keys (on the keyboard) are being pressed.

The `KeyListener` interface defines the following methods:

```
1 public void keyPressed(KeyEvent e);
2 public void keyReleased(KeyEvent e);
3 public void keyTyped(KeyEvent e);
```

A `KeyEvent` is a class which holds information about an event pertaining to a key on the keyboard. If you're curious, details of the class are found here: https://docs.oracle.com/javase/7/docs/api/java/awt/event/KeyEvent.html
For our purposes, the main thing we care about is a method called `getKeyChar()` which returns the character of the key in the event. See the implementations of the above methods in `KeyboardSpheres.java` to see how it's used.
We'll learn more about interfaces formally soon, but see if you can figure out what they're doing here by examining the code.

Running the Program

Compile and run the program and start pressing keys.

You should see that not much is happening in the main window, but if you look at the terminal things should be printing out.
Currently, they are pretty boring — they simple print out what key was pressed/released/typed.

Do some experimentation by pressing keys to figure out when each gets called.

(Re)familiarize yourself as necessary with the code. In this lab you'll only need to edit:
`keyReleased`, `keyPressed`, `keyTyped`
(and maybe not all of them). You can create your own additional methods, which may be helpful.

To Complete this Lab

You will be editing one or more of the methods: `keyReleased`, `keyPressed`, `keyTyped`, as well as creating whatever additional methods you want to (helper methods may make your task easier and help cut down on repeated code).

Make it so that we can control gravity with the keyboard. Make four keys each set gravity to be one of the four cardinal directions. Specifically:
Pressing `w` should make gravity be toward the ceiling.
Pressing `s` should make gravity be toward the floor.
Pressing `a` should make gravity be toward the left wall.
Pressing `d` should make gravity be toward the right wall.

If you want a challenge:

- Make keys `1234...` change the colors of the spheres.
- Make `e` increase the strength of gravity, no matter what direction it's current in. Similarly, make `q` decrease the strength of gravity. (What happens when you change direction after in/decreasing the strength is up to you.)
- Make it so that gravity changes only when a key is held down (that is, if no keys are pressed, there is no gravity in any direction).
  - Allow two keys to be pressed at the same time, to make gravity pull toward the corners.
- Make `ijkl` control a single sphere directly. Now detect when that sphere overlaps with another and create a game where the goal is to dodge the other spheres (you may want to reduce the total number of spheres for this).
- Java also has a `MouseListener`. Do some googling to figure out how that works, and then make gravity pull toward wherever the user clicks.
- Refactor the program such that gravity is stored in `World`, rather than in each `Sphere` separately.
- Ever played Asteroids (https://en.wikipedia.org/wiki/Asteroids_(video_game))? Implement your own version.

What you should submit

`KeyboardSpheres.java`

Also remember to fill out the survey: https://cosc112s22.page.link/Lab05Survey

Hints:

1. Start by having gravity only change for one sphere.
2. Use debugging print statements (e.g., `System.out.println(spheres[0].acceleration.x)`, or be fancy and override the `toString` method in Sphere), and be sure to keep an eye on the terminal for the ouput.
3. If you find the need to repeat a chunk of code, create a new method with it, and call that method where appropriate.

If you're interested...

`KeyListener` is a somewhat oldskool way of handling input from the keyboard. It can be problematic when you have a more complicated graphical user interface (GUI), or when you smash a bunch of keys at the same time. A more robust framework for handling this within Java's `swing` (what handles graphics) is using what are called key bindings. See http://www.dreamincode.net/forums/topic/245148-java-key-binding-tutorial-and-demo-program/ for more information.