APCS
Kaplan/Tiveron
Falling Sand

In this lab, you'll create a *falling sand* program. The software resembles a paint program, except that the user is painting particles into the world. The software simulates the physical behavior of those particles, which may move (perhaps falling like grains of sand), change, clone, disappear, interact, etc.

Exercise 0: Getting Started

Download FallingSand.zip. Compile and run SandLab.java. (This will run `SandLab`'s `main` method, which constructs a new `SandLab` and calls its `run` method.) You should see a window pop up. On the left side is a black rectangular canvas which will soon be inhabited by particles. On the right side there is one button for each tool you will be able to paint with: *Empty* (for erasing) and *Metal* (for creating metal particles). You can't actually paint now, because you haven't written the code yet.
Look in the SandLab.java file, and you'll see that a `SandLab` remembers two things:

- `grid` - a 2-dimensional array of `int` values that represent the type of particle found at each location
- `display` - the `SandDisplay` used to show the particles on the screen

Do not add any more fields!

Notice that we're using `int` values to represent particle types, with 0 representing *empty*, 1 representing *metal*, and higher values representing the additional particle types you'll be adding. To avoid confusion, **we never want to see these particle type numbers (0, 1, etc.) in our code!** Instead, we've declared variables for each of these types. You'll see these listed near the top of SandLab.java.

```
public static final int EMPTY = 0;
public static final int METAL = 1;
```

This lets us use meaningful variable names instead of confusing type numbers in our code. For example:

```
if (type == METAL)
```

These variables are marked `final` to indicate that they are constants. (Attempts to re-assign to these variables will not compile.) By convention in Java, we use all-caps names for constants. (Traditionally, constants are also declared as `public` and `static`, so that we can access them from outside the file by writing `SandLab.METAL`, for example.)

Exercise 1: Constructor

The `SandLab` constructor already initializes the `display` field to refer to a new `SandLabDisplay` with appropriate dimensions and tool names. Insert code to initialize the `grid` field to refer to a 2-dimensional array of the same dimensions. (You won't be able to test this code yet.)

Exercise 2: `locationClicked`

The `locationClicked` method is called (by the `run` method) whenever the user clicks on some part of the canvas. The selected tool (*empty*, *metal*, etc.) is passed to the method. Store this value in the corresponding position of the grid array. (You won't be able to test this code yet.)

Exercise 3: `updateDisplay`

The `updateDisplay` method is called (by the `run` method) at regular intervals. Its job is to draw each particle (and empty space) found in grid onto the display, using `SandDisplay`'s `setColor` method. Complete this method so that empty locations are shown in one color (probably black) and metal locations are shown in another color (probably gray).

`class java.awt.Color`

```
Color(int red, int green, int blue) // values range from 0 - 255 inclusive
```

`class SandDisplay`

```
void setColor(int row, int col, Color color)
```

Test that you can now paint metal particles and erase them.

Exercise 4: Sand

Modify your program so that you can also paint with *sand* particles (probably in yellow). For now, these particles won't actually move.

Exercise 5: `step`

The `step` method is called (by the `run` method) at regular intervals. This method should choose a *single random valid location*. (Do not use a loop.) If that location contains a sand particle and the location below it is empty, the particle should move down one row. (Metal particles will never move.) This code should only modify the array. Do not set any colors in the display. Test that your sand particles fall now.
Tip: If particles fall too quickly or too slowly, the speed can be adjusted by adjusting the slider in the display or by changing the dimensions passed to the `SandLab` constructor (from `main`).
Note: Because the `step` method picks a single random particle to move (or act in some way) each time it is called, it is possible that some sand particles will move several times before others have the chance to move at all. In practice, the `step` method is called so rapidly that you are unlikely to notice this effect when you run the code.

Exercise 6: Water

Modify your program so that you can also paint with *water* particles, which move in one of three randomly chosen directions: down, left, or right.
In the `step` method, when the randomly chosen location contains a water particle, pick one of three random directions. If the location in that randomly chosen direction is empty, the water particle moves there. (Look for ways to minimize duplicate code in your `step` method.)
Test that the water behaves roughly like a liquid, taking the shape of a container.

Exercise 7: Dropping Sand Into Water

What happens now when you drop sand particles into water? Right now, sand is only allowed to move into empty spaces. Modify your code so that a sand particle can also move into a space containing a water particle (by trading places with the water particle). (Look for ways to minimize duplicate code in your `step` method.)
Test that you can drop sand into water now (without destroying the water).

Stretches:

-Make an "ERASE ALL" button that will clear the picture and reset the canvas
-Add behavior for when metal (or water) is drawn over sand or water so those particles don't just disappear
-Add in other elements of your choosing (fire -> smoke, laser beam)

Get creative!

Hints:

Exercise 3: Create a nested for loop that will visit each element in grid and then set the color to the appropriate color given the element found in that location. Don't forget to call display.setColor(r, j, color); in your for loops.

Exercise 5: Here is a snippet to help you out:

```
//MOVE SAND DOWN SO THAT IT FALLS
  if(r < grid.length - 2 && grid[r][c] == SAND && grid[r+1][c] == EMPTY) {
          //CODE NOT SHOWN
  }
 else if(r == grid.length-1 && grid[r][c] ==  ???? ) {
           //CODE NOT SHOWN
 }
```

Exercise 7: Here is a snippet to help you out:

```
//LET SAND MOVE BELOW WATER (SWITCH PLACES)
 if(r < grid.length - 2 && grid[r][c] == ??? && grid[r+1][c] == ??? ) {
          /CODE NOT SHKWN
}
```