ParticleLab – based on a class project at Stanford University

In this project, you'll create what is called 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, generate, disappear, interact, etc.

Exercise 0: Getting Started

Download ParticleLab.java and LabDisplay.class. Compile and run ParticleLab.java. (This will run `ParticleLab`'s `main` method, which constructs a new `ParticleLab` and calls its `run` method.) You should see a window pop up. On the left side is a black rectangular canvas which you will create 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). Note that you can't actually paint now, because you haven't written the code yet.

Look in the ParticleLab.java file, and you'll see that a `ParticleLab` remembers two things:

- `particleGrid` - a 2-dimensional array of `int` values that represent the type of particle found at each location
- `display` – the variable for `LabDisplay` used to show the particles on the screen

Do not add any more global fields!

You can ignore the fields and code relating to saving and reading a file until later.

Now, 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 ParticleLab.java.

```
public static final int EMPTY    = 0;
public static final int METAL    = 1;
public static final int SAVEFILE = 2;
```

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 `ParticleLab.METAL`, for example.)

Exercise 1: Constructor

The `ParticleLab` constructor already initializes the `display` field to refer to a new `LabDisplay` with appropriate dimensions and tool names. The program should compile and display a black window with each corner pixel a different color.

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 particleGrid 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 particleGrid onto the display, using `LabDisplay`'s `setColor` method. Complete this method so that empty locations are shown in one color (Color.black) and metal locations are shown in another color (Color.gray). (note: *display* is the variable name)

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

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

```
display.setColor(r,c,Color.gray);  //setColor is a method in the LabDisplay program.
                                   // r = row, c = column, Color.gray is a java constant
```

Valid colors include blue, gray, yellow, black, orange, pink, white, red, cyan, and green.
Here is how to define a new color: Color purple = new Color (192, 0, 255);

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.

How the metal tool is coded is a pattern you can follow when adding sand to the program. Sand should be displayed with the Color.yellow.

Exercise 5: Updating with the Step

The `step` method is called (by the `run` method) at regular intervals. This method should choose a *single random valid location* by row and column. (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.

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: Wrapping and Gravity

Write code to allow moving elements to wrap from bottom to top, right to left, left to right, and top to bottom. Test this VERY thoroughly to make sure your code is solid and works in all situations. Every time you add an element, test wrapping again.

You might consider generalizing the code rather than writing a lot of if-statements.

Create a button to toggle gravity. Your code should work when gravity is up or down. Now you might see the benefit of generalizing the code. When gravity is down, the next row is r + 1. When gravity is going up, the next row is r – 1. Set that early and all the code after that should work ok.

DO NOT continue until your code works in ALL directions.

Exercise 7: 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 in an appropriate amount of time. Water should be displayed with the Color.blue.

Exercise 8: Enhancing Sand Particles

Make your sand particles act like sand does in real life. Sand makes piles.

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).

Saving and Reading a File

You can save the map in a file by clicking on the save file button and clicking on the screen. The screen will be saved to the file name in the NEW_FILE_NAME constant. You will have to go rename that to the name in FILE_NAME to have it read in when you start the program next time. Give it a try!

Further Exercises: Add more required tools and behaviors

Now implement other required behaviors:

Oil – Floats on water, seeks own level. Use Color.orange

Generator – Generates below it any moving element that is on top of it.
Use Color.pink. Play with how often the generator creates a particle.

Destructor – Destroys any moving element that touches it. Use Color.cyan
Creates Vapor particles.

Vapor – Created when a moving element hits a destructor. Use Color.white
Should act like vapor. Think about it! How does vapor act in real life?

Reset - Reloads the saved file

Get creative and add your own!

Some ideas:

Instead of a black background, make a color gradient between two colors.
Clear – Clears the entire screen back to empty
Wall Toggle – metal disappears or reappears everywhere
Random – Clears world and adds selected random elements all over

Ice	plants	fish
plankton	nanites	wet sand
blocks	fire	lava
acid	bubbles	dirt
wood	birds	black hole
wind	wind shield	wind generator
gravity well	explosions	left or right conveyors
jumps	transporters	accelerators
sticky wickets	explosions	bacteria