The code is a class Cell that has three fields: x, y, and g. The field x is an int variable that represents the coordinates of the cell.

The field y is also an int variable that represents the coordinates of the cell's boundary.

The field g is a Grain object which stores information about how to calculate curvature for this particular cell.

The code starts by declaring two variables called d and gamma with values 0 and -1 respectively.

These are used later in calculations when calculating curvature for this particular cell.

Next, there is a method called Cell(int x, int y) which takes two parameters representing the coordinates of this particular cell (x and y).

Inside of it, there are some statements like setting up variables with their initial values as well as creating new ones if they don't exist yet or assigning them to existing ones if they do exist already.

The code declares a class named Cell.

It contains three private fields, x, y and g. The first field is an int which represents the width of the cell in pixels.

The second field is an int which represents the height of the cell in pixels.

The third field is a Grain object that will be used to represent all cells that are not empty (i.e., they have data).

The next line creates two private methods: one called getBoundary() and one called setBoundary().

These two methods will be used to determine whether or not this particular Cell has any boundaries at all.

Next, there are four public fields: curvature, prevGrain, gamma and d. Curvature is defined as a

The code is a class that has three variables: x, y, and grain.

The setD() method sets the value of d to an array of doubles.

The getD() method returns the value of d as an array of doubles.

The setX() and setY() methods change the values for x and y respectively.

The getX(), getY(), and getGrain() methods return the values for x, y, and grain respectively.

The code is an example of a method that contains the following: A setter and getter for a single variable.

A loop that iterates through all elements in the array, setting each element to 0.

A method that returns the x-coordinate of this object.

The code starts by setting the current grain to null.

If the current grain is null, then it sets the new grain to gr and increments n\_cells on gr.

If the current grain is not null, then it checks if this.g == gr or this.g != gr (meaning that either one of these conditions are true).

If both of these conditions are true, then there is no need for further action because this means that they are equal grains and thus will have an identical number of cells in their respective grids.

If only one condition is true, meaning that either this.g == gr or this this g !=gr), then a comparison between them must be made to determine which has more cells than the other: if ((this.g !=gr) || (this .

g ==gr)) { // System .

out .

println("yes"); } else { // System .

out .

println("no"); }

The code is a loop that will iterate through the list of grains.

The first grain in the list is set to be the current grain, and it's value is then compared with gr.

If they are not equal, then this means that there was a change in the list of grains.

If this happens, then we know that there has been a change in the list of grains so we check to see if our current grain equals our new grain (gr).

If they do not equal, then we know that there has been a change in the list of grains and we need to update our previous grain (prevGrain) with our new one (gr).

We also increment n\_cells on both gr and g for each iteration through the

The code is trying to find the boundary of a rectangle.

The code starts by creating a new gr object and setting its n\_cells property to 1.

Next, it creates an array called g that will hold all the rectangles in this program.

The first thing the code does is create a new gr object with n\_cells set to 1 (the number of cells in one side).

Then it creates an array called g that will hold all the rectangles in this program.

Next, it sets up some variables for use later on: this.g = gr; // gr.n\_cells++; This line initializes the variable named "gr" as a new instance of class "Gr".

It then increments its n\_cells property by 1 using method setNCell() .

This line also calls method getNCell() which returns how many cells are currently contained within this Gr object's boundaries - i.e., how many cells are inside or outside of its boundaries at any given time during execution of this program's methods and loops/loops nested within other loops/loops nested within other loops...etc..

In order for these methods to work properly, you must have initialized your Gr objects with

The code is a method that returns the gamma of an object.

The code starts by declaring a variable called "this."

Then, it declares a new variable called "g" which has been initialized to 0.

Then, it assigns gr to this.g and sets its value to be equal to 0.

The next line of code in the method is: // gr.n\_cells++; This line of code increments the value of gr's n\_cells property by 1 each time it executes.