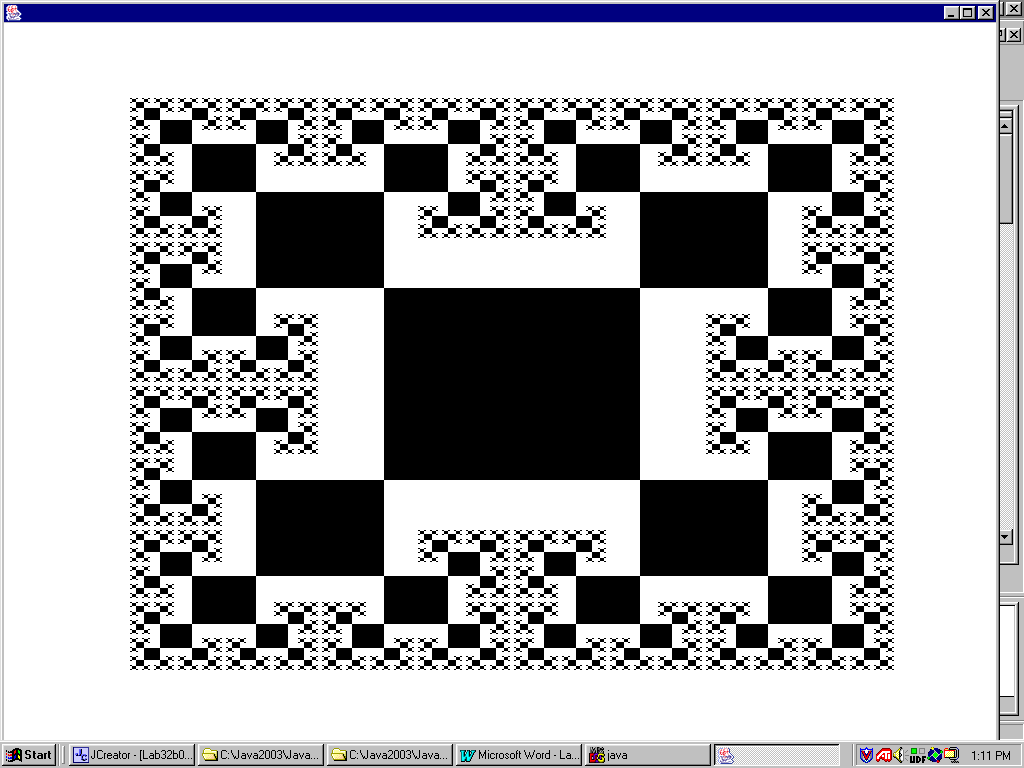
|  |  |
| --- | --- |
| **AP Computer Science** | **GraphicsLab07 Java Assignment** |
| **The "Square Fractal" program** | **100 & 110 Point Versions** |
| **Assignment Purpose:**  The purpose of this program is to display a sophisticated knowledge of recursion by writing a program that uses a variety of recursive features. | |

Write a program that draws a square fractal. Fractals are images that keep repeating their own image in ever-smaller versions. There exists numerous fractals and some fractals rely on computation of advanced mathematics. All fractals have a recursive quality and provide an ideal platform for practicing recursive concepts. This lab assignment is easiest to explain by first looking at the result of the 110-point execution below. You will note that from the four corners of the center rectangle, smaller rectangles are drawn and each rectangle in turn continues to draw three smaller rectangles until the rectangle is the size of one pixel.



The student version of the program, shown below, demonstrates that the **paint** method only makes a single method call. This is a call to method **drawSquare1**, which is the very first square to be drawn, in the center of the monitor. The parameters, **1024** and **768** represent the monitor resolution. Your program needs to be written in such a manner that the graphics display adjusts itself to the initial resolution parameters that are provided. This way you only need to alter one small set of parameters.

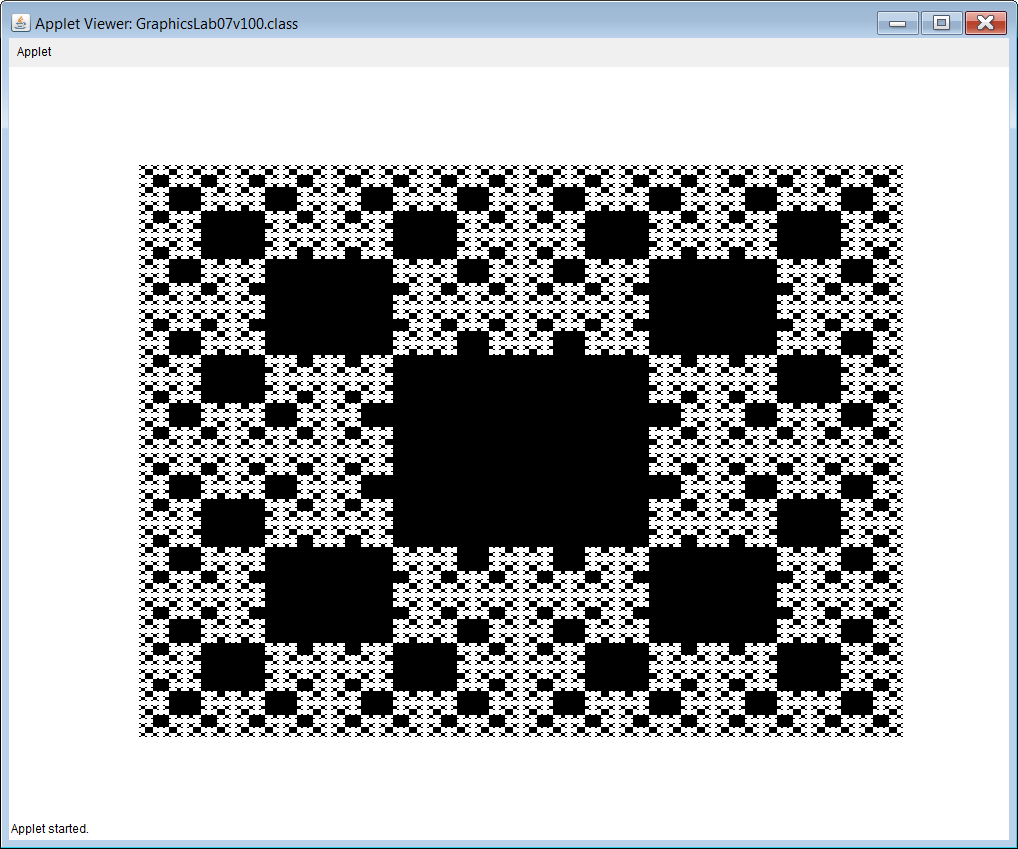
|  |
| --- |
| // GraphicsLab07st.java  // The student, starting version of the GraphicsLab07 assignment.  import java.awt.\*;  import java.applet.\*;  public class GraphicsLab07st extends Applet  {    public void paint (Graphics g)  {  drawSquare1(g,1024,768);  }    public void drawSquare1(Graphics g, int maxX, int maxY)  {  int midX = maxX/2;  int midY = maxY/2;  int startWidth = maxX/4;  int startHeight = maxY/4;  int tlX = midX - (startWidth/2);  int tlY = midY - (startHeight/2);  g.fillRect(tlX,tlY,startWidth,startHeight);  }  public void delay(long n)  {  n \*= 1000;  long startDelay = System.nanoTime();  long endDelay = 0;  while (endDelay - startDelay < n)  endDelay = System.nanoTime();  }    } |

Method **drawSquare1** gets the ball rolling and draws a solid rectangle in the center of the screen. The initial rectangle needs to be 1/4 the size of the monitor. This explains why it is important to know the screen resolution and why this information is passed by parameter. Method **drawSquare1** needs to make four method calls to draw each one of the four rectangles attached to the initial rectangle. Each one of the succeeding rectangles is half the width and height or the preceding rectangle. The **delay** method is provided so that you can observe the recursive sequence as the fractal picture is drawn. You need to experiment with the delay value, which depends on the speed of your computer.

**100-Point Version**

The 80-point version of the fractal assignment draws four rectangles at each corner of every rectangle. This does happen from the initial rectangle, but in the earlier "Square Fractal" display you observed that later rectangle are only attached at three corners. Attaching three rectangles is more complicated. For the 80-point version you can happily and recursive keep on calling rectangle methods until the size of the size of the rectangle reaches one pixel. The result is that later, smaller rectangles will cover over earlier, larger rectangles.

**100-Point Execution**



**110-Point Version**

The 100-point version of the fractal assignment draws only three rectangles at each corner of every rectangle, except for the initial rectangle. This provides you with a unique challenge. How do you prevent drawing over existing rectangles? There are a variety of solutions for this problem, but no hints are provided in this lab assignment.

**110-Point Execution**

