Assignment – 5 & 6

Name : Bijay kumar sah

Roll : 2020CSB062

G-Suite: 2020csb062.bijay@students.iiests.ac.in

Subject :Computer Graphics

Assignment – 5:

Code :

Part 1:

```
// This applet program only works on java jdk 1.8.0_312 or Lower

import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class sc extends Applet implements ActionListener
,MouseWheelListener{
    Button b1 = new Button(" + ");
    Button b2 = new Button(" - ");
    Button gridb = new Button("GRID");
    Button clipp = new Button(" CLIP LINE ");
    int gap = 10;
    int xmin = -20, ymin = -20, xmax = 20 ,ymax = 20;
    boolean gridon = true;
    boolean clip = false;
    // Defining region codes
    int INSIDE = 0; // 0000
    int LEFT = 1; // 0001
    int RIGHT = 2; // 0010
    int BOTTOM = 4; // 0100
    int TOP = 8; // 1000
    //initialization
    public void init() {
        Color buttonColor1 = new Color(0,255,0);
        Color buttonColor2 = new Color(255,0,0);
        Color bgColor = new Color(0,0,0);

        b1.setBackground(buttonColor1);
        b2.setBackground(buttonColor2);
        gridb.setBackground(Color.lightGray);
        clipp.setBackground(Color.CYAN);
        add(gridb);
        add(b1);
        add(b2);
        add(clipp);
        addMouseWheelListener(this);
        b1.addActionListener(this);
        b2.addActionListener(this);
        gridb.addActionListener(this);
        clipp.addActionListener(this);
```

```
        setBackground(bgColor);
    }
    //for zoom using mouse wheel
    public void mouseWheelMoved(MouseWheelEvent e)
    {
        int z = e.getWheelRotation();
        zoom(z);
    }
    //button action listener
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == b1)
            zoom(+10);
        else if(e.getSource()==b2)
            zoom(-10);
        else if(e.getSource()==gridb)
        {
            gridon= !gridon;
            repaint();
        }
        else if(e.getSource()==clipp)
        {
            clip = !clip;
            repaint();
        }
    }

    //function to make a grid with respect to standard cartesian
coordinates
    public void makeGrid(Graphics g, int gap)
    {
        if(gridon==false)
            return;
        if(gap<=0|| gap>getHeight())
            return ;
        int originX = (getX() + getWidth()) / 2;
        int originY = (getY() + getHeight()) / 2;
        g.setColor(Color.DARK_GRAY);
        g.drawLine(originX, originY - getHeight() / 2, originX, originY
+ getHeight() / 2);
        g.drawLine(originX - getWidth() / 2, originY, originX +
getWidth() / 2, originY);
        g.setColor(Color.DARK_GRAY);
        for (int x = gap; x <= getWidth(); x += gap) {
            g.drawLine(originX + x, 0, originX + x, getHeight());
            g.drawLine(originX - x, 0, originX - x, getHeight());
        }
        for (int y = gap; y <= getHeight(); y += gap) {
            g.drawLine(0, originY + y, getWidth(), originY + y);
```

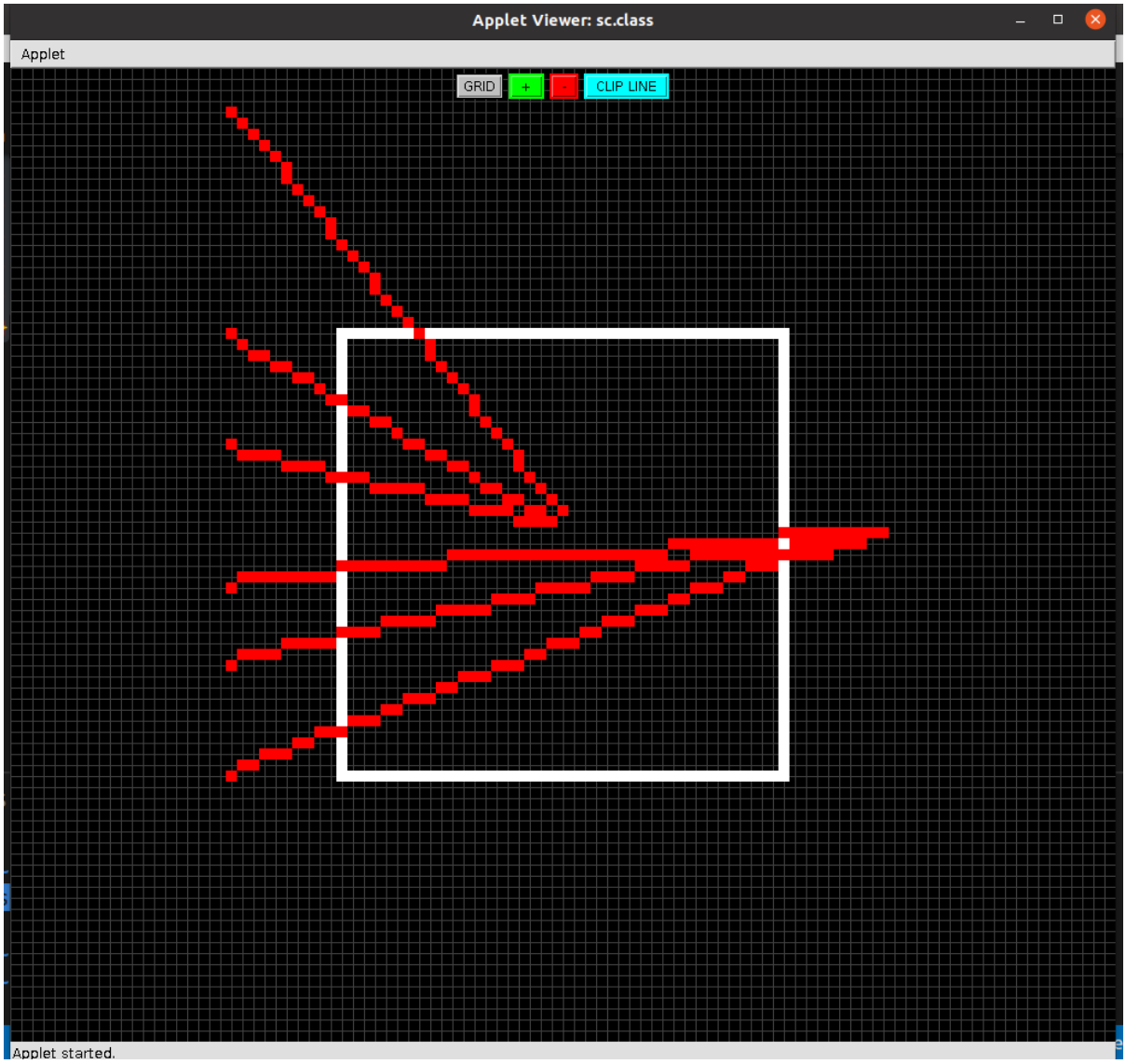
```
            g.drawLine(0, originY - y, getWidth(), originY - y);
        }
    }
    //for zoom feature
    public void zoom(int i)
    {
        if(i>0)
            gap+=gap/10+1;
        else if(i<0)
            gap-=gap/10+1;
        if(gap<0)
            gap = 1000;
        if(gap>1600)
            gap = 2;
        repaint();

    }
    //plots a square at point (Square) x,y in grid
    public void plotpoint(Graphics g, int x,int y ,Color c){
        int originX = (getX() + getWidth()) / 2;
        int originY = (getY() + getHeight()) / 2;
        g.setColor(c);
        g.fillRect(originX+(gap*x)-(gap/2), originY-(gap*y)-(gap/2),gap
,gap);
    }
    //plots a point in x,y (Circle)
    public void plotCircle(Graphics g, int x,int y ,Color c){
        int originX = (getX() + getWidth()) / 2;
        int originY = (getY() + getHeight()) / 2;
        g.setColor(c);
        g.fillOval(originX+(gap*x)-(gap/8), originY-(gap*y)-
(gap/8),gap/4 ,gap/4 );
    }

    public void plotLine(Graphics g,int x1,int y1 ,int x2, int y2,Color
col)
    {
        double x = x1;
        double y = y1;
        int dx = x2-x1;
        int dy = y2-y1;
        int step;
        if(Math.abs(dx) > Math.abs(dy))
            step = Math.abs(dx);
        else
            step = Math.abs(dy);

        for(int i  = 0; i<step; i++)
```

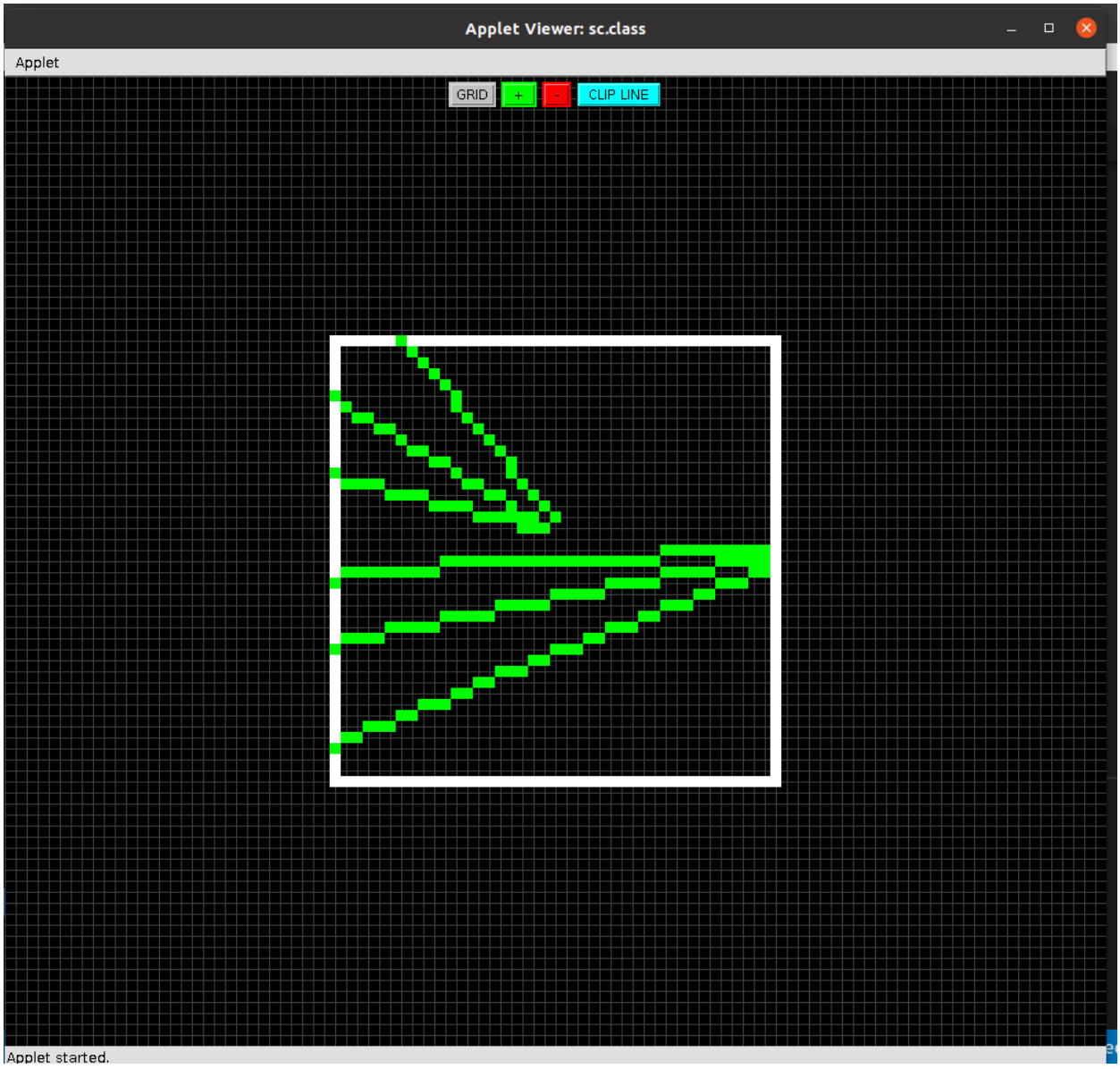
```
        {
            plotpoint(g,(int)x,(int)y,col);
            x  = x+ (double)dx/step;
            y = y + (double)dy/step;
        }
    }

    public void makeWindow(){
        plotLine(getGraphics(), xmin, ymin, xmin, ymax , Color.WHITE);
        plotLine(getGraphics(), xmin, ymin, xmax, ymin , Color.WHITE);
        plotLine(getGraphics(), xmax, ymin, xmax, ymax , Color.WHITE);
        plotLine(getGraphics(), xmin, ymax, xmax, ymax , Color.WHITE);
        plotpoint(getGraphics(),xmax,ymax,Color.WHITE);
    }

    int computeCode(double x, double y){
    // initialized as being inside
        int code = INSIDE;

        if (x < xmin) // to the left of rectangle
            code |= LEFT;
        else if (x > xmax) // to the right of rectangle
            code |= RIGHT;
        if (y < ymin) // below the rectangle
            code |= BOTTOM;
        else if (y > ymax) // above the rectangle
            code |= TOP;
        return code;
    }

    public void cohenSutherlandClip(Graphics g, int x1, int y1, int x2,
int y2) {
        // Compute region codes for P1, P2
        int code1 = computeCode(x1, y1);
        int code2 = computeCode(x2, y2);

        // Initialize line as outside the rectangular window
        boolean accept = false;

        while (true) {
          if ((code1 == 0) && (code2 == 0)) {
            // If both endpoints lie within rectangle
            accept = true;
            break;
          } else if ((code1 & code2) != 0) {
            // If both endpoints are outside rectangle,
            // in same region
            break;
```

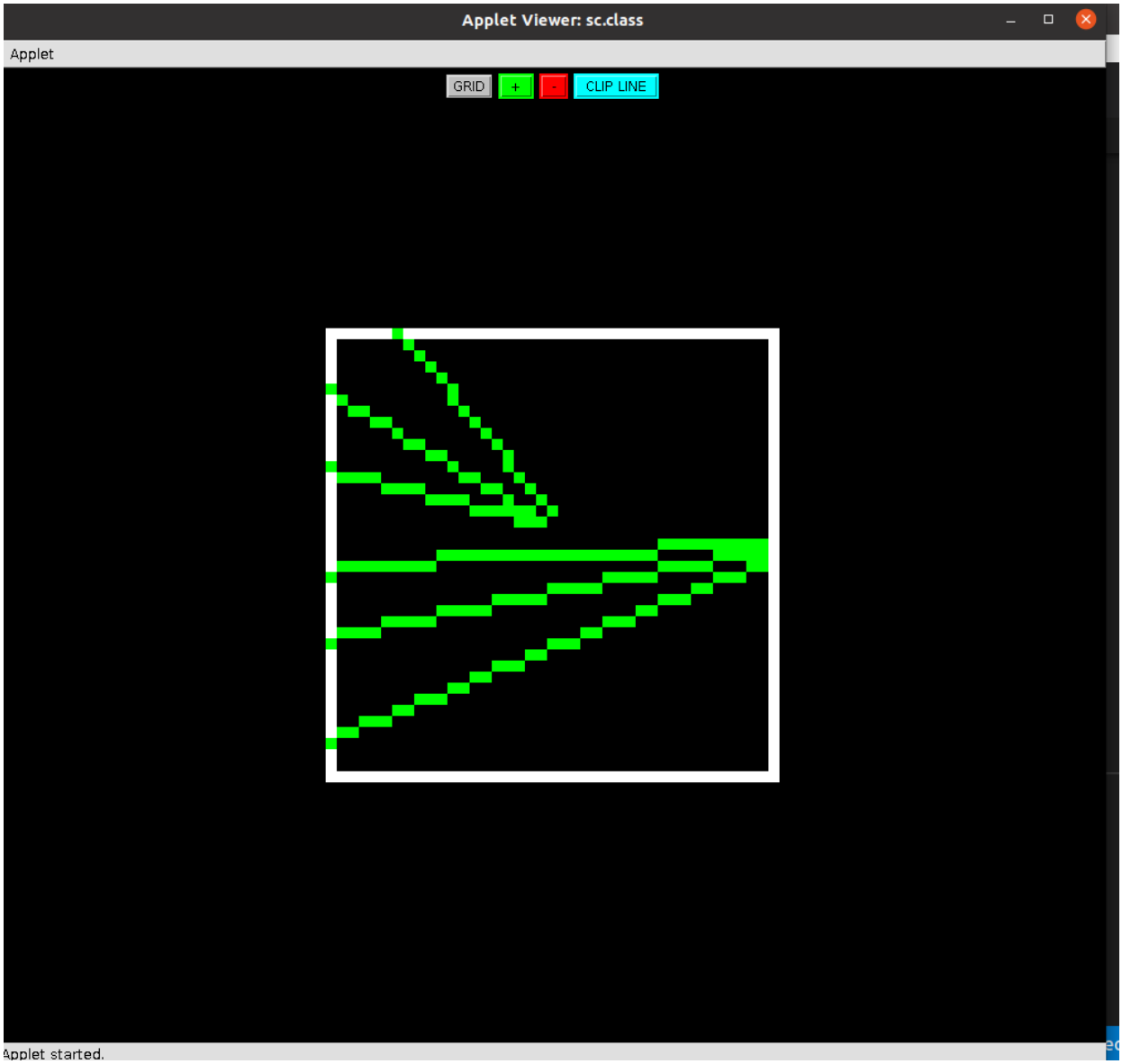
```
        } else {
          // Some segment of line lies within the
          // rectangle
          int code_out;
          int x = 0, y = 0;

          // At least one endpoint is outside the
          // rectangle, pick it.
          if (code1 != 0) code_out = code1; else code_out = code2;

          // Find intersection point;
          // using formulas y = y1 + slope * (x - x1),
          // x = x1 + (1 / slope) * (y - y1)
          if ((code_out & TOP) != 0) {
              // point is above the clip rectangle
              x = x1 + (x2 - x1) * (ymax - y1) / (y2 - y1);
              y = ymax;
            } else if ((code_out & BOTTOM) != 0) {
              // point is below the rectangle
              x = x1 + (x2 - x1) * (ymin - y1) / (y2 - y1);
              y = ymin;
            } else if ((code_out & RIGHT) != 0) {
              // point is to the right of rectangle
              y = y1 + (y2 - y1) * (xmax - x1) / (x2 - x1);
              x = xmax;
            } else if ((code_out & LEFT) != 0) {
              // point is to the left of rectangle
              y = y1 + (y2 - y1) * (xmin - x1) / (x2 - x1);
              x = xmin;
            }

            // Now intersection point x, y is found
            // We replace point outside rectangle
            // by intersection point
            if (code_out == code1) {
              x1 = x;
              y1 = y;
              code1 = computeCode(x1, y1);
            } else {
              x2 = x;
              y2 = y;
              code2 = computeCode(x2, y2);
            }
          }
        }
        if (accept) {
          plotLine(g, x1, y1, x2, y2,Color.green);
```

```
        }
        else
            System.out.println("Line rejected");
      }

    public void paint(Graphics g)
    {
        makeGrid(g,gap);
        plotCircle(g,0,0,Color.yellow);
        makeWindow();

        if(clip){
            cohenSutherlandClip(g, -30, -3, 30, 3);
            cohenSutherlandClip(g, -30, -10, 30, 3);
            cohenSutherlandClip(g, -30, -20, 30, 3);

            cohenSutherlandClip(g, -30, 20, 0, 3);
            cohenSutherlandClip(g, -30, 40, 0, 3);
            cohenSutherlandClip(g, -30, 10, 0, 3);
        }
        else{
            plotLine(g,-30,-3,30,3, Color.RED);
            plotLine(g,-30,-10,30,3, Color.RED);
            plotLine(g,-30,-20,30,3, Color.RED);

            plotLine(g,-30,20,0,3, Color.RED);
            plotLine(g,-30,40,0,3, Color.RED);
            plotLine(g,-30,10,0,3, Color.RED);
        }
    }
}
```

OUTPUT:


Applet Viewer: sc.class
Applet
GRID
+
-
CLIP LINE
Applet started.


Applet Viewer: sc.class
Applet
GRID
+
-
CLIP LINE
Applet started.


Applet Viewer: sc.class
Applet
GRID
+
-
CLIP LINE
Applet started.

Part 2:

Code :

```
// This applet program only works on java jdk 1.8.0_312 or Lower
//Polygon Clipping
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class sc extends Applet implements ActionListener
,MouseWheelListener{
    Button b1 = new Button(" + ");
    Button b2 = new Button(" - ");
    Button gridb = new Button("GRID");
    Button clipp = new Button(" CLIP LINE ");
    int gap = 10;
    int xmin = -20, ymin = -20, xmax = 20 ,ymax = 20;
    boolean gridon = true;
    boolean clip = false;
    // Defining region codes
    int INSIDE = 0; // 0000
    int LEFT = 1; // 0001
    int RIGHT = 2; // 0010
    int BOTTOM = 4; // 0100
    int TOP = 8; // 1000
    //initialization
    public void init() {
        Color buttonColor1 = new Color(0,255,0);
        Color buttonColor2 = new Color(255,0,0);
        Color bgColor = new Color(0,0,0);

        b1.setBackground(buttonColor1);
        b2.setBackground(buttonColor2);
        gridb.setBackground(Color.lightGray);
        clipp.setBackground(Color.CYAN);
        add(gridb);
        add(b1);
        add(b2);
        add(clipp);
        addMouseWheelListener(this);
        b1.addActionListener(this);
        b2.addActionListener(this);
        gridb.addActionListener(this);
        clipp.addActionListener(this);
        setBackground(bgColor);
    }
```

```
    //for zoom using mouse wheel
    public void mouseWheelMoved(MouseWheelEvent e)
    {
        int z = e.getWheelRotation();
        zoom(z);
    }
    //button action listener
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == b1)
            zoom(+10);
        else if(e.getSource()==b2)
            zoom(-10);
        else if(e.getSource()==gridb)
        {
            gridon= !gridon;
            repaint();
        }
        else if(e.getSource()==clipp)
        {
            clip = !clip;
            repaint();
        }
    }

    //function to make a grid with respect to standard cartesian
coordinates
    public void makeGrid(Graphics g, int gap)
    {
        if(gridon==false)
            return;
        if(gap<=0|| gap>getHeight())
            return ;
        int originX = (getX() + getWidth()) / 2;
        int originY = (getY() + getHeight()) / 2;
        g.setColor(Color.DARK_GRAY);
        g.drawLine(originX, originY - getHeight() / 2, originX, originY
+ getHeight() / 2);
        g.drawLine(originX - getWidth() / 2, originY, originX +
getWidth() / 2, originY);
        g.setColor(Color.DARK_GRAY);
        for (int x = gap; x <= getWidth(); x += gap) {
            g.drawLine(originX + x, 0, originX + x, getHeight());
            g.drawLine(originX - x, 0, originX - x, getHeight());
        }
        for (int y = gap; y <= getHeight(); y += gap) {
            g.drawLine(0, originY + y, getWidth(), originY + y);
            g.drawLine(0, originY - y, getWidth(), originY - y);
        }
```

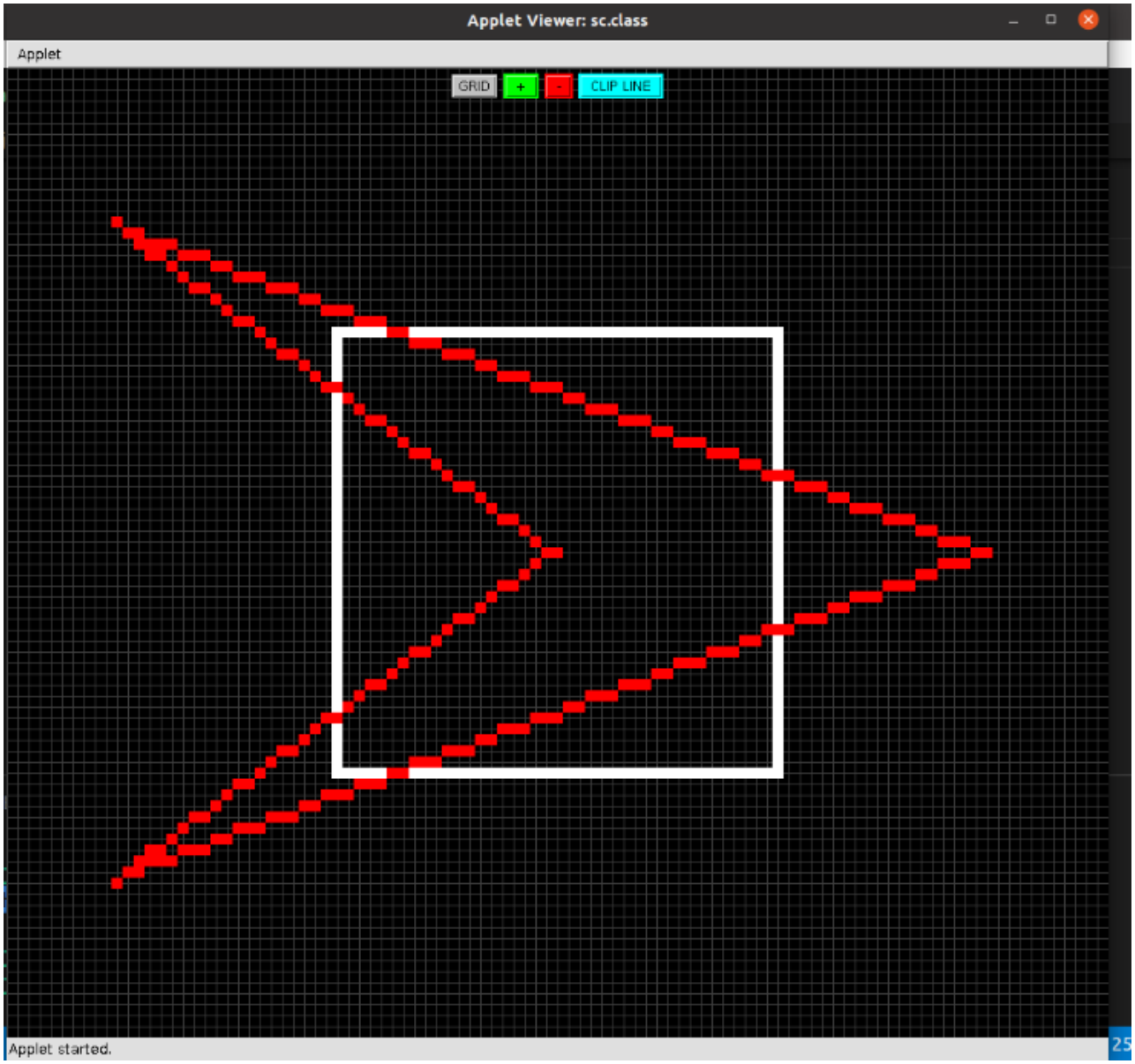
```
    }
    //for zoom feature
    public void zoom(int i)
    {
        if(i>0)
            gap+=gap/10+1;
        else if(i<0)
            gap-=gap/10+1;
        if(gap<0)
            gap = 1000;
        if(gap>1600)
            gap = 2;
        repaint();

    }
    //plots a square at point (Square) x,y in grid
    public void plotpoint(Graphics g, int x,int y ,Color c){
        int originX = (getX() + getWidth()) / 2;
        int originY = (getY() + getHeight()) / 2;
        g.setColor(c);
        g.fillRect(originX+(gap*x)-(gap/2), originY-(gap*y)-(gap/2),gap
,gap);
    }
    //plots a point in x,y (Circle)
    public void plotCircle(Graphics g, int x,int y ,Color c){
        int originX = (getX() + getWidth()) / 2;
        int originY = (getY() + getHeight()) / 2;
        g.setColor(c);
        g.fillOval(originX+(gap*x)-(gap/8), originY-(gap*y)-
(gap/8),gap/4 ,gap/4 );
    }

    public void plotLine(Graphics g,int x1,int y1 ,int x2, int y2,Color
col)
    {
        double x = x1;
        double y = y1;
        int dx = x2-x1;
        int dy = y2-y1;
        int step;
        if(Math.abs(dx) > Math.abs(dy))
            step = Math.abs(dx);
        else
            step = Math.abs(dy);

        for(int i  = 0; i<step; i++)
        {
            plotpoint(g,(int)x,(int)y,col);
```

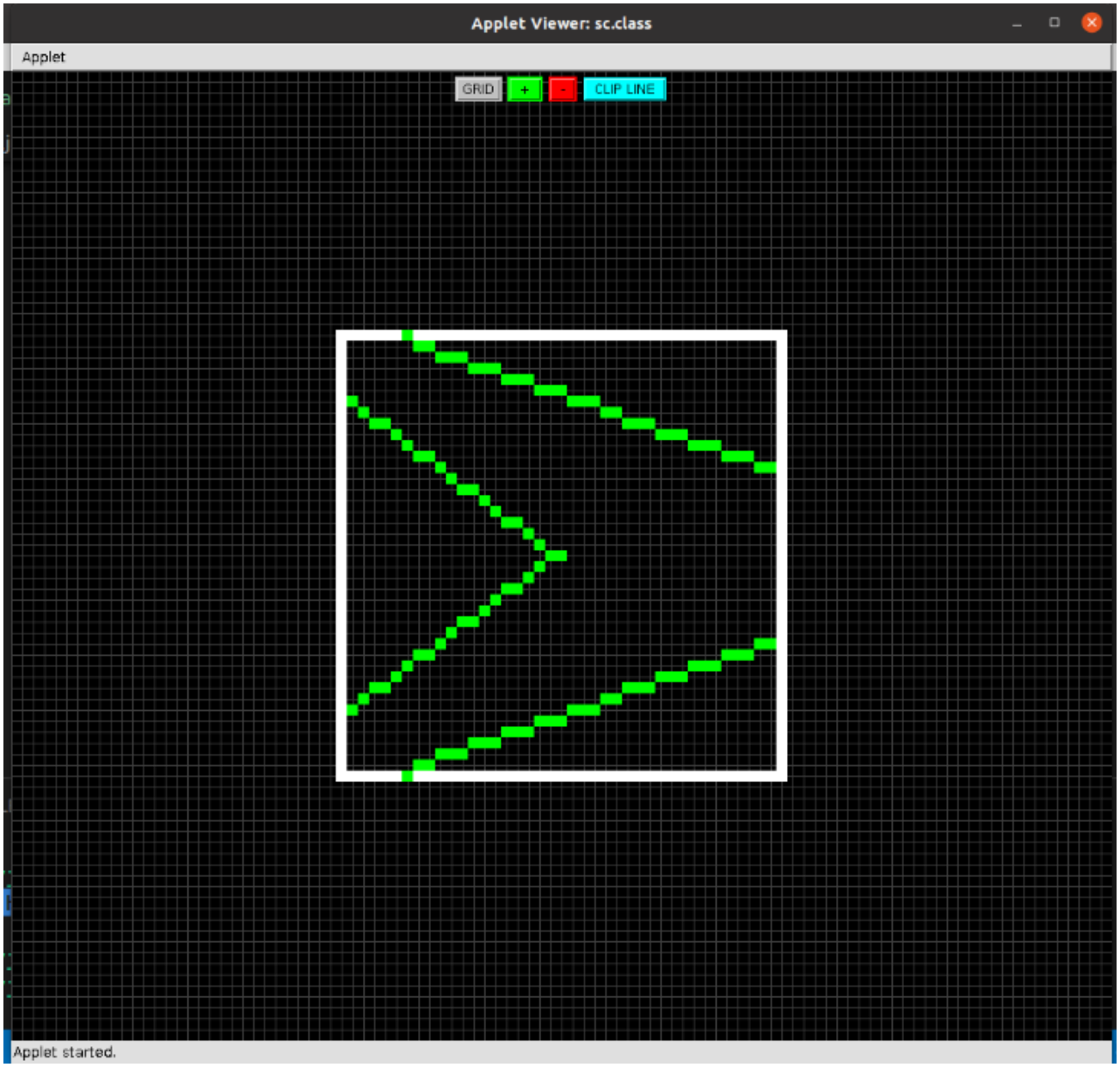
```
            x  = x+ (double)dx/step;
            y = y + (double)dy/step;
        }
    }

    public void makeWindow(){
        plotLine(getGraphics(), xmin, ymin, xmin, ymax , Color.WHITE);
        plotLine(getGraphics(), xmin, ymin, xmax, ymin , Color.WHITE);
        plotLine(getGraphics(), xmax, ymin, xmax, ymax , Color.WHITE);
        plotLine(getGraphics(), xmin, ymax, xmax, ymax , Color.WHITE);
        plotpoint(getGraphics(),xmax,ymax,Color.WHITE);
    }

    int computeCode(double x, double y){
    // initialized as being inside
        int code = INSIDE;

        if (x < xmin) // to the left of rectangle
            code |= LEFT;
        else if (x > xmax) // to the right of rectangle
            code |= RIGHT;
        if (y < ymin) // below the rectangle
            code |= BOTTOM;
        else if (y > ymax) // above the rectangle
            code |= TOP;
        return code;
    }

    public void cohenSutherlandClip(Graphics g, int x1, int y1, int x2,
int y2) {
        // Compute region codes for P1, P2
        int code1 = computeCode(x1, y1);
        int code2 = computeCode(x2, y2);

        // Initialize line as outside the rectangular window
        boolean accept = false;

        while (true) {
          if ((code1 == 0) && (code2 == 0)) {
            // If both endpoints lie within rectangle
            accept = true;
            break;
          } else if ((code1 & code2) != 0) {
            // If both endpoints are outside rectangle,
            // in same region
            break;
          } else {
            // Some segment of line lies within the
```

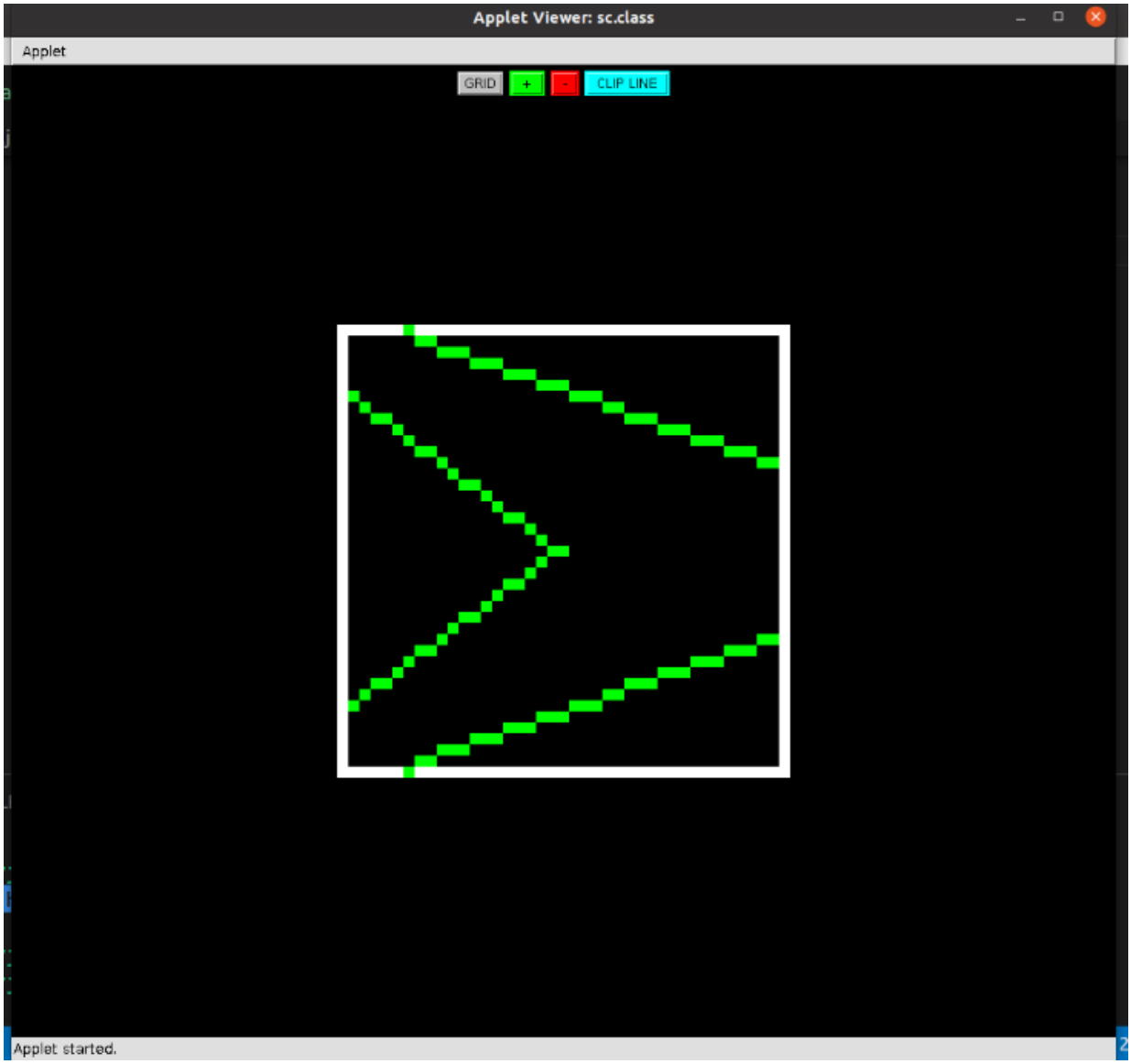
```
            // rectangle
            int code_out;
            int x = 0, y = 0;

            // At least one endpoint is outside the
            // rectangle, pick it.
            if (code1 != 0) code_out = code1; else code_out = code2;

            // Find intersection point;
            // using formulas y = y1 + slope * (x - x1),
            // x = x1 + (1 / slope) * (y - y1)
            if ((code_out & TOP) != 0) {
                // point is above the clip rectangle
                x = x1 + (x2 - x1) * (ymax - y1) / (y2 - y1);
                y = ymax;
              } else if ((code_out & BOTTOM) != 0) {
                // point is below the rectangle
                x = x1 + (x2 - x1) * (ymin - y1) / (y2 - y1);
                y = ymin;
              } else if ((code_out & RIGHT) != 0) {
                // point is to the right of rectangle
                y = y1 + (y2 - y1) * (xmax - x1) / (x2 - x1);
                x = xmax;
              } else if ((code_out & LEFT) != 0) {
                // point is to the left of rectangle
                y = y1 + (y2 - y1) * (xmin - x1) / (x2 - x1);
                x = xmin;
              }

              // Now intersection point x, y is found
              // We replace point outside rectangle
              // by intersection point
              if (code_out == code1) {
                x1 = x;
                y1 = y;
                code1 = computeCode(x1, y1);
              } else {
                x2 = x;
                y2 = y;
                code2 = computeCode(x2, y2);
              }
            }
          }
          if (accept) {
            plotLine(g, x1, y1, x2, y2,Color.green);

          }
          else
```

```
            System.out.println("Line rejected");
      }

    public void paint(Graphics g)
    {
        makeGrid(g,gap);
        plotCircle(g,0,0,Color.yellow);
        makeWindow();

        if(clip){
            cohenSutherlandClip(g, 0, 0, -40, 30);
            cohenSutherlandClip(g, 0, 0, -40, -30);

            cohenSutherlandClip(g, -40, 30, 40, 0);
            cohenSutherlandClip(g, -40, -30, 40, 0);
        }
        else{
            plotLine(g,0,0,-40,30, Color.RED);
            plotLine(g,0,0,-40,-30, Color.RED);

            plotLine(g,-40, 30,40 , 0, Color.RED);
            plotLine(g,-40, -30,40 , 0, Color.RED);
        }
    }
}
```

OUTPUT :


Applet Viewer: sc.class
Applet
GRID
+
-
CLIP LINE
Applet started.


Applet Viewer: sc.class
Applet
GRID
+
-
CLIP LINE
Applet started.


Applet Viewer: sc.class
Applet
GRID
+
-
CLIP LINE
Applet started.

Assignment 6 :

Code :

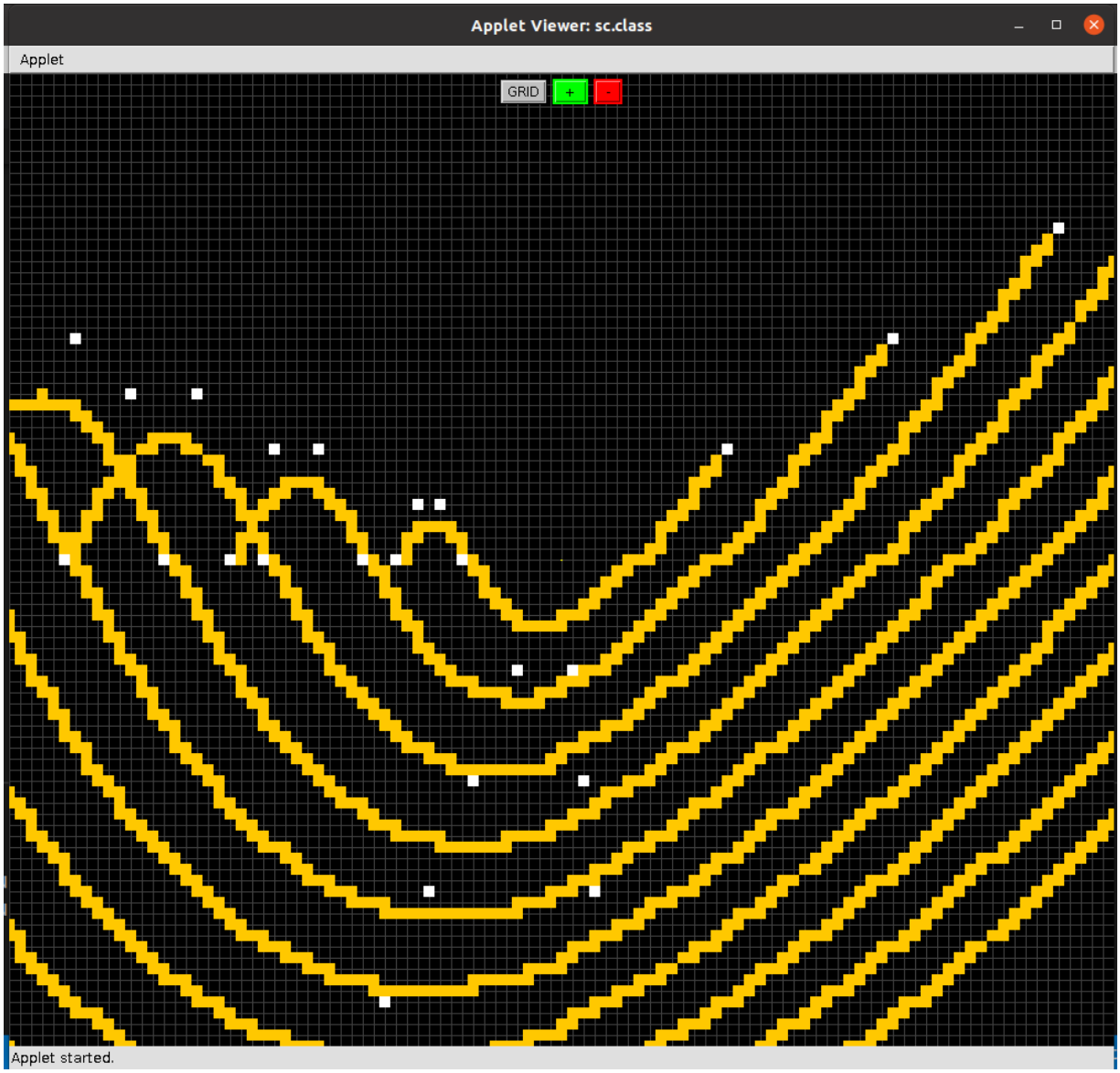
```
// This applet program only works on java jdk 1.8.0_312 or Lower

import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class sc extends Applet implements ActionListener
,MouseWheelListener{
    Button b1 = new Button(" + ");
    Button b2 = new Button(" - ");
    Button gridb = new Button("GRID");

    int gap = 10;
    boolean gridon = true;

    public void init() {
        Color buttonColor1 = new Color(0,255,0);
        Color buttonColor2 = new Color(255,0,0);
        Color bgColor = new Color(0,0,0);

        b1.setBackground(buttonColor1);
        b2.setBackground(buttonColor2);
        gridb.setBackground(Color.lightGray);

        add(gridb);
        add(b1);
        add(b2);

        addMouseWheelListener(this);
        b1.addActionListener(this);
        b2.addActionListener(this);
        gridb.addActionListener(this);

        setBackground(bgColor);
    }
    //for zoom using mouse wheel
    public void mouseWheelMoved(MouseWheelEvent e)
    {
        int z = e.getWheelRotation();
        zoom(z);
    }
    //button action listener
    public void actionPerformed(ActionEvent e) {
```

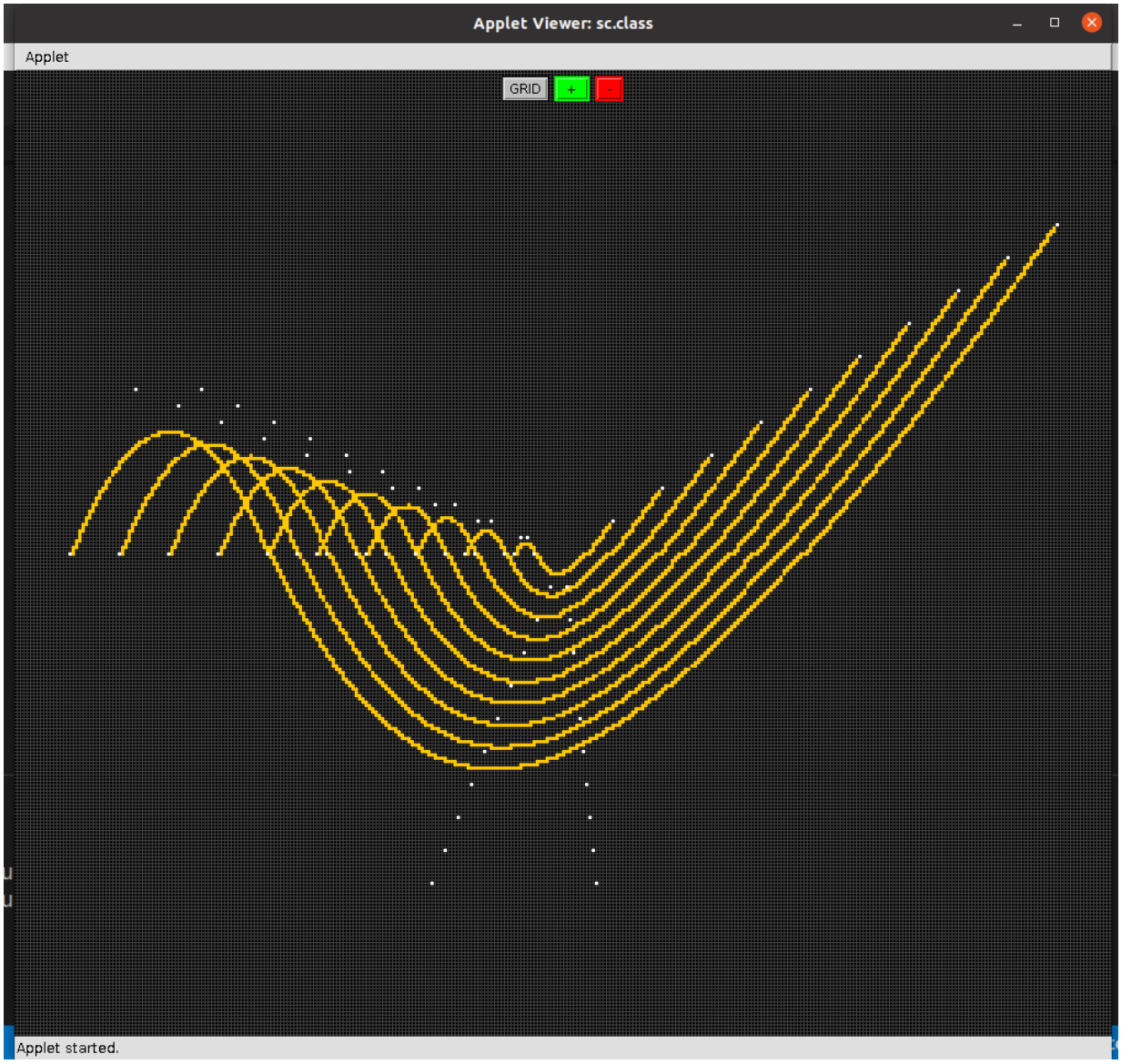
```
        if (e.getSource() == b1)
            zoom(+10);
        else if(e.getSource()==b2)
            zoom(-10);
        else if(e.getSource()==gridb)
        {
            gridon= !gridon;
            repaint();
        }

    }
    //function to make a grid with respect to standard cartesian
coordinates
    public void makeGrid(Graphics g, int gap)
    {
        if(gridon==false)
            return;
        if(gap<=0|| gap>getHeight())
            return ;
        int originX = (getX() + getWidth()) / 2;
        int originY = (getY() + getHeight()) / 2;
        g.setColor(Color.DARK_GRAY);
        g.drawLine(originX, originY - getHeight() / 2, originX, originY
+ getHeight() / 2);
        g.drawLine(originX - getWidth() / 2, originY, originX +
getWidth() / 2, originY);
        g.setColor(Color.DARK_GRAY);
        for (int x = gap; x <= getWidth(); x += gap) {
            g.drawLine(originX + x, 0, originX + x, getHeight());
            g.drawLine(originX - x, 0, originX - x, getHeight());
        }
        for (int y = gap; y <= getHeight(); y += gap) {
            g.drawLine(0, originY + y, getWidth(), originY + y);
            g.drawLine(0, originY - y, getWidth(), originY - y);
        }
    }
    //for zoom feature
    public void zoom(int i)
    {
        if(i>0)
            gap+=gap/10+1;
        else if(i<0)
            gap-=gap/10+1;
        if(gap<0)
            gap = 1000;
        if(gap>1600)
            gap = 2;
        repaint();
```

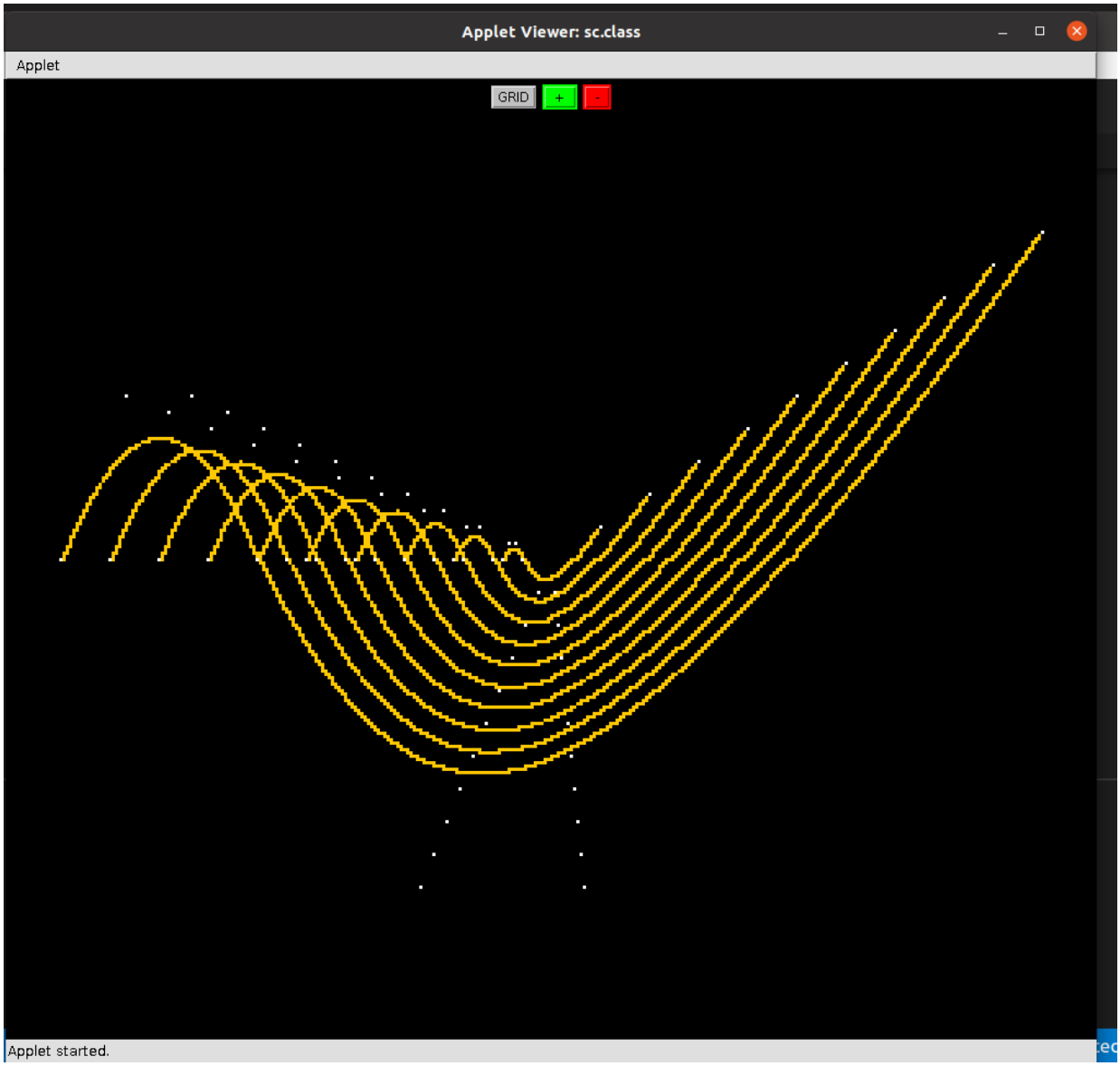
```
    }
    //plots a square at point (Square) x,y in grid
    public void plotpoint(Graphics g, int x,int y ,Color c){
        int originX = (getX() + getWidth()) / 2;
        int originY = (getY() + getHeight()) / 2;
        g.setColor(c);
        g.fillRect(originX+(gap*x)-(gap/2), originY-(gap*y)-(gap/2),gap
,gap);
    }
    //plots a point in x,y (Circle)
    public void plotCircle(Graphics g, int x,int y ,Color c){
        int originX = (getX() + getWidth()) / 2;
        int originY = (getY() + getHeight()) / 2;
        g.setColor(c);
        g.fillOval(originX+(gap*x)-(gap/8), originY-(gap*y)-
(gap/8),gap/4 ,gap/4 );
    }

    public void plotLine(Graphics g,int x1,int y1 ,int x2, int y2,Color
col)
    {
        double x = x1;
        double y = y1;
        int dx = x2-x1;
        int dy = y2-y1;
        int step;
        if(Math.abs(dx) > Math.abs(dy))
            step = Math.abs(dx);
        else
            step = Math.abs(dy);

        for(int i  = 0; i<step; i++)
        {
            plotpoint(g,(int)x,(int)y,col);
            x  = x+ (double)dx/step;
            y = y + (double)dy/step;
        }
    }

    public void plotPoints(int[] x ,int[] y)
    {
        for(int i = 0 ;i< x.length ;i++)
        {
            plotpoint(getGraphics(),x[i],y[i],Color.WHITE);
        }
    }
```

```
    public void bezierCurve(int[] x , int[] y)
    {
        double xu = 0.0 , yu = 0.0 , u = 0.0 ;

        for(u = 0.0 ; u <= 1.0 ; u += 0.0001)
        {
            xu = Math.pow(1-u,3)*x[0]+3*u*Math.pow(1-
u,2)*x[1]+3*Math.pow(u,2)*(1-u)*x[2]
                 +Math.pow(u,3)*x[3];
            yu = Math.pow(1-u,3)*y[0]+3*u*Math.pow(1-
u,2)*y[1]+3*Math.pow(u,2)*(1-u)*y[2]
                +Math.pow(u,3)*y[3];
            plotpoint(getGraphics(), (int)xu , (int)yu,Color.orange) ;
        }
    }

    public void paint(Graphics g)
    {
        makeGrid(g,gap);
        for(int k = 1;k<=10;k++){
            int[] x2 = { 15*k ,  1*k, -4*k, -9*k};
            int[] y2 = {  10*k , -10*k, -10*k,   0*k};
            int[] x3 = {-9*k , -11*k, -13*k,-15*k};
            int[] y3 = {  0*k ,  5*k,  5*k,   0*k};
            bezierCurve(x2,y2);
            bezierCurve(x3,y3);
            plotPoints(x2,y2);
            plotPoints(x3,y3);
        }
        plotCircle(g,0,0,Color.yellow);
    }
}
```

Output:


Applet Viewer: sc.class
Applet
GRID
+
-
Applet started.


Applet Viewer: sc.class
Applet
GRID
+
-
Applet started.



----------X----------X----------X----------