GRAPHICS IN JAVA

## JAVA SWING GRAPHICS ENVIRONMENT AND JFRAME AND JPANEL CLASSES

In Java, two graphics environment (Graphics application development environment) is defined. The first one is awt. The second one is swing library which came out with more sophisticated drawing facilities. In order to understand java drawing capabilities, the first concept is to understand java coordinate system. In java graphics screen, if anything is drawn a graphic screen coordinates is used. Coordinate unit is pixel(Picture elements) in your graphic window. Every computer graphic window consist of pixels.

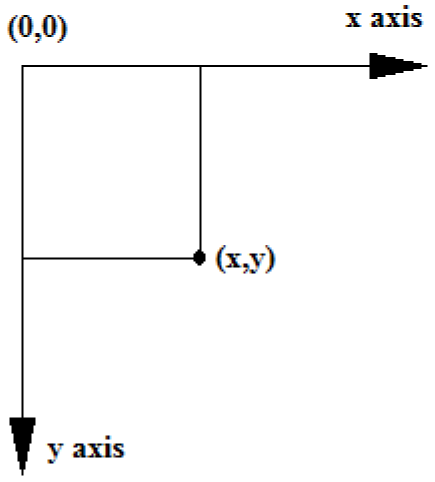


**Figure 10.1 Java Graphic coordinate system**

In order to graph someting, class JFrame is used. Actual drawings will be prepared in another class JPanel and will be added up into JFrame to show the graphics. In order to show graphics in JFrame, a small interface program **FrameGraphic** is prepared for you.

**FrameGraphic**

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class FrameGraphic extends JFrame
{  private static final long serialVersionUID =784768746L;
   JPanel d;
   public FrameGraphic(String a,JPanel di)
   {    super(a);
       d=di;
        add(di);
   }
    public FrameGraphic(String a,JPanel di[],String layout_type)
   {    super(a);
       int n=di.length;
       if(layout_type.equals("GridLayout_column"))
       {setLayout(new GridLayout(n,1));
        for(JPanel x:di)
```

```
     {add(x);}
    } //gridlayout row
    else if(layout_type.equals("GridLayout_row"))
    {setLayout(new GridLayout(1,n));
     for(JPanel x:di)
     {add(x);}
    } //gridlayout column
     else if(layout_type.equals("JTabbedPane"))
     {    JTabbedPane tp=new JTabbedPane();
                 for(int i=0;i<n;i++)
                 {tp.add("Page "+i,di[i]);}
                 add(tp);
           } //JTabbedPane
}

public FrameGraphic(String a,JPanel di,Color bg)
{    super(a);
    d=di;
    add(di);
    getContentPane().setBackground(bg);
}

public static void plot(String a,JPanel di[],String layout_type)
{    FrameGraphic f = new FrameGraphic(a,di,layout_type);
    f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    f.setSize(400,400);
    f.setVisible(true);
}
     public static void plot(String a,JPanel di)
{    FrameGraphic f = new FrameGraphic(a,di);
    f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    f.setSize(400,400);
    f.setVisible(true);
}
   public static void plot(String a,JPanel di,Color bc)
{    FrameGraphic f = new FrameGraphic(a,di,bc);
    f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
     f.getContentPane().setBackground(bc);
    f.setSize(400,400);
    f.setVisible(true);
}

public static void plot(String a,JPanel di,int x,int y)
{    FrameGraphic f = new FrameGraphic(a,di);
    f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    f.setSize(x,y);
    f.setVisible(true);
}

 public static void plot(String a,JPanel di,int x,int y,Color bc)
{    FrameGraphic f = new FrameGraphic(a,di,bc);
```

```
        f.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        f.setSize(x,y);
        f.setVisible(true);
  }
}
```

WelcomeP

```
import javax.swing.*;
import java.awt.Graphics;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class WelcomeP extends JPanel
{
 String isim;
 public WelcomeP()
 {isim=JOptionPane.showInputDialog("enter your name : ");}
 public void paint(Graphics g)
 {
 Graphics2D g2=(Graphics2D)g;
 g2.setFont(new Font("Serif",Font.ITALIC,24));
 g2.drawString("Welcome to Java class " + isim,50,50);
 }
}
```

pictureP

```
       import java.io.*;
       import java.applet.Applet;
       import java.awt.*;
       import java.awt.event.*;
       import java.util.*;
       import javax.swing.*;
       import javax.swing.table.*;
       import java.awt.*;
       import java.awt.event.*;
       import javax.swing.event.*;
       import java.awt.geom.*;
       import java.net.URL;
       import java.awt.image.*;
       import java.util.*;

       public class pictureP extends JPanel
       {
        private Image picture;

        public pictureP(String s)
        {super();
         URL url = getClass().getResource(s);
         picture = getToolkit().getImage(url);
        }


        public void paint(Graphics g)
        {
        Graphics2D g2=(Graphics2D)g;
        Dimension d=getSize();
        int dx = d.width;
        int dy = d.height;
        g2.drawImage( picture, 0, 0,dx,dy, this);
        }
}
```

lineP

```
 import javax.swing.*;
import java.awt.Graphics;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class lineP extends JPanel
{   private static final long serialVersionUID = 957857L;
              public int x0,x1;
   public int y0,y1;
   public float thick;
 public lineP()
 {x0=Integer.parseInt(JOptionPane.showInputDialog("x0 : "));
  y0=Integer.parseInt(JOptionPane.showInputDialog("y0 : "));
  x1=Integer.parseInt(JOptionPane.showInputDialog("x1 : "));
  y1=Integer.parseInt(JOptionPane.showInputDialog("y1 : "));
  thick=10;
  //point are in pixel coordinaes
 }
  public lineP(int x0i,int y0i, int x1i,int y1i,float thicki)
 {x0=x0i;
  y0=y0i;
  x1=x1i;
  y1=y1i;
  thick=thicki;
  //point are in pixel coordinaes
 }
 public void paint(Graphics g)
 {
 Graphics2D g2=(Graphics2D)g;
 g2.setFont(new Font("Serif",Font.BOLD,24));
 this.setBackground(Color.YELLOW);
 g2.setColor(Color.yellow);
 g2.setStroke(new BasicStroke(thick));
 Line2D x=new Line2D.Double(x0,y0,x1,y1);
 g2.draw(x);
 }}
```

rectangleP

```
import java.awt.Graphics;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class rectangleP extends JPanel
{
 public void paintComponent(Graphics g)
 {
 Graphics2D g2=(Graphics2D)g;
 super.paintComponent(g);
 g2.setFont(new Font("Serif",Font.BOLD,24));
 g2.setColor(Color.red);
 this.setBackground(Color.MAGENTA);
 g2.setStroke(new BasicStroke(10.0f));
 Rectangle2D x=new Rectangle2D.Double(50,50,300,200);
 g2.draw(x);
 }
}
```

rectangleP1

```
import javax.swing.*;
import java.awt.Graphics;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class rectangleP1 extends JPanel
{
 public void paintComponent(Graphics g)
 {
 Graphics2D g2=(Graphics2D)g;
 super.paintComponent(g);
 g2.setFont(new Font("Serif",Font.BOLD,24));
 g2.setColor(Color.RED);
 this.setBackground(Color.GREEN);
 g2.setStroke(new BasicStroke(10.0f));
 Rectangle2D x=new Rectangle2D.Double(50,50,300,200);
 g2.fill(x);
 }
}
```

**ellipseP**

```
import javax.swing.*;
import java.awt.Graphics;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class ellipseP extends JPanel
{
    public void paintComponent(Graphics g) {
              super.paintComponent(g);
    Graphics2D g2=(Graphics2D)g;
    g2.setFont(new Font("Serif",Font.BOLD,24));
    g2.setColor(Color.RED);
    this.setBackground(new Color(0,0,255));
    g2.setStroke(new BasicStroke(2.0f));
    Ellipse2D x=new Ellipse2D.Double(50,50,500,200);
    g2.draw(x);
   }
}
```

**ellipseP1**

```
import javax.swing.*;
import java.awt.Graphics;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class ellipseP1 extends JPanel
{
    public void paintComponent(Graphics g) {
              super.paintComponent(g);
    Graphics2D g2=(Graphics2D)g;
    g2.setFont(new Font("Serif",Font.BOLD,24));
    g2.setColor(Color.RED);
    this.setBackground(new Color(255,255,255));
    float dash3[] = {10.0f,3.0f,3.0f};
    BasicStroke d3 = new BasicStroke(3.0f,BasicStroke.CAP_BUTT,
                                      BasicStroke.JOIN_MITER,
                                      3.0f, dash3, 2.0f);
    g2.setStroke(d3);
    Ellipse2D x=new Ellipse2D.Double(50,50,500,200);
    g2.draw(x);
   }
}
```

**graph2P**

```
import javax.swing.*;
import java.awt.Graphics;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
public class graph2P extends JPanel
{

    public void paintComponent(Graphics g)
    {  super.paintComponent(g);
      Graphics2D g2 = (Graphics2D)g;
      g2.setPaint(Color.blue);
      Ellipse2D elips1=new Ellipse2D.Double(55,55,90,30);
      g2.fill(elips1);
      g2.setPaint(Color.black);
      Ellipse2D elips2=new Ellipse2D.Double(50,50,100,40);
      g2.draw(elips2);
      g2.setPaint(Color.black);
      Ellipse2D elips3=new Ellipse2D.Double(50,150,100,40);
      g2.draw(elips3);
      GradientPaint kirmizidanbeyaza=new GradientPaint(250,50,Color.red,350,90,Color.white);
      g2.setPaint(kirmizidanbeyaza);
      Ellipse2D elips4=new Ellipse2D.Double(250,50,100,40);
      g2.fill(elips4);
      GradientPaint kirmizidanmaviye=new GradientPaint(250,150,Color.red,350,190,Color.blue);
      g2.setPaint(kirmizidanmaviye);
      Ellipse2D elips5=new Ellipse2D.Double(250,150,100,40);
      g2.fill(elips5);
      g2.setPaint(Color.black);
      g2.draw(elips5);
   }
}
```

**ellipseP2**

```
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
import java.awt.geom.*;
import java.net.URL;
import java.awt.image.*;
import java.util.*;

public class ellipseP2 extends JPanel
{
   TexturePaint tp = getImageTexture("fall.jpg");

   public TexturePaint getImageTexture(String imageFile)
   {
   URL url = getClass().getResource(imageFile);
   Image img = getToolkit().getImage(url);
   try {
     MediaTracker tracker = new MediaTracker(this);
     tracker.addImage(img, 0);
     tracker.waitForID(0);
   } catch (Exception e) {}
```

```
    int width = img.getWidth(this);
    int height = img.getHeight(this);
    BufferedImage buffImg = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
    Graphics g = buffImg.getGraphics();
    g.drawImage(img, 0, 0, this);
    Rectangle2D rect = new Rectangle(0, 0, width, height);
    return new TexturePaint(buffImg, rect);
    }

            public void paintComponent(Graphics g) {
            super.paintComponent(g);
     Graphics2D g2=(Graphics2D)g;
     g2.setPaint(tp);
     g2.setFont(new Font("Serif",Font.BOLD,24));
     //g2.setColor(Color.RED);
     //this.setBackground(new Color(255,255,255));
     Ellipse2D x=new Ellipse2D.Double(50,50,500,200);
     g2.fill(x);
    }
}
```

**starP**

```
import javax.swing.*;
import java.awt.Graphics;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class starP extends JPanel
{

public void paint(Graphics g)
{
int x[]={55,67,109,73,83,55,27,37,1,43};
int y[]={0,36,36,54,96,72,96,54,36,36};
Graphics2D g2=(Graphics2D)g;
GeneralPath star=new GeneralPath();
star.moveTo(x[0],y[0]);
for(int i=1;i<x.length;i++)
   {star.lineTo(x[i],y[i]);}
star.closePath();
g2.setColor(Color.blue);
g2.draw(star);

  }
}
```

**quadcurveP**

```
import javax.swing.*;
import java.awt.Graphics;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class quadcurveP extends JPanel
{
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D)g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON);
        Dimension boyut=getSize();
        int dx=boyut.width;
        int dy=boyut.height;
        g2.setPaint(Color.BLACK);
        g2.draw3DRect(0,0,dx-3,dy-3,true);
        g2.draw3DRect(3,3,dx-7,dy-7,false);
        g2.setPaint(Color.BLACK);
        QuadCurve2D qc2=new QuadCurve2D.Double(0,125,140,225,225,150);
        g2.draw(qc2);
        QuadCurve2D qc2_1=new QuadCurve2D.Double(0,200,155,225,225,170);
        g2.setPaint(Color.blue);
        g2.fill(qc2_1);
    }

}
```

**qeneralcurveP**

```
import javax.swing.*;
import java.awt.Graphics;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class qeneralcurveP extends JPanel
{
        public void paintComponent(Graphics g) {
        super.paintComponent(g);
                Graphics2D g2 = (Graphics2D)g;
        setBackground(Color.lightGray);
        Dimension boyut=getSize();
        int dx=boyut.width;
        int dy=boyut.height;
        g2.setStroke(new BasicStroke(3));
        g2.draw3DRect(0,0,dx-3,dy-3,true);
        g2.draw3DRect(3,3,dx-7,dy-7,false);
        GeneralPath shape=new GeneralPath(GeneralPath.WIND_EVEN_ODD);
        shape.moveTo(20,20);
        //quadratik ekleme
        shape.quadTo(160,120,245,45);
        //kübik ekleme
        shape.curveTo(195,95,295,145,245,195);
        shape.curveTo(-80,110,345,110,20,195);
        shape.curveTo(400,250,200,250,20,20);
        g2.draw(shape);
    }
}
```

**writeP**

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
public class writeP extends JPanel
{ //grafik penceresine yazı yazar
 String s[];
 public writeP(String si[])
 {super();
 s=new String[si.length];
 for(int i=0;i<si.length;i++){s[i]=si[i];}
 }

 public void paintComponent(Graphics g)
   {   super.paintComponent(g);
            Graphics2D g2 = (Graphics2D)g;
       g2.setFont(new Font("Serif",Font.BOLD,24));
       g2.setColor(Color.blue);
       g2.setBackground(Color.yellow);
       for(int i=0;i<s.length;i++)
             g2.drawString(s[i],50,50+25*i);
   }
}
```

**Interface if_x**

```
/ if_x interface for f(x) one variable input general function
// M. Turhan COBAN
// EGE University, school of Engineering, Department of Mech Engineering
// Sub functions:
// double func(double x)  calculates y=f(x) it is main interface function (undefined to be define in later stage)
// Default functions that depends on func(double x)
// double[][] func(double x[]) calculates and array of yi=f(xi) i=1..n and then return xi,yi matrix
// double[][] func(double x1,double x2,double dx) calculates and array of yi=f(xi) i=1..n and x1<=xi<=x2  xi=x1+i*dx then return xi,yi matrix
// double[][] func(double x1,double x2,int n) calculates and array of yi=f(xi) i=1..n and x1<=xi<=x2   dx=(x2-x1)(n-1) xi=x1+i*dx then return
xi,yi matrix
// double[][] calc(double x1,double x2,double dx) calculates and array of yi=f(xi) i=1..n and x1<=xi<=x2  xi=x1+i*dx then return xi,yi matrix
// double[][] calc(double x1,double x2,int n) calculates and array of yi=f(xi) i=1..n and x1<=xi<=x2   dx=(x2-x1)(n-1) xi=x1+i*dx then return
xi,yi matrix
// dfunc(double x) first derivative of func df(x)/dx (finite difference approach)
// dfunc(double x,double h) first derivative of func df(x)/dx (finite difference approach) dx=h
// dfunc2(double x) second  derivative of func d2f(x)/dx2 (finite difference approach)
// dfunc3(double x) third  derivative of func d2f(x)/dx2 (finite difference approach)
// dfunc(double x,int N,int Mi,double hi) N'th derivative of func dNf(x)/dxN (finite difference approach) dx=h Mi : order of polynomial used
in finite difference definition
// dfunc(double x,int N) N'th derivative of func dNf(x)/dxN (finite difference approach) dx=h h=0.001*N*N
// double[][] gauss_legendre_coefficients(double x1,double x2,int n) Gauss-Legendre coefficients for GL integrations
// integral(double x1,double x2,int n)  Gauss-Legendre integral of degree n
// double adaptive_simpson_integral(double a,double b) Adaptive simpson integral
// double adaptive_gauss_legendre_integral(double x1,double x2)
// double gauss_kronrad_integral(double a,double b,double eps,int n)
// bisection(double a,double b,double eps) bisection root finding methods betwee a<=x<=b with error level eps
// double bisection(double a,double b) bisection root finding methods betwee a<=x<=b with error level eps=10^-10
// double secant(double x) Root finding method secant(Newton-Raphson)
// double illinois(double xa,double xu,double esi)  illinois root finding methods betwee a<=x<=b with error level esi
// double newton_bisection(double x1, double x2) Root finding method: Newton-Raphson - Bisection combined
// double enlarge(double x0,double dx) enlarge root finding section if root is not available
// double[] roots_secant(double a,double b) Find all roots by using secant method double return (can not catch complex roots)
// double[] roots_bisection(double a,double b) Find all roots by using bisection methods (can not catch complex roots)
// double bisection_secant(double x1,double x2) bisection-secant combined root finding
// newton_opt(double x) Newton optimisation algorithm
// double golden_opt(double a,double b) Golden search (Fibonnachi ) optimisation algorithm
// double quadratic_poly_opt(double x0,double x1,double x2) Quadratic polynomial optimisation
// double quadratic_poly_opt(double x0,double x2) Quadratic polynomial optimisation ,x1=x0+(x2-x0)/(2.0+0.01*Math.random());
// double cubic_search_opt(double x0,double x1) Cubic polynomial optimisation
// double[] brent_opt(double ax,double bx,double cx) brent optimisation
// double[] brent_opt(double ax,double cx) brent optimisation
// double dbrent_opt(f_x f,double ax,double bx,double cx) brent optimisation
// double[] dbrent_opt(double ax,double cx) brent optimisation

import java.io.*;
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import static java.lang.Math.*;
  // single function single independent variable
  // example f=x*x
  // includes full set of derivatives
  // Reference :"Generation of Finite Difference Formulas on Arbitrary Spaced Grids",
  // Bength Fornberg, Mathematics of Computation, Volume 51, Number 184, October 1988
  // pages 699-706
  //interface (empty function) version
@FunctionalInterface
interface if_x
{  public double func(double x);
//multiple data
 default double[][] func(double x[])
 { int n=x.length;
   double a[][]=new double[2][n];
   for(int i=0;i<n;i++)
   {a[0][i]=x[i];
    a[1][i]=func(x[i]);
   }
   return a;
 }
  default double[][] func(double x1,double x2,double dx)
 {  double x[]=calc(x1,x2,dx);
                  int n=x.length;
     double a[][]=new double[2][n];
     for(int i=0;i<n;i++)
     {a[0][i]=x[i];
      a[1][i]=func(x[i]);
     }
     return a;
 }
   default double[][] func(double x1,double x2,int n)
 {  double x[]=calc(x1,x2,n);
     double a[][]=new double[2][n];
     for(int i=0;i<n;i++)
     {a[0][i]=x[i];
      a[1][i]=func(x[i]);
     }
     return a;
 }
 default double[] calc(double x1,double x2,double dx)
 {  int n=(int)((x2-x1)/dx+1e-5);
   double xx[]=new double[n];
   for(int i=0;i<n;i++)
   {xx[i]=x1+dx*i;}
   return xx;
 }
  default double[] calc(double x1,double x2,int n)
 { double dx=(x2-x1)/(n-1);
   double xx[]=new double[n];
   for(int i=0;i<n;i++)
   {xx[i]=x1+dx*i;}
   return xx;
 }

//first order derivative
default double dfunc(double x)
{double h=1.0e-6;
 int n=1;
 int M=20;
 return dfunc(x,n,M,h);
}
default double dfunc(double x,double h)
{
 int n=1;
 int M=20;
 return dfunc(x,n,M,h);
}
//second order derivative
```

```
default double dfunc2(double x)
{double h=1.0e-6;
 int n=2;
 int M=10;
 return dfunc(x,n,M,h);
}
//third order derivative
default double dfunc3(double x)
{double h=1.0e-4;
 int n=3;
 int M=10;
 return dfunc(x,n,M,h);
}

default double dfunc(double x,int N,int Mi,double hi)
{// order of the maximum derivative
 // N order of derivative
 // M degree of difference formula
double c[][][];
double alpha[];
double h;
int M=Mi;
double a[]=new double[0];
h=0.01;
double x0=0;
double alphai[]={0,1,-1,2,-2,3,-3,4,-4,5,-5,6,-6,7,-7,8,-8,9,-9,10,-10,11,-11,12,-12,13,-13,14,-14,15,
-15,16,-16,17,-17,18,-18,19,-19,20,-20,21,-21,22,-22,23,-23,24,-24,25,-25,26,-26,27,-27,28,-28,29,-29,30,-30,
31,-31,32,-32,33,-33,34,-34,35,-35,36,-36,37,-37,38,-38,39,-39,40,-40,41,-41,42,-42,43,-43,44,-44,45,-45,46,
-46,47,-47,48,-48,49,-49,50,-50,51,-51,52,-52,53,-53,54,-54,55,-55,56,-56,57,-57,58,-58,59,-59,60,-60,
-61,61,-62,62,-63,63,-64,64,-65,65,-66,66,-67,67,-68,68,-70,70,-71,71,-72,72,-73,73,-74,74,-75,75,
-76,76,-77,77,-78,78,-79,79,-80,80,-81,81,-82,82,-83,83,-84,84,-85,85,-86,86,-87,87,
-88,88,-89,89,-90,90,-91,91,-92,92,-93,93,-94,94,-95,95,-96,96,-97,97,-98,98,-99,99,-100,100};
alpha=alphai;
int N1=alpha.length-1;
// M   degree of highest derivative
// N+1 number of coefficients
double delta[][][]=new double[N1+1][N1+1][M+1];
double c1,c2,c3;
delta[0][0][0]=1.0;
c1=1.0;
for(int n=1;n<=N1;n++)
{  c2=1;
   for(int nu=0;nu<=(n-1);nu++)
   {c3=alpha[n]-alpha[nu];
    c2=c2*c3;
    if(n<=M) delta[n-1][nu][n]=0.0;
    for(int m=0;m<=Math.min(n,M);m++)
    {  if(m==0)
       { delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m])/c3;}
       else
       {delta[n][nu][m]=((alpha[n]-x0)*delta[n-1][nu][m]-m*delta[n-1][nu][m-1])/c3;}
    }//next m
   }//next nu
   for(int m=0;m<=Math.min(n,M);m++)
   {   if(m==0)
                 {delta[n][n][m]=c1/c2*(-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
                 else
                 {delta[n][n][m]=c1/c2*(m*delta[n-1][n-1][m-1]-(alpha[n-1]-x0)*delta[n-1][n-1][m]);}
   }//next m
   c1=c2;
}//next n
c=delta;
  if(Mi<N) M=N;
  else M=Mi;
  h=hi;
  double deriv=0;
  double h1=1/h;
  double h2=1;
  for(int j=0;j<N;j++)
  {h2*=h1;}
  for(int i=0;i<c[0].length;i++)
  {   deriv+=c[M][i][N]*func(x+alpha[i]*h);}
  return deriv*h2;}
default double dfunc(double x,int N)
{int M=30;double h=0.01*N*N;return dfunc(x,N,M,h);}
default double[][] gauss_legendre_coefficients(double x1,double x2,int n)
{   //calculates legendre gauss-coefficients as coefficients of the integral
            //for n terms
            double EPS=3.0e-8;
            int m,j,i;
            double z1,z,xm,xl,pp,p3,p2,p1;
   //double x[]=new double[n];
   //double w[]=new double[n];
   double a[][]=new double[2][n];//a[0][i]=x[i]   a[1][i]=w[i]
            m=(n+1)/2;
            xm=0.5*(x2+x1);
            xl=0.5*(x2-x1);
            for (i=1;i<=m;i++)
            { z=Math.cos(Math.PI*((i-0.25)/(n+0.5)));
                        do {
                                    p1=1.0;
                                    p2=0.0;
                                    for (j=1;j<=n;j++) {
                                                p3=p2;
                                                p2=p1;
                                                p1=((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
                                    }
                                    pp=n*(z*p1-p2)/(z*z-1.0);
                                    z1=z;
                                    z=z1-p1/pp;
                        } while (Math.abs(z-z1) > EPS);
                        a[0][i-1]=xm-xl*z;
                        a[0][n-i]=xm+xl*z;
                        a[1][i-1]=2.0*xl/((1.0-z*z)*pp*pp);
                        a[1][n-i]=a[1][i-1];
            }
return a;
}
//Gauss Legendre integration
default double integral(double x1,double x2)
{ int n=10;
  return integral(x1,x2,n);
}
default double integral(double x1,double x2,int n)
{
//n : number of integral coefficients
// this routine first generates gauss legendre coefficients
// for [x1,x2] band
// then calculates gauss legendre integral
double a[][]=new double[2][n];
a=gauss_legendre_coefficients(x1,x2,n);
double z=0;
for(int i=0;i<n;i++)
{ z+=a[1][i]*func(a[0][i]);
}
return z;
}
default double adaptive_simpson_integral(double a,double b)
{  double eps=1e-20;
   int MAX_ITERATION=100;
            int i;
   double h=(b-a);
   double h2=h/4.0;
            double x[]=new double[5];
   double f[]=new double[5];
   for(i=0;i<5;i+=2) {x[i]=a+i*h2;f[i]=func(x[i]);}
            double sum=h*(f[0]+4.0*f[2]+f[4])/6.0;
            double Q=0;
   Q=simpson1_3(x,f,sum,eps,0,MAX_ITERATION,Q);
   return Q;
```

```
}
default double simpson1_3(double x[],double f[],double sum,double eps,int k,int MAX_ITERATION,double Q)
{       //Newton-cotes integral 3 points
        //adaptive Simpson 1/3 rule
        double h=(x[4]-x[0]);
        double h1=h/2.0;
        double h2=h/4.0;
        int p,i;
        double x1[]=new double[5];
        double f1[]=new double[5];
        double x2[]=new double[5];
        double f2[]=new double[5];
        k++;
        for(i=1;i<5;i+=2) {x[i]=x[0]+i*h2;f[i]=func(x[i]);}
        double sum1=h1*(f[0]+4.0*f[1]+f[2])/6.0;
        double sum2=h1*(f[2]+4.0*f[3]+f[4])/6.0;
        x1[0]=x[0];f1[0]=f[0];
        x1[2]=x[1];f1[2]=f[1];
        x1[4]=x[2];f1[4]=f[2];
        x2[0]=x[2];f2[0]=f[2];
        x2[2]=x[3];f2[2]=f[3];
        x2[4]=x[4];f2[4]=f[4];
        if(1.0/15.0*Math.abs(sum1+sum2-sum)<eps || k>MAX_ITERATION) Q+=sum1+sum2;
        else Q+=simpson1_3(x1,f1,sum1,eps,k,MAX_ITERATION,Q)+simpson1_3(x2,f2,sum2,eps,k,MAX_ITERATION,Q);
        return Q;
}
default double gauss_kronrad_integral(double a,double b,double eps,int n)
    {
                //eps error limit for example 1e-10
        int Aused=0;
                boolean sonuc;
        double integral=0;
        double A[][];
        int Aw = 0;
        double totalerror = 0;
        double[] c;
        double[] xg;
        double[] xk;
        int nn = 0;
        int ng = 0;
        int i = 0;
        int j = 0;
        int h = 0;
        double v = 0;
        double k1 = 0;
        double k2 = 0;
        double intg = 0;
        double intk = 0;
        double ta = 0;
        double tb = 0;
        Aw = 4;
        A = new double[n-1+1][Aw-1+1];
        nn = 61;
        ng = 15;
        c = new double[nn];
        xk = new double[nn];
        xg = new double[nn];
        //Gauss katsayilari
        xg[0] = 0.007968192496166605615465883474674;
        xg[1] = 0.018466468311090959142302131912047;
        xg[2] = 0.028784707883323369349719179611292;
        xg[3] = 0.038799192569627049596801936446348;
        xg[4] = 0.048402672830594052902938140422808;
        xg[5] = 0.057493156217619066481721689402056;
        xg[6] = 0.065974229882180495128128515115962;
        xg[7] = 0.073755974737705206268243850022191;
        xg[8] = 0.080755895229420215354694938460530;
        xg[9] = 0.086899787201082979802387530715126;
        xg[10] = 0.092122522237786128717632707087619;
        xg[11] = 0.096368737174644259639468626351810;
```

```
xg[12] = 0.0995934205867952670627802821035 69;
xg[13] = 0.101762389748405504596428952168554;
xg[14] = 0.102852652893558840341285636705415;
c[0] = 0.999484410050490637571325895705811;
c[1] = 0.996893484074649540271630050918695;
c[2] = 0.991630996870404594858628366109486;
c[3] = 0.983668123279747209970032581605663;
c[4] = 0.973116322501126268374693868423707;
c[5] = 0.960021864968307512216871025581798;
c[6] = 0.944374444748559979415831324037439;
c[7] = 0.926200047429274325879324277080474;
c[8] = 0.905573307699907798546522558925958;
c[9] = 0.882560535792052681543116462530226;
c[10] = 0.857205233546061098958658510658944;
c[11] = 0.829565762382768397442898119732502;
c[12] = 0.799727835821839083013668942322683;
c[13] = 0.767777432104826194917977340974503;
c[14] = 0.733790062453226804726171131369528;
c[15] = 0.697850494793315796932292388026640;
c[16] = 0.660061064126626961370053668149271;
c[17] = 0.620526182989242861140477556431189;
c[18] = 0.579345235826361691756024932172540;
c[19] = 0.536624148142019899264169793311073;
c[20] = 0.492480467861778574993693061207709;
c[21] = 0.447033769538089176780609900322854;
c[22] = 0.400401254830394392535476211542661;
c[23] = 0.352704725530878113471037207089374;
c[24] = 0.304073202273625077372677107199257;
c[25] = 0.254636926167889846439805129817805;
c[26] = 0.204525116682309891438957671002025;
c[27] = 0.153869913608583546963794672743256;
c[28] = 0.102806937966737030147096751318001;
c[29] = 0.051471842555317695833025213166723;
c[30] = 0.000000000000000000000000000000000;
//Kronrod katsayilari
xk[0] = 0.001389013698677007624551591226760;
xk[1] = 0.003890461127099884051267201844516;
xk[2] = 0.006630703915931292173319826369750;
xk[3] = 0.009273279659517763428441146892024;
xk[4] = 0.011823015253496341742232898853251;
xk[5] = 0.014369729507045804812451432443580;
xk[6] = 0.016920889189053272627572289420322;
xk[7] = 0.019414141193942381173408951050128;
xk[8] = 0.021828035821609192297167485738339;
xk[9] = 0.024191162078080601365686370725232;
xk[10] = 0.026509954882333101610601709335075;
xk[11] = 0.028754048765041292843978785354334;
xk[12] = 0.030907257562387762472884252943092;
xk[13] = 0.032981447057483726031814191016854;
xk[14] = 0.034979338028060024137499670731468;
xk[15] = 0.036882364651821229223911065617136;
xk[16] = 0.038678945624727592950348651532281;
xk[17] = 0.040374538951535959111995279752468;
xk[18] = 0.041969810215164246147147541285970;
xk[19] = 0.043452539701356069316831728117073;
xk[20] = 0.044814800133162663192355551616723;
xk[21] = 0.046059238271006988116271735559374;
xk[22] = 0.047185546569299153945261478181099;
xk[23] = 0.048185861757087129140779492298305;
xk[24] = 0.049055434555029778887528165367238;
xk[25] = 0.049795683427074206357811569379942;
xk[26] = 0.050405921402782346840893085653585;
xk[27] = 0.050881795898749606492297473049805;
xk[28] = 0.051221547849258772170656282604944;
xk[29] = 0.051426128537459025933862879215781;
xk[30] = 0.051494729429451567558340433647099;
for(i=nn-1; i>=nn/2; i--)
{c[i] = -c[nn-1-i];}
for(i=nn-1; i>=nn/2; i--)
{ xk[i] = xk[nn-1-i]; }
```

```
for(i=ng-1; i>=0; i--)
{ xg[nn-2-2*i] = xg[i];
  xg[1+2*i] = xg[i];
}
for(i=0; i<=nn/2; i++)
{ xg[2*i] = 0; }
k1 = 0.5*(b-a);
k2 = 0.5*(b+a);
intg = 0;
intk = 0;
for(i=0; i<=nn-1; i++)
{ v = func(k1*c[i]+k2);
  intk = intk+v*xk[i];
  if( i%2==1 )
  {intg = intg+v*xg[i];}
}
intk = intk*(b-a)*0.5;
intg = intg*(b-a)*0.5;
A[0][0] = Math.abs(intg-intk);
A[0][1] = intk;
A[0][2] = a;
A[0][3] = b;
totalerror = A[0][0];
if( totalerror<eps )
{
   sonuc = true;
   integral = intk;
   Aused = 1;
   return integral;
}
Aused = 1;
for(h=1; h<=n-1; h++)
{
   Aused = h+1;
   enter(A, h, Aw);
   totalerror = totalerror-A[h-1][0];
   ta = A[h-1][2];
   tb = A[h-1][3];
   A[h-1][2] = ta;
   A[h-1][3] = 0.5*(ta+tb);
   A[h][2] = 0.5*(ta+tb);
   A[h][3] = tb;
   for(j=h-1; j<=h; j++)
   {
      k1 = 0.5*(A[j][3]-A[j][2]);
      k2 = 0.5*(A[j][3]+A[j][2]);
      intg = 0;
      intk = 0;
      for(i=0; i<=nn-1; i++)
      {   v = func(k1*c[i]+k2);
         intk = intk+v*xk[i];
         if( i%2==1 )
         { intg = intg+v*xg[i];}
      }
      intk = intk*(A[j][3]-A[j][2])*0.5;
      intg = intg*(A[j][3]-A[j][2])*0.5;
      A[j][0] = Math.abs(intg-intk);
      A[j][1] = intk;
      totalerror = totalerror+A[j][0];
   }
   leave(A, h-1, Aw);
   leave(A, h, Aw);
   if( totalerror<eps )
   {break;}
}
sonuc = totalerror<eps;
integral = 0;
for(j=0; j<=Aused-1; j++)
{ integral = integral+A[j][1];}
return integral;
```

```
    }

    default void enter(double A[][],int n,int Awidth)
    {
       int i = 0;
       int p = 0;
       double t = 0;
       int maxcp = 0;
       if( n==1 )
       { return; }
       for(i=0; i<=Awidth-1; i++)
       {   t = A[n-1][i];
          A[n-1][i] = A[0][i];
          A[0][i] = t;
       }//end of for
       p = 0;
       while( 2*p+1<n-1 )
       {   maxcp = 2*p+1;
          if( 2*p+2<n-1 )
          {   if( A[2*p+2][0]>A[2*p+1][0] )
             { maxcp = 2*p+2; }
          }
          if( A[p][0]<A[maxcp][0] )
          {   for(i=0; i<=Awidth-1; i++)
             {   t = A[p][i];
                A[p][i] = A[maxcp][i];
                A[maxcp][i] = t;
             }
             p = maxcp;
          }
          else
          { break; }
       } //end of while
    }
    default double adaptive_gauss_legendre_integral(double x1,double x2)
{ double eps=1e-15;
  int MAXITER=100;
int n=1;
double ans1=integral(x1,x2,n);
double ans2;
do
{ans2=ans1;n++;ans1=integral(x1,x2,n);}
while(Math.abs(ans2-ans1)>eps && n<MAXITER);
if(n==MAXITER) System.out.println("istenilen hata seviyesi tutturulamadi n="+n );
return ans2;
}

    default void leave(double A[][],int n,int Awidth)
    {
       int i = 0;
       int p = 0;
       double t = 0;
       int kk = 0;
       if( n==0 )
       {return;}
       p = n;
       while( p!=0 )
       {   kk = (p-1)/2;
          if( A[p][0]>A[kk][0] )
          { for(i=0; i<=Awidth-1; i++)
             { t = A[p][i];
               A[p][i] = A[kk][i];
               A[kk][i] = t;
             }
             p = kk;
          }
          else
          {break;}
       }
    }
```

```
// Root finding method
default double bisection(double a,double b,double eps)
  {double b1=1.1*b;
   double r=(a+b)/2.0;
   int nmax=100;
   int i=1;
   while(Math.abs(func(r))>eps && i<nmax)
    {if(func(a)*func(r)<0) b=r;
     else a=r;
     r=(a+b)/2.0;
     i++;
    }
   //if(i>=nmax) r=bisection(a,b1);
   return r;
  }
default double bisection(double a,double b)
  {
   double eps=1.0e-10;
   return bisection(a,b,eps);
  }
  //Root finding method secant(Newton-Raphson)
default double secant(double x)
{ double eps=1.0e-20;
  double y=1.0;
  double dy=0.0;
  int miter=100;
  int i=0;
  while(abs(y)>eps && i<miter)
  {y=func(x);
   dy=dfunc(x);
   x=x-y/dy;
   i++;
  }
  if(i>=miter)
    System.out.println("i="+i+"results may not be valid");
  return x;
}
//Root finding methods: Illinois
default double illinois(double xa,double xu,double esi)
{
//Modified False position root finding: Illinois method
double test;
double p=0;
double es,ea;
double fa,fp,fu;
int maxit=100,iter=0;
es=esi;
ea=1.1*es;
double xold;
int ia=0,iu=0;
fa=func(xa);
fu=func(xu);
p=xu;
double xpold;
while(iter<maxit && ea>es)
{
xpold=p;
p=xu-fu*(xa-xu)/(fa-fu);
fp=func(p);
if(signum(fp)!=signum(fu))
{xa=xu;fa=fu;}
else
{fa=fa/2.0;}
xu=p;fu=fp;
iter++;
if(p!=0) ea=abs((p-xpold)/p);

}
if(iter>=maxit) System.out.println("Maximum number of iteration is exceeded ");
```

```
return p;
}
//Root finding method: Newton-Raphson - Bisection combined
default double newton_bisection(double x1, double x2)
{
double xacc=1.0e-10; //accuracy
double h=0.0001; //derivative function step
int MAXIT=100;  //Maximum number of iterations
int j;
double dfr,dx,dxold,fr,fh,fl;
double temp,b,a,r;
fl=func(x1);
fh=func(x2);
//Check if a root is existed in the given region
if ((fl > 0.0 && fh > 0.0) || (fl < 0.0 && fh < 0.0))
System.out.println("There are no root in the given region or double root?");
if (fl == 0.0) return x1;
if (fh == 0.0) return x2;
if (fl < 0.0)   { a=x1;b=x2;}
else            { b=x1;a=x2;}
r=0.5*(x1+x2); //midpoint value
dxold=Math.abs(x2-x1);
dx=dxold;
fr=func(r);   //function value at midpoint
dfr=dfunc(r); //derivative of function value at midpoint
for (j=1;j<=MAXIT;j++)
{
if ((((r-b)*dfr-fr)*((r-a)*dfr-fr) > 0.0) || (Math.abs(2.0*fr) > Math.abs(dxold*dfr)))
   { //bisection step
                 dxold=dx;
     dx=0.5*(b-a);
     r=a+dx;
     if (a == r) return r; //solution!!!!!
   }
else{ //Newton-Raphson (Secant) step
                 dxold=dx;
     dx=fr/dfr;
     temp=r;
     r -= dx;
     if (temp == r) return r;  //solution!!!!!
   }
if (Math.abs(dx) < xacc) return r; //solution!!!!!
fr=func(r);
dfr=dfunc(r);
if (fr < 0.0)
a=r;
else
b=r;
}
System.out.println("Maximum number of iterations are exceeded");
return 0.0; //program should never reach here
          //dummy return
}
//Root finding method: brent
default double brent(double a,double b,double esi)
{
double test;
double p=0;
double es,ea;
double f1,f2,f3,fp;
int maxit=500,iter=0;
double tol=1.0e-15;
es=esi;
double x1=a;f1=func(x1);
double x2=b;f2=func(x2);
double x3=(x1+x2)/2.0;f3=func(x3);
if(f1==0) return x1;
else if(f2==0) return x2;
else if(f3==0) return x3;
if(f1*f2>0) System.out.println("No root is existed in the given region");
```

```
p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
fp=func(p);
ea=Math.abs(x3-p);
while((ea>es)&&(iter<maxit))
{           if(Math.abs(f3)<tol) return x3;
   if((p<x1) && (p>x2))
    {p=(x1+x2)/2.0;
     if(p>x3) {x1=x3;f1=f3;x3=p;f3=fp;}
     else if(p<x3) {x2=x3;f2=f3;x3=p;f3=fp;}
    }
   else
    {
   if(p>x3) {x1=x3;f1=f3;x3=p;f3=fp;}
   else if(p<x3) {x2=x3;f2=f3;x3=p;f3=fp;}
   p=-(f2*f3*x1*(f2-f3)+f3*f1*x2*(f3-f1)+f1*f2*x3*(f1-f2))/((f1-f2)*(f2-f3)*(f3-f1));
   fp=func(p);
   ea=Math.abs(x3-p);
    }
   System.out.println("i="+iter+"p="+p+"f="+func(p));
   iter++;
}
if(iter>=maxit) JOptionPane.showMessageDialog(null,"Warning : MAximum number of iteration is exceeded \n"+
   " result may not be valid","MAXIMUM ITERATION NUMBER WARNING",JOptionPane.WARNING_MESSAGE);
return p;
}
//enlarge root finding section if root is not available
default double enlarge(double x0,double dx)
{  //enlarge region untill a root is existed
            double x1=x0;
            double x2=x1+dx;
            int NTRY=200;
            double a[]=new double[2];
            double FACTOR=1.001;
            int j;
            double f1,f2;
            if (x1 == x2) System.out.println("input variable limits are wrong!");
            f1=func(x1);
            f2=func(x2);
            for (j=1;j<=NTRY;j++)
            { if (f1*f2 < 0.0) {break;}
              else {x2+=dx;f2=func(x2);}
            }
            return x2;
}
//Find sections where roots is existed
default double[][] root_limits(double a,double b)
{ double dx1=3.315431e-2;
  double x1=a;
  double x2=a*(1.0+dx1);
  double f1=func(x1);
  double f2=func(x2);
  double c[][]=new double[300][2];
  int n=0; //array size counter
  while(x2<b)
  {
             if(f1*f2<=0)
   { c[n][0]=x1;
     c[n][1]=x2;
     n++;
     x1=x2+dx1;
     x2=x1+dx1;
     f1=func(x1);
     f2=func(x2);
   }
   else
   { x2=x2+dx1;
     f2=func(x2);
   }
  }
  //Array size correction
  double e[][]=new double[n][2];
  for(int i=0;i<n;i++)
  {e[i]=c[i];}
  return e;
}
//Find all roots by using secant method
default double[] roots_secant(double a,double b)
{double c[][]=root_limits(a,b);
 int n=c.length;
 double r=0.0;
 double x[]=new double[n];
 for(int i=0;i<n;i++)
 {   r=c[i][1];
     x[i]=secant(r);}
 return x;
}
//Find all roots by using bisection methods
default double[] roots_bisection(double a,double b)
{double c[][]=root_limits(a,b);
 int n=c.length;
 double x[]=new double[n];
 for(int i=0;i<n;i++)
 {x[i]=bisection(c[i][0],c[i][1]);}
 return x;
}
//Find all roots by using Bisection-Secant combined methods
default double[] roots_bisection_secant(double a,double b)
{double c[][]=root_limits(a,b);
 int n=c.length;
 double x[]=new double[n];
 for(int i=0;i<n;i++)
 {x[i]=bisection_secant(c[i][0],c[i][1]);}
 return x;
}
default double bisection_secant(double x1,double x2)
{ double f1=func(x1);
  double r=(x1+x2)/2.0;
  double fr=func(r);
  double eps=1.0e-20;
  int Maxiter=200;
  int i=0;
  //apply bisection 5 times
  for(i=0;i<5;i++)
  { if(f1==0) return x1;
    else if(fr==0) return r;
                else if(f1*fr<0)
    {x2=r;}
    else
    {x1=r;f1=fr;}
    r=(x1+x2)/2.0;
    fr=func(r);
    i++;
  }
  //Then apply secant
  while(Math.abs(f1)>eps && i<Maxiter)
  { r=r-f1/dfunc(r);
                f1=func(r);
                i++;
  }
  if(i>=Maxiter) System.out.println("Maximum iteration is exceeded");
  return r;
}
default double newton_opt(double x)
{
double eps=1e-4;
double f1=dfunc(x);
int i=0;
int nmax=100;
while(Math.abs(f1)>eps && i<nmax)
{x=x-dfunc(x)/dfunc2(x);
f1=dfunc(x);
 i++;
}
return x;
}
default double golden_opt(double a,double b)
{
//  find the minimum of the function
// note  maximum f(x) = minimum (-f(x))
   double epsilon=1e-10;
   double delta=1.0e-10;
    double r1 = (Math.sqrt(5.0)-1.0)/2.0; // golden ratio
    double r2 = r1*r1;
    double h = b - a;
    double ya = func(a);
    double yb = func(b);
    double c = a + r2*h;
    double d = a + r1*h;
    double yc = func(c);
    double yd = func(d);
    int k = 1;
    double dp,dy,p,yp;
    while ((Math.abs(yb-ya)>epsilon) || (h>delta))
    {
      k++;
      if (yc<yd)
        {
        b = d;
        yb = yd;
        d = c;
        yd = yc;
        h = b - a;
        c = a + r2 * h;
        yc = func(c);
        }
      else
       {
         a = c;
         ya = yc;
         c = d;
         yc = yd;
         h = b - a;
         d = a + r1 * h;
         yd = func(d);
       }//end of if
    }//end of while
    dp = Math.abs(b-a);
    dy = Math.abs(yb-ya);
    p = a;
    yp = ya;
    if (yb<ya)
    {
      p = b;
      yp = yb;
    }
    return p;
}
default double quadratic_poly_opt(double x0,double x1,double x2)
{  double epsilon=1.0e-10;
   double delta=1.0e-10;
   int maxit=100;
   double f0 = func(x0);
   double f1 = func(x1);
   double f2 = func(x2);
   double f3 = f2;
   double x3 = 0;
   double h = x1 - x0;
   double k=1;
   double dd=Math.abs(f1-f0);
   while ((dd >epsilon) || (h>delta))
    { k++;
      x3=(f0*(x1*x1-x2*x2)+f1*(x2*x2-x0*x0)+f2*(x0*x0-x1*x1))/
            (2*f0*(x1-x2)+2.0*f1*(x2-x0)+2.0*f2*(x0-x1));
      f3=func(x3);
      if(x3 >= x0 && x3<x1)
        {x2=x1;f2=f1;x1=x3;f1=f3;}
      else if(x3 >= x1 && x3 <x2)
        {x0=x1;f0=f1;x1=x3;f1=f3; }
      else if(x3 >x2)
        {x0=x1;f0=f1;x1=x2;f1=f2;x2=x3;f2=f3; }
      else if(x3 < x0)
        {x0=x3;f0=f3;x1=x0;f1=f0;x2=x1;f2=f1; }
      dd=Math.abs(f1-f0);
      h=Math.abs(x1-x0);
      if(k>maxit) break;
    }//end of while
    return x3;
}
default double quadratic_poly_opt(double x0,double x2)
{ double x1=x0+(x2-x0)/(2.0+0.01*Math.random());
   return quadratic_poly_opt(x0,x1,x2);
}
default double cubic_search_opt(double x0,double x1)
{
int nmax=500;
double tolerance=1.0e-15;
double f0,f1,df0,df1,fxs,dfxs;
double h=0.00001;
double xs;
int k=0;
double X,F,G,H,a,b,c;
 f0=func(x0);
 f1=func(x1);
 df0=dfunc(x0,h);
 df1=dfunc(x1,h);
while(Math.abs(x0-x1)>tolerance && k<nmax)
{ k++;
 X=1.0/(x1-x0);
 F=X*(f1-f0);
 G=X*(df1-df0);
 H=X*(F-df0);
 a=X*(G-2.0*H);
 b=3.0*H-G;
 c=df0;
 xs=x0+(Math.sqrt(b*b-3.0*a*c)-b)/(3.0*a);
 fxs=func(xs);
 dfxs=dfunc(xs,h);
 if(dfxs > 0 ) {x1=xs;f1=fxs;df1=dfxs;}
 else         x0=xs;f0=fxs;df0=dfxs;
}
return (x0+x1)/2.0;
}
default double SIGN(double a,double b) {return (b > 0.0 ? Math.abs(a) : -Math.abs(a));}
default double[] brent_opt(double ax,double bx,double cx)
{
// ax,bx,cx is three initial guesses (bx should be located between ax and cx)
// tol tolerance
double tol=1e-10;
int ITMAX=100;
double CGOLD=(3.0-Math.sqrt(5))/2.0; //golden ratio
double ZEPS=1.0e-10;
double xmin;
double aa[]=new double[2];
                int iter;
                double a,b,d,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm;
                double e=0.0;
                d=0.0;
                a=((ax < cx) ? ax : cx);
                b=((ax > cx) ? ax : cx);
                x=w=v=bx;
		fw=fv=fx=func(x);
		for (iter=1;iter<=ITMAX;iter++) {
			xm=0.5*(a+b);
			tol2=2.0*(tol1=tol*Math.abs(x)+ZEPS);
			if (Math.abs(x-xm) <= (tol2-0.5*(b-a))) {
				xmin=x;
	    aa[0]=xmin;
	    aa[1]=fx;
	    return aa;
			}
			if (Math.abs(e) > tol1) {
				r=(x-w)*(fx-fv);
				q=(x-v)*(fx-fw);
				p=(x-v)*q-(x-w)*r;
				q=2.0*(q-r);
				if (q > 0.0) p = -p;
				q=Math.abs(q);
				etemp=e;
				e=d;
				if (Math.abs(p) >= Math.abs(0.5*q*etemp) || p <= q*(a-x) || p >= q*(b-x))
					d=CGOLD*(e=(x >= xm ? a-x : b-x));
				else {
					d=p/q;
					u=x+d;
					if (u-a < tol2 || b-u < tol2)
						d=SIGN(tol1,xm-x);
				}
			} else {
				d=CGOLD*(e=(x >= xm ? a-x : b-x));
			}
			u=(Math.abs(d) >= tol1 ? x+d : x+SIGN(tol1,d));
			fu=func(u);
			if (fu <= fx) {
				if (u >= x) a=x; else b=x;
				{v=w;w=x;x=u;}
				{fv=fw;fw=fx;fx=fu;}
			} else {
				if (u < x) a=u; else b=u;
				if (fu <= fw || w == x) {
					v=w;
					w=u;
					fv=fw;
					fw=fu;
				} else if (fu <= fv || v == x || v == w) {
					v=u;
					fv=fu;
				}
			}
		}
		System.out.println("maximum number of iterations are exceeded in BRENT optimization");
		xmin=x;
		aa[0]=xmin;
		aa[1]=fx;
		return aa;
}

default double[] brent_opt(double ax,double cx)
{ double bx=ax+(cx-ax)/(2.0+0.01*Math.random());
   return brent_opt(ax,bx,cx);
}

default double dbrent_opt(double ax,double bx,double cx)
{
// ax,bx,cx is three initial guesses (bx should be located between ax and cx)
// f : function (defined with f_x abstract class)
// tol tolerance
double tol=1.0e-10;
int ITMAX=100;
double CGOLD=(3.0-Math.sqrt(5))/2.0; //golden ratio
double ZEPS=1.0e-10;
```

```
int iter;
boolean ok1,ok2;
double a,b,d,d1,d2,du,dv,dw,dx,e=0.0;
d=0.0;
double fu,fv,fw,fx,olde,tol1,tol2,u,u1,u2,v,w,x,xm;
double xmin=bx;
a=(ax < cx ? ax : cx);
b=(ax > cx ? ax : cx);
x=w=v=bx;
fw=fv=fx=func(x);
dw=dv=dx=dfunc(x);
for (iter=1;iter<=ITMAX;iter++) {
        xm=0.5*(a+b);
        tol1=tol*Math.abs(x)+ZEPS;
        tol2=2.0*tol1;
        if (Math.abs(x-xm) <= (tol2-0.5*(b-a))) {
                xmin=x;
                return xmin;
        }
        if (Math.abs(e) > tol1) {
                d1=2.0*(b-a);
                d2=d1;
                if (dw != dx)  d1=(w-x)*dx/(dx-dw);
                if (dv != dx)  d2=(v-x)*dx/(dx-dv);
                u1=x+d1;
                u2=x+d2;
                ok1 = (a-u1)*(u1-b) > 0.0 && dx*d1 <= 0.0;
                ok2 = (a-u2)*(u2-b) > 0.0 && dx*d2 <= 0.0;
                olde=e;
                e=d;
                if (ok1 || ok2) {
                        if (ok1 && ok2)
                                d=(Math.abs(d1) < Math.abs(d2) ? d1 : d2);
                        else if (ok1)
                                d=d1;
                        else
                                d=d2;
                        if (Math.abs(d) <= Math.abs(0.5*olde)) {
                                u=x+d;
                                if (u-a < tol2 || b-u < tol2)
                                        d=SIGN(tol1,xm-x);
                        } else {
                                d=0.5*(e=(dx >= 0.0 ? a-x : b-x));
                        }
                } else {
                        d=0.5*(e=(dx >= 0.0 ? a-x : b-x));
                }
        } else {
                d=0.5*(e=(dx >= 0.0 ? a-x : b-x));
        }
        if (Math.abs(d) >= tol1) {
                u=x+d;
                fu=func(u);
        } else {
                u=x+SIGN(tol1,d);
                fu=func(u);
                if (fu > fx) {
                        xmin=x;
                        return xmin;
                }
        }
        du=dfunc(u);
        if (fu <= fx) {
                if (u >= x) a=x; else b=x;
                //MOV3(v,fv,dv, w,fw,dw)
                {v=w;fv=fw;dv=dw;}
                //MOV3(w,fw,dw, x,fx,dx)
                {w=x;fw=fx;dw=dx;}
                //MOV3(x,fx,dx, u,fu,du)
                {x=u;fx=fu;dx=du;}
```

```
                    } else {
                              if (u < x) a=u; else b=u;
                              if (fu <= fw || w == x) {
                                        //MOV3(v,fv,dv, w,fw,dw)
                                        {v=w;fv=fw;dv=dw;}
                                        //MOV3(w,fw,dw, u,fu,du)
                                        {w=u;fw=fu;dw=du;}
                              } else if (fu < fv || v == x || v == w) {
                                        //MOV3(v,fv,dv, u,fu,du)
                                        {v=u;fv=fu;dv=du;}
                              }
                    }
          }
          return xmin;
}
default double[] dbrent_opt(double ax,double cx)
{ double bx=ax+(cx-ax)/(2.0+0.01*Math.random());
  return brent_opt(ax,bx,cx);
}


}
```

**plotP**

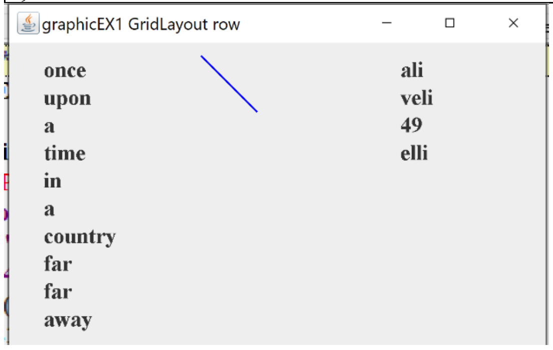
```
 import javax.swing.*;
import java.awt.Graphics;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class plotP extends JPanel
{
 int x[];
 int y[];
 double xmin;
 double xmax;
 int n;


 public plotP(if_x f,double xmini,double xmaxi,int width,int height)
 {
  // height : height of the plot window;
  // width  : width of the plot window
  // xmin minimum x value
  // x maximum x value
  xmin=xmini;
  xmax=xmaxi;
  n=100;
  x=new int[n];
  y=new int[n];
  double xd[]=new double[n];
  double yd[]=new double[n];
  double ymin=1.0e60;
  double ymax=-1.0e60;
  for(int i=0;i<n;i++)
  {xd[i] = (xmax-xmin)*(double)i/(double)n;
  yd[i] =  f.func(xd[i]);
  if(yd[i]<ymin) ymin=yd[i];
  if(yd[i]>ymax) ymax=yd[i];
  }
 for(int i=0;i<n;i++)
 {x[i]=(int)(0.8*width*xd[i]/(xmax-xmin));
  y[i]=height/3+(int)(0.6*height*yd[i]/(ymax-ymin));
 }
 }
 public void paintComponent(Graphics g)
 {
 Graphics2D g2=(Graphics2D)g;
 for(int i=0;i<x.length;i++)
  {g2.drawString("x",x[i],y[i]);}
 }
}
```

plot2P

```
import javax.swing.*;
import java.awt.Graphics;
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;


public class plot2P extends JPanel
{
 int x[];
 int y[];
 double xmin;
 double xmax;
 int n;
 fb f;
 int width;
 int height;
 public plot2P(if_x f,double xmini,double xmaxi,int widthi,int heighti)
 {
  // height : height of the plot window;
  // width  : width of the plot window
  // xmin minimum x value
  // x maximum x value
  xmin=xmini;
  xmax=xmaxi;
  width=widthi;
  height=heighti;
  n=100;
  x=new int[n];
  y=new int[n];
  double xd[]=new double[n];
  double yd[]=new double[n];
  double ymin=1.0e60;
  double ymax=-1.0e60;
  for(int i=0;i<n;i++)
   {xd[i] = (xmax-xmin)*(double)i/(double)n;
  yd[i] =  f.func(xd[i]);
  if(yd[i]<ymin) ymin=yd[i];
  if(yd[i]>ymax) ymax=yd[i];
   }
 for(int i=0;i<n;i++)
 {x[i]=(int)(0.8*width*xd[i]/(xmax-xmin));
  y[i]=height/3+(int)(0.6*height*yd[i]/(ymax-ymin));
 }
 }

 public void paintComponent(Graphics g)
 {
 Graphics2D g2=(Graphics2D)g;
 for(int i=1;i<x.length;i++)
  { Line2D l=new Line2D.Double(x[i-1],y[i-1],x[i],y[i]);
   g2.draw(l);
  }
 Rectangle2D r=new Rectangle2D.Double(0,0,height,width);
 g2.draw(r);
 }

}
```

EX1

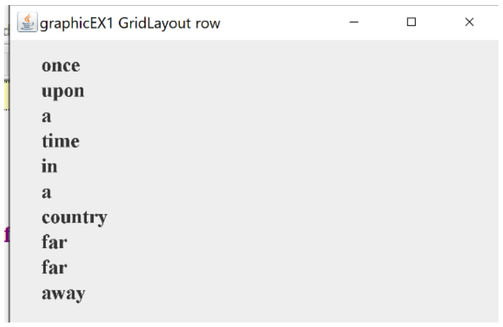
```
 import  javax.swing.*;
public class graphicEX2
{
public static void main(String arg[])
{
   String s="once upon a time in a country far far away";
   String s1[]=s.split("\\s+");
   String s2="ali veli 49 elli";
   String s3[]=s2.split("\\s+");
   JPanel jp=new writeP(s1);
   FrameGraphic.plot("graphicEX1 GridLayout row",jp);
 }
}
```



```
once                    ali
upon                    veli
a                       49
time                    elli
in
a
country
far
far
away
```

EX2

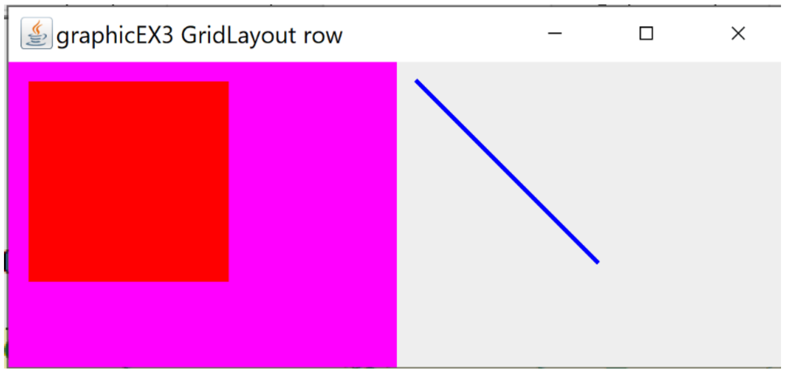
```
import  javax.swing.*;
public class graphicEX1
{
public static void main(String arg[])
{  JPanel jp[]=new JPanel[3];
   String s="once upon a time in a country far far away";
   String s1[]=s.split("\\s+");
   String s2="ali veli 49 elli";
   String s3[]=s2.split("\\s+");
   jp[0]=new writeP(s1);
   jp[1]=new lineP(10,10, 50,50,1);
   jp[2]=new writeP(s3);
   //FrameGraphic.plot("graphicEX1 GridLayout column",jp,"GridLayout_column");
   FrameGraphic.plot("graphicEX1 GridLayout row",jp,"GridLayout_row");
   //FrameGraphic.plot("graphicEX1 JTappedPane",jp,"JTabbedPane");
 }

}
```



```
once
upon
a
time
in
a
country
far
far
away
```

EX3: fill  a draw a rectangle between pixel (10,10) to pixel (100,100) and a line between pixel (10,10) to pixel (100,100)

```
import  javax.swing.*;
public class graphicEX3
{
public static void main(String arg[])
{  JPanel jp[]=new JPanel[2];
   jp[0]=new rectangleP(10,10,100,100,1,false);
   jp[1]=new lineP(10,10, 100,100,2);
   //FrameGraphic.plot("graphicEX3 GridLayout column",jp,"GridLayout_column");
   FrameGraphic.plot("graphicEX3 GridLayout row",jp,"GridLayout_row");
   //FrameGraphic.plot("graphicEX3 JTappedPane",jp,"JTabbedPane");
  }
```

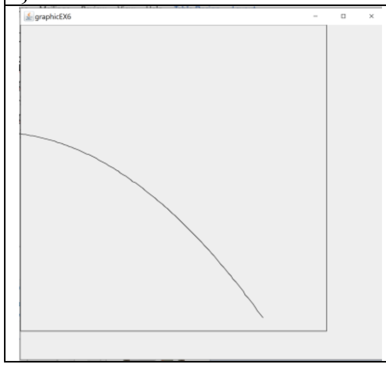


EX4 draw a rectangle between pixel (10,10) to pixel (100,100)

EX 5 fill an ellipse between pixel (10,10) to pixel (100,100)

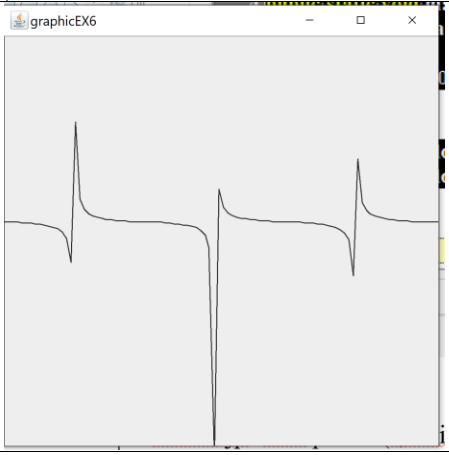
EX 6 draw function f(x)=x*x-2*x+5; between pixels 0 to 10

```
import  javax.swing.*;
public class graphicEX6
{
public static void main(String arg[])
{  if_x f=x->x*x-2.0*x+5.0;
   double xmin=10.0;
   double xmax=00.0;
   int width=500;
   int height=500;
   JPanel jp=new plot2P(f,xmin,xmax,width,height);
   FrameGraphic.plot("graphicEX6 ",jp);
 }

}
```



EX7 draw a function f(x)=tan(x) between pixels 0 to 10

```
import  javax.swing.*;
public class graphicEX7
{
public static void main(String arg[])
{  if_x f=x->Math.tan(x);
   double xmin=0.0;
   double xmax=10.0;
   int width=500;
   int height=500;
   JPanel jp=new plot2P(f,xmin,xmax,width,height);
   FrameGraphic.plot("graphicEX6 ",jp);
 }
}
```



**HOMEWORK PROBLEMS**

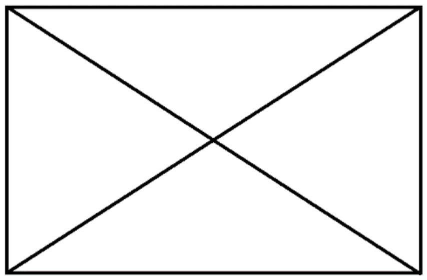
**HW1**

Draw a rectangle between pixel pixel (10,10) to pixel (100,100) and a function f(x)=sin(x) in thesame graphic output

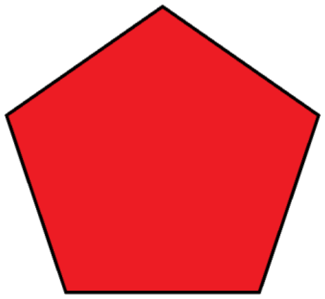
**HW2 by using  general path draw the following data**

| | |
|---|---|
| 0 | 0.765198 |
| 1 | 0.440051 |
| 2 | 0.114903 |
| 3 | 0.019563 |
| 4 | 0.002477 |
| 5 | 0.00025 |
| 6 | 2.09E-05 |

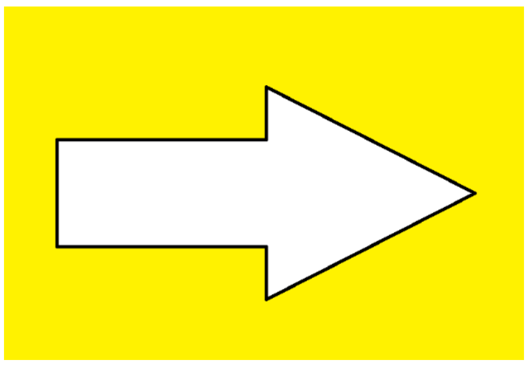
**HW3 draw the folowing shape**



**HW4 fill the folowing shape**



**HW 5 draw the following shape**



**HW6**

Study class plot2P. Plot fonction  y=Math.sin(x)*(0.1*Math.random() )  by using class plot2P between x=0 and x=π .