Dot NET provides device interface classes that can be used to implement graphics effectively. It makes use of the following namespaces.

1. System.Drawing
   It provides several structures, classes and methods to implements graphics.
2. System.Drawing.Drawing2D
   It provides more advanced functionality in implementing the graphics.

The following are the predefined members of the System.Drawing namespaces.

Point(x,y)	An ordered pair of integer x & y values that identifies a point on the graphic object.
PointF(x,y)	An ordered pair of floating point x & y values that identifies a point over the graphic object.
Rectangle(x,y,width,height)	A predefined structure that contains integer x, y, width & height that identifies a rectangle over the graphic object.
RectangleF(x,y,width,height)	A predefined structure that contains floating pont x, y, width & height that identifies a rectangle over the object
Color	A Predefined structure that contains color constants. It also contains the following method FromArgb(): It returns random colors at different proportions of red, green and blue.

Pen

A predefined class that is used to draw the graphic object.

Syntax:
Pen <objectofpen>=new Pen(color)
Ex:
Pen p=new Pen(Color.Red);

Note: Pen contains overloaded constructors.

The pen class uses the following methods to draw different graphic objects.

DrawArc(pen,x,y,width,height,startingangle,sweepangle)	Draws a arc.
DrawBezier(pen,x1,y1,x2,y2,x3,y3,x4,y4)	Draws a Bezier.
DrawBeziers(pen,points[]	Draws multiple Beziers.
DrawEclipse(pen,x,y,width,height)	Draws an eclipse.
DrawPie(pen,x, y, width,height,startingangle,sweeangle)	Draws a pie.
DrawPolygon(pen, point[])	Draws a polygon.
DrawRectangle(pen,x,y,width,height)	Draws a rectangle.
DrawRectangles(pen, Rects[])	Draws multiple rectangles.
DrawString(Brush, Font, text)	Draws a string.

Brush

A predefined class that is used to fill the graphic object. It is further subdivided into the following classes.
The Brush class uses the following methods.

FillEclipse(Brush,x,y,width,height)	Fills the eclipse.
FillPolygon(Brush, point[])	Fills polygon.
FillPie(Brush,x,y,w,h,startangle,sweepangle)	Fills Pie.
FillRectangle(Brush,x,y,width,height)	Fills Rectangle.

1. **SolidBrush**
   This is used to fill the graphic object with a solid color. It is a constructor class.
   Syntax: SolidBrush <object> =new SolidBrush(color);
   Ex: SolidBrush sb=new SolidBrush(Color.Blue);

2. **TextureBrush**
   This brush is used to fill the graphic object with a bitmap image.
   Syntax: TextureBrush <object> =new TextureBrush (Bitmap);
   Ex: TextureBrush tb=new TextureBrush (path);

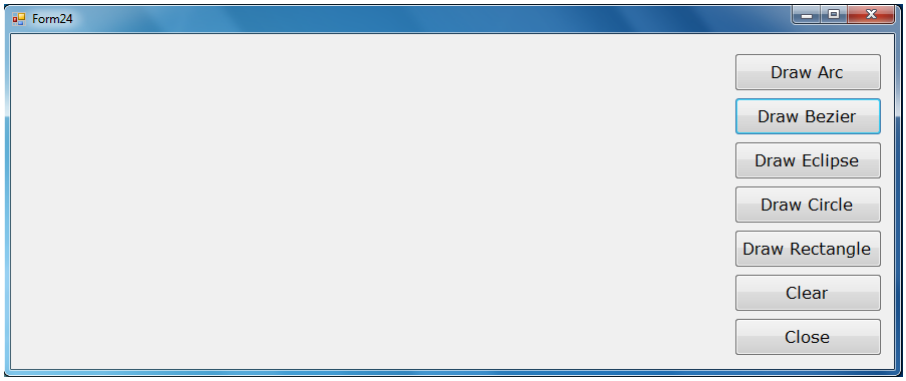
3. **HatchBrush**
   This brush is used to fill the graphic object with various hatch styles.
   Syntax: HatchBrush <object> =new HatchBrush (HatchStyle, color1, color2);
   Ex: HatchBrush hb=new HatchBrush (HatchStyle.DiagonalBrick, Color.Blue, Color.Green);

4. **LinearGradientBrush**
   This brush is used to fill the graphic object with linear gradient colors.
   Syntax: LinearGradientBrush <object> =new LinearGradientBrush (point1,point2,color1,color2);
   Ex: LinearGradientBrush lgb=new LinearGradientBrush (p1, p2, Color.Green, Color.White);

Note:

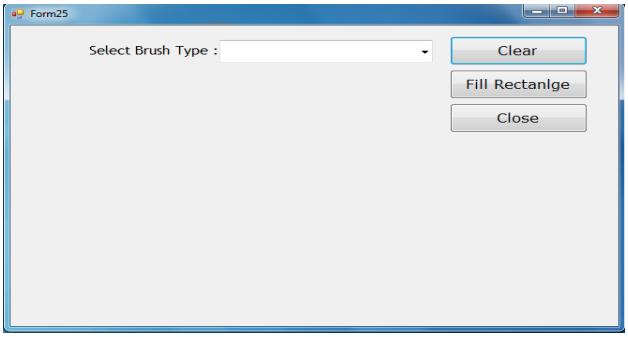
1. SolidBrush & TextureBrush are available in System.Drawing namespace.
2. HatchBrush & LinearGradientBrush are available under System.Drawing.Drawing2D namespace.

Create an interface to implement graphic method when the button is pressed.



```
1, y1, x2, y2, x3, y3, x4, y4;
        Random rnd;
        Pen p;
        Graphics g;


public void Randomize()
        {
            rnd = new Random();
            int red, green, blue, alpha;

            Random a = new Random();
            Random r = new Random();
            Random gr = new Random();
            Random b = new Random();

            alpha = a.Next(0, 255);
            red = a.Next(0, 255);
            green = a.Next(0, 255);
            blue = a.Next(0, 255);

            //p=new Pen(Color.Green);
            p=new Pen(Color.FromArgb(alpha,red,green,blue));
            p.Width=100;

            x1=rnd.Next(this.Width);
            x1=rnd.Next(this.Height);

            x2=rnd.Next(this.Width);
            x2=rnd.Next(this.Height);

            x3=rnd.Next(this.Width);
            x3=rnd.Next(this.Height);

            x4=rnd.Next(this.Width);
            x4=rnd.Next(this.Height);

            g=this.CreateGraphics();
        }
  //Draw Arc
            Randomize();
            g.DrawArc(p, x1, x2, 100, 200, 0,75);
//Draw Bezier
            Randomize();
            g.DrawBezier(p, x1, y1, x2, y2, x3, y3, x4, y4);
  //Draw Eclipse
            Randomize();
            g.DrawEllipse(p, x1, y1, 300, 200);
//Draw Circle
            Randomize();
            g.DrawEllipse(p, x1, y1, 200, 200);
//Draw Rectangle
            Randomize();
            g.DrawRectangle(p, x1, y1, 100, 200);
//Clear
            this.Invalidate();//the entire surface of the control and causes the control to be redrawn
//Close
            this.Close();
```

Create an interface to fill graphic object by selecting a brush.


Form25
Select Brush Type :
Clear
Fill Rectanlge
Close

```
SolidBrush sb;
        TextureBrush tb;
        HatchBrush hb;
        LinearGradientBrush lgb;
        Random rnd;
        int x1, x2, y1, y2;
        Graphics g;
        int red, green, blue, alpha;

public void Randomize()
        {
            rnd = new Random();

            Random a = new Random();
            Random r = new Random();
            Random gr = new Random();
            Random b = new Random();

            alpha = a.Next(0, 255);
            red = a.Next(0, 255);
            green = a.Next(0, 255);
            blue = a.Next(0, 255);

            x1=rnd.Next(this.Width);
            x1=rnd.Next(this.Height);

            x2=rnd.Next(this.Width);
            x2=rnd.Next(this.Height);

            g=this.CreateGraphics();
        }

//Load
            comboBox1.Items.Add("Solid Brush");
            comboBox1.Items.Add("Texture Brush");
            comboBox1.Items.Add("Hatch Brush");
            comboBox1.Items.Add("Linear Gradient Brush");
//clear
            this.Invalidate();
  //close
            this.Close();
//Fill Rectangle
            switch(comboBox1.SelectedIndex)
            {
                case 0:
                    {
                        Randomize();
                        sb=new SolidBrush(Color.FromArgb(alpha,red,green,blue));
                        g.FillRectangle(sb,x1,y1,100,200);
                        break;
                    }
                case 1:
                    {
                        Randomize();
                        Bitmap bmp=new Bitmap(@"D:\C# Course Content\Class #6 C#-
GDI\Stuff\Desert.jpg");
                        tb=new TextureBrush(bmp);
                         g.FillRectangle(tb,x1,y1,100,200);
                        break;
                    }
                case 2:
```

```
            {
                Randomize();
                hb=new HatchBrush(HatchStyle.DiagonalBrick,Color.Blue,Color.White);
                g.FillRectangle(hb,x1,y1,100,200);
                break;
            }
        case 3:
            {
                Randomize();
                Point p1=new Point(x1,y1);
                Point p2=new Point(x2,y2);
                lgb=new LinearGradientBrush(p1,p2,Color.Blue,Color.White);
                g.FillRectangle(lgb,x1,y1,100,200);
                break;
            }
    }
```

Form

A form is a primary source of user interface. It is the building block for the entire windows applications. All forms in a windows application are inherited from the predefined class **System.Windows.Forms.Form.**

Property	Description
Name	The property that uniquely identifies form.
BackgroundImage	Specifies the path of the image that can be set as the background for the form.
BackgroundImageLayout	Specifies the layout for displaying the background image.
Controlbox	Determines if the form should display the default window menu option when the right button of the button is pressed over the title.
Cursor	Specifies the type of cursor to be displayed over the form during the runtime.
FormBorderStyle	Specifies or controls the behavior of the form during the runtime.
Icon	Specifies the path of the icon file. (Extension .ico)
IsMDIContainer	Determines if the form is acting as container for other windows within the application. MDI- Multiple Document Interface
MaximuxBox	Determines if the maximize button is enabled or disabled. By default it is true.
MinimizeBox	Determines if the minimize button is enabled or disabled. By default it is true.
Opacity	Specifies how the form looks like during the runtime. 100% opaque. <100%-Transparent 0-More Transparent 50-Modular Transparent
Text	Sets the caption to display over the title bar.
WindowState	Specifies the state of the window during the runtime. It can be normal minimize or maximize. By default is normal.

Building a Screensaver

Create an interface to draw the line from the center of the form and convert it into a screen saver.
Step 1: Create a form and drag timer on form. Set the following properties.

Timer	Form
Enable-True	BackColor-Black
Time-100	FormBorderStyle-None
	WindowState-Maximize

Step 2:

```
static int cnt;

//Tick
            Random rnd = new Random();
            Random rnd1 = new Random();
            int a, r, g, b;
            int x1, x2, y1, y2;
            x1 = this.Width / 2;
            y1 = this.Height / 2;
            x2 = rnd1.Next(this.Width);
            y2 = rnd1.Next(this.Height);
            a = rnd.Next(0, 225);
            r = rnd.Next(0, 225);
            g = rnd.Next(0, 225);
            b = rnd.Next(0, 225);

            Graphics gr;
            Pen p = new Pen(Color.FromArgb(a, r, g, b), 10);//10 line size
            gr = this.CreateGraphics();
            gr.DrawLine(p, x1, y1, x2, y2);
            cnt++;
            if (cnt > 100)//upto 100 lines
            {
                cnt = 0;
                this.Invalidate();
            }
//keyUP
            this.Close();
//key down
            this.Close();
//key press
            this.Close();
```