using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Reflection;

namespace hw3.Bowden

{

public partial class frmThreadGraphics : Form

{

//class level variables

private const int shapeSize = 16;

private string shapeType = "Circle";

public volatile System.Windows.Forms.Panel panelDraw;

private volatile Graphics graphics;

public static Color shapeColor = Color.Blue;

public Color ThreadColor;

public static int threadCount = 0;

private static bool blnRun = true;

//hashtable to keep track of threads

private Hashtable threadHolder = new Hashtable();

//default constructor

public frmThreadGraphics()

{

InitializeComponent();

//populate the panel variable used for drawing

panelDraw = pnlDraw;

//create the graphics object

graphics = panelDraw.CreateGraphics();

}//end default constructor

//form load event

private void frmThreadGraphics\_Load(object sender, EventArgs e)

{

//use code to set default control behavior

this.cboShape.SelectedIndex = 0;

this.AcceptButton = btnAddShape;

//set the panel to be double buffered to reduce flickering

typeof(Panel).InvokeMember("DoubleBuffered",

BindingFlags.SetProperty | BindingFlags.Instance | BindingFlags.NonPublic,

null, panelDraw, new object[] { true });

}//end form load event

//begin new thread

private void StartThread()

{

//create shapes

Shapes shapes = new Circle(0, 0, shapeSize, shapeSize, shapeColor, GetSpeed(), panelDraw);

switch (shapeType)

{

case "Triangle":

shapes = new Triangle(0, 0, shapeSize, shapeSize, shapeColor, GetSpeed(), panelDraw);

break;

case "Rectangle":

shapes = new Rectangle(0, 0, shapeSize, shapeSize, shapeColor, GetSpeed(), panelDraw);

break;

default:

//Do nothing because a circle is our default object

break;

}

//main loop

while (true)

{

//check to make sure we're supposed to be running

if (blnRun)

{

try

{

//paint the shape object on each pass

shapes.paint(graphics);

}

catch

{

//shut down gracefully

Shutdown();

break;

}

}

}

}//end StartThread

//user has pushed the btnColor and will choose a color

private void btnColor\_Click(object sender, EventArgs e)

{

//show the color dialog

dlgColor.ShowDialog();

shapeColor = dlgColor.Color;

}//end color choice

//user pushed the btnRusume button

private void btnResume\_Click(object sender, EventArgs e)

{

//change our button text

if (btnResume.Text == "Resume")

{

//change the text to Pause

btnResume.Text = "Pause";

//resume all threads

Resume();

}

else

{

//change the text to Resume

btnResume.Text = "Resume";

//pause all threads

Pause();

}

}//end resume/pause

//AddShape\_Click event

private void btnAddShape\_Click(object sender, EventArgs e)

{

//local variables

shapeType = this.cboShape.Text;

//enable the pause/resume button

btnResume.Enabled = true;

//Check to make sure we haven't reached our maximum thread count

if(threadCount >= 50)

{

MessageBox.Show("This program is limited to a maximum of 50 threads to prevent overloading system resources.");

}

else

{

//add a new thread

Thread thread = new Thread(new ThreadStart(StartThread));

threadHolder.Add(threadCount++, thread);

thread.Name = "Thread ID: " + threadCount.ToString();

thread.IsBackground = true;

thread.Start();

this.lblThreadCount.Text = "Thread Count = " + threadCount.ToString();

}

}//end AddShape\_Click event

//pauses all threads

private void Pause()

{

//thread.suspend is deprecated. use a boolean variable to pause.

blnRun = false;

}//end pause

//resumes all threads

private void Resume()

{

//thread.resume is deprecated. use a boolean variable to resume.

blnRun = true;

}//end resume

//returns the speed the shape should move

private int GetSpeed()

{

//Invert the speed displayed on the Numeric Up/Down Control

return 105 - Convert.ToInt32(nudSpeed.Text.Trim());

}//end GetSpeed

//btnExit\_Click event

private void btnExit\_Click(object sender, EventArgs e)

{

//call the shutdown function

Shutdown();

}//end btnExit\_Click event

//shutdown function is broken out into its own method to

//prevent having to pass sender and EventArgs from an error handler

private void Shutdown()

{

//shut down all threads

foreach (Thread thread in threadHolder.Values)

{

if (thread != null && thread.IsAlive)

{

thread.Abort();

}

}

//exit gracefully

frmThreadGraphics.ActiveForm.Close();

}//end shutdown

}

}

using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Drawing;

namespace hw3.Bowden

{

public class Circle: Shapes

{

//constructor

public Circle(int LeftPosition, int TopPosition, int Width, int Height, Color ShapeColor, int Speed, System.Windows.Forms.Panel drwPanel)

{

//fill the variable values with our incoming data

leftPosition = LeftPosition;

topPosition = TopPosition;

width = Width;

height = Height;

shapeColor = ShapeColor;

speed = Speed;

DrwPanel = drwPanel;

}//end constructor

//call the paint method

public override void paint (Graphics graphics)

{

try

{

//save the old value of the circle position

previousLeftPosition = leftPosition;

previousTopPosition = topPosition;

//put this thread to sleep

Thread.Sleep(speed);

//lock thread to prevent thread from running back on itself

lock (typeof(Thread))

{

leftPosition = leftPosition + base.directionX;

topPosition = topPosition + base.directionY;

base.CheckCoordinates();

//grouped the drawing functions together like this to slightly reduce flickering

graphics.DrawEllipse(new System.Drawing.Pen(Color.White), previousLeftPosition, previousTopPosition, width, height);

graphics.DrawEllipse(new System.Drawing.Pen(shapeColor), leftPosition, topPosition, width, height);

}

}

catch

{

//force the thread to end, but only after removing the shape from the screen

graphics.Clear(Color.White);

Thread.CurrentThread.Abort();

}

}//end paint

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Drawing;

namespace hw3.Bowden

{

public class Rectangle : Shapes

{

//constructors

public Rectangle(int LeftPosition, int TopPosition, int Width, int Height, Color ShapeColor, int Speed, System.Windows.Forms.Panel drwPanel)

{

//fill the variable values with our incoming data

leftPosition = LeftPosition;

topPosition = TopPosition;

width = Width;

height = Height;

shapeColor = ShapeColor;

speed = Speed;

DrwPanel = drwPanel;

}//end constructor

//call the paint method

public override void paint (Graphics graphics)

{

try

{

//save the old value of the rectangle position

previousLeftPosition = leftPosition;

previousTopPosition = topPosition;

//put this thread to sleep

Thread.Sleep(speed);

//lock thread to prevent thread from running back on itself

lock(typeof(Thread))

{

leftPosition = leftPosition + base.directionX;

topPosition = topPosition + base.directionY;

base.CheckCoordinates();

//grouped the drawing functions together like this to slightly reduce flickering

graphics.DrawRectangle(new System.Drawing.Pen(Color.White), previousLeftPosition, previousTopPosition, width, height);

graphics.DrawRectangle(new System.Drawing.Pen(shapeColor), leftPosition, topPosition, width, height);

}

}

catch

{

//force the thread to end, but only after removing the shape from the screen

graphics.Clear(Color.White);

Thread.CurrentThread.Abort();

}

}//end paint

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Drawing;

namespace hw3.Bowden

{

public class Triangle: Shapes

{

//constructor

public Triangle(int LeftPosition, int TopPosition, int Width, int Height, Color ShapeColor, int Speed, System.Windows.Forms.Panel drwPanel)

{

//fill the variable values with our incoming data

leftPosition = LeftPosition;

topPosition = TopPosition;

width = Width;

height = Height;

shapeColor = ShapeColor;

speed = Speed;

DrwPanel = drwPanel;

}

//call the paint method

public override void paint (Graphics graphics)

{

try

{

//save the old value of the triangle position

previousLeftPosition = leftPosition;

previousTopPosition = topPosition;

//put this thread to sleep

Thread.Sleep(speed);

//lock thread to prevent thread from running back on itself

lock(typeof(Thread))

{

leftPosition = leftPosition + base.directionX;

topPosition = topPosition + base.directionY;

base.CheckCoordinates();

//draw the colored in part of the triangle

var drawingPen = new Pen(Color.White, 1);

graphics.DrawLine(drawingPen, new Point(previousLeftPosition, previousTopPosition + 16), new Point(previousLeftPosition + 8, previousTopPosition));

graphics.DrawLine(drawingPen, new Point(previousLeftPosition + 8, previousTopPosition), new Point(previousLeftPosition + 16, previousTopPosition + 16));

graphics.DrawLine(drawingPen, new Point(previousLeftPosition, previousTopPosition + 16), new Point(previousLeftPosition + 16, previousTopPosition + 16));

//draw the colored in part of the triangle

drawingPen = new Pen(shapeColor, 1);

graphics.DrawLine(drawingPen, new Point(leftPosition, topPosition + 16), new Point(leftPosition + 8, topPosition));

graphics.DrawLine(drawingPen, new Point(leftPosition + 8, topPosition), new Point(leftPosition + 16, topPosition + 16));

graphics.DrawLine(drawingPen, new Point(leftPosition, topPosition + 16), new Point(leftPosition + 16, topPosition + 16));

}

}

catch

{

//force the thread to end, but only after removing the shape from the screen

graphics.Clear(Color.White);

Thread.CurrentThread.Abort();

}

}//end paint

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

using System.Threading;

namespace hw3.Bowden

{

public abstract class Shapes

{

//class level variables

protected int leftPosition;

protected int topPosition;

protected int width;

protected int height;

protected Color shapeColor;

protected int directionY = 1;

protected int directionX = 1;

protected int speed;

protected int previousLeftPosition;

protected int previousTopPosition;

protected System.Windows.Forms.Panel DrwPanel;

//default constructor

public Shapes()

{

}

//abstract paint event

public abstract void paint(Graphics graphics);

//check to make sure we don't bounce into the edge of panel

public void CheckCoordinates()

{

if ((DrwPanel.Size.Height - 20 < topPosition) || (topPosition <= 0)) directionY = directionY \* (-1);

if ((DrwPanel.Size.Width - 20 < leftPosition) || (leftPosition <= 0)) directionX = directionX \* (-1);

}//end CheckCoordinates

}

}