

Special Topics No. 4: Nondisappearing Shapes

The DrawShape sample program from Chapter 6 does not retain the drawn shape. If there is a window that covers the area in which the drawing takes place or the drawing window is minimized and restored to its normal size, for example, the drawn shape (or portion of it in the case of the overlapping windows) gets erased. We will describe a technique to avoid this disappearance of the drawn shape.

Introduction

In this document, we will show you the standard technique of making the drawn graphics objects stay on the window. With this technique the graphics objects that are drawn on the window will remain visible. They will not be erased, for example, when the window is restored to its normal size after being minimized or when the overlapping windows that covered them are moved or closed.

To introduce this technique, we will rewrite the `DrawShape` class from Chapter 6 without using the `DrawingBoard` class. It is, of course, possible to modify the `DrawingBoard` class so the `DrawShape` requires no changes. However, the `DrawingBoard` class is implemented in Chapter 12 of the textbook, and to modify the `DrawingBoard` class properly so the drawn shapes are not erased requires too many topics that are not mentioned yet. We will therefore defer the discussion on the modification of the `DrawingBoard` class until after Chapter 12.

We still have to introduce topics that are not covered yet. However, we will focus on the core idea and keep the number of new topics to a bare minimum. We will introduce the necessary new topics in this document as they become needed.

1 The paint Method

In Chapter 2, the first sample applet we introduced, `MyFirstApplet`, contains the `paint` method where all the drawing took place. This `paint` method is the key to nondisappearing shapes. Before we modify the real `DrawShape` class, we will write a simplified version that illustrates the `paint` method and its companion method `repaint`. To distinguish the two classes, we will call the new nondisappearing-shape version of the class `DrawShape2`.

The `MiniDraw` class, a simplified version of the `DrawShape2`, allows the user to pick a color and draws a rectangle whose bounds are (100, 100, 400, 300) in the chosen color. We will reuse the same `ListBox` object `colorListBox` for allowing the user to pick a color. If the user does not select a color, then the rectangle will be drawn in black.

In order to illustrate the use of the `paint` method, we will define `MiniDraw` (and consequently `DrawShape2`) as a subclass of `MainWindow`. We can define `MiniDraw` as a direct subclass of the `Frame` class, but we want to use the behavior of `MainWindow` (a subclass of the `Frame` class) such as closing the program when the window is closed, so we will define `MiniDraw` as a subclass of `MainWindow`. The class declaration is as follows:

```
class MiniDraw extends MainWindow
{
    ...
}
```

In the constructor, we will set up the `colorListBox` as we have done in the constructor of the `DrawShape` class with one major difference. In the `DrawShape` constructor, we wrote

```
colorListBox = new ListBox( canvas, "Select Color:" );
```

where `canvas` is a `DrawingBoard` object. We do not have such an object in this program. Then, who should be the owner frame object for `colorListBox`? Any instance of the `Frame` class or its subclasses can be the owner frame object. Does that mean we have to create an instance of `MainWindow`, for example, and set this to be the owner frame object as

```
MainWindow ownerFrame = new MainWindow( );
colorListBox = new ListBox( ownerFrame,
                           "Select Color:" );
```

This will not work. For the program to work correctly, we need to make an instance of `MiniDraw`, which is a descendant class of `Frame`, to be the owner frame. But how do we refer to a `MiniDraw` object inside the `MiniDraw` constructor? We use the reserved word `this` to refer to an instance of a class from the instance methods of the class, including the constructors. Using the reserved word `this`, we create the `colorListBox` in the `MiniDraw` constructor as

```
colorListBox = new ListBox( this, "Select Color:" );
```

When you are reading a statement inside a method of a class, you can interpret the reserved word `this` as “the object which this method belongs to.” The use of the reserved word `this` is explained briefly in Chapter 5, page 203. A more complete explanation is given in Section 9.5. Keep in mind that simply creating another instance of `MiniDraw` is not a correct:

```
MiniDraw ownerFrame = new MiniDraw( );
colorListBox = new ListBox( ownerFrame,
                           "Select Color:" );
```

This code does not change anything from the code that creates an instance of `MainWindow` inside the constructor. All it does is create another instance of `MiniDraw` instead of `MainWindow`. The owner frame must be `this` object for the program to work properly. See Exercise 5.5 on page 225. You should run the wrong versions of the program and see the resulting peculiar behavior the program exhibits for yourself.

this

The top-level `start` method is defined as follows:

```
public void start( )
{
    selectColor( );
    draw( );
}
```

The `selectColor` method is essentially the same as before, so we will not explain it here. The `draw` method in `MiniDraw` is quite different from the one in `DrawShape`. The new `draw` method includes only one statement:

```
private void draw( )
{
    repaint( );
}
```

The `repaint` method is a method inherited from the ancestor superclass `Component`. You call this method whenever you want the window content to be redrawn. Whenever you change the content, such as adding a new shape, changing the color of a shape, erasing the portion of a shape, and so forth, you want the content to be redrawn.

Calling the `repaint` method eventually leads to calling the `paint` method. The `paint` method is also defined in the `Component` class, but the method is empty. We must implement the `paint` method in our class so the content will be drawn. The declaration of the `paint` method is as follows:

```
public void paint( Graphics graphics )
{
    ...
}
```

We described in Chapter 2 how the `Graphics` object is used in drawing the geometric shapes. Please refer to Table 2.1 on page 72. In Chapter 2, we discussed the drawing methods in context of applets, but the `paint` method is available to any descendant subclasses of the `Component` class, not just the `Applet` subclass. `Applet` is just one of the many descendant subclasses of the `Component` class.

Using the passed `Graphics` object `graphics` we set the pen color and draw the fixed rectangle as

```
graphics.setColor( shapeColor );
```

```
graphics.drawRect( 100, 100, 400, 300 );
```

where `shapeColor` is a `Color` object set via the `selectColor` method. The `selectColor` method is the one that uses `colorListBox` for allowing the user to pick a color. Please refer to the final listing for this part of the code as it is straightforward and does not require further explanation.

When you actually run the program at this point, you will notice that a black rectangle is already drawn when the `colorListBox` appears on the screen. What is going on? We just mentioned that the calling of `repaint` will eventually get the paint method executed. But this is not the only way that the paint method is called. When the system detects a situation in which the window content needs to be refreshed, it calls the paint method. One such situation is the opening of a window. When the window opens on the screen, the system calls the paint method, and since `shapeColor` is initialized to black, the black rectangle is drawn.

This call of the paint method by the system is the secret of the nondisappearing shape. By including the code for drawing shapes inside the paint method, we can be sure that the shapes will not disappear because whenever the content needs to be refreshed (e.g., a window is minimized and restored, an overlapping window is moved or closed, etc.), the system will call the paint method and the shapes will be drawn again. The original `DrawShape` class does not retain the drawn shape because the paint method of the `DrawingBoard` class is empty. Here's the key concept to remember:



Perform all drawings inside the paint method so the drawn shapes will not disappear.

To eliminate the drawing of a black rectangle, we use a boolean variable `canDraw`. This variable is initialized to `false`. When the shape color is selected properly, `canDraw` is set to `true`. The final paint method is as follows:

```
public void paint( Graphics graphics )
{
    if ( canDraw ) {
        graphics.setColor( shapeColor );
        graphics.drawRect( 100, 100, 400, 300 );
    }
}
```

We are now ready to list the complete `MiniDraw` class. Notice that this instantiable class is also the main class of the program. Please refer to the Special Topics No. 2 document for information on instantiable classes with the main method.

File: [MiniDraw.java](#)

```
//----- MiniDraw - Simplified DrawShape2 -----//

import javabook.*;
import java.awt.*;

/**
 * A simplified version of DrawShape2 to illustrate the use of paint
 * and repaint methods of the Component class.
 *
 * @author Dr. Caffeine
 */
class MiniDraw extends MainWindow
{
//-----
// Data Members
//-----

    /**
     * Constant for color magenta
     */
    private final int MAGENTA    = 0;

    /**
     * Constant for color cyan
     */
    private final int CYAN      = 1;

    /**
     * Constant for color red
     */
    private final int RED       = 2;

    /**
     * Constant for color blue
     */
    private final int BLUE      = 3;

    /**
     * Constant for color green
     */
    private final int GREEN     = 4;

    /**
     * ListBox for listing colors
     */
    private ListBox    colorListBox;

    /**
     * MessageBox for error messages
     */
    private MessageBox messageBox;

    /**
     * The color selected by the user

```

```

    */
    private int          selectedColor;

    /**
     * The color to draw the selected shape
     */
    private Color        shapeColor;

    /**
     * Boolean flag for input error. False if there's
     * an input error.
     */
    private boolean      canDraw;

//-----
// Constructors
//-----

    /**
     * Default constructor. The default window title is "Drawing Shape".
     */
    public MiniDraw()
    {
        messageBox = new MessageBox( this );

        colorListBox = new ListBox( this, "Select Color:" );
        colorListBox.addItem( "Magenta" );
        colorListBox.addItem( "Cyan" );
        colorListBox.addItem( "Red" );
        colorListBox.addItem( "Blue" );
        colorListBox.addItem( "Green" );

        setTitle( "Nondisappearing Shape" );
        setVisible( true );

        canDraw = false;
    }

//-----
// Public Methods:
//
// static void main ( String[ ] )
//
// void paint ( Graphics ) NOTE: Do not call this method
//
// void start ( )
//
//-----

    /**
     * The main method
     */
    public static void main( String[] args )
    {
        MiniDraw miniD;

        miniD = new MiniDraw( );

        miniD.start( );
    }

```

```

/**
 * Implement the painting method inherited from the
 * Component ancestor class
 *
 * @param graphics the Graphics object where the drawing takes place
 */
public void paint( Graphics graphics )
{
    if ( canDraw ) {
        graphics.setColor( shapeColor );

        graphics.drawRect( 100, 100, 400, 300 );
    }
}

/**
 * Top-level method that calls other private methods
 * to draw the selected shape in user-specified
 * position, size, and color.
 */
public void start( )
{
    selectColor( );
    draw( );
}

//-----
//    Private Methods:
//
//    void    draw                (          )
//    void    selectColor        (          )
//
//-----

/**
 * Draws the selected shape in a chosen position, size,
 * and color.
 *
 */
private void draw( )
{
    repaint( );
}

/**
 * Lets the user select the color from a ListBox.
 */
private void selectColor( )
{
    selectedColor = colorListBox.getSelectedIndex();

    //assume the color is selected okay
    canDraw = true;

    switch (selectedColor) {

        case ListBox.CANCEL:
        case ListBox.NO_SELECTION:
    }
}

```



```

        messageBox.show("No color was selected. " +
                        "Will draw in black.");
        setColor(Color.black);
        break;

    case MAGENTA:
        setColor(Color.magenta);
        break;

    case CYAN:
        setColor(Color.cyan);
        break;

    case RED:
        setColor(Color.red);
        break;

    case GREEN:
        setColor(Color.green);
        break;

    case BLUE:
        setColor(Color.blue);
        break;

    default:
        messageBox.show("ListBox error");
        canDraw = false;
        break;
    }
}

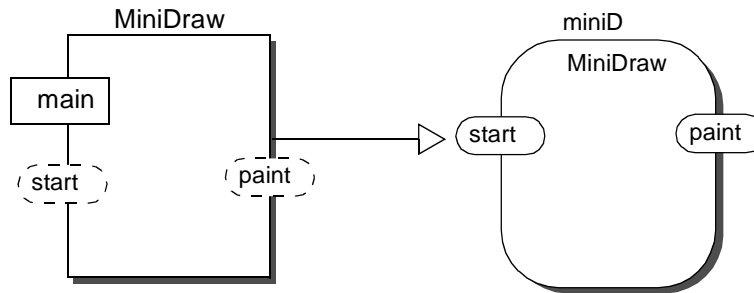
/**
 * Sets the color of shape
 *
 * @param color the new color in which to draw the shape
 */
private void setColor( Color color )
{
    shapeColor = color;
}
}

```

Figure 1 is the program's object diagram.

FIGURE 1

The object diagram for the **MiniDraw** program. Only the public methods are shown here.



Mr. Espresso: *Is it not possible to call the paint method directly from draw instead of calling it via repaint?*

Dr. Caffeine: *For a simple program such as this, yes. It will work here because the drawing is trivial. In general, however, safe and good style demands the calling of repaint and not calling paint directly from your code. The repaint method calls the update method that will perform some background work such as clearing the content before calling paint. Moreover, there are different versions of the repaint method which you can use for finer control. You can't achieve such finer control if you call the paint method directly.*

Dr. & Mr.



Never call the paint method directly from your code. Always call the repaint method instead.

2 The DrawShape2 Class

Once you grasp the key concept of using the paint method in the MiniDraw class, there should be no conceptual hurdle for you to understand the DrawShape2 class, though there are several points that require attention. The start method now includes the calls to the methods we omitted in MiniDraw:

```

public void start( )
{
    describeProgram( );
    selectShape( );
    selectColor( );
    selectDimension( );
    draw( );
}

```

In the original DrawShape class, we have to insert the statement

```
outputBox.waitUntilClose( );
```

at the end of the describeProgram method so the user gets a chance to close the outputBox before proceeding with the drawing. This is necessary because if we don't, the shape drawn beneath the outputBox will not become visible. Since we have now implemented the nondisappearing shape drawing, we can let the outputBox remain on the screen.

The selectShape, selectColor, and selectDimension methods are essentially the same as those in the original DrawShape class. The selectDimension and its subordinate methods getCIRCLEDimension, getLineDimension, and getRectangleDimension set the canDraw variable to true or false depending on the input data values. Any invalid entry will result in setting canDraw to false. The draw method is the same as the one in MiniDraw. No actual drawing will take place inside the draw method, it only calls the repaint method that will eventually cause the paint method to be executed.

The actual drawing will take place inside the paint method so the drawn shape won't disappear. The paint method is defined as follows:

```

public void paint( Graphics graphics )
{
    if ( canDraw ) {
        graphics.setColor( shapeColor );

        switch (selectedShape) {

            case LINE:
                graphics.drawLine( originX, originY,
                                   endX, endY );
                break;

            case CIRCLE:
                graphics.drawOval( originX-radius,
                                   originY-radius,

```

```

                2*radius,
                2*radius );

            break;

        case RECTANGLE:
            graphics.drawRect( originX, originY,
                               width, height );
            break;
    }
}

```

Notice that the Graphics object does not support the drawCircle method, so we have to use the more general drawOval method to draw circles.

Here's the complete DrawShape2 class:

File: [DrawShape2.java](#)

```

//----- DrawShape2 Nondisappearing Shapes -----//

import javabook.*;
import java.awt.*;

/**
 * This class is the top-level controller object for managing
 * all other objects in the program.
 *
 * @author Dr. Caffeine
 */
class DrawShape2 extends MainWindow
{
    //-----
    //  Data Members
    //-----

    /**
     * Constant for a line
     */
    private final int LINE      = 0;

    /**
     * Constant for a circle
     */
    private final int CIRCLE    = 1;

    /**
     * Constant for a rectangle
     */
    private final int RECTANGLE = 2;

    /**
     * Constant for color magenta
     */
    private final int MAGENTA   = 0;
}

```

```

/**
 * Constant for color cyan
 */
private final int CYAN      = 1;

/**
 * Constant for color red
 */
private final int RED       = 2;

/**
 * Constant for color blue
 */
private final int BLUE      = 3;

/**
 * Constant for color green
 */
private final int GREEN     = 4;

/**
 * TextBox for displaying the test messages
 */
private TextBox  outputBox;

/**
 * Input for getting the shape dimension
 */
private InputBox  inputBox;

/**
 * MessageBox for error messages
 */
private MessageBox  messageBox;

/**
 * ListBox for listing shapes
 */
private ListBox  shapeListBox;

/**
 * The shape selected by the user
 */
private int      selectedShape;

/**
 * ListBox for listing colors
 */
private ListBox  colorListBox;

/**
 * The color selected by the user
 */
private int      selectedColor;

/**
 * Boolean flag for input error. False if there's
 * an input error.
 */
private boolean  canDraw;

```

```

/**
 * X-coordinate of the shape's origin point
 */
private int  originX;

/**
 * Y-coordinate of the shape's origin point
 */
private int  originY;

/**
 * X-coordinate of the line's end point
 */
private int  endX;

/**
 * Y-coordinate of the line's end point
 */
private int  endY;

/**
 * The radius of the circle
 */
private int  radius;

/**
 * The width of the rectangle
 */
private int  width;

/**
 * The height of the rectangle
 */
private int  height;

/**
 * The color to draw the selected shape
 */
private Color shapeColor;

//-----
// Constructors
//-----

/**
 * Default constructor. The default window title is "Drawing Shape".
 */
public DrawShape2()
{
    outputBox  = new OutputBox ( this );
    inputBox   = new InputBox  ( this );
    messageBox = new MessageBox( this );

    shapeListBox = new ListBox( this, "Select Shape:" );
    shapeListBox.addItem( "Line" );
    shapeListBox.addItem( "Circle" );
    shapeListBox.addItem( "Rectangle" );

    colorListBox = new ListBox( this, "Select Color:" );

```

```

        colorListBox.addItem( "Magenta" );
        colorListBox.addItem( "Cyan" );
        colorListBox.addItem( "Red" );
        colorListBox.addItem( "Blue" );
        colorListBox.addItem( "Green" );

        setTitle( "Nondisappearing Shape" );
        setVisible( true );
        outputBox.setVisible( true );
    }

//-----
//      Public Methods:
//
//      static void  main  ( String[ ] )
//
//          void  paint  ( Graphics ) NOTE: Do not call this method
//
//          void  start  (          )
//
//-----

/**
 * The main method
 */
public static void main( String[] args )
{
    DrawShape2 drawShape2;
    drawShape2 = new DrawShape2( );
    drawShape2.start( );
}

/**
 * Implement the painting method inherited from the
 * Component ancestor class
 *
 * @param graphics the Graphics object where the drawing takes place
 */
public void paint( Graphics graphics )
{
    if ( canDraw ) {
        graphics.setColor( shapeColor );

        switch (selectedShape) {

            case LINE:
                graphics.drawLine( originX, originY, endX, endY );
                break;

            case CIRCLE:
                graphics.drawOval( originX-radius, originY-radius,
                                   2*radius, 2*radius );
                break;

            case RECTANGLE:
                graphics.drawRect( originX, originY, width, height );
                break;

        }
    }
}

```

```

/**
 * Top-level method that calls other private methods
 * to draw the selected shape in user-specified
 * position, size, and color.
 */
public void start( )
{
    describeProgram( );
    selectShape( );
    selectColor( );
    selectDimension( );
    draw( );
}

//-----
// Private Methods:
//
// void    describeProgram    (        )
// void    draw                (        )
//
// void    getCircleDimension  (        )
// void    getLineDimension    (        )
// void    getRectangleDimension (        )
//
// void    selectColor         (        )
// void    selectDimension     (        )
// void    selectShape         (        )
//
//-----

/**
 * Provides a brief explanation of the program to the user.
 *
 */
private void describeProgram( )
{
    outputBox.skipLine(1);
    outputBox.printLine
        ("This program draws a line, circle, or rectangle");
    outputBox.printLine("you choose in the color of your choice.");

    outputBox.skipLine(1);

    /***** NOTE *****/
    // waitUntilClose is no longer necessary because
    // the shape will be redrawn
    //outputBox.printLine("Close this description window to continue");
    //outputBox.printLine("with the program.");

    //outputBox.waitUntilClose( );
}

/**
 * Draws the selected shape in a chosen position, size,
 * and color.
 *
 */
private void draw( )
{
    repaint( );
}

```



```

    }

    /**
     * Gets the dimension of a circle
     */
    private void getCircleDimension()
    {
        originX = inputBox.getInteger("X-coord of the center");
        originY = inputBox.getInteger("Y-coord of the center");
        radius   = inputBox.getInteger("Radius of the circle");

        //make sure everything is positive
        if (originX < 0 || originY < 0 || radius < 0) {
            messageBox.show("Negative number is entered. " +
                            "Cannot draw a circle.");
            canDraw = false;
        }
        else { //input okay
            canDraw = true;
            // outputBox.println("circle dimension okay"); //TEMP
        }
    }

    /**
     * Gets the dimension of a line
     */
    private void getLineDimension( )
    {
        originX = inputBox.getInteger("X-coord of starting point");
        originY = inputBox.getInteger("Y-coord of starting point");
        endX     = inputBox.getInteger("X-coord of ending point");
        endY     = inputBox.getInteger("Y-coord of ending point");

        //make sure everything is positive
        if (originX < 0 || originY < 0 || endX < 0 || endY < 0) {
            messageBox.show("Negative number is entered. " +
                            "Cannot draw a line.");
            canDraw = false;
        }
        else { //input okay
            canDraw = true;
        }
    }

    /**
     * Gets the dimension of a rectangle.
     */
    private void getRectangleDimension( )
    {
        originX = inputBox.getInteger("X-coord of origin");
        originY = inputBox.getInteger("Y-coord of origin");
        width   = inputBox.getInteger("Rectangle width");
        height  = inputBox.getInteger("Rectangle height");

        //make sure everything is positive
        if (originX < 0 || originY < 0 || width < 0 || height < 0) {
            messageBox.show("Negative number is entered. " +
                            "Cannot draw a rectangle.");
            canDraw = false;
        }
        else { //input okay

```

```

        canDraw = true;
    }
}

/**
 * Lets the user select the color from a ListBox.
 */
private void selectColor( )
{
    selectedColor = colorListBox.getSelectedIndex();

    switch (selectedColor) {

        case ListBox.CANCEL:
        case ListBox.NO_SELECTION:
            messageBox.show("No color was selected. " +
                            "Will draw in black.");
            setColor(Color.black);
            break;

        case MAGENTA:
            setColor(Color.magenta);
            break;

        case CYAN:
            setColor(Color.cyan);
            break;

        case RED:
            setColor(Color.red);
            break;

        case GREEN:
            setColor(Color.green);
            break;

        case BLUE:
            setColor(Color.blue);
            break;

        default:
            messageBox.show("ListBox error");
            break;
    }
}

/**
 * Lets the user select the size and position.
 */
private void selectDimension( )
{
    switch (selectedShape) {

        case ListBox.CANCEL: // user canceled, so do nothing
            canDraw = false;
            break;

        case ListBox.NO_SELECTION: // no shape selected
            canDraw = false;
            break;
    }
}

```

```

        case LINE:
            getLineDimension();
            break;

        case CIRCLE:
            getCircleDimension();
            break;

        case RECTANGLE:
            getRectangleDimension();
            break;

        default:
            messageBox.show("ListBox error");
            canDraw = false;
            break;
    }
}

/**
 * Lets the user select the shape from a ListBox.
 */
private void selectShape( )
{
    selectedShape = shapeListBox.getSelectedIndex();
}

/**
 * Sets the color of shape
 * @param color the new color in which to draw the shape
 */
private void setColor( Color color )
{
    shapeColor = color;
}
}

```

Figure 2 is the object diagram for the DrawShape2 class.

FIGURE 2

The object diagram for the **DrawShape2** program. Notice that there is only one class and one instance of that class. Only the public methods are shown here.

