Programming Language
The fourth day

Inheritance (2)

- “Java AWT (Abstract Windowing Toolkit) -

This document first introduces the technology of “Java AWT (Abstract Windowing Toolkit)”. And “while statement” will be introduced.

1. Java AWT

The java.awt libraries are set of classes provided by java in order to draw shapes on a window. The abbreviation AWT stands for Abstract Windowing Toolkit. Today, the library has been converted into a huge set of classes which allows the user create an entire GUI base application.

AWT supports Graphical User Interface (GUI) programming. AWT features include as follows.

- A set of native user interface components
- A robust event-handling model
- Graphics and imaging tools, including shape, color, and font classes
- Layout managers, for flexible window layouts that do not depend on a particular window size or screen resolution
- Data transfer classes, for cut-and-paste through the native platform clipboard

Graphics and imaging tools and robust event handling models will be introduced in today’s and upcoming classes in “Programming Language.”

2. Basic Model of GUI Program

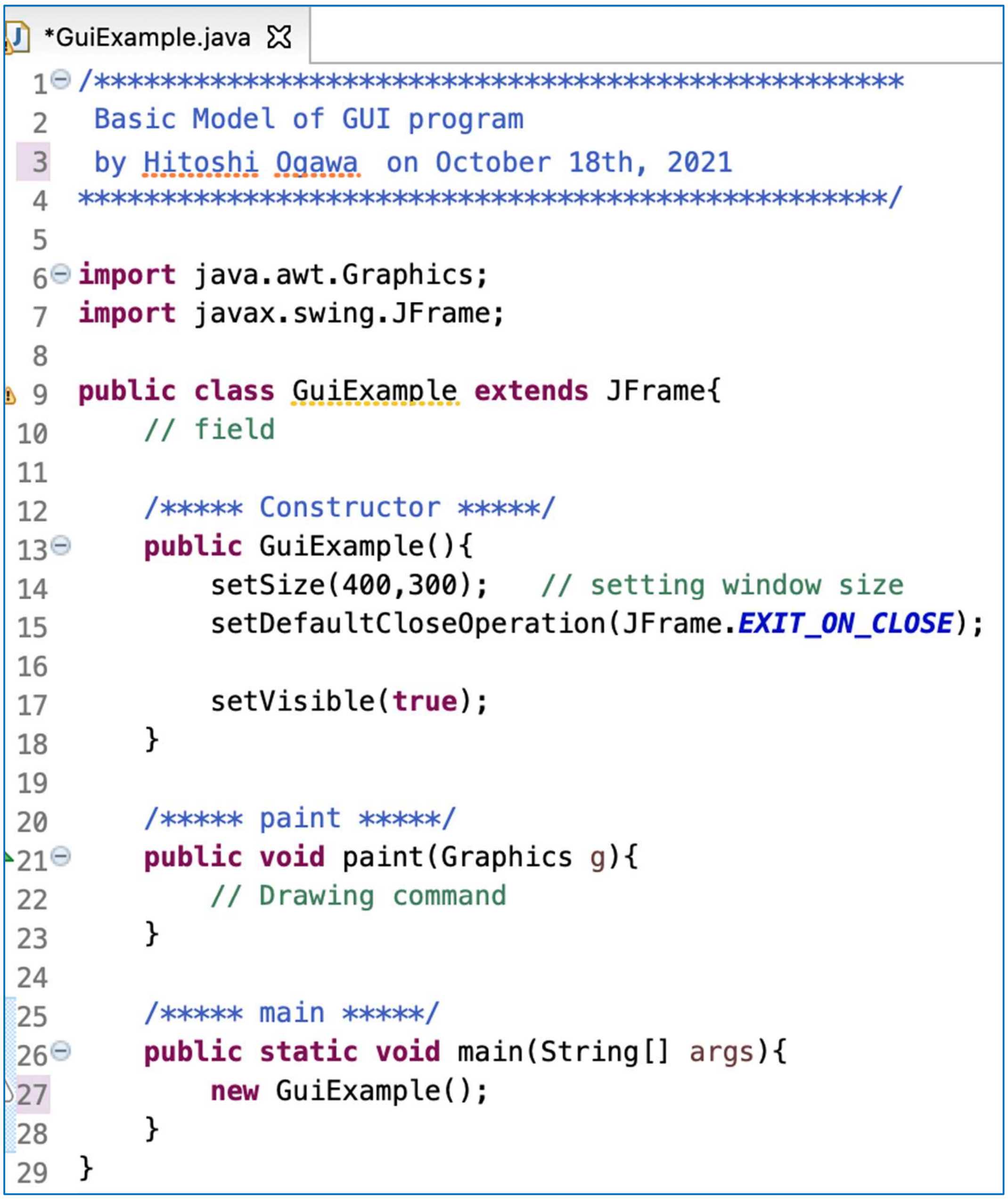
In order to operate and confirm the graphics function of AWT, environment for drawing is necessary. Figure 1 introduces the basic model of the Java program generally used for GUI. Figure 2 shows an example of GUI program.

“Color”, “Font” and “Graphics” classes in package “java.awt” and “JFrame” class in package “javax.swing” are necessary to create a GUI program. So, we use the keyword “import” to make them available in this program as shown on lines 6 and 7 in Figure 1, on lines 6 to 9 in Figure 2. We can use any class in package “java.awt”, if the following statement is used.

```
import java.awt.*;
```

GuiExample class inherits JFrame. JFrame class is in javax.swing package (which will introduced on the seventh day). The three methods on lines 14 to 17 in Figure 1 are provided by JFrame or its superclass. The “setSize” method is used to set window size. A window with 400 horizontal pixels and 300 vertical pixels is created with “setSize(400, 300).” The method on line 15 set to close the window when the X mark at top right of the window is clicked. The method “setVisible(true)” set to make the window visible.

The method “paint(Graphics g)” is a method provided by the superclass of JFrame and is called when drawing is necessary. If we want to draw something in the window, we can write the program in the paint method. The paint method has an argument of type Graphics which is provided by “java.awt” package, we can call the graphics drawing methods of this Graphics object. These methods are introduced in Chapter 3.


*GuiExample.java

```
 1 /************************************************
 2   Basic Model of GUI program
 3   by Hitoshi Ogawa  on October 18th, 2021
 4 ************************************************/
 5
 6 import java.awt.Graphics;
 7 import javax.swing.JFrame;
 8
 9 public class GuiExample extends JFrame{
10     // field
11
12     /***** Constructor *****/
13     public GuiExample(){
14         setSize(400,300);   // setting window size
15         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16
17         setVisible(true);
18     }
19
20     /***** paint *****/
21     public void paint(Graphics g){
22         // Drawing command
23     }
24
25     /***** main *****/
26     public static void main(String[] args){
27         new GuiExample();
28     }
29 }
```

Figure 1. Basic Model of GUI Program

3. Graphics

The "java.awt.Graphics" class is the class for all graphics contexts which allow an application to draw onto components that can be realized on various devices, or onto off-screen images as well.

The Graphics class provides methods for drawing three types of graphical objects.

3. 1 Text string

We use drawString method to display string on a window. However, before using drawstring, we need to set font, style, size and color.

To set font, style and size of character, we need to create an instance of the Font class. The format is as follows.

```
Font font = new Font(font_name, style, size);
```

Note that in order to use java.awt.Font class, we need to import at the beginning of the progam.

font: (Font) the variable assigned an instance of Font class
font_name: (String) use the font registered in MacBook. The registered fonts can be found in "Font Book".
style: (int) select from PLAIN, BOLD and ITALIC defined in the field of Font class. These can be used in combination.
size: (int) Font's point size.

To enable the font we have set, we need to use the setFont method in paint method as follows.

```
g.setFont(Pont font);
```

The variable *g* is given as an argument to the paint method and the variable *font* is assigned an instance Font class.

The color of character is set using setColor method of Graphics class. The format is as follows.

```
g.setColor( Color color);
```

The color is selected from black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white and yellow in the field of Color class. We can also use words with all letters capitalized.

We can use the desired color, if we create an object of Color class as follows.

```
g.setColor(new Color(red, green, blue));
```

The arguments of Color class red, green and blue indicate red green and blue densities in the renge of 0 to 255, respectively.

Finally, the string is displayed by the drawstring method as follows.

```
g.drawString(String str, int xBaseline, int yBaseLine);
```

str: the string to be displayed.
xBaseline: the x-coordinate of the lower left corner of the displayed string
yBaseline: the y-coordinate of the lower left corner of the displayed string

Figure 2 shows examples of writing text in a window, and the program "FontTest.java" that implements them is shown in Figure 3.

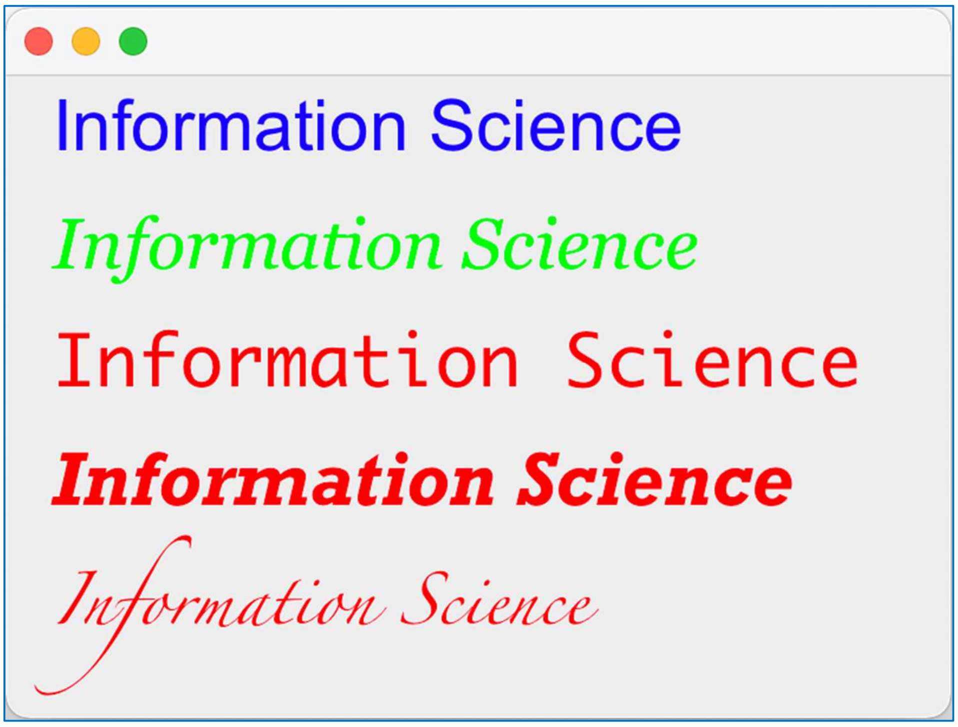
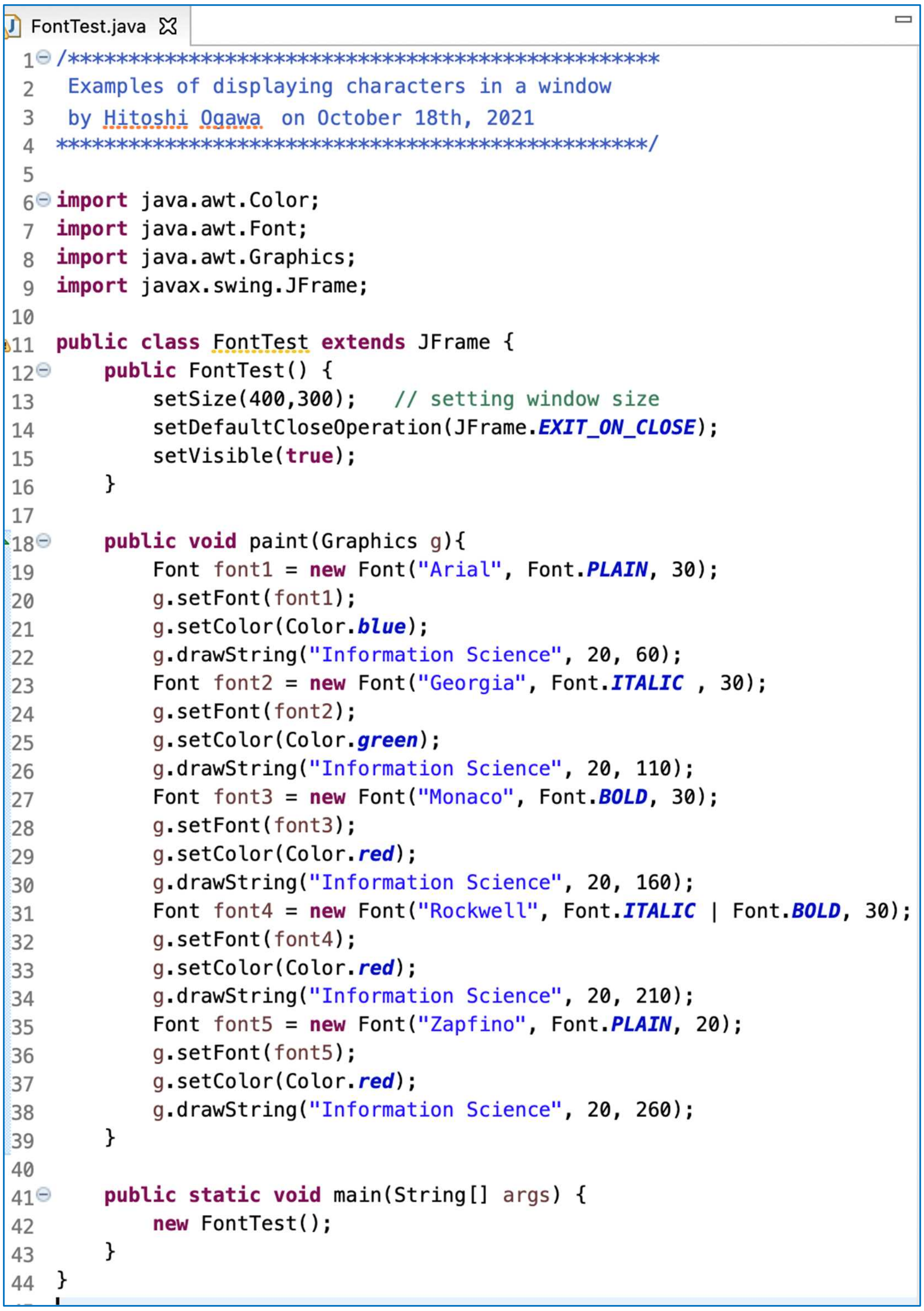


Figure 2. Examples of writing text in a window.



FontTest.java

```java
/***************************************************
 Examples of displaying characters in a window
 by Hitoshi Ogawa  on October 18th, 2021
***************************************************/

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import javax.swing.JFrame;

public class FontTest extends JFrame {
    public FontTest() {
        setSize(400,300);   // setting window size
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    public void paint(Graphics g){
        Font font1 = new Font("Arial", Font.PLAIN, 30);
        g.setFont(font1);
        g.setColor(Color.blue);
        g.drawString("Information Science", 20, 60);
        Font font2 = new Font("Georgia", Font.ITALIC , 30);
        g.setFont(font2);
        g.setColor(Color.green);
        g.drawString("Information Science", 20, 110);
        Font font3 = new Font("Monaco", Font.BOLD, 30);
        g.setFont(font3);
        g.setColor(Color.red);
        g.drawString("Information Science", 20, 160);
        Font font4 = new Font("Rockwell", Font.ITALIC | Font.BOLD, 30);
        g.setFont(font4);
        g.setColor(Color.red);
        g.drawString("Information Science", 20, 210);
        Font font5 = new Font("Zapfino", Font.PLAIN, 20);
        g.setFont(font5);
        g.setColor(Color.red);
        g.drawString("Information Science", 20, 260);
    }

    public static void main(String[] args) {
        new FontTest();
    }
}
```

Figure 3. FontTest.java.

3. 2 Vector-graphics primitives and shapes

Four kinds of drawing methods of Graphics class are introduced. It is necessary to set colors before using them.

(A) Line

drawLine(int *xStart*, int *yStart*, int *xEnd*, int *yEnd*)

draws a line, using the current color, between the points (*xStart*, *yStart*) and (*xEnd*, *yEnd*) in this graphics context's coordinate system.

(B) Rectangle

drawRect(int *topLeftX*, int *topLeftY*, int *width*, int *height*)

draws the outline of the specified rectangle. The left and right edges of the rectangle are at *topLeftX* and *topLeftX* + *width*. The top and bottom edges are at *topLeftY* and *topLeftY* + *height*. The rectangle is drawn using the graphics context's current color.

fillRect(int *topLeftX*, int *topLeftY*, int *width*, int *height*)

fills the specified rectangle. The left and right edges of the rectangle are at *topLeftX* and *topLeftX* + *width* - 1. The top and bottom edges are at *topLeftY* and *topLeftY* + *height* - 1. The resulting rectangle covers an area width pixels wide by height pixels tall. The rectangle is filled using the graphics context's current color.

(C) Ellipse

drawOval(int *topLeftX*, int *topLeftY*, int *width*, int *height*)

draws the outline of an oval. The result is a circle or ellipse that fits within the rectangle specified by the *topLeftX*, *topLeftY*, *width*, and *height* arguments. The oval covers an area that is *width* + 1 pixels wide and *height* + 1 pixels tall.

fillOval(int *topLeftX*, int *topLeftY*, int *width*, int *height*)

fills an oval bounded by the specified rectangle with the current color.

(D) Arc and Fan

drawArc(int *topLeftX*, int *topLeftY*, int *width*, int *height*, int *startAngle*, int *arcAngle*)

draws the outline of a circular or elliptical arc covering the specified rectangle. The resulting arc begins at *startAngle* and extends for *arcAngle* degrees, using the current color. Angles are interpreted such that 0 degrees is at the

3 o'clock position. A positive value indicates a counter-clockwise rotation while a negative value indicates a clockwise rotation.

The center of the arc is the center of the rectangle whose origin (top left) is (*topLeftX*, *topLeftY*) and whose size is specified by the *width* and *height* arguments. The resulting arc covers an area *width* + 1 pixels wide by *height* + 1 pixels tall.

**fillArc(int *topLeftX*, int *topLeftY*, int *width*, int *height*,
int *startAngle*, int *arcAngle*)**

fills a circular or elliptical arc covering the specified rectangle.

Figure 4 shows examples of using functions introduced in (A) to (D), and the program "VectorGraphics.java" that implements them is shown in Figure 5.

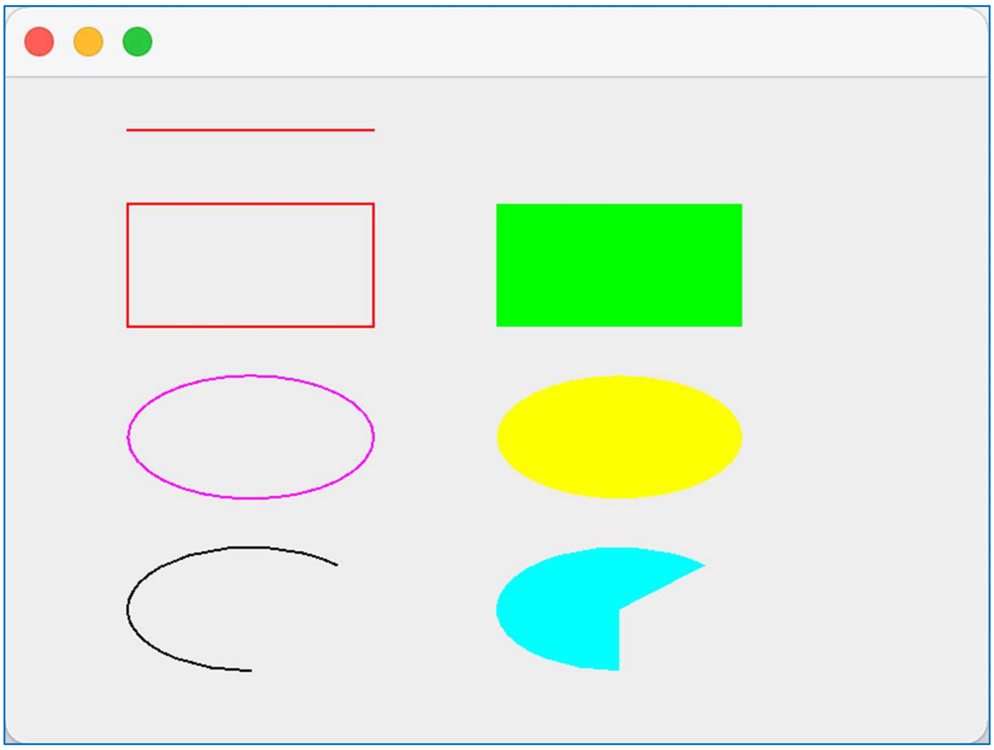
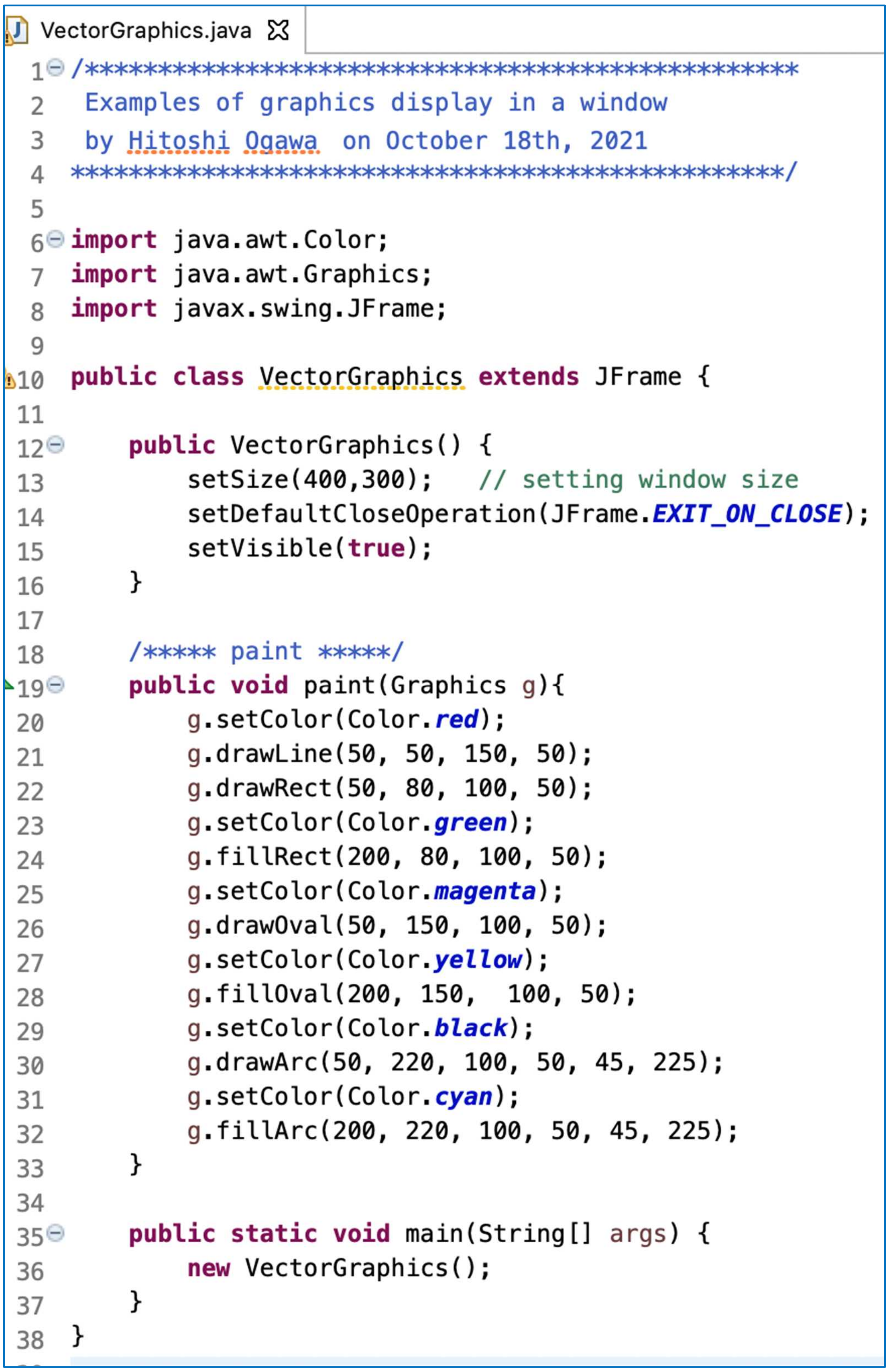


Figure 4. Examples of using functions introduced in (A) to (D).


VectorGraphics.java

```java
/*************************************************
 Examples of graphics display in a window
 by Hitoshi Ogawa  on October 18th, 2021
*************************************************/

import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JFrame;

public class VectorGraphics extends JFrame {

    public VectorGraphics() {
        setSize(400,300);   // setting window size
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    /***** paint *****/
    public void paint(Graphics g){
        g.setColor(Color.red);
        g.drawLine(50, 50, 150, 50);
        g.drawRect(50, 80, 100, 50);
        g.setColor(Color.green);
        g.fillRect(200, 80, 100, 50);
        g.setColor(Color.magenta);
        g.drawOval(50, 150, 100, 50);
        g.setColor(Color.yellow);
        g.fillOval(200, 150,  100, 50);
        g.setColor(Color.black);
        g.drawArc(50, 220, 100, 50, 45, 225);
        g.setColor(Color.cyan);
        g.fillArc(200, 220, 100, 50, 45, 225);
    }

    public static void main(String[] args) {
        new VectorGraphics();
    }
}
```

Figure 5. VectorGraphics.java.

3. 3 Bitmap images

We can use "drawImage" method to draw the specified image. However, since we have not learned how to load and process images, the methods for image will be introduced at the tenth day.

4. While Loop

A while loop statement in Java programming language repeatedly executes a target statement as long as a given condition is true.

The syntax of a while loop

```
while(Boolean_expression){
      // statements
}
```

Here, *statements* may be a single statement or multiple statements.

The condition is described using the relational operators and the logical operators. Refer Table 2 on page 7 and Table 3 on page 8 of "Inheritance (1)" (The third day).

In the while header, it is decided whether to execute the statements. If the *Boolean_expression* result is true, then the statements in the loop will be executed. This will continue as long as the expression result is true. If the condition is false, program control is passed to the net line of while statement.

The Java do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and we must have to execute the loop at least once, it is recommended to use do-while loop.

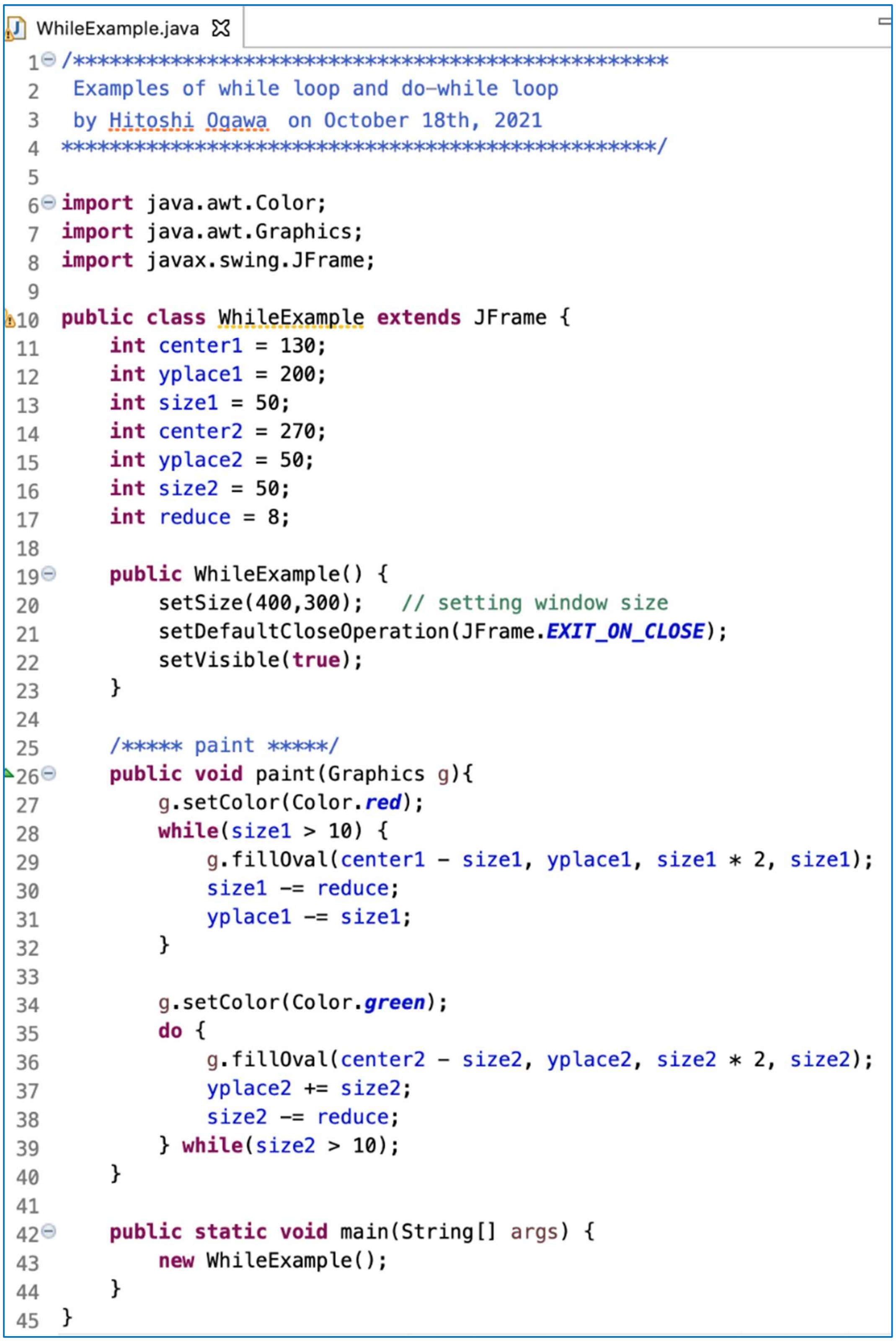
The syntax of a do-while loop

```
do {
      // statements
} while(Boolean_expression);
```

An example program using while loop and do-while loop is shown in Figure 6. The result of the program is shown in Figure 7.

The variable center1 (center2) shows the x-coordinate of the center of the ellipse. The variable yplace1 (yplace2) shows the upper left y coordinate of the rectangle circumscribing the ellipse. The variable size1 (size2) indicates the length of the minor axis of the ellipse (half the length of the major axis).

The red ellipses are drawn using a while loop. The green ellipses are drawn using a do-while loop.



```java
/**************************************************
 Examples of while loop and do-while loop
 by Hitoshi Ogawa  on October 18th, 2021
**************************************************/

import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JFrame;

public class WhileExample extends JFrame {
    int center1 = 130;
    int yplace1 = 200;
    int size1 = 50;
    int center2 = 270;
    int yplace2 = 50;
    int size2 = 50;
    int reduce = 8;

    public WhileExample() {
        setSize(400,300);   // setting window size
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    /***** paint *****/
    public void paint(Graphics g){
        g.setColor(Color.red);
        while(size1 > 10) {
            g.fillOval(center1 - size1, yplace1, size1 * 2, size1);
            size1 -= reduce;
            yplace1 -= size1;
        }

        g.setColor(Color.green);
        do {
            g.fillOval(center2 - size2, yplace2, size2 * 2, size2);
            yplace2 += size2;
            size2 -= reduce;
        } while(size2 > 10);
    }

    public static void main(String[] args) {
        new WhileExample();
    }
}
```

Figure 6. An example of while loop and do-while loop.

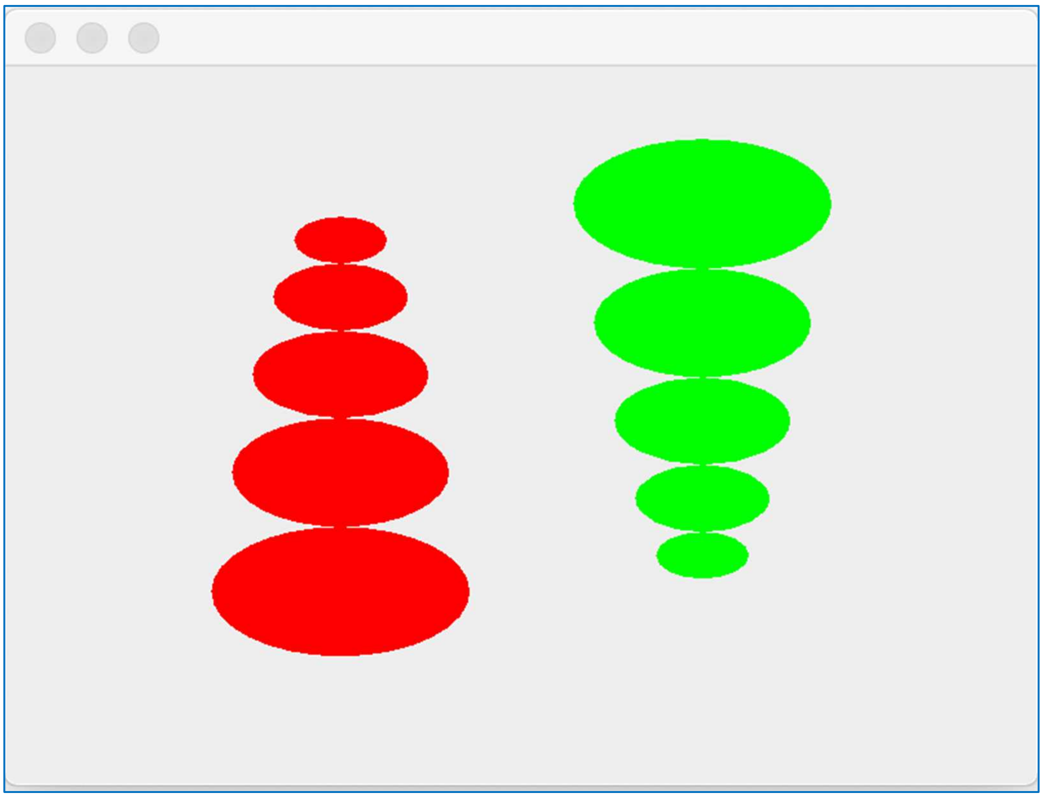


Figure 7. The result of the program in Figure 3.