

## Working with Images

### 1. BufferedImage

Images are described by a width and a height, measured in pixels, and have a coordinate system that is independent of the drawing surface. There are a number of common tasks when working with images.

- Loading an external GIF, PNG, JPEG image format file into the internal image representation used by Java 2D API\*.
- Directly creating a Java 2D image and rendering to it.
- Drawing the contents of a Java 2D image on to a drawing surface.
- Saving the contents of a Java 2D image to an external GIF, PNG, or JPEG image file.

The “java.awt.Image” class is the superclass that represents graphical images as rectangular arrays of pixels. The “java.awt.Image” is a subclass of Image class, and allows the application to operate directly with image data (for example, retrieving or setting up the pixel color). It is used to handle and manipulate the image data. A BufferedImage is made of ColorModel of image data. All BufferedImage objects have an upper left corner coordinate of (0, 0).

A syntax of BufferedImage is as follows.

`BufferedImage(int width, int height, int imageType)`

It creates an image of *imageType* with width *width* and vertical *height*.

The main *imageType* provided in the field of BufferedImage is shown in Table 1. It's possible to produce any desired color by blending three colors of light—red, green, and blue—each with its own intensity. This is known as additive synthesis—generating a color by adding unique amounts of three "primary" colors of light.

---

\* : Java 2D API classes are organized into the following packages.

- *java.awt*: The main package for the Java Abstract Window Toolkit.
- *java.awt.geom*: The Java standard library of two dimensional geometric shapes such as lines, ellipses, and quadrilaterals.
- *java.awt.font*: The library for manipulating glyphs in Java.
- *java.awt.color*: The library dealing with the many different ways that color can be represented.
- *java.awt.image*: The library for manipulating graphical images.
- *java.awt.print*: The library of tools for writing to paper.

Table 1. The main *imageType* provided in the field of BufferedImage

imageType	Description
<b>TYPE_3BYTE_BGR</b>	Represents an image with 8-bit RGB color components, corresponding to a Windows-style BGR color model with the colors Blue, Green, and Red stored in 3 bytes.
<b>TYPE_BYTE_GRAY</b>	Represents an unsigned byte grayscale image, non-indexed.
<b>TYPE_INT_ARGB</b>	Represents an image with 8-bit RGBA color components packed into integer pixels.
<b>TYPE_INT_BGR</b>	Represents an image with 8-bit RGB color components, corresponding to a Windows- or Solaris- style BGR color model, with the colors Blue, Green, and Red packed into integer pixels.
<b>TYPE_INT_RGB</b>	Represents an image with 8-bit RGB color components packed into integer pixels.

Thus, we can describe any colored light in terms of its intensity at the three frequencies that correspond to the specific colors red, green, and blue. A fourth plane is often useful for what is known as the alpha channel—a channel that stores information about how transparent a pixel should be when overlaid on another image.

The main methods of BufferedImage are introduced in Table 2.

## 2. Reading/Loading and Writing/Saving an Image

The utility class “javax.imageio.ImageIO” provides lots of utility method related to images processing in Java. Most common of them is reading from image file and writing images to file in java. We can write any of “.jpg”, “.png”, “.bmp” or “.gif” images to file in Java.

### 2. 1 Reading/Loading an Image

The method “BufferedImage read(File input)” of ImageIO class is used to load an image from a specific file. It returns a BufferedImage as the result of decoding a supplied File. An example code is as follows.

```
BufferedImage img = null;
try {
    img = ImageIO.read(new File(filename));
} catch (IOException e) {
}
```

Table 2. The main method of BufferedImage.

Method & Description of BufferedImage
<b>int getHeight()</b> Returns the height of the BufferedImage.
<b>Graphics getGraphics()</b> This method returns a Graphics.
<b>int getRGB(int x, int y)</b> Returns an integer pixel in the default RGB color model (TYPE_INT_ARGB) and default sRGB colorspace.
<b>BufferedImage getSubimage(int x, int y, int w, int h)</b> Returns a subimage defined by a specified rectangular region. The point (x, y) is the coordinate of the upper-left corner of the specified rectangular region. The parameter w and h are the width and the height of the specified rectangular region respectively.
<b>int getType()</b> Returns the image type.
<b>int getWidth()</b> Returns the width of the BufferedImage.
<b>void setRGB(int x, int y, int rgb)</b> Sets a pixel in this BufferedImage to the specified RGB value.

The value *filename* is the name of the file containing the image (for example, "image.jpg").

## 2. 2 Writing/Saving an Image

The ImageIO class provides a simple way to save images in a variety of image formats in the following example.

```
ImageIO.write(BufferedImage image, String format, File output)
                                                    throws IOException
```

*image*: a BufferedImage to be written.

*format*: a String containing the informal name of the format (for example, "png", "jpg", "gif", etc.).

*output*: a File to be written to.

An example is as follows.

```
BufferedImage bimage; // Image data has been substituted.
try{
    File output = new File("saved.jpg");
    ImageIO.write(bimage, "jpg", output);
} catch (IOException e){
}
```

### 3. Drawing an Image

To draw the image in the window, the “drawImage” method of the Graphics class is used. There are various ways to use “drawImage”, and we can specify the drawing position, size, drawing range, etc. How to use “drawImage” is introduced below.

#### (A) **drawImage(Image *img*, int *x*, int *y*, ImageObserver *observer*)**

*img*: the specified image to be drawn. This method does nothing if *img* is null.

*x*: the x coordinate.

*y*: the y coordinate.

*observer*: object to be notified as more of the image is converted (usually “this”).

Draws as much of the specified image as is currently available. The image is drawn with its top-left corner at (*x*, *y*) in this graphics context's coordinate space.

#### (B) **drawImage(Image *img*, int *x*, int *y*, int *width*, int *height*, ImageObserver *observer*)**

*img*: the specified image to be drawn. This method does nothing if *img* is null.

*x*: the x coordinate.

*y*: the y coordinate.

*width*: the width of the rectangle.

*height*: the height of the rectangle.

*observer*: object to be notified as more of the image is converted (usually “this”).

Draws the specified image to fit inside the specified rectangle (*width* by *height*). The image is drawn inside the specified rectangle of this graphics context's coordinate space, and is scaled if necessary.

(C) **drawImage**(Image *img*, int *dx1*, int *dy1*, int *dx2*, int *dy2*,  
int *sx1*, int *sy1*, int *sx2*, int *sy2*, ImageObserver *observer*)

*img*: the specified image to be drawn. This method does nothing if *img* is null.

*dx1*: the x coordinate of the first corner of the destination rectangle.

*dy1*: the y coordinate of the first corner of the destination rectangle.

*dx2*: the x coordinate of the second corner of the destination rectangle.

*dy2*: the y coordinate of the second corner of the destination rectangle.

*sx1*: the x coordinate of the first corner of the source rectangle.

*sy1*: the y coordinate of the first corner of the source rectangle.

*sx2*: the x coordinate of the second corner of the source rectangle.

*sy2*: the y coordinate of the second corner of the source rectangle.

*observer*: object to be notified as more of the image is converted (usually “this”).

Draws as much of the specified area of the specified image as is currently available, scaling it to fit inside the specified area of the destination drawable surface.

This method always uses the unscaled version of the image to render the scaled rectangle and performs the required scaling on the fly.

Scaling of the image from source to destination is performed such that the first coordinate of the source rectangle is mapped to the first coordinate of the destination rectangle, and the second source coordinate is mapped to the second destination coordinate. The subimage is scaled and flipped as needed to preserve those mappings.

#### 4. Examples

In this section, some examples are introduced using the image file “image1.jpg”. Refer to Appendix A for how to download images.

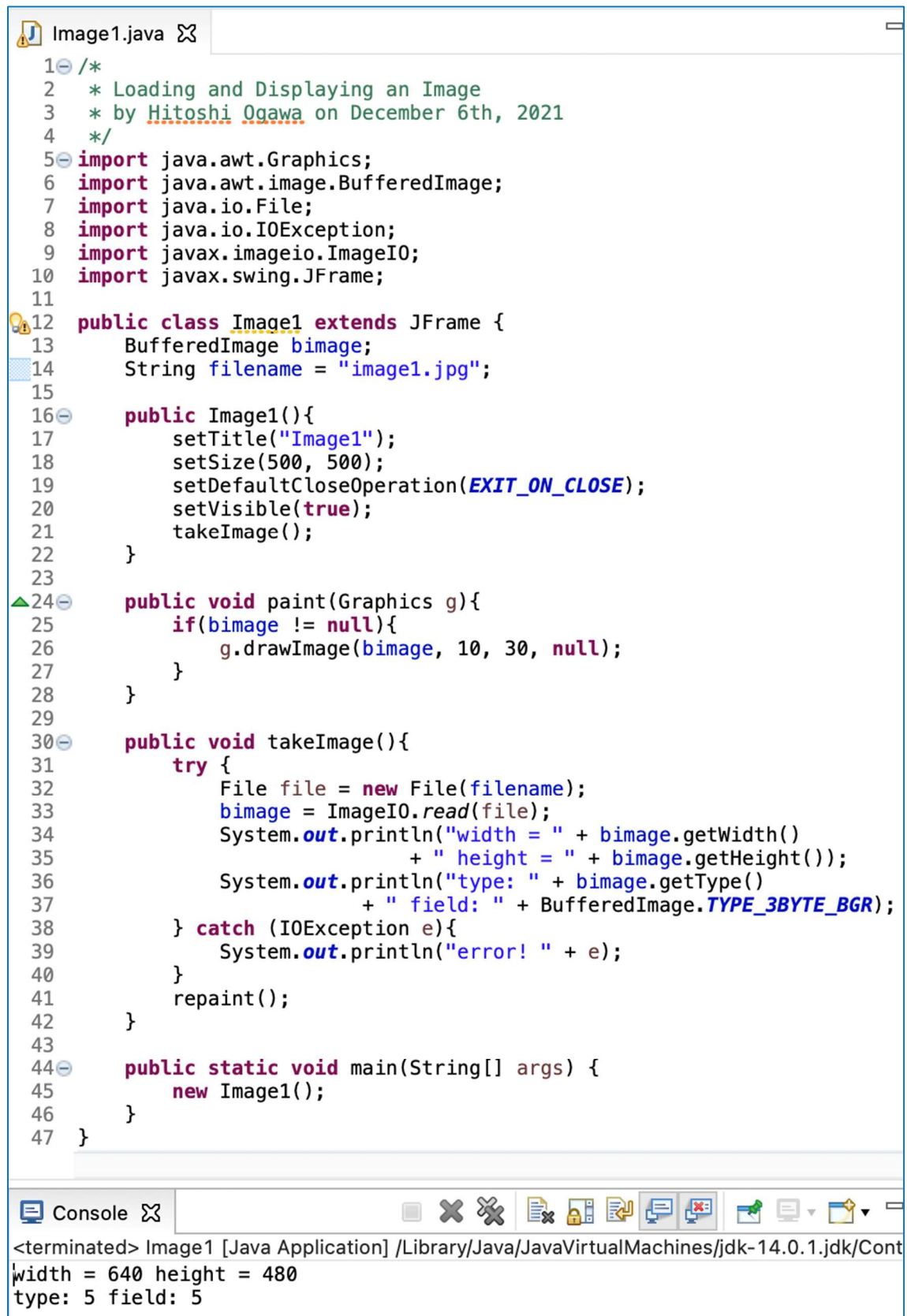
##### (1) Loading and Displaying an Image

The program “Image1.java” to load and draw an image from “image1.jpg” is shown in Figure 1.

The variable “bimage” is declared as BufferedImage type (line 13). The file name “image1.jpg” is set in the String variable “filename”.

In the paint method, the image is drawn from the upper left (10, 30) of the window using “drawImage” of 3.(A) only when the bimage has contents. For the fourth argument, “this” is used to indicate the component class “Image1” to display.

The “takeImage” method called from the constructor “Image1” loads the image from “image1.jpg” (lines 32 and 33) and outputs its parameters (lines 34 to 37).



```
1 /*
2  * Loading and Displaying an Image
3  * by Hitoshi Ogawa on December 6th, 2021
4  */
5 import java.awt.Graphics;
6 import java.awt.image.BufferedImage;
7 import java.io.File;
8 import java.io.IOException;
9 import javax.imageio.ImageIO;
10 import javax.swing.JFrame;
11
12 public class Image1 extends JFrame {
13     BufferedImage bimage;
14     String filename = "image1.jpg";
15
16     public Image1(){
17         setTitle("Image1");
18         setSize(500, 500);
19         setDefaultCloseOperation(EXIT_ON_CLOSE);
20         setVisible(true);
21         takeImage();
22     }
23
24     public void paint(Graphics g){
25         if(bimage != null){
26             g.drawImage(bimage, 10, 30, null);
27         }
28     }
29
30     public void takeImage(){
31         try {
32             File file = new File(filename);
33             bimage = ImageIO.read(file);
34             System.out.println("width = " + bimage.getWidth()
35                               + " height = " + bimage.getHeight());
36             System.out.println("type: " + bimage.getType()
37                               + " field: " + BufferedImage.TYPE_3BYTE_BGR);
38         } catch (IOException e){
39             System.out.println("error! " + e);
40         }
41         repaint();
42     }
43
44     public static void main(String[] args) {
45         new Image1();
46     }
47 }
```

Console

<terminated> Image1 [Java Application] /Library/Java/JavaVirtualMachines/jdk-14.0.1.jdk/Cont  
width = 640 height = 480  
type: 5 field: 5

Figure 1. Loading and Displaying an Image.

The parameters are shown at Console in Figure 1. It can be seen that the size of the image is 640 pixels wide by 480 pixels high. Since the window is 500 X 500, only a part of the image is displayed. From the second line , we can see that the type of the image is 5 with the number. The number 5 indicates *TYPE\_3BYTE\_BGR*.

The result of displaying the image on the window is shown in Figure 2. This program cannot display the entire image because the window is smaller than the image.

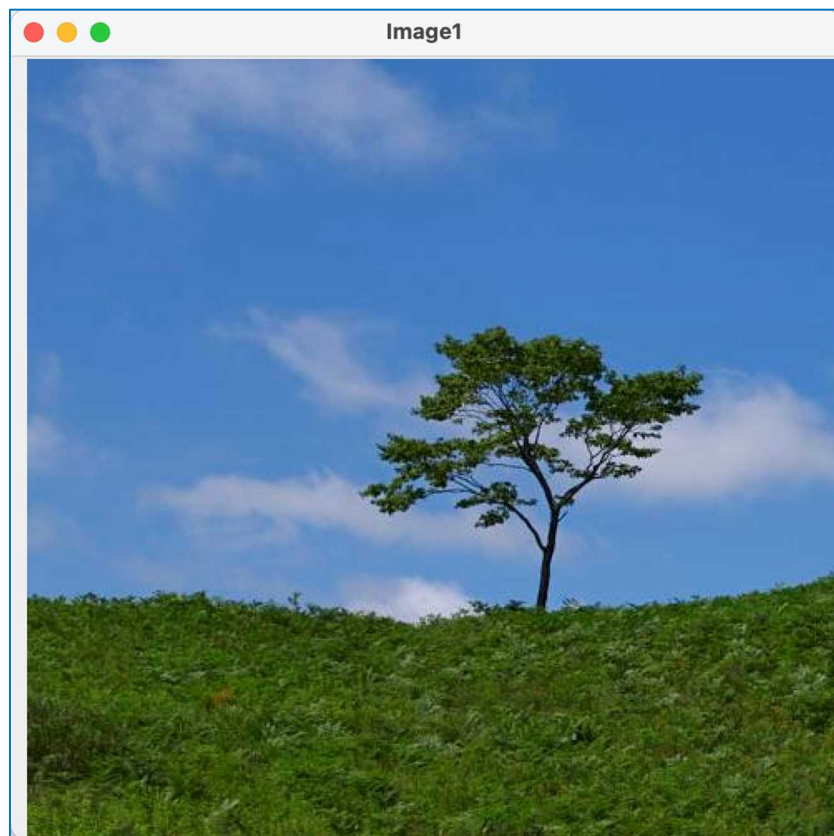


Figure 2. The result of displaying the image.

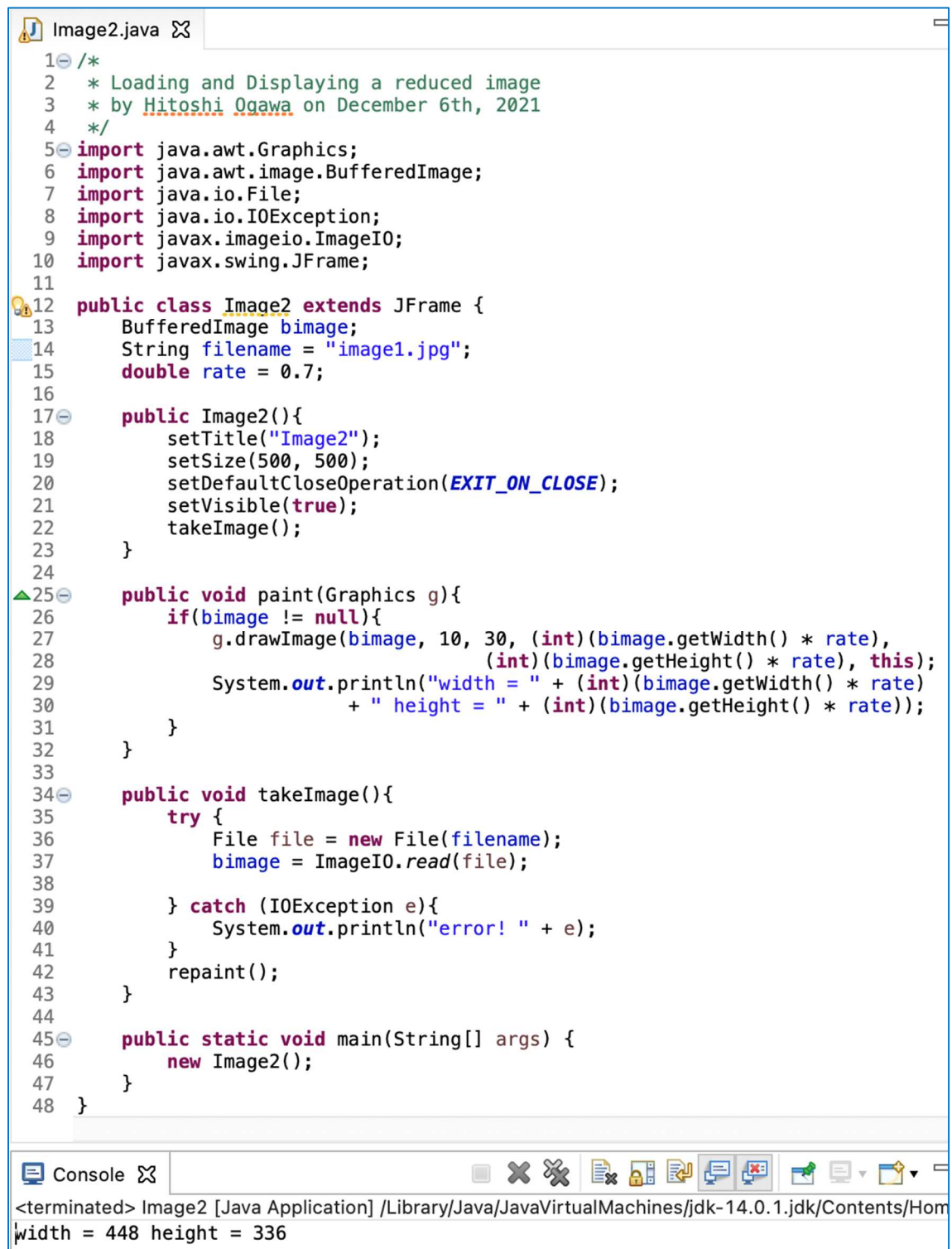
## (2) Scaling Display of Image

Since the entire image was not displayed using "Image1.jpg", the program "Image2.java" which displays reduced image using drawImage of 3.(B) is introduced in Figure 3.

In order to make the image 70% in size, 0.7 is substituted as the initial value for real variable "rate" (line 15). The size of the image is multiplied by "rate", and the result of calculation is cast to int type and used (lines 27 and 28). As a result, the image of size 448 X 336 is drawn.

Figure 4 shows the display the result of "Image2".

It is also possible to enlarge and display the image by this method.



```
1  /*
2  * Loading and Displaying a reduced image
3  * by Hitoshi Ogawa on December 6th, 2021
4  */
5  import java.awt.Graphics;
6  import java.awt.image.BufferedImage;
7  import java.io.File;
8  import java.io.IOException;
9  import javax.imageio.ImageIO;
10 import javax.swing.JFrame;
11
12 public class Image2 extends JFrame {
13     BufferedImage bimage;
14     String filename = "image1.jpg";
15     double rate = 0.7;
16
17     public Image2(){
18         setTitle("Image2");
19         setSize(500, 500);
20         setDefaultCloseOperation(EXIT_ON_CLOSE);
21         setVisible(true);
22         takeImage();
23     }
24
25     public void paint(Graphics g){
26         if(bimage != null){
27             g.drawImage(bimage, 10, 30, (int)(bimage.getWidth() * rate),
28                       (int)(bimage.getHeight() * rate), this);
29             System.out.println("width = " + (int)(bimage.getWidth() * rate)
30                               + " height = " + (int)(bimage.getHeight() * rate));
31         }
32     }
33
34     public void takeImage(){
35         try {
36             File file = new File(filename);
37             bimage = ImageIO.read(file);
38         } catch (IOException e){
39             System.out.println("error! " + e);
40         }
41         repaint();
42     }
43
44     public static void main(String[] args) {
45         new Image2();
46     }
47 }
48 }
```

Console

<terminated> Image2 [Java Application] /Library/Java/JavaVirtualMachines/jdk-14.0.1.jdk/Contents/Home  
width = 448 height = 336

Figure 3. An example of displaying the reduced image.



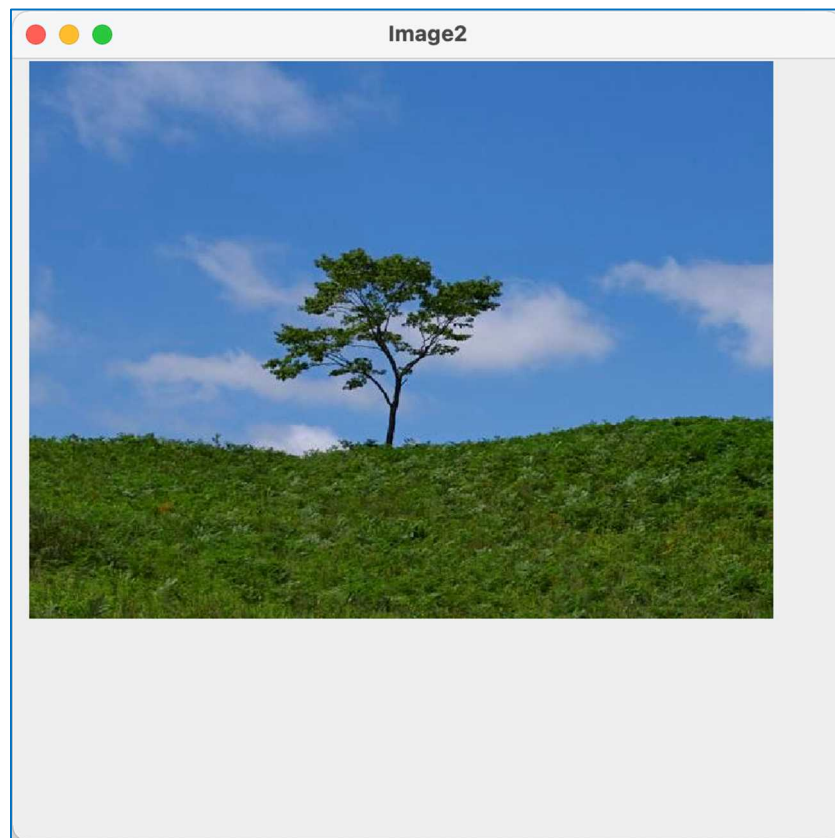


Figure 4. The result of “Image2”.

### (3) Image Segmentation and Display

If we use `drawImage` of 3.(C), a part of the image is selected and displayed. Figure 5 shows the result of changing lines 27 to 30 of “Image2.java” as follows.

```
g.drawImage(bimage, 10, 30, 310, 330, 150, 100, 450, 400, this);
```

From the `image1.jpg`, the part of the upper left coordinate (150, 100) and the lower right coordinate (450, 400) is cut out and drawn of the region of the upper left coordinate (10, 30) and the lower right coordinate (310, 330) of the window.

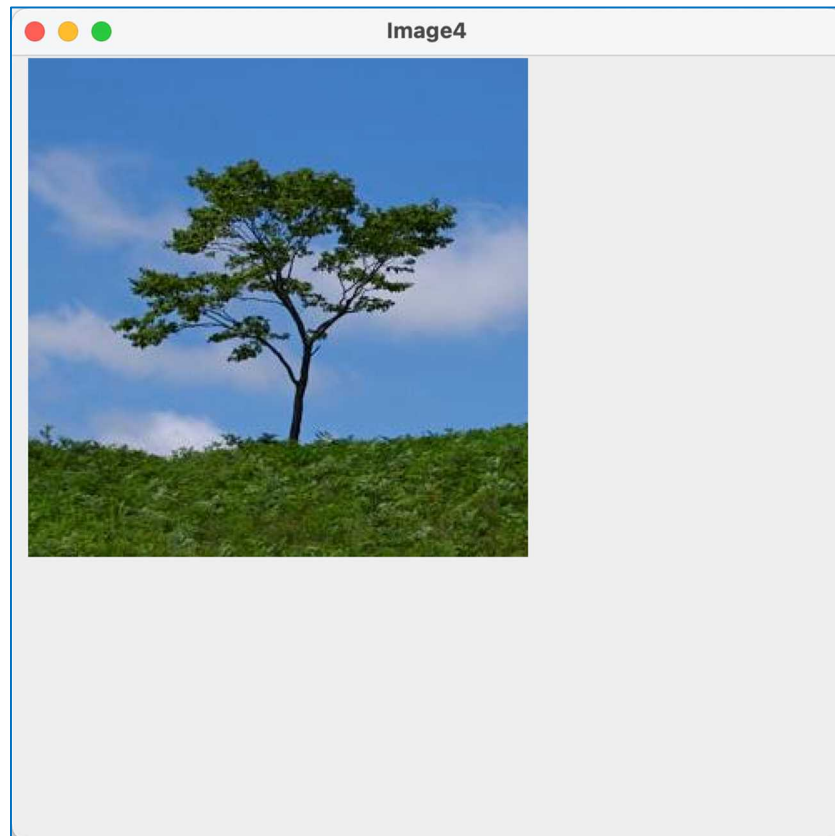


Figure 5. The result of Image segmentation and display.

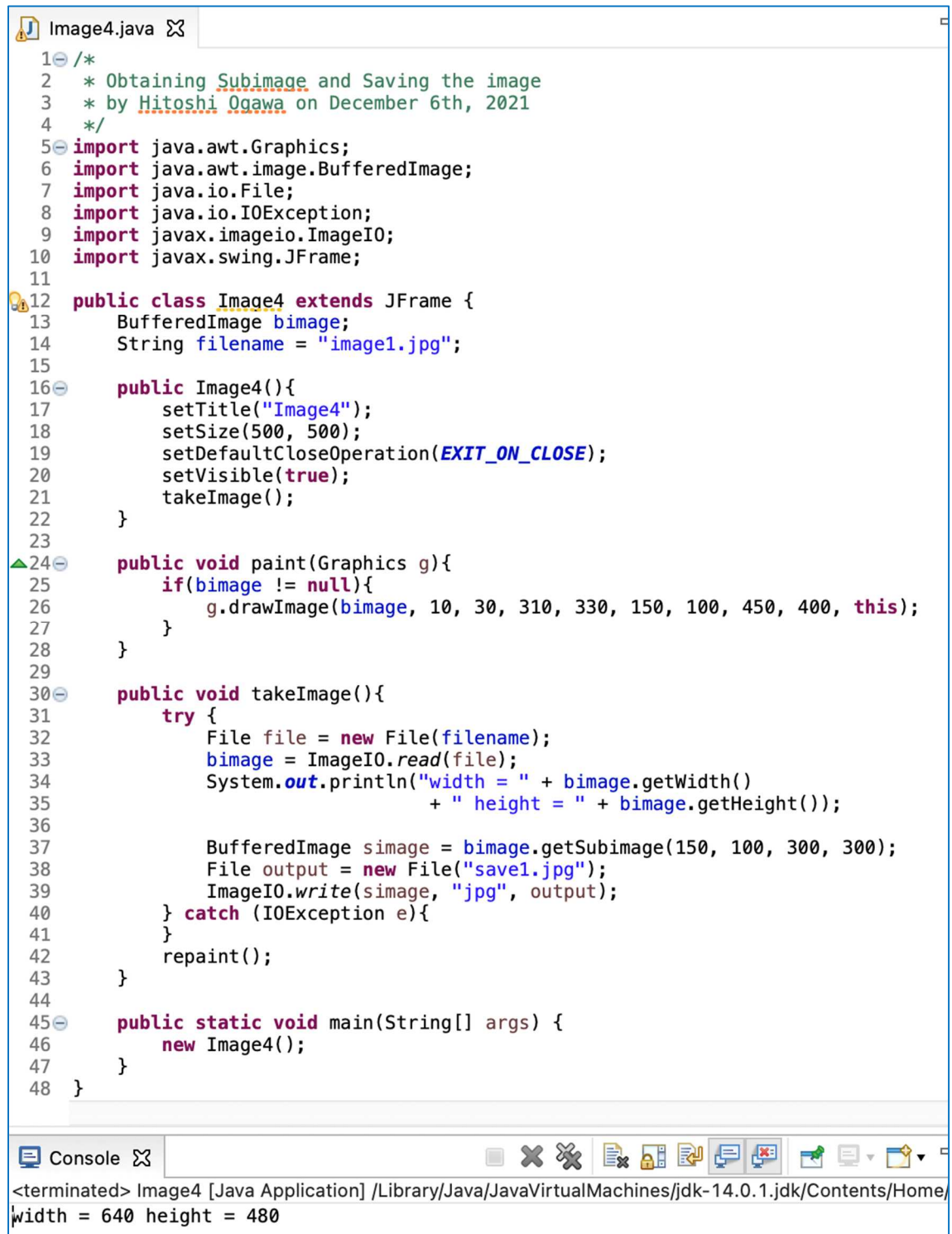
#### (4) Obtaining Subimage and Saving an Image

Figure 6 shows the program “Image4” that cuts out a part of “image1.jpg”, creates a new image, and saves it in the file “save1.jpg”.

The “getSubimage” of the BufferedImage class is used to get a part of the image. An image with a size of 300 X 300 with the coordinate (150, 100) of “image1.jpg” as the upper left coordinate is acquired and assigned to the BufferedImage variable “simage” (line 37).

The file with the name “save1.jpg” is created and assigned to File variable “output” (line 38). The “write” method of the “ImageIO” class is used to save the contents of “simage” to “output” in the form of “jpg” (line 39).

The file “save1.jpg” appears in “Package Explorer”. The contents are displayed and can be confirmed by double-clicking “save1.jpg” (Figure 7).



```
1  /*
2   * Obtaining Subimage and Saving the image
3   * by Hitoshi Ogawa on December 6th, 2021
4   */
5  import java.awt.Graphics;
6  import java.awt.image.BufferedImage;
7  import java.io.File;
8  import java.io.IOException;
9  import javax.imageio.ImageIO;
10 import javax.swing.JFrame;
11
12 public class Image4 extends JFrame {
13     BufferedImage bimage;
14     String filename = "image1.jpg";
15
16     public Image4(){
17         setTitle("Image4");
18         setSize(500, 500);
19         setDefaultCloseOperation(EXIT_ON_CLOSE);
20         setVisible(true);
21         takeImage();
22     }
23
24     public void paint(Graphics g){
25         if(bimage != null){
26             g.drawImage(bimage, 10, 30, 310, 330, 150, 100, 450, 400, this);
27         }
28     }
29
30     public void takeImage(){
31         try {
32             File file = new File(filename);
33             bimage = ImageIO.read(file);
34             System.out.println("width = " + bimage.getWidth()
35                               + " height = " + bimage.getHeight());
36
37             BufferedImage simage = bimage.getSubimage(150, 100, 300, 300);
38             File output = new File("save1.jpg");
39             ImageIO.write(simage, "jpg", output);
40         } catch (IOException e){
41         }
42         repaint();
43     }
44
45     public static void main(String[] args) {
46         new Image4();
47     }
48 }
```

Console

<terminated> Image4 [Java Application] /Library/Java/JavaVirtualMachines/jdk-14.0.1.jdk/Contents/Home/  
width = 640 height = 480

Figure 6. An example program of obtaining subImage and saving.

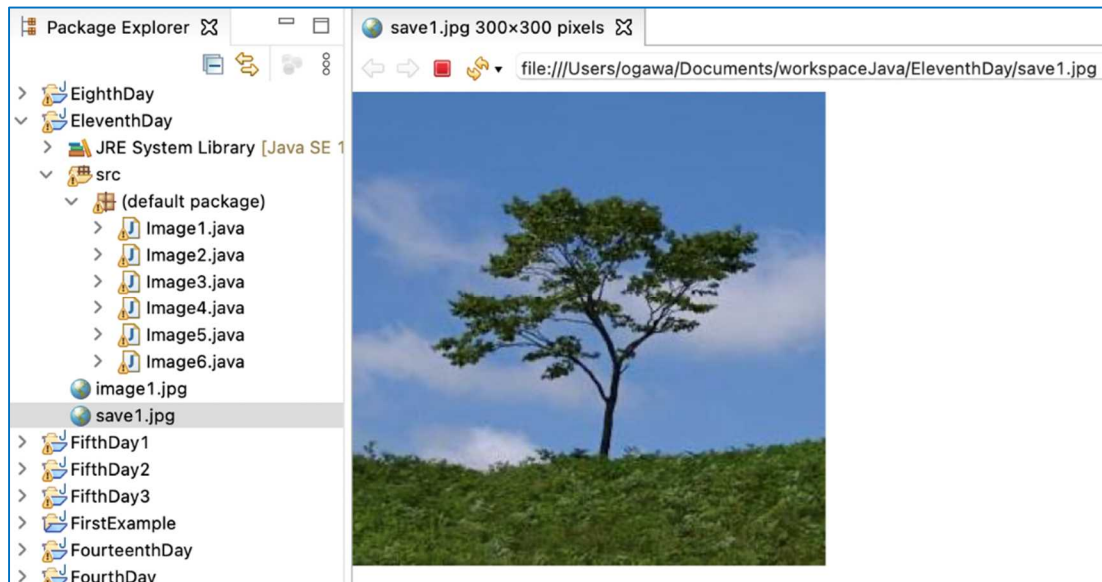


Figure 7. Image of “save1.jpg”.

#### (5) Image Segmentation by Mouse

Figure 8 shows a program that cuts out arbitrary areas using the `mousePressed` method and the `mouseDragged` method. Figure 9 shows an execution example of “Image5.java”.

In the field of “Image5.java”, the following variables are prepared (lines 16 to 18).

- ✓ The `BufferedImage` variables “bimage” and “getImage” which assign the original image and the cutout image, respectively.
- ✓ The File variable “filename” which substitutes the file name of the original image.
- ✓ Int array variable “rect” for cutting out: {{A, B}, {C, D}}
  - A: the x coordinate of the upper left corner of the cutout image
  - B: the y coordinate of the upper left corner of the cutout image
  - C: the x coordinate of the lower right corner of the cutout image
  - D: the y coordinate of the lower right corner of the cutout image

The width of the window is set to 700 to display the original image in full size, and the height of the window is set to 800 to draw the cutout image below the original image (line 22).

In the paint method, the following three are rendered.

- ✓ “bimage” whose upper left coordinate is (30, 30) is drawn, if bimage is available.

- ✓ “getImage” whose upper left coordinate is (30, hposition) is drawn, if getImage is available. The size of hposition is set to 520, since the vertical size of “bimage” is 480.
- ✓ A red rectangle is drawn according to the information of the array “rect”.

The following three methods are used in the inner class MyListener which inherits MouseAdapter.

mousePressed: The x and y coordinates when the mouse is pressed are assigned to rect[0][0] and rect[0][1], respectively.

mouseDragged: Every time the mouse is dragged, the x and y coordinates of the mouse are substituted into rect[1][0] and rect[1][1] respectively. “repaint” is executed for redrawing.

mouseReleased: A partial image indicated by a red rectangle is cut out and substituted into “getImage”. The x and y coordinates of the upper left of the red rectangle on the image are “rect[0][0] - 30” and “rect[0][1] - 30” respectively.

```

1  /*
2   * Image Segmentation by Mouse
3   * by Hitoshi Ogawa on December 6th, 2021
4   */
5  import java.awt.Color;
6  import java.awt.Graphics;
7  import java.awt.event.MouseAdapter;
8  import java.awt.event.MouseEvent;
9  import java.awt.image.BufferedImage;
10 import java.io.File;
11 import java.io.IOException;
12 import javax.imageio.ImageIO;
13 import javax.swing.JFrame;
14
15 public class Image5 extends JFrame{
16     BufferedImage bimage, getImage;
17     String filename = "image1.jpg";
18     int rect[][] = {{0, 0}, {0, 0}};
19
20     public Image5(){
21         setTitle("Image5");
22         setSize(700, 800);
23         setDefaultCloseOperation(EXIT_ON_CLOSE);
24
25         MyListener myListener = new MyListener();
26         addMouseListener(myListener);
27         addMouseMotionListener(myListener);
28
29         takeImage();
30         setVisible(true);
31     }
32

```

Figure 8. Image segmentation by mouse.

```

33 public void paint(Graphics g){
34     int hposition = 520;
35
36     g.clearRect(0, 0, 700, 800);
37     if(bimage != null){
38         g.drawImage(bimage, 30, 30, this);
39     }
40     if(getImage != null){
41         g.drawImage(getImage, 30, hposition, this);
42     }
43     g.setColor(Color.red);
44     g.drawRect(rect[0][0], rect[0][1],
45               rect[1][0] - rect[0][0], rect[1][1] - rect[0][1]);
46 }
47
48 public void takeImage(){
49     try {
50         File file = new File(filename);
51         bimage = ImageIO.read(file);
52     } catch (IOException e){
53     }
54     repaint();
55 }
56
57 public static void main(String[] args){
58     new Image5();
59 }
60
61 class MyListener extends MouseAdapter {
62     int cx, cy;
63     String str;
64
65     public void mousePressed(MouseEvent e) {
66         rect[0][0] = e.getX();
67         rect[0][1] = e.getY();
68     }
69
70     public void mouseReleased(MouseEvent e){
71         getImage = bimage.getSubimage(rect[0][0] - 30, rect[0][1] - 30,
72                                       rect[1][0] - rect[0][0], rect[1][1] - rect[0][1]);
73         repaint();
74     }
75
76     public void mouseDragged(MouseEvent e) {
77         rect[1][0] = e.getX();
78         rect[1][1] = e.getY();
79         repaint();
80     }
81 }
82 }

```

Figure 8. Image segmentation by mouse (continue).



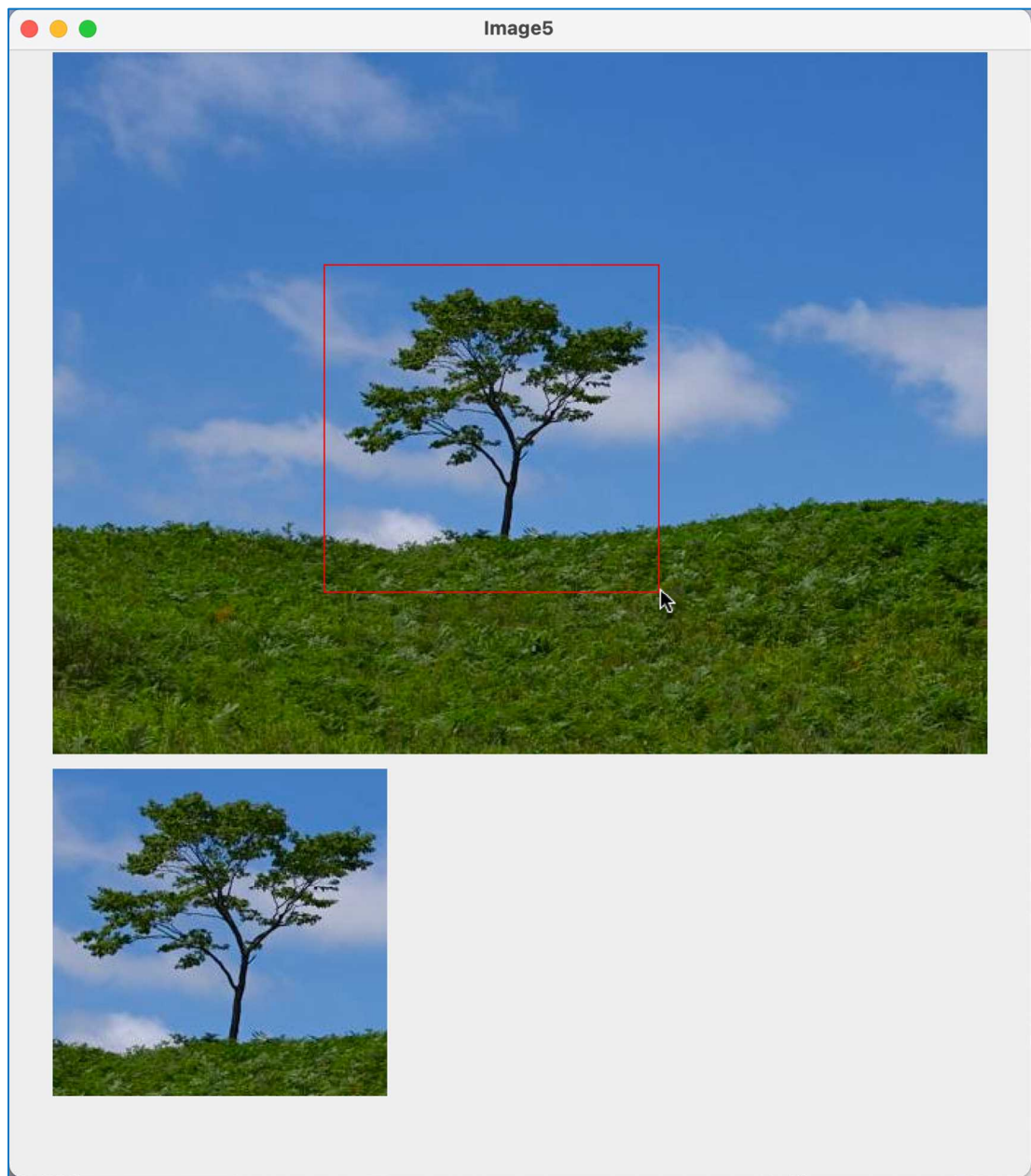


Figure 9. An execution example of Image5.java

## Appendix A.

How to download an image file from manaba+R and put it in Eclipse project.

1. Download an image file from manaba+R.

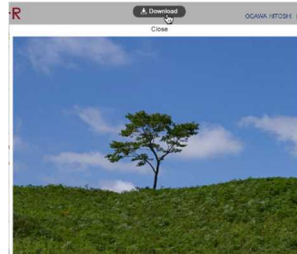
➤ Method 1

Right click (tap with two fingers for trackpad) on the image and select "Download Linked File"

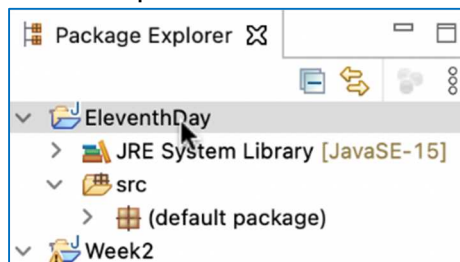


➤ Method 2

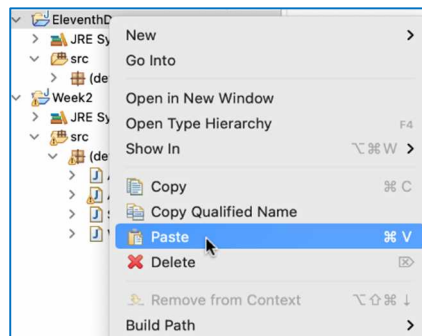
Click on the image and click "download"



2. Copy the image file from Download folder.
3. In Package Explorer of Eclipse, right click (tap with two fingers for trackpad) on the name of the project you want to put in.



4. Select "Paste".



5. Image file name can be confirmed in Package Explorer.

