JAVA DIP

digital image processing

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

This tutorial gives a simple and practical approach of implementing algorithms used in digital image processing. After completing this tutorial, you will find yourself at a moderate level of expertise, from where you can take yourself to next levels.

Audience

This reference has been prepared for the beginners to help them understand and implement the basic to advance algorithms of digital image processing in java.

Prerequisites

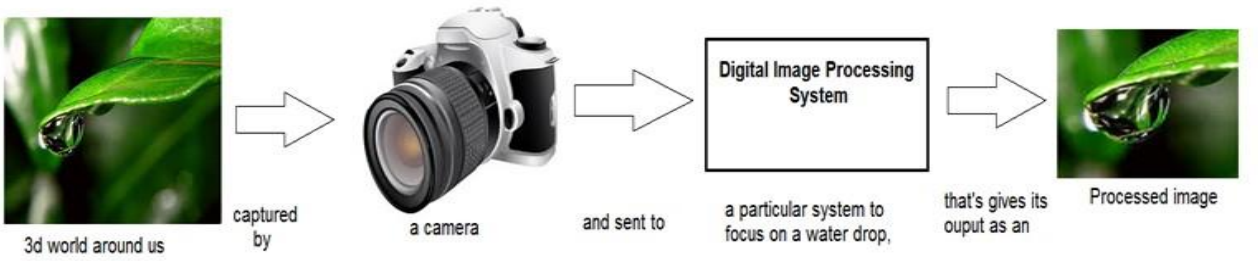
Before proceeding with this tutorial, you need to have a basic knowledge of digital image processing and Java programming language.

Disclaimer & Copyright

Contents

1. JAVA DIP — INTRODUCTION

Digital Image Processing (DIP) deals with manipulation of digital images using a computer. It is a subfield of signals and systems but focuses particularly on images. DIP focuses on developing a computer system that is able to perform processing on an image. The input of such system is a digital image. The system processes the image using efficient algorithms, and gives an image as an output.



Java is a high level programming language that is widely used in the modern world. It can support and handle digital image processing efficiently using various functions.

2. JAVA DIP — JAVA BUFFEREDIMAGE CLASS

Java BufferedImage class is a subclass of Image class. 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).

Constructors

This class supports three types of constructors. The first constructor constructs a new BufferedImage with a specified ColorModel and Raster.

```
BufferedImage(ColorModel cm, WritableRaster raster,

boolean isRasterPremultiplied, Hashtable<?,?> properties)
```

The second constructor constructs a BufferedImage of one of the predefined image types.

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

The third constructor constructs a BufferedImage of one of the predefined image types: TYPE_BYTE_BINARY or TYPE_BYTE_INDEXED.

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

Methods

Sr. No.	Methods
1	copyData(WritableRaster outRaster) It computes an arbitrary rectangular region of the BufferedImage and copies it into a specified WritableRaster.
2	getColorModel() It returns object of class ColorModel of an image.
3	getData() It returns the image as one large tile.
4	getData(Rectangle rect) It computes and returns an arbitrary region of the BufferedImage.

5	getGraphics() This method returns a Graphics2D, retains backwards compatibility.
6	getHeight() It returns the height of the BufferedImage.
7	getMinX() It returns the minimum x coordinate of this BufferedImage.
8	getMinY() It returns the minimum y coordinate of this BufferedImage.
9	getRGB(int x, int y) It returns an integer pixel in the default RGB color model (TYPE_INT_ARGB) and default sRGB colorspace.
10	getType() It returns the image type.

Example

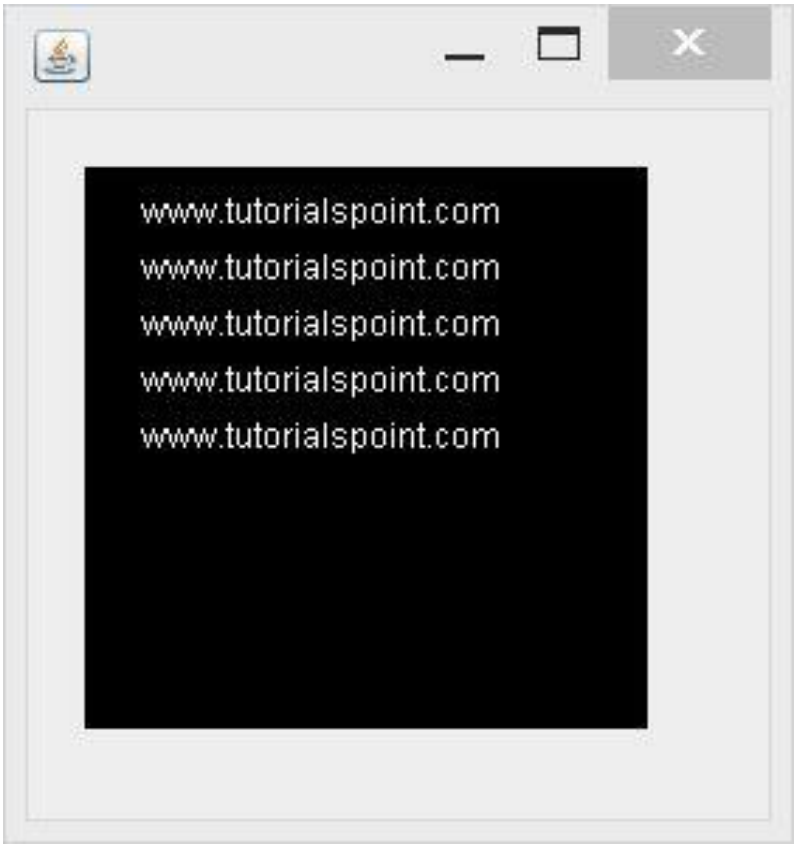
The following example demonstrates the use of java BufferedImage class that draws some text on the screen using Graphics Object:

```
import java.awt.Graphics;

import java.awt.Image;

import java.awt.image.BufferedImage;


import javax.swing.JFrame;

import javax.swing.JPanel;


public class Test extends JPanel {


   public void paint(Graphics g) {

      Image img = createImageWithText();
```

```
      g.drawImage(img, 20,20,this);
   }

   private Image createImageWithText(){
      BufferedImage bufferedImage = new
      BufferedImage(200,200,BufferedImage.TYPE_INT_RGB);
      Graphics g = bufferedImage.getGraphics();

      g.drawString("www.tutorialspoint.com", 20,20);
      g.drawString("www.tutorialspoint.com", 20,40);
      g.drawString("www.tutorialspoint.com", 20,60);
      g.drawString("www.tutorialspoint.com", 20,80);
      g.drawString("www.tutorialspoint.com", 20,100);
      return bufferedImage;
   }
   public static void main(String[] args) {
      JFrame frame = new JFrame();
      frame.getContentPane().add(new Test());

      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(200, 200);
      frame.setVisible(true);
   }

}
```

Output

When you execute the given code, the following output is seen:


www.tutorialspoint.com
www.tutorialspoint.com
www.tutorialspoint.com
www.tutorialspoint.com
www.tutorialspoint.com

3. JAVA DIP — DOWNLOADING / UPLOADING IMAGES

In this chapter we are going to see how you can download an image from internet, perform some image processing techniques on the image, and then again upload the processed image to a server.

Downloading an Image

In order to download an image from a website, we use Java class named URL, which can be found under **java.net** package. Its syntax is given below:

```
String website = "http://tutorialspoint.com";

URL url = new URL(website);
```

Apart from the above method, there are other methods available in class URL as described briefly:

Sr. No.	Methods
1	public String getPath() It returns the path of the URL.
2	public String getQuery() It returns the query part of the URL.
3	public String getAuthority() It returns the authority of the URL.
4	public int getPort() It returns the port of the URL.
5	public int getDefaultPort() It returns the default port for the protocol of the URL.
6	public String getProtocol() It returns the protocol of the URL.
7	public String getHost()

	It returns the host of the URL.

Example

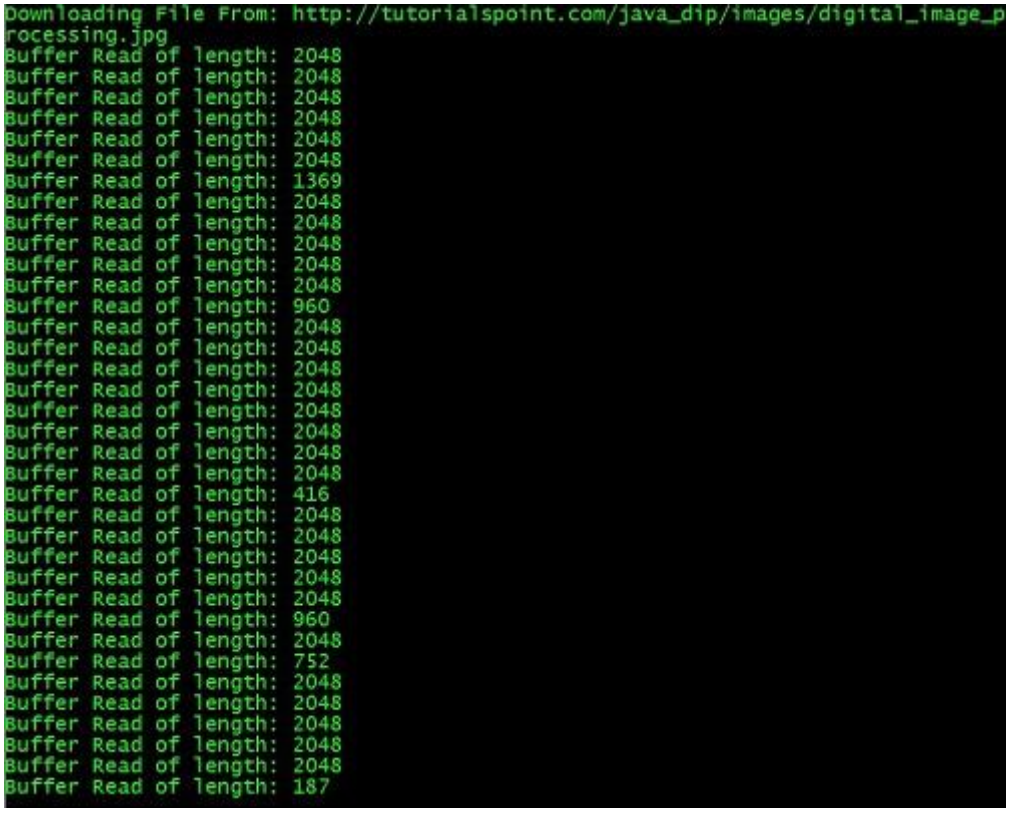
The following example demonstrates the use of java URL class to download an image from the internet:

```
import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.net.URL;


public class Download {


   public static void main(String[] args) throws Exception {

      try{

         String fileName = "digital_image_processing.jpg";

         String website =

               "http://tutorialspoint.com/java_dip/images/"+fileName;

         System.out.println("Downloading File From: " + website);

         URL url = new URL(website);

         InputStream inputStream = url.openStream();

         OutputStream outputStream = new FileOutputStream(fileName);

         byte[] buffer = new byte[2048];

         int length = 0;

         while ((length = inputStream.read(buffer)) != -1) {

            System.out.println("Buffer Read of length: " + length);

            outputStream.write(buffer, 0, length);

         }
```

```
         inputStream.close();

         outputStream.close();

        }catch(Exception e){

           System.out.println("Exception: " + e.getMessage());

        }

   }

}
```

Output

When you execute the given code, the following output is seen:



It would download the following image from the server.

Uploading an Image

Let us see how to upload an image to a webserver. We convert a BufferedImage to byte array in order to send it to server.

We use Java class ByteArrayOutputStream, which can be found under **java.io** package. Its syntax is given below:

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();

ImageIO.write(image, "jpg", baos);
```

In order to convert the image to byte array, we use toByteArray() method of ByteArrayOutputStream class. Its syntax is given below:

```
byte[] bytes = baos.toByteArray();
```

Apart from the above method, there are other methods available in the ByteArrayOutputStream class as described briefly:

Sr. No.	Methods
1	public void reset() This method resets the number of valid bytes of the byte array output stream to zero, so that all the accumulated output in the stream is discarded.
2	public byte[] toByteArray()

	This method creates a newly allocated Byte array. Its size would be the current size of the output stream and the contents of the buffer will be copied into it. It returns the current contents of the output stream as a byte array.
3	public String toString() Converts the buffer content into a string. Translation will be done according to the default character encoding. It returns the String translated from the buffer's content.
4	public void write(int w) It writes the specified array to the output stream.
5	public void write(byte []b, int of, int len) It writes len number of bytes starting from offset off to the stream.
6	public void writeTo(OutputStream outSt) It writes the entire content of this Stream to the specified stream argument.

Example

The following example demonstrates ByteArrayOutputStream to upload an image to the server:

Client Code

```
import javax.swing.*;

import java.net.*;

import java.awt.image.*;

import javax.imageio.*;

import java.io.*;

import java.awt.image.BufferedImage;

import java.io.ByteArrayOutputStream;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;
```

```
public class Client{

   public static void main(String args[]) throws Exception{

      Socket soc;

      BufferedImage img = null;

      soc=new Socket("localhost",4000);

      System.out.println("Client is running. ");

      try {

         System.out.println("Reading image from disk. ");

         img = ImageIO.read(new File("digital_image_processing.jpg"));

         ByteArrayOutputStream baos = new ByteArrayOutputStream();


               ImageIO.write(img, "jpg", baos);

         baos.flush();

         byte[] bytes = baos.toByteArray();

         baos.close();

         System.out.println("Sending image to server. ");


               OutputStream out = soc.getOutputStream();

         DataOutputStream dos = new DataOutputStream(out);

         dos.writeInt(bytes.length);

         dos.write(bytes, 0, bytes.length);

         System.out.println("Image sent to server. ");


         dos.close();

         out.close();

      }catch (Exception e) {

         System.out.println("Exception: " + e.getMessage());

         soc.close();
```

```
      }
         soc.close();
   }
}
```

Server Code

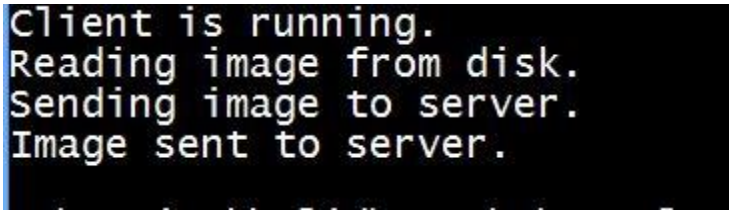
```
import java.net.*;
import java.io.*;
import java.awt.image.*;
import javax.imageio.*;
import javax.swing.*;

class Server {
   public static void main(String  args[]) throws Exception{
      ServerSocket server=null;
      Socket socket;
      server=new ServerSocket(4000);
      System.out.println("Server Waiting for image");

      socket=server.accept();
      System.out.println("Client connected.");
      InputStream in = socket.getInputStream();
      DataInputStream dis = new DataInputStream(in);

      int len = dis.readInt();
      System.out.println("Image Size: " + len/1024 + "KB");
      byte[] data = new byte[len];
      dis.readFully(data);
      dis.close();
```

```
         in.close();


         InputStream ian = new ByteArrayInputStream(data);

         BufferedImage bImage = ImageIO.read(ian);


         JFrame f = new JFrame("Server");

         ImageIcon icon = new ImageIcon(bImage);

         JLabel l = new JLabel();

         l.setIcon(icon);

         f.add(l);

         f.pack();

         f.setVisible(true);

   }
}
```

Output

Client Side Output

When you execute the client code, the following output appears on client side:



Server Side Output

When you execute the server code, the following output appears on server side:

After receiving the image, the server displays the image as shown below:


Server
tutorialspoint.com

4. JAVA DIP — IMAGE PIXELS

An image contains a two dimensional array of pixels. It is actually the value of those pixels that make up an image. Usually an image could be color or grayscale.

In Java, the BufferedImage class is used to handle images. You need to call getRGB() method of the BufferedImage class to get the value of the pixel.

Getting Pixel Value

The pixel value can be received using the following syntax:

```
Color c = new Color(image.getRGB(j, i));
```

Getting RGB Values

The method getRGB() takes the row and column index as a parameter and returns the appropriate pixel. In case of color image, it returns three values which are (Red, Green, Blue). They can be get as follows:

```
c.getRed();

c.getGreen();

c.getBlue();
```

Getting Width and Height of Image

The height and width of the image can be get by calling the getWidth() and getHeight() methods of the BufferedImage class. Its syntax is given below:

```
int width = image.getWidth();

int height = image.getHeight();
```

Apart from these methods, there are other methods supported in the BufferedImage class. They are described briefly:

Sr. No.	Methods
1	copyData(WritableRaster outRaster) It computes an arbitrary rectangular region of the BufferedImage and copies it into a specified WritableRaster.

2	getColorModel() It returns ColorModel of an image.
3	getData() It returns the image as one large tile.
4	getData(Rectangle rect) It computes and returns an arbitrary region of the BufferedImage.
5	getGraphics() This method returns a Graphics2D, but is here for backwards compatibility.
6	getHeight() It returns the height of the BufferedImage.
7	getMinX() It returns the minimum x coordinate of this BufferedImage.
8	getMinY() It returns the minimum y coordinate of this BufferedImage.
9	getRGB(int x, int y) It returns an integer pixel in the default RGB color model (TYPE_INT_ARGB) and default sRGB colorspace.
10	getType() It returns the image type.

Example

The following example demonstrates the use of java BufferedImage class that displays pixels of an image of size (10 x 10):

```
import java.awt.*;

import java.awt.image.BufferedImage;

import java.io.*;

import javax.imageio.ImageIO;
```

```
import javax.swing.JFrame;

class Pixel {

   BufferedImage  image;

   int width;

   int height;

   public Pixel() {

      try {

         File input = new File("blackandwhite.jpg");

         image = ImageIO.read(input);

         width = image.getWidth();

         height = image.getHeight();

         int count = 0;

         for(int i=0; i<height; i++){

            for(int j=0; j<width; j++){

               count++;

               Color c = new Color(image.getRGB(j, i));

               System.out.println("S.No: " + count + " Red: " + c.getRed() +

               " Green: " + c.getGreen() + " Blue: " + c.getBlue());

               }

            }

      } catch (Exception e) {}

   }

   static public void main(String args[]) throws Exception

   {

      Pixel obj = new Pixel();

   }

}
```

Output

When you execute the above example, it would print the pixels of the following image:

Original Image

Pixels Output

```
S.No: 1 Red: 0 Green: 0 Blue: 0
S.No: 2 Red: 0 Green: 0 Blue: 0
S.No: 3 Red: 0 Green: 0 Blue: 0
S.No: 4 Red: 0 Green: 0 Blue: 0
S.No: 5 Red: 0 Green: 0 Blue: 0
S.No: 6 Red: 0 Green: 0 Blue: 0
S.No: 7 Red: 0 Green: 0 Blue: 0
S.No: 8 Red: 0 Green: 0 Blue: 0
S.No: 9 Red: 0 Green: 0 Blue: 0
S.No: 10 Red: 0 Green: 0 Blue: 0
```

If you scroll down the ouput, the following pattern is seen:

```
S.No: 90 Red: 255 Green: 255 Blue: 255
S.No: 91 Red: 255 Green: 255 Blue: 255
S.No: 92 Red: 255 Green: 255 Blue: 255
S.No: 93 Red: 255 Green: 255 Blue: 255
S.No: 94 Red: 255 Green: 255 Blue: 255
S.No: 95 Red: 255 Green: 255 Blue: 255
S.No: 96 Red: 255 Green: 255 Blue: 255
S.No: 97 Red: 255 Green: 255 Blue: 255
S.No: 98 Red: 255 Green: 255 Blue: 255
S.No: 99 Red: 255 Green: 255 Blue: 255
S.No: 100 Red: 255 Green: 255 Blue: 255
```

End of ebook preview

If you liked what you saw…

Buy it from our store @ **https://store.tutorialspoint.com**