การมองเห็นของคอมพิวเตอร์

แบบฝึกหัดเขียนโปรแกรมชุดที่ 1:Enter the Image

แบบฝึกหัดนี้เป็นจำพวกเรียนรู้กลางอากาศ คือให้ทำไป เข้าใจไปเรื่อย ๆ เรียนรู้สิ่งใหม่ไปตามทาง เป็นทั้งแบบฝึกหัดและการสอนไปในตัว

ส่วนที่ 1 การอ่านไฟล์ข้อมูลภาพ

1. อ่านค่าความกว้างและความสูงของภาพ [ImageReading]

จงเขียนโปรแกรมที่ทำการอ่านข้อมูลพื้นฐานของไฟล์รูปภาพ Classical Building ซึ่งเป็นภาพแบบเฉดเทา 8 บิต จากนั้นให้พิมพ์ความกว้างและความสูงของภาพออกมาทางเทอร์มินอล [ที่มาของภาพ]

หมายเหตุ โดยปรกติ การใช้ `System.out.print` มันจะให้ผลลัพธ์ออกมาทางเทอร์มินอลอยู่แล้ว เพียงแต่บางคนอาจจะยังไม่ทราบว่าเทอร์มินอลคืออะไร

ในเวลานี้ เราต้องทำความรู้จักกับไลบรารีในภาษาจาวาที่สำคัญทางด้านการอ่านภาพก่อน ซึ่งจะมีทั้งแบบที่เป็นไลบรารีมาตรฐานของภาษาจาวา คือมาพร้อมกับการติดตั้ง JDK ด้วยเสมอ และก็มีไลบรารีเสริมที่สามารถดาวน์โหลดมาติดตั้ง เช่น OpenCV ซึ่งไลบรารีเสริมที่มีชื่อเสียงอย่าง OpenCV นี้ มักจะมีความสามารถสูงกว่าไลบรารีมาตรฐานของภาษาจาวามาก อย่างไรก็ตาม มันก็มีความซับซ้อนที่สูงขึ้นมาด้วย

ในตอนต้นของการเรียนรู้ เราต้องการมุ่งเน้นที่ความเข้าใจเรื่องรูปภาพ และหลักการทั่วไปในการประมวลผลภาพ ดังนั้นเราจะเรียนรู้จากไลบรารีมาตรฐานของภาษาจาวาซึ่งมีสองส่วนด้วยกันคือ `javax.imageio` และ `java.awt.image`

Follow Me: import คลาสสำคัญ

เริ่มแรกมา ให้สร้างไฟล์และคลาส `ImageReading` ขึ้นมา และให้ `import` คลาสสำคัญสี่ตัวดังนี้

```
import java.awt.image.BufferedImage;
import java.io.File;
```

```
import java.io.IOException;
import javax.imageio.ImageIO;
```

เรื่องน่ารู้: คลาสพวกนั้นมีประโยชน์อะไร

เมื่อเราอ่านรูปภาพเข้ามา เราจะเก็บข้อมูลภาพไว้กับวัตถุชนิด `BufferedImage` ซึ่งตัวคลาส `BufferedImage` เป็นสิ่งที่อำนวยความสะดวกในการจัดการข้อมูลภาพอย่างมีประสิทธิภาพ ดังนั้นอันดับแรกเราจึงต้องเตรียมตัวแปรอ้างอิงวัตถุชนิดนี้ไว้

เนื่องจากกลไกการอ่านภาพ ก็ทำผ่านกลไกของไฟล์ เราจึงต้องอาศัยวัตถุชนิด `File` และสิ่งที่เป็นผลพวงตามมาก็คือ การพยายามเปิดไฟล์อาจจะนำไปสู่เหตุการณ์ผิดปรกติ `IOException` ทำให้เราต้องอ้างอิงถึงมันในการทำ `try-catch`

ท้ายที่สุดหลังจากเข้าถึงไฟล์แบบทั่วไปได้แล้ว เราต้องทำการอ่านไฟล์นั้น*ในฐานะไฟล์รูปภาพ* ซึ่งการอ่านไฟล์รูปภาพทำได้โดยเมธอดภายใน `ImageIO` และสิ่งที่คืนกลับมาก็คือวัตถุชนิด `BufferedImage` ไว้ให้เราใช้งานต่อไปนั่นเอง

Follow Me: อ่านไฟล์ภาพเข้ามาในวัตถุ `BufferedImage`

เราสามารถนำความรู้ต่าง ๆ มาสร้างเป็นเมธอด `main` ที่อ่านภาพได้ดังนี้

```
public static void main(String[] args) {
  BufferedImage img = null;
  try {
    File imgFile = new File("C:/classical_building.png");
    img = ImageIO.read(imgFile);
  } catch(IOException ex) {
    System.err.println("Error loading image");
    return;
  }
}
```

หมายเหตุ ตรงชื่อไฟล์ (สีแดง) ต้องเปลี่ยนไปตามชื่อไฟล์และโฟลเดอร์ที่เราบันทึกไว้จริง

เรื่องควรทำ ลองพิจารณาโค้ดที่ให้ไว้ กับเรื่องน่ารู้ที่เราอ่านมาก่อนหน้า แล้วเชื่อมโยงหน้าที่ของวัตถุต่าง ๆ กับโค้ดให้ได้

แต่เดี๋ยวก่อน สิ่งที่เราต้องการคือความกว้างและความสูงของภาพ แบบนี้แสดงว่างานเรายังไม่เสร็จแน่นอน **ว่าแต่เราจะเอาขนาดของภาพมาได้จากไหน?** ลองสังเกตดูเราก็จะพบว่า สิ่งที่ใกล้เคียงกับความเป็นภาพที่สุดก็น่าจะเป็นตัวแปร **`img`** ดังนั้นเราน่าจะอ่านขนาดภาพมาจากตัวแปร **`img`** นี้ แต่จะเขียนว่าอะไรดี

เรื่องน่ารู้: เอกสารกับการค้นหาข้อมูลที่เกี่ยวข้อง

`BufferedImage` เป็นคลาสที่มีประโยชน์มาก มากชนิดที่ว่าจะให้อธิบายให้หมดในวิชานี้คงยาก เนื่องจากเรายังมีเนื้อหาที่ต้องศึกษาอีกมากยิ่งกว่า แต่ในชีวิตจริง ความรู้ของเราต้องขยายออกไป หยุดอยู่แค่ในชั้นเรียนไม่ได้ ดังนั้นเราต้องมาดูว่า โดยทั่วไปแล้วเราจะศึกษาหาความรู้เกี่ยวกับความสามารถของคลาสแต่ละอันได้จากไหน

ในกรณีนี้ เป็นคลาสจากไลบรารีมาตรฐานของจาวาเอง เราสามารถดูได้จากเอกสารที่บริษัท `Oracle` จัดไว้ให้ได้ (ถ้าเป็นไลบรารีจากองค์กรหรือบุคคลอื่น เราก็มักจะต้องอ่านจากเอกสารขององค์กรหรือบุคคลนั้น) ซึ่งเอกสารมีอยู่สองกลุ่มหลัก คือ เอกสารสอนให้ความรู้ `(Tutorial)` และเอกสารอ้างอิง `(Reference)`

เอกสารสอนให้ความรู้มักจะมีเนื้อหาเป็นการแนะนำอธิบายการใช้งานและยกตัวอย่างการประยุกต์ใช้ ซึ่งเราอาจจะไม่ต้องมีความรู้ในเนื้อหานั้นก็อ่านรู้เรื่อง ดูคล้ายหนังสือเรียนในรูปแบบหนึ่ง

แต่เอกสารอ้างอิงของไลบรารีมักจะให้รายละเอียดที่เอาไว้ใช้หาข้อมูลที่ค่อนข้างเจาะจง และเรามักจะต้องมีความรู้พื้นฐานอยู่ในระดับหนึ่งจึงจะอ่านเข้าใจ เอกสารอ้างอิงจะคล้ายกับพจนานุกรม คือเราต้องรู้ภาษาอังกฤษมาก่อน เข้าใจวิธีการเขียน รู้ว่าคำนาม คำกิริยาคืออะไร ไม่เช่นนั้นจะไม่สามารถใช้พจนานุกรมให้เกิดประโยชน์ได้เต็มที่ เอกสารอ้างอิงการของไลบรารีก็มักจะเป็นเช่นนั้น

สำหรับเอกสารสอนให้ความรู้จากออราเคิลสามารถเข้าถึงได้ที่ลิงค์นี้[1] ส่วนเอกสารอ้างอิงสามารถเข้าถึงได้ที่ลิงค์นี้[2]

[1] ค้นหาด้วยคำว่า Oracle Java Tutorial

[2] ค้นหาด้วยคำว่า Java SE API (SE ย่อมาจาก Standard Edition)

เรื่องน่ารู้: API กับรายละเอียดของคลาส BufferedImage

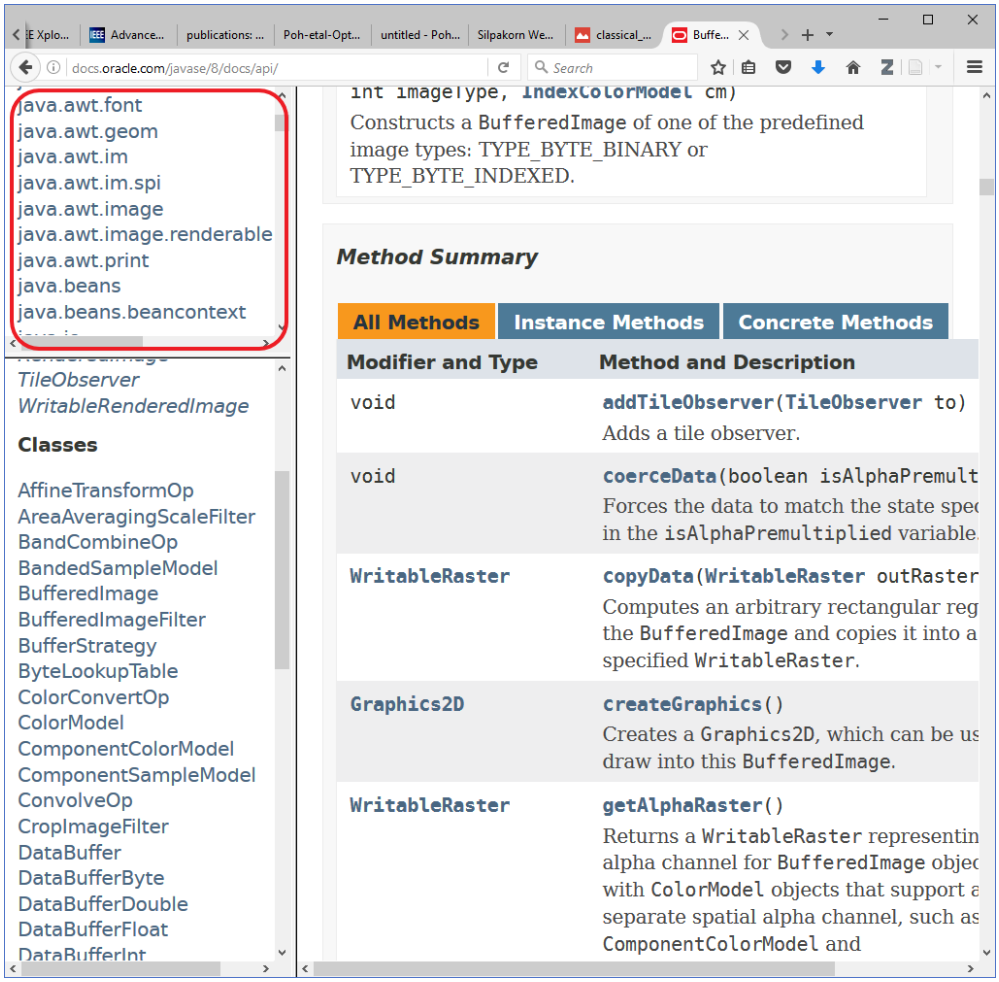
วกกลับมาเรื่องของ **BufferedImage** ในที่นี้เราจะใช้ความสามารถของมันรวมถึงคลาสอื่น ๆ ที่เกี่ยวข้องผ่านส่วนเชื่อมต่อ **(interface)** ซึ่งส่วนเชื่อมต่อนี้เป็นการเชื่อมต่อผ่านโปรแกรมประยุกต์ ส่วนเชื่อมต่อนี้จึงมีชื่อเรียกในโลกของการเขียนโปรแกรมว่า **Application Programming Interface (API)** ซึ่งเมื่อเราเปิดลิงค์เอกสารอ้างอิงแล้ว เราจะพบรายการของแพ็คเกจในจาวาจำนวนมากอยู่ทางมุมบนซ้าย (**รูปที่ 1**)

แต่เรารู้ว่าชื่อแพ็คเกจของ **BufferedImage** คือ **java.awt.image** (ชื่อแพ็คเกจในมาตรฐานจาวาจะเป็นตัวพิมพ์เล็ก) เราจึงเลือกเข้าใช้งานลิงค์ของแพ็คเกจดังกล่าว

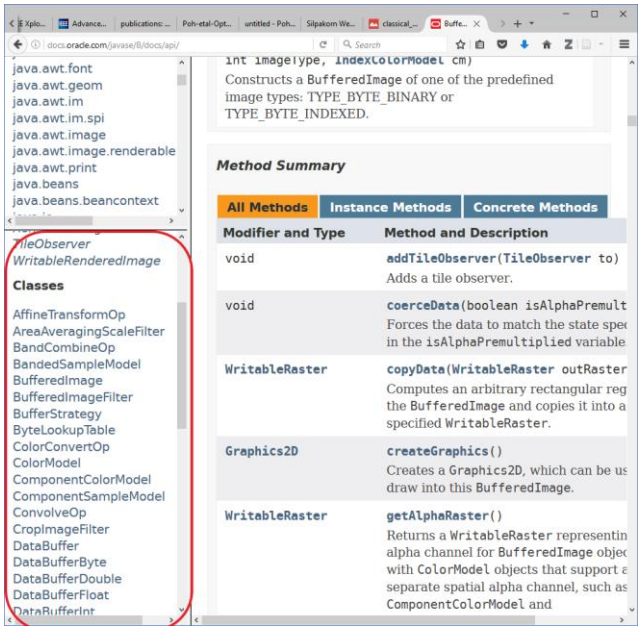
เมื่อเข้าไปแล้ว เราจะเห็นชื่อของคลาสและอินเตอร์เฟซอยู่ทางด้านซ้ายล่างของจอภาพ (**รูปที่ 2**) ให้เราเลือกไปที่ **BufferedImage** เราก็จะเห็นรายละเอียดของคลาส **BufferedImage** อยู่ทางขวาของเว็บบราวเซอร์ และเนื้อหาที่เราสนใจในที่นี้อยู่ในส่วนสรุปเมธอด **(Method Summary)** ดังแสดงใน **รูปที่ 3**

ถึงตาของคุณแล้ว: เรียกใช้เมธอดที่เกี่ยวข้องกับงานของเรา

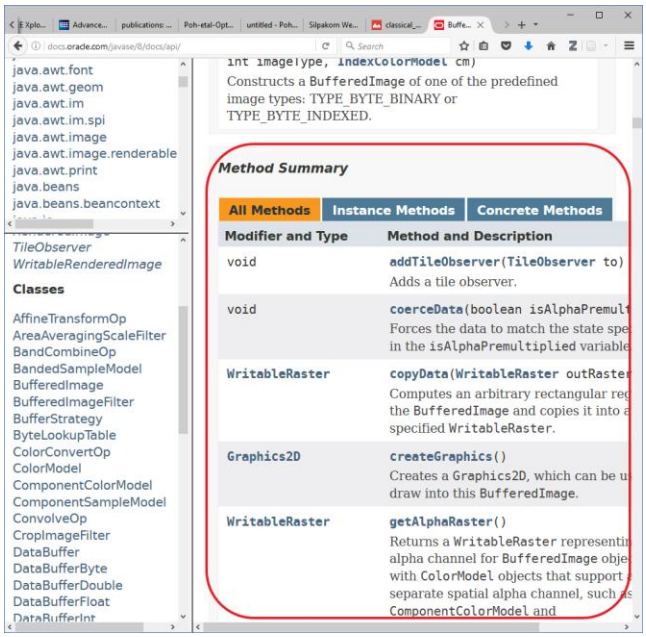
จากส่วนสรุปเมธอดของ **BufferedImage** ลองค้นหาอะไรที่เกี่ยวกับความกว้าง **(width)** และความสูง **(height)** แล้วลองพิจารณาหาทางพิมพ์ค่าความกว้างและความสูงออกมาเป็นผลลัพธ์ให้ได้



รูปที่ 1 *รายการแพ็คเกจในเอกสารเอพีไอ*



รูปที่ 2 *รายชื่อคลาสและอินเตอร์เฟซในแพ็คเกจที่เราเลือก*



รูปที่ 3 *ข้อมูลสรุปเกี่ยวกับเมธอด*

2. การอ่านไฟล์รูปภาพเพื่อเก็บไว้ในอาเรย์สองมิติ [PixelReader]

เมื่อทราบความกว้างและความสูงของภาพแล้ว เราต้องการอ่านค่าเฉดเทามาเก็บไว้ในอาเรย์สองมิติเพื่อความสะดวกในการประมวลผลต่อไป เพื่อความง่ายในการเรียนรู้พื้นฐาน เราจะใช้ภาพที่มีเพียงสีเทาเพียงสีเดียว และเราจะอ่านค่าเฉดเทาของพิกเซลในภาพมาเก็บไว้ในอาเรย์ `int[][]`

โดยเรามีข้อกำหนดเพิ่มเติมด้วยว่า เราจะสร้างเป็นเมธอด `loadToArray` ซึ่งมีลายเซ็น (signature) ของเมธอดเป็นดังนี้

```
int[][] loadToArray(BufferedImage img)
```

นั่นคือเมธอดนี้จะรับ `BufferedImage` เป็นอินพุตผ่านพารามิเตอร์ จากนั้นจะ

(1) อ่านค่าความกว้างและความสูงของ `img` มาเก็บไว้ในตัวแปร `width` และ `height`

(2) สร้างอาเรย์สองมิติชื่อ `arImg` ด้วย `new int[height][width]`

(3) วนลูปอ่านค่าพิกเซลใน `img` มาทีละแถวและนำค่าที่ได้ไปใส่ไว้ใน `arImg`

(4) คืนค่า (return) `arImg` มาในฐานะผลลัพธ์ของเมธอด

Follow Me: เข้าถึงค่าพิกเซลผ่าน `java.awt.image.Raster`

วัตถุจากคลาส `Raster` เป็นสิ่งที่อำนวยความสะดวกในการอ่านค่าพิกเซลจากภาพ ซึ่งเราสามารถเรียกใช้วัตถุนี้ได้จากภาพที่อยู่ในรูปแบบ `BufferedImage` (ในที่นี้ก็คือ `img`) ซึ่งเราสามารถเขียนในเมธอด `loadToArray` ว่า

```
Raster raster = img.getRaster();
```

[อย่าลืม `import` คลาสเข้ามาด้วย]

แต่ตัว `Raster` นี้จะอ่านภาพแบบเฉดเทาหรือภาพสีก็ได้ ซึ่งถ้าจะอ่านภาพสีแบบ RGB ซึ่งมีสามช่องสัญญาณ (channel) `Raster` ก็จะอ่านมาทีละสามค่า แต่ถ้าเป็นแบบเฉดเทาซึ่งมีหนึ่งช่องสัญญาณ ก็จะอ่านมาทีละค่า

วิธีควบคุมให้ Raster อ่านค่าเฉดเทามาทีละค่าตามที่ควรเป็น ทำได้โดยใช้ที่พักข้อมูล (buffer) แบบ int[] ซึ่งมีขนาดหนึ่งช่อง เราสามารถสร้างที่พักข้อมูลได้ในลักษณะดังโค้ดข้างล่างนี้

```
int[] pixelBuffer = new int[1];
```

ตอนนี้วัตถุ Raster ก็มีแล้ว ที่พักข้อมูลสำหรับอ่านค่าก็มีแล้ว เราสามารถอ่านค่าพิกเซลจากตำแหน่งที่ต้องการได้ สมมติว่าเราต้องการอ่านค่าจากคอลัมน์ที่ 8 แถวที่ 12 เราสามารถเขียนโค้ดในลักษณะดังแสดงข้างล่างนี้ (คอลัมน์และแถวเริ่มนับจาก 0)

```
raster.getPixel(8, 12, pixelBuffer);
```

เพียงเท่านี้เราก็ได้ค่าพิกเซลที่ต้องการมาเก็บไว้ในที่พักข้อมูล และเราสามารถอ่านค่าพิกเซลออกมาจากที่พักข้อมูลได้เหมือนกับการอ่านค่าอาเรย์ทั่วไป และในกรณีของปัญหานี้ เราต้องการคัดลอกค่ามาเก็บไว้ที่ arImg สิ่งที่เราควรจะเขียนก็คือ

```
arImg[12][8] = pixelBuffer[0];
```

เรื่องน่ารู้: ข้อควรระวังในการอ่านค่าและเขียนค่า

สิ่งที่เราจะสับสนเป็นอันดับแรก ๆ มักจะเป็นเรื่องลำดับการบอกตำแหน่ง สังเกตให้ดีว่าตอนที่เราใช้เมธอด getPixel เราใช้ลำดับเป็นแบบ (x, y) คือคอลัมน์และแถวตามลำดับ แต่พอเป็นการเขียนค่าในอาเรย์ arImg เราทำเป็น [row][column] ซึ่งเป็นแถวและคอลัมน์ ตามลำดับ

ถึงตาของคุณแล้ว: อ่านค่าจาก Raster มาทีละพิกเซล และเก็บข้อมูลลงใน arImg ให้ครบถ้วน

ในตัวอย่างที่แล้ว เราอ่านค่าจากตำแหน่งเดียวคือคอลัมน์ 8 และแถว 12 แต่ที่จริงเราต้องอ่านค่าทุกพิกเซลจากแถวที่ 0 ถึง height-1 และจากคอลัมน์ที่ 0 ถึง width-1 ดังนั้นเราต้องตั้งลูปสองชั้นขึ้นมาเพื่อวนอ่านค่าจาก Raster และเขียนค่าลงไปในอาเรย์ ขอให้ลองใช้โครงสร้างลูปที่ให้ไปนี้ เพื่อดัดแปลงให้โปรแกรมจัดการค่าพิกเซลให้ครบถ้วน

```
... ... เติมโค้ดสำหรับอ่านขนาดภาพและเตรียมอาเรย์ไว้ที่นี้
```

```
Raster raster = img.getRaster();
int[] pixelBuffer = new int[1];
for(int row = 0; row < height; ++row) {
  for(int col = 0; col < width; ++col) {
     ... ... ... เติมโค้ดลงไปตรงนี้เพื่ออ่านและคัดลอกค่าพิกเซล
  }
}
... ... คุณยังเหลืองานอีกอย่างหนึ่ง จำได้หรือเปล่าว่าเมธอดนี้สุดท้ายต้องทำอะไร
```

Follow Me: ทดสอบว่าโปรแกรมเราทำงานถูกต้องจริงหรือไม่

เมธอด `loadToArray` ทำให้เราสามารถโหลดรูปภาพมาเก็บไว้ในอาเรย์ ปัญหาคือแล้วเราจะรู้ได้อย่างไรว่าเมธอดที่เราเขียนมานั้นทำงานถูกต้องจริง เมื่อเราพบกับเหตุการณ์เช่นนี้ เราควรจะเริ่มจากสิ่งที่เราทราบว่าคำตอบคืออะไร เช่นในงานนี้ เราจะใช้ภาพที่มีเพียงสีเทาเพียงสีเดียว ซึ่งเราดูค่าเฉดเทาในภาพด้วยโปรแกรม XnView[3] เมื่อทราบค่าเฉดเทาแล้ว ให้ลองเรียกใช้ `loadToArray` และเช็คค่าในอาเรย์ดูว่ามันมีค่าตามที่เราคาดไว้หรือไม่

ด้วยแนวคิดดังกล่าว เราสามารถเขียนเมธอด `main` ใหม่ให้เป็นดังข้างล่างนี้

```
BufferedImage img = null;
try {
  File imgFile = new File("C:/gray.png");
  img = ImageIO.read(imgFile);
} catch(IOException ex) {
  System.err.println("Error loading image");
  return;
}
ImageReading reader = new ImageReading();
int[][] arImg = reader.loadToArray(img);
System.out.println(arImg[0][0]);
```

จากนั้น ลองสังเกตดูว่าค่าพิกเซลที่พิมพ์ออกมาทางเทอร์มินอลเป็นไปตามที่คาดไว้หรือไม่ เมื่อผลลัพธ์จากภาพง่าย ๆ นี้ถูกต้องแล้ว ลองทดสอบกับภาพที่ซับซ้อนขึ้นอย่าง

[3] เมื่อโหลดภาพใน XnView แล้วให้กด Ctrl+Shift+I (ตัวไอ) เพื่อเรียกดูค่าสีหรือเฉดเทาในภาพ หรือไปที่เมนู View->Display Color Information จากนั้นเลื่อนเคอร์เซอร์เมาส์ไปตำแหน่งที่พิกเซลที่เราต้องการทราบค่า

`classical_building` ดูอีกครั้ง โดยเลือกตำแหน่งพิกเซลที่สนใจมาหลาย ๆ จุดและเปรียบเทียบค่าในอาเรย์และค่าที่อ่านได้จาก `XnView`[4]

เช่น ถ้าจุดแรกเราเลือกตำแหน่งแถวที่ 7 คอลัมน์ที่ 5 ส่วนจุดที่สองเราเลือกแถวที่ 6 คอลัมน์ที่ 2 เราจะทดสอบการพิมพ์ค่าออกมาด้วยคำสั่ง

```
System.out.println(arImg[7][5]);
System.out.println(arImg[6][2]);
```

การทดสอบค่าพิกเซลในตำแหน่งที่สนใจเป็นวิธีที่สำคัญกับปัญหาบางประเภท โดยเฉพาะตอนที่เราพบว่าค่ามันดูผิดไปจากที่ควรเป็นในบางจุด อย่างไรก็ตาม วิธีที่อาจจะถือว่าง่ายที่สุดสำหรับการทดสอบการอ่านภาพลงในอาเรย์ ก็คือการนำข้อมูลในอาเรย์นั้นไปบันทึกลงเป็นภาพอีกไฟล์หนึ่ง ถ้าไฟล์ข้อมูลเข้ากับผลลัพธ์เหมือนกันทุกประการ ก็เรียกได้ว่ามีเหตุให้ควรเชื่อว่าเมธอดของเราทำงานถูกต้องจริง

เรื่องน่ารู้: ในการสร้างอาเรย์เก็บพิกเซล จะเลือกอะไรดีระหว่าง `int[height][width]` กับ `int[width][height]` หรือว่าที่จริงมันเหมือนกัน?

ในประเด็นการเก็บข้อมูล การเขียนทั้งสองแบบจะให้ผลเหมือนกัน ทำงานได้ถูกต้องทั้งคู่ อย่างไรก็ตาม เรานิยมจะเก็บแบบใช้แถวเป็นหลัก (`row major`) เพราะมันสอดคล้องกับรูปแบบตำแหน่งการเก็บข้อมูลในดิสก์ และอาเรย์สองมิติในจาวา (รวมถึงหลาย ๆ ภาษา) ก็เก็บแบบใช้แถวเป็นหลักคือข้อมูลในแถวเดียวกันจะอยู่ติดกันไปจนจบแถว แล้วจึงต่อด้วยข้อมูลในแถวอื่น ๆ

นอกจากนี้การเก็บและเข้าถึงข้อมูลแบบใช้แถวเป็นหลัก ยังส่งผลถึงความเร็วในการประมวลผล อันเป็นผลสืบเนื่องจากลักษณะการทำงานของหน่วยความจำแคช (`cache`) ในหน่วยประมวลผลกลาง (`CPU`)

[4] ในขณะที่เรากำลังดูค่าพิกเซลต่าง ๆ โปรแกรม `XnView` จะแสดงทั้งค่าพิกเซลและตำแหน่งพิกเซลมาด้วยกัน ให้เราจำตำแหน่งและค่าพิกเซลไว้ จากนั้นนำตำแหน่งไปใส่ในโปรแกรมเรา และดูว่าค่าจากอาเรย์ตรงกับค่าพิกเซลที่เราเห็นจาก `XnView` หรือไม่

ส่วนที่ 2 การเขียนไฟล์ภาพและแก้ไขค่าพิกเซล

3. เขียนไฟล์ภาพ [ImageCopy]

เมื่อประมวลผลภาพเสร็จแล้ว สิ่งที่เรามักจะทำอยู่บ่อย ๆ ก็คือการเขียนภาพนั้นเป็นไฟล์ผลลัพธ์ ในแบบฝึกหัดนี้เราจะใช้ค่าใน `BufferedImage img;` จากข้อที่แล้วไปเขียนเป็นไฟล์อีกไฟล์หนึ่ง ทำให้ผลลัพธ์ที่ได้เป็นเหมือนการคัดลอกไฟล์นั่นเอง

Follow Me: ใช้ ImageIO ในการเขียนไฟล์ภาพ

คำว่า IO ย่อมาจาก Input/Output ซึ่งสะท้อนถึงความสามารถในการอ่านและเขียนข้อมูล ตัวคลาส ImageIO ที่เราใช้มาก่อนหน้า ก็มีทั้งความสามารถในการอ่านและเขียนไฟล์ภาพ ซึ่งกระบวนการทำงานก็คล้ายกับการอ่านภาพ โดยจะเริ่มจากการสร้างวัตถุ File ที่เชื่อมไปยังพาธที่เราต้องการเขียน เช่น D:/myOutputImage.png เป็นต้น จากนั้นก็เรียกเมธอด write ในคลาส ImageIO

ซึ่งหากเรามีตัวแปรชื่อ img ชนิด BufferedImage และเราต้องการเขียน img ลงไปในไฟล์ ใจความของโค้ดก็มีสองบรรทัดดังนี้

```
File file = new File("D:/ myOutputImage.png");
ImageIO.write(img, "png", file);
```

ขอให้สังเกตสังเกตด้วยว่าแม้ชื่อไฟล์ในวัตถุ file จะสื่อว่าเราจะเขียนไฟล์แบบ PNG ถึงกระนั้นเราก็ยังต้องระบุลงในพารามิเตอร์ตัวที่สองว่าจะเขียนไฟล์แบบ PNG ส่วนถ้าเราต้องการไฟล์รูปแบบอื่นเช่น JPG หรือ GIF เราก็ควรเปลี่ยนพารามิเตอร์ตัวที่สองให้เป็น “jpg” หรือ “gif” ตามลำดับ และควรแก้ชื่อไฟล์ใน file ให้สอดคล้องด้วย[5]

หมายเหตุ เรามีความจำเป็นที่จะต้องจัดการกับ IOException คล้ายกับตอนอ่านไฟล์ภาพ ขอให้ลองทำจุดนี้ด้วยตนเอง หากยังไม่เข้าใจ ลองวกกลับไปอ่านเนื้อหาตรง Follow Me ของปัญหา ImageReading

[5] รายละเอียดเพิ่มเติมเกี่ยวกับการเขียนไฟล์ภาพ สามารถศึกษาได้จาก https://docs.oracle.com/javase/tutorial/2d/images/saveimage.html

ถึงตาของคุณแล้ว: จงเขียนโปรแกรมอ่านไฟล์รูปภาพจากพาธที่ผู้ใช้ระบุ และเขียนไฟล์ดังกล่าวออกไปในพาธที่ผู้ใช้ระบุ โดยทำผ่าน `ImageIO` และ `BufferedImage`

ข้อมูลเข้า

1. บรรทัดแรกเป็นสตริงที่ระบุพาธของไฟล์รูปภาพที่เป็นข้อมูลเข้า
2. บรรทัดที่สองเป็นสตริงที่ระบุพาธของไฟล์ผลลัพธ์

ผลลัพธ์

ถ้าพาธของไฟล์รูปภาพทั้งข้อมูลเข้าและผลลัพธ์ถูกต้อง โปรแกรมจะต้องสร้างไฟล์ใหม่ที่พาธของผลลัพธ์ได้ แต่หากพาธของไฟล์ข้อมูลเข้าผิดหรืออ่านรูปไม่สำเร็จ ให้โปรแกรมพิมพ์ข้อความว่า `Wrong input` และจบการทำงาน แต่หากข้อมูลเข้าถูกต้อง แต่โปรแกรมไม่สามารถบันทึกไฟล์ผลลัพธ์ได้ ให้พิมพ์ว่า `Cannot write file`

ตัวอย่างข้อมูลเข้า

```
C:/images/castle_building.png
D:/myOutputImage.png
```

หมายเหตุ คุณสามารถเปลี่ยนพาธของข้อมูลเข้าและผลลัพธ์ให้เหมาะสมได้ และพาธอาจมีช่องว่าง ดังนั้นเวลาอ่านข้อมูลเข้า ถ้าใช้เมธอด `nextLine` ในคลาส `Scanner` จะเป็นวิธีที่ค่อนข้างง่าย

ตัวอย่างผลลัพธ์

1. ในกรณีที่อ่านไฟล์ข้อมูลเข้าได้ตามปรกติและโปรแกรมสามารถเขียนไฟล์ผลลัพธ์ได้ คุณควรจะพบไฟล์รูปอยู่พาธในบรรทัดที่สอง
2. ถ้าพาธในบรรทัดแรกผิดหรือรูปเสีย โปรแกรมจะพิมพ์ว่า `Wrong input` และจบการทำงาน
3. ถ้าพาธและรูปข้อมูลเข้าไม่มีปัญหา แต่มีปัญหากับพาธผลลัพธ์ (เช่น โฟลเดอร์หรือไดรฟ์ผิด หรือดิสก์เต็ม เป็นต้น) โปรแกรมจะพิมพ์คำว่า `Cannot write file` และจบการทำงาน

หมายเหตุ มีคำแนะนำให้ทางด้านใต้ แต่ขอให้ลองพยายามทำด้วยต้นเองให้ได้ก่อนอ่านคำแนะนำ

ถึงแม้ว่าจุดประสงค์หลักในข้อ `ImageCopy` จะถือว่าง่ายและตรงไปตรงมามาก แต่การจะจัดการปัญหาที่เกี่ยวข้องกับความผิดพลาดในระบบไฟล์เป็นเรื่องที่ต้องคิดสักหน่อย เวลาที่เราจะพัฒนาโปรแกรมต่าง ๆ เราควรเริ่มจากจุดที่เข้าใจได้ง่ายแล้วขยายความสามารถของโปรแกรมให้ครอบคลุมขึ้น

อย่างในข้อนี้ เราควรเริ่มจากกรณีที่พาธข้อมูลเข้าและผลลัพธ์รวมถึงรูปภาพต่างก็ถูกต้อง และดูว่าโปรแกรมเราทำการสร้างไฟล์ใหม่สำเร็จหรือยัง เมื่อทำสำเร็จแล้วก็ให้จัดการกรณีที่โปรแกรมมีปัญหาจากตัวของข้อมูลเข้า ดังแสดงให้เห็นถึงการจัดการ `IOException` ที่จะอธิบายต่อไป

เรื่องน่ารู้: เทคนิคการจัดการ `IOException`

ในทางภาษา เราสามารถครอบส่วนของการการอ่านและเขียนไฟล์ด้วยบล็อก `try-catch` เพียงอันเดียว เพราะทั้ง `ImageIO.read` และ `ImageIO.write` ต่างก็โยนเหตุการณ์ผิดปรกติชนิด `IOException` ทั้งคู่

แต่ในทางปฏิบัติ การทำเช่นนั้นจะทำให้แยกแยะได้ยากว่าเหตุการณ์ผิดปรกติที่เกิดขึ้นมาจากขั้นตอนก่อนอ่านหรือเขียน ดังนั้นการแยกบล็อก `try-catch` ออกเป็นสองชุด โดยชุดแรกครอบคลุมส่วนการอ่าน และอีกชุดครอบคลุมส่วนการเขียนจะทำให้เราแยกแยะเหตุการณ์ผิดปรกติได้ง่ายขึ้น

ดังนั้นในปัญหานี้ ถ้าการ `catch` ชุดแรกจะพิมพ์คำว่า `Wrong input` และสั่ง `return` เพื่อจบการทำงาน เราจะได้ผลลัพธ์ที่ตรงตามเป้าหมาย ส่วนตรง `catch` อีกชุด คาดว่าคุณน่าจะคิดและทดลองทำดูด้วยตัวเองได้แล้ว

`Follow Me`: อยากเห็นเหตุการณ์ที่ `ImageIO` พยายามอ่านไฟล์ที่มีอยู่จริงแต่อ่านไม่สำเร็จ

บางทีเราอยากจะทดสอบโปรแกรมในตอนที่ `ImageIO` อ่านข้อมูลไม่สำเร็จ ซึ่งวิธีที่ง่ายก็คือให้เราเปลี่ยนนามสกุลไฟล์ เช่น ไฟล์ที่มีนามสกุลเป็น `.txt .doc` ให้กลายเป็นนามสกุลแบบไฟล์ภาพ เช่น `.jpg .png` แน่นอนว่าเนื้อหาภายในไฟล์มันไม่ใช่รูปภาพ แต่ชื่อไฟล์ดูเหมือนเป็นรูปภาพ เวลาที่ `ImageIO` พยายามอ่านไฟล์แบบนี้ มันก็จะเกิดเหตุการณ์ผิดปรกติขึ้น

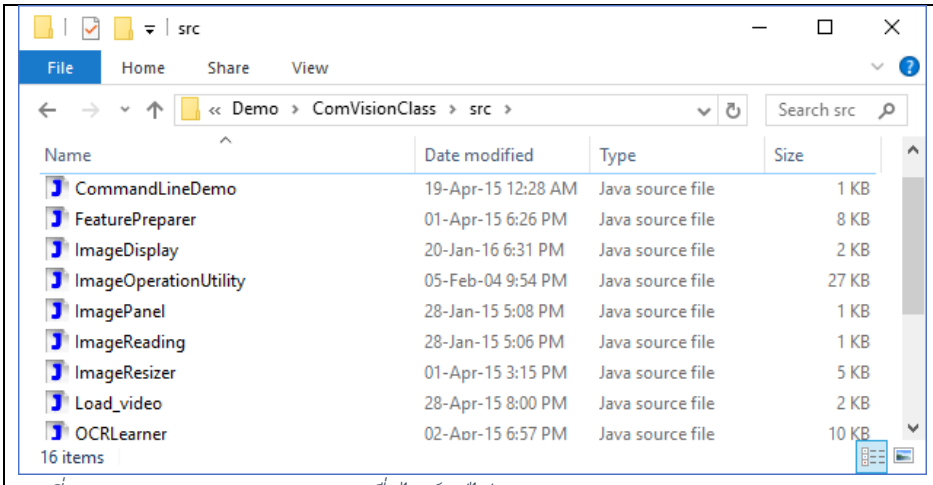
เพื่อเป็นการเรียนรู้ ให้เราสร้างไฟล์ข้อความหรือไฟล์ไมโครซอฟต์เวิร์ดที่ว่างเปล่าขึ้นมา บันทึกไฟล์ เปลี่ยนนามสกุลให้เป็น `jpg` แล้วลองให้โปรแกรมอ่านไฟล์นั้น (สำหรับผู้ที่ไม่ทราบวิธีเปลี่ยนนามสกุลไฟล์ ลองอ่านเรื่องน่ารู้อันถัดไปแล้วลองทำตาม)

เรื่องน่ารู้: วิธีเปลี่ยนนามสกุลไฟล์ (บนวินโดวส์)

การแก้ไขนามสกุลไฟล์เป็นเรื่องง่าย แต่การตั้งค่าการดูไฟล์ในระบบปฏิบัติการอาจจะทำให้มันเป็นเรื่องยาก เช่นในกรณีของระบบปฏิบัติการไมโครซอฟต์วินโดวส์ ไฟล์ที่มีนามสกุลแบบที่รู้จักกันอย่างแพร่หลาย จะไม่ถูกแสดงออกมาใน `Windows Explorer`[6] เมื่อไม่ถูกแสดงออกมา จะพิมพ์เปลี่ยนเป็นอย่างอื่นก็ทำไม่ได้ ในกรณีเช่นนี้เราควรเปลี่ยนการตั้งค่าใน `Windows Explorer` ให้แสดงนามสกุลไฟล์ทุกอันออกมา ดังแสดงในภาพด้านใต้นี้[7]

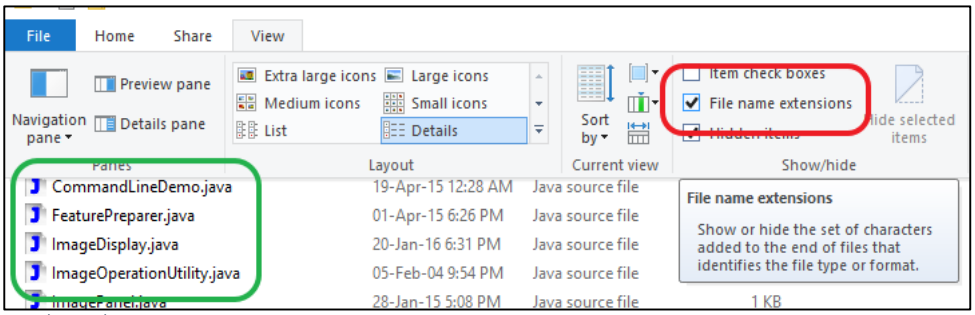
[6] หลายคนเรียก Windows Explorer ว่าโปรแกรมเปิดไฟล์ หรือเรียกสั้น ๆ ว่า Explorer

[7] ตัวอย่างวิธีการนี้ ใช้ได้กับ Windows 10 Redstone หากผู้อ่านใช้รุ่นอื่น อาจจะค้นหาจากเว็บว่า windows display file extension in explorer พร้อมบอกชื่อเวอร์ชัน เช่น 7 หรือ 8.1



รูปที่ 4 Windows Explorer แสดงชื่อไฟล์แต่ไม่แสดงนามสกุล

ในรูปที่ 4 เราจะเห็นไฟล์โค้ดจาวาเต็มไปหมด ซึ่งไฟล์พวกนี้มีนามสกุลเป็นแบบ **.java** ทว่าการแสดงผลในวินโดวส์ตอนนี้จะไม่แสดงนามสกุลไฟล์ที่เป็นที่รู้จัก เราสามารถเปลี่ยนให้มันแสดงนามสกุลไฟล์ทุกไฟล์ออกมาได้ ด้วยการไปที่เมนู **View** แล้วเลือก **“File name extension”** ดังแสดงในรูปที่ 5



รูปที่ 5 เมื่อเลือกออปชัน File name extension (กรอบเน้นสีแดง) Windows Explorer จะแสดงนามสกุลของไฟล์ให้เราเห็น สังเกตว่าตอนนี้ชื่อไฟล์มี .java ตามมาด้วย (กรอบเน้นสีเขียว)

เมื่อเห็นนามสกุลไฟล์แล้ว เราก็สร้างไฟล์ข้อความหรือไฟล์ไมโครซอฟต์เวิร์ดเปล่า ๆ บันทึกและแก้นามสกุลให้เป็นแบบ **jpg** แล้วลองให้โปรแกรมเราอ่านไฟล์ **jpg** นั้น เราก็จะได้เห็นเหตุการณ์ผิดปรกติที่เกิดขึ้นเพราะ **ImageIO** ไม่สามารถอ่านไฟล์ภาพได้ (แม้ไฟล์ภาพนั้นจะมีอยู่จริง)

เรื่องน่ารู้: ทบทวนเรื่องเมธอดสถิตและการเรียกใช้งาน

เมธอดสถิต (static method) เป็นเมธอดที่เราสามารถเรียกใช้ได้โดยไม่ต้องมีวัตถุ (object) นั่นคือเราสามารถเรียกใช้งานผ่านชื่อคลาสได้เลย ส่วนเมธอดแบบทั่วไป เราต้องเรียกผ่านวัตถุ

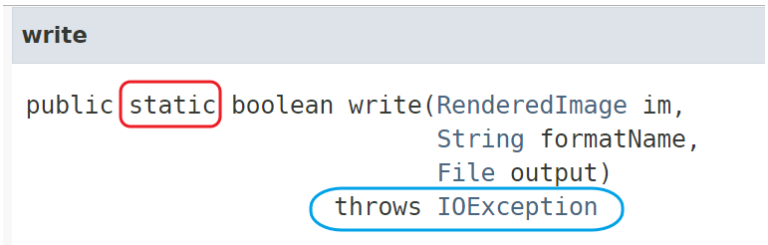
ถึงแม้เรื่องนี้จะเป็นความรู้พื้นฐานที่เราน่าจะทราบมาก่อนหน้า แต่การทบทวนและทำความเข้าใจกับตัวอย่างจริงก็นับว่าดีไม่น้อย เรามาเริ่มจากเมธอดสถิตที่เราใช้มาในแบบฝึกหัด ซึ่งก็คือ `ImageIO.read` และ `ImageIO.write` ซึ่งถ้าเราสังเกต เราจะพบว่าเราไม่ต้องสร้างวัตถุ ไม่ต้องการตัวแปรชนิด `ImageIO` แต่เรียกใช้เมธอด `read` และ `write` ได้ผ่านชื่อคลาส `ImageIO` ได้เลย

แต่ในกรณีทั่วไป เมธอดจะไม่เป็นแบบสถิต เพราะเมธอดผูกอยู่กับข้อมูลของวัตถุโดยตรง เช่น หากเราต้องการทราบความยาวของสตริงในตัวแปร `str` เราจะเขียนว่า `str.length();` เราไม่สามารถเขียนว่า `String.length();` ได้ เพราะแบบที่สองนี้ไม่ได้สื่อความหมายเลยว่าจะหาความยาวของข้อความใด เพราะตัวของคลาสเองไม่ได้มีข้อความ แต่เป็นตัววัตถุที่มีข้อความ

วกกลับมาที่ `ImageIO.write` ซึ่งก็ต้องการข้อมูลภาพสำหรับเขียนลงไฟล์เหมือนกัน ขอให้สังเกตว่าตัว `ImageIO` ไม่มีทางมีข้อมูลภาพอยู่แน่ ดังนั้นเราจึงต้องใส่ตัวแปรวัตถุชนิด `BufferedImage` เข้าไปในพารามิเตอร์เพื่อเป็นข้อมูลภาพ ดังเห็นได้จากตัวอย่างที่แสดงไว้ก่อนหน้า

```
ImageIO.write(img, "png", file);
```

สุดท้าย หลายคนอาจจะสงสัยว่าแล้วเราจะรู้ได้อย่างไรว่าเมธอดไหนเป็นแบบสถิต ซึ่งเรื่องนี้เราสามารถดูได้จากเอกสารเอพีไอ ซึ่งมักค้นได้จากอินเตอร์เน็ต เช่นในทีนี้ถ้าเราค้นหาข้อมูลเกี่ยวกับ `ImageIO.write` เราก็มักจะมาถึงลิงค์นี้ ตามภาพด้านล่าง



รูปที่ 6 เอกสารเอพีไอแสดงลักษณะของเมธอดอย่างละเอียด รวมถึงเหตุการณ์ผิดปรกติที่อาจถูกโยนออกมาจากเมธอดด้วย

เจาะลึก: **แล้วทำไมเมธอด `write` ไม่ไปอยู่ใน `BufferedImage` และไม่ถูกแปลงเป็นเมธอดธรรมดา ทำไมต้องไปอยู่ใน `ImageIO` และเป็นแบบสถิต**

วิธีจัดการข้อมูลนั้นขึ้นอยู่กับมุมมองของผู้ออกแบบเอพีไอ และธรรมชาติของปัญหาที่ต้องการแก้ไข เพราะหากมองย้อนกลับไปถึงการมาของคลาสต่าง ๆ เราจะพบว่าคลาส `BufferedImage` มีมาก่อน `ImageIO` และการจะไปแก้ไขคลาส `BufferedImage` ให้เขียนไฟล์ภาพออกมาได้หลากหลายรูปแบบ อาจจะเป็นการเปลี่ยนแปลงที่ส่งผลกระทบที่ไม่พึงประสงค์ต่อผู้ที่ใช้คลาส `BufferedImage`

นอกจากนี้ ผู้ออกแบบเอพีไอยังต้องการคลาสที่สามารถเขียนไฟล์จากข้อมูลวัตถุหลายชนิด ไม่ได้จำกัดอยู่เฉพาะ `BufferedImage` การสร้างคลาสใหม่ที่มุ่งเน้นความสามารถด้านการอ่านเขียนไฟล์ภาพโดยตรงจะทำให้เอพีไอเรียบง่าย ดูเป็นหมวดหมู่ที่เข้าใจง่าย และใช้งานสะดวกกว่า

4. เปลี่ยนค่าพิกเซลในภาพ [PixelWriter]

การเปลี่ยนค่าพิกเซลในภาพเป็นภารกิจที่ต้องทำเป็นประจำในงานประมวลผลภาพและการมองเห็นของคอมพิวเตอร์ เพราะงานทั้งสองมักเกี่ยวพันกับการกรองภาพหรือวาดผลลัพธ์ลงไปในภาพ ในแบบฝึกหัดนี้ เราจะเรียนรู้การพื้นฐานแก้ไขภาพในระดับพิกเซล

สิ่งแรกที่เราต้องแยกให้ออกก็คือ ราสเตอร์ (`Raster`) ที่ใช้ในการอ่านค่าพิกเซลในภาพ เป็นราสเตอร์ที่อ่านค่าได้อย่างเดียว. ในกรณีที่เราต้องการแก้ไขค่าพิกเซลในภาพ เราจะต้องใช้ราสเตอร์แบบเขียนค่าได้ (`WritableRaster`). เราไม่จำเป็นต้องเปลี่ยนคำสั่งในการเรียกใช้ราสเตอร์จาก `BufferedImage` แต่ประการใด. ขอเพียงแต่เปลี่ยนชนิดราสเตอร์ให้เป็นแบบเขียนได้ก็เพียงพอแล้ว. ในตัวอย่างนี้ เราสมมติว่าเรามีค่าพิกเซลของภาพต้นฉบับอยู่ใน `pixelArray` เรียบร้อยแล้วจากการโหลดภาพสีเทา gray 150 และเราต้องการจะวาดเส้นสีขาวลงไปที่กลางภาพดังแสดงในรูปที่ 7

รูปที่ 7 ภาพสีเทาที่มีเส้นสีขาวตรงกลางภาพ

Follow Me: ใช้ราสเตอร์แบบเขียนได้ในการเปลี่ยนค่าพิกเซล

เราจะเขียนเส้นสีขาวลงไปเป็นเส้นกลางภาพ อันดับแรกเราจะเตรียมราสเตอร์สำหรับเขียนภาพไว้ก่อน ซึ่งทำได้โดย

```
WritableRaster wraster = img.getRaster();
```

อันดับต่อมา เราจะต้องคำนวณเลขแถวที่เป็นตำแหน่งกลางภาพ นั่นคือเราต้องรู้ความสูงของภาพก่อน จากนั้นจึงนำความสูงของภาพไปหารสอง เพื่อให้ได้ตำแหน่งแถวตรงกลางภาพ

```
final int height = img.getHeight();
int middleRow = height / 2;
```

เราพิจารณาได้ว่าสีขาวมีค่าความสว่างที่ 255 และเราใช้กระบวนวิธีอ่านค่าลงในที่พักข้อมูล (buffer) ในลักษณะเดิม และทำการเขียนสีขาวลงไปในทุกคอลัมน์ เนื่องจากทุกพิกเซลที่จะเขียนลงไปเป็นสีเดียวกัน เราจึงสามารถเตรียมค่าสีขาวลงไปในที่พักข้อมูลได้ก่อน

```
final int white = 255;
int[] pixelBuffer = new int[1];
pixelBuffer[0] = white;
```

ในโค้ดทางด้านบน ที่จริงเราใส่ค่า 255 ลงไปในที่พักข้อมูลตรง ๆ โดยไม่ต้องผ่านตัวแปร white ก็ได้ แต่การใส่ผ่านตัวแปรมันทำให้เห็นเจตนาได้โดยง่ายว่าค่า 255 นี้มีความหมายคือสีขาว ไม่ได้เป็นอย่างอื่น

สุดท้าย เราจะทำการวนลูปจากคอลัมน์ซ้ายสุด ไปคอลัมน์ขวาสุด (คอลัมน์ที่ตำแหน่งความกว้าง – 1) โดยวนเพียงแถวเดียวคือแถวหมายเลข middleRow

```
final int width = img.getWidth();
for(int col = 0; col < width; ++col) {
  wraster.setPixel(col, middleRow, pixelBuffer);
}
```

เอ๊ะ แล้วจะรู้ได้ว่าเราเขียนโปรแกรมวาดเส้นสีขาวถูกต้องแล้ว? ปัญหานี้สามารถตอบได้ด้วยการใช้สิ่งที่เรียนมาจากข้อที่แล้ว นั่นก็คือเขียนผลลัพธ์ใน `BufferedImage` ลงไปในดิสก์ด้วย `ImageIO.write` นั่นเอง

สรุปขั้นตอนที่ต้องทำในข้อ `PixelWriter` ก็คือ (1) อ่านภาพสีเทาจากไฟล์ (2) เตรียมราสเตอร์แบบเขียนได้และที่พักข้อมูล (3) ใช้ราสเตอร์เขียนค่าพิกเซลลงไปใหม่ และ (4) เขียนภาพที่แก้แล้วลงดิสก์

5. กลับดำเป็นขาว กลับขาวเป็นดำ [BrightnessInverse]

ในแบบฝึกหัดที่แล้ว เราได้เรียนรู้การแก้ค่าพิกเซลให้มีค่าความสว่างตามที่ต้องการ ครั้งนี้เราจะลองเอาความรู้มาประยุกต์กับการใช้งานที่มีประโยชน์ในทางปฏิบัติกัน ซึ่งก็คือการกลับความสว่าง ซึ่งมีการใช้ในงานกราฟฟิกทั่วไป งานสร้างอะนิเมชันหรือภาพยนตร์ เช่นการสร้างภาพที่ดูน่ากลัวหรือแสดงถึงแรงกดดันซึ่งเป็นภาพโทนมืด แท้จริงอาจจะสร้างมาจากภาพโทนสว่างแล้วจึงกลับความสว่างก็เป็นได้ ดังแสดงในตัวอย่างข้างล่างนี้ [ภาพจากอะนิเมชันเรื่องนารุโตะ]

เนื่องจากมีการใช้คอมพิวเตอร์ช่วยในการลงสีในอะนิเมชัน การเปลี่ยนโทนสีเพื่อสะท้อนอารมณ์ตามบทจึงเป็นสิ่งที่ทำให้สามารถทำได้อย่างมีคุณภาพ อย่างในกรณีของรูปที่ 8 ภาพโทนสว่างซึ่งอาจจะลงสีง่ายกว่า ตัวละครที่เป็นจุดเด่นในภาพ (คาคาชิ) ซึ่งถูกตรึงไว้มีเครื่องแต่งกายอยู่ในโทนสีปรกติที่ปรากฏในฉากทั่ว ๆ ไปรูปที่ 8 สีเครื่องแต่งกายของตัวละครหลักจะมีสีตามปรกติ แต่ในรูปที่ 9 สีจะถูกกลับขั้วเพื่อแสดงความน่าสะพรึงกลัวของสิ่งที่ตัวละครกำลังเผชิญอยู่

รูปที่ 8 ภาพโทนสว่างซึ่งอาจจะลงสีง่ายกว่า ตัวละครที่เป็นจุดเด่นในภาพ (คาคาชิ) ซึ่งถูกตรึงไว้มีเครื่องแต่งกายอยู่ในโทนสีปรกติที่ปรากฏในฉากทั่ว ๆ ไป

รูปที่ 9 ภาพหลังจากกลับโทนสีที่ให้บรรยากาศน่าสะพรึงกลัวตามเทคนิคของคาถาที่ใช้โจมตีคาคาชิ

ถึงตาของคุณแล้ว: จงเขียนโปรแกรมอ่านไฟล์รูปภาพจากพาธที่ผู้ใช้ระบุ จากนั้นให้โปรแกรมกลับค่าความสว่างจากโทนมืดเป็นสว่างและจากโทนสว่างเป็นโทนมืด

การกลับความสว่างนี้มีแนวคิดอยู่ว่า ยิ่งพิกเซลต้นฉบับมีค่าความสว่างมากเท่าใด พิกเซลผลลัพธ์ก็ต้องมืดลงเท่านั้น ในทำนองเดียวกัน ยิ่งพิกเซลต้นฉบับดูมืดมากเท่าใด พิกเซลผลลัพธ์ก็ต้องสว่างมากขึ้นเท่านั้น ซึ่งกลไกนี้สามารถทำได้โดยการใช้สมการ

```
output = 255 – input
```

สังเกตให้ดีว่าหาก `input` มีความสว่างมาก ค่าของมันก็จะยิ่งสูง ทำให้ค่าของ `output` ลดลงและดูมืดนั่นเอง

ข้อมูลเข้า

1. บรรทัดแรกเป็นสตริงที่ระบุพาธของไฟล์รูปภาพที่เป็นข้อมูลเข้า
2. บรรทัดที่สองเป็นสตริงที่ระบุพาธของไฟล์ผลลัพธ์

ผลลัพธ์

เป็นไฟล์ภาพที่กลับความสว่าง

ตัวอย่างข้อมูลเข้า

```
C:/images/castle_building.png
D:/myOutputImage.png
```

หมายเหตุ ให้ใช้กลไกการจัดการเหตุการณ์ผิดปรกติตามแบบที่เคยใช้ในข้อก่อนหน้าด้วย แล้วก็**อย่า `hard code` พาธไว้ในโปรแกรม ให้รับพาธมาจากอินพุตเก็บไว้ในสตริง แล้วค่อยนำไปเปิดไฟล์ตามพาธที่ได้มา** ถ้า `hard code` ไว้จะโดนตัดคะแนน

คำแนะนำ ใน `Scanner` มีคำสั่ง `nextLine()` ให้เรียกใช้

ตัวอย่างภาพข้อมูลเข้าและผลลัพธ์

ภาพข้อมูลเข้า	ภาพผลลัพธ์
	




สามารถดาวน์โหลดภาพข้อมูลเข้าได้จากลิงค์ต่อไปนี้ ภาพศิลปากร ภาพคาคาชิกับกระจกเงาหมื่นบุปผา และ ภาพปราสาทโบราณ