การมองเห็นของคอมพิวเตอร์

แบบฝึกหัดเขียนโปรแกรมชุดที่ 2: Pixel Processing

แบบฝึกหัดนี้เป็นจำพวกเรียนรู้กลางอากาศ คือให้ทำไป เข้าใจไปเรื่อย ๆ เรียนรู้สิ่งใหม่ไปตามทาง เป็นทั้งแบบฝึกหัดและการสอนไปในตัว สิ่งที่ต่างจากชุดที่แล้วก็คือเราจะเน้นที่การประยุกต์มากขึ้น และมีการใช้ลูปกันอย่างเต็มที่

ส่วนที่ 1 เธรสโฮลด์

1. การสร้างภาพทวิภาค [ImageBinarization]

เราสามารถประมวลผลภาพเพื่อแบ่งพิกเซลออกเป็นกลุ่ม ซึ่งวิธีที่พื้นฐานและพบได้บ่อยก็คือการแบ่งพิกเซลออกเป็นกลุ่มสีขาวและสีดำ ซึ่งก็คือกระบวนการสร้างภาพทวิภาค (binary image) ดังแสดงในตัวอย่างข้างล่างนี้

รูปที่ 1 ภาพข้อมูลเข้าก่อนการทำเธรสโฮลด์เพื่อแปลงเป็นภาพทวิภาค

รูปที่ 2 ภาพผลลัพธ์เมื่อใช้ค่าเธรสโฮลด์เป็น 96 คือพิกเซลใดที่มีค่าความสว่างต่ำกว่าค่านี้จะถูกจัดเป็นสีดำ แต่หากมีค่าความสว่างสูงพอก็จะเป็นสีขาว

รูปที่ 3 ภาพผลลัพธ์เมื่อใช้ค่าเธรสโฮลดเป็น 128 สังเกตด้วยว่าเมื่อเราตั้งค่าเธรสโฮลด์ไว้สูงขึ้น พิกเซลก็จะถูกจัดเป็นสีดำมากขึ้นด้วย

การแบ่งพิกเซลออกเป็นกลุ่มเพื่อสร้างภาพทวิภาคแบบง่าย (simple threshold) จะมีค่าเธรสโฮลด์ T ที่กำหนดมาหนึ่งค่า จากนั้นจะใช้ค่านี้ในการกำหนดค่าในภาพผลลัพธ์ ซึ่งค่าในภาพผลลัพธ์จะมีความเป็นไปได้แค่สองค่า เช่น 0 และ 1 หรือ 0 และ 255 เป็นต้น โดยมากเรามักจะกำหนดว่าถ้าค่าพิกเซลในภาพ I ที่ตำแหน่ง (x, y) น้อยกว่า T เราจะกำหนดให้ค่าในภาพผลลัพธ์เป็น 0 ไม่เช่นนั้นก็จะมีค่าเป็น 1 กล่าวคือ ค่าในภาพผลลัพธ์ I' ถูกกำหนดโดย

$$I'(x,y) = \begin{cases} 0 & \text{เมื่อ } I(x,y) < T \\ 1 & \text{สำหรับกรณีอื่น ๆ} \end{cases} \qquad (1)$$

อย่างไรก็ตาม เราควรจะเข้าใจว่าตัวอย่างที่ยกมาในสมการด้านบนนี้ เป็นเพียงทางเลือกหนึ่งที่เป็นไปได้ ในตอนที่เราจะใช้งานจริง เราจะต้องพิจารณาเลือกวิธีกำหนดค่าให้เหมาะสม เช่น ถ้าหากเราต้องการนำผลลัพธ์ไปแสดงผลเป็นภาพเฉดเทา 8 บิตที่ดูเข้าใจง่าย เราจะเปลี่ยนเลข 1 ให้กลายเป็น 255 เพื่อให้จุดในภาพผลลัพธ์เป็นสีขาว นอกจากนี้เราสามารถเลือกให้ผลลัพธ์กลับกันกับแบบเดิม คือทำให้ค่าในภาพผลลัพธ์เป็น 0 เมื่อค่าในภาพข้อมูลเข้ามีค่ามากก็ได้ เช่น

$$I'(x,y) = \begin{cases} 1 & \text{เมื่อ } I(x,y) < T \\ 0 & \text{สำหรับกรณีอื่น ๆ} \end{cases} \qquad (2)$$

ทางเลือกแต่ละทางเป็นสิ่งที่เราต้องวิเคราะห์ตามบริบทของปัญหา ไม่ใช่ว่าแต่ละแบบจะถูกหรือผิดในทุกสถานการณ์

Follow Me: ทำการแก้ไขพิกเซลโดยใช้ค่าเธรสโฮลด์ที่ผู้ใช้กำหนดให้

อาศัยความรู้จากแบบฝึกหัดที่แล้ว เราสามารถโหลดรูปภาพและใช้ราสเตอร์เพื่ออ่านค่าพิกเซลได้ ในครั้งนี้ เรามีเป้าหมายว่าหากเราทราบค่าเธรสโฮลด์ T จากผู้ใช้ และเราต้องการสร้างภาพผลลัพธ์ใหม่ในลักษณะเดียวกับสมการ (1) โดยการแปลงค่าเลข 1 ในที่นี้จะเปลี่ยนเป็นเลข 255 เพื่อให้ได้พื้นที่สีขาว

ดังนั้นโค้ดตรงส่วนที่ทำการเปลี่ยนค่าพิกเซลในภาพเพื่อทำภาพทวิภาคจะเป็นดังนี้

```
int[] pixelBuffer = new int[1];
for(int row = 0; row < height; ++row) {
  for(int col = 0; col < width; ++col) {
    raster.getPixel(col, row, pixelBuffer);
    if(pixelBuffer[0] < T)
      pixelBuffer[0] = 0;
    else
      pixelBuffer[0] = 255;
    raster.setPixel(col, row, pixelBuffer);
  }
}
```

โค้ดตรงส่วนสีม่วง เป็นจุดเดียวที่ต่างกับงานที่ผ่านมา กล่าวได้ว่าการสร้างภาพทวิภาคแบบง่ายนี้มีโครงสร้างการทำงานคล้ายกับการกลับสีพิกเซลมาก

ถึงตาของคุณแล้ว: นำโค้ดที่เรียนมารวมกันและทดสอบเปลี่ยนค่าเธรสโฮลด์

ให้ลองนำโค้ดจากแบบฝึกหัดชุดที่แล้วมาดัดแปลงจากความเข้าใจ เพื่อให้ได้โปรแกรมการสร้างภาพทวิภาคด้วยเธรสโฮลด์แบบง่าย จากนั้นลองทดสอบกับภาพคาสเซิล และลองเปลี่ยนค่าเธรสโฮลด์หลาย ๆ ค่า เช่น 72 96 และ 128 พร้อมกับสังเกตว่าภาพมีแนวโน้มจะดูเข้มขึ้นหรือสว่างขึ้นอย่างไรบ้าง

ส่วนที่ 2: การจัดระเบียบโค้ดให้ง่ายต่อการพัฒนาโปรแกรมอย่างเป็นระบบ

แบบฝึกหัดชุดนี้สอนให้เห็นถึงเรื่องราวพื้นฐานที่เกิดขึ้นในภาษาจาวาและภาษาเชิงวัตถุอีกหลายภาษา ซึ่งจริง ๆ แล้วเราอาจจะรู้เรื่องนี้ไปพอสมควรแล้ว แต่อาจจะยังไม่เคยเขียนโปรแกรมตามแนวทางที่ดีและพิจารณาประเด็นบางอย่างโดยละเอียด แบบฝึกหัดนี้จะพาพวกเราสัมผัสกับเหตุการณ์จริงและทำความเข้าใจกลไกในภาษาเชิงวัตถุให้ดียิ่งขึ้น

2. พิจารณางานที่ซ้ำซากและสร้างเป็นเมธอดอรรถประโยชน์ [UtilMethods]

งานในสาขาวิชาหนึ่งมักมีอะไรที่คล้ายกัน ทำให้เราสามารถใช้ประสบการณ์ที่ผ่านมาทำงานให้สำเร็จได้อย่างมีประสิทธิภาพมากขึ้น ในตอนนี้ขอให้เราสังเกตว่า เรา

การกลับค่าสีในแบบฝึกหัดที่แล้วและการทำเธรสโฮลด์ในข้อนี้ดูคล้ายกันมาก เราสามารถ **"ยืม"** โค้ดจากข้อที่แล้วมาใช้ได้อีกโดยสะดวก

เพื่อให้เห็นภาพลองมาพิจารณากระบวนการโหลดภาพ ซึ่งในทั้งสองข้อเราเขียนลงไปในเมธอด **main** ว่า

```
BufferedImage img = null;
try {
    File imgFile = new File(inputPath);
    img = ImageIO.read(imgFile);
} catch(IOException ex) {
    System.err.println("Error loading image");
}
```

โค้ดด้านบนนี้ต้องการโหลดภาพเข้าไปยังตัวแปร **img** ซึ่งโดยปรกติเราก็ต้องโหลดภาพด้วยวิธีทำนองนี้อยู่เป็นประจำ ถ้าอย่างนั้น มันจะดีกว่าไหมที่เราจะแยกโค้ดตรงนี้ไปใส่คลาสอรรถประโยชน์ และทำให้เราโหลดภาพได้ง่าย ๆ ผ่านการเรียกเมธอดเดียว

Follow Me: สร้างแพ็คเกจสำหรับคลาสอรรถประโยชน์

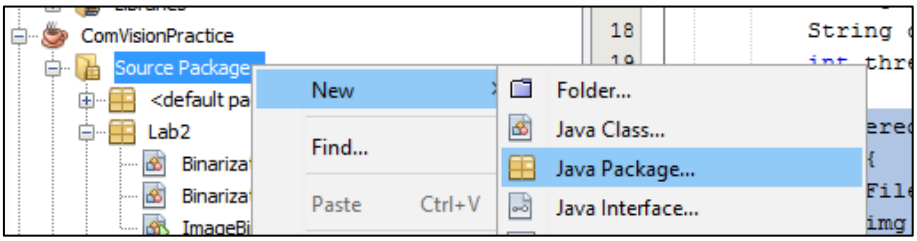
โดยมากเราจะใช้แพ็คเกจในการจัดกลุ่มคลาสในภาษาจาวาด้วยเหตุสองประการ[1] คือ (1) จัดตามหน้าที่ ซึ่งหากคลาสมีหน้าที่ที่สัมพันธ์สอดคล้องกัน ก็มักจะอยู่ในแพ็คเกจเดียวกัน และ (2) เพื่อความสะดวกในการประยุกต์ใช้ในโปรแกรมอื่น

เมื่อแยกแพ็คเกจแล้ว หากเรามีโปรแกรมจาวาจำนวนมากที่ต้องการใช้ประโยชน์จากมัน เราจะอ้างอิงมายังแพ็คเกจนี้ได้ในลักษณะเดียวกัน นอกจากนี้เราสามารถจัดเป็น **Java Archive** (**JAR**) ที่เหมาะกับการแจกจ่ายใช้งานในวงกว้างได้ด้วย

การสร้างแพ็คเกจในจาวานั้นจะทำให้เกิดโฟลเดอร์สำหรับเก็บไฟล์แยกออกมาตามชื่อแพ็คเกจนั้น เช่น ถ้าเราต้องการแพ็คเกจชื่อ **imageutil** เราก็จะสร้างโฟลเดอร์ชื่อ **imageutil** ขึ้นมาสำหรับเก็บไฟล์โค้ดจาวาไว้ในนั้นด้วย นอกจากนี้เรานิยมจะใช้ชื่อแพ็คเกจโดยใช้ตัวพิมพ์เล็กทั้งหมด

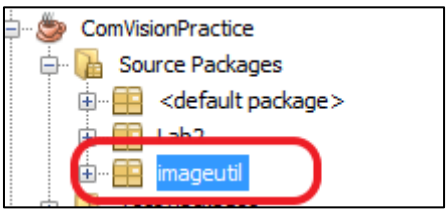
[1] ยังมีเหตุผลอีกหลายอย่างที่เกี่ยวข้องกับการแยกแพ็คเกจ เช่น ป้องกันชื่อคลาสซ้ำกัน เพราะเป็นงานที่มาจากคนละทีมพัฒนา เป็นต้น ที่ยกมานี้เป็นเหตุผลหลักที่พบได้บ่อย

เอาล่ะ มาลองสร้างแพ็คเกจ **imageutil** สำหรับเก็บคลาสอรรถประโยชน์ด้านการจัดการภาพใน **NetBeans IDE** กันดูดีกว่า อันดับแรกให้คลิกขวาไปที่ **Source Packages** ภายในโปรเจ็คที่เราจะสร้างแพ็คเกจ เลือกที่ **New -> Java Package…** ดังแสดงในรูปที่ 4



รูปที่ 4 การเลือกเมนูสำหรับสร้างแพ็คเกจใหม่ใน NetBeans IDE

จากนั้นจะมีไดอะล็อกสำหรับระบุชื่อแพ็คเกจที่ต้องการ ให้เราพิมพ์ว่า **imageutil** จากนั้นเลือก **Finish** ซึ่งหากเราทำทุกอย่างถูกต้อง เราจะได้แพ็คเกจใหม่ในโฟลเดอร์รากของโปรเจ็คดังแสดงในรูปที่ 5



รูปที่ 5 แพ็คเกจใหม่ที่ได้มา

Follow Me: สร้างคลาสสำหรับบรรจุเมธอดอรรถประโยชน์

เราต้องมีคลาสสำหรับเก็บเมธอดที่เราต้องการ ซึ่งอาจจะมีหลายคลาสก็ได้ เนื่องจากคลาสหนึ่ง ๆ ไม่ควรจะรับหน้าที่ที่หลากหลายจนเกินไป เพราะเราควรจะจัดหมวดหมู่เมธอดให้ดูเข้าใจง่ายมากกว่าที่จะเอาทุกสิ่งทุกอย่างไปปนกัน

สำหรับเมธอดจัดการภาพพื้นฐานในตัวอย่างนี้ เราจะสร้างคลาส **Util** ภายในแพคเกจ **imageutil** ซึ่งทำได้โดยการคลิกขวาไปที่แพคเกจ **imageutil** ในรูปที่ 5 เพื่อเรียกเมนูขึ้นมา จากนั้นให้สร้างคลาสใหม่ ผ่านเมนู **New -> Java Class**

หมายเหตุ งานนี้แม้เป็นงานที่ง่ายและตรงไปตรงมา แต่สำหรับผู้ที่หัดใช้เครื่องมือใหม่ ๆ และยังไม่ค่อยได้สัมผัสการใช้แพคเกจอาจจะยังไม่คุ้นเคยถึงวิธีสร้างคลาสใหม่ในแพคเกจที่ตนต้องการ ซึ่งวิธีการที่ใช้ใน NetBeans นี้คล้ายกับวิธีในเครื่องมืออีกหลายตัว และ

Follow Me: พิจารณาข้อกำหนดพื้นฐานของเมธอด loadImage ในคลาส Util

ณ จุดนี้เราจะย้ายการโหลดรูปภาพ ซึ่งมักต้องใช้บ่อย ๆ ไปไว้ในคลาส **Util** ภายในแพคเกจ **imageutil** อันดับแรก เราต้องเข้าใจจุดประสงค์ของเมธอดนี้ว่า มันจะถูกนำมาใช้ในบรรทัดที่ก่อนหน้านี้เราเขียนว่า **BufferedImage img = null;** นั้นคือแทนที่เราจะกำหนดค่าเป็น **null** และโหลดภาพตามมาทีหลังใส่ตัวแปร **img** เราจะเรียกเมธอด **loadImage** เพื่อกำหนดค่าให้ **img** ทันที

ดังนั้นเมธอด **loadImage** จะต้องคืนข้อมูลชนิด **BufferedImage** กลับมา และเนื่องจากว่าเราต้องทราบพาธของไฟล์ที่จะโหลด เราจึงกำหนดให้ **String** ที่เก็บพาธของไฟล์เป็นพารามิเตอร์

สิ่งสำคัญที่ต้องพิจารณาตามมาก็คือว่า เราต้องการเรียกใช้เมธอดนี้โดยไม่ต้องเสียเวลาไปสร้างวัตถุชนิด **Util** ขึ้นมา ดังนั้นเราต้องกำหนดให้เมธอดเป็นแบบสถิต **(static)** และสุดท้าย เนื่องจากคลาสที่จะเรียกใช้งานเมธอดนี้อยู่นอกแพคเกจ **imageutil** ดังนั้นเมธอดจึงควรมีระดับการเข้าถึงเป็นแบบ **public**

ด้วยข้อกำหนดเหล่านี้ เราจึงต้องประกาศเมธอด **loadImage** ให้เป็น

```
public static BufferedImage loadImage(String imagePath)
```

จะเห็นได้ว่า องค์ประกอบของเมธอด **loadImage** ที่ประกาศไว้นั้นต่างล้วนมีเหตุผลและที่มาอันสมควร ไม่ได้เกิดขึ้นมาลอย ๆ และการพิจารณาในตัวอย่างที่ยกขึ้นมานี้เป็นสิ่งที่เราพึงกระทำในการเขียนโปรแกรมเพื่อให้ได้เมธอดที่เหมาะสมกับการใช้งาน

Follow Me: เขียนโค้ดการทำงานของเมธอด loadImage

จากงานที่เราต้องทำ เราสามารถโอนย้ายโค้ดที่เคยใช้มาใส่ไว้ในเมธอดนี้ได้เลย อย่างไรก็ตาม เราควรทำความเข้าใจว่า เราต้องคืนผลลัพธ์ ซึ่งก็คือ **BufferedImage** กลับไปด้วย ดังนั้นโค้ดของ **loadImage** ก็จะเป็นดังนี้

```
public static BufferedImage loadImage(String imagePath) {
```

```
    BufferedImage img = null;
    try {
        File imgFile = new File(imagePath);
        img = ImageIO.read(imgFile);
    } catch(IOException ex) {
        System.err.println("Error loading image");
    }
    return img;
}
```

3. เขียนเมธอดสำหรับบันทึกภาพลงไฟล์ [saveImageMethod]

ในเหตุผลทำนองเดียวกับโหลดไฟล์ภาพ คือเป็นงานที่ต้องทำซ้ำบ่อย ๆ เราจึงควรสร้างเมธอดสำหรับการบันทึกไฟล์ภาพในคลาสอรรถประโยชน์ด้วยเช่นกัน

ถึงตาของคุณแล้ว: สร้างเมธอด saveImage ในคลาส Util

ลองสำรวจดูโค้ดที่จะต้องย้ายมาจากเมธอด **main** ที่เราเขียนมาก่อนหน้า พร้อมทั้งพิจารณาลักษณะของเมธอดที่ต้องประกาศตามแนวทางที่กล่าวไว้ในข้อที่แล้ว จากนั้นนำข้อสรุปเกี่ยวกับพารามิเตอร์ที่จำเป็นสำหรับ **saveImage** มาสร้างเป็นเมธอดที่สามารถเรียกใช้เพื่อบันทึกรูปได้โดยสะดวกจากคลาสอื่น ๆ

หมายเหตุ พารามิเตอร์ของ **saveImage** จะแตกต่างจากของ **loadImage** อยู่เล็กน้อย ให้สังเกตดูภายในตัวโค้ดที่จะย้ายมาว่ามันสัมพันธ์กับข้อมูล/ตัวแปรใดบ้าง และควรจะส่งข้อมูลและตัวแปรนั้นมาเป็นพารามิเตอร์หรือไม่

4. นำเมธอด loadImage และ saveImage ไปใช้ [ApplyMethod]

เราจะนำเมธอดทั้งสองมาประยุกต์ใช้แทนโค้ดในเมธอด **main** เพื่อทำให้โค้ดดูเข้าใจง่ายและเป็นระเบียบมากขึ้น ซึ่งเหตุที่มันดูเข้าใจง่ายขึ้นนั้นมาจากการที่ชื่อเมธอดมันแสดงถึงงานที่ต้องการโดยชัดแจ้ง ดังจะแสดงให้ต่อไป

Follow Me: ใช้เมธอด loadImage และพิจารณาโค้ดที่ได้

[ให้ทำข้อนี้ด้วยการสร้างเป็นคลาสใหม่ชื่อ BinarizationWithUtil1 เพราะเราจะทบทวนความเปลี่ยนแปลงระหว่างเวอร์ชันแรกและเวอร์ชันสุดท้าย เพื่อดูว่าโค้ดของเราดูง่ายกว่าเดิมเพียงใด]

สำหรับการใช้ loadImage เราจะนำมาแทนที่บรรทัดที่เขียนว่า

```
BufferedImage img = null;
try {
    File imgFile = new File(inputPath);
    img = ImageIO.read(imgFile);
} catch(IOException ex) {
    System.err.println("Error loading image");
}
```

โดยจะแก้ให้เหลือเพียงบรรทัดเดียวเป็น

```
BufferedImage img = Util.loadImage(inputPath);
```

นั่นคือจำนวนบรรทัดในเมธอด main จะลดลงไปถึง 6 บรรทัด

แต่ที่สำคัญที่สุดก็คือชื่อเมธอดที่เราเรียกใช้มันส่อถึงงานที่เราต้องการโดยชัดแจ้ง ทำให้โค้ดใหม่ที่ได้อ่านง่ายขึ้น ซึ่งในโค้ดที่ผ่านมา กว่าเราจะรู้ว่าตรงนี้คือการโหลดรูปเราจะต้องพิจารณาโค้ดมากถึง 7 บรรทัด

Follow Me: ใช้เมธอด saveImage และพิจารณาโค้ดที่ได้

ในทำนองเดียวกัน เราจะใช้ saveImage มาแทนบรรทัด

```
try {
    File file = new File(outputPath);
    ImageIO.write(img, "png", file);
} catch(IOException ex) {
    System.err.println("Error writing image");
}
```

โดยเราจะแก้ให้เหลือเพียงบรรทัดเดียวเป็น

```
Util.saveImage(outputPath, img);
```

เช่นเดียวกัน ตัวโค้ดตรงนี้แสดงให้เห็นถึงงานที่ต้องการทำโดยชัดแจ้ง ซึ่งก็คือการบันทึกภาพลงไฟล์ ในขณะที่โค้ดก่อนหน้า ไม่ได้แสดงถึงเจตนาโดยชัดแจ้งและเราต้องอ่านโค้ดหลายบรรทัดเพื่อทำความเข้าใจ

หมายเหตุ โค้ดของพวกเราอาจจะต่างกันไปบ้าง ณ จุดนี้ขึ้นอยู่กับการประกาศเมธอด **saveImage** ของแต่ละคนในข้อที่ผ่านมา ในแบบฝึกหัดจะยึดถือตามที่แสดงอยู่ในข้อนี้ ส่วนผู้เรียนสามารถเลือกปรับมาเป็นแบบเดียวกับแบบฝึกหัดเพื่อให้เข้าใจไปในทางเดียวกันได้ง่ายขึ้น หรือจะรักษาแนวทางเดิมที่ตนถนัดไว้ก็ได้

ถึงตาของคุณแล้ว: ทดสอบโปรแกรม BinarizationWithUtil1

เรามาถึงจุดที่เหมาะสมในการทดสอบโปรแกรมของเราว่ายังให้ผลลัพธ์ที่ถูกต้องตามเดิมหรือไม่ ขอให้ผู้เรียนลองรันโปรแกรมเหมือนข้อ **ImageBinarization** เพื่อตรวจสอบความถูกต้องของผลลัพธ์ก่อนที่จะไปทำข้อต่อไป

5. เขียนเมธอดดึงข้อมูลภาพลงอาเรย์ [ImageArrayMethod]

การเขียนโปรแกรมเกี่ยวกับภาพเป็นงานใหญ่ มีขั้นตอนจำนวนมากที่ต้องทำ ความจำเป็นในการแบ่งงานออกเป็นเมธอด/ฟังก์ชันจึงมีสูงมาก ดังที่เราแสดงให้เห็นในข้อที่ผ่านมาว่า **“เมธอดและชื่อของมันช่วยให้โค้ดอ่านง่ายขึ้น”**

เราควรใช้แนวคิดในการแบ่งงานเป็นเมธอดอย่างจริงจัง ซึ่งในงานจัดการภาพหนึ่งในสิ่งที่นิยมทำก็คือการส่งวัตถุเกี่ยวกับข้อมูลภาพ เช่น อาเรย์ที่เก็บค่าพิกเซลไปให้เมธอดประมวลผลอย่างใดอย่างหนึ่ง

ในข้อที่แล้ว เราโอนงานการโหลดและเขียนภาพไปยังคลาส **Util** ทำให้โค้ดใน **main** อ่านง่ายขึ้นมาก แต่เมื่อเราพิจารณาลูปสำหรับเธรสโฮลด์ภาพ เราจะเห็นว่ากระบวนการในลูปดูซับซ้อนกว่าตัวเนื้อหาพอสมควร เพราะโดยหลักการเราต้องการเทียบและเปลี่ยนค่าพิกเซล แต่ในโค้ดตัวอย่างที่ให้ไว้ในข้อแรก **(ImageBinarization)** กลับวุ่นวายอยู่กับการ **getPixel** และ **setPixel** และการจัดการค่าผ่าน **pixelBuffer** ทำให้ใจความของลูปดูซับซ้อนเกินควร

เมื่อพบสถานการณ์เช่นนี้ เราควรพิจารณาจัดการกับสิ่งที่คล้าย **“เนื้อหาส่วนเกิน”** ออกไป และแทนที่ด้วยสิ่งที่เข้าใจง่ายและเขียนผิดพลาดยาก ซึ่งสิ่งที่นิยมทำกันบ่อย ๆ ในงานการวิเคราะห์ภาพก็คือการถ่ายโอนข้อมูลภาพทั้งหมดลง

ในอาเรย์สองมิติ และเข้าถึงพิกเซลผ่านอาเรย์นั้นโดยตรง แทนที่จะต้องคอยเรียก `getPixel` และ `setPixel` บ่อย ๆ

ในเมื่อการโอนข้อมูลภาพเป็นสิ่งที่ต้องทำบ่อย ๆ การสร้างเมธอดนี้ในคลาสอรรถประโยชน์จึงเป็นสิ่งที่เกิดขึ้นตามมาโดยธรรมชาติ

Follow Me: ตั้งเป้าหมายของเมธอด

ขอให้สังเกตว่าการใช้ `Raster` ดึงค่าจาก `BufferedImage` เราจะต้องคอยอ่านค่ามาใส่ที่พักข้อมูลและนำมาประมวลผลต่อ ซึ่งขั้นตอนตรงนี้ดูซับซ้อนพอสมควร หากเรานำงานคำนวณที่ซับซ้อนอื่นมาทำร่วมด้วย โปรแกรมของเราจะอ่านยาก เสี่ยงต่อการเขียนผิด และมันจะนำไปสู่โปรแกรมที่ด้อยคุณภาพในท้ายที่สุด ยิ่งไปกว่านั้น หากเราต้องอ่านค่าพิกเซลเดิมซ้ำหลายรอบ ก็จะทำให้โปรแกรมทำงานช้าลงด้วย

ดังนั้น แทนที่เราจะมามัวอ่านค่าที่ละพิกเซลและทำการคำนวณไปพร้อมกันในจุดเดียว เราจะสร้างอาเรย์สองมิติมาเก็บข้อมูลภาพทั้งหมดไว้ ต่อมาหากจะต้องอ่านค่าพิกเซล เราก็จะดึงค่าจากอาเรย์นี้แทน ไม่ต้องทำผ่าน `Raster` และที่พักข้อมูล

ด้วยเหตุนี้ เราจะกำหนดให้เมธอดรับพารามิเตอร์มาเป็น `BufferedImage` และคืนอาเรย์สองมิติที่เก็บค่าพิกเซลกลับไปเป็นผลลัพธ์ โดยประเภทข้อมูลในอาเรย์จะเป็น `int` ซึ่งเป็นข้อมูลแบบ 32 บิต แม้ว่าค่าในรูปภาพจะเป็น `byte` แบบ 8 บิตก็ตาม และเราควรจะประกาศเมธอดในคลาส `Util` เป็น

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

Follow Me: สร้างเมธอด `loadToArray` ให้สมบูรณ์

เพื่อที่จะโอนข้อมูลภายลงในอาเรย์ได้ เราจะต้องสร้างอาเรย์ที่มีขนาดสัมพันธ์กับขนาดภาพ ดังนั้นเราจะเริ่มจากการเก็บขนาดภาพไว้ในตัวแปร `width` และ `height`

```
Raster raster = img.getRaster();
int height = img.getHeight();
int width = img.getWidth();
int[][] I = new int[height][width];
```

ต่อจากนั้น เราจะสร้างที่พักข้อมูลพิกเซลอย่างที่เราเคยทำมาก่อน

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

สุดท้าย เราจะวนลูปอ่านค่าเก็บไว้ในที่พักข้อมูลแล้วจึงโอนค่าไปเก็บไว้ในอาเรย์ผลลัพธ์ I ด้วยลูปสองชั้น แล้วจึงคืนอาเรย์กลับไปเป็นผลลัพธ์ของเมธอด ดังแสดงข้างล่างนี้

```
for(int row = 0; row < height; ++row) {
    for(int col = 0; col < width; ++col) {
        raster.getPixel(col, row, pixelBuffer);
        I[row][col] = pixelBuffer[0];
    }
}
return I;
```

หมายเหตุ ชื่ออาเรย์นั้นจะเป็นอย่างอื่นก็ได้เหมือนตัวแปรทั่วไป แต่ในแบบฝึกหัดนี้จะนิยมใช้ตัว I เพราะเป็นสัญลักษณ์เดียวกับฟังก์ชันภาพในบริบทของคณิตศาสตร์

เรื่องน่ารู้: จำนวนบรรทัดในเมธอด

ทางเลือกในการเขียนโปรแกรมนั้นมีหลายทาง ในครั้งนี้เราจะลองเลือกทางที่แบ่งงานออกเป็นขั้นตอนที่ค่อนข้างสั้น ดูเข้าใจง่าย และพยายามจัดขั้นตอนลงเป็นเมธอดที่สั้น ซึ่งเมธอดที่สั้นนี้มักจะเป็นหนึ่งในมาตรวัดคุณภาพของโค้ดทีมพัฒนาซอฟต์แวร์ และในแบบฝึกหัดนี้เราจะพยายามควบคุมไม่ให้เมธอดแต่ละอันยาวเกิน 15 บรรทัด (ไม่นับบรรทัดชื่อเมธอด วงเล็บปิด และบรรทัดเปล่า)

เจาะลึก: การคำนวณตัวเลขจำนวนเต็มในจาวาจะเป็นแบบ `int` เกือบตลอดเวลา

ในเมธอดอ่านภาพลงในอาเรย์ หลายคนอาจจะรู้สึกว่าการใช้ `int` ทำให้สิ้นเปลืองหน่วยความจำ ซึ่งอันที่จริงก็อาจจะกล่าวได้ว่าการใช้ `int` ทำให้เราต้องใช้หน่วยความจำเพิ่มมากกว่า `byte` ถึง 4 เท่า แต่นั่นอาจจะถือเป็นทางเลือกที่คุ้มค่า เนื่องจากในภาษาจาวานั้นแม้เราจะนำข้อมูลชนิด `byte` มาบวกลบคูณหารกัน จาวาก็จะเปลี่ยนข้อมูลเป็น `int` ก่อนแล้วจึงดำเนินการบวกลบคูณหารต่อไป ดังแสดงในตัวอย่างต่อไปนี้

```
byte b1 = 5;
byte b2 = 7;
byte b3 = b1 + b2;
```

จากตัวอย่างข้างบน บรรทัดที่สามจะผิดไวยากรณ์ภาษา ไม่สามารถคอมไพล์โปรแกรมได้ เพราะ **b1 และ b2 จะถูกเปลี่ยนเป็น int ก่อนบวก และผลบวกของมันจึงเป็น int ตามไปด้วย ทำให้เราไม่สามารถเก็บผลลัพธ์ไว้ใน byte เนื่องจากมันมีขอบเขตข้อมูลที่แคบกว่า int**

แน่นอนว่าเราสามารถที่จะบังคับผลลัพธ์ให้เป็น byte ด้วยการแคสต์ข้อมูล เช่น

```
byte b3 = (byte)(b1 + b2);
```

แต่การแคสต์ข้อมูลไปมาอาจทำให้โค้ดดูซับซ้อนและทำงานช้ากว่าเดิมได้ ในยุคที่เครื่องคอมพิวเตอร์มีหน่วยความจำมาก การเปลี่ยนข้อมูลให้เป็น int อาจจะเป็นทางเลือกที่เหมาะสมกว่าการพยายามประหยัดหน่วยความจำก็เป็นได้

6. การเธรสโฮลด์ภาพผ่านข้อมูลอาเรย์ [ArrayThreshold]

จากแนวทางการเปลี่ยนแปลงที่ผ่านมา ทำให้เราต้องทำเธรสโฮลด์ค่าในอาเรย์แทนที่จะเป็นค่าในที่พักข้อมูลแบบเดิม ในแนวทางนี้เราจะแบ่งงานออกเป็นสองส่วนคือ (1) การโอนข้อมูลภาพจาก BufferedImage ลงในอาเรย์สองมิติ และ (2) การแปลงค่าพิกเซลให้ได้ภาพทวิภาคโดยใช้ข้อมูลในอาเรย์

Follow Me: เรียกใช้ loadToArray ในเมธอด main

งานนี้ตรงไปตรงมามาก หลังจากที่โหลดรูปภาพและได้ BufferedImage แล้ว เราสามารถที่จะโอนย้ายข้อมูลภาพลงในอาเรย์สองมิติด้วยการเรียก loadToArray ได้ทันทีในลักษณะดังโค้ดข้างล่าง

```
BufferedImage img = Util.loadImage(inputPath);
int[][] I = Util.loadToArray(img);
```

เพียงเท่านี้ เราก็จะได้ข้อมูลภาพอยู่ในอาเรย์สองมิติที่พร้อมใช้งานโดยสะดวก

Follow Me: แปลงค่าพิกเซลในอาเรย์

เนื่องจากเรามีข้อมูลพิกเซลที่สามารถอ้างถึงได้โดยสะดวกในอาเรย์แล้ว เราจึงสามารถเทียบค่าพิกเซลกับค่าเธรสโฮลด์ T ได้โดยสะดวก และได้ลูปที่ดูกระชับขึ้นดังนี้

```
int height = img.getHeight();
int width = img.getWidth();
```

```
for(int row = 0; row < height; ++row) {
    for(int col = 0; col < width; ++col) {
        if(I[row][col] < threshold)
            I[row][col] = 0;
        else
            I[row][col] = 255;
    }
}
```

7. เมธอดถ่ายข้อมูลจากอาเรย์ลงใน BufferedImage [ArrayToImage]

อุปสรรคอย่างหนึ่งในการโอนข้อมูลภาพลงในอาเรย์ก็คือว่า ผลลัพธ์จะอยู่ในอาเรย์ ในขณะที่เราอยากให้ผลลัพธ์อยู่ใน BufferedImage เพื่อความสะดวกในการบันทึกข้อมูลลงไฟล์ผ่าน ImageIO ด้วยเหตุนี้เราจึงต้องสร้างเมธอดเพื่อแปลงข้อมูลอาเรย์กลับไปเป็น BufferedImage

การแปลงข้อมูลกลับไปมาเช่นนี้ อันที่จริงนับเป็นภาระการคำนวณอย่างหนึ่ง แต่โดยรวมแล้วถือว่าคุ้มค่ามาก โดยเฉพาะตอนที่เราต้องเรียกใช้ getPixel จากราสเตอร์บ่อย ๆ เนื่องจากเมธอด getPixel ทำงานได้ช้าเมื่อเทียบกับการอ้างถึงค่าพิกเซลในอาเรย์สองมิติโดยตรง

ความคุ้มค่าเช่นนี้ ยังครอบคลุมถึงเหตุการณ์ที่เราใช้ไลบรารีอย่าง OpenCV ด้วย เนื่องจากว่าการอ่านค่าพิกเซลจากภาพผ่านเมธอด OpenCV มีการสื่อสารข้ามบริบทของจาวาไป C++ ไปมา ดังนั้นในกรณีที่ต้องอ่านค่าพิกเซลเดิมซ้ำกันหลายครั้ง การโอนค่าไปมาระหว่างอาเรย์นอกจากจะให้โค้ดที่อ่านง่ายก็มักจะให้ประสิทธิภาพที่ดีขึ้นด้วย

Follow Me: พิจารณาการสร้างเมธอด saveToImage ในคลาส Util

เราจะทำการเพิ่มความสามารถของคลาส Util อีกครั้ง โดยเราจะสร้างเมธอด saveToImage เพิ่มเข้าไปใน Util แต่ก่อนอื่นเราต้องเข้าใจเกี่ยวกับผลลัพธ์ ซึ่งเป็นวัตถุชนิด BufferedImage ว่าจะให้มันเขียนทับลงไปในภาพอันเดิม หรือเป็นการสร้างวัตถุ BufferedImage อันใหม่ขึ้นมา

ทั้งสองทางเลือกนั้นจะนำไปสู่การเขียนโค้ดที่แตกต่างกัน ตลอดจนมีจุดประสงค์และข้อดีข้อเสียที่แตกต่างกันมาก โดยในแบบที่เขียนทับอันเดิม เรามีจุดประสงค์ที่จะใช้วัตถุเดิมซ้ำ ทำให้ประหยัดหน่วยความจำ แต่มันมีข้อจำกัดว่าเราจะเสียภาพเดิมไป และถ้าหากอาเรย์ภาพของเราเป็นของใหม่ เช่น มีขนาดหรือโมเดลสีที่ต่างจากภาพเดิม เราจะใช้วิธีนี้ไม่ได้

ส่วนการสร้างวัตถุ **BufferedImage** ขึ้นมาใหม่ เราจะไม่มีข้อจำกัดเรื่องความแตกต่างจากวัตถุภาพที่มีมาก่อนหน้า แต่นั่นหมายถึงการที่เราต้องการหน่วยความจำเพิ่มขึ้นมาอีกชุดหนึ่ง ซึ่งอาจจะเป็นปัญหาในระบบที่มีหน่วยความจำน้อย และถ้าหากภาพมีขนาดใหญ่ การสร้างพื้นที่ข้อมูลใหม่จะใช้เวลาพอสมควร อย่างไรก็ตาม ในกรณีที่เวลาและหน่วยความจำไม่ใช่ประเด็นวิธีนี้ถือว่าสะดวกกว่ามาก

แล้วเราจะเลือกทางไหนดี? อันที่จริงทางที่สองดูสะดวกดี แต่เมธอดที่จะสร้างขึ้นมาจะมีพารามิเตอร์และวิธีเขียนที่ซับซ้อนกว่า เป็นต้นว่า เราต้องกำหนดเพิ่มเติมว่าจากอาเรย์ที่ได้ไปนั้น เราจะใช้โมเดลสีแบบใด และเราต้องนำโมเดลสีนั้นไปเป็นตัวระบุการสร้าง **BufferedImage** อันใหม่ขึ้นมา สืบเนื่องจากเทคนิคในการเขียนโปรแกรมของวิธีนี้ดูซับซ้อนสำหรับคนที่มีประสบการณ์น้อย ดังนั้นเราจะเลือกใช้วิธีแรกไปก่อนจนกว่าจะถึงเวลาที่เราจำเป็นต้องใช้วิธีที่สอง เราจึงจะกลับมาเรียนรู้วิธีสร้างวัตถุ **BufferedImage** กันอีกครั้ง

จากเป้าหมายในการใช้วัตถุ **BufferedImage** ซ้ำ เราจะประกาศเมธอดในรูป **`public static void saveToImage(int[][] I, BufferedImage img)`** ซึ่งพารามิเตอร์ **I** เป็นอาเรย์ที่เก็บค่าพิกเซล และ **img** เป็นวัตถุภาพที่เราจะเปลี่ยนค่าพิกเซลมันให้เป็นไปตามค่าใน **I** ซึ่งในกรณีนี้ **img** จะต้องเป็นภาพเฉดเทาที่เก็บเลขจำนวนเต็ม

หมายเหตุ เมธอดนี้ไม่ได้คืนค่าใดกลับไป เพราะผลลัพธ์ถูกเขียนลงในตัวแปรวัตถุ **img** แล้ว จุดนี้แตกต่างจากเมธอด **loadToArray** ซึ่งสร้างอาเรย์ขึ้นมาในเมธอดและคืนอาเรย์นั้นกลับไปเป็นผลลัพธ์

Follow Me: สร้างเมธอด saveToImage ในคลาส Util

ในเมธอดนี้เราต้องการเขียนค่าลงใน **BufferedImage** เราจึงต้องใช้ราสเตอร์แบบเขียนได้ ดังนั้นงานแรกของเราจึงเป็น

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

ต่อมาเราจะเตรียมที่พักข้อมูลพิกเซลสำหรับสำเนาค่าลงในภาพ

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

ตามด้วยการอ่านขนาดภาพจาก **img**

```
int height = img.getHeight();
int width = img.getWidth();
```

สุดท้าย เราจะวนลูปเพื่ออ่านค่าพิกเซลจากอาเรย์ **I** แล้วนำไปเขียนลงใน **img** ผ่านที่พักข้อมูลที่เตรียมไว้

```
for(int row = 0; row < height; ++row) {
    for(int col = 0; col < width; ++col) {
        pixelBuffer[0] = I[row][col];
        raster.setPixel(col, row, pixelBuffer);
    }
}
```

8. สร้างคลาสสำหรับการทำเธรสโฮลด [Thresholder]

การเธรสโฮลด์ภาพเป็นสิ่งที่พบได้บ่อย ซึ่งการสร้างภาพทวิภาคก็อาศัยกระบวนการนี้ ดังนั้นเราจะสร้างเมธอดสำหรับการเธรสโฮลด์ไว้ แต่เราจะไม่สร้างไว้ในคลาส **imageutil.Util** เราจะสร้างคลาสใหม่เป็น **imageutil.Thresholder** เพราะการเธรสโฮลด์นั้นมีหลายวิธีการ หากเราจะสร้างวิธีเธรสโฮลด์ที่แตกต่างไปจากเดิม เช่น วิธีของ **Otsu** เราก็สามารถนำมารวมไว้ในนี้ได้

แบบฝึกหัดข้อนี้จะเป็นการทดสอบว่าเราเข้าใจกระบวนการพัฒนาโปรแกรมที่ผ่านมา และนำมาประยุกต์ใช้ในบริบทที่แตกต่างจากเดิมได้หรือไม่

ถึงตาของคุณแล้ว: สร้างคลาส Thresholder ในแพ็คเกจ imgageutil

ลองทบทวนสิ่งที่เคยทำมาตอนนร้างคลาส **Util** แล้วสร้างคลาส **Thresholder** ขึ้นมาในแพ็คเกจดังกล่าว

ถึงตาของคุณแล้ว: สร้างเมธอด binarize ใน Thresholder

กำหนดเมธอด binarize ภายใต้รูปแบบ

```
public static void binarize(int[][] I, final int T,
        int outputLow, int outputHigh)
```

โดยที่ I เป็นค่าพิกเซลซึ่งเป็นทั้งข้อมูลเข้าและที่เก็บผลลัพธ์ (คือผลการแปลงเป็นภาพทวิภาคจะอยู่ในนี้)

T คือค่าเธรสโฮลด์ที่จะใช้แบ่งผลลัพธ์ออกเป็นสองกลุ่มคือกลุ่มล่างและกลุ่มบน

outputLow คือค่าผลลัพธ์ของพิกเซลที่มีค่าน้อยกว่า T เช่น ถ้า outputLow = 0 หากพิกเซลใดมีค่าน้อยกว่า T ผลลัพธ์ก็จะเป็น 0 แต่ถ้า outputLow = 1 ผลลัพธ์จากพิกเซลดังกล่าวก็จะมีค่าเป็น 1

outputHigh คือค่าผลลัพธ์ของพิกเซลที่มีค่ามากกว่าหรือเท่ากับ T เช่น ถ้า outpuHigh = 1 หากพิกเซลใดมีค่าน้อยกว่า T ผลลัพธ์ก็จะเป็น 1 แต่ถ้า outputLow = 255 ผลลัพธ์จากพิกเซลดังกล่าวก็จะมีค่าเป็น 255

สาเหตุที่เราเพิ่มพารามิเตอร์ outputLow และ outputHigh ขึ้นมาก็เพราะว่าเราต้องการทำให้เมธอดของเรายืดหยุ่นสามารถใช้ได้กับหลายบริบท ยกตัวอย่างเช่น ถ้าเราอยากกลับค่าให้พิกเซลโทนมืด (คือพวกที่ค่าน้อยกว่า T) เป็นพิกเซลสีขาวในภาพผลลัพธ์ ในขณะที่พิกเซลโทนสว่างกลายเป็นสีดำ เราจะกำหนดค่า outputLow = 255 และ outputHigh = 0 อย่างไรก็ตาม ในตัวอย่างที่เรากำลังศึกษาอยู่นี้ เราจะส่งค่า outputLow = 0 และ outputHigh = 255

9. ใช้ประโยชน์จากเมธอดทั้งหมดที่สร้างไว้ [FinalThreshold]

หลังจากที่เราเตรียมการไว้หลายอย่าง เราจะเรียกใช้เมธอดเหล่านี้ร่วมกันในคลาสใหม่ชื่อ BinarizationWithUtil2 นั่นหมายความว่าเมธอด main อันใหม่ของเราจะสร้างภาพทวิภาคผ่านการเรียกเมธอดพื้นฐานที่เราสร้างขึ้นหลาย ๆ อันต่อกัน ซึ่งสรุปได้เป็นลำดับดังนี้

(1) โหลดภาพจากพาธที่ผู้ใช้กำหนดมาไว้ใน BufferedImage

(2) แปลง BufferedImage เป็นอาเรย์สองมิติ

(3) นำอาเรย์สองมิติดังกล่าวไปผ่านการเธรสโฮลด์ โดยกำหนดให้ `ouputLow` และ `outputHigh` เป็น `0` และ `255` ตามลำดับ ส่วนค่าเธรสโฮลด์เป็นสิ่งที่กำหนดมาโดยผู้ใช้

(4) แปลงผลลัพธ์ที่ได้ไปเป็น `BufferedImage`

(5) บันทึก `BufferedImage` ดังกล่าวลงไฟล์ตามพาธที่ผู้ใช้กำหนดให้

ถึงตาของคุณแล้ว: นำสิ่งต่าง ๆ มารวมกันและลองทดสอบโปรแกรมว่าได้ผลลัพธ์ตามที่คาดหวังไว้หรือไม่

ในข้อแรก เราสร้างภาพทวิภาคด้วยการทำทุกอย่างในเมธอด `main` แต่ต่อมาเราได้พยายามโอนงานออกจากเมธอด `main` เพื่อให้การคำนวณหลักของเราดูเรียบง่าย พร้อมทั้งลดความซ้ำซ้อนในการเขียนโค้ดพื้นฐานอย่างการโหลดภาพในโปรแกรมอื่น ๆ ที่จะมีต่อมา กล่าวคือเราได้สร้างเมธอดอรรถประโยชน์เกี่ยวกับการประมวลผลภาพเพื่อที่จะได้เรียกใช้ได้อีกบ่อย ๆ ในโปรแกรมอื่น

แบบฝึกหัดข้อนี้ได้ชี้แนะแนวทางในการเรียกใช้งานเมธอดเหล่านั้นไว้แล้ว เหลือแต่เราที่จะต้องทำความเข้าใจและเรียกใช้งานให้ถูกต้อง แน่นอนว่าการแบ่งงานไปเป็นส่วนย่อยในเมธอด ไม่ควรทำให้ผลลัพธ์เปลี่ยนแปลงหรือผิดพลาด ดังนั้นก่อนที่จะก้าวไปเขียนโปรแกรมในแบบฝึกหัดถัดไป เราจะต้องทดสอบโปรแกรมของเราให้เรียบร้อย นั่นคือ ให้เราลองรันโปรแกรมของเราตามแนวทางที่ให้ไว้ในส่วน **“ถึงตาของคุณแล้ว”** ในข้อแรก และดูว่าผลลัพธ์ที่ได้ถูกต้องอย่างที่ควรเป็นหรือไม่

เรื่องน่ารู้: โค้ดของเราง่ายลงกว่าเดิมแค่ไหน

หลังจากที่เราใช้เวลาในการปรับสภาพโค้ดอยู่นานพอสมควร เราควรจะทบทวนสิ่งต่าง ๆ ที่เราทำและผลลัพธ์ที่เกิดขึ้นเพื่อให้เราได้ตระหนักถึงผลของงาน โดยงานของเรานั้นอาจจะแบ่งได้เป็น `3` เวอร์ชันใหญ่ ๆ คือ เวอร์ชันที่ได้จากแบบฝึกหัดข้อหนึ่ง `(ImageBinarization)` เวอร์ชันสองจากโปรแกรม `BinarizationWithUtil1` และเวอร์ชันสุดท้ายจากโปรแกรม `BinarizationWithUtil2`

ลองสำรวจเมธอด `main` เวอร์ชันแรกและสุดท้ายของเราดูสิว่ามันต่างกันขนาดไหน!