การเขียนโปรแกรมคอมพิวเตอร์ 2

Computer Programming II

ชนิดข้อมูล อาเรย์ และกฎพื้นฐานในจาวา

Data Types, Array, and Basic Java Rules

ภิญโญ แท้ประสาทสิทธิ์

ภาควิชาคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศิลปากร

(taeprasartsit_p at silpakorn dot edu, pinyotae at gmail dot com)

Web site: http://webserv.cp.su.ac.th/~pinyotae/compro2/

Facebook group: ComputerProgramming@CPSU

สัปดาห์ที่ 2

เราจะเรียนอะไรในวันนี้

- ชนิดข้อมูลในจาวา
  - แบบพื้นฐาน (primitive data type)
  - แบบอ้างอิงวัตถุ (object reference data type)
- สัญพจน์ (Literal)
- อาเรย์
  - การประกาศอาเรย์ในจาวา
  - การใช้งาน
  - เหตุการณ์ผิดปรกติเมื่อใช้งานอาเรย์ผิดพลาด

ชนิดของข้อมูลในจาวา

- ในภาษาจาวา ชนิดข้อมูล (data type) แบ่งออกเป็นสองกลุ่ม
  - แบบพื้นฐาน (primitive data type)
  - แบบอ้างอิงวัตถุ (object-reference data type)
- สำหรับแบบพื้นฐานนั้น มีอยู่ 4 จำพวก 8 ชนิดคือ
  1. กลุ่มจำนวนเต็ม: int, long, short, และ byte
  2. กลุ่มจำนวนจริง: float และ double
  3. อักขระ (character)
  4. ค่าตรรกะ (boolean)
- ส่วนแบบอ้างอิงวัตถุมีชนิดข้อมูลจำนวนมาก (ในทางทฤษฎี มีได้ไม่จำกัด)
  - ทั้งแบบที่นิยามไว้แล้วในคลาสไลบรารี
  - และแบบที่เรานิยามขึ้นมาเพิ่มใหม่ได้เรื่อย ๆ ไม่จำกัด

ว่ากันด้วยเรื่องข้อมูลกลุ่มจำนวน

- ในภาษาจาวามีทั้งจำนวนเต็มและเลขทศนิยมคล้ายกับภาษาซี
- แต่ก็มีความแตกต่างกันอยู่บ้าง
  - โดยเฉพาะจำนวนเต็มจะมีความแตกต่างเรื่องเครื่องหมาย
  - ส่วนการดำเนินการบนตัวเลขนั้น ถ้าเป็นภาษาจาวาเราจะไม่สามารถหารด้วยศูนย์ได้เลย รับประกันว่าโปรแกรมจะแครช (crash)
  - ส่วนในภาษาซีถ้าหารด้วย 0 โปรแกรมอาจจะไม่แครช แต่ผลลัพธ์ที่ตามมาอาจจะผิด
- นอกจากนี้ในไลบรารีมาตรฐานของภาษาจาวา เรายังมีข้อมูลแบบคลาสจำพวก BigDecimal ซึ่งทำให้เราจัดการเลขจำนวนเต็มขนาดใหญ่ได้
  - แต่เราจะยังไม่พูดถึงมันในวิชานี้ เราจะใช้จำนวนแบบพื้นฐานเป็นหลัก
  - ถ้าแบบคลาสก็จะเป็นพวก Integer, Double ซึ่งจะกล่าวถึงภายหลัง

ความเป็นบวก/ลบของชนิดข้อมูลกลุ่มจำนวนเต็ม

- ในภาษาจาวา ชนิดข้อมูลแบบจำนวนเต็ม จะเป็นแบบ signed เสมอ
  - คือแสดงได้ทั้งเลขบวกและลบ
  - ต่างกับภาษาซี ซึ่งอาจจะเป็นแบบ unsigned ก็ได้
  - ในภาษาซี เราสามารถบังคับให้ตัวแปรจำนวนเต็ม จัดการเฉพาะเลขศูนย์และบวกได้ แต่ในจาวาตัวแปรจำนวนเต็มจะจัดการเลขลบได้ด้วยเสมอ
- ความยุ่งยากของ unsigned ในภาษาซีมีประโยชน์ในกรณีที่เราไม่ต้องการเก็บเลขลบ
  - การใช้ unsigned จะสอดคล้องกับแนวคิดของการนับจำนวน เช่น เราอาจจะอยากใช้ unsigned กับจำนวนประชากรซึ่งไม่มีทางเป็นลบ
  - ค่าสูงสุดของข้อมูลแบบ unsigned จะสูงกว่าแบบ signed เพราะเอาพื้นที่สำหรับเก็บค่าลบไปใช้เก็บค่าบวกได้

ค่าอักขระในภาษาจาวา

- เราสร้างตัวแปรค่าอักขระในภาษาจาวาได้แบบเดียวกับภาษาซี

```
char c = ‘A’;
```

- แต่ในจาวาจะดีกว่าหน่อยตรงที่อักขระมันเป็นแบบยูนิโค้ด (Unicode) ทำให้เก็บภาษาอื่น ๆ ได้อีกมาก (รวมถึงภาษาไทยด้วย)
- อย่างไรก็ตาม การเก็บภาษาอื่นที่ไม่ใช่ภาษาอังกฤษ เราอาจจะต้องใช้รหัสตามมาตรฐานยูนิโค้ด ทำให้อาจจะดูยากสำหรับคนที่ไม่ชำนาญ
  - แต่ถ้าหากซอฟต์แวร์ที่เราใช้เขียนรองรับยูนิโค้ดโดยตรง เราอาจจะใส่อักขระภาษาไทยหรือภาษาอื่น ๆ เข้าไปได้โดยตรง (แต่ก็ไม่ควรทำ)
  - ในกรณีที่ตัดสินใจใช้ยูนิโค้ด การกำหนดและแสดงค่า ก จะเป็นแบบนี้

```
char c = '\u0E01';
System.out.println(c);
```

ค่าตรรกะ (boolean)

- เป็นชนิดข้อมูลเกี่ยวกับ*ค่าความจริง* ซึ่งไม่มีชนิดข้อมูลนี้ในภาษาซี
- ค่าที่เป็นไปได้ของชนิดข้อมูลแบบนี้มีเพียงสองค่าคือ *true* และ *false*
- เรานิยมเรียกชนิดตัวแปรนี้ทับศัพท์ว่า *บูลีน*
- ตัวแปรบูลีนมีการประยุกต์ใช้งานที่หลากหลาย เช่น
  - ใช้บันทึกเรื่องที่มีสองสถานะ เช่น มี/ไม่มี, เปิด/ปิด, หรือ ไทย/ต่างชาติ
  - ใช้ค่า true เพื่อแทนว่าเป็นความจริงตลอด (เหมาะกับลูป while ที่จะวนแบบไม่รู้จบ)
  - ใช้บันทึกว่าเคยเกิดเหตุการณ์ใดเหตุการณ์หนึ่งขึ้นหรือยัง เช่น พบว่ามีเลขคู่หรือไม่ หรือมีพื้นที่น้ำท่วมขังเกิดขึ้นในฤดูฝนหรือยัง เป็นต้น

ตัวอย่างการใช้บูลีน

- **เป้าหมาย** เราจะรับเลขเข้ามา 10 ค่า และจะหาผลบวกของเลขทั้งหมด สุดท้ายเมื่อพิมพ์ผลบวกออกมาแล้ว ...
  - ถ้าพบว่ามีเลขติดลบอยู่ในกลุ่มตัวเลขที่ให้มา จะพิมพ์ว่า "Negative number was found" (ไม่ต้องใส่อัญประกาศในผลลัพธ์)
  - ถ้าไม่พบเลขติดลบจะพิมพ์ว่า "No negative number"
- **วิเคราะห์** การหาผลบวกเป็นเรื่องง่าย ๆ ธรรมดา ส่วนการบันทึกว่ามีเลขติดลบหรือไม่นั้น เราจะใช้เทคนิคทำนองนี้
  1. ตั้งตัวแปรบูลีนเพื่อบันทึกว่าพบเลขติดลบหรือไม่ ซึ่งในตอนเริ่มเรายังไม่พบ จึงตั้งค่าตัวแปรให้เป็น false ก่อน
  2. ถ้าหากในระหว่างการวนลูปพบเลขติดลบให้แก้ค่าตัวแปรนั้นเป็น true
  3. พอจบลูปและพิมพ์ผลบวกเสร็จ ให้เช็คค่าตัวแปรบูลีนนั้นว่าเป็น true หรือ false

โค้ดการประยุกต์ใช้บูลีน

```
int sum = 0;
boolean negativeFound = false;

for(int i = 0; i < 10; ++i) {
  int x = scan.nextInt();
  sum += x;
  if(x < 0)
    negativeFound = true;
}
System.out.println(sum);
if(negativeFound)
  System.out.println("Negative number was found");
else
  System.out.println("No negative number");
```

เราเซ็ตตรงนี้เป็น false ไว้ก่อน ซึ่งแปลว่า ยังไม่เคยเจอเลขติดลบ

ถ้าพบเลขติดลบเราก็เปลี่ยนค่าเป็น true เพื่อแสดงว่าเคยเจอเลขติดลบ

ตรวจสอบค่าก่อนเลือกพิมพ์ผลลัพธ์

แนวคิดสำคัญในตัวอย่างที่แล้ว

- คนที่จะใช้เทคนิคนี้แรก ๆ จะมัวแต่กังวลโค้ดในลูปตรงนี้

```
if(x < 0)
  negativeFound = true;
```

- เพราะจะมัวแต่คิดว่า เขาควรจะเขียนแบบข้างล่างนี้ต่างหาก (ซึ่งผิด)

```
if(x < 0)
  negativeFound = true;
else
  negativeFound = false;
```

- ตรงท่อน else สีแดงนั้นทำให้ผลลัพธ์ผิด เพราะเราใช้ตัวแปรผิดไปจากความหมายที่ควรเป็น
  - ความหมายที่ควรเป็นคือตัวแปรใช้จำว่า 'เคย' พบเลขติดลบหรือไม่
  - ถ้าเปลี่ยนกลับเป็น false ได้แสดงว่าเราเปลี่ยนจาก 'เคย' เป็น 'ไม่เคย'

เรื่องที่คนมักจะกังวลแบบผิด ๆ

- ที่ตรงเดิม คือ if ในลูปตรงนี้ ยังสร้างความกังวลให้มือใหม่อีกอย่างหนึ่ง

```
if(x < 0)
  negativeFound = true;
```

- มือใหม่จำนวนมากจะกังวลว่า ถ้ามีเลขติดลบหลายตัว โปรแกรมจะทำคำสั่ง `negativeFound = true;` ซ้ำหลายครั้ง
  - มือใหม่จะรู้สึกว่ามันผิดที่จะเซ็ตค่าเป็น `true` ซ้ำหลายครั้ง
  - เขาจะมัวแต่วุ่นวายกับการหาทางป้องกันการเซ็ตค่าซ้ำ
- แต่อันที่จริง เราไม่ต้องกังวลกับการเซ็ตค่าซ้ำหรอก เพราะความหมายและเจตนาของตัวแปรยังคงเดิม ถูกต้องทุกอย่าง
  - เพราะค่าที่ซ้ำเป็น true ยังคงความหมายไว้ว่า 'เคย' พบเลขติดลบ
  - จะเขียนค่าซ้ำกี่ครั้งความหมายก็เหมือนเดิมและถูกต้องตามการใช้งาน

เทคนิคการเก็บค่าสถานะไว้รอสรุปผลยังคงคล้ายเดิม

- ตอนเรียนคอมโปร 1 เราพูดถึงเทคนิค 'ตัวแปรเก็บสถานะ' ซึ่งเอาไว้เก็บบันทึกเหตุการณ์ต่าง ๆ
  - ซึ่งเราอาจจะนำมาใช้เพื่อสรุปเหตุการณ์รอบเดียวในตอนท้าย
  - หรือเอาไว้ตรวจสอบในสถานการณ์ทั่วไป
- ลักษณะการใช้ตัวแปรบูลีนที่แสดงให้ดูนั้น เป็นการใช้ตัวแปรเก็บสถานะแบบหนึ่ง ซึ่งเป็นการเก็บเพื่อรอสรุปผล
- แต่การใช้บูลีนมันจะสื่อความหมายได้ชัดเจนกว่าในกรณีนี้
  - เพราะเราบันทึกเหตุการณ์ที่มีแค่สองด้านที่เป็นขั้วตรงข้าม
  - คือ 'เคย' หรือ 'ไม่เคย' พบเลขติดลบ
  - เราเพียงกำหนดเป้าไว้ก่อนว่าในตัวแปร negativeFound นี้ ถ้าเป็น true แสดงว่าเคย เป็น false แสดงว่าไม่เคย
  - จากนั้นเราก็ใช้ตัวแปรให้สอดคล้องกับเจตนานี้ไปเรื่อย ๆ

เราจะเรียนอะไรในวันนี้

- ชนิดข้อมูลในจาวา
  - แบบพื้นฐาน (primitive data type)
  - แบบอ้างอิงวัตถุ (object reference data type)
- **สัญพจน์ (Literal)**
- อาเรย์
  - การประกาศอาเรย์ในจาวา
  - การใช้งาน
  - เหตุการณ์ผิดปรกติเมื่อใช้งานอาเรย์ผิดพลาด

สัญพจน์ (Literal)

- คือสิ่งที่เป็นเหมือนกับ "ค่าคงที่" ที่เราระบุไว้ในตัวโค้ด เช่น ถ้าเราเขียนว่า int x = 7; หรือ double k = 3.5; หรือ String str = "Silpakorn";
  - แบบนี้ 7, 3.5 และ `"Silpakorn"` ก็คือสัญพจน์
- สัญพจน์จะคงคุณสมบัติตามชนิดที่แท้จริงของมันไว้ ไม่ต่างกับตัวแปร
  - เช่น เราสามารถเขียนว่า `"Silpakorn".length()` ได้ ซึ่งมันจะมีเท่ากับ 9
  - กล่าวคือมันทำตัวเหมือนกับสตริงตามรูปแบบชนิดที่แท้จริงของมัน
  - แต่เราจะไม่สามารถเขียนว่า 7.length() ได้เพราะ int เป็น primitive data type ไม่มีคุณสมบัติอื่นที่ซ่อนอยู่ให้เรียกใช้ได้ในรูปแบบนี้
  - อย่างไรก็ตาม ในภาษาที่ใหม่กว่าอย่าง C# ข้อมูลชนิด int และ double จะมีคุณสมบัติพิเศษอื่น ๆ ที่คอมไพเลอร์ให้เรียกใช้ได้ แต่ไม่มีในจาวา

เราจะเรียนอะไรในวันนี้

- ชนิดข้อมูลในจาวา
  - แบบพื้นฐาน (primitive data type)
  - แบบอ้างอิงวัตถุ (object reference data type)
- สัญพจน์ (Literal)
- **อาเรย์**
  - การประกาศอาเรย์ในจาวา
  - การใช้งาน
  - เหตุการณ์ผิดปรกติเมื่อใช้งานอาเรย์ผิดพลาด

อาเรย์ 1 มิติ

- อาเรย์เป็นข้อมูลประเภทคลาสแบบหนึ่งทำให้ต้องสร้างเป็นวัตถุขึ้นมา
- ถึงแม้ว่าตอนนี้เราจะยังไม่รู้ว่าคลาสหรือวัตถุคืออะไร เราก็สามารถใช้อาเรย์ได้อย่างถูกต้อง เพราะพื้นฐานตรงนี้ตรงไปตรงมาพอสมควร
- สมมติว่าเราต้องการสร้างอาเรย์จำนวนเต็มที่มีความยาว 10 ช่อง เราจะเขียนว่า

```
int[] A = new int[10];
```

- ถ้าเราต้องการสร้างอาเรย์ของเลขทศนิยมแบบ double ความยาว 1,000,000 ช่อง เราจะเขียนว่า

```
double[] A = new double[1000000];
```

เรื่องลึก ๆ เกี่ยวกับการสร้างอาเรย์

- เรื่องที่สะดวกในภาษาจาวาก็คือว่า เราสร้างอาเรย์ขนาดใหญ่ ๆ ไว้ในตัวเมธอด main ได้เลย ไม่ต้องยกออกไปข้างนอกแบบภาษาซี
- คือในภาษาซี เราอาจจะเคยเขียนแบบนี้มาก่อน

```
int A[1000000];
void main() {
    ...
}
```

- แต่พอมาเป็นจาวาเราเขียนแบบนี้ได้เลย

```
public static void main(String[] args) {
    int[] A = new int[1000000];
}
```

- ที่เป็นอย่างนี้เพราะสิ่งที่เกิดขึ้นภายในจาวากับซีแตกต่างกัน

แล้วการสร้างอาเรย์ในจาวากับซีมันต่างกันอย่างไร

- ในกรณีของจาวา เราจะสร้างพื้นที่อาเรย์ตัวจริงไว้ในหน่วยความจำกลุ่มที่เรียกว่า ฮีป (Heap) ที่อนุญาตให้สร้างพื้นที่ใหญ่ ๆ ได้
- แต่ในภาษาซี ถ้าเราสร้างอาเรย์ด้วยโค้ดที่อยู่ในลักษณะดังกล่าว มันจะสร้างอาเรย์ไว้ในหน่วยความจำกลุ่มที่เรียกว่า สแต็ก (Stack)
  - พื้นที่ในสแต็กมีแนวโน้มจะมีขนาดเล็กกว่าฮีป
  - แต่การบริหารจัดการข้อมูลในสแต็กจะมีประสิทธิภาพและมักรวดเร็วกว่า
- ภาษาซีเกิดมาในยุคที่เครื่องคอมช้าและมีหน่วยความจำน้อยมาก
  - วิธีของซีจึงเหมาะกับสถานการณ์ในสมัยนั้น แต่โปรแกรมเมอร์จะรู้สึกไม่ค่อยสะดวก ภาษาจาวามาทีหลังต้องการอะไรที่สะดวกและปลอดภัยขึ้น
- แต่ภาษาซีก็มีวิธีเขียนแบบให้อาเรย์ไปอยู่ในฮีปเช่นกัน ทว่าการเขียนและจัดการจะวุ่นวายกว่ามาก (ถึงขนาดที่เราตัดสินใจไม่สอนในชั้นเรียน)

เราจะเรียนอะไรในวันนี้

- ชนิดข้อมูลในจาวา
  - แบบพื้นฐาน (primitive data type)
  - แบบอ้างอิงวัตถุ (object reference data type)
- สัญพจน์ (Literal)
- **อาเรย์**
  - การประกาศอาเรย์ในจาวา
  - **การใช้งาน**
  - เหตุการณ์ผิดปรกติเมื่อใช้งานอาเรย์ผิดพลาด

ค่าเริ่มต้นของข้อมูลในอาเรย์

- ตัวอาเรย์ในจาวามีข้อกำหนดว่า เมื่ออาเรย์ถูกสร้างขึ้นมาแล้วค่าในอาเรย์จะถูกกำหนดเป็นค่าปริยาย (default) ให้ทันทีเสมอ
- ซึ่งถ้าเป็นอาเรย์ของตัวเลข ค่าปริยายที่ได้จะเป็นเลขศูนย์

```
int[] A = new int[10];
System.out.println(A[7]);
```

- ในกรณีข้างบนนี้ เราจะได้เลข 0 ออกมา เพราะข้อมูลในทุกช่องของ A ตอนเริ่มต้นจะมีค่าเป็นศูนย์
- แต่ถ้าเป็นอาเรย์ที่เก็บตัวแปรอ้างอิงวัตถุ (object reference) ค่าเริ่มต้นจะเป็นค่า null
  - ตอนนี้เรายังไม่รู้ว่าตัวแปรอ้างอิงวัตถุหรือ null คืออะไรก็ไม่ต้องตกใจ

เรื่องแปลก ๆ ของอาเรย์ในจาวา

- เนื่องจากอาเรย์ในจาวาเป็นวัตถุ พฤติกรรมของการจัดการตัวแปรจึงต่างจากตัวแปรทั่วไป ลองศึกษาดูจากตัวอย่างข้างล่างนี้

กรณีตัวแปรธรรมดา

```
int x = 3;
int y = 5;
x = y;
x = 7;
print(x);
print(y);
```

ผลลัพธ์ที่ได้คือ 7 และ 5

กรณีตัวแปรอาเรย์

```
int[] x = new int[10];
int[] y = new int[10];
x = y;
x[0] = 7;

System.out.println(x[0]);
System.out.println(y[0]);
```

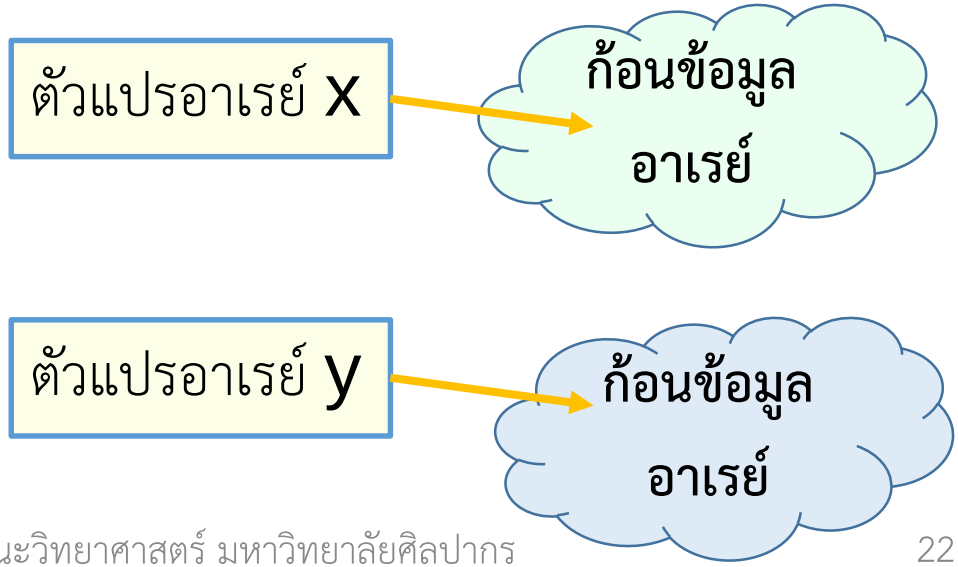
ผลลัพธ์ที่ได้คือ 7 และ 7

ทำไมจึงเป็นเช่นนั้น (1)

- การจัดการข้อมูลจำพวกวัตถุจะต่างกับข้อมูลพื้นฐาน (primitive data type) เพราะข้อมูลจำพวกวัตถุตัวแปรจะแบ่งออกเป็นสองส่วน
  - ส่วนที่หนึ่งคือ “ก้อนข้อมูล” ซึ่งเป็นการเก็บข้อมูลในหน่วยความจำที่แท้จริง
  - ส่วนที่สองคือตัวแปรอ้างอิงวัตถุ ซึ่งทำการเชื่อมโยงไปยังก้อนข้อมูลอีกต่อหนึ่ง
  - แนวคิดนี้เป็นลักษณะเดียวกับตัวชี้ (pointer) ในภาษาซี

สภาพข้อมูลตอนที่สร้างอาเรย์ขึ้นมาใหม่ ๆ

```
int[] x = new int[10];
int[] y = new int[10];
```

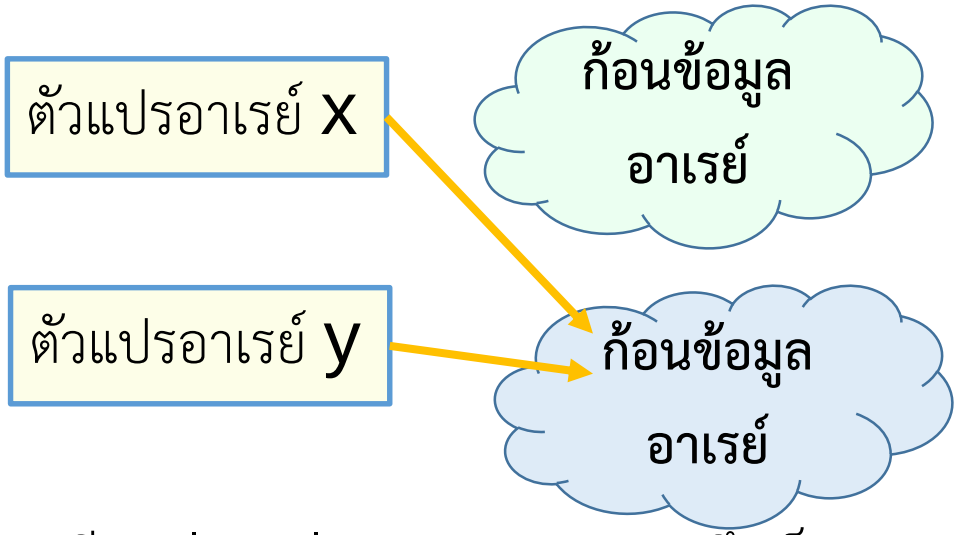


ทำไมจึงเป็นเช่นนั้น (2)

- เมื่อเรากำหนดค่าใส่ตัวแปรตรง ๆ มันคือการเปลี่ยนตัวชี้ ไม่ได้เปลี่ยนก้อนข้อมูล ดังนั้นการเขียนว่า

```
x = y;
```

จึงเป็นการเปลี่ยนให้ตัวชี้ x เชื่อมโยงไปก้อนข้อมูลเดียวกับ y



- ดังนั้นการที่เราเขียนต่อมาว่า `x[0] = 7;` จึงเป็นการแก้ไปที่ก้อนข้อมูลสีน้ำเงิน ไม่ได้แก้ไปที่ก้อนข้อมูลสีเขียว

แล้วจะเกิดอะไรขึ้นกับก้อนข้อมูลสีเขียว

- ในจาวา (และอีกหลายภาษา) เมื่อเกิดเหตุการณ์ที่ก้อนข้อมูลไม่ได้ถูกเชื่อมโยงจากตัวแปรใด เราจะถือว่าก้อนข้อมูลนั้นเป็น*ขยะ* (*garbage*)
- ซึ่งก้อนข้อมูลเขียวนี้ ไม่มีตัวแปรใดเชื่อมโยงไปถึง จึงอยู่ในภาวะถูกทิ้งไม่มีใครเป็นเจ้าของ และกลายเป็นขยะไป
- คำว่าขยะนี้สมเหตุผลดี เพราะสภาพนั้นมันเป็นก้อนข้อมูลที่ทำประโยชน์อะไรไม่ได้อีกต่อไปแล้ว
- จากตัวอย่างนั้นเราจะเห็นได้ว่า เราไม่มีทางอ้างถึงก้อนข้อมูลเขียวได้อีกต่อไป เพราะไม่มีตัวแปรไหนรู้แล้วว่าก้อนข้อมูลเขียวอยู่ที่ใด
- เมื่อไม่มีใครอ้างถึงข้อมูลได้ ก็แสดงว่าไม่มีใครใช้ประโยชน์จากก้อนข้อมูลนั้นได้ และจาวาจะทำลายก้อนข้อมูลขยะเหล่านี้ในภายหลัง

อยากรู้ว่าอาเรย์ยาวเท่าใด

- ด้วยความเป็นวัตถุ (object) เราจะพบว่าอาเรย์ในจาวามาพร้อมกับข้อมูลเกี่ยวกับความยาวของอาเรย์มาด้วย
- นั่นคือเราสามารถถามถึงความยาวของอาเรย์ได้ด้วย length

```
int[] A = new int[10];
System.out.println(A.length);
```

- ซึ่งโค้ดข้างบนนี้จะพิมพ์เลข 10 ออกมา เพราะ A.length คือเลขจำนวนเต็มที่บอกความยาวของอาเรย์ A
- เรามักจะใช้ค่าความยาวอาเรย์ที่หาได้จากวิธีนี้ไปใช้ในการวนลูปจัดการอาเรย์ เพราะเราอาจจะไม่รู้ค่าความยาวอาเรย์ล่วงหน้า
  - แต่เมื่อใช้ A.length เราจะรู้ความยาวอาเรย์ ณ ขณะนั้นเลย

เราจะเรียนอะไรในวันนี้

- ชนิดข้อมูลในจาวา
  - แบบพื้นฐาน (primitive data type)
  - แบบอ้างอิงวัตถุ (object reference data type)
- สัญพจน์ (Literal)
- **อาเรย์**
  - การประกาศอาเรย์ในจาวา
  - การใช้งาน
  - **เหตุการณ์ผิดปรกติเมื่อใช้งานอาเรย์ผิดพลาด**

ArrayIndexOutOfBoundsException

- Exception คือเหตุการณ์ผิดปรกติ
- ArrayIndexOutOfBoundsException คือเหตุการณ์ผิดปรกติซึ่งเกิดจากการอ้างถึงช่องข้อมูลที่ไม่อยู่ในอาเรย์
  - อินเด็กซ์อาจจะติดลบ หรือล้นออกไปข้างนอกอาเรย์
- เป็นหนึ่งในความผิดพลาดที่พบได้บ่อยที่สุดในการเขียนโปรแกรมที่ต้องจัดการข้อมูลจำนวนมากที่เก็บอยู่ในอาเรย์ (หรืออาเรย์ลิสต์)
- เมื่อเกิดเหตุการณ์ผิดปรกตินี้ขึ้น จาวาจะให้ข้อมูลสำหรับกับเรามาสองอย่างคือ
  - อินเด็กซ์ที่เป็นปัญหา
  - บรรทัดที่เกิดเหตุการณ์ผิดปรกตินี้
- ซึ่งเราควรใช้ข้อมูลสองอย่างนี้ในการหาที่ผิดและแก้ไขโปรแกรม

เราจะเรียนอะไรในวันนี้

- ชนิดข้อมูลในจาวา
  - แบบพื้นฐาน (primitive data type)
  - แบบอ้างอิงวัตถุ (object reference data type)
- สัญพจน์ (Literal)
- อาเรย์
  - การประกาศอาเรย์ในจาวา
  - การใช้งาน
  - เหตุการณ์ผิดปรกติเมื่อใช้งานอาเรย์ผิดพลาด
- **ของแถม: ArrayList อาเรย์ที่หดขยายได้**

ArrayList อาเรย์ที่ขยายและลดขนาดได้

- การที่เราไม่ทราบจำนวนข้อมูลที่แน่นอน (หรืออาจจะทราบ แต่คิดว่าค่อย ๆ ใส่เพิ่มเข้าไปดูจะเข้าใจง่ายกว่า)
  - การสร้างอาเรย์แบบเผื่อที่ไว้เยอะ ๆ มักจะไม่ใช่เรื่องดี
  - เพราะอาเรย์ที่ใหญ่ใช้หน่วยความจำมากและเสียเวลาสร้างนาน
  - ถ้าสร้างอาเรย์ใหญ่มา แต่ใช้จริงแค่นิดเดียวจะเสียทรัพยากรมาก
  - ถ้าสร้างอาเรย์เล็ก แต่บางทีต้องการพื้นที่เยอะ ๆ ก็จะพื้นที่ไม่พอ
- ภาษาสมัยใหม่จึงนิยมสร้าง*ชุดข้อมูล* (*Collection*) ที่เป็นอาเรย์ที่ขยายขนาดได้ (และที่จริงจะลดขนาดก็ได้)
  - ในภาษา C++ มี std::vector ส่วนภาษาจาวามี java.util.ArrayList
  - ภาษาอื่น ๆ ก็จะมีของทำนองนี้เช่นกัน แต่อาจใช้คนละชื่อ

ArrayList เป็นคลาสอีกชนิด

- ชนิดข้อมูลในจาวานั้นนอกจากพวก int, double, float, ... ซึ่งเป็นแบบ primitive ที่เหลือคือจะเป็นคลาสทั้งหมด
  - ชนิดข้อมูลแบบคลาสมีเป็นจำนวนมากใน JDK
  - เราจำเป็นที่จะต้องเข้าใจเรื่องคลาสให้ถ่องแท้จึงจะใช้ได้อย่างถูกต้อง ไม่คลาดเคลื่อน แต่ถ้าจะเริ่มใช้งานแบบพื้นฐานก็พอจะทำได้บ้าง
- สิ่งที่ต้องเข้าใจในการใช้อาเรย์ลิสต์เก็บข้อมูลที่สำคัญอันดับต้น ๆ ก็คือ
  - ArrayList ไม่เก็บข้อมูลแบบ primitive คือไม่สามารถใช้เก็บ int, float, double, boolean อะไรทำนองนี้ได้ (ฟังดูผิวเผินเหมือนไร้ประโยชน์)
  - แต่เก็บข้อมูลแบบคลาสเช่น Integer, Float, Double และ Boolean ได้ (สังเกตว่าคลาสพวกนี้ขึ้นต้นด้วยตัวใหญ่ และ Integer เป็นคำเต็ม)
  - ยังเก็บคลาสอื่น ๆ ได้สารพัด แต่ขอเอาคลาสพวกนี้มาเป็นตัวอย่าง

ลองใช้งานกันสักหน่อยเผื่อจะเข้าใจได้ง่ายขึ้น

- อันดับแรกเวลาเราสร้างอาเรย์เราต้องเขียนทำนองว่า

```
int[] A = new int[10];
```

หรือถ้าเปลี่ยนชนิดข้อมูลเป็น **double** ก็จะเขียนว่า

```
double[] B = new double[10];
```

- นั่นคือเราต้องบอกชนิดข้อมูลไปด้วย ซึ่งอาเรย์ลิสต์ก็เป็นเช่นเดียวกัน แต่จะเขียนว่า

```
ArrayList<Integer> A = new ArrayList<Integer>();
```

หรือถ้าเปลี่ยนชนิดข้อมูลเป็น **Double** ก็จะเขียนว่า

```
ArrayList<Double> A = new ArrayList<Double>();
```

- นอกจากนี้ เรายังเขียนแบบย่อได้เป็น

```
ArrayList<Integer> A = new ArrayList<>();
ArrayList<Double> A = new ArrayList<>();
```

ลองสร้างและเติมเลขเข้าไปสัก 5 ตัว

```
ArrayList<Integer> A = new ArrayList<>();
A.add(5);
A.add(2);
A.add(4);
A.add(1);
A.add(3);
```

- สังเกตว่าเราเติมข้อมูลเข้าไปด้วยเมธอด **add** และตามด้วยค่าที่ต้องการ
- และเราสามารถดึงค่าที่อยู่ภายในอาเรย์ลิสต์ออกมาด้วยเมธอด **get**

```
for(int i = 0; i < A.size(); ++i) {
    System.out.println(A.get(i));
}
```

- เราไม่ใช้เครื่องหมาย [] สำหรับอาเรย์ลิสต์ และเราสามารถหาค่าความยาวของอาเรย์ลิสต์ได้ด้วยเมธอด **size()**

แล้ว int กับ Integer มันต่างกันจริงหรือไม่

- จากหน้าที่แล้ว ดูเหมือนอาเรย์ลิสต์เก็บ `Integer` แต่เราใส่ค่า `int` เข้าไปได้แบบดื้อ ๆ เลย ตกลงมันอันเดียวกันหรือคนละอัน
- ที่จริง**มันคนละอันกัน** เพราะตัวของ `Integer` นั้นเป็นคลาสที่ภายในบรรจุค่า `int` อีกรอบหนึ่ง
  - แต่จาวาอำนวยความสะดวกให้ คือถ้าเราใส่ `int` เข้าไปตรง `add` จาวาจะสร้าง `Integer` มาห่อค่า `int` ให้เราอัตโนมัติ
  - และถ้าเราต้องการอ่านค่าของ `Integer` อย่างตอน `println` จาวาก็อำนวยความสะดวกให้เรา คือดึงเอาค่า `int` ที่อยู่ข้างในออกมาให้เรา
- การห่อค่า int แบบอัตโนมัติด้วย Integer เราเรียกว่า *auto-boxing*
- ส่วนการดึงค่า int ที่ถูกห่ออยู่ข้างใน Integer ออกมาแบบอัตโนมัติเช่นนี้ เราเรียกว่า *auto-unboxing*

เมธอดสำคัญในอาเรย์ลิสต์ (และชุดเก็บข้อมูลตัวอื่น ๆ)

- `boolean  add(E e)`
- `void  clear()`
- `boolean  contains(Object o)`
- `E  get(int index)`
- `int  indexOf(Object o)`
- `E  remove(int index)`
- `boolean  remove(Object o)`
- `int  size()`
- `Object[]  toArray()`
- `void  sort(Comparator<? super E> c)`