ตัวอย่างการอ่านข้อมูลเข้าโดยใช้คลาส Scanner

จาวาฉบับที่ ๕ (Edition 5) หรือ jdk 1.5 ได้เพิ่มคลาส Scanner เข้ามาซึ่งทำให้การอ่านข้อมูลเข้าเป็นชนิดต่าง ๆ ได้ง่ายขึ้น ก่อนที่จะมีคลาสนี้ การนำเข้าข้อมูลประเภทข้อความ (text) จำเป็นต้องอ่านข้อมูลเข้ามาเป็นตัวอักขระหรือไม่ก็เป็นสายอักขระ (string) ข้อมูลที่นำเข้านั้น ถ้าเป็นตัวเลขที่ต้องการนำไปคำนวณ ผู้เขียนโปรแกรมจำเป็นต้องเขียนข้อความสั่งเพื่อแปลงข้อมูลประเภทอักขระหรือสายอักขระที่อ่านเข้ามาให้เป็นข้อมูลประเภทตัวเลขเช่น `int` หรือ `double` เสียก่อนจึงจะนำไปคำนวณได้ คลาส **Scanner** ช่วยลดขั้นตอนให้สั้นและสะดวกขึ้นโดยการอ่านและแปลงเป็นข้อมูลชนิดที่ต้องการในขั้นตอนเดียว

Scanner จะอ่านข้อมูลประเภทข้อความเข้ามาแล้วแบ่งเป็นส่วน ๆ ด้วยตัวคั่น (limiter) ซึ่งถ้าไม่ระบุตัวคั่นเป็นอย่างอื่นจะมี white spaces เป็นตัวคั่นโดยปริยาย ข้อความแต่ละส่วนที่ถูกตัดแบ่งมาแล้วอาจนำไปแปลงให้เป็นข้อมูลชนิดต่าง ๆ ด้วยเมท็อดที่ต้องการเช่น ถ้าต้องการข้อมูลเป็นชนิด `int` สามารถใช้เมท็อด `nextInt()` ในการแปลง ถ้าต้องการแปลงให้เป็นชนิด `double` สามารถแปลงโดยใช้เมท็อด `nextDouble()` เป็นต้น

ก่อนที่จะใช้เมท็อดที่กล่าวมานี้ได้จำเป็นต้องสร้างอ็อบเจกต์ของคลาส **Scanner** ขึ้นมาก่อน เช่น

```
Scanner sn = new Scanner(System.in);
```

ตัวอย่างนี้สร้างอ็อบเจกต์ของคลาส **Scanner** โดยให้อ็อบเจกต์นี้เป็นตัวอ่านข้อมูลเข้ามาทาง `System.in` ซึ่งเป็นช่องทางนำเข้าข้อมูลมาตรฐานซึ่งโดยปรกติแล้วหมายถึงการนำเข้าข้อมูลจากแผงแป้นพิมพ์ หลังจากสร้างอ็อบเจกต์แล้วจึงเรียกใช้เมท็อดที่ต้องการ เช่นถ้าต้องการอ่านข้อมูลเข้ามาเป็นชนิด `int` ให้ใช้เมท็อด `nextInt()` ดังนี้

```
sn.nextInt();
```

ซึ่งเมท็อดนี้จะกราด(scan) ข้อมูลที่อ่านเข้ามาไปเรื่อย ๆ จนเจอตัวคั่นซึ่งในที่นี้คือที่ว่าง (white space) จึงทำการแปลงข้อมูลที่อ่านเข้ามาให้เป็นชนิด `int`

เมท็อดสำหรับการกราดข้อมูลนำเข้าแล้วแปลงเป็นข้อมูลพื้นฐานชนิดต่าง ๆ มีดังนี้ `nextByte()` `nextShort()` `nextInt()` `nextLong()` `nextFloat()` และ `nextDouble()` นอกจากนี้ยังมีเมท็อด `nextLine()` สำหรับกราดจากตำแหน่งของตัวคั่นที่พบครั้งหลังสุดไปจนจบบรรทัด จะให้ผลลัพธ์เป็นสายอักขระ (string) ที่เหลืออยู่ท้ายบรรทัดแต่ไม่รวมตัวอักขระบอกจุดจบของบรรทัด หลังจากนั้น **Scanner** จะเตรียมพร้อมสำหรับการอ่านข้อมูลจากจุดเริ่มต้นบรรทัดใหม่ต่อไป

คลาส **Scanner** บรรจุอยู่ในแพคเกจ `java.util` ดังนั้นจึงควรระบุ `import java.util.Scanner` หรือ `import.java.util.*` ไว้ในบรรทัดต้น ๆ ของโปรแกรมที่เขียนด้วย

ตัวอย่างที่ ๑ เป็นตัวอย่างแสดงการสร้างอ็อบเจกต์จากคลาส **Scanner** ขึ้นมาเพื่ออ่านข้อมูลจากแผงแป้นพิมพ์เข้าไปแล้วแปลงเป็นข้อมูลชนิดต่าง ๆ โดยผู้ใช้โปรแกรมสามารถป้อนข้อมูลหลายตัวลงในบรรทัดเดียวได้ แต่ต้องป้อนข้อมูลให้ถูกต้องตามลำดับของชนิดของข้อมูลที่ต้องการ ข้อควรระวังสำหรับการใช้โปรแกรมนี้คืออย่าป้อน

ข้อมูลผิดประเภท เช่นขณะที่ใช้เมท็อด `nextInt()` ข้อมูลที่กราดด้วยเมท็อดนี้ต้องเป็นข้อมูลตัวเลขล้วน จะมีตัวอักษรหรือเครื่องหมายพิเศษอื่นใดปะปนเข้ามาไม่ได้ ถ้ามีอักขระอื่นใดที่ไม่ใช่ตัวเลขปะปนเข้ามาด้วยก่อนที่ถึงตัวคั่น มันจะแปลงอักขระเหล่านั้นเป็นชนิด `int` ให้ไม่ได้ จะเกิดสิ่งผิดปรกติที่เรียกว่า `InputMismatchException` ขึ้น โปรแกรมจะทำงานต่อไปไม่ได้ เว้นเสียแต่ว่าจะมีการจัดการสิ่งผิดปรกตินี้อย่างเหมาะสม

ตัวอย่างที่ 1 แสดงการใช้คลาส Scanner ในการอ่านข้อมูลชนิดต่าง ๆ

```
1:   class ScanData {
2:       public static void main(String [] args) {
3:           String pname;
4:           int qty;
5:           double price;
6:           Scanner sn = new Scanner(System.in);
7:           System.out.println("Please enter quantity, price and " +
8:                              "product description then press
     Enter");
9:           qty = sn.nextInt(); // scan for integer
10:          price = sn.nextDouble(); // scan for double
11:          pname = sn.nextLine(); // scan for string until end of
     line
12:          System.out.println("Product: " + pname);
13:          System.out.println("Quantity: " + qty);
14:          System.out.println("Price: " + price);
15:          System.out.println("Amount: " + qty * price);
16:      }
17:  }
```

ตัวอย่างที่ ๑ จะอ่านข้อมูลเข้าเพียงบรรทัดเดียวแล้วจบโปรแกรม ถ้าต้องการอ่านข้อมูลต่อเนื่องหลายตัวหรือหลายบรรทัด เราอาจเขียนโปรแกรมให้ทำงานเป็นวงวน โดยอ่านข้อมูลแล้วนำไปประมวลผลซ้ำ ๆ จนกว่าเงื่อนไขที่ต้องการจะเป็นจริงหรือไม่มีข้อมูลให้อ่านอีกต่อไป ในกรณีเช่นนี้เราอาจตรวจสอบว่ายังมีข้อมูลให้อ่านต่อไปหรือไม่โดยใช้เมท็อด `hasNext()` เมท็อดนี้จะให้ผลลัพธ์เป็นค่าตรรกะ โดยให้ผลเป็น `true` ถ้ายังมีข้อมูลให้อ่านต่อไปได้ และจะให้ผลลัพธ์เป็น `false` ถ้าไม่มีข้อมูลให้อ่านอีกต่อไป

ตัวอย่างที่ 2 แสดงการใช้คลาส `Scanner` ในการอ่านข้อมูล โดยการอ่านซ้ำจนกว่าไม่มีข้อมูลให้อ่าน

```
1:   import java.util.Scanner;
2:   class ScanAndSum {
3:       public static void main(String [] args) {
4:           double sum=0;
5:           Scanner sn = new Scanner(System.in);
6:           while (sn.hasNext())
7:               sum += sn.nextDouble();
8:           System.out.println("Sum of all data: " + sum);
9:       }
10:  }
```

ตัวอย่างที่ ๒ แสดงการวนอ่านข้อมูลซ้ำจนกว่าไม่มีข้อมูลให้อ่าน โปรแกรมนี้จะกราดข้อมูลเข้ามาเป็นชนิด double แล้วนำไปบวกรวมเข้ากับตัวแปร sum จะวนทำเช่นนี้ไปเรื่อย ๆ จนกว่าจะไม่มีข้อมูลให้กราดอีกต่อไป ซึ่งในกรณีที่ป้อนข้อมูลเข้าทางแผงแป้นพิมพ์ เราสามารถป้อนข้อมูลได้บรรทัดละหลายตัว และป้อนได้หลายบรรทัด โดยที่ข้อมูลแต่ละตัวจะคั่นด้วย white space เมื่อป้อนข้อมูลจนครบแล้ว เราจะบอกว่าจบข้อมูลแล้วโดยการกด Ctrl-Z (กดปุ่ม Ctrl ค้างไว้แล้วกดปุ่ม z บนจอภาพจะแสดงผลเป็น ^Z) การทำเช่นนี้เป็นการส่งสัญญาณบอกระบบปฏิบัติการว่าผู้ใช้โปรแกรมไม่มีข้อมูลจะป้อนอีกต่อไป ซึ่งส่งผลให้การเรียกเมท็อด hasNext() ให้ผลลัพธ์เป็นเท็จ จึงทำให้หลุดออกจากวงวนไปพิมพ์ผลลัพธ์ที่ได้จากการบวกค่าข้อมูลทั้งหมดที่อ่านเข้ามา

ตัวอย่างที่ ๓ ปรับปรุงมาจากตัวอย่างที่ ๑ แสดงการกราดข้อมูลจากอินพุทซึ่งเป็นสายอักขระซึ่งเขียนตายตัวไว้ในโปรแกรม (String literal) และมีการดักจับสิ่งผิดปรกติกรณีที่ข้อมูลเข้าผิดรูปแบบ

ตัวอย่างที่ ๓ แสดงการกราดจากสายอักขระและการดักสิ่งผิดปรกติ

```
1:  // Test Exception handling when input non-numeric data with nextInt()
    method
2:  import java.util.*;
3:
4:  import java.util.Scanner;
5:  class TestScan {
6:      public static void main(String [] args) {
7:          String inputStr="10 233.50 356p tail of string";
8:          int qty;
9:          double price;
10:         String theRest;
11:         Scanner sn = new Scanner(inputStr);
12:         System.out.println("input string: " + inputStr);
13:         qty = sn.nextInt(); // scan for integer
14:         System.out.println("Quantity: " + qty);
15:         price = sn.nextDouble(); // scan for double
16:         System.out.println("Price: " + price);
17:         try {
18:             int x = sn.nextInt();  // exception occurs here
19:         }
20:         catch(InputMismatchException e) {
21:             String s = sn.next();
22:             System.out.println("The token that caused exception
    was: " + s );
23:         }
24:         theRest = sn.nextLine(); // scan for string to the end of
    line
25:         System.out.println("The rest of line: " + theRest);
26:     }
27: }
```

ตัวอย่างที่ ๔ เป็นตัวอย่างการอ่านข้อมูลจากแฟ้มข้อความ (textfile) รูปแบบ Comma-Separated Values (CSV) ซึ่งใช้จุลภาค (comma)คั่นข้อมูลแต่ละรายการ ดังตัวอย่าง

```
John,Khondee,0815557777, john@gmall.com
สมชัย,ใจดี,0811112222,somchai@jaidee.com
สมชาย,คล้ายหญิง,0982224444,somchild@coldmail.com
สมหญิง,มิ่งขวัญ,0812223333,somying@yahaa.com
สมคิด,นิดหน่อย,0863335555,
สมบัติ,พลัดกันชม,,sombat@falsemail.co.tu
สมจิตร,,0816667777,somchit@coldmail.com
```

จากตัวอย่างข้อมูลจะเห็นว่ามีการใช้จุลภาคคั่นระหว่างข้อมูลแต่ละรายการ ส่วนข้อมูลรายการสุดท้ายของบรรทัดจะไม่มีจุลภาคปิดท้าย แต่จะปิดด้วยตัวอักขระขึ้นบรรทัดใหม่ (newline) ซึ่งเป็นที่ว่างสีขาว (white space) จึงมองไม่เห็น

ตัวอย่างโปรแกรมที่ ๓ นี้ผูกโยงอ็อบเจกต์ของคลาส `Scanner` เข้ากับอ็อบเจกต์ของคลาส `File` ซึ่งใช้เป็นอินพุทไฟล์ เนื่องจากมีการใช้คลาส `File` ซึ่งอาจเกิดสิ่งผิดปรกติขึ้นถ้าหาแฟ้มที่ต้องการไม่พบ (เกิด `FileNotFoundException`) ดังนั้น จึงต้องใส่ไว้ใน try block และมีการดักจับสิ่งผิดปรกติชนิดนี้ตรงบรรทัดที่ ๒๔

หลังจากสร้างอ็อบเจกต์ของคลาส `Scanner` ได้แล้วจะเรียกเมท็อด `useDelimiter(",")` เพื่อกำหนดให้จุลภาค(,) เป็นตัวคั่นรายการข้อมูลที่ต้องการกราด หลังจากนั้นจึงเข้าส่วนของการทำงานเป็นวงวนเพื่อกราดข้อมูลแล้วนำไปแสดงผล เนื่องจากข้อมูลแต่ละรายการที่กราดเข้ามานั้นเป็นข้อความที่เป็นตัวอักขระ จึงใช้เมท็อด `next()` ในการกราดข้อมูลแต่ละรายการเข้ามาแล้วนำไปแสดงผล ยกเว้นรายการสุดท้ายของบรรทัด จะใช้เมท็อด `nextLine()` ในการกราด ทั้งนี้เพื่อให้กราดไปจนจบบรรทัดเพื่อให้ข้ามตัวบอกจุดจบบรรทัด (line delimiter) นั่นเอง ซึ่งถ้าไม่ใช้เมท็อดนี้ในการกราด มันจะกราดไปจนพบจุลภาคของบรรทัดถัดไป ซึ่งจะรวมเอาชื่อของบรรทัดถัดไปรวมอยู่ในสายอักขระที่ได้จากการกราดด้วย ซึ่งเราไม่ต้องการอย่างนั้น

การใช้เมท็อด `nextLine()` นั้น มันจะคืนค่ากลับมาเป็นสายอักขระนับตั้งแต่ตัวคั่นที่พบจากการกราดก่อนหน้านี้ ซึ่งในกรณีนี้คือเครื่องหมายจุลภาคซึ่งเราไม่ต้องการ จึงใช้เมท็อด `substring` ตัดเอามาเฉพาะส่วนท้ายของสายอักขระที่ได้จากการกราด การเรียกเมท็อด `substring()` ด้วยการส่งค่าพารามิเตอร์ `1` ไปให้ดัง `substring(1)` ตรงบรรทัดที่ ๒๐ หมายความว่าให้ตัดสายอักขระซึ่งเป็นผลลัพธ์ที่ได้จากเมท็อด `nextLine()` โดยเริ่มจากตำแหน่งที่ offset เป็น 1 คือตั้งแต่ตัวอักขระตัวที่ ๒ เป็นต้นไปจนถึงตัวอักขระตัวสุดท้ายเพื่อนำมาสร้างเป็นสายอักขระตัวใหม่ตามที่ต้องการ

ตัวอย่างที่ ๔ แสดงการกราดข้อมูลจากแฟ้มข้อมูล

```
1:   import java.util.*;
2:   import java.io.*;
3:
4:   import java.util.Scanner;
5:   class ScanCSV {
6:       public static void main(String [] args) {
7:           String inputfile ="contact.csv";
8:           String inputStr;
9:           Scanner sn;
10:          try {
11:              sn = new Scanner(new File(inputfile));
12:              sn.useDelimiter(",");
13:              while (sn.hasNext()) {
14:                  System.out.println("Name: " + sn.next());
15:                  System.out.println("Last Name: " + sn.next());
16:                  System.out.println("Phone: " + sn.next());
17:                  //the last item of the line should be read with
     nextLine()
18:                  //but the result string will contain comman (,) at
     the beginning
19:                  //so we have to cut it off by using substring(1)
20:                  System.out.println("email: " +
     sn.nextLine().substring(1));
21:                  System.out.println(); // blank line
22:              }
23:          }
24:          catch (FileNotFoundException e) {
25:              System.err.println("Input file not found: " +
     inputfile);
26:              System.exit(0);
27:          }
28:      }
29:  }
```

API `java.util.Scanner` (เริ่มมีใช้ตั้งแต่จาวารุ่น 5)

สรุปวิธีใช้ตัวสร้างบางตัว

- `Scanner(File source)`
  ตัวสร้างอ็อบเจกต์ `Scanner` รับพารามิเตอร์เป็นอ็อบเจกต์ของคลาส `File` **เพื่อการกราดข้อมูลจากแฟ้มข้อมูล**
- `Scanner(InputStream source)`
  ตัวสร้างอ็อบเจกต์ `Scanner` รับพารามิเตอร์เป็นอ็อบเจกต์ของคลาส `InputStream` เพื่อกราดข้อมูลจากสายข้อมูลเข้า

- `Scanner(String source)`
  ตัวสร้างอ็อบเจกต์ `Scanner` รับพารามิเตอร์เป็นอ็อบเจกต์ของคลาส `String` เพื่อกราดข้อมูลจากสายอักขระที่กำหนด

สรุปวิธีใช้เมท็อดบางตัว

- `boolean hasNext()`
  คืนค่าเป็น `true` ถ้ายังมีข้อมูลให้กราด
- `String next()`
  กราดหาและคืนค่าสายอักขระ
- `String next(String pattern)`
  กราดหาและคืนค่าสายอักขระถ้าเข้ากับ *`pattern`* ที่กำหนด (ดูหมายเหตุเกี่ยวกับ *`pattern`* ด้านล่าง)
- `BigDecimal nextBigDecimal()`
  กราดหาข้อมูลลำดับถัดไปแล้วแปลงให้เป็น `BigDecimal`
- `BigInteger netxtBigInteger()`
  กราดหาข้อมูลลำดับถัดไปแล้วแปลงให้เป็น `BigInteger`
- `boolean nextBoolean()`
  กราดหาข้อมูลลำดับถัดไปแล้วแปลงให้เป็น `boolean`
- `byte nextByte()`
  กราดหาข้อมูลลำดับถัดไปเป็นชนิด `byte`
- `double nextDouble()`
  กราดหาข้อมูลถัดไปแล้วแปลงให้เป็น `double`
- `float nextFloat()`
  กราดหาข้อมูลถัดไปแล้วแปลงเป็น `float`
- `long nextLong()`
  กราดหาข้อมูลถัดไปเป็นชนิด `long`
- `short nextShort()`
  กราดหาข้อมูลถัดไปเป็นชนิด `short`
- `Scanner skip(String pattern)`
  ข้ามข้อมูลตาม *`pattern`* ที่กำหนด
- `Scanner userDelimiter(String pattern)`
  กำหนดตัวคั่นตามโดยใช้ *`pattern`* ที่กำหนด

หมายเหตุเกี่ยวกับ *`pattern`*

`pattern` ในที่นี้หมายถึงสายอักขระที่ใช้กำหนดแบบอย่างของตัวคั่นหรือตัวแบ่ง ที่จะค้นหาในสายอักขระที่ต้องการกราด ตามรูปแบบของ regular expression ซึ่งมีความซับซ้อนมากพอสมควร สามารถดูรายละเอียดเพิ่มเติมได้ที่ http://download.oracle.com/javase/tutorial/essential/regex/intro.html