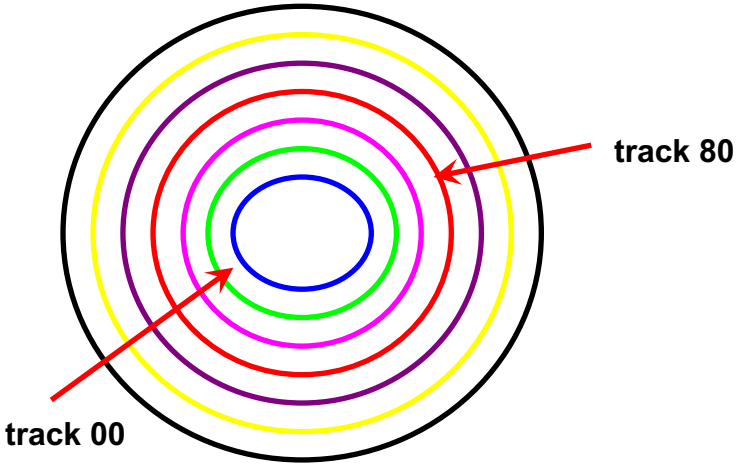
Files, Streams and I/O

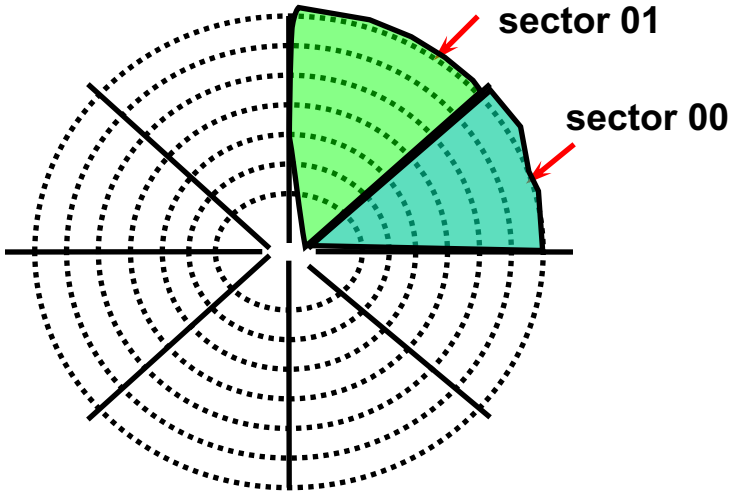
By Assoc. Prof. Rangsit Sirirangsi

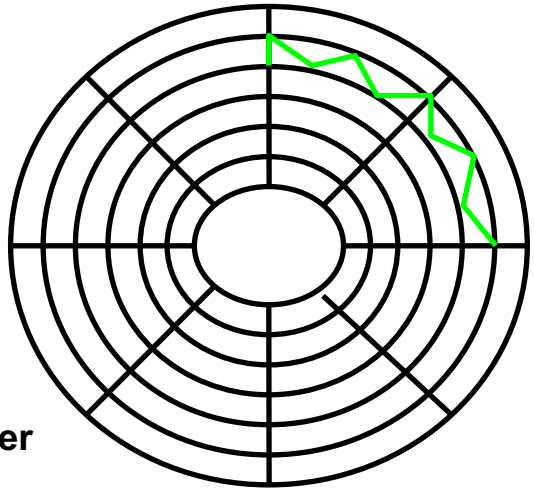
Files

- ไฟล์ใช้สำหรับการจัดกลุ่มของข้อมูลที่ถูกจัดเก็บไว้ในอุปกรณ์ภายนอก เช่น disk หรือ CD-ROM
- ในมุมมองระดับล่างสุดข้อมูลภายในไฟล์จะถูกจัดเก็บในรูปของอะเรย์ไบต์ ภายในอุปกรณ์ภายนอก เช่น บรรทัด เรคคอร์ด หรือ ออปเจค เป็นต้น
- การเข้าถึงข้อมูลภายในไฟล์สามารถทำได้ทั้งแบบ sequential หรือ direct access
- ส่วนทางด้านกายภาพไฟล์จะถูกจัดเก็บในรูปของ tracks และ sectors บน disk ที่มีการ linked เข้าด้วยกัน
- File Allocation Table (FAT) ใช้สำหรับการตรวจสอบตำแหน่งของไฟล์ที่อยู่ภายใน disk

Disk Internal

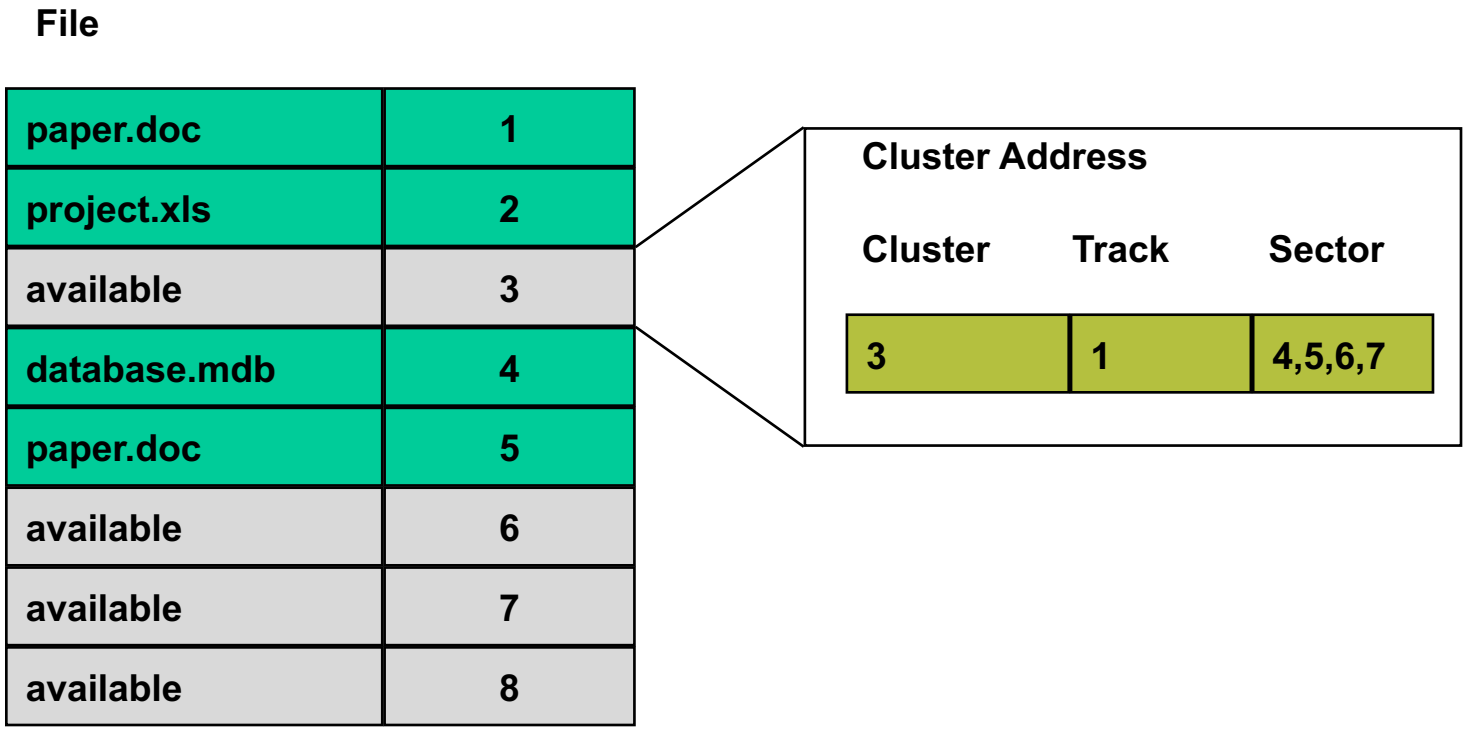






File Allocation Table

- *File Allocation Table* เก็บตำแหน่งข้อมูล โดยปกติจะเก็บไว้ที่ sector 0
- FAT จะบอกตำแหน่งของไฟล์ โดยการระบุตำแหน่งของ Cluster, Track และ Sector ตามลำดับ



Files (2)

- การอ่านข้อมูลจากไฟล์ทีละไบต์จะไม่มีประสิทธิภาพทั้งนี้เนื่องจากความเร็วในการเข้าถึงข้อมูลภายใน disk จะช้ากว่าการเข้าถึงข้อมูลภายใน RAM หรือ ROM เป็นพัน ๆ เท่า
- ดังนั้นการอ่านหรือเขียนข้อมูลลงไฟล์แต่ละครั้ง จึงกระทำผ่านส่วนที่เรียกว่า "buffered" (ขนาด 4096 bytes)
- Data buffers จะอยู่ภายในเมมโมรี่ ซึ่งเป็นส่วนหนึ่งของ File Control Block (FCB) ที่ใช้ในการแสดงรายละเอียดของไฟล์
- ในการเข้าถึงไฟล์จำเป็นต้องเปิดไฟล์ก่อนเสมอ ซึ่งจะเกี่ยวข้องกับ FCB ที่อยู่ภายในเมมโมรี่พร้อมกับไฟล์
- เมื่อเสร็จสิ้นการทำงานกับไฟล์เรียบร้อยแล้วจะต้องปิดไฟล์เสมอ

Two kinds of sequential files

- **Text files**
  - เป็นไฟล์ที่สามารถอ่านได้ ข้อมูลที่อยู่ภายในไฟล์จะถูกแปลงให้อยู่ในรูปของ ASCII codes และมีสัญลักษณ์ End of lines เพื่อใช้ระบุอักขระพิเศษ (Ctrl-lf ในระบบปฏิบัติการ Windows หรือ lf ในระบบปฏิบัติการ Unix)
- **Binary files**
  - เป็นไฟล์ที่ข้อมูลไม่สามารถแปลงให้อยู่ในรูปของ ASCII codes ได้ ตัวอย่างเช่น ไฟล์ที่ประกอบไปด้วยภาษาในระดับเครื่อง และ Java bytecode ถือเป็นไฟล์แบบ binary
  - ascii codes จะเป็น 7-bit codes ที่ใช้ในการสร้าง subset ของ Unicode

Binary & Text representation

- ตัวเลขแบบ integer ที่มีค่า 123526 จะถูกนำเสนอในรูปของ 32-bit binary number ดังนี้

```
00000000 00000001 11100010 10000110
```

(0*256*256*256 + 1*256*256 + 226*256 + 134)

- ส่วนข้อมูลแบบ string ที่มีค่า "123526" จะถูกนำเสนอโดยใช้ ASCII codes จำนวน 6 bytes

```
00110001 00110010 00110011 00110101
00110010 00110110
```

Java 1.5 (5.0)'s Scanner class

- ในจาวา 1.5 ได้มีการพัฒนาคลาส Scanner เพื่อช่วยในการรับข้อมูลจากผู้ใช้สามารถทำได้ง่ายยิ่งขึ้น นอกจากนั้นยังสามารถรับค่าได้ทั้งจากไฟล์หรือแหล่งอื่น ๆ ได้

```
Scanner  scan = new Scanner(System.in);
String temp = scan.next();
```

- สำหรับการอ่านข้อมูลจากไฟล์ สามารถทำได้โดยการสร้าง File ออปเจค และผ่านค่าในรูปของพารามิเตอร์ไปยัง Scanner ออปเจค ดังรูปแบบต่อไปนี้

```
Scanner <name> = new Scanner(new File("<file name>"));
```

```
Scanner input = new Scanner(new File("numbers.txt");
```

Using Scanners to read from files

- จากนั้นจึงอ่านค่าที่ต้องการเข้าสู่ตัวแปรที่กำหนดไว้ เช่น

```
Scanner input = new Scanner(new File(“numbers.txt”);
String first = input.next();
```

- ทุกครั้งที่มีการเรียกใช้เมธอด next() จาก Scanner ออปเจคจะเป็นการอ่านค่า string ถัดไปจากไฟล์ดังกล่าว

Catching Exceptions when Opening a File

- ในทางปฏิบัติแล้ว การเรียกใช้ File ออปเจคจำเป็นต้องป้องกันความผิดปกติที่เกิดขึ้นโดยใช้กลไกแบบ try-catch ดังตัวอย่างต่อไปนี้

```
String fname = "mydata.txt";
try {
  Scanner myInput = new Scanner( new File(fname) );
}
catch (Exception e) {
   System.out.println("Error: Cannot open file: " + fname);
}
```

Use of Scanner methods

- ในกรณีที่ต้องการอ่านค่า integer ถัดไปจากต้นทาง เรียกใช้เมธอด nextInt()

```
int  n = scan.nextInt();
```

- ในกรณีที่ต้องการอ่านค่า double ถัดไป เรียกใช้เมธอด nextDouble()

```
double  n = scan.nextDouble();
```

- การอ่านค่า String เรียกใช้เมธอด next() สิ้นสุดการรับค่าจากอักขระช่องว่าง (Space) แทป (Tab) และ newline

```
String temp = scan.next();
```

- การอ่านค่าข้อความทั้งบรรทัดจากต้นทาง เรียกใช้เมธอด nextLine()

```
String oneLine  =  scan.nextLine();
```

Scanner next…() Method

Method	*Returns*
int nextInt()	คืนค่า token ถัดไปในรูปของ int ในกรณีที่ไม่ใช่ค่าที่กำหนดไว้ จาวาเรียกใช้ InputMismatchException
long nextLong()	คืนค่า token ถัดไปในรูปของ long
float nextFloat()	คืนค่า token ถัดไปในรูปของ float
double nextDouble()	คืนค่า token ถัดไปในรูปของ long
String next()	คืนค่า token ถัดไปในรูปของ string โดยปกติจะสิ้นสุดการอ่านค่าด้วยอักขระที่เป็น whitespace เช่น blank หรือ line break หากไม่พบ token จาวาจะเรียกใช้ NoSuchElementException
String nextLine()	คืนค่าข้อความจากบรรทัดปัจจุบันที่เหลือทั้งหมดไม่รวมสัญลักษณ์ line separator ที่อยู่ท้ายสุด

Typical Use of Scanner

- ขั้นตอนการอ่านข้อมูลจากไฟล์
  - สร้าง scanner ออปเจค
  - เรียกใช้เมธอด **hasNext()** เพื่ออ่านค่าในลำดับถัดไป
  - อ่านค่าที่ได้จากเมธอด next() และดำเนินการซ้ำตามต้องการ
- เช่น การอ่านค่า String ไปจนถึง “end of file” หรือ “end of input”

```
Scanner scanner = new Scanner("Have a nice day");
 while (scanner.hasNext())    {
        System.out.println(scanner.next());
}
```

```
Have
a
nice
day
```

- **hasNext()** คืนค่า true ในกรณีที่มี token อื่นอีกภายใน input ดังนั้นจึงเหมาะสมที่จะใช้สำหรับการทดสอบก่อนการอ่านค่าเสมอ

Scanner read from file


```
import java.io.*;
import java.util.Scanner;

public class ScannerReadFile1 {

    public static void main(String[] args) {

        try {
            File file = new File("Sample.txt");
            Scanner scanner = new Scanner(file);

            while (scanner.hasNext()) {
              System.out.print(" " +scanner.next());
            }
            scanner.close();

          } catch (FileNotFoundException e) {
            e.printStackTrace();
          }
    }
}
```

This is demo from a text file.

Files and input cursor

- กำหนดให้ไฟล์ numbers.txt ประกอบไปด้วยค่าดังต่อไปนี้

```
308.2
  14.9 7.4  2.8

3.9 4.7   -15.4
   2.8
```

- Scanner จะมองเห็นข้อมูลทั้งหมดในรูปของ stream ของอักขระ โดยมีลักษณะดังนี้

```
308.2\n   14.9 7.4  2.8\n\n\n3.9 4.7 -15.4\n2.8\n
```

- ดังนั้นการเรียกใช้เมธอดต่าง ๆ จากคลาส Scanner จึงจำเป็นต้องเหมาะสมกับ token ที่มีอยู่ภายในไฟล์เป็นหลัก

Consuming tokens

- เมธอดประเภท next(), nextInt(), nextDouble() หลังจากการอ่านค่าแล้วจะเลื่อนตำแหน่งของ cursor ไปล่วงหน้าไปที่ตำแหน่งสิ้นสุดของ token ปัจจุบัน โดยข้าม whitespace ใด ๆ ที่มีอยู่
  - การทำงานในลักษณะนี้จะถูกเรียกว่า *consuming* input เช่น

    input.nextDouble()
  - **308.2**\n 14.9 7.4 2.8\n\n\n3.9 4.7 -15.4\n2.8\n
    ^

    input.nextDouble()
  - 308.2\n **14.9** 7.4 2.8\n\n\n3.9 4.7 -15.4\n2.8\n
    ^

Scanner : boolean methods

- เมธอดจากคลาส Scanner ที่ใช้ในการอ่านค่าตัวเลขจะ thorw InputMismatchException exception ในกรณีที่ค่าถัดไปไม่สามารถอ่านได้
- เพื่อป้องกันปัญหาดังกล่าวเกิดขึ้นอาจใช้เมธอดแบบ Boolean แทนได้ดังต่อไปนี้

Method	***Returns***
boolean hasNextLine()	คืนค่า true ในกรณีที่ scanner มีบรรทัดอื่นในค่าของอินพุต
boolean hasNextInt()	คืนค่า true ในกรณีที่มี token ถัดไปที่ scanner สามารถแปลงให้อยู่ในรูปของค่า int
boolean hasNextDouble()	คืนค่า true ในกรณีที่มี token ถัดไปที่ scanner สามารถแปลงให้อยู่ในรูปของค่า double

File input answer


```
import java.util.*;
import java.io.*;
public class ScannerReadDouble {

public static void main(String[] args) {
     double sum = 0.0;
     try {
         Scanner scanner = new Scanner( new
                                File("numbers.txt"));

         while (scanner.hasNextDouble())       {
            double next = scanner.nextDouble();
            System.out.println("number = " + next);
            sum += next;
         }
         scanner.close();
       } catch (FileNotFoundException e) {
            e.printStackTrace();
      }
    System.out.println("Sum = " + sum);
   }
}
```

```
308.2
  14.9 7.4  2.8

3.9 4.7    -15.4
  2.8
```

numbers.txt

```
number = 308.2
number = 14.9
number = 7.4
number = 2.8
number = 3.9
number = 4.7
number = -15.4
number = 2.8
Sum = 329.29999999999995
```

File processing question

- ในกรณีที่มีข้อมูลต่างชนิดกันภายในไฟล์ อาจใช้เงื่อนไขช่วยในการอ่านข้อมูลร่วมกับเมธอดที่เหมาะสมในการอ่านข้อได้เช่นกัน
- ไฟล์ข้อมูลอาจมีข้อมูลหลาย ๆ แบบผสมกัน แต่ผลลัพธ์ของโปรแกรมต้องการหาผลรวมเฉพาะข้อมูลที่เป็นตัวเลขทศนิยมเท่านั้น
- ตัวอย่างเช่น

```
308.2  hello
   14.9 7.4  bad stuff 2.8

3.9 4.7  oops  -15.4
:-)    2.8  @#*($&
```

File processing answer


```
import java.util.*;
import java.io.*;
public class ScannerRead {

public static void main(String[] args) {
double sum = 0.0;
try {
Scanner input = new Scanner(new File("numbers.txt"));

        while (input.hasNext()) {
            if (input.hasNextDouble()) {
                double next = input.nextDouble();
                System.out.println("number = " + next);
                sum += next;
            } else {
                input.next();   // throw away bad token
            }
        }
        input.close();
    } catch (FileNotFoundException e) {
         e.printStackTrace();
   }
           System.out.println("Sum = " + sum);
       }
}
```

```
number = 308.2
number = 14.9
number = 7.4
number = 2.8
number = 3.9
number = 4.7
number = -15.4
number = 2.8
Sum = 329.29999999999995
```

Calculate Hours worked from File (1)

- ตัวอย่าง การอ่านค่าจากไฟล์ hours.txt เพื่อคำนวณจำนวนชั่วโมงการทำงานรวมของพนักงานที่มีรายชื่ออยู่ภายในไฟล์ดังนี้

Somchai 12.5 8.1 7.6 3.4
Somsri 4.0 11.6 6.5 2.7 12
Somsak 8.0 8.0 8.0 8.0 7.5 7.0

- ผลลัพธ์จากการทำงาน :

Total hours worked by Somchai = 31.6
Total hours worked by Somsri = 36.8
Total hours worked by Somsak = 46.5

Calculate Hours worked from File (1)

```
import java.util.*;
import java.io.*;
public class HoursWorked {

public static void main(String[] args) {

 try {
        File file = new File("hours.txt");
        Scanner scanner = new Scanner(file);
          while (scanner.hasNext()) {
             String name = scanner.next();
              double sum = 0.0;
             while (scanner.hasNextDouble()) {
                sum += scanner.nextDouble();
             }
             System.out.println("Total hours worked by " + name + " = " + sum);
          }
          scanner.close();
 } catch (FileNotFoundException e) {
        e.printStackTrace();
   }
  }
}
```

อ่านชื่อ

อ่านชั่วโมงทำงาน

Calculate Hours worked from File (2)

- ตัวอย่าง การอ่านค่าจากไฟล์ hours1.txt เพื่อคำนวณจำนวนชั่วโมงการทำงานรวมของพนักงานที่มีรายชื่ออยู่ภายในไฟล์ดังนี้

123 Somchai 12.5 8.1 7.6 3.4
456 Somsri 4.0 11.6 6.5 2.7 12
789 Somsak 8.0 8.0 8.0 8.0 7.5 7.0

- ผลลัพธ์จากการทำงาน :

Total hours worked by Somchai (id#123) = 31.6
Total hours worked by Somsri (id#456) = 36.8
Total hours worked by Somsak (id#789) = 46.5

Calculate Hours worked from File (2)


```
import java.util.*;
import java.io.*;
public class HoursWorked {
 public static void main(String[] args) {
  try {
        File file = new File("hours1.txt");
        Scanner scanner = new Scanner(file);
          while (scanner.hasNext()) {
             String text = scanner.nextLine();
             Scanner data = new Scanner(text);
             int id = data.nextInt();
             String name = data.next();
             double sum = 0.0;
             while (data.hasNextDouble()) {
                sum += data.nextDouble();
             }
             System.out.println("Total hours worked by " + name +
                                " (id#" + id + ") = " + sum);
          }  scanner.close();
    } catch (FileNotFoundException e) {
          e.printStackTrace();
      }
  }
}
```

Read Mixed Data From File

Hello 1 3.6 123456789000 true
Goodbye 4 9.2 987654321000 false

```
import java.util.*;
import java.io.*;

public class ScannerReadMix {
      public static void main(String[] args) {
      try {
           File file = new File("Testmix.txt");
             Scanner scanner = new Scanner(file);
             while (scanner.hasNext())
             {
              System.out.println(scanner.next());
              System.out.println(scanner.nextInt());
              System.out.println(scanner.nextDouble());
              System.out.println(scanner.nextLong());
              System.out.println(scanner.nextBoolean());
              }
              scanner.close();
         } catch (FileNotFoundException e) {
             e.printStackTrace();
         }
       }
}
```

```
Hello
1
3.6
123456789000
true
Goodbye
4
9.2
987654321000
false
```

Changing the token delimiter

- Scanner ออปเจคสามารถอ่านค่าจากต้นทางได้โดยอาศัยการกำหนดเครื่องหมายแบ่งวรรคตอน โดยการเรียกใช้เมธอด useDelimiter() ได้ดังต่อไปนี้
  - . (period) ตรงกับอักขระหนึ่งๆ
  - .* ตรงกับอักขระไม่จำกัดจำนวน รวมไปถึงไม่มีอักขระ
  - \\s ตรงกับหนึ่งอักขระที่เป็น whitespace
  - \\s+ ตรงกับหนึ่งอักขระที่เป็น whitespace หรือมากกว่า
  - [abc] ตรงกับหนึ่งอักขระใด ๆ ที่อยู่ภายในเครื่องหมายก้ามปู

Command	String	returns
sc.useDelimiter("::");	"one::two::three"	"one" "two" "three"
sc.useDelimiter("\\s+,\\s+")	"one , two , three"	"one" "two" "three"

useDelimiter()

```
import java.util.Scanner;

public class DelimiterDemo {

    public static void main(String[] args) {
    Scanner s = new Scanner("hello, world \n hello world");

    s.useDelimiter(",|\\n");          // , or \n

      while(s.hasNext()) {
          System.out.println(s.next());
      }
   }
}
```

```
hello
 world
 hello world
```


```
import java.util.Scanner;
import java.io.*;

public class ScannerDelimiter {
 public static void main(String[] args) {
     try {
       Scanner scanner = new Scanner(new File("Test2.txt"));
       while (scanner.hasNext()) {
              parseLine(scanner.nextLine());
       }
       scanner.close();
     } catch (FileNotFoundException e) {
       e.printStackTrace();
     }
   }

   public static void parseLine(String line) {
     Scanner lineScanner = new Scanner(line);
     lineScanner.useDelimiter("\\s*, \\s*");
     String name = lineScanner.next();
     int age = lineScanner.nextInt();
     boolean isCertified = lineScanner.nextBoolean();
     System.out.println("It is " + isCertified +" that " + name + ", age "
       + age + ", is certified.");
   }
 }
```

```
Joe, 38, true
 Kay,    27,    true
 Lou, 33,      false
```

```
It is true that Joe, age 38, is certified.
It is true that    Kay, age 27, is certified.
It is false that    Lou, age 33, is certified.
```

Read File with token delimiter

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

Read File with token delimiter


```
import java.io.*;
import java.util.Scanner;
class ScanCSV {
 public static void main(String [] args) {
  try {
     Scanner scanner = new Scanner(new File("contact.txt"));
     scanner.useDelimiter(",");
     while (scanner.hasNext()) {
             System.out.println("Name: " + scanner.next());
             System.out.println("Last Name: " + scanner.next());
             System.out.println("Tel: " + scanner.next());
             System.out.println("email: " + scanner.nextLine().substring(1));
             System.out.println(); // blank line
      }
     }
     catch (FileNotFoundException e) {
             System.err.println("Input file not found: ");
      }
     }
}
```

Writing *Character* Data to a File

- คลาส ***FileWriter***
  - สืบทอดมาจาก *OutputStreamWriter* ซึ่งสืบทอดมาจากคลาส *Writer*
  - คอนสรัคเตอร์
    - public **FileWriter** (String fileName) throws IOException
    - public **FileWriter** (String fileName, boolean **append**) throws IOException
  - เมธอดที่น่าสนใจ
    - void **write**(char c) throws IOException
    - void **write**(**char[]** buf, int start, int len) throws IOException
    - void **write**(**String** str, int start, int len) throws IOException
    - void **flush**() throws IOException
    - void **close**() throws IOException

Writing a sequential file

- ในทำนองเดียวกันชุดคำสั่งในการเขียนข้อมูลลงไฟล์ทั้ง 2 ชุดคำสั่ง สามารถนำมารวมกันได้ เช่น

```
FileWriter fw = new FileWriter("c:\\output.txt");
fw.write("Hello Test");
```

Write to File

```
import java.io.*;
import java.util.Scanner;
public class FileWriter1 {
        public static void main(String args[])  {
                try{
                    FileWriter  out = new FileWriter("out.txt");
                        out.write(" hello, my little file ");
                        out.write(" I will print a line in you!");
                        out.close();
                }  catch(Exception e){
                        System.err.println("Error: "+e.getMessage());
                  }
          }
}
```

File Write to & Read From

แทนการใช้ try/catch

```
import java.io.*;
import java.util.Scanner;

public class FileWriter1 {
     public static void main(String args[]) throws IOException {

        FileWriter fout = new FileWriter("testw.txt");
        fout.write("2, 3.4,    5,6, 7.4, 9.1, 10.5, done");
        fout.close();

        Scanner src = new Scanner(new File(“testw.txt“));

        while (src.hasNext()) {
            System.out.print( " "+src.next());
          }
        fin.close();
       }
}
```