IS311 Programming Concepts

JAVA

Input and Output

วัตถุประสงค์

- เพื่อให้เข้าใจแนวคิด stream ที่นำมาใช้กับการ input/output
- เพื่อให้สามารถอ่านและเขียนแฟ้มข้อมูล
- เพื่อให้รู้ข้อแตกต่างระหว่าง text file และ binary file
- ให้รู้จักใช้แฟ้มทั้งแบบ sequential และ random
- ให้รู้จัก Serialization

Introduction

- Storage of data in variables and arrays is temporary
- Files used for long-term retention of large amounts of data, even after the programs that created the data terminate
- Persistent data – exists beyond the duration of program execution
- Files stored on secondary storage devices
- Stream – ordered data that is read from or written to a file

Data Hierarchy

- **bit -** smallest data item on a computer, can have values 0 or 1
- **byte** - 8 bits
- **Character** - Consists of decimal digits, letters and special symbols
- **Fields** – a group of characters or bytes that conveys meaning
- **Record** – a group of related fields
- **File** – a group of related records
- **Database** – a group of related files
- Database Management System (**DBMS**)– a collection of programs designed to create and manage databases (จาวามีวิธีติดต่อหรือใช้ฐานข้อมูลได้หลายวิธี เช่น JDBC แต่ไม่เรียนในวิชานี้)

หน่วยที่เล็กที่สุดในการอ่าน/เขียนข้อมูลคือ byte

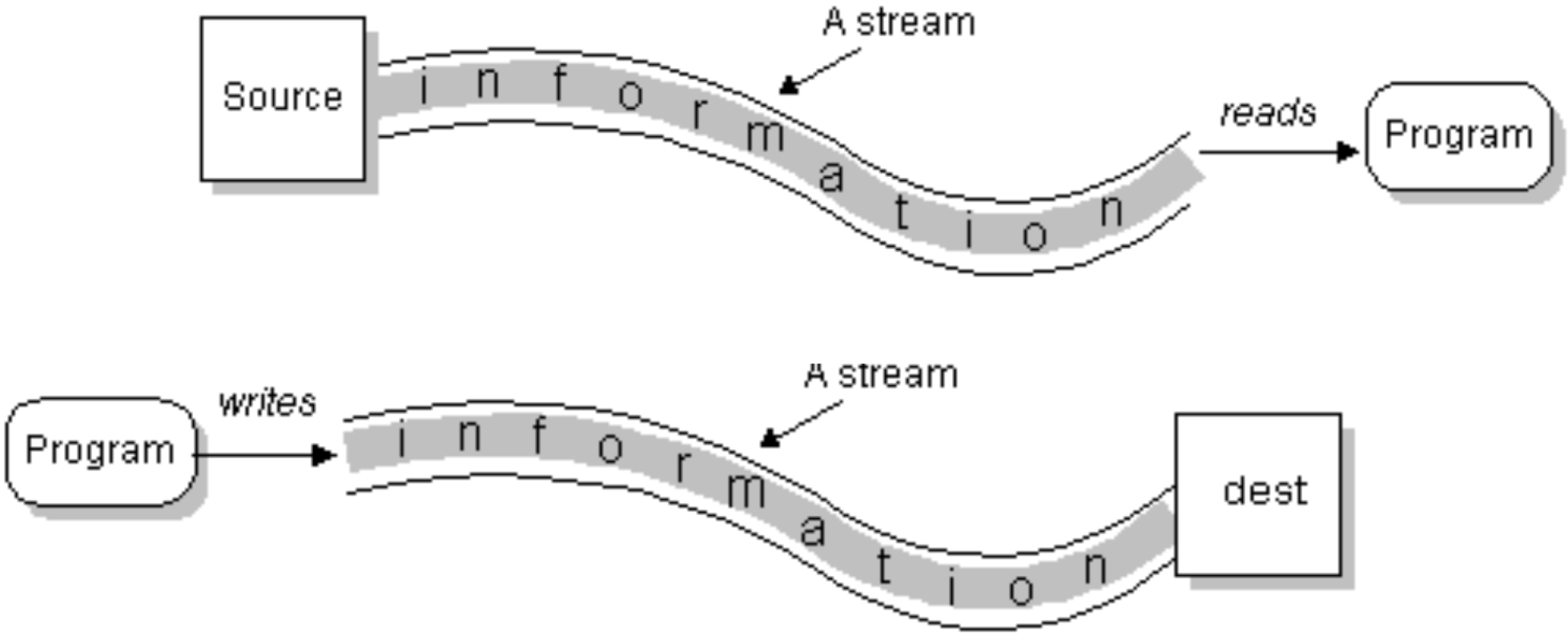
Java Input/Output

- Java I/O is accomplished using objects that represent streams of data
- A *stream* is a sequence of bytes that flow from a source to a destination
- In a program, we read information from an input stream and write information to an output stream
- A program can manage multiple streams simultaneously

Streams

Java I/O is defined in terms of **streams**.

Streams are ordered sequences of data that have a source or destination.



Java Streams

Java Streams can be used to represent any source or destination of data in the form of a sequence of bytes, including network connections, files, memory blocks and other forms of I/O. If program is writing to a stream, it is the stream's *source*. If it is reading from a stream, it is the stream's *destination*.

Java Streams (cont.)

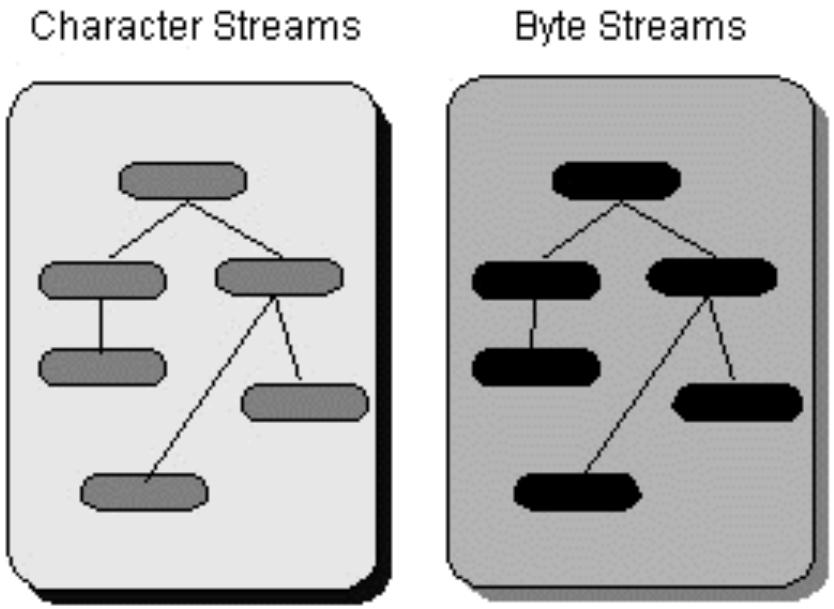
A key characteristic of Java streams is that their read and write operations are *synchronous* and *blocking*. This mean that connection to the source (when reading) or destination (when writing) must be established first and that connection must be remains open until the read or write operation is complete. Also, when reading data, if the data is not available immediately, the thread in which the read operation takes place is blocked until the data becomes available.

I/O Package

The **java.io** package contains a collection of stream classes that support these algorithms for reading and writing.

These classes are divided into two class hierarchies based on the data type (either characters or bytes) on which they operate.

Readers & Writers



Input & Output

Java IO Classes

- Java I/O is based on a class hierarchy.
- Base classes are used to describe the basic functionality required.
- Derived classes provided the functionality for specific kinds of I/O environments.
  - Files, Networks, Memory block, etc.

คลาสหลัก

- **File**: An object of this class is either a file or a directory. อ๊อบเจ็กท์ที่เก็บข้อมูลของไฟล์หรือไดเร็กทอรี่ ไม่ได้ทำหน้าที่ input/output
- **OutputStream**: base class for byte output streams
- **InputStream**: base class for byte input streams

คลาสหลัก (ต่อ)

- **Writer**: base class for character output streams.
- **Reader**: base class for character input streams.
- **RandomAccessFile**: provides support for random access to a file.
- Note that the classes **InputStream**, **OutputStream**, **Reader**, and **Writer** are *abstract* classes.

การเก็บข้อมูลในแฟ้ม

- **Text format**

  เก็บข้อความ โดยใช้รหัสเลขฐานสองแทนตัวอักขระ

  เช่นใช้มาตรฐาน ASCII ในการเก็บข้อความ

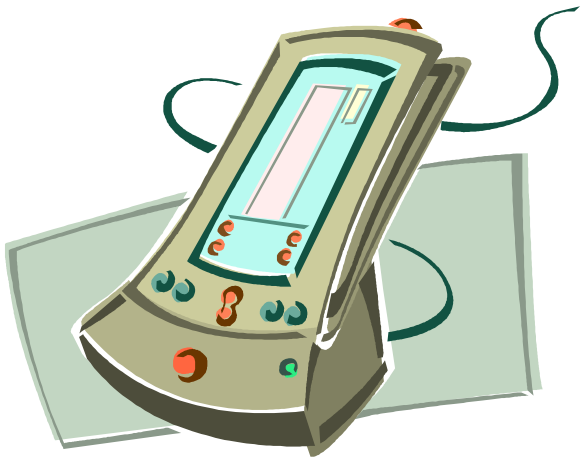
- **Binary format**

  เก็บข้อมูลชนิดต่าง ๆ หลากหลายชนิด

  เช่น รูปภาพ เสียง โปรแกรมคอมพิวเตอร์ที่

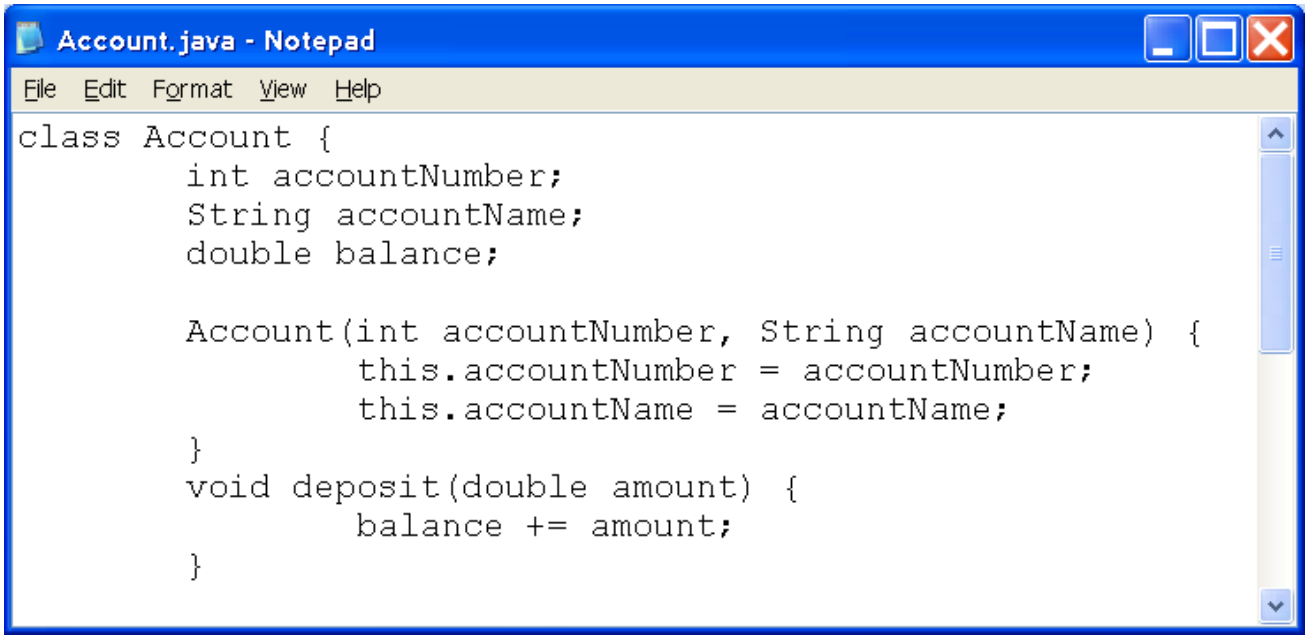
  คอมไพล์แล้ว

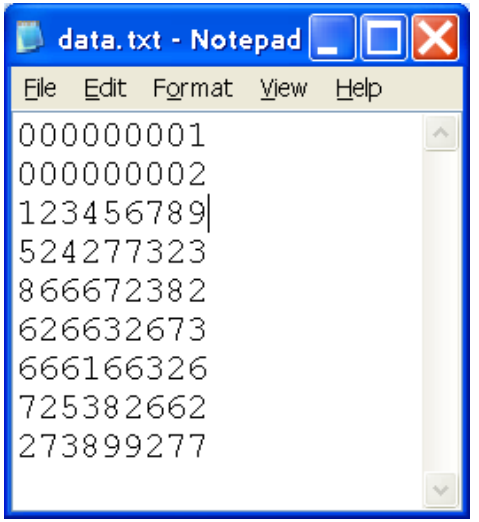
  เป็นต้น



Text Format

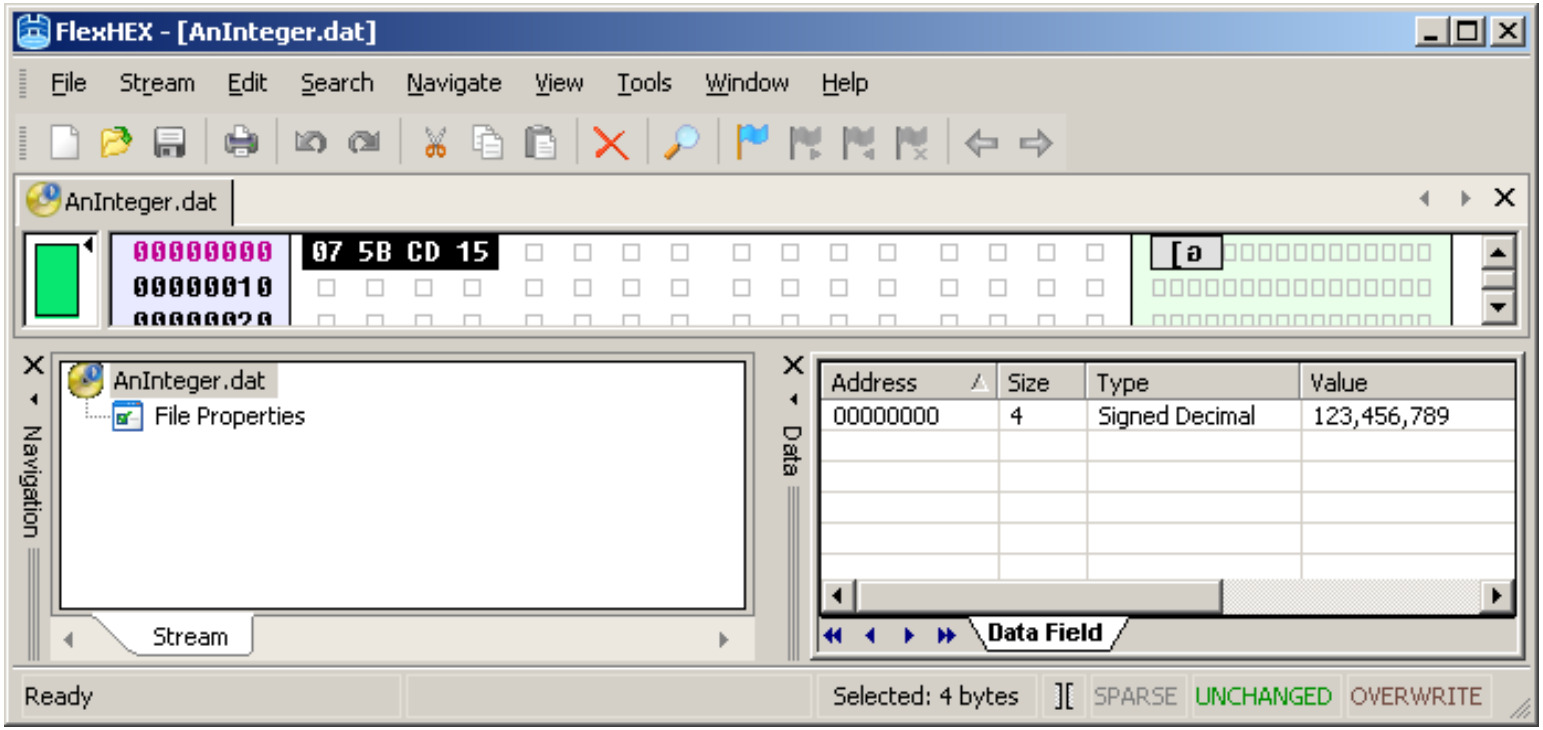
- ใช้เนื้อที่ในการจัดเก็บมากแต่สามารถอ่านได้ง่าย (Human-readable form โดยใช้โปรแกรมอ่านหรือแก้ไขข้อความ เช่น Notepad)
- เก็บตัวอักขระเรียงต่อเนื่องกัน
- กรณีที่เก็บข้อมูลตัวเลข จะเก็บเป็นตัวอักขระ เช่นเลขจำนวนเต็ม 12345 จะเก็บเป็นอักขระ ‘1’ ‘2’ ‘3’ ‘4’ ‘5’ ถ้าจะนำไปคำนวณต้องแปลงให้เป็นข้อมูลชนิดตัวเลขก่อน





Binary Format

- ใช้พื้นที่ในการจัดเก็บน้อยกว่า Text Format
- นำไปคำนวณได้ทันทีไม่ต้อง มีประสิทธิภาพในการประมวลผล
- ตัวอย่าง เช่น เลขจำนวนเต็ม123456789 ตรงกับเลขฐาน 16 คือ 07 5B CD 15 ใช้เนื้อที่ในการเก็บ 4 ไบต์ ถ้าเก็บด้วย text format แบบ ASCII ใช้เนื้อที่ 9 ไบต์ ถ้าเก็บแบบ Unicode ใช้เนื้อที่ 18 ไบต์
- มักใช้โปรแกรมเฉพาะด้านในการเขียนและอ่านข้อมูล หรือใช้โปรแกรมสำหรับอ่านในรูปแบบเลขฐาน 16 ดังเช่นตัวอย่าง



Simple Java I/O

General Principles

Why Java I/O is hard

- Java I/O is very powerful, with an overwhelming number of options
- Any given kind of I/O is not particularly difficult
- The trick is to find your way through the maze of possibilities

Streams

- All modern I/O is stream-based
- A stream is a connection to a source of data or to a destination for data (sometimes both)
- An input stream may be associated with the keyboard
- An input stream or an output stream may be associated with a file
- Different streams have different characteristics:
  - A file has a definite length, and therefore an end
  - Keyboard input has no specific end

How to do I/O

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

- *Open* the stream
- *Use* the stream (read, write, or both)
- *Close* the stream

ดู package java.io ใน Java API
http://docs.oracle.com/javase/7/docs/api/

Streams

The algorithm for reading and writing is pretty much the same, no matter what the data is.

Reading

```
open a stream
while more information
    read information
close the stream
```

Writing

```
open a stream
while more information
    write information
close the stream
```

open a stream ในจาวาคือการสร้างอ๊อบเจ็กต์ผูกโยงกับแหล่งข้อมูล

การใช้สตรีม

ขั้นตอนสำหรับ input stream

- สร้างอ๊อบเจ็กต์ผูกโยงกับแหล่งกำเนิดข้อมูล (data source)
- อ่านข้อมูลจาก input stream โดยใช้เมท็อดใดเมท็อดหนึ่งของอ๊อบเจกต์
- เมื่ออ่านข้อมูลเสร็จสิ้นแล้วให้เรียกเมท็อด `close()`

ขั้นตอนสำหรับ output stream

- สร้างอ๊อบเจ็กต์ผูกพันกับแหล่งรับข้อมูลปลายทาง (data destination)
- เขียนข้อมูลไปยัง output stream โดยใช้เมท็อดใดเมท็อดหนึ่งของอ๊อบเจกต์
- เมื่อเขียนข้อมูลเสร็จสิ้นแล้วให้เรียกเมท็อด `close()`

Opening a stream

- There is data external to your program that you want to get, or you want to put data somewhere outside your program
- When you open a stream, you are making a connection to that external place
- Once the connection is made, you forget about the external place and just use the stream

Example of opening a stream

- A FileReader is a used to connect to a file that will be used for input:

```
FileReader fileReader =
        new FileReader(fileName);
```

- The fileName specifies where the (external) file is to be found
- You never use fileName again; instead, you use fileReader

Using a stream

- Some streams can be used only for input, others only for output, still others for both
- *Using* a stream means doing input from it or output to it
- But it’s not usually that simple--you need to manipulate the data in some way as it comes in or goes out

Example of using a stream

```
int charAsInt;
charAsInt = fileReader.read( );
```

- The fileReader.read() method reads one character and returns it as an integer, or -1 if there are no more characters to read
- The meaning of the integer depends on the file encoding (ASCII, Unicode, other)
- You can *cast* from int to char:

```
char ch = (char)fileReader.read( );
```

Manipulating the input data

- Reading characters as integers isn't usually what you want to do
- A BufferedReader will convert integers to characters; it can also read whole lines
- The constructor for BufferedReader takes a FileReader parameter:

```
BufferedReader bufferedReader =
    new BufferedReader(fileReader);
```

Reading lines

```
String s;
s = bufferedReader.readLine( );
```

- A BufferedReader will return null if there is nothing more to read

Closing

- A stream is an expensive resource
- There is a limit on the number of streams that you can have open at one time
- You should not have more than one stream open on the same file
- You must close a stream before you can open it again
- *Always close your streams!*

- Java will normally close your streams for you when your program ends, but it isn’t good style to depend on this

Text input/output

Scanner

เป็นคลาสใหม่อยู่ในแพกเกจ `java.util` ไม่ใช่ `java.io`

ตัวอย่าง Constructors เช่น

- **`Scanner`**`(File source)`
- **`Scanner`**`(InputStream source)`
- **`Scanner`**`(Readable source)`
- **`Scanner`**`(String source)`

อ่านชีทเรื่อง Scanner ในเว็บไซต์

Scanner (cont.)

Methods

- **Scanner useDelimiter**(String pattern)
- boolean **hasNext**()
- boolean **hasNextByte**()
- boolean **hasNextShort**()
- boolean **hasNextInt**()
- boolean **hasNextLong**()
- boolean **hasNextFloat**()
- boolean **hasNextDouble**()
- boolean **hasNextBigInteger**()
- boolean **hasNextBigDecimal**()
- boolean **hasNextLine**()

Scanner (cont.)

Methods

- String **next**()
- byte **nextByte**()
- short **nextShort**()
- int **nextInt**()
- long **nextLong**()
- float **nextFloat**()
- double **nextDouble**()
- BigInteger **nextBigInteger**()
- BingDecimal **nextBigDecimal**()
- String **nextLine**()
- void **close**()

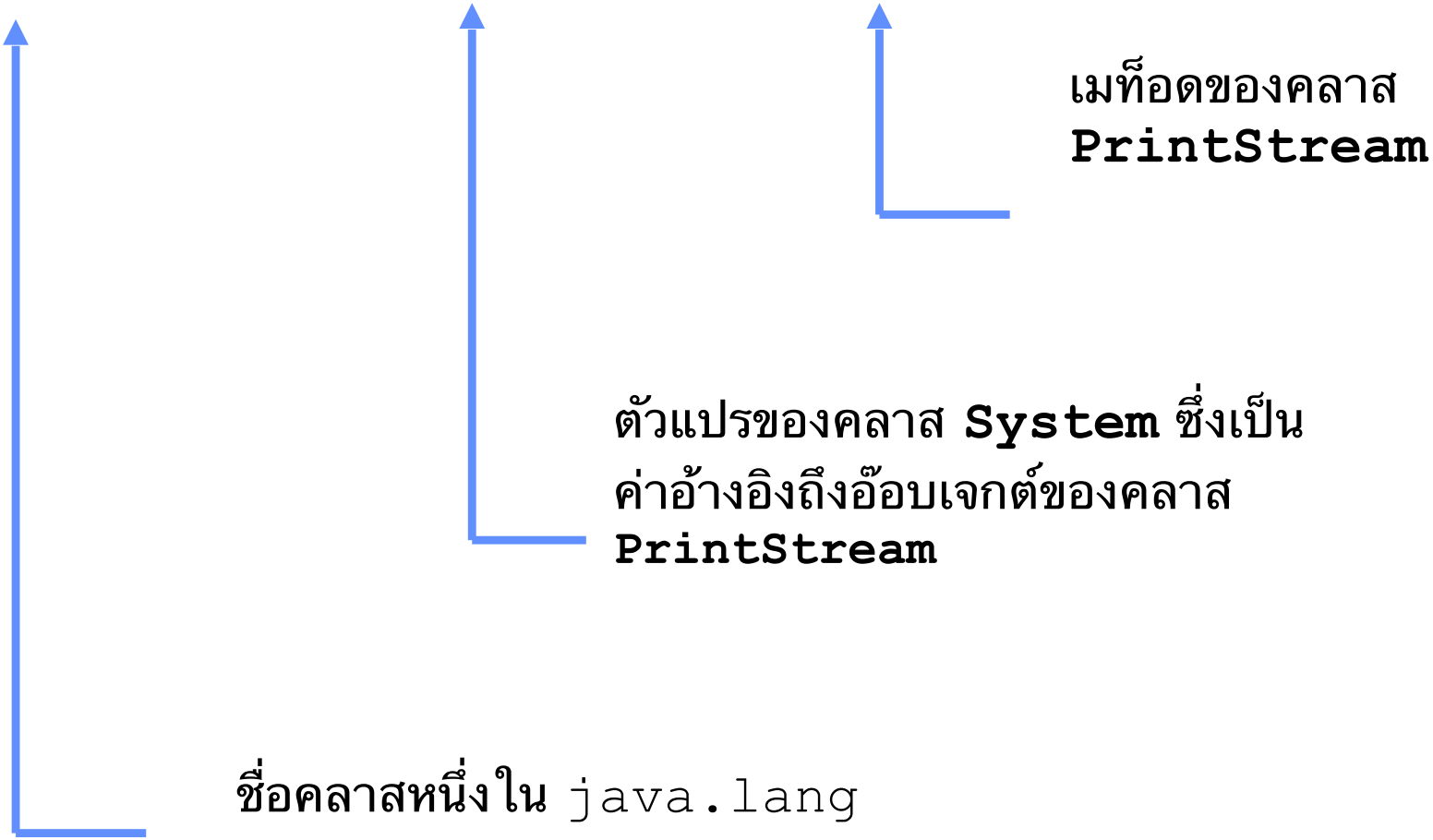
PrintStream

Print formatted representations of objects to a text-output stream

- `void print(boolean b)`
- `void print(char c)`
- `void print(char []c)`
- `void print(double d)`
- `void print(float f)`
- `void print(int i)`
- `void print(long l)`
- `void print(Object o)`
- `void print(String s)`
- `PrintStream printf(String format, Object... args)`
- `void flush()`
- ข้อมูลที่พิมพ์จะค้างอยู่ใน buffer และจะ flush โดยอัตโนมัติเมื่อพบ newline
- เมธอด `println` ใช้เหมือน `print` แต่จะขึ้นบรรทัดใหม่หลังจากพิมพ์

What is `System.out.print`?

`System.out.println(...)`



printf(...)

```
public PrintStream printf(
  String format, Object... args)
```

format อยู่ในรูป

%[argument_index$][flags][width][.precision]*conversion*

[...] เป็น option

conversion เป็นตัวอักษรบอกให้แปลงข้อมูลที่ส่งเข้ามาเป็น argument จากชนิดใดให้เป็นสตริง เช่น

d (decimal) หมายถึง argument เป็นเลขจำนวนเต็ม

f (float) หมายถึง argument เป็นเลขทศนิยมลอยตัว

s (string) หมายถึง argument เป็น string

ดูตัวอย่างเพิ่มเติมที่ https://dzone.com/articles/java-string-format-examples

ตัวอย่างการใช้ printf()

```
class PrinfDemo {
  public static void main(String [] args) {
      int quantity = 5;
      double price = 12345.6789;
      String currency = "Baht";
      System.out.println(
           "Quantity " + quantity +
           " Price "+ price + " " + currency);
      System.out.printf(
           "Quantity %d Price %10.2f %s",
      quantity, price, currency);
  }
}
```

```
Quantity 5 Price 12345.6789 Baht
Quantity 5 Price   12345.68 Baht
```

Standard I/O Data Streams

- There are three predefined standard I/O streams:
  - *standard input* – defined by `System.in`
  - *standard output* – defined by `System.out`
  - *standard error* – defined by `System.err`
- `System.in` typically represents keyboard input
- `System.out` a buffered **`PrintStream`** object, usually the screen or an active window
- `System.err` an unbuffered **`PrintStream`** object usually associated with the screen or console window

The class **System** variables

```
public static final InputStream in
```

System.in standard input stream

```
public static final PrintStream out
```

System.out standard output stream

```
static final PrintStream err
```

System.err standard error stream

อ่านชีทเรื่อง “การรับเข้าและส่งออกข้อมูล” ในเว็บไซต์

PrintWriter

คลาส PrintWriter เป็นคลาสใหม่ที่สร้างขึ้นมาสำหรับจัดการการพิมพ์อักขระซึ่งเก็บโดยใช้รหัส Unicode 16bit แทนการใช้คลาส PrintStream (คลาส PrintStream จัดการอักขระตัวละ 1 byte) จึงควรใช้คลาส PrintWriter จะปลอดภัยกว่า ซันแนะนำให้ใช้ PrintWriter แทน PrintStream

คลาสนี้มีเมท็อดให้ใช้แบบเดียวกับคลาส PrintStream

InputStreamReader

Constructor

- **InputStreamReader**(**InputStream** in)

Methods

- public boolean **ready**()
- public int **read**()
- public int **read**(char[]cbuf, int off,
  int len)
- public void **close**()

BufferedReader

Constructors

- **BufferedReader**(Reader **in**)
- **BufferedReader**(Reader **in, int size**)

Methods

- public int **read**()
- public int **read**(char[]cbuf, int off,
  int len)
- public String **readLine**()

BufferedReader (ต่อ)

Methods

- `public long skip(long n)`
- `public boolean ready()`
- `public boolean markSupported()`
- `public void mark(int readAheadLimit)`
- `public void reset()`
- `public void close()`

Standard Input

```
BufferedReader in =
    new BufferedReader(new
  InputStreamReader(System.in));
```

- `InputStreamReader` converts the byte input stream to a character input stream
- `BufferedReader` allows us to use the `readLine` method to get an entire line of input
-

Reading Strings

```
import java.io.*;
  …
public … main( String []args ) throws
  IOException
{
  String s;
  InputStreamReader ir =
                  new InputStreamReader(System.in);
  BufferedReader in = new BufferedReader( ir );

  while( (s = in.readLine()) != null ) {
      System.out.println( s );
  }
  …
}
```

คลาส `File`

ตัวช่วยในการจัดการแฟ้มข้อมูล เช่น สอบถามข้อมูลเกี่ยวกับแฟ้ม การเปลี่ยนชื่อแฟ้ม การลบแฟ้ม

The class `File`

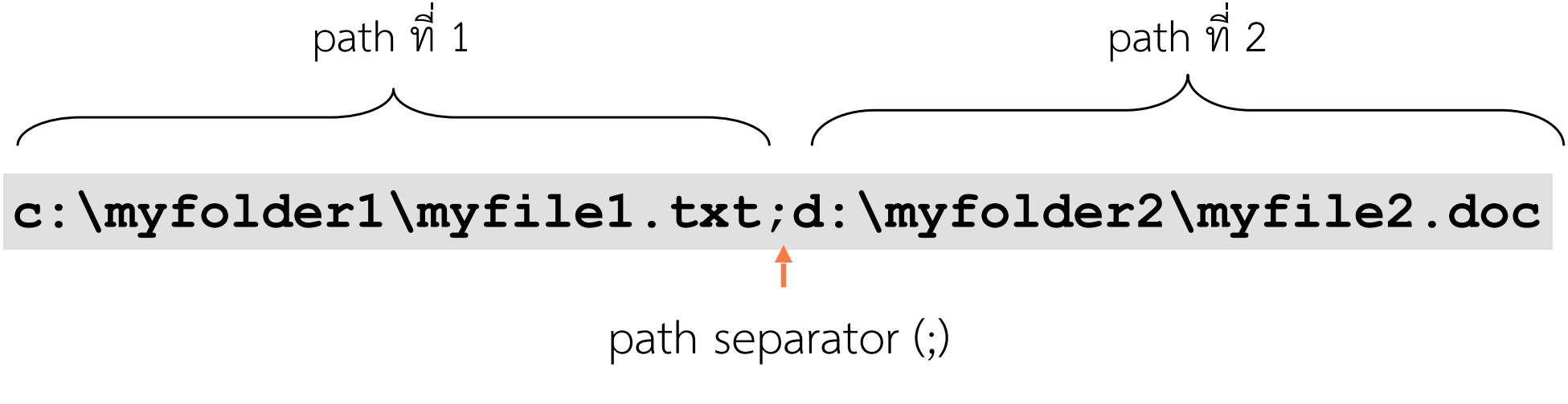
`File` เป็นคลาสที่ใช้ตรวจสอบข้อมูลบางอย่างเกี่ยวกับแฟ้มข้อมูลหรือ directory และยังมีเมท็อดสำหรับกระทำการบางอย่างเกี่ยวกับแฟ้มข้อมูลหรือ directory โดยที่ไม่มีการ input หรือ output แต่อย่างใด

เรื่องของ path

path เป็นเส้นทางนำไปสู่แฟ้ม มักเริ่มต้นจาก drive แล้วผ่านโฟลเดอร์ต่าง ๆ ไปจนถึงชื่อแฟ้มข้อมูล

บางครั้งมีการระบุ path หลาย path ไว้ในคราวเดียวกันเพื่อบอกทางเลือกในการค้นหาแฟ้ม ถ้าไม่พบใน path แรก ให้ค้นหาใน path ถัดไป

อักขระที่ใช้ในการคั่นระหว่างชื่อโฟลเดอร์กับชื่อไฟล์ และอักขระที่ใช้คั่นชือ path อาจใช้แตกต่างกันในแต่ละระบบปฏิบัติการ



ตามตัวอย่างนี้ separator สำหรับ DOS และ Windows คือ back slash (\)

ชื่อ path ใน OS ต่าง ๆ

path ใช้ในการอ้างถึงไฟล์หรือโฟลเดอร์ ซึ่งแต่ละระบบปฏิบัติการจะมีวิธีการอ้างที่แตกต่างกัน เช่น

- **Unix**
  `"/home/users/wanchai/file.ext"`
- **DOS**
  `"C:\home\users\wanchai\file.ext"`

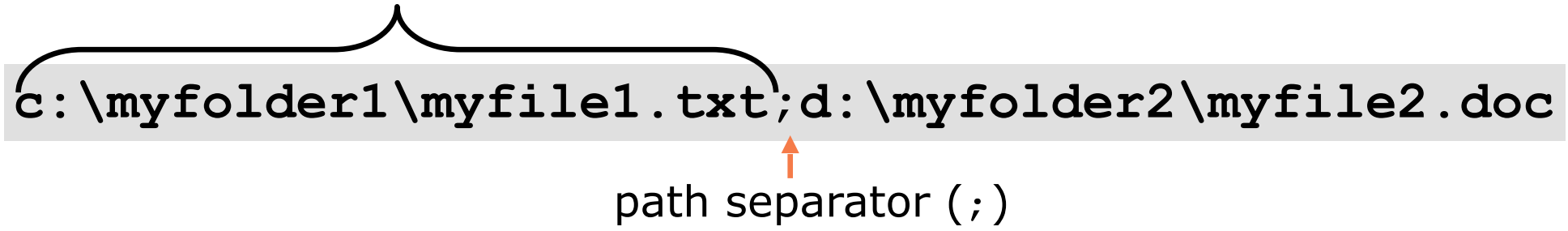
คลาส File

Fields

- `static char` **pathSeparatorChar**
- `static String` **pathSeparator**
- `static char` **separatorChar**
- `static String` **separator**

Constructors

- **File** `(String pathname) // constructor`



ตามตัวอย่างนี้ separator สำหรับ DOS และ Windows คือตัว ****

File Methods

Methods

- `public boolean` **`exists`**`()`
- `public boolean` **`isDirectory`**`()`
- `public boolean` **`isFile`**`()`
- `public boolean` **`canRead`**`()`
- `public boolean` **`canWrite`**`()`
- `public long` **`lastModified`**`()`
- `public long` **`length`**`()`
- `public boolean` **`delete`**`()`
- `public void` **`deleteOnExit`**`()`

`File` Methods (ต่อ)

- `public boolean` **`mkdir`**`()`
- `public boolean` **`renameTo`**`(`**`File`**` dest)`
- `public boolean` **`setReadOnly`**`()`
- `String []` **`list`**`()`

  if the **`File`** object is a directory, returns a String array of all the files and directories contained in the directory; otherwise, `null`

System Properties

- to obtain the path to the current working directory use

```
System.getProperty("user.dir");
```

- to obtain the file or path separator use

```
System.getProperty("file.separator");
System.getProperty("path.separator");
```