



LECTURE 12

File Management and Regular Expression

ITCS123 Object Oriented Programming

Dr. Siripen Pongpaichet
Dr. Petch Sajjacholapunt
Asst. Prof. Dr. Ananta Srisuphab

(Some materials in the lecture are done by Aj. Suppawong Tuarob)

Ref: Java Concepts Early Objects by Cay Horstmann

Review Exception Handling: Try, Catch, Finally

Syntax

```
try
{
    statement
    statement
    . . .
}
catch (ExceptionClass exceptionObject)
{
    statement
    statement
    . . .
}
finally
{
    statement
    statement
    . . .
}
```

```
Scanner scan = new Scanner(System.in);
try{
    System.out.println("Enter any number: ");
    int numSure = scan.nextInt();
    System.out.println("Your number is " + numSure);
    System.out.println("Enter any number again: ");
    String stringSure = scan.next();
    double num2 = Double.parseDouble(stringSure);
    System.out.println("Your number is " + num2);
    System.out.println("Good job. Bye!");

} catch (InputMismatchException e){
    System.out.println("Your input is not a number");
} catch (NumberFormatException e){
    System.out.println("Cannot convert your input to number");
} finally {
    System.out.println("Here in finally block");
    scan.close();
}
```

Review Exception Handling: throw, throws

The throws Clause

Syntax `modifiers returnType methodName(parameterType parameterName, . . .)
throws ExceptionClass, ExceptionClass, . . .`

```
public void readData(String filename)  
    throws FileNotFoundException, NumberFormatException
```

You **must** specify all checked exceptions
that this method may throw.

You may also list unchecked exceptions.

Principle:

Throw an exception as soon as
a problem is detected.

Catch it only when the problem
can be handled.

```
public void withdraw(double amount) throws IllegalArgumentException {  
    if (amount > balance) {  
        throw new IllegalArgumentException("Amount exceeds balance");  
    } else {  
        balance = balance - amount;  
    }  
}
```

Review Read File with Scanner

- The most convenient mechanism for reading text file: 'Scanner' class
- To read input from a disk file, you need 'File' Class as well
 - File describes file's name and directory
- Then, you can use the Scanner methods such as **nextInt**, **nextDouble**, and **next** to read data from the input file.

```
File inputFile = new File("input.txt");
Scanner in = null;

try{
    in = new Scanner(inputFile);
    while (in.hasNextDouble()){
        double value = in.nextDouble();
        System.out.println("read: " + value);
    }
} catch (FileNotFoundException e){
    e.printStackTrace();
} finally{
    in.close();
}
```

input.txt ×	
1	10
2	20
3	30
4	40
5	50

```
read: 10.0
read: 20.0
read: 30.0
read: 40.0
read: 50.0
```

Review Write File with FileWriter

- To write output to a file, you can use 'PrintWriter' object with the specific file name
- If the file already exist, the new data will replace the old one.
- If not, an empty file is created and the data is written.
- You can use the **print**, **println**, and **printf** methods (similar to PrintStream class)

```
try {  
    out = new PrintWriter("output.txt");  
    out.println("Hi, How are you?");  
    out.printf("Total: %8.2f\n", 200.22);  
} catch (FileNotFoundException e) {  
    e.printStackTrace();  
} finally {  
    if(out != null) out.close();  
}
```

output.txt ×

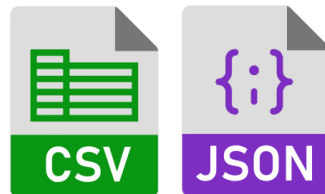
```
1 Hi, How are you?  
2 Total:    200.22  
3 |
```

Today Learning Outcomes

- After finished this class, students are able to
 - *Explain* how to manage CSV files and JSON files correctly
 - *Explain* how to process text or String in java
 - *Implement* a program to manage files and process text in the files

- Topics

1. File I/O
2. Read/Write CSV files
3. Parsing JSON files



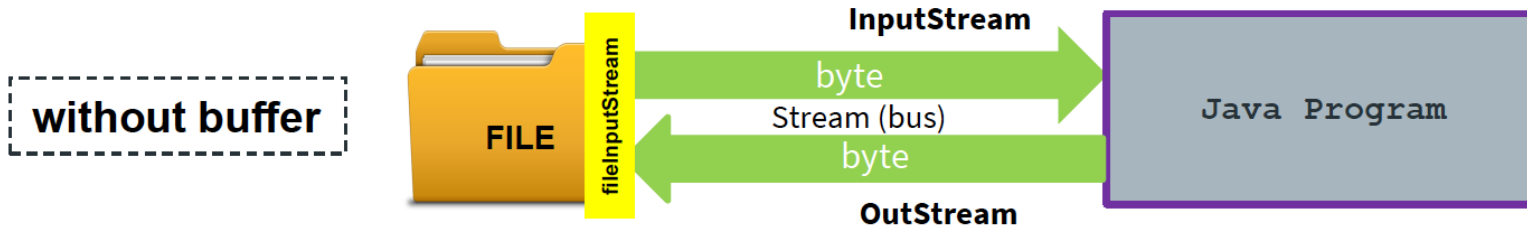
4. String Manipulation
5. Regular Expression



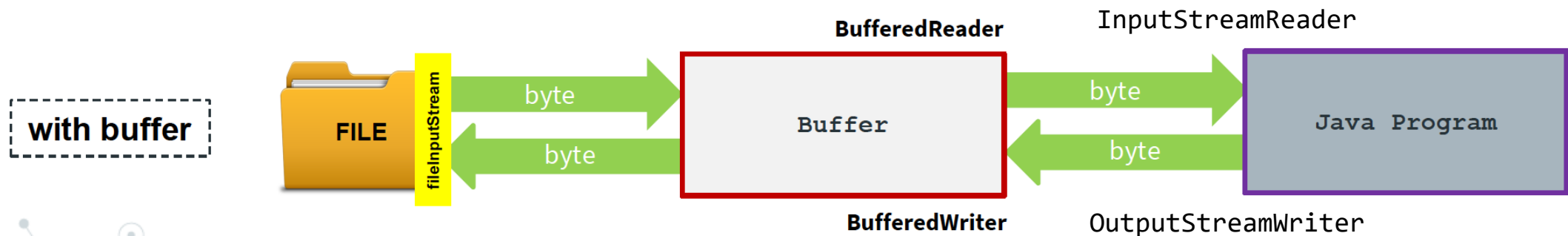
File Input/Output

1. File Input/Output with Buffer

Scanner, FileWriter Class



Inefficient way to read/write files by accessing data directly from the disk. For a small data is okay.



Efficient way to read/write files. A data buffer is temporary memory that can be faster access than disk.

2. Common Java Class to Manage File

- **File:** An object of this class is either a file or a directory.
- **FileInputStream:** obtains input bytes from a file in a file system.
- **OutputStream:** base class for byte-output streams
- **InputStream:** base class for byte-input streams
- **BufferedReader:** Reads text from a character-input stream.
- **BufferedWriter:** Writes text from a character-input stream.
- **Writer:** base class for character-output streams.
- **Reader:** base class for character-input streams.

Example: Read File one Line at a time with/without Buffer

```
public static void main(String[] args) throws FileNotFoundException{
    File file = new File("ICTStudent.csv");
    try {
        FileInputStream fs = new FileInputStream(file);
        InputStreamReader fr = new InputStreamReader(fs);
        int line;
        while((line = fr.read()) != -1) {
            System.out.print((char) line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
public static void main(String[] args) throws FileNotFoundException{
    File file = new File("ICTStudent.csv");
    try {
        FileInputStream fs = new FileInputStream(file);
        InputStreamReader fr = new InputStreamReader(fs);
        BufferedReader br = new BufferedReader(fr);
        int line;
        while((line = br.read()) != -1) {
            System.out.print((char) line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Example: Write File with/without Buffer



```
public static void main(String[] args) throws FileNotFoundException{  
    File file = new File("ICTStudent.csv");  
    try {  
        FileOutputStream fs = new FileOutputStream(file);  
        OutputStreamWriter fr = new OutputStreamWriter(fs);  
        fr.write("Hello World");  
        fr.close();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

```
public static void main(String[] args) throws FileNotFoundException{  
    File file = new File("ICTStudent.csv");  
    try {  
        FileOutputStream fs = new FileOutputStream(file);  
        OutputStreamWriter fr = new OutputStreamWriter(fs);  
        BufferedWriter br = new BufferedWriter(fr);  
        br.write("Hello World");  
        br.close();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

Appending mode
= new FileOutputStream(file, true)

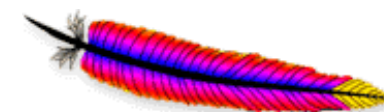
2. Using External Library: Apache Commons

```
File file= newFile("log.txt");

try{ // To write
    FileUtils.write(file, "Hello World.\n", "UTF-8");
} catch(IOException) {e.printStackTrace();}

try{ // To append
    FileUtils.write(file, "Hello Mars.\n", "UTF-8", true);
} catch(IOException) {e.printStackTrace();}

try{ // To read
    List<String> lines= FileUtils.readlines(file, "UTF-8");
    for(String line: lines) {
        System.out.println(line);
    }
} catch(IOException) {e.printStackTrace();}
```



Apache CommonsTM
<http://commons.apache.org/>



Read/Write CSV files

1. CSV File Format

- CSV stands for Comma-Separated Values
- It is one of the simple file formats yet widely used to store tabular data in simple text form, such as a spreadsheet or database.
- Each field may or may not be enclosed in double-quotes.
- A CSV file may or may not have a header row
- When dealing with CSV files, it's OK to use MS Excel to view it, but **avoid saving CSV files with Excel**. It can mess up with the format and character encoding!
- Other variants of character-delimited file formats include TSV (tab), SCSV (;), etc.
- CSV files are used a lot in data science due to its simple, platform-independent format that can directly translate into tabular data structure and are supported by many popular tools such as Pandas, Weka, Orange, etc.

Without headers

```
1995,Java,James Gosling  
"1995","Java","James Gosling"  
"March 22, 2022",Java, Quotes in "Cell"
```

With headers

```
year, language, name  
1995,Java,James Gosling  
"1995","Java","James Gosling"  
"March 22, 2022",Java, Quotes in "Cell"
```

Snippet of real CSV file in a machine learning project

winequality-red.csv

```
1 "fixed.acidity","volatile.acidity","citric.acid","residual.sugar",  
  "chlorides","free.sulfur.dioxide","total.sulfur.dioxide","density","pH",  
  "sulphates","alcohol","quality"  
2 7.4,0.7,0,1.9,0.076,11,34,0.9978,3.51,0.56,9.4,5  
3 7.8,0.88,0,2.6,0.098,25,67,0.9968,3.2,0.68,9.8,5  
4 7.8,0.76,0.04,2.3,0.092,15,54,0.997,3.26,0.65,9.8,5  
5 11.2,0.28,0.56,1.9,0.075,17,60,0.998,3.16,0.58,9.8,6  
6 7.4,0.7,0,1.9,0.076,11,34,0.9978,3.51,0.56,9.4,5  
7 7.4,0.66,0,1.8,0.075,13,40,0.9978,3.51,0.56,9.4,5  
8 7.9,0.6,0.06,1.6,0.069,15,59,0.9964,3.3,0.46,9.4,5  
9 7.3,0.65,0,1.2,0.065,15,21,0.9946,3.39,0.47,10,7  
10 7.8,0.58,0.02,2,0.073,9,18,0.9968,3.36,0.57,9.5,7  
11 7.5,0.5,0.36,6.1,0.071,17,102,0.9978,3.35,0.8,10.5,5  
12 6.7,0.58,0.08,1.8,0.097,15,65,0.9959,3.28,0.54,9.2,5  
13 7.5,0.5,0.36,6.1,0.071,17,102,0.9978,3.35,0.8,10.5,5  
14 5.6,0.615,0,1.6,0.089,16,59,0.9943,3.58,0.52,9.9,5  
15 7.8,0.61,0.29,1.6,0.114,9,29,0.9974,3.26,1.56,9.1,5  
16 8.9,0.62,0.18,3.8,0.176,52,145,0.9986,3.16,0.88,9.2,5  
17 8.9,0.62,0.19,3.9,0.17,51,148,0.9986,3.17,0.93,9.2,5  
18 8.5,0.28,0.56,1.8,0.092,35,103,0.9969,3.3,0.75,10.5,7  
19 8.1,0.56,0.28,1.7,0.368,16,56,0.9968,3.11,1.28,9.3,5  
20 7.4,0.59,0.08,4.4,0.086,6,29,0.9974,3.38,0.5,9,4  
21 7.9,0.32,0.51,1.8,0.341,17,56,0.9969,3.04,1.08,9.2,6  
22 8.9,0.22,0.48,1.8,0.077,29,60,0.9968,3.39,0.53,9.4,6  
23 7.6,0.39,0.31,2.3,0.082,23,71,0.9982,3.52,0.65,9.7,5  
24 7.9,0.43,0.21,1.6,0.106,10,37,0.9966,3.17,0.91,9.5,5  
25 8.5,0.49,0.11,2.3,0.084,9,67,0.9968,3.17,0.53,9.4,5  
26 6.9,0.4,0.14,2.4,0.085,21,40,0.9968,3.43,0.63,9.7,6  
27 6.3,0.39,0.16,1.4,0.08,11,23,0.9955,3.34,0.56,9.3,5  
28 7.6,0.41,0.24,1.8,0.08,4,11,0.9962,3.28,0.59,9.5,5
```


Snippet of real CSV file in a machine learning project



goemotion-1.csv

[illegible]

Snippet of real CSV file in a machine learning project

raw-news.csv

```
1 datetime,comment,tag,title,content
2 2017-04-03 21:09,0,ยังสาวครูสภาฯฆ่าคู่สวามี;ยังกันตาย;ข่าววันนี้;ข่าวด่วน,"มือ ยิง สาว ซี อ่าง ตกลง คืบเงิน กู้ร่วม ไม่ได้ จึง ยิง ดับ ผู้บุกรุกการ ตำรวจนครบาล แกลง จับกุม มือ ยิง สาว ครูสภา เสียชีวิต อ่าง ตนเอง ถูก แจ้งความ สักทรัพย์ และ ตกลง คืบเงิน ที่ กู้ สวัสดิการ ร่วมกัน ไม่ได้ จึง ก่อเหตุ ยิง พล ตำรวจ ทศ นิติย-
การ ผู้บุกรุกการ ตำรวจนครบาล พร้อม ตำรวจ ฝ่าย สืบสวน ร่วม สอบปากคำ นาย จ่าลอง พุ่ม มาลัย ชาว จังหวัด สมุทรปราการ ผู้ต้องหา ใช้อาวุธ ปืน ขนาด มม. ก่อเหตุ ยิง นาง วาสนา บุ รา อายุ ปี เจ้าหน้าที่ ซี ฝ่าย ทบเย็น คลัง สำนักงาน คณะกรรมการ การ ส่งเสริม สวัสดิการ และ สวัสดิภาพ ครู และ บุคลากรทางการ ศึกษา-
หรือ ที่ บริเวณ สนามหญ้า หน้า อาคาร ครูสภา เมื่อ วันที่ มีนาคม โดย หลัง เกิดเหตุ นาย จ่าลอง ได้ หลบหนี ไป อยู่ ที่ จังหวัด อ่างทอง ตำรวจ ได้ สืบสวน จน สามารถ ติดตาม ตัว นาย จ่าลอง ได้ จึง นัดหมาย ให้ เข้า มอบตัว ที่ วัด เขากแก้ว อำเภอ โพธิ์ทอง จังหวัด อ่างทอง พร้อมกับ ของกลาง ที่ ใช้ ก่อเหตุ คือ ปืน ขนาด มิลลิเมตร
พร้อม กระสุน นัด จาก การ สอบถาม นาย จ่าลอง อ้างว่า ที่ ก่อเหตุ ไป เนื่องจาก ก่อนหน้านี้ ผู้เสียชีวิต แจ้งความ ว่า ตัวเอง ก่อเหตุ สักทรัพย์ รักษารายานยนต์ บิ๊ก ไป ซึ่ง นาย จ่าลอง ยืนยัน ว่า รัก กัน เป็น รก ที่ ชื่อ ด้วย เงิน ของ ตัวเอง นอกจากนั้น ยัง มี การ กู้เงิน สวัสดิการ ร่วมกัน และ เมื่อ เลิก รัก กัน พยายาม ขอ เงิน คืน แต่
ผู้เสียชีวิต พยายาม ป้ายเบี่ยง จน มา ปลอม ลม รถยนต์ ของ ผู้เสียชีวิต ใน วันเกิดเหตุ เพื่อที่จะ ตกลง กัน แต่ ตกลง กัน ไม่ได้ และ ยัง มา พบ ว่า ผู้เสียชีวิต คบหา กับ ผู้ชาย อีก คน หนึ่ง ทำให้ ตัดสินใจ ใช้ ปืน ที่ พก มา ด้วย ก่อเหตุ ทั้งนี้ เจ้าหน้าที่ ตำรวจ จะ ควบคุมตัว ผู้ต้องหา ไป ทำ แผน ประกอบ คำรับสารภาพ ใน จุด เกิดเหตุ
เวลา ประมาณ น."
```

3 2017-04-03 20:42,0,อาชญากรรม,"ตร. ตรวจค้น แก๊ง ปลอย-เงินกู้ หั่ว ปท. จับ ราย ตำรวจ เปิดบัญชี นายทุน ปลอย-กู้ นอก ระบบ พบ เก็บ ราย คืบ เป้าหมายหลัก ยึดอาวุธปืน ของกลาง และ เอกสาร ไว้ ตรวจสอบ จำนวนมาก พล.ต.อ. เฉลิม เกียรติ ศร ขาน รอง ผู้บัญชาการ ตำรวจ แห่งชาติ ในฐานะ หัวหน้า คณะทำงาน
ขับเคลื่อน การ แก้ไขปัญหาหนี้ นอก ระบบ อย่าง บูรณาการ และ ยั่งยืน แกลง ผล การขับเคลื่อน โครงการ หนี้ นอก ระบบ เพื่อ ต้องการ ขจัด กลุ่ม นายทุน ที่เก็บ ดอกเบี้ย หนี้ ไม่เป็นธรรม หรือ มี พฤติกรรม ติดตาม ทวงหนี้ ข่มขู่ ใช้ ความรุนแรง และ กำหนด ให้ มี การ ประกอบธุรกิจ สินเชื่อ รายย่อย ประเภท ใหม่ เรียก ว่า สินเชื่อ
รายย่อย ระดับ จังหวัด โดย ได้ ส่งการ ให้ ทูท หน่วย ในสังกัด ดำเนินการรวบรวม รายชื่อ กลุ่ม นายทุน เงินกู้ นอก ระบบ ทั้งหมด จำนวน ทั้งสิ้น ราย และ ให้ ระดม ปิดล้อม ตรวจค้น เป้าหมาย พร้อมกัน ทั่วประเทศ วัน นี้ เมย. เบื้องต้น สามารถ จับกุม ผู้กระทำ ความผิด ตาม พ.ร.บ. อาวุธ ปืน และ ประมวลกฎหมายอาญา ที่ เกี่ยว ของ
จำนวน ราย สำหรับ เครือข่าย ที่ นำสนธิ อาติ ทราบ จัด หน้า ได้ ปลอย-เงินกู้ นอก ระบบ เก็บ กว่า ที่ กฎหมาย กำหนด และ ใช้ ขยายการจับ จำนวนมาก กว่า นาย ตระเวน ปลอย-เงิน และ เก็บเงิน ใน พื้นที่ จว. เพชรบุรี และ พื้นที่ ใกล้เคียง จึง ยื่นคำร้อง ต่อ ศาลจังหวัด เพชรบุรี อนุมัติ หมายค้น จำนวน หมาย ผล การ ตรวจค้น สามารถ
จับกุม ผู้ต้องหา ชาว จังหวัด ราชบุรี พร้อม ของกลาง การ อาติ อาวุธ ปืน เอกสาร แฟ้ม ต่างๆ"

4 2017-04-03 19:31,0,แท็กซี่หญิง;แท็กซี่ไล่ผู้โดยสาร;ข่าววันนี้;ข่าวด่วน;อาชญากรรม,"เหยื่อ อีก ราย ร้อง ตำรวจ ถูก แท็กซี่ หญิง ลก ทรัพย์ ปี ผู้เสียหาย อีก ราย ที่ ถูก คนขับ แท็กซี่ หญิง ลก ทรัพย์ เมื่อ ปี ร้อง รอง ผบ.ชน. หลัง คดี ไม่ คืบ นาง สาว ยาก-ยมนา นนท์ เดินทาง มา ที่ สน. พหลโยธิน เพื่อ ติดตาม คดี หลังจาก ที่
ปี-เคย-แจ้งความ เอาผิด นาง ผุ สด ภา ดิ คนขับ แท็กซี่ หญิง ลก เงิน จำนวน กว่า บาท และ ไล่ ลง จาก รถ จน ทำให้ ประตุ รก หนีบ ขา จน ได้รับ บาดเจ็บ แต่ คดี ไม่ คืบ ซึ่ง คนขับ แท็กซี่ ราย ดังกล่าว เป็น คนเดียวกับ ที่ ปรากฏ ใน คลิป วิดีโอ ไล่ ผู้โดยสาร ลง จาก รถ แท็กซี่ เมื่อไม่นานมานี้ นาง สาว ยาก-กล่าวว่า เมื่อ ช่วง
ประมาณ เดือน กันยายน ปี ได้ เรียก แท็กซี่ คน ดังกล่าว จาก ห้างสรรพสินค้า แห่ง หนึ่ง ไป ย่าน ลาดพร้าว แต่ ตลอดทาง ถูก คนขับ พยายาม ใช้ วาจา ไม่สุภาพ ต่อว่า ตนเอง ว่า เรื่องมาก จะ ให้ เข้าไป สง ใน ขอย แคล ซึ่ง ตนเอง ก็ ได้ ขอ ลง เพื่อ จะ ต่อ แท็กซี่ คน อื่น แต่ แท็กซี่ ราย ดังกล่าว ก็ ไม่ยอม จอด รถ และ พยายาม ไล่
ตนเอง ลง จาก รถ ตลอดเวลา จน ไป ถึง บริเวณ ลาดพร้าว ขอย ที่ เป็น จุด เปลี่ยน และ มีด แท็กซี่ ก็ จอด รถ และ ไล่ พร้อม กระชาก เงิน ไป จาก มือ ประมาณ บาท และ ก็ กระชาก รถ ออก ไป ทันที ทำให้ ประตุ รก ตี กระแทก ที่ บริเวณ หน้าแข้ง จน เป็นแผล ฟกช้ำ จึง ตัดสินใจ กระโดด ลง จาก รถ เพราะ กลัว อันตราย หลังจากนั้น
ได้ เข้า แจ้งความ ไว้ ที่ สน. พหลโยธิน โดย ไม่ คิด ว่า หญิง ราย ดังกล่าว จะ กลับมา ขับรถ แท็กซี่ อีก และ ยัง มี พฤติกรรม เหมือนเดิม วันที่ จึง เดินทาง เข้า ให้ ข้อมูล เพิ่มเติม พลตำรวจตรี มนต์รี ยิ้มแย้ม รอง ผู้บัญชาการ ตำรวจนครบาล ระบุ ว่า เบื้องต้น ตำรวจ จะ แจ้ง ข้อหา หลงหนี้ยา กักขัง และ รักราว ทรัพย์ และ หลังจาก ได้ ผล
ตรวจร่างกาย ผู้เสียหาย จาก แพทย์ ที่ มี การ ตรวจ ไว้ ตั้งแต่ ปี ก็ จะ พิจารณา แจ้ง ข้อกล่าวหา ทำร้ายร่างกาย เพิ่มเติม นอกจากนี้ หาก ผู้ใด พบเห็น นางสาว ผุ สด ก็ สามารถ แจ้ง ตำรวจ ให้ จับกุม ได้ทันที เนื่องจาก เป็น บุคคล มี หมายจับ อยู่ แล้ว"

5 2017-04-03 18:24,0,อาชญากรรม,"วี.ป.บอ รับทราบ ข้อหา ปม ว่า ตร. แก๊ง วิ.สม ความคิด พบ.ตร.บอ รับทราบ ข้อกล่าวหา กรณี โพสต์ ขบ ว่า ตำรวจ ใช้ อำนาจ แกล้ง ประชาชน นาย วิ.สม ความคิด ประธาน กลุ่ม ฟังทัก สิทธิ เสรีภาพ ของ ประชาชน และ เลขาธิการ เครือข่าย ประชาชน ด้าน คอร์รัปชัน ต. พร้อม หมายความ
เดินทาง เข้าพบ พ.ต.ท. หนู ทอง รอง ผกก. สอบสวน กก. บก ปอ เพื่อ มา สอบ-ถาม และ ติดตาม ความคืบหน้า ใน คดี ที่ ถูก แจ้ง พ.ร.บ. คอมพิวเตอร์ กรณี โพสต์ ข้อความ ผ่าน ชป ส่วนตัว ว่า ไม่ได้ ไป รวม ชุมชน คัดค้าน พ.ร.บ. ปีเตอร์ เลียม เพียงแต่ ไปหา เพื่อน แต่ โดน เหมมา ว่า ร่วม ด้วย และ ตำรวจ ใช้ อำนาจ แกล้ง ประชาชน
โดย ใช้เวลา เกือบ ชั โมง นาย วิ.เปิดเผย ว่า วันที่ มา ตาม ความคืบหน้า คดี ที่ ถูก กล่าวหา เรื่อง โพสต์ โหล สารวจ รัฐบาล และ มา รับทราบ ข้อกล่าวหา ไม่แน่ใจ ขอ หา หมิ่นประมาท หรือไม่ แต่ เป็นคดี ใหม่ คาด ว่า เป็นเรื่อง เกี่ยวกับ ที่ ตน ไป โพสต์ ลง ชป ว่า ตำรวจ ใช้ อำนาจ แกล้ง ประชาชน เมื่อ วันที่ เมย. ที่ผ่าน มา ด้วย
คดี นี้ ทราบ ว่า พล.ต.อ. ตร.ราช วิ.พร.ณ.ก.ร.อง.พ.น.ตร.ให้ ลูกน้อง มา แจ้งความ ดำเนิน คดี นอกจากนี้ นาย วิ.กล่าวว่า ตนเอง ทราบ มา ว่า ตำรวจ สน. ดุสิต ได้ ออกหมายเรียก ให้ ไป พบ วันที่ เมย. นี้ โดย ยัง ไม่รู้ว่า ข้อหา อะไร แต่ หาก เกี่ยวข้อง กับ เรื่อง กับ การ ที่ ตนเอง ไป อยู่ แกว สวนสัตว์ ดุสิต วันนั้น ไม่ได้ ไป รวม
ชุมนุม ไม่ได้ ก็คือ โท โขง เลย ไปหา เพื่อน แล้ว ก็ กลับ เลย ไม่รู้ ตำรวจ ดำเนินคดี อะไร กับ ตนเอง ส่วน หมายเรียก ต้อง กลับ ไปดู ที่ บ้าน เนื่องจาก ไม่รู้ มา ได้รับ หรือยัง"

6 2017-04-03 16:04,6,ยังสาวครูสภาฯฆ่าคู่สวามี;ยังกันตาย;ข่าววันนี้;ข่าวด่วน,"รวม แล้ว หนุ่ม ยิง สาว ซี ดับ คากระทรวง หนี ชุก อ่างทอง ตำรวจ สืบสวน ภูธร ภาค รบ-คนร้าย ยิง เจ้าหน้าที่ สาว ของ กระทรวงศึกษาธิการ หลัง หลบหนี กบดาน บ้าน ญาติ ใน จังหวัด อ่างทอง พ.ต.อ. นครินทร์ คน รอง ผู้บังคับการ ตำรวจนครบาล
เปิดเผย ว่า ฝ่าย สืบสวน ตำรวจ ภาค จับกุม ตัว นาย จ่าลอง พุ่ม มาลัย ผู้ จ้อง หา ตาม หมายจับ ฐาน ฆ่าผู้อื่น จาก กรณี ยิง นางสาว วาสนา บุ ชาราชการ กระทรวงศึกษาธิการ เสียชีวิต ภายใน สนามหญ้า กระทรวงศึกษาธิการ โดย สามารถ จับกุม ตัว ได้ เมื่อ ช่วง สาย ของ วันที่ ที่ บ้าน ญาติ ใน พื้นที่ อำเภอก โพธิ์ทอง จ.อ่างทอง
พร้อม อาวุธ ปืน และ เงินสด กว่า ล้าน บาท ซึ่ง การจับกุม ตัว ใน ครั้งนี้ สืบเนื่อง มาจาก ตำรวจ พบ เบาะแส ว่า ผู้ต้องหา พยายาม จะ ติดต่อกับ ภรรยา ซึ่งเป็น ข้าราชการ ครู ใน จ.อ่างทอง หลัง มี การ เบิก กอนเงิน กว่า ล้าน บาท เพื่อ ทำการ หลบหนี อย่างไรก็ตาม ขณะนี้ ตำรวจ อยู่ ระหว่าง ควบคุมตัว ผู้ต้องหา มา สอบปากคำ
เพิ่มเติม ที่ กรุงเทพมหานคร อีก ทั้ง ตำรวจ อาจจะมี เชิญ ตัว ภรรยา ที่ เป็น ข้าราชการ มา สอบปากคำ ด้วยเช่นกัน เนื่องจาก อาจ มี เกี่ยวข้อง กับ เงิน จำนวน ล้าน บาท ที่ มี การ เบิก กอน ก่อนหน้านี้ ชาว ที่ เกี่ยวข้อง หนุ่ม ชัก ปืน ยิง หัว สาว ดับ กลาง กระทรวงศึกษาธิการ"

7 2017-04-03 16:00,3,ขโมยทอง;หนุ่มบาร์เกย์ปล้นทอง;ข่าววันนี้;ข่าวด่วน,"ซ่อน-แผน จับ หนุ่ม บาร์เกย์ ลก แหวน ทอง จาก ร้าน เลย มีประวัติ โขก โชน ตำรวจ พยายาม ซ่อน-แผน จับ หนุ่ม บาร์เกย์ ลก แหวน ทอง เดิน ออกจาก ร้าน หน้าตาเลย สารภาพ หางเงิน ใช้หนี้ เอียนต์ และ เทีย ช่าง สงกรานต์ เผยประวัติ ก่อเหตุ
โขก โชน จาก-กรณี คนร้าย เป็น ชาย ไทย ทำให้ เป็น ลูกค้า เข้าไป เลิก-ขอ ดู แหวน ทอง ตำรวจ พรพรรณ หน้าหลัก สลึง กอน อาศัย ช่วง จังหวัด เจ้าของร้าน เลย หยิบ แหวน วัง ดังกล่าว เดิน ออกจาก ร้าน ไป อย่าง หน้าตาเลย โดย กล้องวงจรปิด สามารถ บันทึก เหตุการณ์ และ ระบุ ราย คนร้าย ไว้ ได้ อย่างชัดเจน เหตุ เกิดขึ้น ที่ ห้าง ทอง
บัว ชาว พัทธยา กลาง ม.ต.หนอง ปลือ อ.บาส ละ มุง จ.ชลบุรี ตามที่ เสนอ เข้า ไป แล้ว นั้น สำหรับ ความคืบหน้า ล่าสุด ผู้สื่อข่าว รายงาน ว่า พ.ต.อ.เอกชัย กรอบ-เพชร ผกก. สก. เมือง พัทธยา จ.ชลบุรี พร้อมด้วย พ.ต.ท. พงศ์วัจ สาร สว สส.ร.ด.อ. ณ กร จังหวัด อด รอง สว สส และ กาลัง ชุด สืบสวน สามารถ จับกุม ตัว คนร้าย แล้ว
ทราบ ชื่อ คือ นาย เมย ชย ชา-ภู ศิลป อยู่ ปี ผู้ต้องหา ตาม หมายจับ คดี รักราว ทรัพย์ ของ ศาลจังหวัด พัทธยา ที่ จ.ลพบุรี เมื่อเปิดเผย ว่า หลังจาก เกิดเหตุ ได้ ส่งการ ชุด สืบสวน ลงพื้นที่ ตรวจสอบ ภาพ กล้องวงจรปิด กอน จะ ทราบ ว่า นาย ชย ชา เป็น ผู้ก่อเหตุ ใน ครั้งนี้ จึง รวบรวม พยาน และ หลักฐาน
อนุมัติ ศาล เพื่อ ออกหมายจับ กอน ได้ สืบ ทราบ ว่า นาย ชย ชา หลังจาก ก่อเหตุ ได้ หลบหนี กบดาน ไป ยัง บ้านเกิด ที่ จังหวัด นครสวรรค์ จึง นำ กาลัง ไป จับกุม ตัว แต่ ปรากฏ ว่า นาย ชย ชา ไหวตัว หนี หลบหนี การจับกุม ไป ได้ เจ้าหน้าที่ ใช้เวลา พยายาม ติดตาม ตัว มา หลาย วัน แต่ ก็ ไม่สามารถ จับกุม ได้ จึง ใช้วิธี ส่ง
สายลับ ติดต่อกับ นาย ชย ชา โดย ออกอุบาย ให้ นาย ชย ชา มา ขอใช้ ค่าเสียหาย และ ค่าปรับ อีก จำนวน หนึ่ง โดย ทาง เจ้าของร้าน ทอง จะ ยินยอม ไม่ผิด ระหว่าง นาย ชย ชา กาลัง ยื่น เงิน ขอใช้ ค่าเสียหาย เจ้าหน้าที่ ตำรวจ ที่ ปลอมตัว เป็น พนักงาน ร้าน ทอง จึง แสดง หมายจับ เข้า ทำการ จับกุม ตัว นาย ชย ชา ทันที ทำเอา
นาย ชย ชา คอตก หลังจาก ถูก ตำรวจ ซ่อน-แผน จับกุม เบื้องต้น นาย ชย ชา ให้การ รับสารภาพ ว่า ปลด ได้ ประกอบอาชีพ เป็น พนักงาน ต้น อยู่ ที่ บาร์ เกย์ แห่ง หนึ่ง ย่าน ถนน เลียบ ชายหาด เมือง พัทธยา ซึ่ง ยอมรับ ว่า ตนเอง เป็น ผู้ก่อเหตุ ใน ครั้งนี้ จึง เนื่องจาก ต้องการ นำ-เงิน ไป ใช้หนี้ เอียนต์ ยาเสพติด และ
เทียเรต ช่วง วัน สงกรานต์ นอกจากนี้ ยัง พบ ว่า นาย ชย ชา เคย ถูกจับ-กุม คดี ยาเสพติด มา แล้ว ถึง ครั้ง อีก ทั้ง ได้ หลบหนี หมายจับ คดี ชกชน และ อานาจร ใน พื้นที่ สก.หนอง จ.นครสวรรค์ อีกด้วย ซึ่ง เป็น ผู้ต้องหา ที่ ตำรวจ กาลัง ต้องการ ตัว อยู่พอดี อย่างไรก็ตาม จึง ควบคุมตัว ส่ง พนักงานสอบสวน ดำเนินคดี
ตามกฎหมาย ต่อไป"

Reading CSV Files in Java

- Java provides several classes and methods to read files, including CSV files. The `BufferedReader` and `FileReader` classes are commonly used for this purpose, offering an efficient way to read text files line by line. When working with CSV files, which are essentially text files with values separated by commas, these classes enable you to process each row of data individually.

```
try (BufferedReader br = new BufferedReader(new
FileReader("data.csv"))) {
    String line;
    while ((line = br.readLine()) != null) {
        System.out.println(line); // Process the line as
needed
    }
} catch (IOException e) {
    e.printStackTrace();
}
```



Read/Write CSV files

With Apache Commons CSV

2. Reading/Writing a CSV File

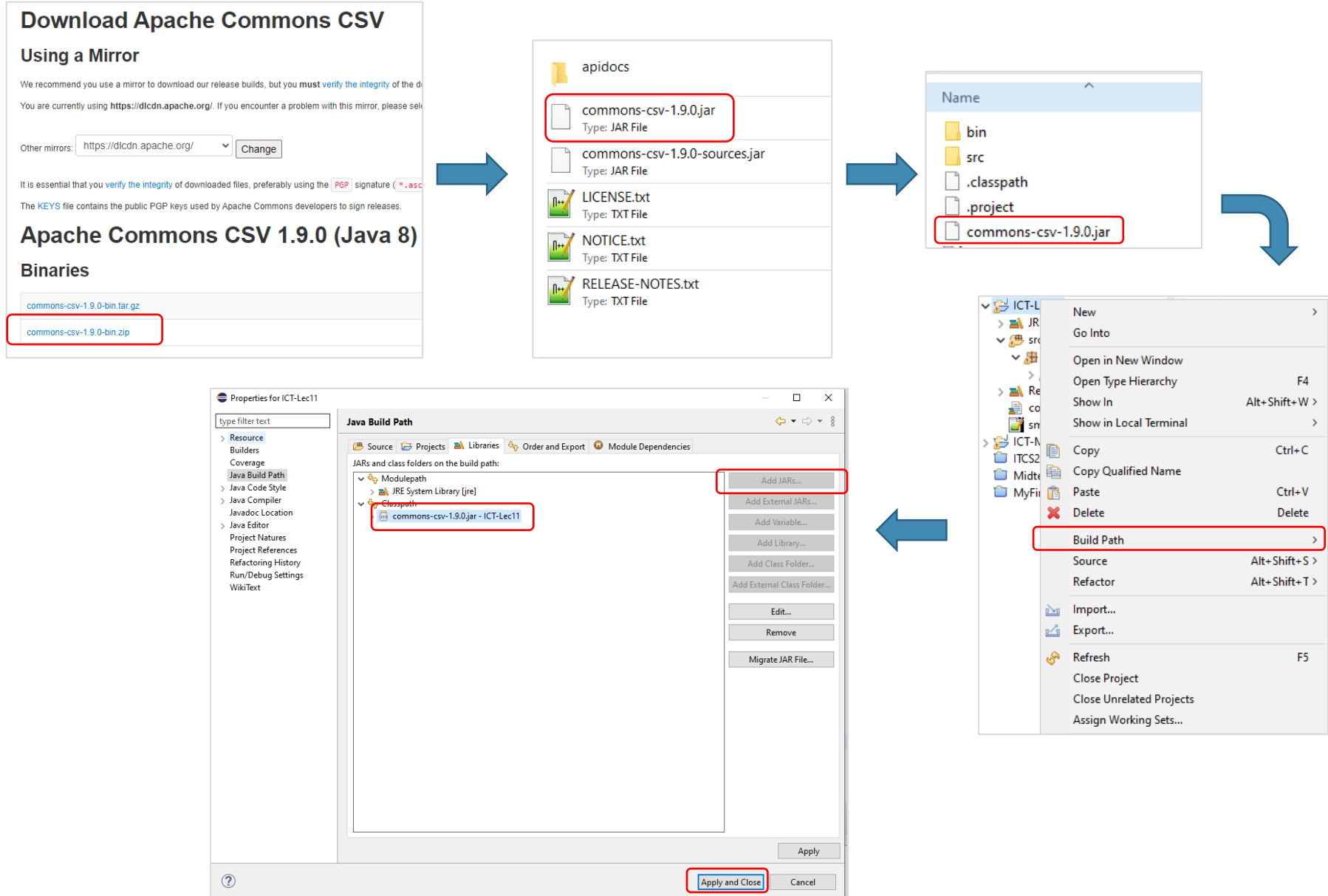
- CSV files can be large. **It is advised to read and write a CSV file in a buffer manner.** That is you should not store a bunch of string and dump it all to a file at once, nor should you read all the content into memory before processing.
- There are many ways to read/write CSV files. Due to its simple format, you can write your own CSV reader/writer from scratch.
- But here, we will show you how to use **Apache Commons CSV**, which can save you some implementation time and help you deal with some rare cases.



Commons CSV™

<https://commons.apache.org/proper/commons-csv/>

Download and Import Commons CSV Library



Writing CSV File

```
public static final String[] header = {"ID", "Name", "E-mail"};
public static final String[][] students = {
    {"6488125", "David Beckham", "dback@school.edu"},
    {"6488126", "Christina Aguilera", "caguilera@school.edu"},
    {"6488127", "Lady Gaga", "lgaga@school.edu"}
};

public static void writeCSV(String outCsvFilename, String[] header, String[][] input)
{
    CSVPrinter printer = null;
    try {
        //create a CSV printer handler
        printer = new CSVPrinter(new FileWriter(outCsvFilename), CSVFormat.DEFAULT);

        //print headers
        printer.printRecord(Arrays.asList(header));

        //print data each row
        for(String[] row: input)
        {
            printer.printRecord(Arrays.asList(row));
        }

    } catch (IOException e) {
        e.printStackTrace();
    } finally
    {
        if(printer != null) try { printer.close(); } catch (IOException e) { e.printStackTrace(); }
    }
}
```

```
writeCSV("test-students.csv", header, students);
```



test-students.csv

```
ID,Name,E-mail
6488125,David Beckham,dback@school.edu
6488126,Christina Aguilera,caguilera@school.edu
6488127,Lady Gaga,lgaga@school.edu
```


Reading CSV (Do not care about headers)

```
public static void readCSVSimple(String csvFilename)
{
    CSVParser csvParser = null;
    try {
        //create a parser
        csvParser = new CSVParser(new FileReader(csvFilename), CSVFormat.DEFAULT);

        //parse each row using column IDs as indexes
        for (CSVRecord record : csvParser) {
            for(int colID = 0; colID < record.size(); colID++)
            {
                System.out.print(record.get(colID)+"|");
            }
            System.out.println();
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    finally {
        try {
            if(csvParser != null) csvParser.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
readCSVSimple("small-no-headers.csv");
```

small-no-headers.csv

```
1995,Java,James Gosling
"1995","Java","James Gosling"
"March 22, 2022",Java, Quotes in "Cell"
```



Editor console

```
1995|Java|James Gosling|
1995|Java|James Gosling|
March 22, 2022|Java| Quotes in "Cell"|
```

Reading (Header-less) CSV with Custom Headers

```
public static void readCSVwithCustomHeaders(String csvFilename, String[] headers)
{
    System.out.println("Custom headers: "+Arrays.toString(headers));
    CSVParser csvRecordsWithHeader = null;
    try {
        //create a parser with assigned custom headers
        csvRecordsWithHeader = CSVFormat.DEFAULT.withHeader(headers).parse(new FileReader(csvFilename));
        //Though deprecated, still usable

        //parse each row using String headers as indexes
        for (CSVRecord record : csvRecordsWithHeader) {
            for(int colID = 0; colID < headers.length; colID++)
            {
                System.out.print(headers[colID]+":"+record.get(headers[colID])+"|");
            }
            System.out.println();
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    finally {
        try {
            if(csvRecordsWithHeader != null) csvRecordsWithHeader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
readCSVwithCustomHeaders("small-no-headers.csv",
    new String[]{"year", "language", "name"});
```

small-no-headers.csv

```
1995,Java,James Gosling
"1995","Java","James Gosling"
"March 22, 2022",Java, Quotes in "Cell"
```



Editor console

```
Custom headers: [year, language, name]
year:1995|language:Java|name:James Gosling|
year:1995|language:Java|name:James Gosling|
year:March 22, 2022|language:Java|name: Quotes in "Cell"|
```

//updated: 30Mar2023

```
public static void readCSVwithCustomHeaders(String csvFilename, String[] headers) {
    CSVParser csvRecordWithHeader = null;
    CSVFormat format = CSVFormat.DEFAULT.builder()
        .setHeader(header)
        .build();

    try {
        csvRecordWithHeader = new CSVParser(new FileReader(csvFilename), format);
    }
```


Reading CSV with Headers

```
public static void readCSVwithHeaders(String csvFilename)
{
    CSVParser csvRecordsWithHeader = null;
    try {
        //create a parser and auto detect headers
        csvRecordsWithHeader = CSVFormat.DEFAULT.withFirstRecordAsHeader().parse(new FileReader(csvFilename));
        //Though deprecated, still usable

        List<String> headers = csvRecordsWithHeader.getHeaderNames();
        System.out.println("Detected Headers: "+headers);

        //parse each row using String headers as indexes
        for (CSVRecord record : csvRecordsWithHeader) {
            for(int colID = 0; colID < record.size(); colID++)
            {
                System.out.print(headers.get(colID)+":"+record.get(headers.get(colID))+ "|");
            }
            System.out.println();
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    finally {
        try { if(csvRecordsWithHeader != null) csvRecordsWithHeader.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
readCSVwithHeaders("small-with-headers.csv");
```

small-with-headers.csv

```
year, language, name
1995,Java,James Gosling
"1995","Java","James Gosling"
"March 22, 2022",Python, Quotes in "Cell"
```



Editor console

```
Detected Headers: [year, language, name]
year:1995| language:Java| name:James Gosling|
year:1995| language:Java| name:James Gosling|
year:March 22, 2022| language:Python| name: Quotes in "Cell"|
```

//updated: 30Mar2023

```
public static void readCSVwithHeaders(String csvFilename) {
    CSVParser csvRecordWithHeader = null;
    CSVFormat format = CSVFormat.DEFAULT.builder()
        .setHeader()
        .setSkipHeaderRecord(true)
        .build();

    try {
        csvRecordWithHeader = new CSVParser(new FileReader(csvFilename), format);
    }
```



Parsing JSON Files

With Libraries

1. What is JSON (JavaScript Object Notation)

- It is a lightweight data interchange standard
- It is easy for humans to read and write, as well as for machines to parse and generate
- If the data has a complex structure, JSON is better than CSV format.
- JSON is built on two structures:
 - **Object**: a collection of name/value pairs
 - **Array**: an ordered list of values or objects

```
[
  {
    "course_code": "ITCS209",
    "course_name": "Object Oriented Programming",
    "credit": 3,
    "instructors": ["Siripen", "Petch", "Suppawong"]
  },
  {
    "course_code": "ITCS201",
    "course_name": "Fundamentals of Programming",
    "credit": 3,
    "instructors": ["Akara", "Tippajin"]
  }
]
```

2. Working with JSON Format

- Two things are required:
 - **Serialization**: encode Java Object to its JSON representation
 - **Deserialization**: decode String back to an equivalent Java Object

```
public class Course {  
    String course_code: new Course("ITCS209", "OOP", 3)  
    String course_name:  
    int credit;  
  
    Public Course (String c, String n, int credit) {  
        ...  
    }  
}
```

Serialization

Deserialization

```
{  
    "course_code": "ITCS209",  
    "course_name": "OOP",  
    "credit": 3  
}
```

- How to do that in Java?
 - Option 1: Write your own code to parse JSON text file **// NOT recommended**
 - Option 2: Using external Java library **// YES YES YES**

JSON External JAR Libraries

- JSON.simple by Yidong Fang
 - <https://github.com/fangyidong/json-simple> (last updated is 10 years ago)
- GSON by Google
 - <https://github.com/google/gson>
- Jackson by Faster XML
 - <https://github.com/FasterXML/jackson>
- JSONP (JSON Processing) by Oracle
 - <https://javaee.github.io/jsonp/>

Ref: Benchmak -> <https://www.overops.com/blog/the-ultimate-json-library-json-simple-vs-gson-vs-jackson-vs-json/>

Simple Serialization GSON



```
public class Course {  
    private String name;  
    private int credit;  
    private List<String> instructors;  
  
    public Course(String name, int credit, List<String> instructors) {  
        this.name = name;  
        this.credit = credit;  
        this.instructors = instructors;  
    }  
    ...  
}  
  
// -- main method --  
Course course = new Course("OOP", 3, Arrays.asList("Siripen", "Petch", "Suppawong"));  
String serializedCourse = new Gson().toJson(course);  
System.out.println(serializedCourse);
```

OUTPUT

```
{"name":"OOP","credit":3,"instructors":["Siripen","Petch","Suppawong"]}
```

Simple Deserialization GSON

```
public class Course {
    private String name;
    private int credit;
    private List<String> instructors;

    public Course(String name, int credit, List<String> instructors) {
        this.name = name;
        this.credit = credit;
        this.instructors = instructors;
    }

    public String toString() {
        return "name=" + name + "::credit=" + credit + "::instructors=" + instructors.toString();
    }
}

// -- main method --
String courseJson = "{\\"name\\":\\"OOP\\",\\"credit\\":3," +
    "\\"instructors\\":[\\"Siripen\\",\\"Petch\\",\\"Suppawong\\"]}";

Course oopCourse = new Gson().fromJson(courseJson, Course.class);
System.out.println(oopCourse);
```

OUTPUT

name=OOP::credit=3::instructors=[Siripen, Petch, Suppawong]



String Manipulation

1. String Class

- Strings are "**immutable**" objects: once instantiated, a String object is constant and not changeable.
- because String objects are immutable, they can be shared fearlessly: no one can change your object.
- Java optimizes memory by maintaining a **pool of shared Strings**:
the constants "A", "A", "A" will have three references to the same String object.
- Note: an empty string object is not null
 - literal: "" is a String object with a length() of 0

1.1 Strings are Special in Java

- Strings are used so often in programming, Java makes special allowances for coding them

- **Construct** a String without new String()

```
String s1 = new String("some text");
```

```
String s2 = "more text";
```

- the only overloaded **operators (e.g., +)** in Java are for Strings

```
s1 += s2; // s1 = s1.concat(s2);
```

```
s1 = s2 + "etc"; // s1 = s2.concat("etc");
```

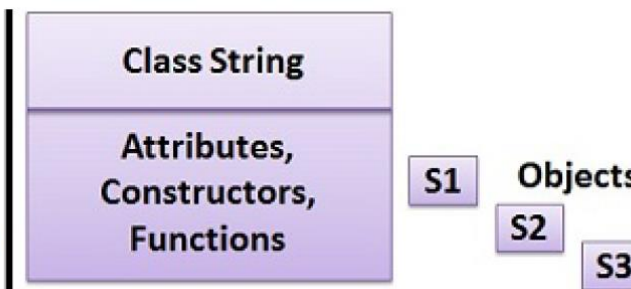
1.2 String Comparision

- **comparison** of two String objects:
 - `thisString.equals(thatString)` *compares contents*
 - *this is what most of us mean most of the time*
 - `thisString == thatString` *compares obj.ref.*
 - *this may seem like it works but is unreliable*
- when is thisString really thatString
- an array of characters vs a String object

name[0] name[1] name[2] name[3] name[4]

'A'	'j'	'I'	'n'	'g'
-----	-----	-----	-----	-----

Character Array -> `char[] name = new char[5]`



2. Useful String Class Methods

- **length()** returns int of character count
- **trim()** returns String exclusive of lead/trail blanks
- **toUpperCase(), toLowerCase()** returns consistent case
- **valueOf()** returns String of any primitive
- **indexOf()** returns int locating a char or substring
- **charAt()** allows processing of string like char[]
- **substring()** returns a substring from this string
- **replace()** changes characters
- **replaceAll()** changes strings with regular expressions
- **split()** splits a string into an array of strings using reg.exp.

3. Anything can be a String, just ask

- static method `String.valueOf()`
 - returns a String
 - can take almost anything as a parameter: all primitives, char array, any object.
- all objects inherit or override the Object class `toString()` method
- `System.out.println()` automatically calls `toString()` on any object in the parameter list

```
System.out.println(myObject); // is same as  
System.out.println(myObject.toString() );
```

4. StringBuilder Class

- `StringBuilder` class is a **mutable version** of `String`.
- You can keep `append()`'ing **without** having to create a new string object.
 - Save a lot of time and memory when dealing with massive text.
- For example, if you need to process and write 1M double values into a text file one by one.
 - It's faster to keep appending them to a `StringBuilder` object first, then write the whole thing into the file.
 - Disk I/O is an expensive operation.



Regular Expression

Can you extract information from text?

We can easily extract information from CSV file which contains structured data.

ID,Name,E-mail

6488125,David Beckham,dback@school.edu

6488126,Christina Aguilera,caguilera@school.edu

6488127,Lady Gaga,lgaga@school.edu

How about extract person's information from an unstructured data?

My name is Siripen Pongpaichet. My student ID is 6488999. My phone number is 02-441-0909 and 02-441-0990. Please contact me via my student email siripen.pon@student.mahidol.ac.th or my personal email siripen.pon@mail.com

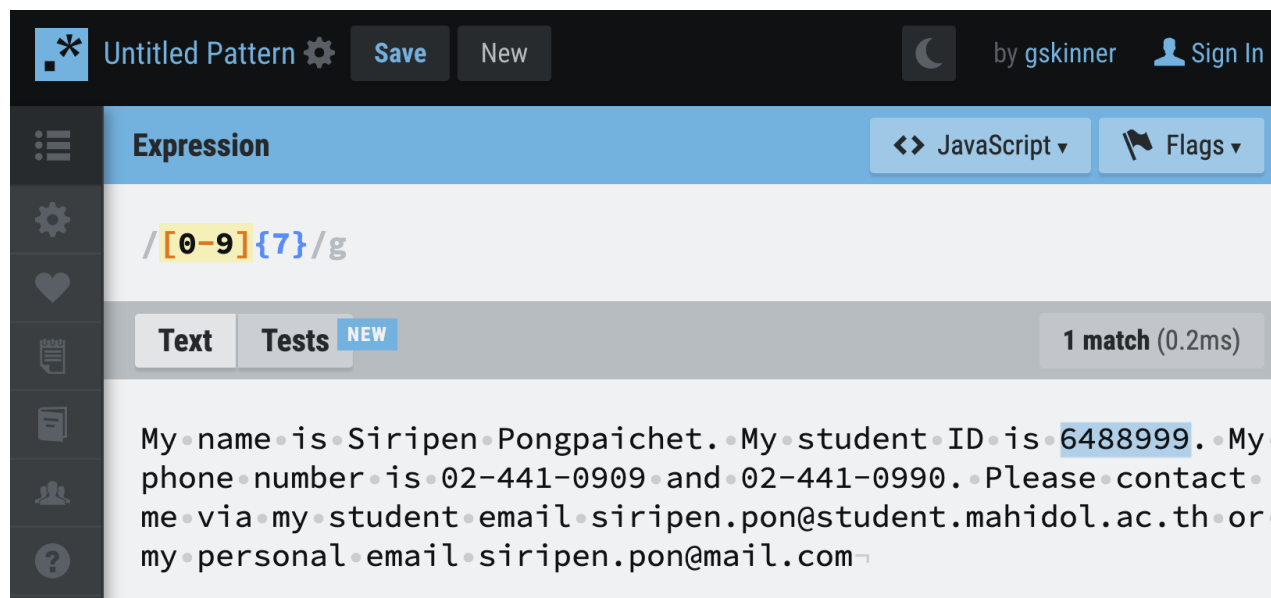
Do you see any patterns for the following information:
Student's ID, Phone Number, and email?

1. What is Regular Expressions (Regex)

- Regular expressions (or Regex) describe character patterns
 - For example, the regex of a student ID is `[0-9]{7}`.
 - `[0-9]` denotes any digit between 0 and 9, and `{7}` means 7 occurrences
- A regular expression either matches the text (or part of the text), or it fails to match
 - If a regular expression matches a part of the text, then you can easily find out which part
 - If a regular expression is complex, then you can easily find out which parts of the regular expression match which parts of the text
 - With this information, you can readily extract parts of the text, or do substitutions in the text
- Regular expressions are an extremely useful tool for manipulating text

Let's try RegEx Online Tool

- <https://regexr.com/>
- <https://regex101.com/>
- <https://www.regexpal.com/>



The screenshot shows the regexr.com interface. At the top, it says "Untitled Pattern" with "Save" and "New" buttons. The "Expression" field contains the regex `/[0-9]{7}/g`. Below the expression, there are tabs for "Text" and "Tests" (with a "NEW" badge). The "Text" tab is active, showing a sample text: "My name is Siripen Pongpaichet. My student ID is 6488999. My phone number is 02-441-0909 and 02-441-0990. Please contact me via my student email siripen.pon@student.mahidol.ac.th or my personal email siripen.pon@mail.com". A single match is highlighted in blue, corresponding to the student ID "6488999". The match count is "1 match (0.2ms)".



The screenshot shows the same regexr.com interface, but with a different regex pattern: `/[0-9]{2}-[0-9]{3}-[0-9]{4}/g`. The sample text is the same. Two matches are highlighted in blue, corresponding to the phone numbers "02-441-0909" and "02-441-0990". The match count is "2 matches (0.3ms)".

2. Understand the Basic Pattern

abc exactly this sequence of three letters

[abc] any *one* of the letters **a**, **b**, or **c**

[^abc] any character *except* one of the letters **a**, **b**, or **c**
(immediately within an open bracket, **^** means “not,”
but anywhere else it just means the character **^**)

[a-z] any *one* character from **a** through **z**, inclusive

[a-zA-Z0-9] any *one* letter or digit

.

any one character except a line terminator

\d

a digit: **[0-9]**

\D

a non-digit: **[^0-9]**

\s

a whitespace character: **[\t\n\x0B\f\r]**

\S

a non-whitespace character: **[^\s]**

\w

a word character: **[a-zA-Z_0-9]**

\W

a non-word character: **[^\w]**

Notice the space.
Spaces are **significant**
in regular expressions!

Useful links:

<https://docs.oracle.com/javase/tutorial/essential/regex/>

<https://mkyong.com/regular-expressions/10-java-regular-expression-examples-you-should-know/>

<https://regexone.com/> (for learning and practicing online)

3. Quantifier

Assume X represents some pattern

$X?$ optional, X occurs once or not at all

X^* X occurs zero or more times

X^+ X occurs one or more times

$X\{n\}$ X occurs exactly n times

$X\{n, \}$ X occurs n or more times

$X\{n, m\}$ X occurs at least n but not more than m times

Note that these are all *postfix* operators, that is, they come *after* the operand

3.1 Types of Quantifiers

- A greedy quantifier will match as much as it can, and back off if it needs to
 - We will do examples in a moment
- A reluctant quantifier will match as little as possible, then take more if it needs to

- You make a quantifier reluctant by appending a ?:

$X??$ $X*?$ $X+?$ $X\{n\}?$ $X\{n,\}?$ $X\{n,m\}?$

- A possessive quantifier will match as much as it can, and never let go

- You make a quantifier possessive by appending a +:

$X?+$ $X*+$ $X++$ $X\{n\}+$ $X\{n,\}+$ $X\{n,m\}+$

3.2 Quantifier Examples

- Suppose your text is `aardvark`
- Using the pattern `a*ardvark` (`a*` is greedy):
 - The `a*` will first match `aa`, but then `ardvark` won't match
 - The `a*` then “backs off” and matches only a single `a`, allowing the rest of the pattern (`ardvark`) to succeed
- Using the pattern `a*?ardvark` (`a*?` is reluctant):
 - The `a*?` will first match zero characters (the null string), but then `ardvark` won't match
 - The `a*?` then extends and matches the first `a`, allowing the rest of the pattern (`ardvark`) to succeed
- Using the pattern `a*+ardvark` (`a*+` is possessive):
 - The `a*+` will match the `aa`, and will not back off, so `ardvark` never matches and the pattern match fails

4. Sequence and Alternative

- If one pattern is followed by another, the two patterns must match consecutively

- For example,

`[A-Za-z]+[0-9]`

will match `one or more letters` immediately followed by `one digit`

- The vertical bar, `|`, is used to separate **alternatives**

- For example,

`abc|xyz`

will match either `abc` **or** `xyz`

Example Patterns

My name is Siripen_Pongpaichet.
My student ID is 6288999
My phone number is 02-441-0909
My line contact is @inging99
My email is siripen.pon@mahidol.ac.th

Information	Observation	Actual RegEx Patter
Name	{one or more letter}_{one or more letter}	[a-zA-Z]+_[a-zA-Z]+
Student ID	{7digits number only}	[0-9]{7}
Phone Number	{2digits}-{3digits}-{4digits}	[0-9]{2}-[0-9]{3}-[0-9]{4}
Line	@{text/number}	@[\w]+
Email	{one or more letter}.{3letter}@{text}.{th com}	[\w]+.[a-zA-Z]{3}@[\w.]+.[th com]

5. RegEx in Java

1. Import library (`import java.util.regex.*;`)
2. Define **RegEx** pattern in String format
3. Create and compile **Pattern** from the RegEx String in step 2
4. Create **Matcher** from the Pattern in step 3
5. *Scan the input sequence and find the subset of text that matches the pattern*

5.1 RegEx Code in Java

1

```
import java.util.regex.*;

public class Part1_BasicRegex {

    public static void main(String[] args) {

        String text = "abcdefgabcd";

        String regex = "abc";

        Pattern p = Pattern.compile(regex);

        Matcher m = p.matcher(text);

        System.out.println("matches the entire string: " + m.matches());

        System.out.println("matches at the beginning of string: " + m.lookAt());

        System.out.println("matches any part of the text string: " + m.find());

    }

}
```

2

```
String regex = "abc";
```

```
Pattern p = Pattern.compile(regex);
```

4

```
Matcher m = p.matcher(text);
```

```
System.out.println("matches the entire string: " + m.matches());
```

```
System.out.println("matches at the beginning of string: " + m.lookAt());
```

```
System.out.println("matches any part of the text string: " + m.find());
```

```
}
```

```
}
```



Text Source

3

Notice that neither Pattern nor Matcher has a public constructor; you create them using methods in the Pattern class.

5

false

true

true

5.2 Methods of Matcher class

- Now that we have a matcher `m`,
- `m.matches()` returns `true` if the pattern matches the *entire text string*, and `false` otherwise
- `m.lookingAt()` returns `true` if the pattern matches at *the beginning of the text string*, and `false` otherwise
- `m.find()` returns `true` if the pattern matches *any part of the text string*, and `false` otherwise
 - If called again, `m.find()` will start searching from where the last match was found
 - `m.find()` will return `true` for as many matches as there are in the string; after that, it will return `false`
 - When `m.find()` returns `false`, matcher `m` will be *reset* to the beginning of the text string (and may be used again)

Let's explore Step 5:

How many times subset of text matches the pattern?

```
String text = "1abc234";
```

```
String regexNum = "\\d+";
```

```
Pattern pNum = Pattern.compile(regexNum);
```

```
Matcher mNum = pNum.matcher(text);
```

```
System.out.println("\n\nFirst group of text found: " + mNum.find());
```

```
System.out.println("Second group of text found: " + mNum.find());
```

```
System.out.println("Third group of txt found: " + mNum.find());
```

1	a	b	c	2	3	4
---	---	---	---	---	---	---

true



true

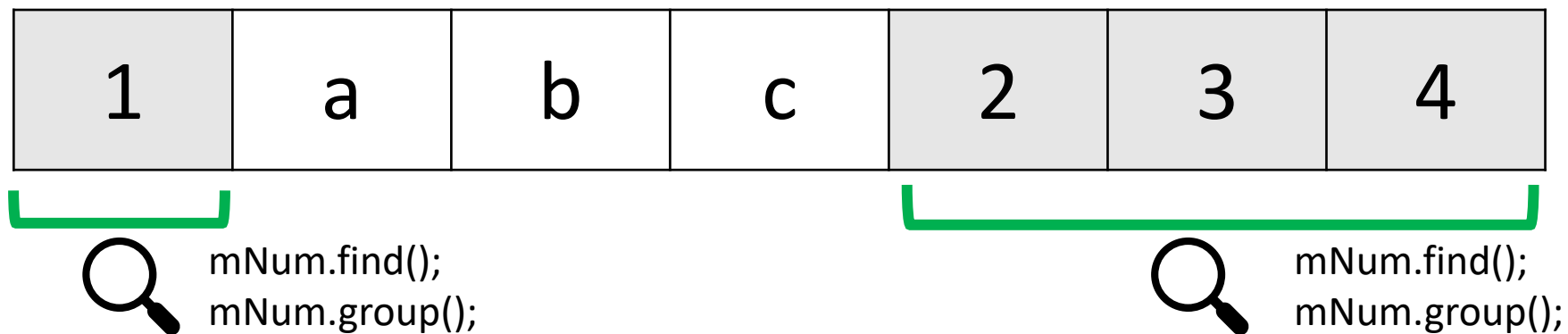


false



Let's explore Step 5:

How to get "all" text that matches the pattern?



```
mNum.reset(); // The matcher's region is set to index 0
```

```
System.out.println("\ntext: " + text + ", regex: " + regexNum);
```

```
while (mNum.find()) {
```

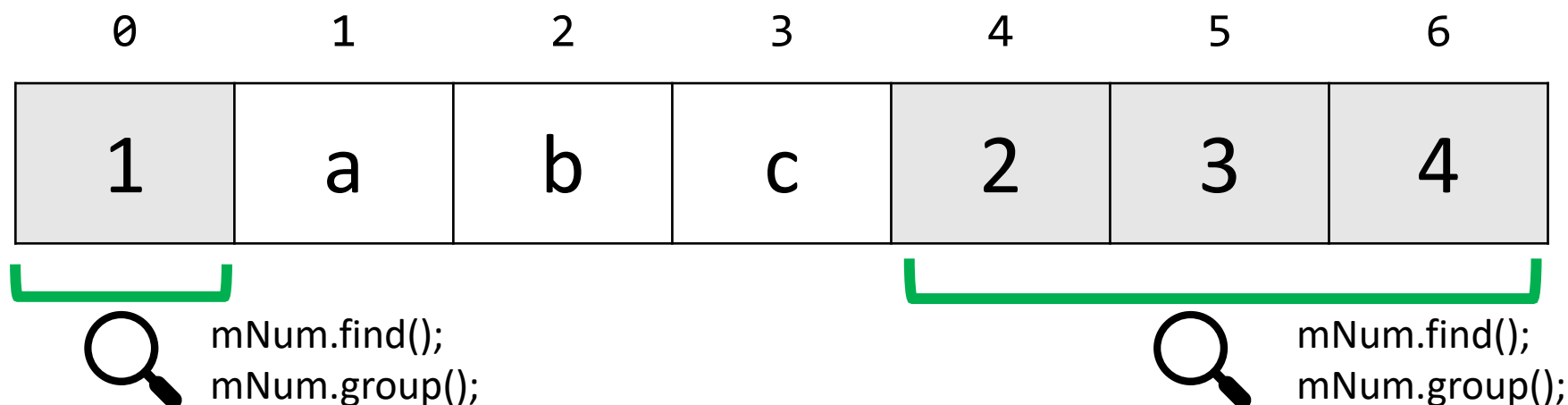
```
    System.out.println("found: " + mNum.group());
```

```
}
```

```
text: 1abc234, regex: \d+  
found: 1  
found: 234
```

Let's explore Step 5:

How to get "all" text that matches the pattern?



```
mNum.reset();  
while(mNum.find()) {  
    String found = text.substring(mNum.start(), mNum.end());  
    System.out.println("[ " + mNum.start() + ", " + mNum.end() + " ] -> " + found) ;  
}
```

[0, 1] -> 1
[4, 7] -> 234

After a successful match, `m.start()` will return the index of the first character matched, and `m.end()` will return the index of the last character matched, *plus one*.

5.3 Additional methods of Matcher class

- If **m** is a matcher, then
- **m.replaceFirst(*replacement*)** returns a new String where the **first substring** matched by the pattern has been replaced by ***replacement***
- **m.replaceAll(*replacement*)** returns a new String where **every substring** matched by the pattern has been replaced by ***replacement***
- **m.find(*startIndex*)** looks for the next pattern match, starting at the specified index
- **m.reset()** resets this matcher
- **m.reset(*newText*)** resets this matcher and gives it new text to examine (which may be a String, StringBuffer, or CharBuffer)

5.4 More Examples

```
String info = "My name is Siripen_Pongpaichet."
+ "My student ID is 6288999\n"
+ "My phone number is 02-441-0909\n"
+ "My line contact is @inging99\n"
+ "My email is siripen.pon@mahidol.ac.th";

String regexName = "[a-zA-Z]+_[a-zA-Z]+";
String regexID = "[0-9]{7}";
String regexPhone = "[0-9]{2}-[0-9]{3}-[0-9]{4}";
String regexLine = "@[\\w]+";
String regexEmail = "[\\w]+.[a-zA-Z]{3}@[\\w.]+.[th|com]";
```

Name: Siripen_Pongpaichet
ID: 6288999
Phone: 02-441-0909
Line: @inging99
Email: siripen.pon@mahidol.ac.th

```
Pattern p = Pattern.compile(regexName);
Matcher m = p.matcher(info);
m.find();
System.out.println("Name: " + m.group());

m = Pattern.compile(regexID).matcher(info);
m.find();
System.out.println("ID: " + m.group());

m = Pattern.compile(regexPhone).matcher(info);
m.find();
System.out.println("Phone: " + m.group());

m = Pattern.compile(regexLine).matcher(info);
m.find();
System.out.println("Phone: " + m.group());

m = Pattern.compile(regexEmail).matcher(info);
m.find();
System.out.println("Phone: " + m.group());
```


Checkpoint

- T F 1) A pattern class is used to compile regular expression?
- T F 2) A Matcher class interprets both patterns and performs match operations in the string
- T F 3) After a successful match, `m.start()` will return the index of the *last* character matched
- T F 4) `\w` regular expression matches nonword character
- T F 5) `[0-9]{3}` regular expression matches three digits
- T F 6) `[L|l]?ove` regular expression matches with LOVE

Just play it for fun! <http://play.inginf.units.it/>



6. Capturing (Groups) RegEx

- In regular expressions, **parentheses** are used for grouping, but they also capture (keep for later use) anything matched by that part of the pattern
 - Example: `([a-zA-Z]*)([0-9]*)` matches any number of letters followed by any number of digits
 - If the match succeeds, `\1` holds the matched letters and `\2` holds the matched digits
 - In addition, `\0` holds everything matched by the entire pattern
- Capturing groups are numbered by counting their opening parentheses from left to right:

<code>(</code>	<code>(</code>	<code>A</code>	<code>)</code>	<code>(</code>	<code>B</code>	<code>(</code>	<code>C</code>	<code>)</code>	<code>)</code>	<code>)</code>
1	2			3		4				

`\0 = \1 = ((A)(B(C)))`, `\2 = (A)`, `\3 = (B(C))`, `\4 = (C)`
- Example: `([a-zA-Z]) \1` will match a double letter, such as letter

Example (Groups) RegEx

Expression

```
/([0-9]+),([A-Z])([a-z]+),(\d{2}-\d{3}-\d{4})/g
```

Text

Tests

NEW

6388999,Siripen,02-441-0909

match: 6388999,Siripen,02-441-0909
range: 0-26

group #1: 6388999
group #2: S
group #3: iripen
group #4: 02-441-0909

```
m.matches() // true  
m.group()    // 6388999,Siripen,02-441-0909  
m.group(1)   // 6388999  
m.group(2)   // S  
m.group(3)   // iripen  
m.group(4)   // 02-441-0909
```

6.1 Groups RegEx: use for validating text format

```
public static void main(String[] args) {  
  
    String info = "6388999,Siripen,02-441-0909";  
  
    String valid = "(\\d{7}),([A-Z][a-z]+),(\\d{2}-\\d{3}-\\d{4})";  
    Pattern p = Pattern.compile(valid);  
    Matcher m = p.matcher(info);  
    System.out.println(m.matches());  
  
    if(m.matches()) {  
        System.out.println("ALL MATCH: " + m.group());  
        System.out.println("ID: " + m.group(1));  
        System.out.println("NAME: " + m.group(1));  
        System.out.println("PHONE:" + m.group(3));  
    }  
  
    String wrongInfo = "6388999,Siripen,";  
    m.reset(wrongInfo);  
    System.out.println(m.matches());  
}
```

Why \\d?

If m is a matcher,

m.group(n) returns the String matched by capturing group n

- This could be an empty String

m.group() is the same as m.group(0)

If m didn't match (or wasn't tried), then these methods will throw an `IllegalStateException`

true

ALL MATCH: 6388999,Siripen,02-441-0909

ID: 6388999

NAME: 6388999

PHONE:02-441-0909

false

7. Double backslashes

- Backslashes (\) have a special meaning in **regular expressions**; for example, \b means a *word boundary*
- The Java compiler treats backslashes specially; for example, \b in a **String** or as a char means the *backspace character*
- **Java syntax rules apply first!**
 - If you write "\bdog\b" you get a string with backspace characters in it --this is not what you want!
 - Remember, you can quote a backslash with another backslash, so "\\bdog\\b" gives the correct string

[e.g., this regex matches with dog but not doggie]

8. Escaping Characters

- A lot of special characters--parentheses, brackets, braces, stars, plus signs, etc.--are used in defining regular expressions; these are called metacharacters
- Suppose you want to search for the character sequence `a*` (an `a` followed by a star)
 - `"a*"`; doesn't work; that means "zero or more `a`s"
 - `"a\"*`"; doesn't work; since a star doesn't need to be escaped (in Java String constants), Java just ignores the `\`
 - `"a\\\"*`" does work; it's the three-character string `a, \, *`
- Just to make things even more difficult, it's illegal to escape a non-metacharacter in a regular expression
- Hence, you can't backslash special characters "just in case"

Example: regex contains |

```
public static void main(String[] args) {  
  
    String info = "6388999|Siripen|02-441-0909";  
  
    String valid = "(\\d{7})|([A-Z][a-z]+)|(\\d{2}-\\d{3}-\\d{4})";  
    Pattern p = Pattern.compile(valid);  
    Matcher m = p.matcher(info);  
    System.out.println(m.matches());  
  
    if(m.matches()) {  
        System.out.println("ALL MATCH: " + m.group());  
        System.out.println("ID: " + m.group(1));  
        System.out.println("NAME: " + m.group(1));  
        System.out.println("PHONE: " + m.group(3));  
    }  
}
```

Be careful!
Vertical Bar (|)
represents "Alternative"
in regex pattern

false

9. Groups RegEx: use for generating a secret code

Pig Latin is a spoken “secret code” that many English-speaking children learn

- There are some minor variations (regional dialects?)

The rules for (written) Pig Latin are:

- If a word begins with a consonant cluster, move it to the end and add “ay”
- If a word begins with a vowel, add “hay” to the end
- Example:

regular expressions are fun! →

egularray expressionshay arehay unfay!

```
Pattern wordPlusStuff = Pattern.compile("([a-zA-Z]+)(^[a-zA-Z]*)");
Pattern consonantsPlusRest = Pattern.compile("([^aeiouAEIOU]+)([a-zA-Z]*)");

public String translate(String text) {
    Matcher m = wordPlusStuff.matcher(text);
    String translatedText = "";
    while (m.find()) {
        translatedText += translateWord(m.group(1)) + m.group(2);
    }
    return translatedText;
}

private String translateWord(String word) {
    Matcher m = consonantsPlusRest.matcher(word);
    if (m.matches()) {
        return m.group(2) + m.group(1) + "ay";
    }
    else
        return word + "hay";
}
```


Final Thought

- Regular Expressions are a language, and not easy to use at first
 - It's a bunch of punctuation, not words
 - The individual pieces are not hard, but it **takes practice** to learn to put them together correctly
 - Regular expressions form a miniature programming language
 - It's a different kind of programming language than Java, and requires you to learn new thought patterns.
- In Java you can't just use regular expression; you have to first create **Patterns** and **Matchers**
 - Java's syntax for String constants doesn't help, either
- Despite all this, regular expressions bring so much **power and convenience** to String manipulation that they are well worth the effort of learning.
- After all, many programming tools enable RegEx functionalities

Consider Alternatives

- Regular expressions are complex
 - They are often used when you cannot guarantee “good” input, so you have to make them fail-safe
- Test thoroughly!
 - Break a complex regular expression into its components, and test each separately
 - Every pattern is a program, and needs to be treated with respect
 - Pay special attention to edge cases
- Consider alternatives
 - Regular expressions are powerful, **but**... If you can get the job done with a few simple String methods, you probably are better off doing it that way