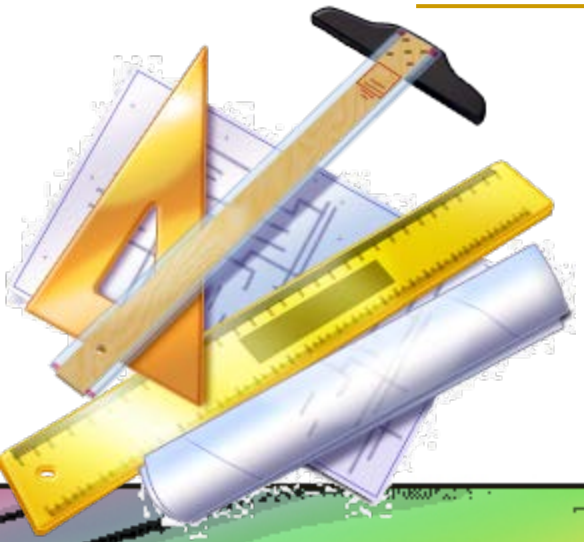
Regular Expression &Text Processing



Lecture 12

Yaowadee Temtanapat

เยาวดี เต็มธนาภัทร์

วัตถุประสงค์ของการเรียนในวันนี้

- ศึกษาแนวคิดเกี่ยวกับ Regular Expression
  - สัญกรณ์ (Notation)
  - แพทเทิร์น (Pattern) ของ Regular Expression เพื่อจับคู่สายอักษร
- ศึกษาการใช้งาน Regular Expression เพื่อการประมวลผลข้อความใน Java

Regular Expression

- **Regular Expression** คือ แพทเทิร์น (Pattern) สำหรับอธิบายรูปแบบของสายอักษร (String, text)
  - ใช้เปรียบเทียบและจับคู่ (Match) ระหว่างแพทเทิร์นที่กำหนดกับสายอักษรใดๆว่า**ตรงกัน**หรือไม่
  - ใช้ในแยกส่วนประกอบของสายอักษร เพื่อ
    - วิเคราะห์หรือค้นหาสายอักษรตามแพทเทิร์นในสายอักษรหนึ่งๆ
    - แก้ไขหรือสร้างสายอักษร ให้มีแพทเทิร์นตามที่ต้องการ
- อาจเรียกย่อๆว่า **regex** (reg – ex)
- ประกอบด้วย
  - Symbol – สัญลักษณ์เพื่อใช้สร้าง Pattern
  - Metacharacters – กำหนดเป็นตัวแทนของอักษรประเภทต่างๆ
  - Quantifier – สัญลักษณ์บอกจำนวน
- ตัวอย่าง – [a-z]+ - จะตรงกับตัวสายอักษรตั้งแต่ 1 ตัวขึ้นไป เช่น aaa, abc

Matching Symbols

- **.** – แทนตัวอักษรใดๆ 1 ตัวอักษร (ที่ไม่ใช่ \n)
- **^** - แทนจุดเริ่มต้นบรรทัด
  - เช่น ^ab จะตรงกับ ab ใน abc ได้แต่ไม่ตรงกับส่วน ab ใน aab
- **$** - แทนจุดสิ้นสุดบรรทัด
  - เช่น ab$ จะตรงกับ ab ใน aab ได้แต่ไม่ตรงกับ ab ใน abc
- **ab** – ตรงกับตัวอักษร ab
- **[abc]** – ตรงกับอักษร a b หรือ c
- **[^abc]** – ตรงกับอักษรใดๆหนึ่งตัวที่ไม่ใช่ a, b หรือ c (^ใน [หมายถึง “not”)

Matching Symbols (ต่อ)

- **[a-d]** – ตรงกับตัวอักษรหนึ่งตัวอักษรตั้งแต่ a ถึง d
- **[a-zA-Z0-9]** – ตรงกับตัวอักษรหรือตัวเลขใดๆหนึ่งตัว
- **[abc][xy]** – ตรงกับอักษร a หรือ b หรือ c ตามด้วย x หรือ y
- **ab|xy** – ตรงกับชุดอักษร ab หรือ xy
- สังเกตว่า
  - ถ้ามีแพทเทิร์นต่อกัน สายอักษรที่นำมา match จะต้องตรงกับแพทเทิร์นที่ต่อกันตามลำดับ
  - สัญลักษณ์ | ใช้ในการแยกแพทเทิร์นที่เป็นตัวเลือก

Metacharacters

- Metacharacters เป็นสัญลักษณ์ที่ถูกกำหนดมาให้แทนอักษรประเภทต่างๆเพื่อความง่ายในการเขียนแพทเทิร์น เช่น
- **\d** – ตัวเลขใดๆ : [0-9]
- **\D** – อักษรใดๆที่ไม่ใช่ตัวเลข : [^0-9]
- **\s** – ตัวอักษร whitespace : [\t\n\x0b\r\f]
- **\S** – ตัวอักษรที่ไม่ใช่ whitespace : [^\s]
- **\w** – อักษรที่เป็น Word character : [a-zA-Z_0-9]
- **\W** – อักษรที่ไม่ใช่ word character : [^\w]
- **\b** – ขอบ (Boundary) ของคำ เช่น \bis\b ตรงกับ is แต่ไม่ตรงกับ is ใน island
- **\B** – ไม่ใช่ขอบของคำ

สังเกตว่าช่องว่าง (space) ถือเป็นตัวอักษรใน regex

Quantifiers

- เป็นสัญลักษณ์เพื่อช่วยกำหนดจำนวนครั้งที่เกิดขึ้นของแพทเทิร์น
- X* – เกิด X ตั้งแต่ 0 ครั้งขึ้นไป : {0,}
- X+ – เกิด X ตั้งแต่ 1 ครั้งขึ้นไป : {1,}
- X? – ไม่เกิด X เลยหรือเกิด 1 ครั้ง : {0,1}
- X{n} – เกิด X ขึ้นจำนวน n ครั้ง
- X{n,m} – เกิด X ขึ้นตั้งแต่ n จนถึง m ครั้ง

Examples - 1

Pattern	Example Matches
[Mm]ary	Mary, mary
cat|dog|pig	cat, dog, pig
[a-zA-Z]\d\d	a23, S59, c30
\d{2}-?\d{4}-?\d{4}| \d{2}-?\d{3}-?\d{4}	0834523490, 08-3435-1890, 025969120, 02-4561234, 93-234-5213
c.*t	cat, ct, cart, count
0[1-7]*	0, 01, 02, 013, 0753
a?b+	ab, bbbbb, abb

Examples - 2

- **User Name** – ตัวอักษรหรือตัวเลข มี _ ได้ ตั้งแต่ 6-10 ตัว
  - ^[a-zA-Z0-9_]{6,10}$
- **เลขฐาน 16** ในภาษา Java ขึ้นด้วย 0x หรือ 0X ตามด้วยตัวเลขอย่างน้อย 1 ตัว
  - ^0[xX][0-9a-f]+
- **เลขฐาน 10** ในภาษาจาวา – ห้ามขึ้นด้วย 0 ยกเว้นค่า 0
  - 0|[^0][0-9]*

Quiz

- ถ้าให้ Text **bbbxybz**
- ผลของการจับคู่กับ **b*bxybz** คือ ?
  - **ตรง** - b* จับ bbb แล้วถอยกลับ ไปจับแค่ bb เพื่อให้ตรงแพทเทิร์น
- ผลของการจับคู่กับ **b*?bxybz** คือ ?
  - **ตรง** – b*? ค่อยๆจับ b เพิ่มทีละตัวจนเหลือ bxybz ให้ตรงแพทเทิร์นที่เหลือ
- ผลของการจับคู่กับ **b*+bxybz** คือ ?
  - **ไม่ตรง** - เนื่องจาก b*+ จับ b ไปทั้งสามตัว bxybz ที่เหลือไม่สามารถจับคู่อักษรที่เหลือได้

การใช้ \ เพื่อ escape

- เนื่องจากบางตัวอักษรที่ใช้ใน regex มีความหมายพิเศษ ใน Java
  - เช่น \b ใน Java คือ backspace แต่ใน regex \b คือ ขอบของคำ
- ในการใช้ String literal ต้องเติม \ เพื่อ escape ให้ Java เห็น \ ใน \b เป็นเพียง \
  - เช่นถ้าเขียน “\b[a-z]” จะได้สายอักษรที่ขึ้นด้วยตัว backspace
  - ต้องเขียนเป็น “\\b[a-z]” จึงจะตรงกับตัวหนังสือใดๆ a-z ขึ้นต้นคำ
- อาจใช้ \\ ในการ escape สัญลักษณ์พิเศษอื่นๆ ของ regex เช่น
  - . หมายถึงตัวอักษรใดๆ ถ้าอยากได้ . ต้องใช่ \\.
  - ตัวอักษรอื่นๆ เช่น [ ] { } ( ) * + - ? ^ $ |

String Methods ที่ใช้ regex

- boolean match(String regex)
- String replaceAll(String regex, String replacement)
- String replaceFirst(String regex, String replacement)
- String[] split(String regex)
- String[] split(String regex, int limit)
  - ถ้า limit มากกว่า 0 จะจับคู่ regex ให้ limit -1 ครั้ง ถ้า limit <= 0 จะจับคู่ให้มากครั้งที่สุดเท่าที่จะทำได้

Matching String Example

```
public class StringTester {
public static void main(String[] args) {
        String [] input = {"173-23-2342",
                           "1113232222",
                           "456679087",
                           "45667-9087",
                           "1245-23-2354",
                           "abs-23-s3fg",
                           "123-32-4a23",
                           "2123-25-233523"};
        String pattern1 = "\\d{3}-?\\d{2}-[a-z0-9]{4}";

        System.out.println("Testing Pattern:  " + pattern1);
        for (String s : input){
           if (s.matches(pattern1))
              System.out.println ( "\t" + s + " matches the pattern");
        }

    }
}
```

```
Testing Pattern:  \d{3}-?\d{2}-[a-z0-9]{4}
        173-23-2342 matches the pattern
        45667-9087 matches the pattern
        123-32-4a23 matches the pattern
```

Replacing String Example

```
public class StringReplacer {

    public static void main(String[] args) {
        String names = "Obama Aladin Shisuka";
        String patterns = "^a|i";
        String newNames = names.replaceFirst(patterns, "E");
        System.out.println("After replace first with E:" + newNames);
        newNames = names.replaceAll(patterns, "E");
        System.out.println("After replace All with E:" + newNames);

    }
}
```

```
After replace first with E:Obama AladEn Shisuka
After replace All with E:Obama AladEn ShEsuka
```

Splitting Example

```
public class Spliter {
public static void main(String[] args) {
        String tester = "53;42 ;78  ;40;  9";
        String pattern = "\\s*;\\s*";
        String [] split1 = tester.split(pattern);
        String [] split2 = tester.split(pattern, 3);
        String [] split3 = tester.split(pattern, 0);
        System.out.println("Split with no limit");
        printArray(split1);
        System.out.println("Split with limit = 3");
        printArray(split2);
        System.out.println("Split with limit = 0");
        printArray(split3);

    }
    private static void printArray(String[] strings){
        for (String s:strings){
            System.out.println("\t " + s);
        }
        System.out.println("--------------------");
    }

}
```

Util.regex

- Java มีคลาสเพื่อทำงานกับ regex โดยเฉพาะ
  - Pattern และ Matcher จาก java.util.regex
- Pattern ใช้ในการคอมไพล์แพทเทิร์นที่เขียนด้วย regular expression - ไม่สามารถ new ได้ ต้องสร้างวัตถุจากเมท็อด compile

```
Pattern p = Pattern.compile("[a-z]+");
```

- Matcher ใช้ในการจับคู่สายอักษรที่เป็นข้อมูลเข้ากับวัตถุ Pattern ที่คอมไพล์แล้ว
  - ไม่สามารถ new แต่สร้างได้เมื่อเรียก matcher เมท็อดของวัตถุ Pattern

```
Matcher m = p.matcher("Now is the time");
```

เมท็อดของ Matcher

- matches() – คืนค่า true ถ้าแพทเทิร์นตรงกับทั้ง Sting ที่เป็น input
- lookingAt() – คืนค่า true ถ้าแพทเทิร์น ตรงกับส่วนเริ่มของ String ที่เป็น input
- find() – คืนค่า true ถ้าแพทเทิร์น ตรงกับส่วนใดๆของ String ที่เป็น input
  - ถ้าเรียกซ้ำไปเรื่อยๆ find() จะเริ่มหาต่อจากจุดที่เจอส่วนที่ตรงครั้งสุดท้าย ไปเรื่อยๆจนกว่าจะหมด จึงคืนค่า false
  - เมื่อคืนค่า false matcher จะล้างค่าตำแหน่งการค้นหา (reset) กลับไปที่จุดเริ่มต้นของข้องมูลเข้า
- start() – คืน index ตัวอักษรแรกของส่วนที่ตรงกับแพทเทิร์นที่เพิ่งเจอ
- end() – คืน index ตัวอักษรสุดท้าย + 1 ของส่วนที่ตรงกับแพทเทิร์นที่เพิ่งเจอ
- ถ้าเรียกใช้ start หรือ end ก่อนค้นหา หรือหาไม่เจอจะ start และ end จะ throw IllegalStateException

Example

```
public class RegexTest {
    public static void main(String args[]) {
        String pattern = "[a-z]+";
        String text = " I love CS111 very much ";
        Pattern p = Pattern.compile(pattern);
        Matcher m = p.matcher(text);
        while (m.find()) {
            System.out.println("Found :" + text.substring(m.start(), m.end()) );
        }
    }
}
```

```
Found :love
Found :very
Found :much
```

Example 2

```
public class RegexTest2 {
    public static void main(String args[]) {
        String pattern = ".*foo";
        String text = "xfooxxxxxxfoo";
        Pattern p = Pattern.compile(pattern);
        Matcher m = p.matcher(text);
        String newText = "";
        while (m.find()) {
            System.out.println("Found :" + text.substring(m.start(), m.end()) );
        }
    }
}
```

```
Found :xfooxxxxxxfoo
```

สังเกตว่าการจับคู่ .* จะพยายามจับคู่กับตัวอักษรให้ได้มากที่สุดก่อนแล้วค่อยถอยมาเพื่อให้ได้ตรงตามแพทเทิร์น

Capturing Groups

- เราสามารถใช้วงเล็บเพื่อแยกกลุ่มของ regular expression ได้ โดยที่ข้อมูลที่ตรงของแต่ละกลุ่มในแพทเทิร์นจะถูกเก็บแยกไว้ – Capturing Group
- เช่น ([a-zA-Z]*)([0-9]*) จะตรงกับตัวหนังสือกี่ตัวก็ได้ตามด้วยตัวเลขกี่ตัวก็ได้
  - มีสองกลุ่ม กลุ่ม 1 คือชุดตัวหนังสือ กลุ่มที่ 2 คือตัวเลข
  - กลุ่มพิเศษกลุ่มที่ 0 เก็บชุดตัวหนังสือที่ตรงทั้งแพทเทิร์น
  - เช่น letter05 – group1= letter, group2 = 05, group0 = letter05
- การนับกลุ่มนับจากจำนวนของวงเล็บเปิดจากซ้ายไปขวา
  - ((A) (B (C)))

    1 2 3 4

    group0 = group1 = ((A)(B(C))), group2 = (A), group3 = (B(C)), group4 = (C)

เมท็อด group ของ Matcher

- ใน Java ใช้เมท็อด group ของ Matcher เพื่อเข้าถึงข้อมูล capturing group
  - ทำหลังจากการ match หรือ find สำเร็จ
  - group(n) อ้างถึง capturing group ที่ n ตามกฎการนับ capturing group
  - อาจคืน null ถ้าแพทเทิร์นตรงแต่ capturing group ไม่ตรง
- ถ้าไม่มี match จะ throw IllegalStateException

Example – Name Formatter

```
public class RegexTest {
   public static void main(String args[]) {
      String pattern = " *([A-Z][a-z]+) *([A-Z][a-z]+) *;";
      String text = "John Smith; Eearl Gray  ; Harry Potter James Harriet; tim Cook; ";
      Pattern p = Pattern.compile(pattern);
      Matcher m = p.matcher(text);
      String newText = "";
      while (m.find()) {
         System.out.println("Found :" + text.substring(m.start(), m.end()) );
         newText = newText + m.group(2) + "," + m.group(1)+":";
      }
      System.out.println("New Text:" + newText);
   }
}
```

ตรวจ format ข้อมูลและ
กลับชื่อกับนามสกุล

```
Found :John Smith;
Found : Eearl Gray  ;
Found : James Harriet;
New Text:Smith,John:Gray,Eearl:Harriet,James:
```