

INTRODUCTION TO NETWORK PROGRAMMING

030513362 - Network programming (teach by : DCK)
Credit : Assoc. Prof. Dr. Choopan Rattanapoka

หัวข้อเรียน

- Introduction to Network Programming
- Overview Java
- Java I/O
- Thread
- Looking up IP address and Socket
- Client Applications (Web)
- ServerSocket
- Client/Server Applications (File Transfer)
- UDP

การประเมินผล (DCK)

<input type="checkbox"/> เวลาเข้าเรียน	0 คะแนน
<input type="checkbox"/> การบ้าน + งานในห้อง	5+5 คะแนน
<input type="checkbox"/> สอบปฏิบัติ 3 ครั้ง รวม <input checked="" type="checkbox"/> ครั้งละ	30 คะแนน
<input type="checkbox"/> สอบกลางภาค (midterm)	20 คะแนน
<input type="checkbox"/> สอบปลายภาค (final)	30 คะแนน

A	≥ 80
B+	$\geq ???$
B	$\geq ???$
C+	$\geq ???$
C	$\geq ???$
D+	$\geq ???$
D	≥ 35

หนังสือหลักที่ใช้

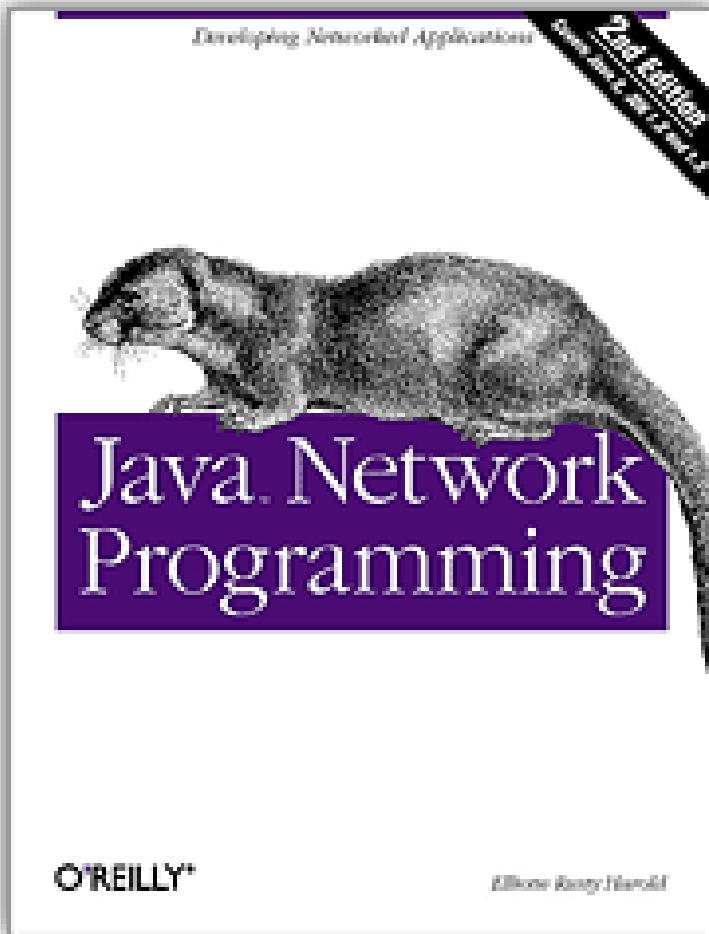
การโปรแกรมระบบเครือข่าย
Network Programming

```
import java.net.*;  
import java.io.*;  
public class ServerSocketExample {  
    public static void main(String args[]){  
        try {  
            ServerSocket server = new ServerSocket(8080);  
            System.out.println("Phone service started");  
            while(true){  
                Socket client = server.accept();  
                System.out.println("Client connected " + client.getInetAddress().getHostAddress());  
                BufferedReader reader = new BufferedReader(new InputStreamReader(  
                    client.getInputStream()));  
                PrintWriter writer = new PrintWriter(new OutputStreamWriter(client.  
                    getOutputStream()));  
                String msg = reader.readLine();  
                if(msg.equals("Hello")){  
                    writer.println("Hello");  
                    writer.flush();  
                }  
                else{  
                    Dimension d = Dimension.getPreferredSize();  
                }  
            }  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

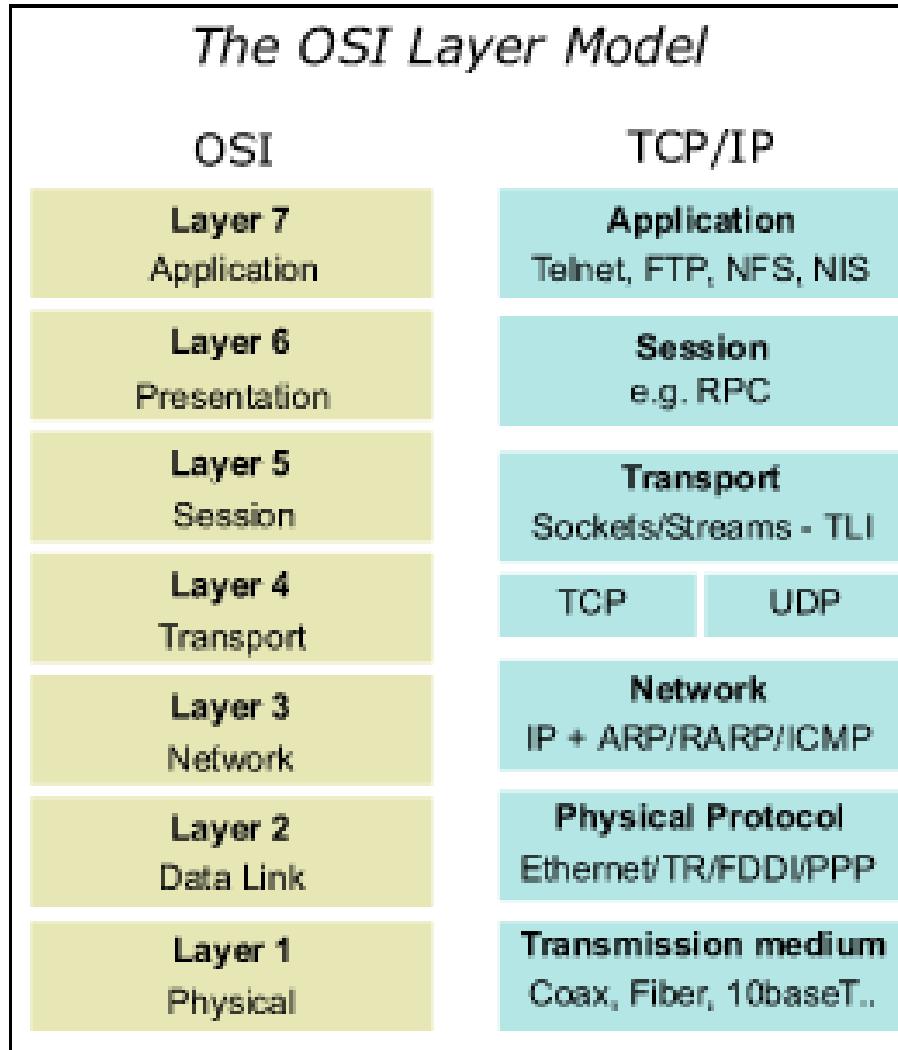
```
public void readData() {  
    try {  
        BufferedReader reader = new BufferedReader(new InputStreamReader(  
            System.in));  
        String str = reader.readLine();  
        System.out.println("User input : " + str);  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}  
public static void main(String args[]){  
    try {  
        readData();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

ผู้ช่วยศาสตราจารย์ ดร.ชูพันธุ์ รัตนโกค้า
ภาควิชาเก็คโนโลยีวิศวกรรมอิเล็กทรอนิกส์
วิทยาลัยเก็คโนโลยีอุตสาหกรรม
มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

หนังสือที่แนะนำ



OSI layer model



ทำไมต้องใช้ Java !!

- Java เป็นภาษาที่ถูกออกแบบมาดั้งเดิมเพื่อการใช้งานกับระบบเครือข่ายซึ่งมีจุดเริ่มต้นมาจาก Java Applet
- Java ถือว่าเป็นภาษาที่ใช้ในการเขียน **network application** ได้มากกว่าภาษาอื่นๆ เช่น ภาษา C
- Java มีคุณสมบัติ “**write-once run anywhere**” ทำให้สะดวกในการเขียนโปรแกรมเพื่อใช้งานกับเครื่องที่มีระบบปฏิบัติการต่างกัน

อะไรคือ โปรแกรมระบบเครือข่าย

- ดึงข้อมูลและแสดง
 - เป็นการทำงานพื้นฐานของโปรแกรมระบบเครือข่าย คือ ดึงข้อมูลจากเครื่องคอมพิวเตอร์อื่นๆ ในระบบเครือข่าย เพื่อนำมาแสดงผล เช่น **web browser**
- ดึงข้อมูลแบบทำซ้ำ
 - ตั้งระยะเวลาในดึงข้อมูล เช่น โปรแกรมซื้อขายหุ้น
- รับ / ส่งข้อมูล
 - โปรแกรม **download/upload** แฟ้มข้อมูล
 - โปรแกรม **Massively parallel computing**
 - เกมออนไลน์, โปรแกรม **chat**, โปรแกรมเชิฟเวอร์ต่างๆ

อุ่นเครื่อง Java (1) : ผลการรัน

```
import java.io.*;

public class Ex1 {
    public static void main(String[] args) {
        int sum = 0;

        for(int i = 0; i < 5; i++)
            sum += i;

        System.out.println("Sum = " + sum);
    }
}
```

อุ่นเครื่อง Java (2): ハウผลการรัน

```
public class Ex2 {  
    int x;  
    int y;  
  
    public Ex2() {  
        this.x = 0;  
        this.y = 0;  
    }  
  
    public Ex2(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public int getSum() {  
        return x + y;  
    }  
  
    public static void main(String[] args) {  
        Ex2 a = new Ex2();  
        Ex2 b = new Ex2(3,5);  
        System.out.println("Result = " + (a.getSum() + b.getSum()));  
    }  
}
```

อุ่นเครื่อง Java (3): ハウผลการรัน

```
public class EP {  
    int a, b;  
  
    public EP(int a, int b) {  
        this.a = a;  
        this.b = b;  
    }  
  
    public int funcP() {  
        return (a+b)*2;  
    }  
  
    public int func1() {  
        return a+b;  
    }  
}
```

```
public class Ex3 extends EP {  
    int a, b;  
  
    public Ex3(int a, int b) {  
        super(a,b);  
    }  
  
    public int func1() {  
        return a - b;  
    }  
  
    public int func2() {  
        return a * b;  
    }  
  
    public static void main(String[] args) {  
        Ex3 a = new Ex3(5,4);  
  
        System.out.println("FuncP = " + a.funcP());  
        System.out.println("Func1 = " + a.func1());  
        System.out.println("Func2 = " + a.func2());  
    }  
}
```

อุ่นเครื่อง Network

- IP address ของ localhost ?
- ถ้าต้องการทราบว่าเครื่อง ect.cit.kmutnb.ac.th มี หมายเลข IP อะไร
ต้องใช้บริการ (network service) อะไร ?
- หมายเลข port มาตรฐานของ HTTP ?
- TCP กับ UDP ต่างกันอย่างไร ?
 - ขนาดของ Header ?
 - ขนาดของข้อมูลที่ส่งได้ต่างกันใหม่ ?

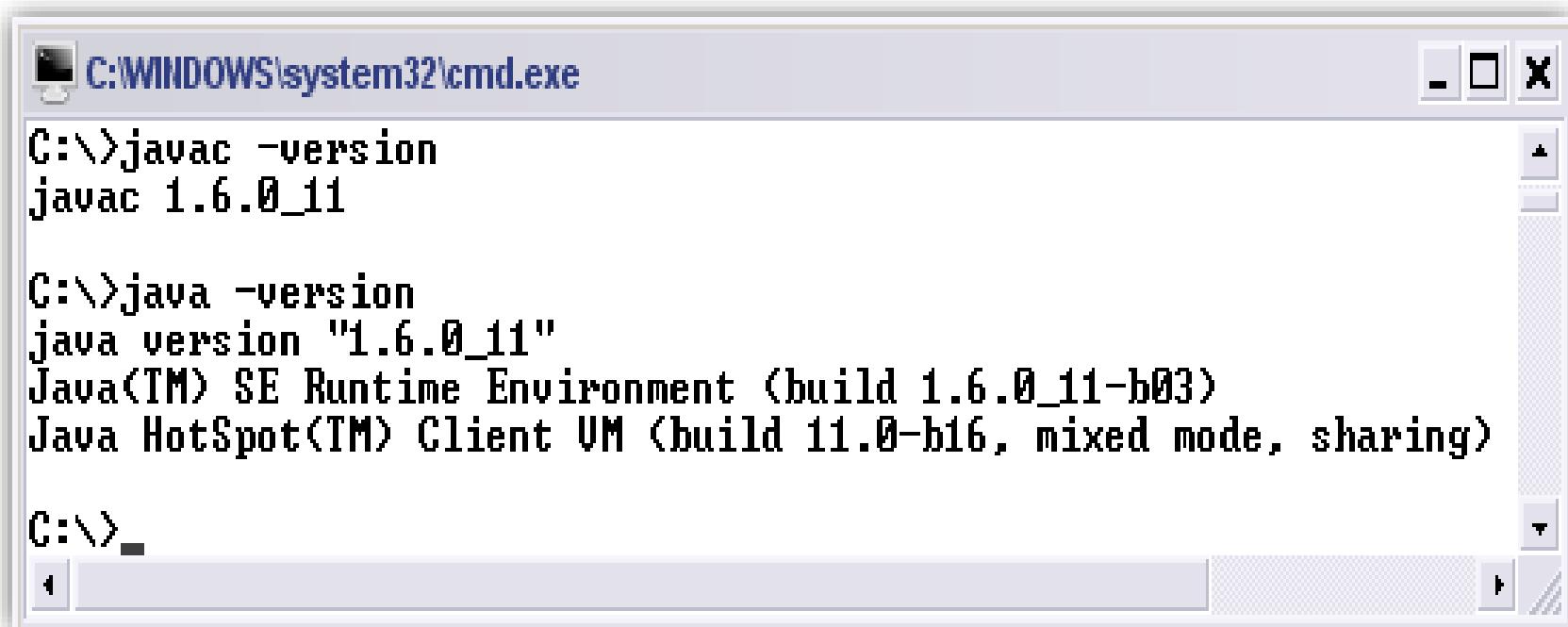
OVERVIEW JAVA

030523313 - Network programming
Asst. Prof. Dr. Choopan Rattanapoka

Introduction

- Java จะคู่กับ motto “write once run anywhere”
- การพัฒนาโปรแกรมด้วยภาษา Java จะต้องติดตั้ง JDK ซึ่งจะประกอบด้วย
 - **javac** (java compiler) เป็นตัว compiler ที่แปลง source code (.java) ให้เป็น java bytecode (.class)
 - **java bytecode** เปรียบเสมือน executable file ที่สามารถนำไปใช้งานได้กับทุก OS ที่ support java
 - **java** เป็นตัว interpreter ที่จะอ่าน java bytecode เป็นภาษาเครื่อง

ตรวจสอบความพร้อมของเครื่องที่ใช้พัฒนา Java



C:\>javac -version
javac 1.6.0_11

C:\>java -version
java version "1.6.0_11"
Java(TM) SE Runtime Environment (build 1.6.0_11-b03)
Java HotSpot(TM) Client VM (build 11.0-b16, mixed mode, sharing)

C:\>

โครงสร้างภาษา Java

```
1 import java.net.*;
2 import java.io.*;
3
4 public class MyClass
5 {
6     public static void main(String[] args)
7     {
8         System.out.println("Hello World!");
9     }
10 }
```

- บรรทัด 1 – 2 : ส่วนของ **import** สำหรับเรียกใช้ **class** ที่มีอยู่ใน **package** นั้นๆ
- บรรทัด 4 : การประกาศชื่อ **class** จะต้องมีชื่อเหมือนกับชื่อ **file** ดังนั้น **file** นี้ต้องชื่อว่า **MyClass.java**
- บรรทัด 6 – 9 : เป็นส่วน **main** ของ **java** โดย **java** จะเริ่มทำงานที่บรรทัดนี้

Java Package ที่จำเป็น

- ในการเขียนโปรแกรมภาษา Java ให้ทำงานเกี่ยวกับ **network** จะต้อง **import 2 packages** ที่จำเป็นคือ
 - **java.io** (บรรจุ Class ที่ทำงานเกี่ยวกับ **input/output**)
 - **java.net** (บรรจุ Class เกี่ยวกับการทำงานกับระบบเครือข่าย)
- ดังนั้นควรจะมี 2 บรรทัดนี้ขึ้นต้นใน **source code**

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

Java API

- เนื่องจาก **java** มี **class** และ **method** ใช้ให้อย่างมากมาย ทำให้มี
สามารถจำได้หมด
- ในการพัฒนาโปรแกรมด้วยภาษา **Java** นั้นควรดู **API** จากเว็บ

<https://docs.oracle.com/javase/8/docs/api/>
(สำหรับ **java version 1.8.X**) ควบคู่ไปกับการพัฒนาโปรแกรม

Java Command Line

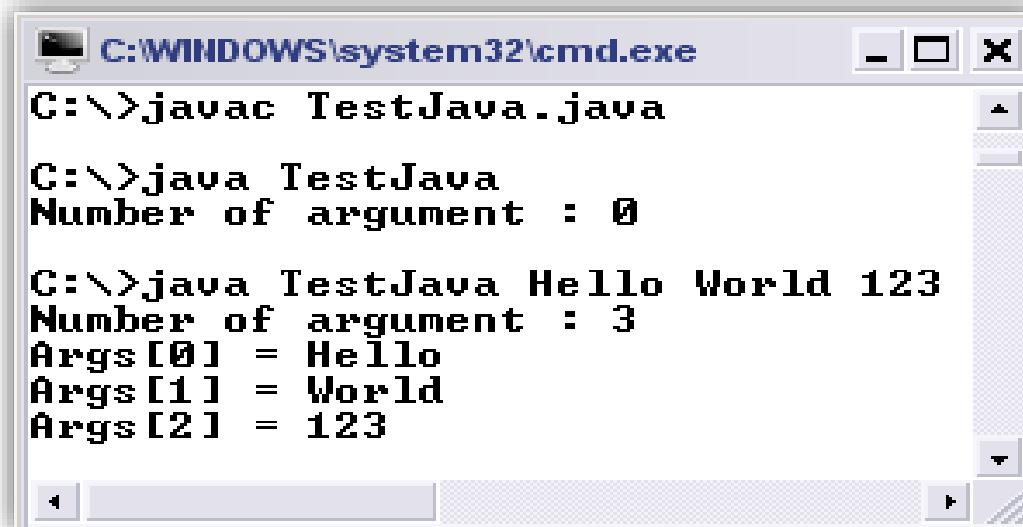
- ในโปรแกรมภาษา Java จะเหมือนกับภาษา C คือจะมีการทำงานเริ่มต้นที่ **function main**
- **function main** จะต้องอยู่ในรูปแบบต่อไปนี้เท่านั้น

public static void main(String[] args)

- โดย **args** ที่เป็น **parameter** ของ **function main** จะเป็นตัวรับค่า **arguments** จาก **command line** ซึ่งจะอยู่ในรูป **array** ของ **string**

Arguments

```
1 public class TestJava
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Number of argument : " + args.length);
6         for(int i = 0; i < args.length; i++)
7             System.out.println("Args[" + i + "] = " + args[i]);
8     }
9 }
```



Type Conversion

- เนื่องจากการรับ **argument** จาก **command line** นั้นจะอยู่ในรูป **String**
- ดังนั้นถ้าอยากรับค่าที่เป็นตัวเลขเพื่อใช้ในการคำนวณจะต้องแปลง **String** เป็น **data type** ที่เป็นตัวเลข สามารถเรียกใช้จาก **static class** ต่างๆ ต่อไปนี้ เช่น
 - **Integer.parseInt(String intValue)**
 - **Float.parseFloat(String floatValue)**
 - **Double.parseDouble(String doubleValue)**

ตัวอย่าง: Type Conversion

```
import java.io.*;

public class TypeConversion {
    public static void main(String[] args) {
        String num1 = "1";
        int num2 = 2;
        System.out.println("Result1 = " + (num1 + num2));
        System.out.println("Result2 = " + (Integer.parseInt(num1)+num2));
    }
}
```

แบบฝึกหัด

```
import java.io.*;

public class Exo1 {
    public static void main(String[] args) {
        if(args.length != 2) {
            System.out.println("Please enter 2 arguments");
            System.exit(1);
        }
        int num1 = Integer.parseInt(args[0]);
        int num2 = Integer.parseInt(args[1]);
        System.out.println("Result = " + (num1 + num2));
    }
}
```

- จงหาผลการรัน ถ้าผู้ใช้เรียกใช้งานด้วยคำสั่ง

- `java Exo1`
 - `java Exo1 125`
 - `java Exo1 25 15`
 - `java Exo1 25 a`

Java try-catch

- ในภาษา Java เมื่อมีการเรียกใช้งาน **method** และเกิดข้อผิดพลาดขึ้น (**error**) จะมีเหตุการณ์ที่เรียกว่า **Exception** เกิดขึ้น
- ใน **Class** มาตรฐานของ Java แต่ละ **method** จะมีการโยน (**throw**) **Exception** ออกจาก **method** เพื่อให้ผู้เรียกใช้ **method** สามารถจัดการเอง
- การเขียนโปรแกรมเพื่อดัก **Exception** ที่เกิดขึ้นใน **method** จะใช้คำสั่ง **try** ตามด้วย { } และให้นำชุดคำสั่งที่ต้องการจะดักจับ **Exception** ไว้ข้างใน { }
- เมื่อมี **Exception** เกิดขึ้นโปรแกรมจะกระโดดไปยังส่วนของ **catch**

ตัวอย่าง Class IOException

- [java.lang.Exception](#)
 - [java.io.IOException](#)
 - [java.io.CharConversionException](#)
 - [java.io.EOFException](#)
 - [java.io.FileNotFoundException](#)
 - [java.io.InterruptedIOException](#)
 - [java.io.ObjectStreamException](#)
 - [java.io.InvalidClassException](#)
 - [java.io.InvalidObjectException](#)
 - [java.io.NotActiveException](#)
 - [java.io.NotSerializableException](#)
 - [java.io.OptionalDataException](#)
 - [java.io.StreamCorruptedException](#)
 - [java.io.WriteAbortedException](#)
 - [java.io.SyncFailedException](#)
 - [java.io.UnsupportedEncodingException](#)
 - [java.io.UTFDataFormatException](#)

ตัวอย่าง Class Exception

- class [java.lang.Exception](#)
 - class [java.lang.ClassNotFoundException](#)
 - class [java.lang.CloneNotSupportedException](#)
 - class [java.lang.IllegalAccessException](#)
 - class [java.langInstantiationException](#)
 - class [java.lang.InterruptedException](#)
 - class [java.lang.NoSuchFieldException](#)
 - class [java.lang.NoSuchMethodException](#)
 - class [java.lang.RuntimeException](#)
 - class [java.lang.ArithmaticException](#)
 - class [java.lang.ArrayStoreException](#)
 - class [java.lang.ClassCastException](#)
 - class [java.lang.IllegalArgumentException](#)
 - class [java.lang.IllegalThreadStateException](#)
 - class [java.lang.NumberFormatException](#)
 - class [java.lang.IllegalMonitorStateException](#)
 - class [java.lang.IllegalStateException](#)
 - class [java.lang.IndexOutOfBoundsException](#)
 - class [java.lang.ArrayIndexOutOfBoundsException](#)
 - class [java.lang.StringIndexOutOfBoundsException](#)
 - class [java.lang.NegativeArraySizeException](#)
 - class [java.lang.NullPointerException](#)
 - class [java.lang.SecurityException](#)
 - class [java.lang.UnsupportedOperationException](#)

ตัวอย่างการดู API ใน Java

□ Class Integer

- **public static int parseInt(String s) throws NumberFormatException**

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign '-' ('\u002D') to indicate a negative value.

- **Parameters:**s - a String containing the int representation to be parsed
- **Returns:**the integer value represented by the argument in decimal.
- **Throws:** NumberFormatException - if the string does not contain a parsable integer.

ตัวอย่างโปรแกรมที่ไม่มีการดัก Exception

```
import java.io.*;

public class Exo1 {
    public static void main(String[] args) {
        int num = Integer.parseInt(args[0]);
        System.out.println("Result = " + num);

    }
}
```

- คิดว่าผลการรันจะเป็นเช่นไรถ้าผู้ใช้เรียกใช้งานด้วยคำสั่ง
 - **java Exo1**

แก้ปัญหา `ArrayIndexOutOfBoundsException`

```
import java.io.*;

public class Exo1 {
    public static void main(String[] args) {
        try {
            int num = Integer.parseInt(args[0]);
            System.out.println("Result = " + num);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Usage : java Exo1 <number>");
        }
    }
}
```

- คิดว่าผลการรันจะเป็นเช่นไรถ้าผู้ใช้เรียกใช้งานด้วยคำสั่ง
 - `java Exo1 Hello`

แก้ปัญหา NumberFormatException

```
import java.io.*;

public class Ex01 {
    public static void main(String[] args) {
        try {
            int num = Integer.parseInt(args[0]);
            System.out.println("Result = " + num);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Usage : java Ex01 <number>");
        } catch (NumberFormatException ee) {
            System.out.println("Usage : java Ex01 <number>");
        }
    }
}
```

วิธีเขียนดักที่ Class Exception

- เนื่องจาก Class Exception เป็น Class แม่ของ
 - Class NumberFormatException
 - Class ArrayIndexOutOfBoundsException
- ดังนั้นการดักที่ Class แม่อย่างเดียวจะทำให้ดักได้หมด

```
import java.io.*;

public class Ex01 {
    public static void main(String[] args) {
        try {
            int num = Integer.parseInt(args[0]);
            System.out.println("Result = " + num);
        } catch (Exception e) {
            System.out.println("Usage : java Ex01 <number>");
        }
    }
}
```

JAVA I/O

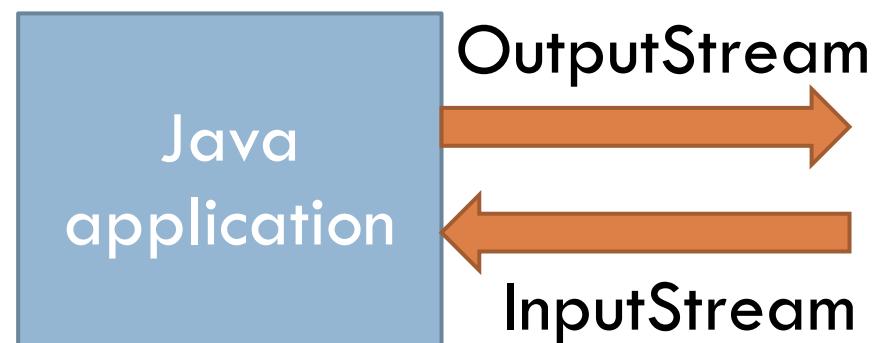
030523313 - Network programming
Asst. Prof. Dr. Choopan Rattanapoka

Introduction

- โปรแกรมระบบเครือข่ายการทำงานส่วนใหญ่จะแบ่งออกเป็น 2 ส่วน
 - การส่งข้อมูล
 - การรับข้อมูล
- การรับ / ส่งข้อมูลข้ามเครื่องคอมพิวเตอร์ผ่านระบบเครือข่ายนั้น มีลักษณะการรับ / ส่งเป็นไปด้วย
- การรับข้อมูลข้ามเครื่องคอมพิวเตอร์ มีลักษณะการทำงานคล้ายคลึงกับการอ่านเพิ่มข้อมูล
- การส่งข้อมูลข้ามเครื่องคอมพิวเตอร์ มีลักษณะการทำงานคล้ายคลึงกับการเขียนเพิ่มข้อมูล

I/O ในภาษา Java

- I/O ในภาษา Java มีการทำงานแบบ **stream**
 - Input stream (ท่อเข้า) เอาไว้อ่านข้อมูล
 - Output stream (ท่อออก) เอาไปสำหรับเขียนข้อมูล
- ใน Java สามารถสร้าง I/O โดยการสร้าง **object** ของ **Class**
 - InputStream
 - OutputStream



OutputStream

- Class พื้นฐานสำหรับการส่งข้อมูลคือ
java.io.OutputStream
- ซึ่งมี method สำคัญที่ใช้งานคือ
 - **public abstract void write(int b) throws IOException**
 - **public void write(byte[] data) throws IOException**
 - **public void write(byte[] data, int offset, int length)**
throws IOException
 - **public void flush() throws IOException**
 - **public void close() throws IOException**

InputStream

- Class พื้นฐานสำหรับการรับข้อมูลคือ
java.io.InputStream
- ซึ่งมี method สำคัญที่ใช้งานคือ
 - **public abstract int read() throws IOException**
 - **public int read(byte[] input) throws IOException**
 - **public int read(byte[] input, int offset, int length)**
throws IOException
 - **public long skip(long n) throws IOException**
 - **public void available() throws IOException**
 - **public void close() throws IOException**

Java กับ แฟ้มข้อมูล

- การจัดการกับแฟ้มข้อมูลใน Java สามารถทำได้หลายวิธี
- วิธีที่ง่ายที่สุดคือการใช้ **Class File**
- ตัวอย่าง :
 - `File f = new File(String ชื่อไฟล์);`
 - `File f = new File(String ชื่อ path, String ชื่อไฟล์);`
 - `File f = new File(File path, String ชื่อไฟล์);`

Java File Methods

- รายละเอียดเกี่ยวกับทุก method ดูได้จาก JavaDoc manual
- Method ที่สำคัญๆ มีดังนี้
 - **boolean delete();** ลบแฟ้มข้อมูล
 - **boolean exists();** ตรวจสอบว่ามีแฟ้มข้อมูลอยู่หรือไม่
 - **boolean isDirectory();** ตรวจสอบว่าเป็น directory หรือไม่
 - **boolean isFile();** ตรวจสอบว่าเป็น file หรือไม่
 - **long length();** คืนขนาดของแฟ้มข้อมูล
 - **File[] listFiles();** คืนชื่อ file ที่อยู่ใน directory นั้น
 - **String[] list();** คืนชื่อ file ที่อยู่ใน directory นั้น
 - **String getName();** เอาชื่อ file ออกมา

ตัวอย่าง 1

```
import java.io.*;

public class Example1 {
    public static void main(String[] args) {
        File f = new File("myFile.txt");
        if(!f.exists()) {
            System.out.println("File does not exist");
            System.exit(1);
        }

        if(f.isFile()) {
            System.out.println("myFile.txt is a File");
            System.out.println("File size = " + f.length());
        } else if(f.isDirectory()) {
            System.out.println("myFile.txt is a directory");
        } else {
            System.out.println(".....");
        }
    }
}
```

ตัวอย่าง 2

```
import java.io.*;

public class Example2 {
    public static void main(String[] args) {
        File f = new File("C:\\AppServ");
        File list[] = f.listFiles();

        for(int i = 0; i < list.length; i++) {
            if(list[i].isFile()) {
                System.out.println(list[i].getName() + " is a file");
            } else if (list[i].isDirectory()) {
                System.out.println(list[i].getName() + " is a directory");
            } else {
                System.out.println(list[i].getName() + " is unknown");
            }
        }
    }
}
```

การเขียน/อ่าน เพิ่มข้อมูลด้วย Java

- ในการเขียนหรืออ่านเพิ่มข้อมูลด้วยจาวาจะนั้น จะใช้ **Class**

- ในการอ่านข้อมูล

public **FileInputStream**(File file) throws FileNotFoundException

- ในการเขียนข้อมูล

public **FileOutputStream**(File file) throws FileNotFoundException

- ทั้ง 2 **Class** นี้สืบทอดมาจาก **InputStream** และ **OutputStream**
- ทั้ง 2 **Class** นี้จึงสามารถใช้ **method** ต่างๆ ของ **Class** แม่ได้หมด

การอ่านข้อมูลจากไฟล์ข้อมูล

```
import java.io.*;

public class ReadFile {
    public static void main(String[] args) {
        try {
            int n;
            byte[] b = new byte[16];

            File f = new File("myfile.txt");
            FileInputStream fin = new FileInputStream(f);
            while((n = fin.read(b)) > 0) {
                String data = new String(b, 0, n);
                System.out.print(data);
            }

            fin.close();
        } catch(Exception e) { e.printStackTrace(); }
    }
}
```

การเขียนข้อมูลลงไฟล์ข้อมูล

```
import java.io.*;

public class WriteFile {
    public static void main(String[] args) {
        try {
            String msg = "Hello World";
            File f = new File("myfile.txt");
            FileOutputStream fout = new FileOutputStream(f);
            byte[] b = msg.getBytes();
            fout.write(b);
            fout.close();
        } catch(Exception e) { e.printStackTrace(); }
    }
}
```

Filter Stream

- ทั้ง **InputStream** และ **OutputStream** เป็น **low-level API** ซึ่งสามารถอ่าน และ เขียน ข้อมูลในระดับไบต์
- ถ้าต้องการรับและส่งข้อมูลที่เป็นตัวอักษร หรือตัวเลข อย่างง่ายสามารถใช้ **Class** ของ **Java** ที่ออกแบบมาเพื่อให้ใช้งานง่ายขึ้น คือ
 - **PrintWriter**
 - สำหรับเขียนข้อมูล
 - **BufferedReader**
 - สำหรับอ่านข้อมูล
 - **Class** ทั้ง 2 นี้จะช่วยลดความยุ่งยากของ **InputStream** และ **OutputStream** เอาไว้

PrintWriter

□ Class PrintWriter

□ Constructor

- **public PrintWriter (OutputStream out)**

□ Method ในการเขียนค่า String

- **public void print(String s)**
- **public void println(String x)**
- **public void flush();**

ตัวอย่างการใช้งาน PrintWriter

```
import java.io.*;

public class PwTest {
    public static void main(String[] args) {
        try {
            String msg = "Hello World";
            File f = new File("myfile.txt");
            FileOutputStream fout = new FileOutputStream(f);
            PrintWriter pout = new PrintWriter(fout);
            pout.print(msg);
            pout.flush();
            fout.close();
        } catch(Exception e) { e.printStackTrace(); }
    }
}
```

BufferedReader

- **Class InputStreamReader (เป็น Class ลูกของ Reader)**
 - **Constructor**
 - `public InputStreamReader(InputStream in)`
- **Class BufferedReader**
 - **Constructor**
 - `public BufferedReader(Reader in)`
 - **Method**
 - `public String readLine() throws IOException`

ตัวอย่างการใช้งาน BufferedReader (1)

```
import java.io.*;  
  
public class BrTest {  
    public static void main(String[] args) {  
        try {  
            String msg;  
            File f = new File("myfile.txt");  
            FileInputStream fin = new FileInputStream(f);  
            InputStreamReader ir = new InputStreamReader(fin);  
            BufferedReader br = new BufferedReader(ir);  
  
            while((msg = br.readLine()) != null)  
                System.out.println(msg);  
  
            fin.close();  
        } catch(Exception e) { e.printStackTrace(); }  
    }  
}
```

ตัวอย่างการใช้งาน BufferedReader (2)

```
import java.io.*;

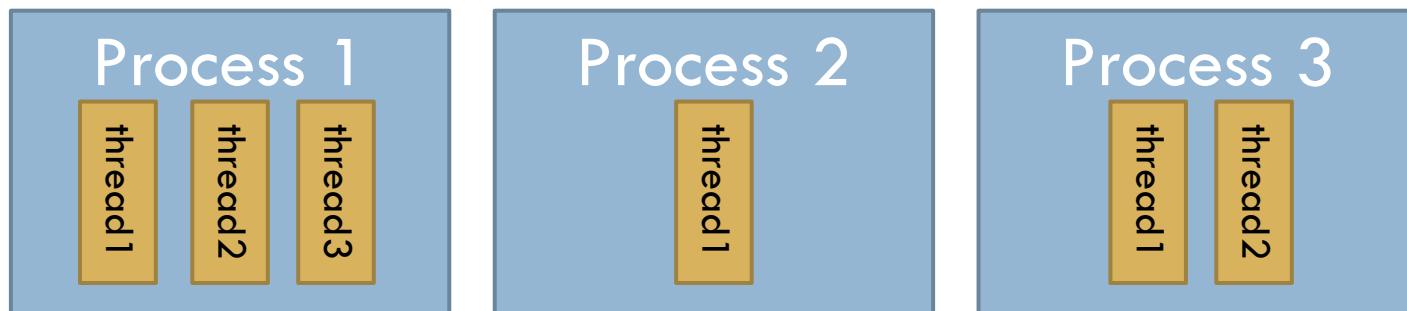
public class BrTest2 {
    public static void main(String[] args) {
        try {
            String msg;
            File f = new File("myfile.txt");
            BufferedReader br    = new BufferedReader(
                new InputStreamReader(
                    new FileInputStream(f)));
            while((msg = br.readLine()) != null)
                System.out.println(msg);
        } catch(Exception e) { e.printStackTrace(); }
    }
}
```

JAVA THREAD

030523313 - Network programming
Asst. Prof. Dr. Choopan Rattanapoka

อะไรคือ Thread

- ในระบบปฏิบัติการสมัยใหม่ เวลาเรียกใช้งานโปรแกรมก็ต้องสร้าง **process** ของโปรแกรมนั้น
- Process คือ โปรแกรมที่กำลังทำงานอยู่ในระบบ
- แต่ละ Process จะมีอย่างน้อย 1 Thread ที่ทำงานอยู่
- Thread เรียกอีกอย่างว่า **lightweight process**



Multiple Threads ?

- แล้วทำไมถึงต้องมีการใช้ **Thread** หลาย **Thread** ใน **process** เดียว
 - การตอบสนองของโปรแกรมที่ดีกับผู้ใช้
 - โปรแกรม **GUI** เช่น **winzip** ขณะที่ทำการบีบอัดแฟ้มข้อมูลถ้าไม่มี **Thread** ตัวหน้าโปรแกรม **GUI** จะค้างจนกว่าการบีบอัดแฟ้มข้อมูลจะเสร็จสิ้น
 - สำหรับงานที่สามารถทำพร้อมกันได้
 - โปรแกรมเซิฟเวอร์ เช่น **Web server** สามารถรองรับผู้ใช้หลายคนพร้อมกันได้
 - ใช้ประโยชน์จาก **CPU** แบบ **multicore**, หรือ **Multiple CPU** อย่างเต็มประสิทธิภาพ
 - โปรแกรมเช่น สร้างภาพ 3 มิติ สามารถใช้หลาย **Thread** ช่วยกันประมวลผลได้

การใช้งาน Thread

- **Thread** มีข้อดีหลายอย่าง แต่ก็ไม่เหมาะสมสำหรับงานบางประเภท
- การใช้ **Thread** เป็นการเพิ่มการใช้งานทรัพยากรของระบบ
 - **RAM** ที่ใช้เก็บตัวแปรต่างๆ ของแต่ละ **thread**
 - **Overhead** ของการทำ **context switch** ของ **CPU**
- ตัวอย่าง โปรแกรมที่ไม่จำเป็นต้องใช้ **Thread**
 - โปรแกรมตรวจสอบ **e-mail** ทุกๆ 5 นาที
 - เราสามารถจะใช้ 1 **thread** หลับรอแล้วตื่นขึ้นทุกๆ 5 นาทีเพื่อตรวจสอบ **e-mail** ซึ่งจะดีกว่า สร้าง **thread** ใหม่ทุกครั้งเมื่อครบ 5 นาที
- **Thread** เป็นเรื่องง่ายในการเริ่มต้นใช้งาน แต่ยากมากที่จะชำนาญ

การสร้าง Thread ใน Java

- การสร้างและใช้งาน **Thread** ในภาษา Java มีด้วยกัน 2 วิธี
 - สร้าง **class** ที่ **extends** มาจาก **Class Thread**
 - เป็นวิธีที่ง่ายที่สุดในการเขียนโปรแกรมเพื่อใช้ **Thread** ในภาษา Java
 - สร้าง **class** ที่ **implements Runnable Interface**
 - เป็นวิธีที่ถูกใช้งานมากที่สุดในโปรแกรมทั่วไป

Extend จาก Class `java.lang.Thread`

- การสร้าง Class ที่สามารถทำงานเป็น Thread ให้เพิ่มคำว่า **extends Thread** ไปข้างหลังชื่อ Class

```
public class TwoThread extends Thread {  
    ....  
    ....  
}
```

- จะต้องทำการ **Override** เมธอดที่ชื่อว่า `run()`
 - `public void run() { }`
 - เมธอดนี้เป็นจุดเริ่มต้นการทำงานของ Thread

ตัวอย่างการสร้าง Class ที่สามารถทำ Thread

```
import java.io.*;  
  
public class TwoThread extends Thread {  
    public void run( ) {  
        for(int i = 0; i < 10; i++) {  
            System.out.println("New Thread");  
        }  
    }  
}
```

- Class TwoThread มีการ **extends Thread**
- มีการ **override** เมетодด **run()**
- การทำงานของ **Thread** จะวนลูป 10 รอบเพื่อพิมพ์คำว่า “New Thread”

การเรียกใช้งาน Thread

- การสร้าง Thread ขึ้นในโปรแกรม (**Spawning**) จะต้องสร้างขึ้นจาก Thread ที่กำลังทำงานอยู่
 - การสร้าง Thread นั้นเหมือนกับการสร้าง Object ปกติในภาษาจาวา
- TwoThread tt = new TwoThread();**
- เมื่อต้องการให้ Thread ทำงาน ก็เรียกใช้เมธอด **start()** ของ Object นั้น
- tt.start();**
- หลังจากนั้น Thread จะเริ่มทำงานในเมธอด **run()** ในขณะที่ผู้เรียก ก็ทำงานคำสั่งถัดไปต่อได้ทันที

ตัวอย่างการเรียกใช้งาน Thread

```
import java.io.*;

public class TwoThread extends Thread {
    public void run( ) {
        for(int i = 0; i < 10; i++) {
            System.out.println("New Thread");
        }
    }

    public static void main(String[] args) {
        TwoThread tt = new TwoThread();
        tt.start();

        for(int i = 0; i < 10; i++) {
            System.out.println("Main Thread");
        }
    }
}
```

คิดว่าผลการรันจะเป็นอย่างไร

การ delay หรือ sleep ในภาษาจาวา

- ในการทำงานบางอย่าง โปรแกรมจำเป็นต้องมีการ delay หรือ sleep รอ

```
long startTime = System.currentTimeMillis();  
long stopTime = System.currentTimeMillis() + 6000;  
while(System.currentTimeMillis() < stopTime) { }
```
- การแบบนี้จะให้ CPU ทำงานตลอดเวลา และล้าบเปลี่ยงทรัพยากร
- ถ้าต้องการให้โปรแกรม delay หรือ sleep จะต้องส่งผ่าน Class Thread
Thread.sleep(เวลาเป็น ms);
- เนื่องจากเมธอด sleep จะมีการโยน Exception ดังนั้นต้องนำ try-catch มาครอบคำสั่งเอาไว้

ตัวอย่างการใช้ Sleep

```
import java.io.*;  
  
public class TestSleep {  
    public static void main(String[] args) {  
        System.out.println("Hello");  
  
        try {  
            Thread.sleep(3000);  
        } catch (Exception e) {}  
  
        System.out.println("Bye Bye");  
    }  
}
```

ตัวอย่างการทำงานหลัก Thread กับ Sleep

```
import java.io.*;

public class MultiThread extends Thread {
    String myName;
    long sleepTime;

    public MultiThread(String myName, long sleepTime) {
        this.myName = myName;
        this.sleepTime = sleepTime;
    }

    public void run() {
        for(int i = 0; i < 5; i++) {
            System.out.println(myName);
            try {
                Thread.sleep(sleepTime);
            } catch(Exception e) {}
        }
    }

    public static void main(String[] args) {
        MultiThread t1 = new MultiThread("-1-", 1000);
        MultiThread t2 = new MultiThread("-2-", 2000);
        MultiThread t3 = new MultiThread("-3-", 3000);

        t1.start();
        t2.start();
        t3.start();
    }
}
```

ผลการรัน

-1-
-3-
-2-
-1-
-2-
-1-
-3-
-1-
-2-
-1-
-2-
-1-
-3-
-2-
-2-
-3-
-3-

ข้อเสียของการใช้ Extends Thread

- ในการใช้งาน **Thread** ผ่านวิธี **extends Thread** นั้นใช้งานง่าย
- แต่มีข้อจำกัดของภาษาจาวา **ที่กัน** การสืบทอดจากหลาย **Class**
 - ภาษาจาวาอนุญาตให้มีการสืบทอด **Class** มาจาก **Class** แม่เพียง **Class** เดียวเท่านั้น ซึ่งเป็นการแกบัญหาที่เกิดขึ้นจากการสืบทอด **Class** หลาย **Class** จากภาษา **C++**
- ทำให้การทำใช้งานบางประเภทไม่สามารถทำได้
 - เช่น **Class** ที่ทำ **GUI** เช่น **JComponent** โปรแกรมจำเป็นต้องเขียน **extends JComponent** เพื่อใช้งาน ทำให้ไม่สามารถเขียน **extends Thread** ได้
- วิธีแก้ไขคือต้องเขียน **Thread** ด้วยวิธีที่ 2 คือ **implements Runnable**

การสร้าง Thread ด้วยวิธี implements Runnable

- การสร้าง Class ที่สามารถทำงานเป็น Thread ให้เพิ่มคำว่า **implements Runnable** ไปข้างหลังชื่อ Class

```
public class TwoThread implements Runnable {  
    ....  
    ....  
}
```

- จะต้องทำการ **Override** เมออดที่ชื่อว่า **run()** [เห็นอนวิธี **extends**]
 - **public void run() { }**
 - เมออดนี้เป็นจุดเริ่มต้นการทำงานของ Thread

ตัวอย่างการสร้าง Class ที่สามารถทำ Thread

```
import java.io.*;  
  
public class TwoThread implements Runnable {  
    public void run( ) {  
        for(int i = 0; i < 10; i++) {  
            System.out.println("New Thread");  
        }  
    }  
}
```

- Class TwoThread มีการ implements Runnable
- มีการ override เมธอด run()
- การทำงานของ Thread จะวนลูป 10 รอบเพื่อพิมพ์คำว่า “New Thread”

การเรียกใช้งาน Thread จาก Class ที่ implements Runnable

- สร้าง Object ของ Class นั้นตามปกติ

```
TwoThread tt = new TwoThread();
```

- สร้าง Object Thread สำหรับ Class นั้น

```
Thread t = new Thread(tt);
```

- เรียกใช้งาน Thread ผ่าน Object Thread ที่สร้างขึ้น

```
t.start();
```

- หลังจากนั้น Thread จะเริ่มทำงานในเมธอด run() ในขณะที่ผู้เรียกเก็ททำงานคำสั่งถัดไปต่อได้ทันที

ตัวอย่างการเรียกใช้งาน Thread

```
import java.io.*;

public class TwoThread implements Runnable {
    public void run( ) {
        for(int i = 0; i < 10; i++) {
            System.out.println("New Thread");
        }
    }

    public static void main(String[] args) {
        TwoThread tt = new TwoThread();
        Thread t = new Thread(tt);
        t.start();

        for(int i = 0; i < 10; i++) {
            System.out.println("Main Thread");
        }
    }
}
```

ข้อแตกต่างของ 2 วิธีในการสร้างและใช้งาน Thread

(1) วิธี extends Thread

```
import java.io.*;  
  
public class TwoThread extends Thread {  
    public void run() {  
        for(int i = 0; i < 10; i++) {  
            System.out.println("New Thread");  
        }  
    }  
  
    public static void main(String[] args) {  
        TwoThread tt = new TwoThread();  
        tt.start();  
  
        for(int i = ; i < 10; i++) {  
            System.out.println("Main Thread");  
        }  
    }  
}
```

- การเขียน Class
- (1) **extends Thread**
 - (2) **implements Runnable**

```
import java.io.*;  
  
public class TwoThread implements Runnable {  
    public void run() {  
        for(int i = 0; i < 10; i++) {  
            System.out.println("New Thread");  
        }  
    }  
  
    public static void main(String[] args) {  
        TwoThread tt = new TwoThread();  
        Thread t = new Thread(tt);  
        t.start();  
  
        for(int i = 0; i < 10; i++) {  
            System.out.println("Main Thread");  
        }  
    }  
}
```

การสร้างและเรียกใช้งาน Object
ของ Thread

(2) วิธี implements Runnable

ตัวอย่าง: โปรแกรมหา summation

```
import java.io.*;

public class Sum {
    int from;
    int where;
    int result = 0;

    public Sum(int from, int where) {
        this.from = from;
        this.where = where;
    }

    public void run() {
        for(int i = from; i <= where; i++) {
            result += i;
        }
    }

    public int getResult() {
        return result;
    }

    public static void main(String[] args) {
        Sum s = new Sum(0, 1000000);
        s.run();
        System.out.println("Result = " + s.getResult());
    }
}
```

นำ Thread มาประยุกต์ใช้

- จากตัวอย่าง เป็นการหา **Summation** ของตัวเลข 0-1000000 โดยใช้
- เราสามารถใช้นำ **Thread** มาประยุกต์ใช้ได้ เช่น
 - แบ่งการทำงานของ **Summation** ของเป็น 2 ส่วน คือ
 - **Summation** ของ 0 – 499999 (**Thread 1**)
 - **Summation** ของ 500000 – 1000000 (**Thread 2**)
 - จากนั้นนำผลลัพธ์ของทั้ง 2 **Thread** รวมกันเป็นค่าตอบ
- **คำเตือน** ประสิทธิภาพที่ได้จากการแบ่งการทำงานเป็น **2 Thread** ของ **Summation** นี้ เวลาที่ใช้ประมวลผลจะไม่แตกต่างกันมากนัก แต่ถ้าเปลี่ยนจากการทำ **Summation** เป็นการประมวลผลอย่างอื่นที่ใช้เวลานานๆ การทำงานแบบ **Multi-thread** จะใช้เวลาห้อยกว่า

การนำ Thread มาประยุกต์ใช้

```
import java.io.*;  
  
public class SumThreadWrong implements Runnable {  
    int from;  
    int where;  
    int result = 0;  
  
    public SumThreadWrong(int from, int where) {  
        this.from = from;  
        this.where = where;  
    }  
  
    public void run() {  
        for(int i = from; i <= where; i++) {  
            result += i;  
        }  
    }  
  
    public int getResult() {  
        return result;  
    }  
}
```

มีปัญหาในการรวมค่าของ
Thread 1 และ Thread 2

```
public static void main(String[] args) {  
    int s = 0;  
    SumThreadWrong s1 = new SumThreadWrong(0, 499999);  
    SumThreadWrong s2 = new SumThreadWrong(500000, 1000000);  
    Thread t1 = new Thread(s1);  
    Thread t2 = new Thread(s2);  
    try {  
        t1.start(); t2.start();  
        s = s1.getResult() + s2.getResult();  
    } catch(Exception e){}  
    System.out.println("Result = " + s);  
}
```

การหยุดรอ Thread ให้ทำงานเสร็จ

- ในการทำงานกับ **Thread** ใน **Java** ในการนี้ที่เราต้องการรอจนกว่า **Thread** จะทำงานเสร็จ จะใช้ เมธอด **join()**
- **Thread** ที่เรียกใช้งาน **join()** จะหยุดรอจนกว่าทั้ง **Thread** ที่รอนั้น เสร็จสิ้นการทำงาน หรือ ตาย
- จากนั้น **Thread** ที่เรียกใช้งาน **join()** จึงจะสามารถทำงานต่อไปได้

การใช้งาน join()

```
import java.io.*;

public class SumThread implements Runnable {
    int from;
    int where;
    int result = 0;

    public SumThread(int from, int where) {
        this.from = from;
        this.where = where;
    }

    public void run() {
        for(int i = from; i <= where; i++) {
            result += i;
        }
    }

    public int getResult() {
        return result;
    }
}

public static void main(String[] args) {
    int s = 0;
    SumThread s1 = new SumThread(0, 499999);
    SumThread s2 = new SumThread(500000, 1000000);
    Thread t1 = new Thread(s1);
    Thread t2 = new Thread(s2);

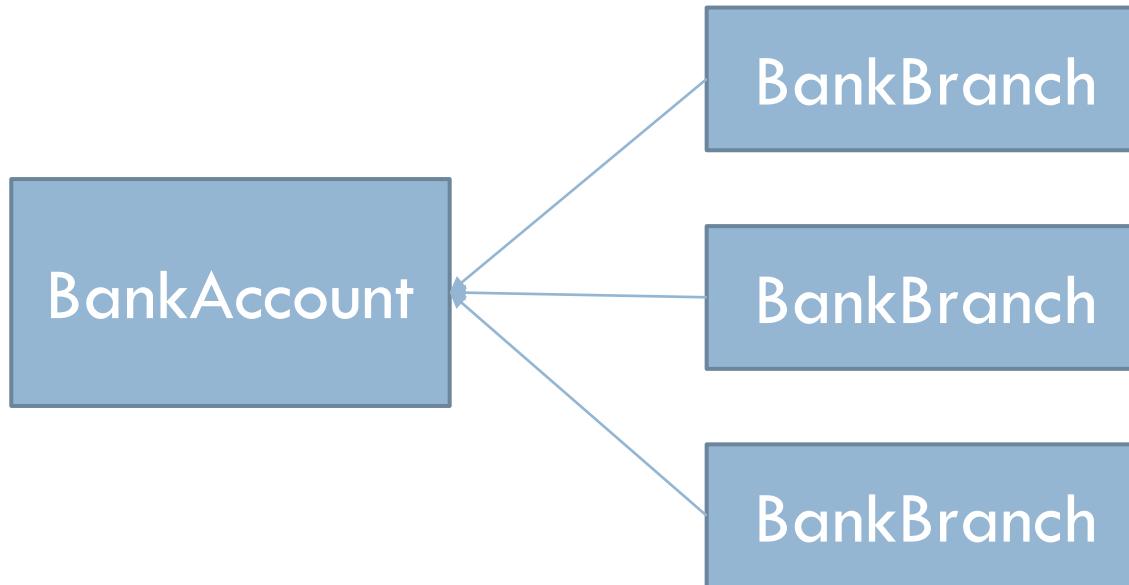
    try {
        t1.start(); t2.start();
        t1.join(); t2.join();
        s = s1.getResult() + s2.getResult();
    } catch(Exception e){}
    System.out.println("Result = " + s);
}
```

Thread Concurrency

- การทำงานแบบ **Multithread** ควรจะให้ความสำคัญกับคำว่า “**Thread safe**”
- ซึ่งหมายถึงเมื่อมี **Thread** ทำงานพร้อมกันหลายตัว และแต่ละตัวมีการเปลี่ยนแปลงค่าตัวแปรตัวเดียวกัน ถ้าไม่มีการจัดการให้ดี อาจจะทำให้ค่าตัวแปรนั้นผิดไป
- ในภาษา **Java** มีคำขยายชื่อ **synchronized** ขึ้นมาเพื่อทำงานกับ **critical region**.
- **Critical region** คือส่วนของโปรแกรมที่ควรอนุญาตให้ **Thread** สามารถเข้าช่วงนี้ได้ทีละ 1 **thread** เท่านั้น

ตัวอย่างปัญหาของ Race Condition

- ตัวอย่างการทำงานของธนาคารโดยจะมี Class อิสระ 3 Class
 - **BankAccount** : เก็บเงินในบัญชี
 - **BankBranch** : เหนี่ยนสาขาหรือตู้ ATM ที่รับฝาก/ถอนเงินในบัญชี
 - **BadThread** : คลาส main ยกตัวอย่างการทำงาน



Class : BankAccount

```
public class BankAccount {  
    volatile int money = 0;  
  
    public BankAccount(int money) {  
        this.money = money;  
    }  
  
    public void deposit(int money) {  
        for(int i = 0; i < money; i++) {  
            this.money++;  
        }  
    }  
    public void withdraw(int money) {  
        for(int i = 0; i < money; i++) {  
            this.money--;  
        }  
    }  
  
    public int getBalance() {  
        return money;  
    }  
}
```

Class : BankBranch

```
public class BankBranch extends Thread {  
    BankAccount bankAcct = null;  
    String method = null;  
    int money = 0;  
  
    public BankBranch(BankAccount bankAcct, String method, int money) {  
        this.bankAcct = bankAcct;  
        this.method = method;  
        this.money = money;  
    }  
  
    public void deposit(int money) {  
        bankAcct.deposit(money);  
    }  
  
    public void withdraw(int money) {  
        bankAcct.withdraw(money);  
    }  
    public void run() {  
        if(method.equals("deposit")) deposit(money);  
        else withdraw(money);  
    }  
}
```

Class: BadThread (main)

```
public class BadThread {
    public static void main(String[] args) {
        BankAccount bankAcct = new BankAccount(1000);

        BankBranch b1 = new BankBranch(bankAcct, "deposit", 10000);
        BankBranch b2 = new BankBranch(bankAcct, "withdraw", 10000);

        b1.start();
        b2.start();

        try {
            b1.join(); b2.join();
        } catch(Exception e) {}
        System.out.println("Balance = " + bankAcct.getBalance());
    }
}
```

ผลการทำงานไม่เหมือนกันในแต่ละครั้ง

Balance = 1000

Balance = -3820

Balance = 941

Balance = -2888

Balance = 937

Balance = -386

Balance = 937

Balance = -386

Balance = -4906

การแก้ไขปัญหาการแย่งกันเข้าถึงข้อมูลของ Thread

- วิธีที่ 1 ใส่ **synchronized** ในเมธอดที่ต้องการทำ Critical Region
 - ทุก method ที่มี synchronized จะถูก block ทั้งหมดเพื่อให้ thread เพียงตัวเดียวเข้าใช้งาน เพียง method ใด method หนึ่ง
 - รูปแบบตัวอย่าง
- public **synchronized** void deposit(int money)
- ควรจะใส่ที่ตรงไหนบ้าง เพื่อบังกันปัญหา race condition ?
- วิธีที่ 2 เลือกทำ Critical Region เช่นบางส่วนของเมธอด วิธีนี้จะทำให้ลดหย่อนและสามารถเลือก lock แยกสำหรับกรณีที่เข้าถึงตัวแปรคงละตัวได้

- จะต้องสร้าง Class ที่เป็น Object เพื่อใช้ในการ lock

- ตัวอย่าง :

```
public void deposit(int money) {  
    synchronized(o) {  
        for(int i = 0; i < money; i++) {  
            this.money++;  
        }  
    }  
}
```

Keyword

- ทบทวนเรื่อง **Keyword** หน้าชี้อตัวแปร

- **volatile**

- ตัวอย่าง: **volatile int X;**

- เป็นการประกาศตัวแปรที่จะบังคับให้เหมือนมีการเปลี่ยนแปลงข้อมูลใน **cache** ให้เขียนกลับลงไปในหน่วยความจำหลักทันที และเดือน **thread** อื่นที่มีการใช้งานตัวแปรนี้ให้อัพเดทข้อมูลจากหน่วยความจำหลัก
 - สามารถใช้ในการควบคุม **race condition** เบื้องต้นได้ แต่ในงานทั่วไปควรใช้คู่กับ **synchronized** แต่ถ้าใช้มากไปจะทำให้การทำงานโดยรวมของโปรแกรมช้าลง

- **static**

- ตัวอย่าง: **static int X;**
 - เป็นการประกาศตัวแปรที่มีที่เก็บข้อมูลเพียงที่เดียว และทุก **Object** จะอ้างอิงถึงที่เก็บข้อมูลที่เดียวกัน

JAVA THREAD 2

030523313 - Network programming
Asst. Prof. Dr. Choopan Rattanapoka

DeadLock (1)

- Deadlock ที่สถานการณ์ที่เมื่อมี thread จำนวนตั้งแต่ 2 ตัวขึ้นไป ถูก block ตลอดเวลาเพื่อรอกันเอง

```
public class Friend {  
    private String name;  
  
    public Friend(String name) {  
        this.name = name;  
    }  
  
    public String getName() {  
        return this.name;  
    }  
  
    public synchronized void bow(Friend bower) {  
        System.out.println(name + ": " + bower.getName() + " has bowed to me.");  
        bower.bowBack(this);  
    }  
    public synchronized void bowBack(Friend bower) {  
        System.out.println(name + ": " + bower.getName() + " has bowed back to me.");  
    }  
    public void run() {  
    }
```

Deadlock (2)

```
public static void main(String[] args) {
    final Friend tom = new Friend("Tom");
    final Friend bob = new Friend("Bob");
    new Thread(new Runnable() {
        public void run() { tom.bow(bob); }
    }).start();
    new Thread(new Runnable() {
        public void run() { bob.bow(tom); }
    }).start();
}
```

```
C:\tmp>java Friend
Tom: Bob has bowed to me.
Bob: Tom has bowed back to me.
Bob: Tom has bowed to me.
Tom: Bob has bowed back to me.

C:\tmp>java Friend
Tom: Bob has bowed to me.
Bob: Tom has bowed back to me.
Bob: Tom has bowed to me.
Tom: Bob has bowed back to me.

C:\tmp>java Friend
Tom: Bob has bowed to me.
Bob: Tom has bowed to me.
```

Producer-Consumer Problem



- ปัญหา producer-consumer problem คือ
 - ผู้ผลิต (producer) ผลิตสินค้าแล้วนำไปเก็บในโกดังสินค้า (warehouse)
 - ผู้บริโภค (consumer) จะนำสินค้าออกจากโกดัง
 - Warehouse จะเก็บสินค้าได้แค่ 1 ชิ้น
 - Producer จะต้องรอ ไม่ผลิตของถ้าโกดังเต็ม
 - Consumer จะต้องรอ ไม่นำสินค้าออกจากโกดังถ้าโกดังว่าง
- เวลาในการผลิตหรือนำสินค้าออกจะสุ่มระหว่าง 0 – 999 ms

Class : Warehouse (v.1)

```
public class Warehouse {  
    volatile int productID;  
    volatile boolean empty = true;  
  
    public synchronized void put(int productID) {  
        while (!empty) { }  
        empty = false;  
        this.productID = productID;  
    }  
  
    public synchronized int take() {  
        while (empty) { }  
        int result = this.productID;  
        empty = true;  
        return result;  
    }  
}
```

Class: Producer (v.1)

```
import java.util.*;  
  
public class Producer extends Thread {  
    Warehouse w;  
  
    public Producer(Warehouse w) {  
        this.w = w;  
    }  
  
    public void run() {  
        Random r = new Random();  
        for(int i = 0; i < 10; i++) {  
            int id = r.nextInt(100);  
            System.out.println("Producer: try to put product with id = " + id);  
            w.put(id);  
            System.out.println("Producer: put product with id = " + id);  
            try {  
                Thread.sleep(r.nextInt(1000));  
            } catch(Exception e) {}  
        }  
    }  
}
```

Class: Consumer (v.1)

```
import java.util.*;  
  
public class Consumer extends Thread {  
    Warehouse w;  
  
    public Consumer(Warehouse w) {  
        this.w = w;  
    }  
  
    public void run() {  
        Random r = new Random();  
        for(int i = 0; i < 10; i++) {  
            System.out.println("Consumer : try to take product");  
            int id = w.take();  
            System.out.println("Consumer : take product with id = " + id);  
            try {  
                Thread.sleep(r.nextInt(1000));  
            } catch(Exception e){}  
        }  
    }  
}
```

Class : ProducerConsumer (main)

```
public class ProducerConsumer {  
    public static void main(String[] args) {  
        Warehouse w = new Warehouse();  
        Producer p = new Producer(w);  
        Consumer c = new Consumer(w);  
        p.start(); c.start();  
    }  
}
```

The screenshot shows a Windows Command Prompt window titled "Command Prompt - java ProducerConsumer". The command entered is "java ProducerConsumer". The output displayed is:
C:\tmp\warehouse1>java ProducerConsumer
Consumer : try to take product
Producer: try to put product with id = 20

Deadlock...??!!
ເຂົ້ານິດຕຽງໃກ້ນເໜີຍ ??!

แก้: Warehouse (v.2)

```
public class Warehouse {
    volatile int productID;
    volatile boolean empty = true;

    public synchronized boolean put(int productID) {
        if (!empty) return false;
        empty = false;
        this.productID = productID;
        return true;
    }

    public synchronized int take() {
        if (empty) return -1;
        int result = this.productID;
        empty = true;
        return result;
    }
}
```

แก้: Producer (v.2)

```
import java.util.*;  
  
public class Producer extends Thread {  
    Warehouse w;  
  
    public Producer(Warehouse w) {  
        this.w = w;  
    }  
  
    public void run() {  
        Random r = new Random();  
        for(int i = 0; i < 10; i++) {  
            int id = r.nextInt(100);  
            System.out.println("Producer: try to put product with id = " + id);  
            while(!w.put(id));  
            System.out.println("Producer: put product with id = " + id);  
            try {  
                Thread.sleep(r.nextInt(1000));  
            } catch(Exception e) {}  
        }  
    }  
}
```

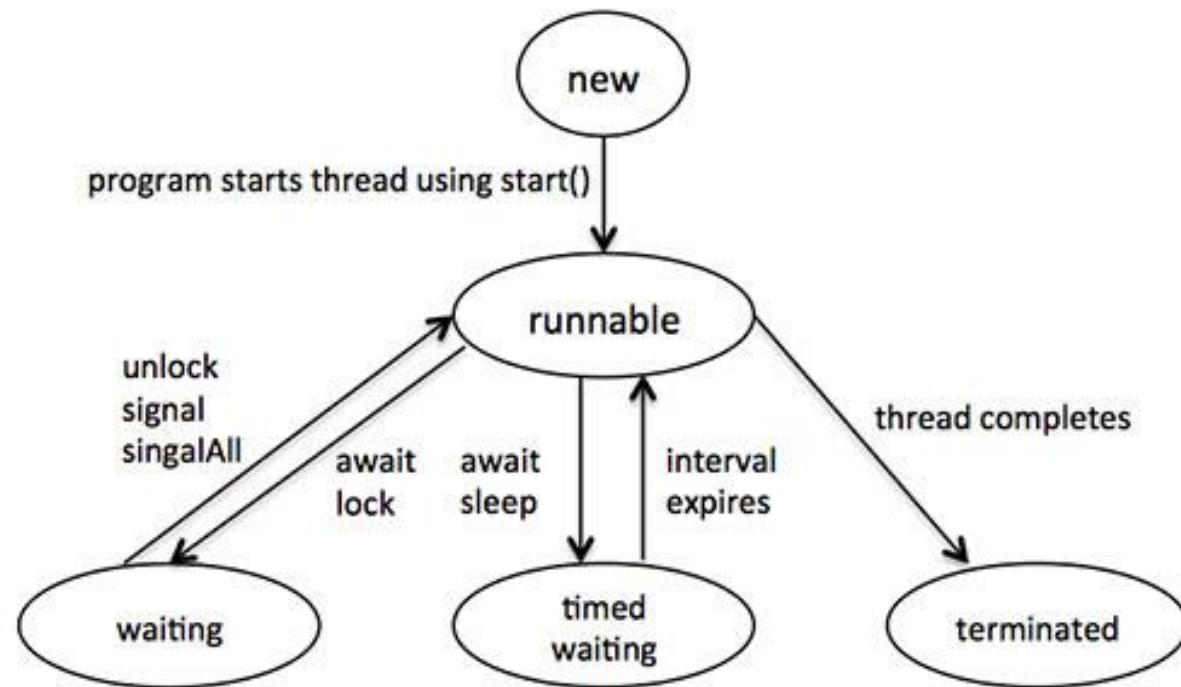
แก้: Consumer (v.2)

```
import java.util.*;  
  
public class Consumer extends Thread {  
    Warehouse w;  
  
    public Consumer(Warehouse w) {  
        this.w = w;  
    }  
  
    public void run() {  
        int id;  
        Random r = new Random();  
        for(int i = 0; i < 10; i++) {  
            System.out.println("Consumer : try to take product");  
            while((id = w.take()) == -1);  
            System.out.println("Consumer : take product with id = " + id);  
            try {  
                Thread.sleep(r.nextInt(1000));  
            } catch(Exception e){}  
        }  
    }  
}
```

คิดยังไงกับโปรแกรมนี้ ??

Wait/Notify/NotifyAll

- Thread เมื่อจำเป็นจะต้องรอข้อมูลอะไรที่อาจจะกินเวลา สามารถเรียกใช้งานเมธอด **wait()** เพื่อให้หยุดรอจนกว่าจะมีการแจ้งจาก Thread อื่น
- **notify()** เป็นเมธอดสำหรับส่งสัญญาณให้ Thread 1 ตัว เพื่อให้ตื่นจากการ wait()
- **notifyAll()** เป็นเมธอดสำหรับส่งสัญญาณให้ทุก Thread ที่อยู่ในสถานะ wait ให้ตื่นขึ้นเพื่อทำงาน



แก้: Warehouse (v.3)

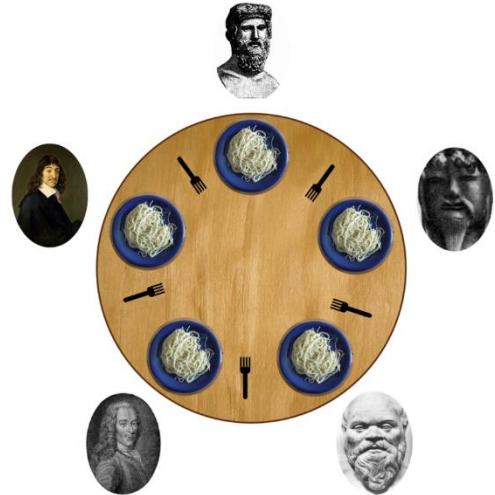
```
public class Warehouse {  
    volatile int productID;  
    volatile boolean empty = true;  
  
    public synchronized void put(int productID) {  
        if(!empty) {  
            try {  
                wait();  
            } catch(Exception e) {}  
        }  
        this.productID = productID;  
        empty = false;  
        notify();  
    }  
  
    public synchronized int take() {  
        if(empty) {  
            try {  
                wait();  
            } catch(Exception e) {}  
        }  
        int result = this.productID;  
        empty = true;  
        notify();  
        return result;  
    }  
}
```

ใช้ Class Producer และ Consumer ของ v.1

ปัญหา Classic ☺

□ Dining Philosophers

- มีนักปรัชญาอยู่ 5 คน นั่งบนโต๊ะกลม
- มีส้อมอยู่ทั้งหมด 5 อัน
- นักปรัชญาจะมีวิธีการทำงานคือ
 - คิด (think)
 - หิว (hungry)
 - กิน (eat)
- การที่นักปรัชญาจะกินได้ จะต้องหยิบส้อมด้านซ้าย และด้านขวาของตัวเองเท่านั้น และต้องหยิบได้ทั้ง 2 อันถึงจะสามารถกินได้
- **ตัวช่วย :** ถ้าต้องการกินแล้วไม่ได้ส้อมทั้ง 2 อันให้กลับไปคิดต่อ



- Idea (Mini Project กลุ่มละ 3 คน)

- โปรแกรม Chat
 - เกม Online แบบง่ายง่าย (ox, เกมไฟ, ...)
 - โปรแกรมค้นหาและแชร์ไฟล์ (napster)
 - โปรแกรมควบคุมคอมพิวเตอร์ปลายทาง
 - (บางคำสั่ง เช่น บันทึกหน้าจอ, เปิด App , ...)

- ตารางเวลา

- 3 ก.ย. 2564 เสนอรายชื่อสมาชิกในกลุ่ม, ชื่อ mini-project, ขอบเขตการทำงานของ mini-project
 - 10 ก.ย 2564 ส่งแก้ไขงานจากรอบที่ 1 (ถ้ามีแก้ไข)
 - 17 ก.ย 2564 ส่งแก้ไขงานจากรอบที่ 2 (ถ้ามีแก้ไข)
 - ส่งงาน หลังจากตารางสอบ Final เสร็จสิ้น 1 สัปดาห์ (นัดอีกที)

JAVA THREAD 3

030523313 - Network programming
Asst. Prof. Dr. Choopan Rattanapoka

ปัญหาที่ 1 : ในการใช้งาน Thread

- ในการทำงานจริงกับ **Thread** สำหรับ **Application** ทั่วไป และ **Server Application** การสร้าง **Thread** โดยไม่มีการ จำกัดจำนวนอาจจะทำให้เครื่อง คอมพิวเตอร์ที่เรียกใช้งาน โปรแกรมมีปัญหา

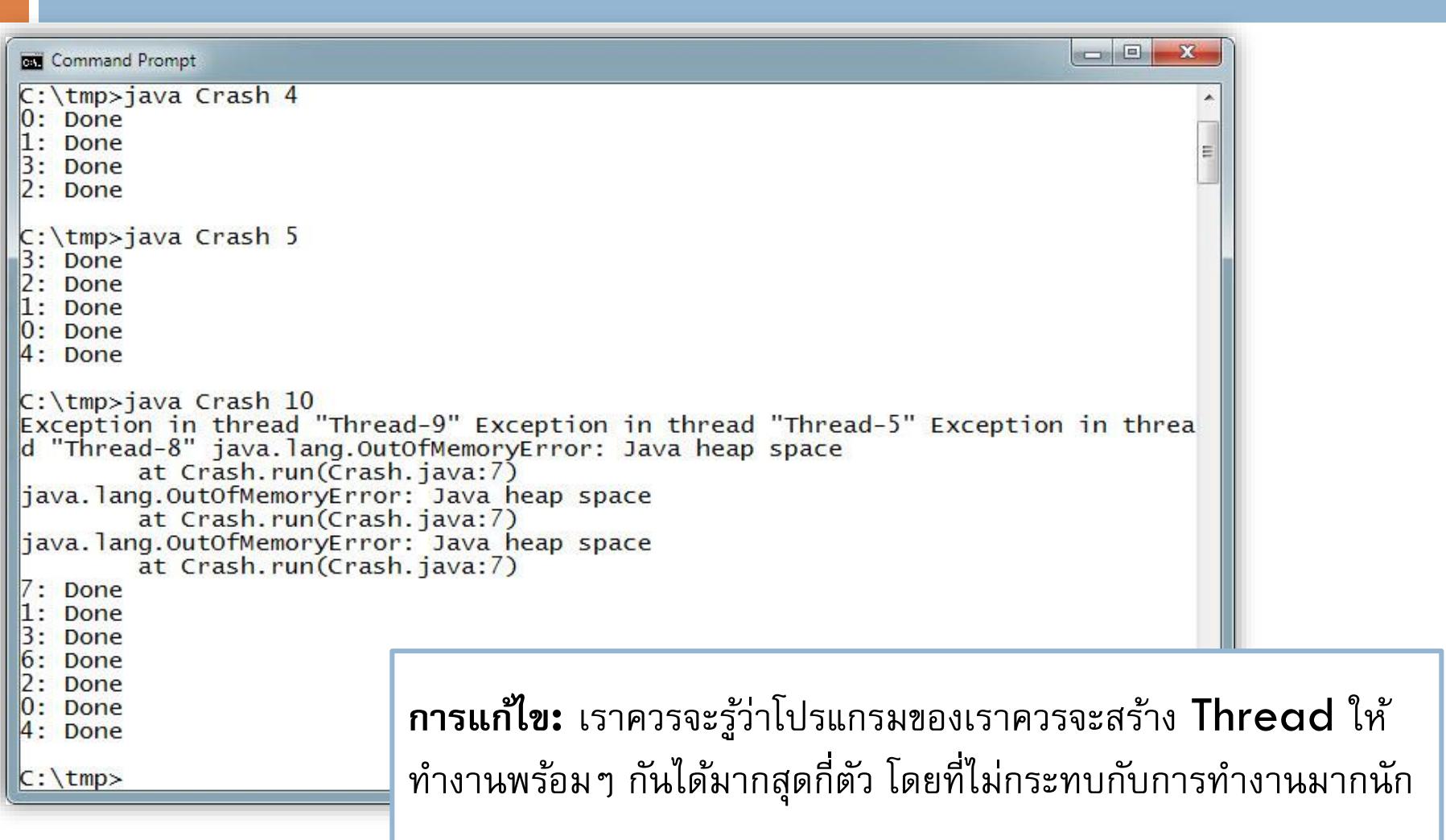
Thread แต่ละตัวจะใช้จองเนื้อ ที่หน่วยความจำ **50 MB**

จำลองให้ทำงาน **2** วินาที



```
public class Crash implements Runnable {  
    int id;  
  
    public Crash(int id) { this.id = id; }  
  
    public void run() {  
        int buffer[] = new int[50000000];  
        try {  
            Thread.sleep(2000);  
        } catch(Exception e) { }  
        System.out.println(id + ": Done");  
    }  
  
    public static void main(String[] args) {  
        int n = Integer.parseInt(args[0]);  
        for(int i = 0; i < n; i++) {  
            Crash c = new Crash(i);  
            Thread t = new Thread(c);  
            t.start();  
        }  
    }  
}
```

ปัญหาที่ 1: เรียกใช้งานโปรแกรม



The screenshot shows a Windows Command Prompt window titled "Command Prompt". The window contains the following text output:

```
C:\tmp>java Crash 4
0: Done
1: Done
3: Done
2: Done

C:\tmp>java Crash 5
3: Done
2: Done
1: Done
0: Done
4: Done

C:\tmp>java Crash 10
Exception in thread "Thread-9" Exception in thread "Thread-5" Exception in thread "Thread-8" java.lang.OutOfMemoryError: Java heap space
        at Crash.run(Crash.java:7)
java.lang.OutOfMemoryError: Java heap space
        at Crash.run(Crash.java:7)
java.lang.OutOfMemoryError: Java heap space
        at Crash.run(Crash.java:7)

7: Done
1: Done
3: Done
6: Done
2: Done
0: Done
4: Done

C:\tmp>
```

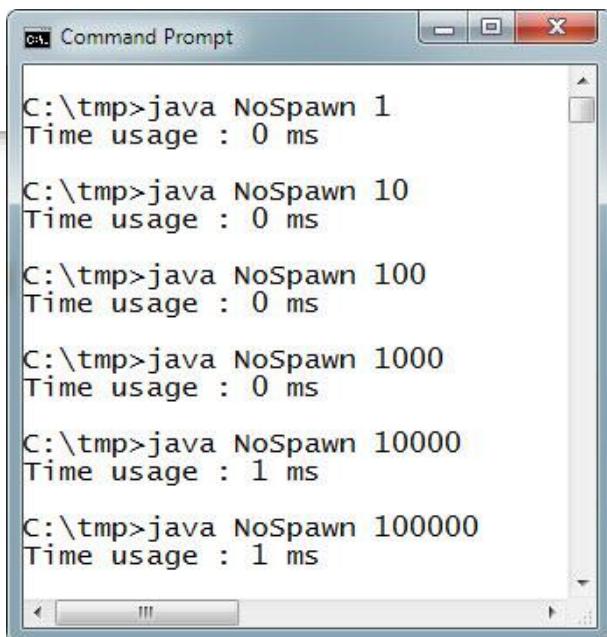
การแก้ไข: เราชาระวังว่าโปรแกรมของเราควรจะสร้าง **Thread** ให้ทำงานพร้อมๆ กันได้มากสุดกี่ตัว โดยที่ไม่กระทบกับการทำงานมากนัก

ปัญหาที่ 2 : ในการใช้งาน Thread

```
public class Spawn implements Runnable {  
    public void run() { }  
  
    public static void main(String[] args) {  
        int n = Integer.parseInt(args[0]);  
        long startTime = System.currentTimeMillis();  
        for(int i = 0; i < n; i++) {  
            Spawn s = new Spawn();  
            Thread t = new Thread(s);  
            t.start();  
            try {  
                t.join();  
            } catch(Exception e) { }  
        }  
        long stopTime = System.currentTimeMillis();  
        System.out.println("Time usage : " + (stopTime - startTime) + " ms");  
    }  
}
```

ปัญหาที่ 2: เรียกใช้งานโปรแกรม

```
public class NoSpawn {  
  
    public static void main(String[] args) {  
        int n = Integer.parseInt(args[0]);  
        long startTime = System.currentTimeMillis();  
        for(int i = 0; i < n; i++) { }  
        long stopTime = System.currentTimeMillis();  
        System.out.println("Time usage : " + (stopTime - startTime) + " ms");  
    }  
}
```



C:\tmp>java NoSpawn 1
Time usage : 0 ms

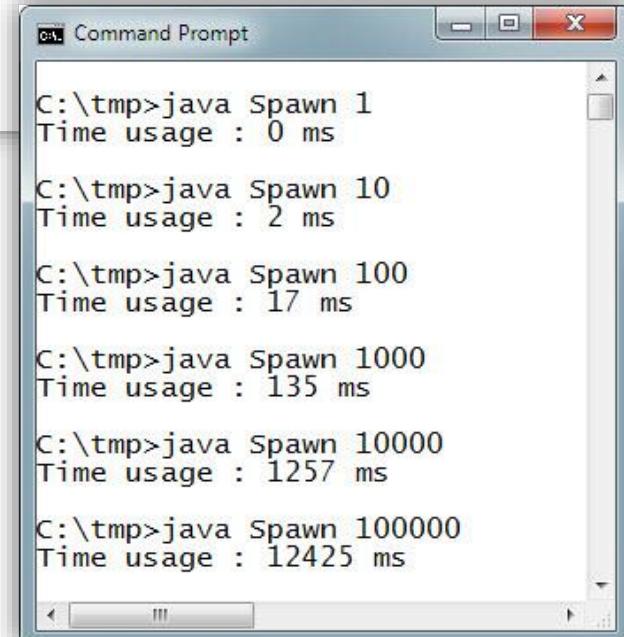
C:\tmp>java NoSpawn 10
Time usage : 0 ms

C:\tmp>java NoSpawn 100
Time usage : 0 ms

C:\tmp>java NoSpawn 1000
Time usage : 0 ms

C:\tmp>java NoSpawn 10000
Time usage : 1 ms

C:\tmp>java NoSpawn 100000
Time usage : 1 ms



C:\tmp>java Spawn 1
Time usage : 0 ms

C:\tmp>java Spawn 10
Time usage : 2 ms

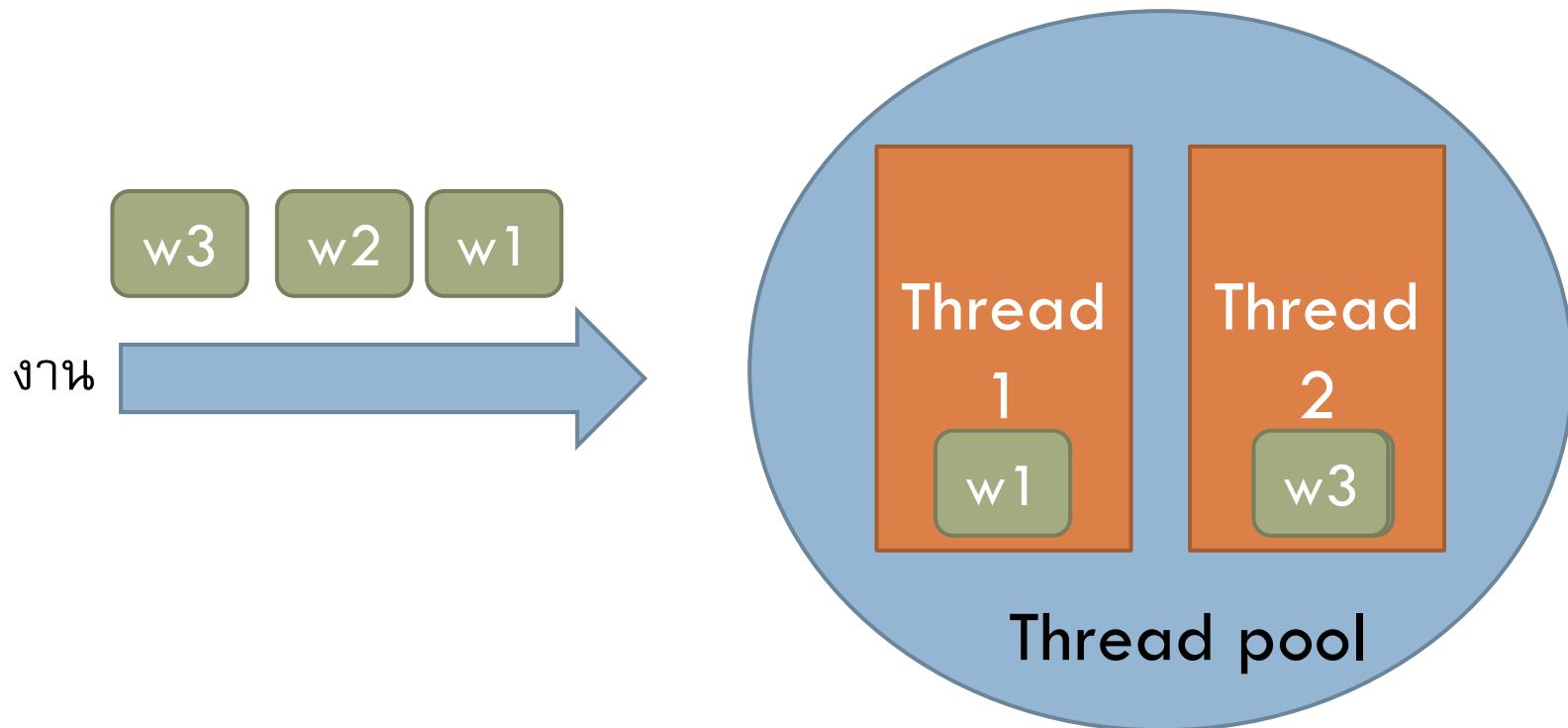
C:\tmp>java Spawn 100
Time usage : 17 ms

C:\tmp>java Spawn 1000
Time usage : 135 ms

C:\tmp>java Spawn 10000
Time usage : 1257 ms

C:\tmp>java Spawn 100000
Time usage : 12425 ms

การแก้ไขปัญหา ด้วย Thread Pool

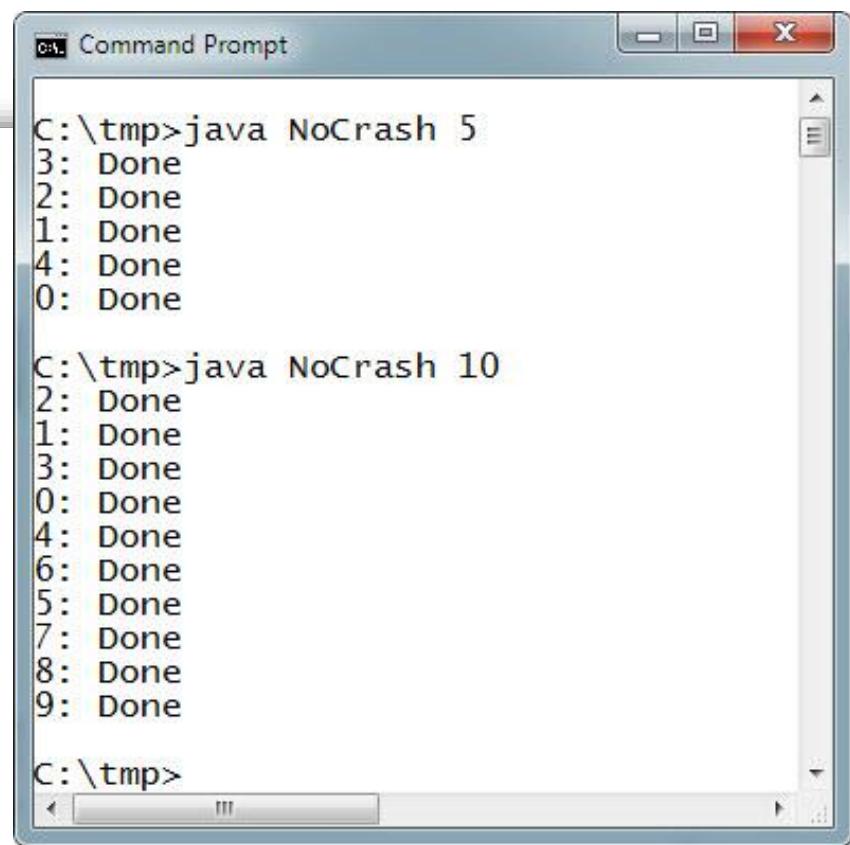


Java ใจดีมี Class มาให้ใช้งาน

- มี 2 class ที่ทำงานร่วมกันเพื่อทำงานในรูปแบบของ Thread Pool
 - ▣ **java.util.concurrent.Executors**
 - เป็น class ที่มี static method สำหรับสร้าง ExecutorService
 - เมธอด newFixedThreadPool(int nThread) สำหรับสร้าง ThreadPool ที่มี thread ภายในการทำงานจำนวน nThread ตัว
 - เมธอด newSingleThreadExecutor() สำหรับสร้าง Threadpool ที่มี Thread เดียว
 - ▣ **java.util.concurrent.ExecutorService**
 - เป็น class ที่ทำหน้าที่เป็น ThreadPool
 - เมธอด execute(Runnable r) ใช้สำหรับงาน r เข้าไปเข้าคิวของการทำงาน
 - เมธอด shutdown() เมื่อต้องการหยุดการทำงานให้บริการ
 - เมธอด isTerminated() ตรวจสอบว่า ExecutorService ได้ปิดการทำงานเสร็จแล้วหรือไม่

ແກ້ໄຂປົງກາທີ 1

```
import java.util.concurrent.*;  
  
public class NoCrash implements Runnable {  
    int id;  
  
    public NoCrash(int id) { this.id = id; }  
  
    public void run() {  
        int buffer[] = new int[50000000];  
        try {  
            Thread.sleep(2000);  
        } catch(Exception e) { }  
        System.out.println(id + ": Done");  
    }  
  
    public static void main(String[] args) {  
        int n = Integer.parseInt(args[0]);  
        ExecutorService es = Executors.newFixedThreadPool(5);  
        for(int i = 0; i < n; i++) {  
            NoCrash c = new NoCrash(i);  
            es.execute(c);  
        }  
        es.shutdown();  
    }  
}
```



The screenshot shows a Windows Command Prompt window titled "Command Prompt". It displays the output of two separate Java executions. The first execution, "java NoCrash 5", shows five threads (0, 1, 2, 3, 4) all reporting "Done". The second execution, "java NoCrash 10", shows ten threads (0 through 9) all reporting "Done". This demonstrates that the code does not crash even when multiple threads are executed simultaneously.

```
C:\tmp>java NoCrash 5  
3: Done  
2: Done  
1: Done  
4: Done  
0: Done  
  
C:\tmp>java NoCrash 10  
2: Done  
1: Done  
3: Done  
0: Done  
4: Done  
6: Done  
5: Done  
7: Done  
8: Done  
9: Done  
  
C:\tmp>
```

แก้ไขปัญหาที่ 2

```
import java.util.concurrent.*;  
  
public class SpawnThreadPool implements Runnable {  
    public void run() { }  
  
    public static void main(String[] args) {  
        int n = Integer.parseInt(args[0]);  
        long startTime = System.currentTimeMillis();  
  
        ExecutorService executor = Executors.newSingleThreadExecutor();  
        for(int i = 0; i < n; i++) {  
            SpawnThreadPool s = new SpawnThreadPool();  
            executor.execute(s);  
        }  
        executor.shutdown();  
        while(!executor.isTerminated()) {}  
        long stopTime = System.currentTimeMillis();  
        System.out.println("Time usage : " + (stopTime - startTime) + " ms");  
    }  
}
```

ผลการรันเปรียบเทียบ

```
cmd Command Prompt
C:\tmp>java Spawn 1
Time usage : 0 ms

C:\tmp>java Spawn 10
Time usage : 2 ms

C:\tmp>java Spawn 100
Time usage : 17 ms

C:\tmp>java Spawn 1000
Time usage : 135 ms

C:\tmp>java Spawn 10000
Time usage : 1257 ms

C:\tmp>java Spawn 100000
Time usage : 12425 ms
```

```
cmd Command Prompt
C:\tmp>java SpawnThreadPool 1
Time usage : 5 ms

C:\tmp>java SpawnThreadPool 10
Time usage : 4 ms

C:\tmp>java SpawnThreadPool 100
Time usage : 5 ms

C:\tmp>java SpawnThreadPool 1000
Time usage : 8 ms

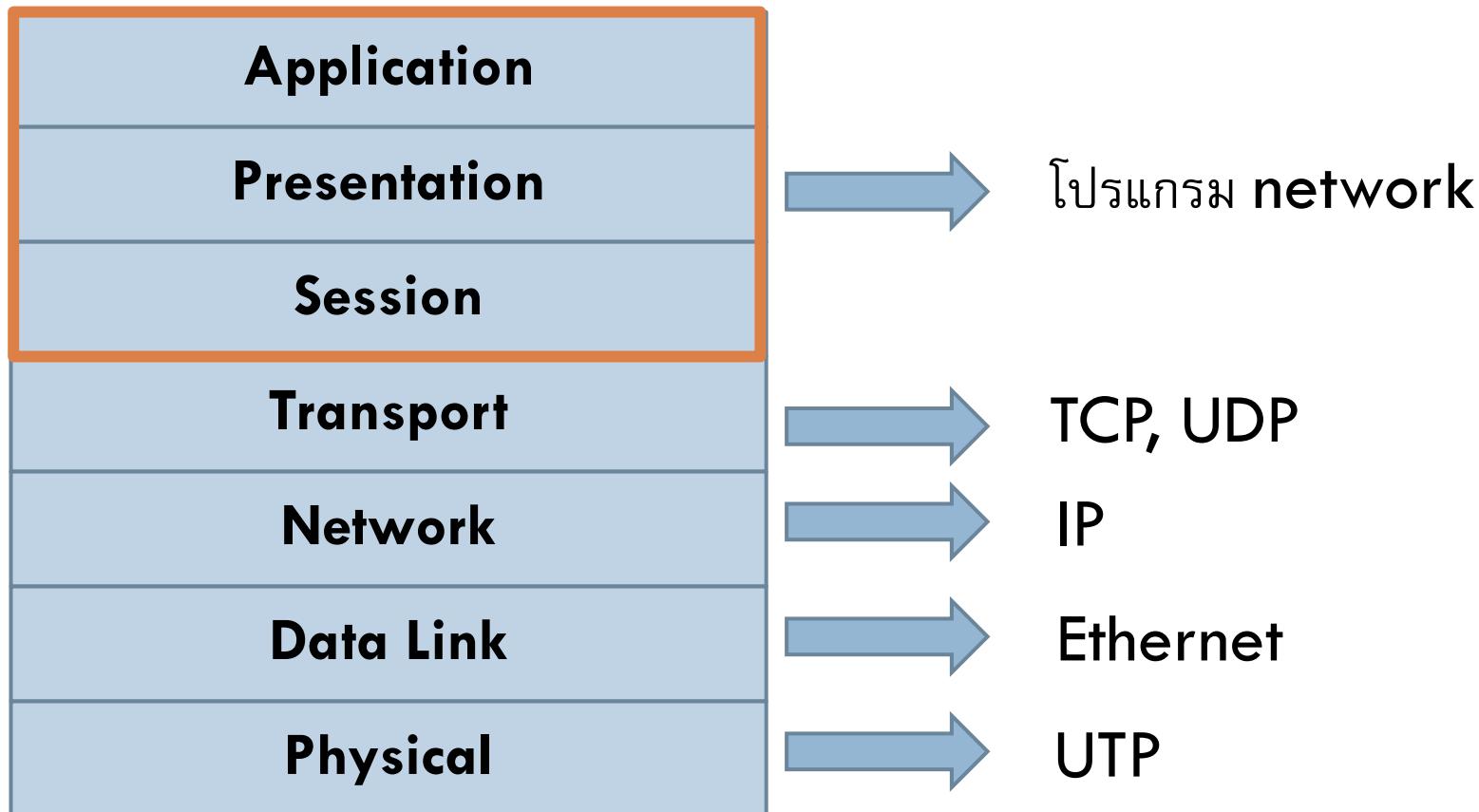
C:\tmp>java SpawnThreadPool 10000
Time usage : 22 ms

C:\tmp>java SpawnThreadPool 100000
Time usage : 76 ms
```

LOOKING UP IP

030523313 - Network programming
Asst. Prof. Dr. Choopan Rattanapoka

ทบทวน OSI layer

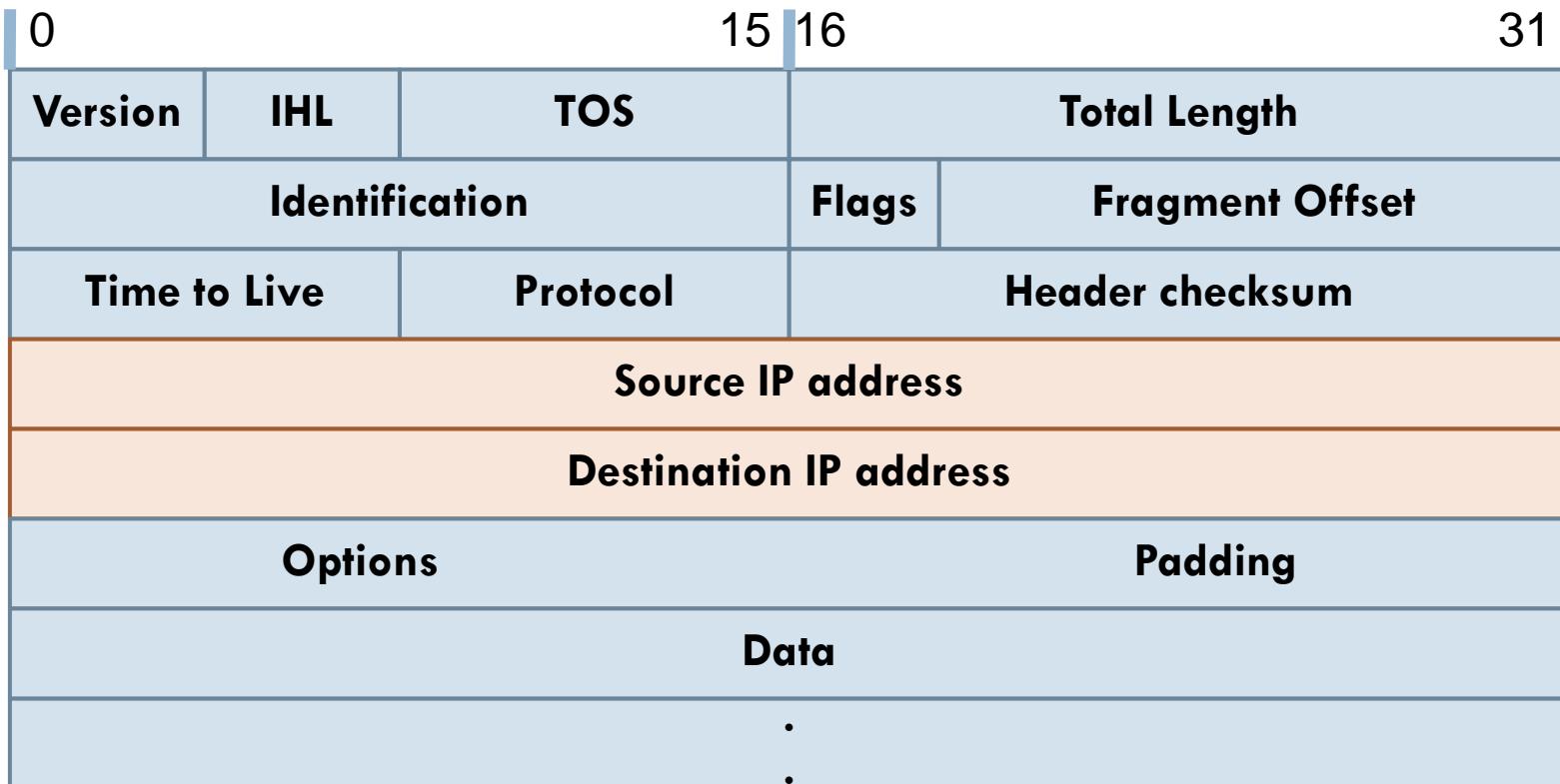


Ethernet Header

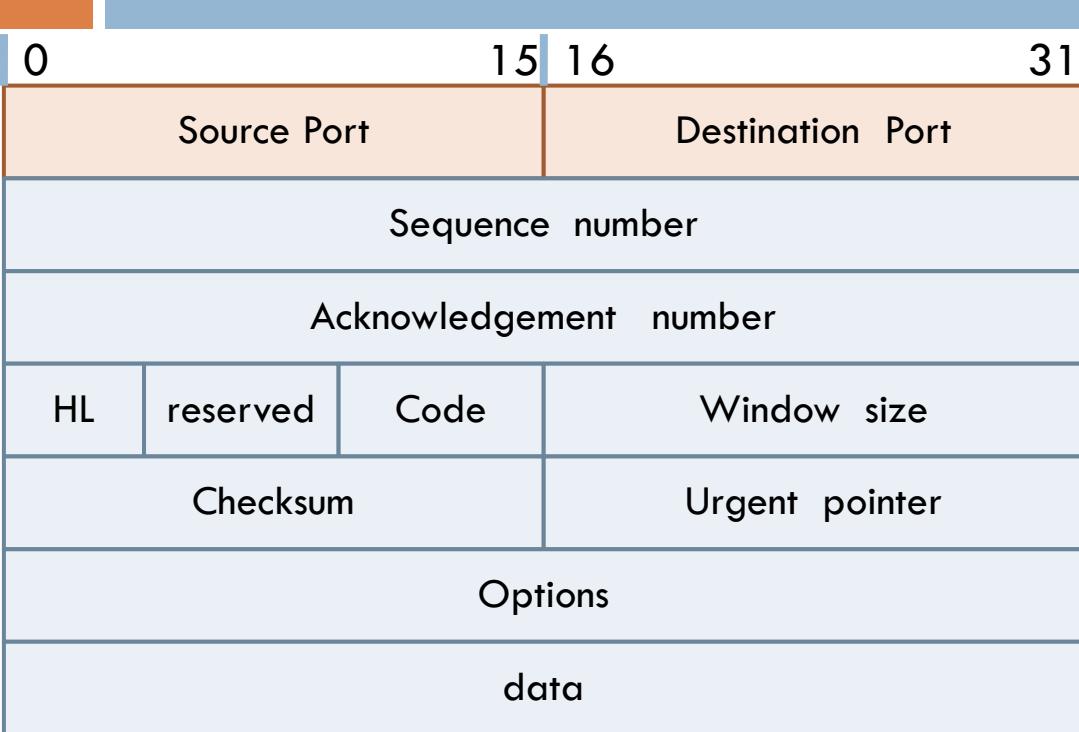


- **Preamble** (8 bytes)
- **Dst MAC** : MAC address ของเครื่องปลายทาง (6 bytes)
- **Src MAC** : MAC address ของเครื่องต้นทาง (6 bytes)
- **Type** (2 bytes) : ใช้บอก protocol ของชั้น network
 - 0x0800 Internet Protocol, Version 4 ([IPv4](#))
 - 0x0806 Address Resolution Protocol ([ARP](#))
 - 0x8035 Reverse Address Resolution Protocol ([RARP](#))
 - 0x8137 Novell [IPX](#) (alt)
 - 0x86DD Internet Protocol, Version 6 ([IPv6](#))

IP header



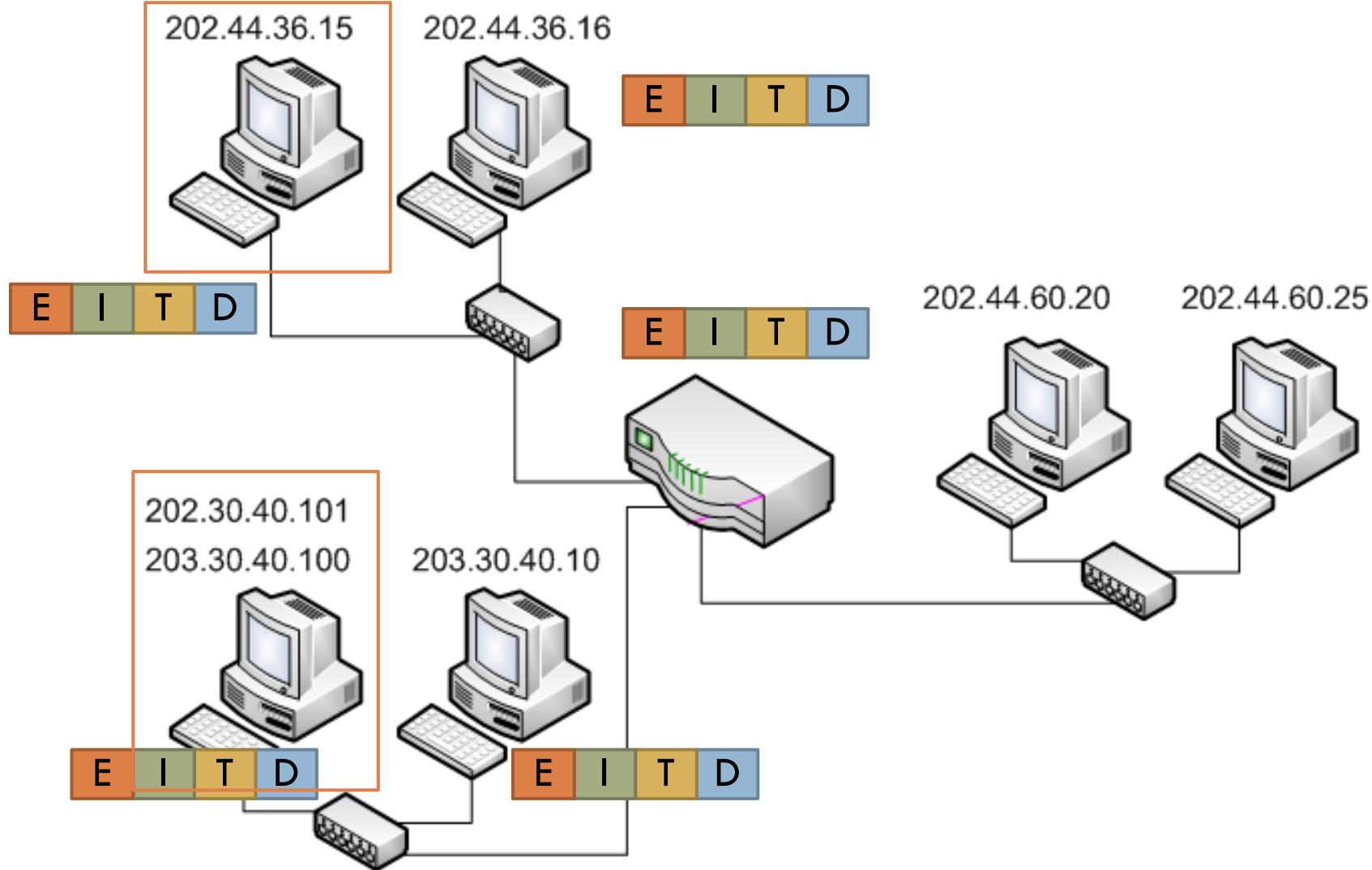
TCP and UDP Header



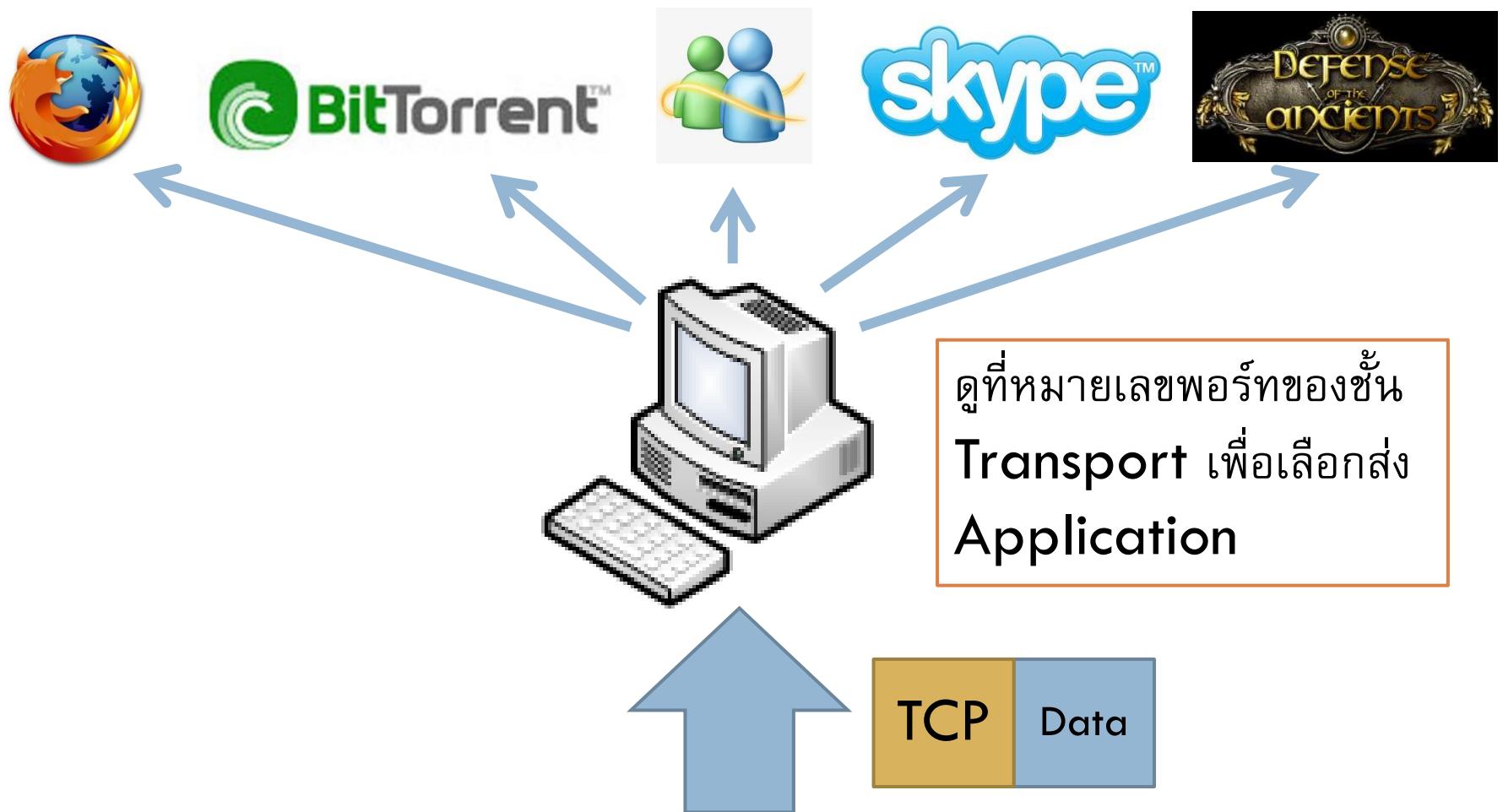
TCP header



ตัวอย่างการส่งข้อมูลในระบบเครือข่าย



การใช้งานของพอร์ท



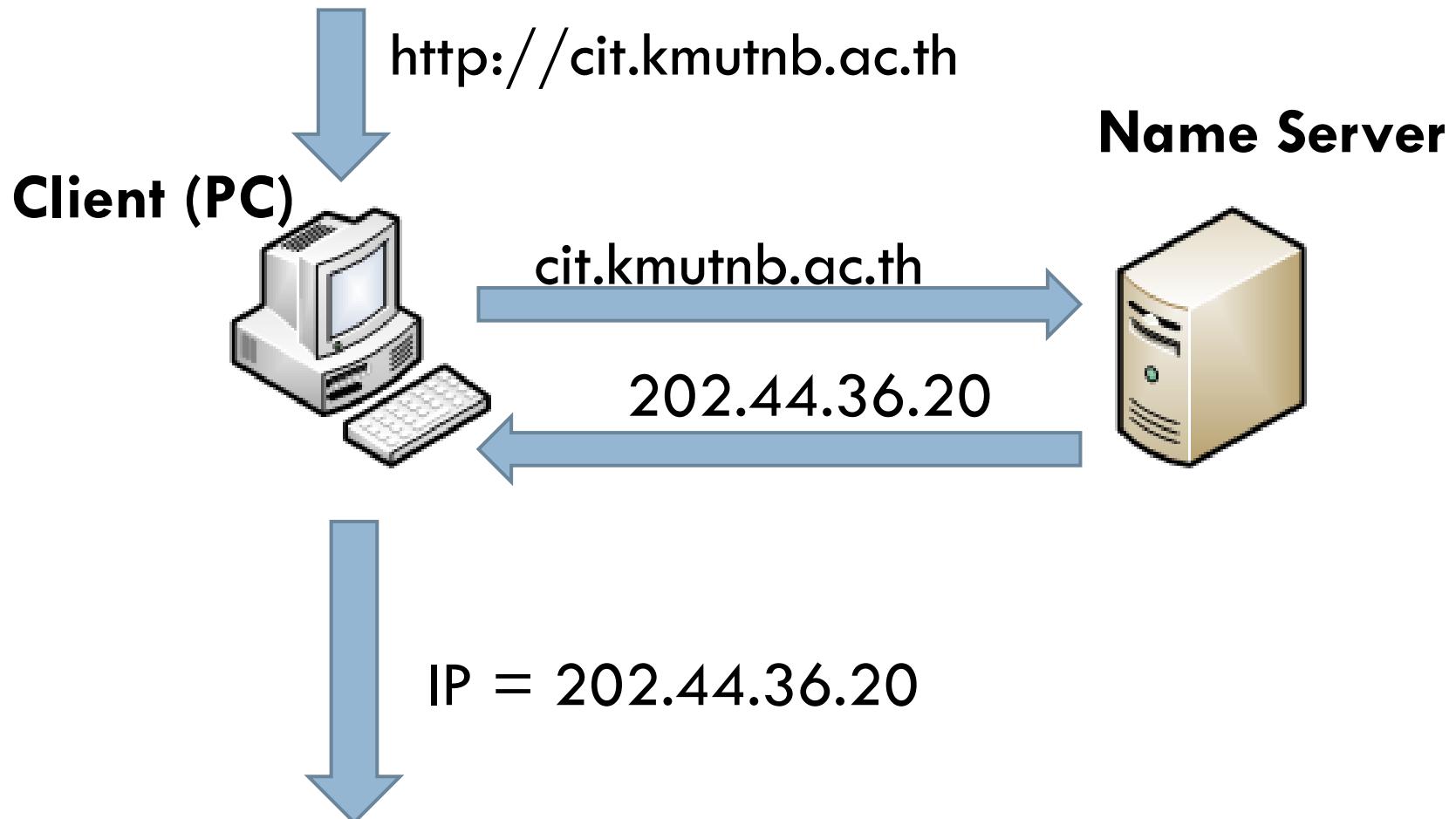
หมายเลขพอร์ท

- Port จะเป็นจำนวนเต็มขนาด 16 bits (1-65535)
- Port หมายเลข 1 – 1023 ถูกสงวนไว้เพื่อการของระบบ เรียกว่า “well-known ports”
- ตัวอย่างของ Port 1-1023 ที่ถูกสงวนไว้เพื่อการต่างๆ เช่น
 - **Port 20,21** : FTP (File Transfer Protocol)
 - **Port 23** : Telnet
 - **Port 25** : SMTP (Simple Mail Transfer Protocol)
 - **Port 53** : Domain
 - **Port 80** : HTTP (HyperText Transfer Protocol)
 - **Port 110** : POP3 (Post Office Protocol version 3)
- Well-known ports ถูกกำหนดไว้ในไฟล์
 - **Windows** -> C:\WINDOWS\system32\drivers\etc
 - **Linux** -> /etc/service

IP Address

- IP Address เปรียบเสมือนเลขที่บ้านของเครื่องคอมพิวเตอร์
- เมื่อมีการเชื่อมต่อกันระหว่างเครื่องจำเป็นต้องใช้หมายเลข IP เพื่อระบุตัวตน
- ในทางปฏิบัติการจำหมายเลข IP เพื่อติดต่อไปเครื่องอื่นนั้นไม่สะดวก จึงมีบริการที่ชื่อ DNS (Domain name system) เข้ามาช่วย
- DNS ทำหน้าที่แปลงจากชื่อ host ให้เป็น IP และแปลงจากหมายเลข IP ให้เป็นชื่อ host

การทำงานของ DNS



การหาหมายเลข IP ด้วยภาษา Java

- จะใช้ class InetAddress ซึ่งมี static เมธอดคือ
 - ▣ **public static InetAddress getByName(String hostname)**
throws UnknownHostException
 - คืน object ของ InetAddress ที่เก็บข้อมูลของ hostname
 - ▣ **public static InetAddress[] getAllByName(String hostname)** **throws UnknownHostException**
 - คืน array object ของ InetAddress ที่เก็บข้อมูลของ hostname
 - ▣ **public static InetAddress getLocalHost() throws UnknownHostException**
 - คืน object ของ InetAddress ที่เก็บข้อมูลของ localhost

InetAddress

- ใน class InetAddress เองมีเมธอดที่ใช้ดึงข้อมูลออกมากคือ
 - public String getHostName()
 - ใช้สำหรับดึงชื่อ host
 - public byte[] getAddress()
 - ใช้สำหรับดึง IP address ซึ่งอยู่ในรูปของ byte array
 - public String getHostAddress()
 - ใช้สำหรับดึง IP address ในรูปของ String

ตัวอย่าง 1

```
import java.net.*;  
public class TestInet {  
    public static void main(String[ ] args) {  
        try {  
            InetAddress ad = InetAddress.getByName("ect.cit.kmutnb.ac.th");  
            System.out.println("IP = " + ad.getHostAddress());  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

តាមរយៈ 2

```
import java.net.*;  
public class TestInet2 {  
    public static void main(String[ ] args) {  
        try {  
            InetAddress[ ] ad =  
                InetAddress.getAllByName("www.google.com");  
            for(int i = 0; i < ad.length; i++) {  
                System.out.println("IP = " + ad[i].getHostAddress());  
            }  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

តាមរយៈ 3

```
import java.net.*;  
public class TestInet2 {  
    public static void main(String[ ] args) {  
        try {  
            InetAddress ad = InetAddress.getLocalHost();  
            System.out.println("Host = " + ad.getHostName());  
            System.out.println("IP = " + ad.getHostAddress());  
        }  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

ตัวอย่าง 4

```
import java.net.*;  
  
public class TestInet {  
    public static void main(String[ ] args) {  
        try {  
            InetAddress ad = InetAddress.getByName("202.44.32.13");  
            System.out.println("host = " + ad.getHostName());  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Socket

- **Socket** หรือ **Socket Address** หมายถึง คู่ของ **IP Address** กับ **Port**
- เครื่องแต่ละเครื่องจะมี IP ไม่ซ้ำกัน
- ในเครื่องหนึ่งเครื่องแต่ละ **Application** จะใช้พอร์ตไม่ซ้ำกัน
- **Socket** เมื่อมองในแง่ของการเขียนโปรแกรม อาจมองได้ว่าเป็นท่อเชื่อมระหว่างเครื่อง 2 เครื่องที่จะติดต่อสื่อสารกัน

Class Socket

- ในภาษา Java มี Class ชื่อ **Socket** เพื่อทำหน้าที่เชื่อมต่อเครื่องคอมพิวเตอร์เข้าด้วยกัน โดยใช้ TCP
- Class **Socket** มี constructor ออยู่หลายตัวแต่ตัวที่นิยมใช้คือ
 - **host** ชื่อเครื่องหรือหมายเลข IP ของเครื่องปลายทาง
 - **port** หมายเลขพอร์ตของเครื่องปลายทาง
 - **UnknownHostException** ถ้า host ที่ใส่ไม่มีอยู่จริง
 - **IOException** เมื่อมีปัญหาในการเชื่อมต่อ

ตัวอย่างการเชื่อมต่อเครื่องปลายทาง

```
1 import java.io.*;
2 import java.net.*;
3
4 public class TestJava {
5     public static void main(String[] args) {
6         try {
7             Socket socket = new Socket("127.0.0.1", 80);
8         }
9         catch (UnknownHostException unknownEx) {
10             unknownEx.printStackTrace();
11         }
12         catch (IOException ioEx) {
13             ioEx.printStackTrace();
14         }
15     }
16 }
```

ดักจับ Exception

```
1 import java.io.*;
2 import java.net.*;
3
4 public class TestJava {
5     public static void main(String[] args) {
6         try {
7             Socket socket = new Socket("127.0.0.1", 80);
8         }
9         catch (Exception e) {
10             e.printStackTrace();
11         }
12     }
13 }
```

NETWORK APPLICATION MODELS

- CLIENT/SERVER MODEL

- TELNET

- SMTP

- HTTP

- PEER-TO-PEER MODEL

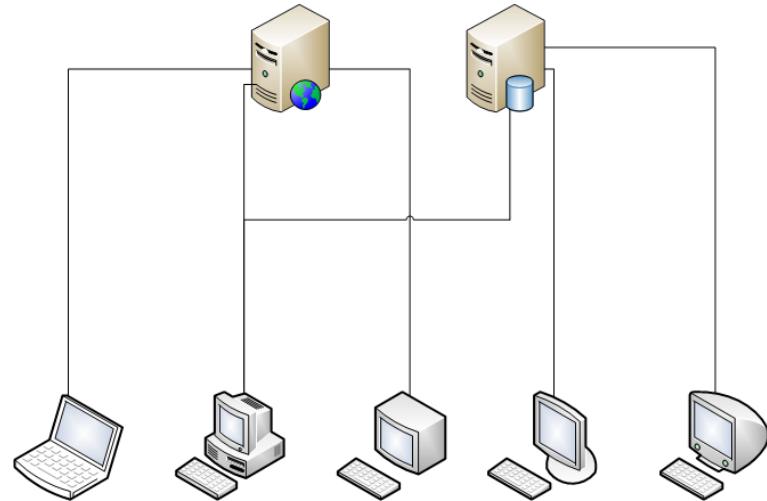
JAVA SOCKET

การโปรแกรมระบบเครือข่าย

- ในการพัฒนาโปรแกรมระบบเครือข่ายสามารถแบ่งออกเป็น 2 สถาปัตยกรรมใหญ่ๆ คือ
 - ▣ **Client/Server Architecture**
 - ▣ **Peer to Peer (P2P) Architecture**
- ในการพัฒนาโปรแกรมแบบ **Client/Server Architecture** จะประกอบไปด้วย โปรแกรม 2 ประเภท
 - ▣ **Client** → ผู้ขอใช้บริการ และ
 - ▣ **Server** → ผู้ให้บริการ

Client/Server Architecture

- หน้าที่การทำงานของโปรแกรม **Client**
 - จะเป็นผู้เริ่มต้นติดต่อ **Server** เพื่อขอใช้บริการ
 - รอการตอบรับจาก **Server** เพื่อเริ่มใช้บริการ
- หน้าที่การทำงานของโปรแกรม **Server**
 - รอการขอใช้บริการจาก **Client**
 - เมื่อ **Client** ขอใช้บริการ และ **Server** พร้อมที่จะให้บริการจะตอบกลับหา **Client** เพื่อเริ่มให้บริการ
- ในสถาปัตยกรรมแบบ **Client/Server** โปรแกรม **Client** จะเชื่อมต่อและแลกเปลี่ยนข้อมูลกับ **Server** เท่านั้น โปรแกรม **Client** จะไม่สามารถเชื่อมต่อและแลกเปลี่ยนข้อมูลกับ **Client** อื่นๆ ได้



ข้อดีและข้อเสียของ Client/Server Architecture

□ ข้อดี

- เมื่อบริการต่างๆที่ **Client** จะขอใช้งานอยู่ที่ **Server** ทำให้การบำรุงรักษา เช่น **upgrade** โปรแกรม, เปลี่ยนอุปกรณ์ สามารถทำที่ **Server** อย่างเดียวไม่จำเป็นต้องไปยุ่งกับเครื่อง **Client**
- ในการเก็บข้อมูลถ้าข้อมูลถูกเก็บใน **Server** ทั้งหมดแล้ว **Server** สามารถตั้งความปลอดภัยในการเข้าถึงข้อมูลนั้นๆ ได้
- ง่ายต่อผู้ดูแลระบบในการตรวจสอบการให้บริการของโปรแกรมระบบเครือข่าย
- **Server** สามารถตั้งความปลอดภัยในการให้บริการต่างๆได้ง่าย

□ ข้อเสีย

- ภาระการทำงานตกอยู่กับเครื่อง **Server** เป็นส่วนใหญ่
- เมื่อ **Server** มีปัญหาอาจทำให้ผู้ใช้งานทั้งระบบเครือข่ายไม่สามารถทำงานได้

Peer-to-Peer Architecture

- ส่วนใหญ่จะเขียนย่อว่า P2P
- จะเรียกแทนระบบโปรแกรมของเครือข่าย ที่เครื่องทุกเครื่องมีหน้าที่ในการทำงานเท่าเทียมกัน คือ เป็นได้ทั้ง เชิฟเวอร์ และ โคลอีน ในแต่ละช่วงเวลา
- บางครั้ง P2P จะหมายถึงระบบที่มีการเปลี่ยนแปลงตลอดเวลาโดยไม่มีผลกระทบต่อระบบรวม
- ถูกประยุกต์ใช้ในช่วงเวลาที่ผ่านมาไม่นานหลังจาก **internet** ได้รับความนิยม เนื่องจากในการทำงานในระบบ **internet** บางอย่างไม่สามารถที่จะหาผู้รับผิดชอบ **server** ได้

ประเภทของ Peer-to-Peer

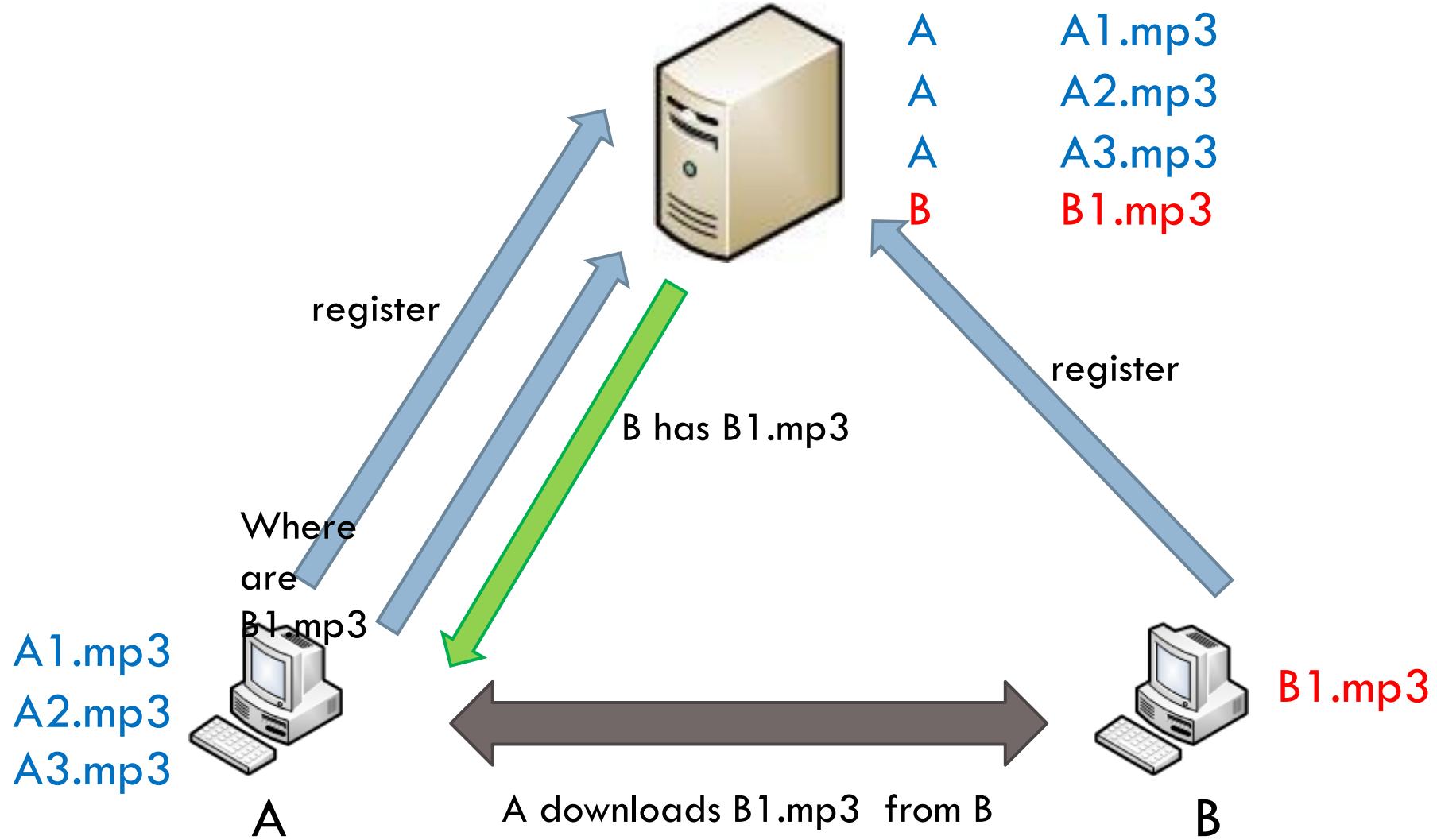
- P2P แบ่งออกเป็น 2 พากใหญ่ๆ คือ
 - ▣ **Centralized** แบบรวมศูนย์กลาง
 - ▣ **Decentralized** แบบกระจาย ซึ่งแบ่งออกอีก 3 ประเภทคือ
 - **Unstructured** แบบไร้โครงสร้าง
 - **Structured** แบบมีโครงสร้าง
 - **Hybrid** แบบผสม

Peer-to-Peer Architecture

- Application แรกๆที่ทำให้สถาปัตยกรรมแบบ P2P โดยดังก็คือ การแชร์ไฟล์ (file sharing)
- ซึ่งเริ่มต้นที่ประมาณปี 1999 ได้มีการพัฒนาโปรแกรมที่ทำการแชร์ไฟล์ mp3 โดยเป็นแบบ **centralized** ชื่อว่า napster
- Napster
 - เป็นโปรแกรมที่พัฒนาเพื่อแลกเปลี่ยนไฟล์ mp3 ของผู้ใช้ตามบ้านทั่วไป
 - Napster ถือว่าเป็น P2P แบบ **centralized** เพราะจำเป็นต้องมี server ที่ทำหน้าที่ เมื่อൺสมุดหน้าเหลือง ที่เก็บตำแหน่งที่อยู่ของไฟล์ mp3



การทำงานของ Napster



ข้อดีและข้อเสียของระบบ Napster

□ ข้อดี

- กระจายเนื้อที่ในการเก็บ mp3 และ bandwidth เพราะใครจะออกเงินซื้อ server ที่มี harddisk ขนาดใหญ่เพื่อจุ mp3 และเช่าสายสัญญาณ internet เพื่อรับรองรับคนทั่วโลก
- ระบบทำงานเร็ว เพราะ server แค่ค้นหา IP address หรือชื่อโฮทส์ที่มี mp3 ที่ต้องการ

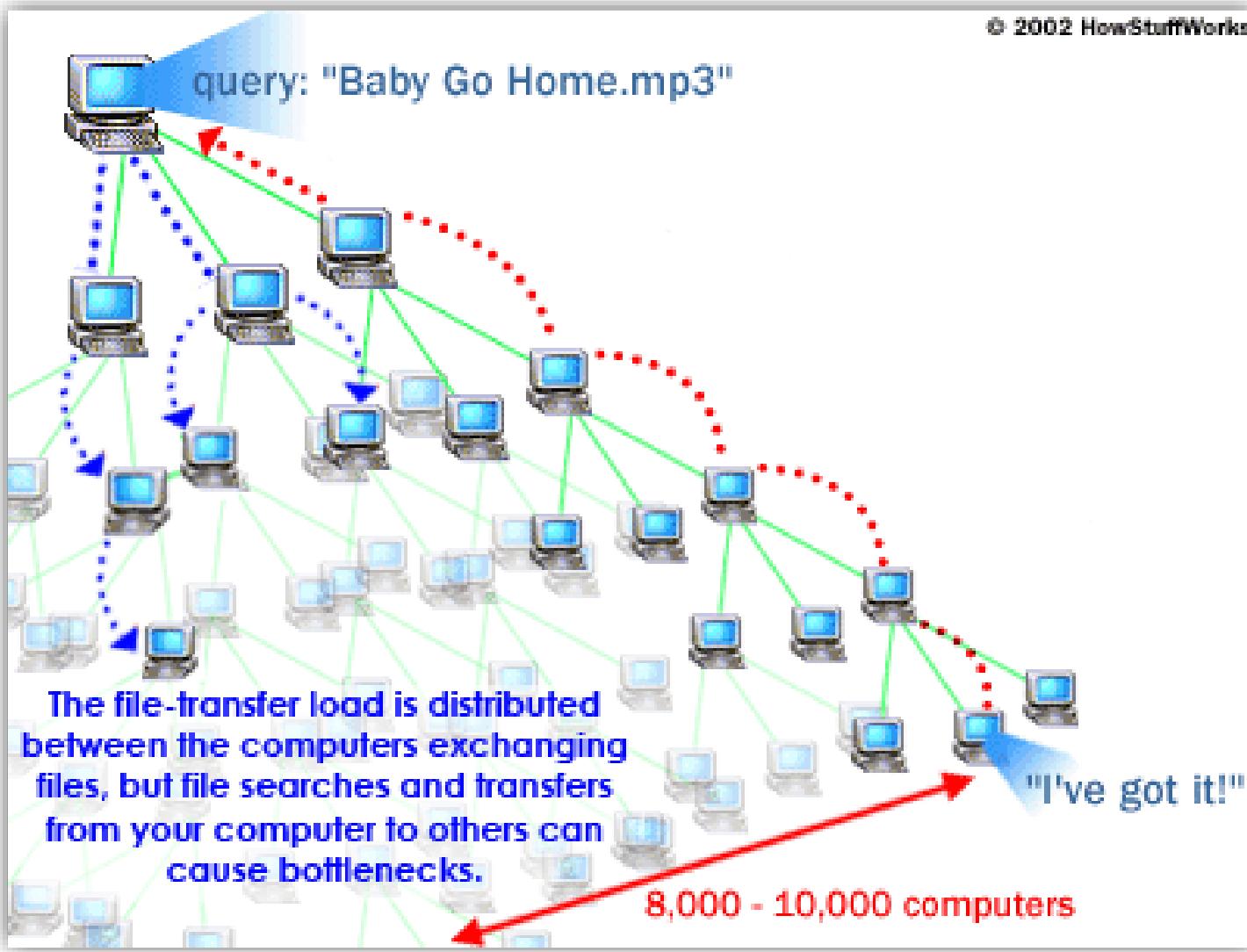
□ ข้อเสีย

- ผู้ใช้จำเป็นต้องติดต่อกับ server เพื่อจะทราบว่าจะ download ที่ไหน ถ้า server มีปัญหา ก็จะทำให้ระบบล่มทั้งระบบ
- จากข้อเสียข้อนี้ทำให้ napster หยุดการให้บริการเพราะศาลสั่งให้ปิดเซิฟเวอร์เนื่องจาก กฎหมาย copyright ของ mp3

Gnutella

- หลังจาก Napster ถูกปิด ทางออกใหม่สำหรับการแชร์ไฟล์คือ การพัฒนาระบบ P2P แบบ **decentralized** ซึ่งไม่จำเป็นต้องมี server
- Gnutella เป็น 1 ใน application ต้นๆ ที่ได้ใช้ระบบ decentralized P2P ถูกพัฒนาขึ้นประมาณปี 2001
- Gnutella เป็น **decentralized P2P system** แบบ **unstructured**
- การค้นหาข้อมูลใช้วิธีการที่เรียกว่า **Flooding**

Gnutella



Gnutella

□ ข้อดี

- กระจายเนื้อที่ในการเก็บ mp3 และ **bandwidth** เพราะโครงจะออกเงินซื้อ **server** ที่มี **harddisk** ขนาดใหญ่เพื่อจุ mp3 และเชื่อมต่อสัญญาณ **internet** เพื่อรับรองรับคนทั่วโลก
- ไม่มี **server** ที่เก็บข้อมูลกลางทำให้ไม่มีโครงสามารถปิดระบบได้

□ ข้อเสีย

- วิธีการ **flood** อาจทำให้เครื่องผู้ใช้งานทำงานมากกว่าปกติ
- ไม่มีการรับประกันว่าข้อมูลที่ถูกหาจะสามารถหาเจอ แม้ว่าข้อมูลนั้นจะมีอยู่จริงในระบบ
- การค้นหาข้อมูลจะช้ากว่าแบบ **centralized** ภายหลังจึงมีการเพิ่มส่วนที่เรียกว่า **supernode** เพื่อเก็บ **cache**

CHORD, PASTRY

- ในด้านการวิจัยได้พัฒนาระบบ P2P แบบ **decentralized** ที่มีโครงสร้างขึ้น (**structured**)
- โครงสร้างพื้นฐานจะเป็นรูปวงแหวน (ring) โดยแต่ละ node จะมี **nodeID**
- วัตถุที่ต้องการจะเก็บเข้าในระบบ จะมี **objectID**
- ระบบจะพยายามวาง **objectID** ลงใน **nodeID** ที่มีค่าใกล้เคียงกันที่สุด
- ทำให้การค้นหาข้อมูลสามารถหาได้ง่ายและรวดเร็ว

Bittorrent

- เป็นการผสมผสานระบบแบบ **centralized** และ **decentralized** เรียกว่า **hybrid**
- ใน **bittorrent** จะมี **tracker** ที่ทำหน้าที่เหมือนเป็น **server** ที่เก็บ **IP address** ของเครื่องที่กำลังแชร์ไฟล์นั้นอยู่
- **Client** จะติดต่อกับ **tracker** เพื่อขอ **IP address** ของเครื่องที่แชร์ไฟล์นั้นๆ แล้วติดต่อกับ **client** อื่นเพื่อ **download** ไฟล์ข้อมูลนั้น
- ได้แพร่ระบบ **tit-and-tat** ยิงแชร์มากยิง **download** เร็วเพื่อบังกันปัญหาคนที่เอาแต่ **download** แต่ไม่ยอมแชร์คืนให้ระบบ

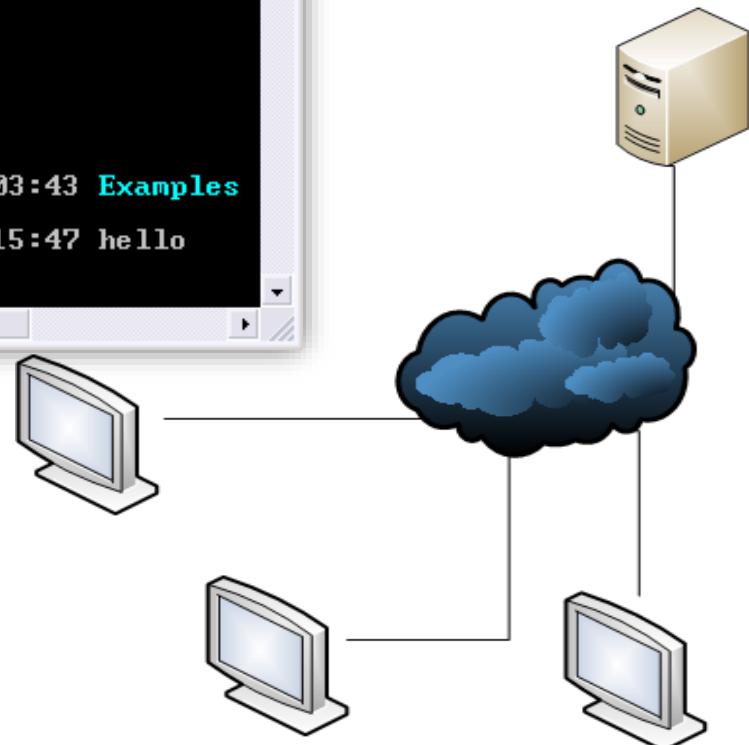
Telnet (**Telecommunication network**)

ประวัติ

- ก่อนที่ PC จะแพร่หลาย ผู้ใช้คอมพิวเตอร์จะใช้บริการผ่านทางเทอร์มินัลที่เรียกว่า **Dumb terminal**
- ที่ต่อเขื่อมตรงอยู่กับ Server โดยคำสั่งต่างๆ เมื่อพิมพ์จะถูกส่งไปหา Server โดยตรง
- Server จะทำหน้าที่ทั้งเก็บข้อมูล และประมวลผลคำสั่ง และส่งผลลัพธ์กลับมาหา terminal โดยจะไม่มีการประมวลผลคำสั่งและเก็บข้อมูลใน terminal
- เมื่อระบบเครือข่ายแพร่หลาย สถานีส่วนบุคคลมีความสามารถในการประมวลผลมากขึ้น แต่การขอเข้าไปทำงานใน Server ก็ยังมีการใช้งานอยู่จึงเป็นที่มาของ telnet ซึ่งจะหน้าที่เสมือน PC ต่อตรงอยู่กับ Server

รูปแบบการทำงานของ Telnet

```
Telnet 202.44.36.14
ect-student@ect-webserver:~$ ls -l
total 0
lrwxrwxrwx 1 ect-student ect-student 26 2008-11-12 03:43 Examples
example-content
ect-student@ect-webserver:~$ ls
Examples
ect-student@ect-webserver:~$ pwd
/home/ect-student
ect-student@ect-webserver:~$ cat /etc/dmesg
cat: /etc/dmesg: No such file or directory
ect-student@ect-webserver:~$ cat /etc/me
mediaprm      menu-methods/
ect-student@ect-webserver:~$ touch hello
ect-student@ect-webserver:~$ ls -l
total 0
lrwxrwxrwx 1 ect-student ect-student 26 2008-11-12 03:43 Examples
example-content
-rw-r--r-- 1 ect-student ect-student 0 2008-11-17 15:47 hello
ect-student@ect-webserver:~$
```



การใช้งาน Telnet

- การเชื่อมโยงด้วย Telnet จะทำโดย
 - เครื่อง Client ขอสถาปนาการเชื่อมต่อด้วย TCP กับเครื่อง Server ที่ port หมายเลข 23
 - การส่งข้อมูลจะอยู่ในรูป ASCII code
- ใน Windows และ Linux มีคำสั่ง telnet มาให้อยู่แล้ว และสามารถเรียกใช้งานได้เลย ด้วยคำสั่ง

telnet <ชื่อ telnet server> หรือ

telnet เพื่อเข้าสู่ telnet prompt

การใช้งาน Telnet (2)

- ในปัจจุบัน บริการ **Telnet** แทบจะไม่เปิดให้บริการแล้ว เนื่องจาก ข้อมูลที่ส่งผ่านเครือข่ายไม่มีการเข้ารหัส ทำให้ผู้บุกรุกระบบเครือข่ายสามารถถักข้อมูลที่ส่งผ่านกันในเครือข่าย และ ขโมยรหัสผ่านได้ เปลี่ยนมาใช้ **SSH** แทน
- **Telnet** สามารถจำลองการทำงานของ **TCP** ได้โดยการบังคับหมายเลข **port** เพื่อ debug application อื่นๆ ที่แลกเปลี่ยนข้อมูล เป็น **text** และใช้ **TCP** เช่น
 - **telnet mail.kmutnb.ac.th 25** คือการต่อเข้า **server** : **mail.kmutnb.ac.th** ที่ **port** หมายเลข 25 ด้วย **TCP**

SMTP (Simple Mail Transfer Protocol)

- เป็นบริการส่ง mail แบบง่ายๆ
- ให้บริการแบบ TCP ที่ port หมายเลข 25
- มีคำสั่งอยู่ในรูป ASCII
- เราสามารถใช้ **telnet** เชื่อมต่อเข้าที่ port หมายเลข 25 เพื่อส่ง mail ได้
 - **telnet <ชื่อ mail server> 25**

Protocol การส่ง mail ของ SMTP

- เริ่มต้นด้วยการใช้คำสั่ง HELO หรือ EHLO และแต่ version ของ SMTP server ตามด้วยชื่อ host ที่จะรับ mail ตอบกลับ
 - HELO www.microsoft.com
- ใส่ชื่อผู้ส่งด้วยคำสั่ง MAIL FROM:<**email ผู้ส่ง**>
 - MAIL FROM:<billgates@microsoft.com>
- ใส่ชื่อผู้รับด้วยคำสั่ง RCPT TO:<**email ผู้รับ**>
 - RCPT TO:<stevejobs@apple.com>

การส่ง mail ทาง SMTP

- พิมพ์ DATA และ enter เพื่อบอกว่าต่อไปจะเป็นเนื้อหาของ email
- เมื่อเขียน email เรียบร้อยแล้วให้ enter 1 ที่เพื่อให้บรรทัดว่างแล้วพิมพ์จุด (.) ที่คอลั่มแรกของถ้าใหม่แล้วกด enter
- จากนั้นพิมพ์ QUIT เพื่อปิดการเชื่อมต่อกับ mail server

ตัวอย่างการใช้ SMTP

```
Command Prompt
220 mail.kmutnb.ac.th ESMTP NextMessage
HELO www.whitehouse.gov
250 mail.kmutnb.ac.th
MAIL FROM:<bush@whitehouse.gov>
250 OK
RCPT TO:<choopanr@kmutnb.ac.th>
250 OK
DATA
354 End data with <CR><LF>.<CR><LF>
Subject: Free VISA to USA

    Hi there, contact me for free VISA ?

250 Ok: queued as 3D3E93E5FCC3
QUIT
221 Bye

Connection to host lost.

C:\>
```

Fakemail, mailbomb

- สมัยก่อน การติดต่อเข้าใช้บริการ **SMTP** สามารถทำ **mail** ปลอม หรือที่เรียกว่า **fakemail** โดยการปลอมชื่อ **email** ผู้ส่งแบลกฯ ซึ่งเปลี่ยนตรงส่วน **MAIL FROM:<ชื่อ email แบลก ๆ>**
- **Mailbomb** คือการเขียนโปรแกรมวนลูปให้เชื่อมต่อใช้บริการ **SMTP** เพื่อส่ง **fakemail** เป็นจำนวนมาก
- แต่ในสมัยนี้ **SMTP server** จะมีการป้องกันความปลอดภัยเอาไว้คือจะกรองให้ผู้ส่งต้องทำการ **authentication** ก่อนจะส่ง **email** ข้ามออก **server**
- และในปัจจุบันสามารถตรวจสอบและค้นหาผู้ที่ส่ง **fakemail** ได้ดังนั้น **โปรดระวังในการใช้งาน**

HyperText Transfer Protocol (HTTP)

- เป็นบริการเกี่ยวกับ web page โดยเป็น protocol ที่ใช้ในการติดต่อระหว่าง HTTP client (web browser IE, firefox) และ HTTP server หรือที่เรียกว่า web server
- ตามมาตรฐานแล้ว Web server ให้บริการบน TCP ที่ port หมายเลข 80
- ตอนนี้มีใช้กัน 2 version คือ HTTP 1.0 และ HTTP 1.1
- สามารถใช้ telnet ติดต่อไปยัง port หมายเลข 80 ของ web server เพื่อ debug กิจกรรมของ HTTP ได้
 - ในการที่ใช้ Windows ผู้ใช้จะ **ไม่สามารถเห็นข้อความที่พิมพ์**
 - ในการที่ใช้ Linux ผู้ใช้สามารถเห็นข้อความที่พิมพ์

HTTP version 1.0

- คำสั่งที่ใช้ในการ download ทรัพยากร (resources) จาก web server คือ

GET <ชื่อทรัพยากร> **HTTP/1.0**

- **GET** เป็นคำสั่งที่ใช้สำหรับของโหลดทรัพยากรจาก web server
- ชื่อทรัพยากร จะรวมถึง path ที่เข้าถึงทรัพยากร
- **HTTP/1.0** เป็นการระบุบอก web server จะติดต่อแบบ HTTP version 1.0

Example : GET

- ตัวอย่าง

- ถ้าต้องการโหลดหน้าเวป

http://www.somehost.com/*path/file.html*

1. ใช้คำสั่ง **telnet www.somehost.com 80**
2. พิมพ์

GET /*path/file.html* HTTP/1.0

3. กด **Enter** 2 ที

HTTP version 1.0

- ส่วนขยายของการ **request** มีคำสั่งอื่นๆ เช่น
 - **From :** ปั๊บออก **email** ของผู้ติดต่อ กับ **web server** ใช้เฉพาะในการณ์พิเศษ
 - **User-Agent :** ใช้ในการบ่งบอกว่า **web browser** คืออะไร
- รหัสที่ตอบสนองจาก **web server** หลังจากที่ **client** ติดต่อขอ **download** ทรัพยากร
 - **2XX** สำเร็จ
 - **3XX** **redirect** ไปที่หน้าอื่น
 - **4XX** มีปัญหาที่ตัวของ **client**
 - **5XX** มีปัญหาที่ตัวของ **server**

HTTP version 1.0

- นอกจากห้ามการตอบกลับแล้ว **web server** ยังมีข้อความบางอย่างกลับมาด้วยเพื่อเป็นข้อมูล เช่น
 - **Server :** บ่งบอกว่า **web server** ใช้โปรแกรมชื่อว่าอะไร
 - **Last-Modified :** บอกว่าทรัพยากรที่ถูก **request** นี้แก้ไขครั้งล่าสุด เมื่อใด
 - **Content-type :** บอกว่าทรัพยากรที่ถูก **request** นี้เป็นไฟล์ประเภทไหน (**MIME-type**) เช่น **text/html**, **image/gif**
 - **Content-length :** บอกว่าทรัพยากรที่ถูก **request** มีขนาดกี่ไบต์

Example : HTTP/1.0

```
choopan@ect-webserver:~$ telnet 202.44.36.14 80
Trying 202.44.36.14...
Connected to 202.44.36.14.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 04 Dec 2008 20:40:44 GMT
Server: Apache/2.2.8 (Ubuntu) PHP/5.2.4-2ubuntu5.3 with Suhosin-Patch
Last-Modified: Tue, 17 Jun 2008 11:08:31 GMT
ETag: "3a078-70-44fdac3a9e9c0"
Accept-Ranges: bytes
Content-Length: 112
Connection: close
Content-Type: text/html

<html>
<head>
<title>Coming soon</title>
</head>
<body>
<center><h1>Coming soon..</h1></center>
</body>
</html>
Connection closed by foreign host.
choopan@ect-webserver:~$
```

HTTP version 1.0

- นอกจาก **GET** แล้วใน **HTTP version 1.0** ยังมีคำสั่งอื่นๆ อีก เช่น
 - **HEAD** ใช้สำหรับดึงหัวข้อมูลอย่างเดียว โดยไม่ **download** ทรัพยากรที่ **request** มาจริงๆ
 - **POST** ใช้สำหรับส่งค่าให้กับทรัพยากรนั้น เช่น ส่งค่าให้กับ **CGI** เช่น **PHP**
- หลังจาก **HTTP/1.0** ได้รับความนิยมจึงมีการพัฒนา **HTTP version 1.1** ขึ้นมาเพื่อรองรับการทำงานที่มากขึ้น และให้การตอบสนองรวดเร็วขึ้น

HTTP version 1.1

- ใน HTTP/1.1 มีการรองรับ **multi-homed** หมายถึง การที่ **web server** เครื่องเดียวสามารถให้บริการหลายเวปไซต์ เช่น
 - ให้บริการ **www.host1.com** และ **www.host2.com** ในเครื่องเดียวคือมี IP เดียว
- ดังนั้นใน HTTP/1.1 จึงบังคับให้ใส่ชื่อ **host** ทุกครั้งหลังจากใช้คำสั่ง **GET** ด้วยการใช้คำสั่ง **Host :**

Example : HTTP/1.1

□ ตัวอย่าง

- ถ้าต้องการโหลดหน้าเวป

http://www.somehost.com/path/file.html

1. ใช้คำสั่ง **telnet www.somehost.com 80**

2. พิมพ์

GET /path/file.html HTTP/1.1

Host : www.somehost.com

User-Agent: choopan

3. กด Enter 2 ที

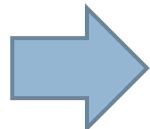
HTTP version 1.1

- ใน HTTP version 1.0 การเชื่อมต่อจะปิดลงทุกครั้ง หลังจากการ transfer ทรัพยากร เสร์วิสสิ้นลง
- การขอเปิดการเชื่อมต่อ และ ปิดการเชื่อมต่อทุกครั้งเพื่อ download ทรัพยากรเพียง 1 อย่าง จะเพิ่มโหลดให้กับ web server อย่างมาก และ ทำให้การตอบสนองช้า
- HTTP version 1.1 จึงได้พัฒนาคือ การเชื่อมต่อจะไม่ปิดตัวลงหลังจาก download ทรัพยากรเสร็จสิ้น เพื่อความเร็วในการตอบสนอง และ ลด ภาระโหลดให้กับ web server

HTTP version 1.1

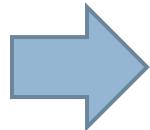
- อย่างไรก็ตาม HTTP version 1.1 ก็ยังให้อ่านจำกับ client ในการเลือกรูปแบบการเชื่อมต่อ ว่าจะให้ปิดการเชื่อมต่อหลังจาก download ทรัพยากรหรือไม่
- ตัวอย่าง : จะโหลด <http://www.somehost.com>

Default : ไม่ปิด
การเชื่อมต่อหลังจาก
โหลดหน้าเวปเสร็จ



GET / HTTP/1.1
Host : www.somehost.com

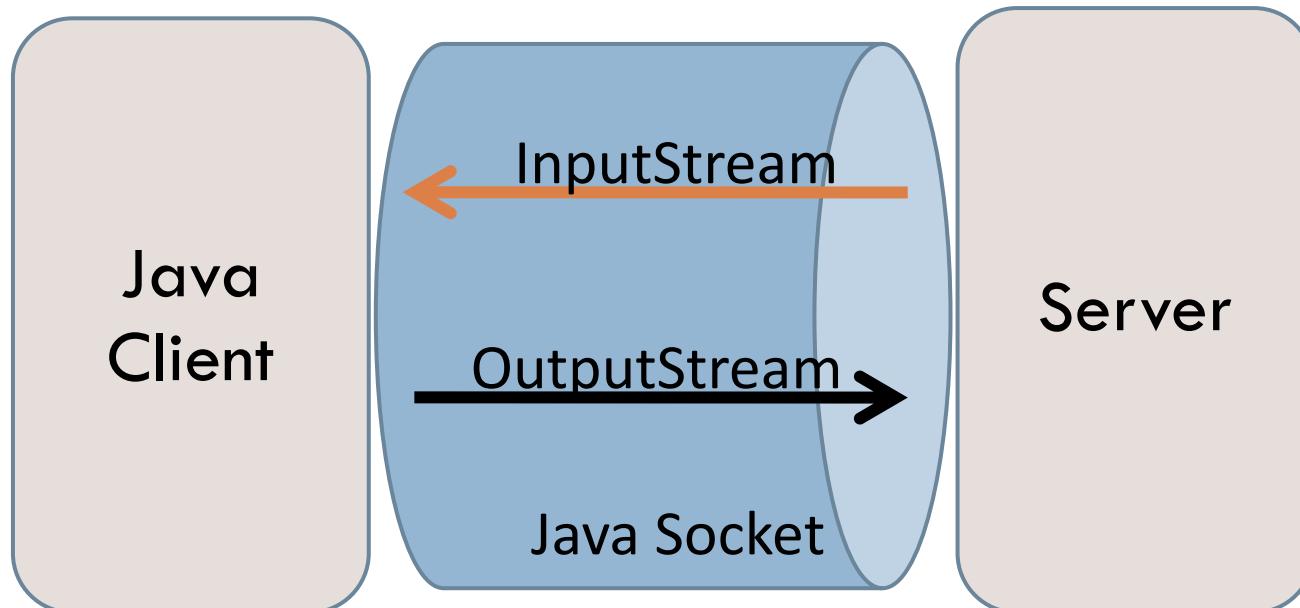
ปิดการเชื่อมต่อหลังจาก
โหลดหน้าเวปเสร็จ



GET / HTTP/1.1
Host : www.somehost.com
Connection : close

Java Socket

- ทบทวน
 - การสร้าง **socket** ใน **java** คือการใช้ **Class Socket**
 - **Socket s = new Socket("ชื่อโฮสต์", หมายเลขพอร์ต);**



การอ่านค่าและส่งค่าผ่าน socket

- จาก Object ของ Socket ที่เราได้สามารถเรียกใช้ method ดังนี้

- การอ่านค่าจาก Socket จะทำผ่าน InputStream

*public InputStream **getInputStream()** throws IOException*

- การส่งค่าลงไปใน Socket จะทำผ่าน OutputStream

*public OutputStream **getOutputStream()** throws IOException*

- เมื่อสิ้นสุดการทำงานทุกรอบ จะต้องเรียก method **close();** เพื่อปิดการใช้งานของ InputStream และ OutputStream

ตัวอย่างการเอา InputStream และ OutputStream ออกจาก Socket

```
1 import java.io.*;
2 import java.net.*;
3
4 public class TestOne {
5     public static void main(String[] args) {
6         try {
7             Socket s = new Socket("127.0.0.1", 80);
8             InputStream in = s.getInputStream();
9             OutputStream out = s.getOutputStream();
10            in.close();
11            out.close();
12            s.close();
13        } catch (Exception e) {
14            e.printStackTrace();
15        }
16    }
17 }
```

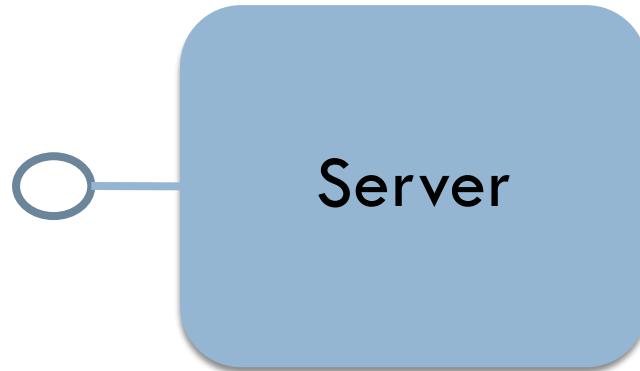
ประยุกต์ใช้ BufferedReader และ PrintWriter

```
1 import java.io.*;
2 import java.net.*;
3
4 public class TestTwo {
5     public static void main(String[] args) {
6         try {
7             Socket s = new Socket("127.0.0.1", 80);
8             InputStream in = s.getInputStream();
9             OutputStream out = s.getOutputStream();
10
11
12             BufferedReader sin = new BufferedReader(
13                     new InputStreamReader(in));
14
15             PrintWriter sout = new PrintWriter(out);
16
17             sin.close();
18             sout.close();
19             in.close();
20             out.close();
21             s.close();
22         } catch (Exception e) {
23             e.printStackTrace();
24         }
25     }
26 }
```

JAVA SERVERSOCKET

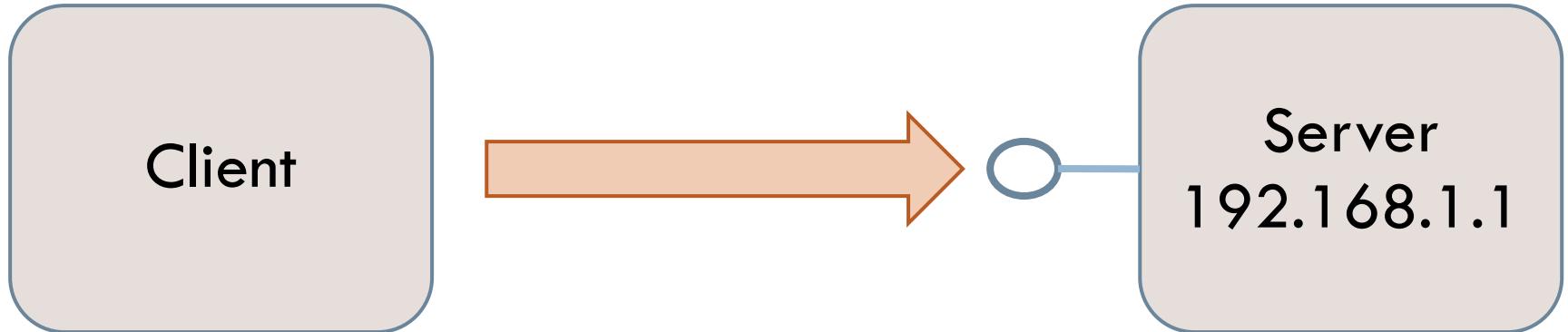
030523313 - Network programming
Asst. Prof. Dr. Choopan Rattanapoka

Java ServerSocket



- Server มีหน้าที่รอการติดต่อจาก Client ใน port ที่ตัวเองจะให้บริการ
 - ใน Java จะสามารถทำได้โดยการสร้าง Object ของ ServerSocket
 - ตัวอย่าง :
- ServerSocket server = new ServerSocket(หมายเลข port);**

Client and Server



```
ServerSocket server = new ServerSocket(2000);
```

```
Socket s = new Socket("192.168.1.1", 2000);
```

ฝ่าย **Server** จะเป็นจะต้องมี **Socket** เพื่อติดต่อกับ **Client** สามารถทำได้โดย

Socket client = server.accept();

โปรแกรมจะค้างที่บรรทัดนี้จนกว่าจะมี **Client** มาร้องขอการเชื่อมต่อ

ตัวอย่างโปรแกรมของฝั่ง server

```
1 import java.io.*;
2 import java.net.*;
3
4 public class Server {
5     public static void main(String[] args) {
6         ServerSocket server = null;
7         try {
8             server = new ServerSocket(2000);
9         } catch (Exception e) {
10            e.printStackTrace();
11        }
12
13        while(true) {
14            try {
15                Socket client = server.accept();
16
17                //CODE HERE
18
19                client.close();
20
21            } catch(Exception e) {
22                e.printStackTrace();
23            }
24        }
25    }
26 }
```

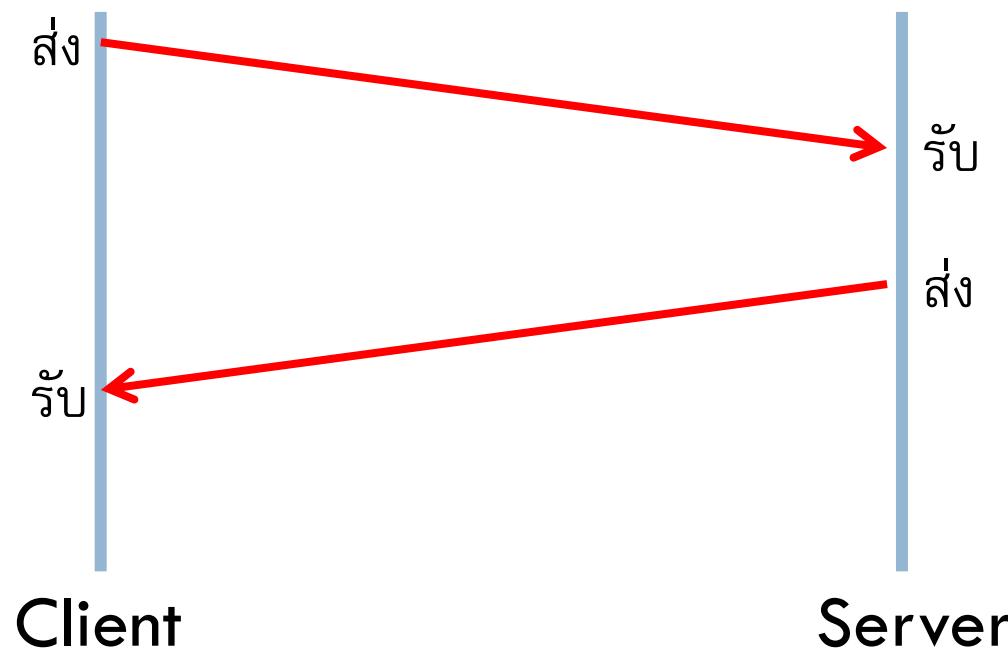
วน loop รอรับ
การติดต่อจาก client

เอา socket ออกมาใช้งาน

ปิดเมื่อใช้งานเรียบร้อย

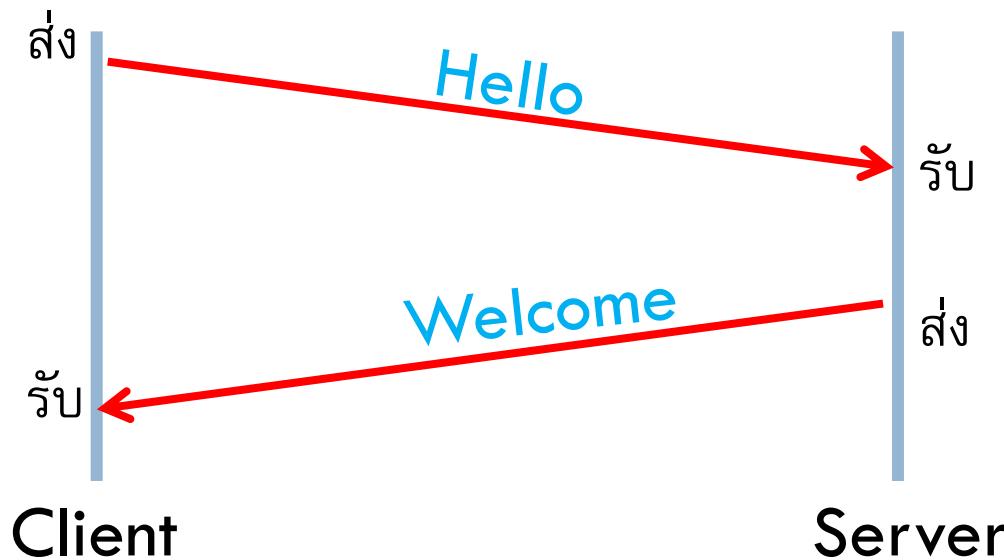
การเขียน server/client

- การเขียน **server** และ **client** ด้วยตัวเองนั้นจะต้องออกแบบ **protocol**
- คือการออกแบบช่วงไหนจะรับข้อมูล ช่วงไหนจะส่งข้อมูล โดย **server** และ **client** จะต้องทำหน้าที่สลับกัน



ตัวอย่าง 1: โปรแกรม server/client

- ถ้าต้องการเขียนโปรแกรมให้
 - Server ให้บริการที่ port หมายเลข 10000
 - Client ติดต่อกับ Server พร้อมทั้งส่งคำว่า Hello
 - เมื่อ Server ได้รับคำว่า Hello, server จะตอบคำว่า Welcome
 - จากนั้นปิดการเชื่อมต่อ



โปรแกรม Client

```
import java.net.*;
import java.io.*;
public class Client {
    public static void main(String[] args) {
        try {
            Socket s = new Socket("127.0.0.1", 10000);
            BufferedReader br = new BufferedReader(
                new InputStreamReader(
                    s.getInputStream()));
            PrintWriter pw = new PrintWriter(s.getOutputStream());
            pw.println("Hello");
            pw.flush();
            String msg = br.readLine();
            System.out.println(msg);
            br.close();
            pw.close();
            s.close();
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

ส่ง Hello
รับ

โปรแกรม Server

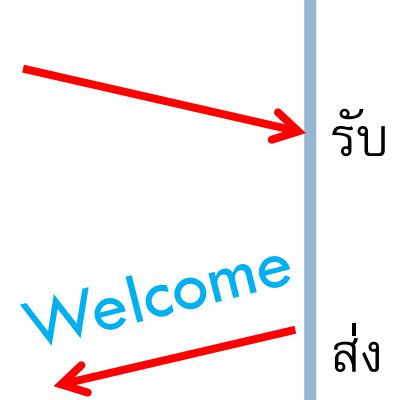
```
import java.io.*;
import java.net.*;

public class Server {
    public static void main(String[] args) {
        ServerSocket servSocket = null;
        try {
            servSocket = new ServerSocket(10000);
        } catch(Exception e) {e.printStackTrace();}

        while(true) {
            try {
                Socket s = servSocket.accept();

                BufferedReader br = new BufferedReader(
                    new InputStreamReader(
                        s.getInputStream()));
                PrintWriter pw = new PrintWriter(
                    s.getOutputStream());
                String msg = br.readLine();

                pw.println("Welcome");
                pw.flush();
                pw.close();
                br.close();
                s.close();
            } catch(Exception e) {}
        }
    }
}
```



ตัวอย่าง 2 : โปรแกรม login, password

- เขียนโปรแกรมให้ **server** เป็นตัวตรวจสอบ **login** และ **password** อย่างง่าย โดยมีการทำงานดังต่อไปนี้
 - **Server** ให้บริการที่ **port 20000**
 - **Client** จะเชื่อมต่อไปยัง **Server** จากนั้นจะส่ง
 - **Login** (มาจาก **parameter** ตัวที่ 1)
 - **Password** (มาจาก **parameter** ตัวที่ 2)
 - **Server** ตรวจสอบว่า **login = admin** และ **password = admin** หรือไม่
 - ถ้าใช้ให้ส่งข้อความ **OK** กลับไปยัง **client**
 - ถ้าไม่ใช้ให้ส่งข้อความ **NO** กลับไปยัง **client**

โปรแกรม Client

```
import java.net.*;
import java.io.*;
public class Client2 {
    public static void main(String[] args) {
        try {
            Socket s = new Socket("127.0.0.1", 20000);
            BufferedReader br = new BufferedReader(
                new InputStreamReader(
                    s.getInputStream()));
            PrintWriter pw = new PrintWriter(s.getOutputStream());

            pw.println(args[0]);
            pw.println(args[1]);
            pw.flush();

            String msg = br.readLine();

            System.out.println(msg);

            br.close();
            pw.close();
            s.close();
        } catch (Exception e) { e.printStackTrace(); }

    }
}
```

โปรแกรม Server

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

public class Server2 {
    public static void main(String[] args) {
        ServerSocket servSocket = null;
        try {
            servSocket = new ServerSocket(20000);
        } catch(Exception e) {e.printStackTrace();}
        while(true) {
            try {
                Socket s = servSocket.accept();
                BufferedReader br = new BufferedReader(
                    new InputStreamReader(
                        s.getInputStream()));
                PrintWriter pw = new PrintWriter(
                    s.getOutputStream());
                String login = br.readLine();
                String passwd = br.readLine();
                if(login.equals("admin") && passwd.equals("admin"))
                    pw.println("OK");
                else
                    pw.println("NO");
                pw.flush();
                pw.close();
                br.close();
                s.close();
            } catch(Exception e) {}
        }
    }
}
```

ตัวอย่าง 3 : โปรแกรมที่ **server** ใช้เวลาทำงานนาน

- เขียนโปรแกรมให้ **server** โดยกำหนดให้งานที่ **server** ให้บริการต่อ **client** ใช้เวลานานมาก
 - **Server** ให้บริการที่ **port** หมายเลข 30000
 - **Client** ติดต่อไปยังส่งเลข 1 ค่า (วินาที) [มาจาก **args[0]**]
 - **Server** จะหลับรอเป็นจำนวนวินาทีที่ **Client** ส่งเข้ามา จากนั้นส่งคำว่า **OK** คืนให้กับ **Client**

โปรแกรม Client

```
import java.net.*;
import java.io.*;
public class Client3 {
    public static void main(String[] args) {
        try {
            Socket s = new Socket("127.0.0.1", 30000);
            BufferedReader br = new BufferedReader(
                new InputStreamReader(
                    s.getInputStream()));
            PrintWriter pw = new PrintWriter(s.getOutputStream());

            pw.println(args[0]);
            pw.flush();

            String msg = br.readLine();
            System.out.println(msg);

            br.close();
            pw.close();
            s.close();
        } catch (Exception e) { e.printStackTrace(); }

    }
}
```

โปรแกรม Server

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

public class Server3 {
    public static void main(String[] args) {
        ServerSocket servSocket = null;
        try {
            servSocket = new ServerSocket(30000);
        } catch(Exception e) {e.printStackTrace();}
        while(true) {
            try {
                Socket s = servSocket.accept();
                BufferedReader br = new BufferedReader(
                    new InputStreamReader(
                        s.getInputStream()));
                PrintWriter pw = new PrintWriter(
                    s.getOutputStream());
                String sleep = br.readLine();
                long sleepTime = Long.parseLong(sleep);
                try {
                    Thread.sleep(sleepTime*1000);
                } catch(Exception se) {}
                pw.println("OK");
                pw.flush();
                pw.close();
                br.close();
                s.close();
            } catch(Exception e) {}
        }
    }
}
```

ผลการรัน

- ถ้าสังเกตุผลการรันของโปรแกรมตัวอย่างที่ 3
 - จะเห็นว่า **Server** สามารถให้บริการ **Client** ได้ทีละ 1 ตัวเท่านั้น
 - **Server** จะให้บริการ **Client** ตัวถัดไปต่อเมื่อ **Server** ได้ให้บริการกับ **Client** เสร็จสิ้นก่อน
 - ชื่นชมการขึ้นจากเมื่ออด **accept** ที่จะถูกเรียกซ้ำต่อเมื่อ **server** ทำงานต่างๆ ใน **loop while** เรียบร้อยก่อน
 - ปัญหาจึงทำให้ **server** ให้บริการบางอย่างที่ต้องใช้เวลาในการทำงานนานมาก **server** จะไม่สามารถให้บริการกับ **client** อื่นได้เลยในช่วงเวลานั้น
 - วิธีแก้ไข เขียนให้ **server** รองรับการทำงานแบบ **multi-thread**

การเขียน Server แบบ multi-thread

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

public class Server extends Thread {
    Socket s;

    public Server(Socket s) {
        this.s = s;
    }

    public void run() {
        // CODE HERE
    }
}

public static void main(String[] args) {
    ServerSocket server = null;
    try {
        server = new ServerSocket(2000);
    } catch (Exception e) {
        System.exit(1);
    }

    while(true) {
        try {
            Socket client = server.accept();
            Thread t = new Server(client);
            t.start();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

แก้ไขตัวอย่างที่ 3 ให้เป็นแบบ multi-thread

1

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

public class Server4 extends Thread {
    Socket s = null;

    public Server4(Socket s) {
        this.s = s;
    }

    public void run() {
        try {
            BufferedReader br = new BufferedReader(
                new InputStreamReader(
                    s.getInputStream()));
            PrintWriter pw = new PrintWriter(
                s.getOutputStream());

            String sleep = br.readLine();
            long sleepTime = Long.parseLong(sleep);
            try {
                Thread.sleep(sleepTime*1000);
            } catch(Exception se) {}

            pw.println("OK");
            pw.flush();
            pw.close();
            br.close();
            s.close();
        } catch(Exception e) { e.printStackTrace(); }
    }
}
```

```
public static void main(String[] args) {
    ServerSocket servSocket = null;
    try {
        servSocket = new ServerSocket(30000);
    } catch(Exception e) {e.printStackTrace();}

    while(true) {
        try {
            Socket s = servSocket.accept();
            Server4 serverThread = new Server4(s);
            serverThread.start();
        } catch(Exception e) {}
    }
}
```

2

Code ของ client ไม่จำเป็นต้องมี
การเปลี่ยนแปลงใดๆ ทั้งสิ้น

ปรับเปลี่ยนให้เป็น ThreadPool

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

public class ServerTP3 implements Runnable {
    Socket s = null;
    public ServerTP3(Socket s) {
        this.s = s;
    }

    public void run() {
        try {
            BufferedReader br = new BufferedReader(
                new InputStreamReader(
                    s.getInputStream()));
            PrintWriter pw = new PrintWriter(s.getOutputStream());

            String sleep = br.readLine();
            long sleepTime = Long.parseLong(sleep);

            try {
                Thread.sleep(sleepTime * 1000);
            } catch(Exception se) {}

            pw.println("OK");
            pw.flush();
            pw.close(); br.close(); s.close();
        } catch(Exception e) {}
    }
}
```

```
public static void main(String[] args) {
    ServerSocket serv = null;
    ExecutorService es = Executors.newFixedThreadPool(10);

    try {
        serv = new ServerSocket(30000);
    } catch(Exception e) {System.exit(1);}

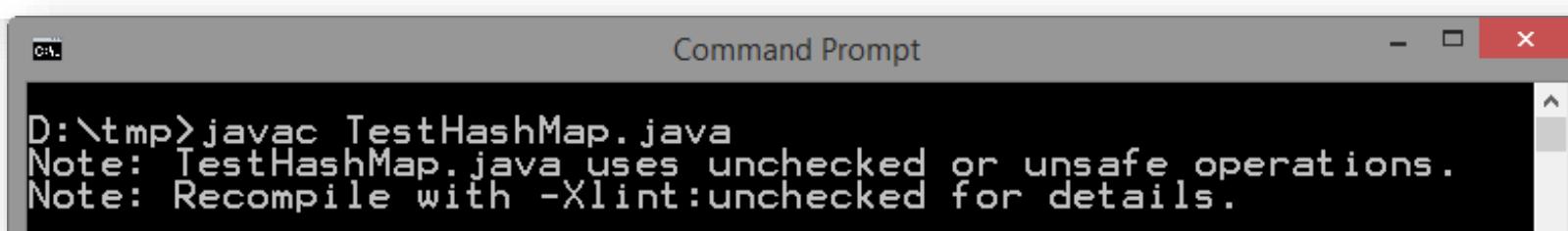
    while(true) {
        try {
            Socket s = serv.accept();
            ServerTP3 st = new ServerTP3(s);
            es.execute(st);
        } catch(Exception e) {}
    }
}
```

Java HashMap

- **HashMap** อยู่ใน package `java.util.*;`
- **HashMap** มีการทำงานในลักษณะของ `<key, value>`
 - คือมี `key` เหมือนกับเป็น ID
 - โดยแต่ละ `key` จะมีค่า (`value`) ของ `key` นั้นๆ
 - `hashMap` จะมีการค้นหาข้อมูลที่รวดเร็ว $O(1)$
- เมื่อต้องการใช้งาน **Class HashMap** มีอยู่ 2 เมธอดคือ
 - **put(Object key, Object value)**
 - เป็นการเก็บข้อมูล `<key, value>` ลงในตาราง `hash`
 - **Object get(Object key)**
 - เป็นการดึงค่า `value` ออกจากตาราง `hash` โดยใช้ `key` เป็นตัวค้นหา
 - คืนค่า `null` ถ้าไม่มี `key` ใน `HashMap`

ตัวอย่าง : การใช้งาน HashMap

```
import java.util.*;  
  
public class TestHashMap {  
    public static void main(String[] args) {  
        HashMap map = new HashMap();  
        map.put("Bob", "0539218230");  
        map.put("Korn", "0885912102");  
        map.put("Pang", "0891200873");  
  
        System.out.println("Pang :" + map.get("Pang"));  
        System.out.println("Bob :" + map.get("Bob"));  
        System.out.println("Korn :" + map.get("Korn"));  
    }  
}
```



D:\tmp>javac TestHashMap.java
Note: TestHashMap.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

ตรวจสอบปัญหาของการใช้งาน HashMap

```
Command Prompt
D:\tmp>javac TestHashMap.java
Note: TestHashMap.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

D:\tmp>javac -Xlint TestHashMap.java
TestHashMap.java:5: warning: [rawtypes] found raw type: HashMap
    ^      HashMap map = new HashMap();
           missing type arguments for generic class HashMap<K,V>
           where K,V are type-variables:
             K extends Object declared in class HashMap
             V extends Object declared in class HashMap
TestHashMap.java:5: warning: [rawtypes] found raw type: HashMap
    ^      HashMap map = new HashMap();
           missing type arguments for generic class HashMap<K,V>
           where K,V are type-variables:
             K extends Object declared in class HashMap
             V extends Object declared in class HashMap
TestHashMap.java:6: warning: [unchecked] unchecked call to put(K,V) as a member
of the raw type HashMap
    ^      map.put("Bob", "0539218230");
           where K,V are type-variables:
             K extends Object declared in class HashMap
             V extends Object declared in class HashMap
TestHashMap.java:7: warning: [unchecked] unchecked call to put(K,V) as a member
of the raw type HashMap
    ^      map.put("Korn", "0885912102");
           where K,V are type-variables:
             K extends Object declared in class HashMap
             V extends Object declared in class HashMap
TestHashMap.java:8: warning: [unchecked] unchecked call to put(K,V) as a member
of the raw type HashMap
    ^      map.put("Pang", "0891200873");
           where K,V are type-variables:
             K extends Object declared in class HashMap
             V extends Object declared in class HashMap
5 warnings
D:\tmp>
```

แก้ปัญหาต้องระบุประเภทของข้อมูล

```
import java.util.*;  
  
public class TestHashMap {  
    public static void main(String[] args) {  
        HashMap<String, String> map = new HashMap<String, String>();  
        map.put("Bob", "0539218230");  
        map.put("Korn", "0885912102");  
        map.put("Pang", "0891200873");  
  
        System.out.println("Pang :" + map.get("Pang"));  
        System.out.println("Bob  :" + map.get("Bob"));  
        System.out.println("Korn :" + map.get("Korn"));  
    }  
}
```

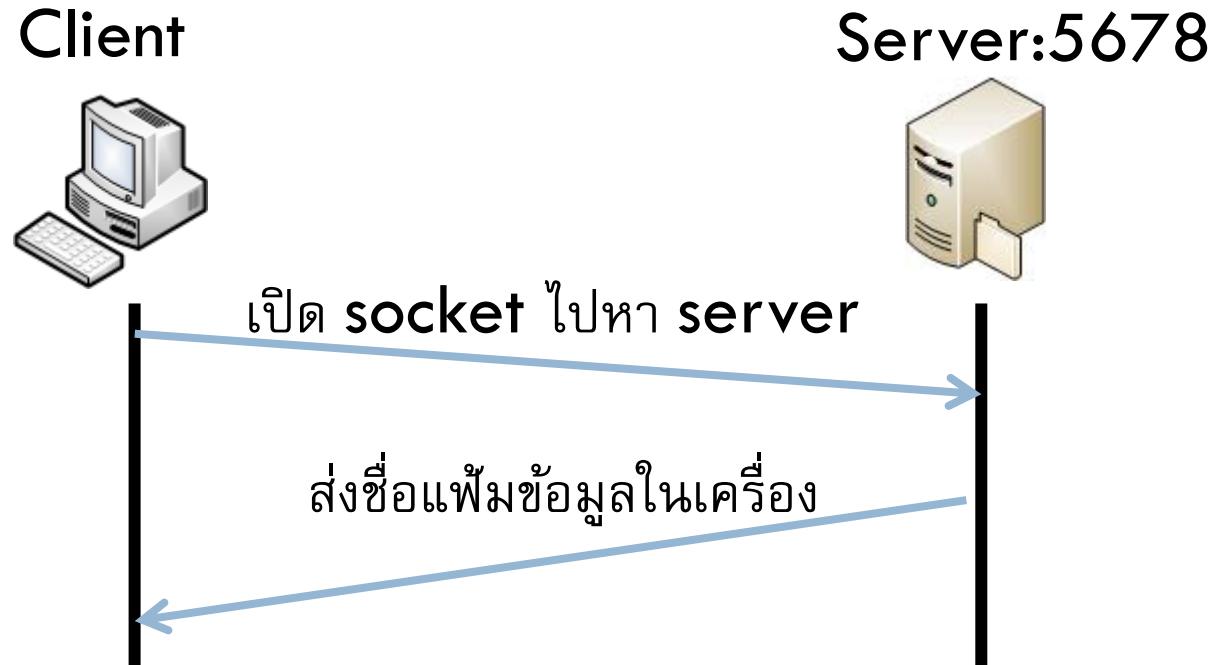
CLIENT/SERVER APPLICATION (FILE SERVER)

030523313 - Network programming
Asst. Prof. Dr. Choopan Rattanapoka

Introduction

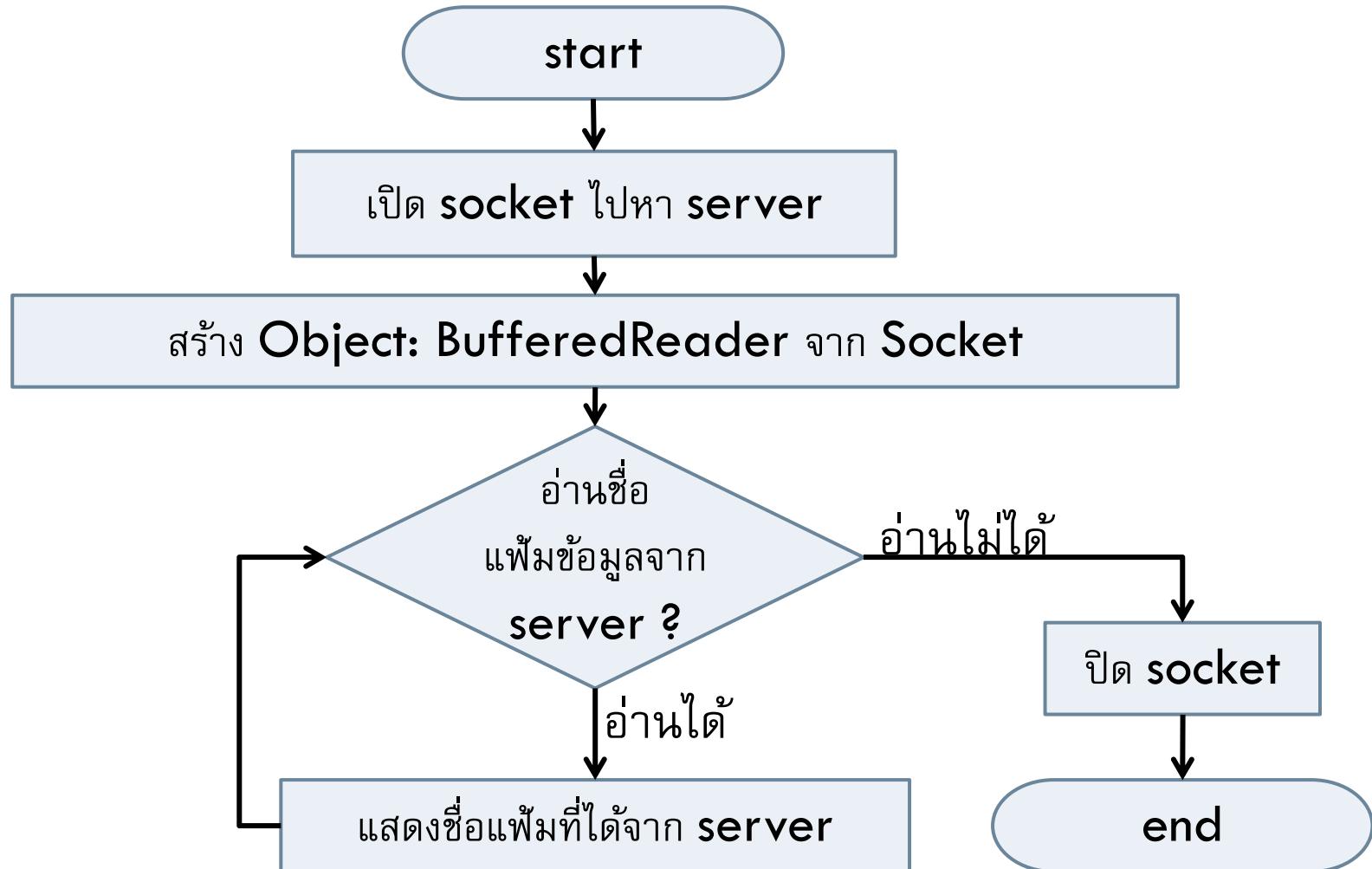
- **Server/Client** ของระบบเพิ่มข้อมูล โดยจะมีตัวอย่างการทำงานแบบง่ายๆ ของการทำงาน 3 อย่างคือ
 - **List** ดูรายชื่อเพิ่มข้อมูลที่เครื่อง **server**
 - **Upload** เพิ่มข้อมูล
 - **Download** เพิ่มข้อมูล

การดูรายชื่อของแฟ้มข้อมูลบน Server (LIST)

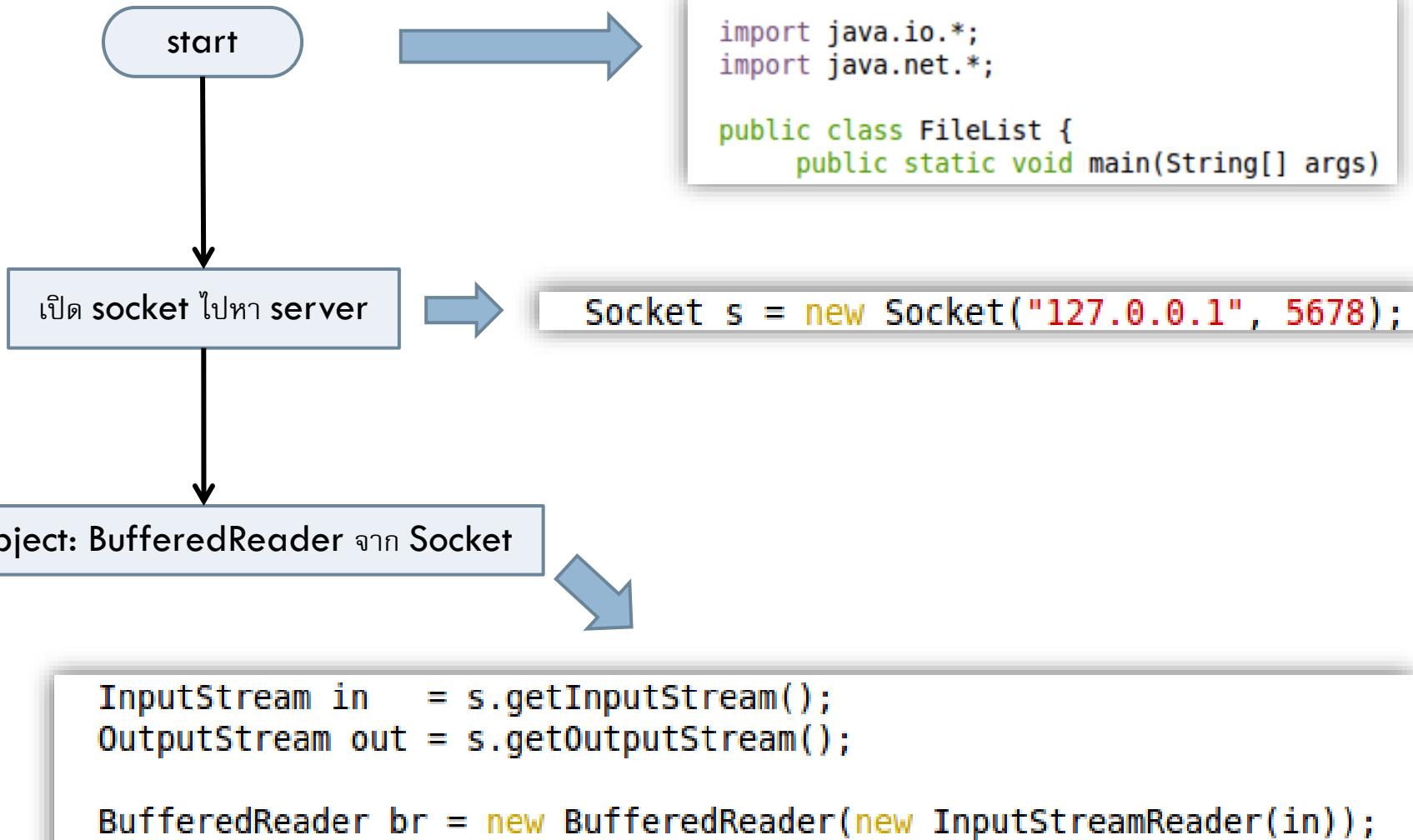


- เตือนความจำ
 - ใน Java สามารถดูรายชื่อใน Directory ได้ด้วยการใช้เมธอด **list()** ของ **Object File**

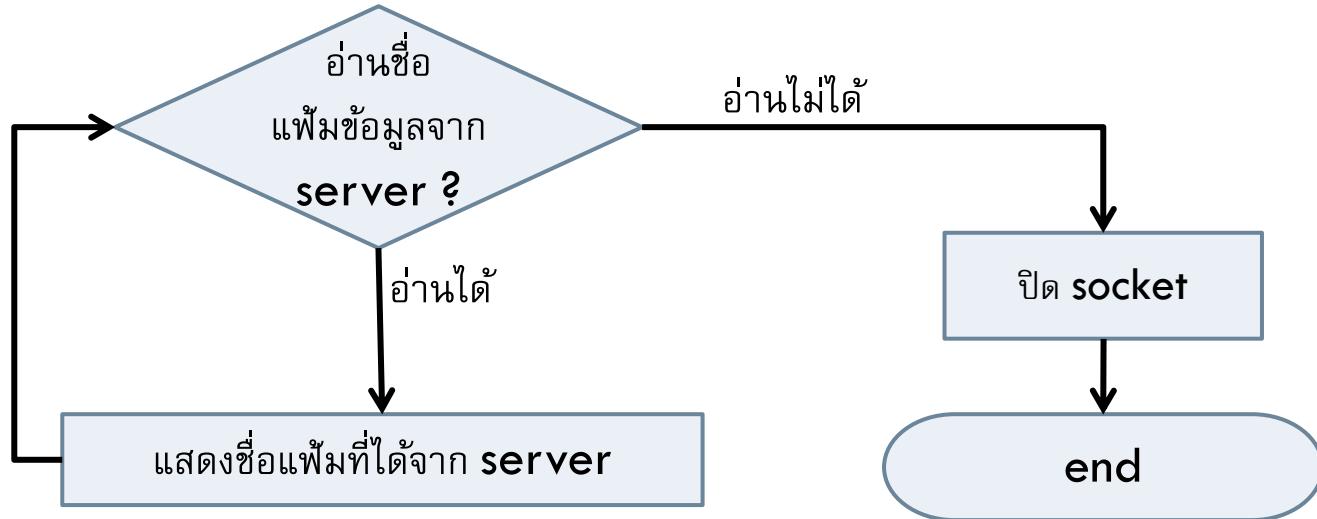
Flow Chart : Client (LIST)



Flow Chart to Code (1)



Flow Chart to Code (2)



```
String filename;  
while((filename = br.readLine()) != null) {  
    System.out.println(filename);  
}  
in.close();  
out.close();  
s.close();
```

Source Code : FileList.java

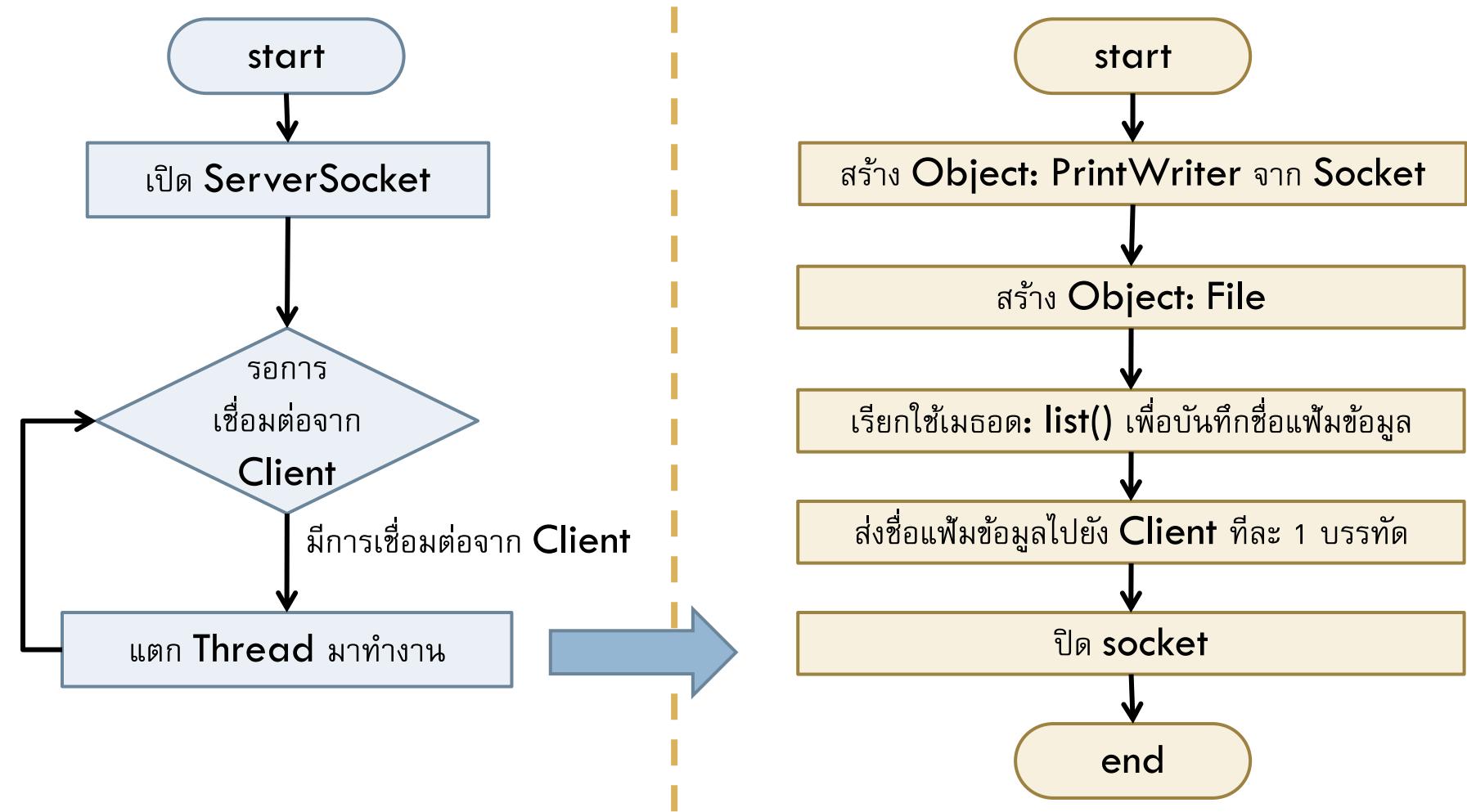
```
import java.io.*;
import java.net.*;

public class FileList {
    public static void main(String[] args) {
        try {
            Socket s = new Socket("127.0.0.1", 5678);
            InputStream in   = s.getInputStream();
            OutputStream out = s.getOutputStream();

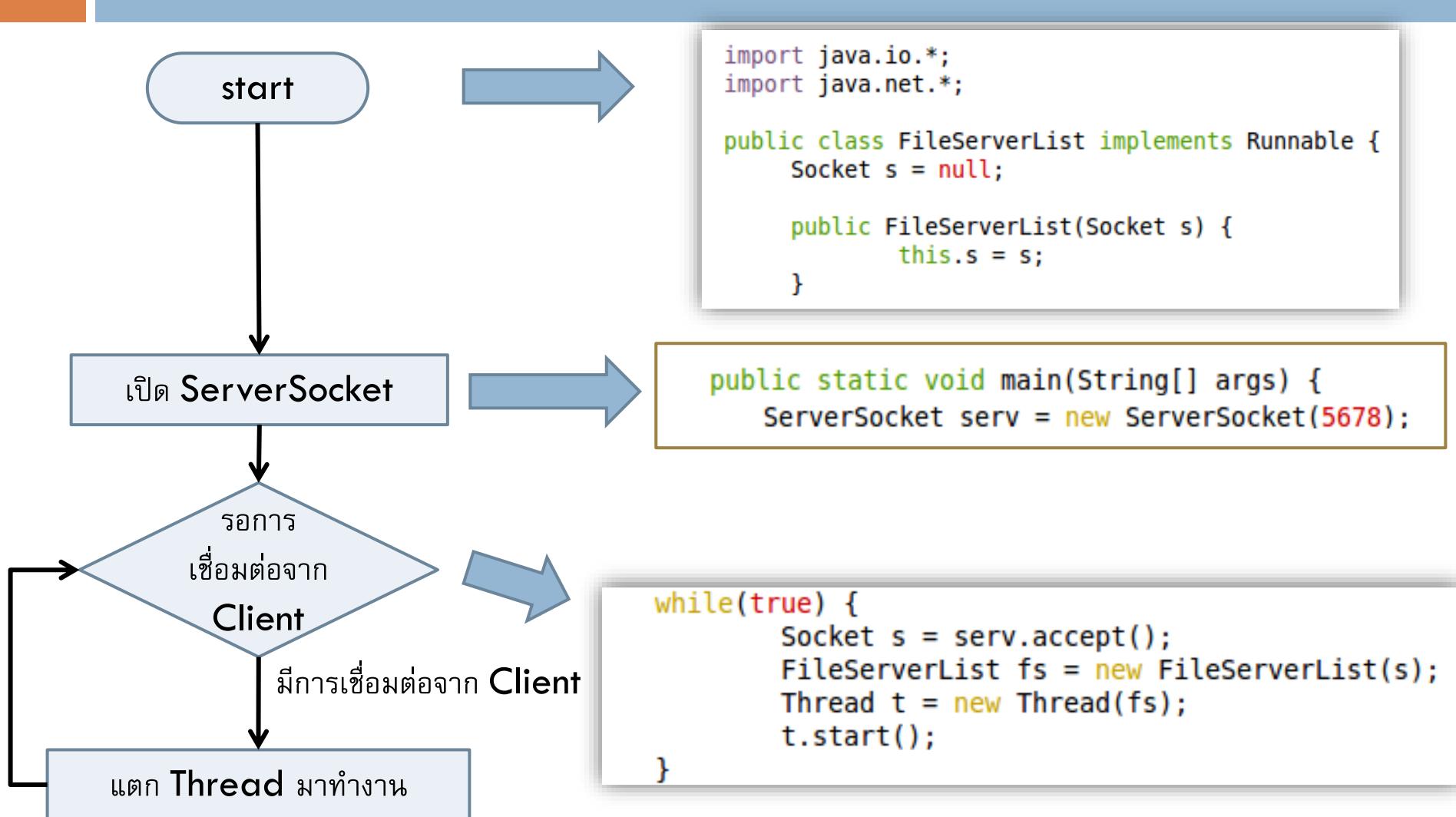
            BufferedReader br = new BufferedReader(new InputStreamReader(in));

            String filename;
            while((filename = br.readLine()) != null) {
                System.out.println(filename);
            }
            in.close();
            out.close();
            s.close();
        } catch(Exception e) { e.printStackTrace(); }
    }
}
```

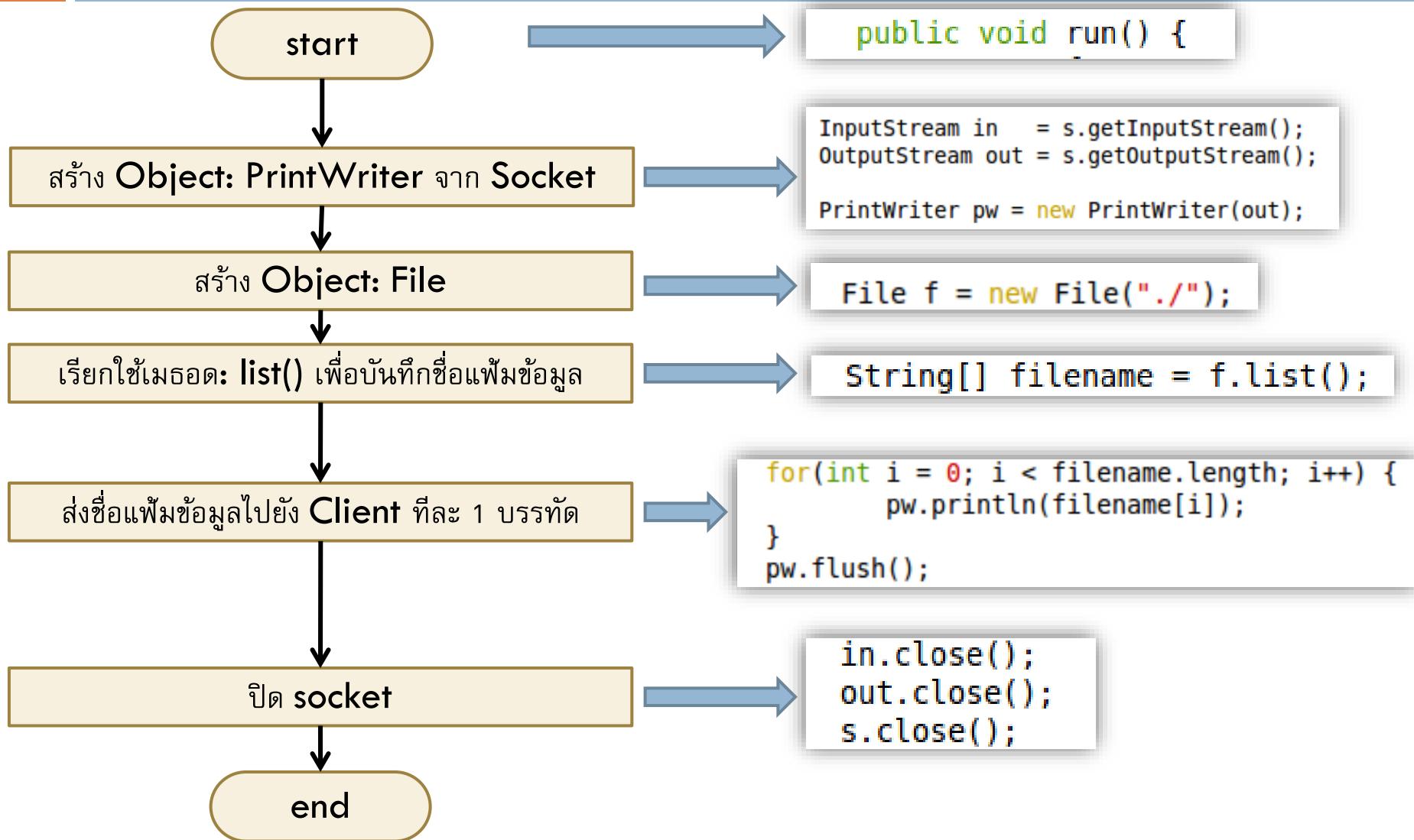
Flow Chart : Server (LIST)



Flow Chart to Code (1)



Flow Chart to Code (2)



Source Code : FileServerList.java

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

public class FileServerList implements Runnable {
    Socket s = null;

    public FileServerList(Socket s) {
        this.s = s;
    }

    public void run() {
        try {
            InputStream in = s.getInputStream();
            OutputStream out = s.getOutputStream();

            PrintWriter pw = new PrintWriter(out);

            File f = new File("./");
            String[] filename = f.list();
            for(int i = 0; i < filename.length; i++) {
                pw.println(filename[i]);
            }
            pw.flush();
            in.close();
            out.close();
            s.close();
        } catch (Exception e) { e.printStackTrace(); }
    }
}
```

```
public static void main(String[] args) {
    try {
        ServerSocket serv = new ServerSocket(5678);
        while(true) {
            Socket s = serv.accept();
            FileServerList fs = new FileServerList(s);
            Thread t = new Thread(fs);
            t.start();
        }
    } catch(Exception e) { e.printStackTrace(); }
}
```

Client-Server Communication

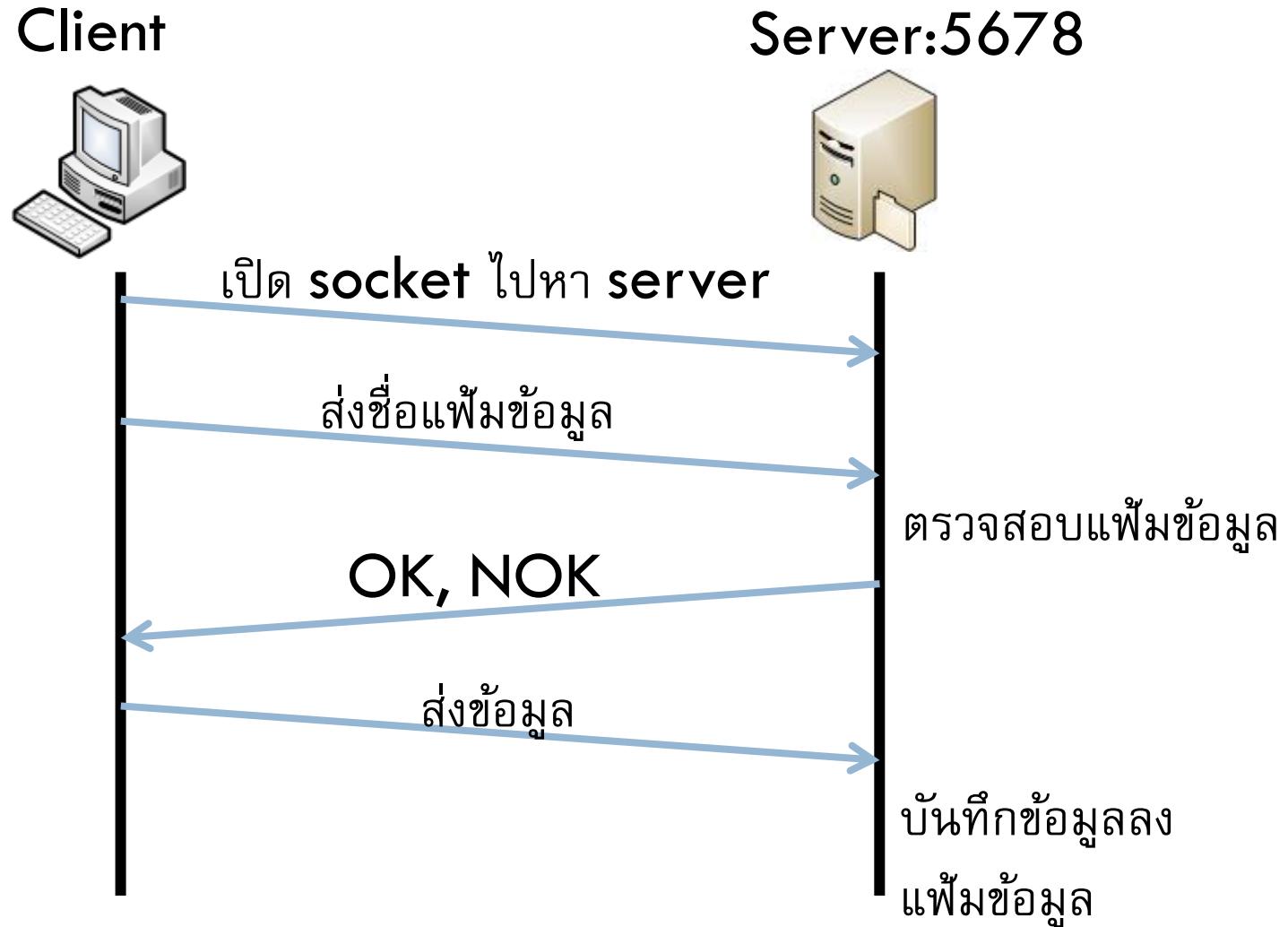
Client

```
try {  
    Socket s = new Socket("127.0.0.1", 5678);  
    InputStream in = s.getInputStream();  
    OutputStream out = s.getOutputStream();  
  
    BufferedReader br = new BufferedReader(new InputStreamReader(in));  
  
    String filename;  
    while((filename = br.readLine()) != null) {  
        System.out.println(filename);  
    }  
    in.close();  
    out.close();  
    s.close();  
} catch(Exception e) { e.printStackTrace(); }
```

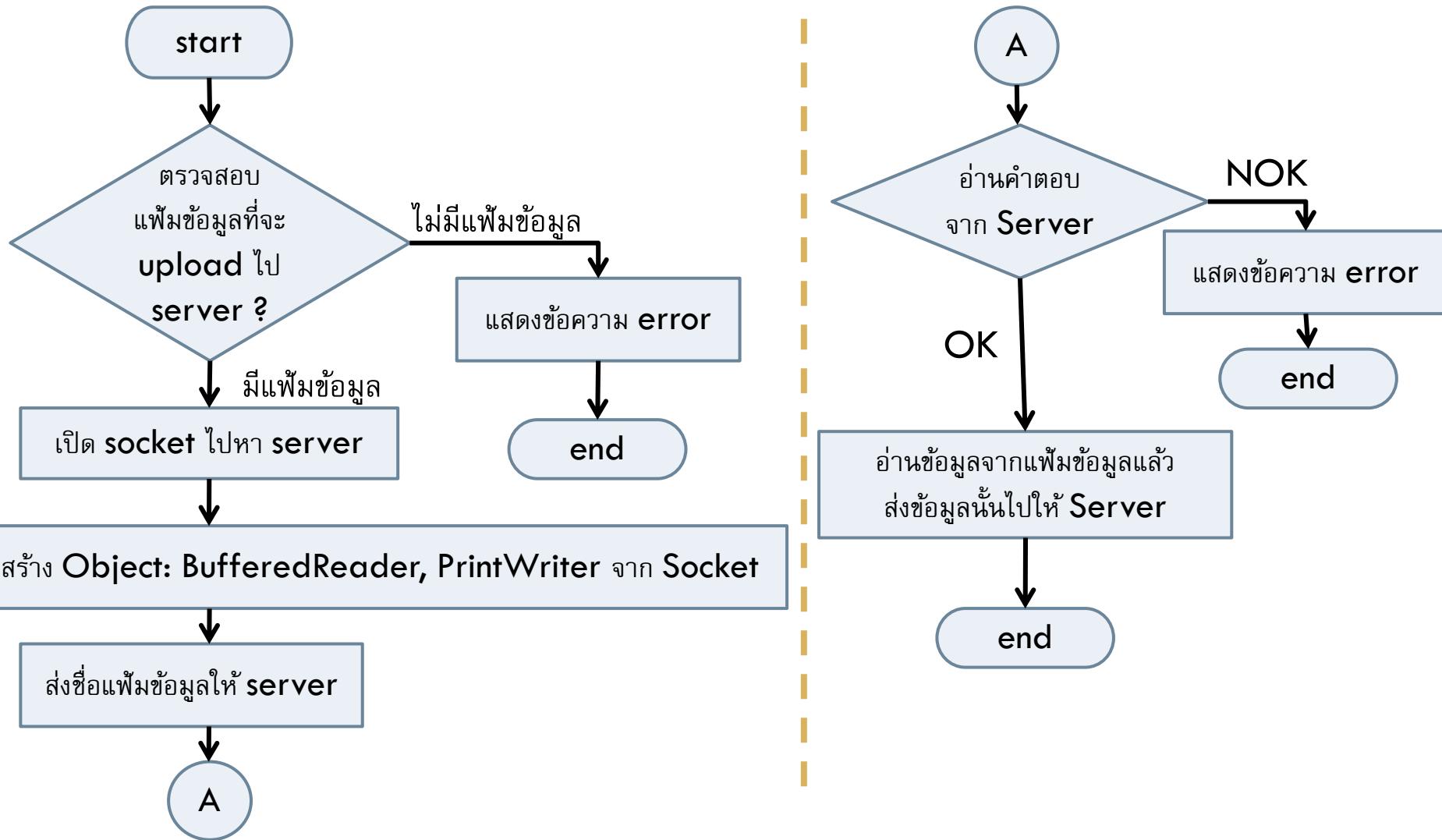
Server

```
public void run() {  
    try {  
        InputStream in = s.getInputStream();  
        OutputStream out = s.getOutputStream();  
  
        PrintWriter pw = new PrintWriter(out);  
  
        File f = new File("./");  
        String[] filename = f.list();  
        for(int i = 0; i < filename.length; i++) {  
            pw.println(filename[i]);  
        }  
        pw.flush();  
        in.close();  
        out.close();  
        s.close();  
    } catch (Exception e) { e.printStackTrace(); }  
}
```

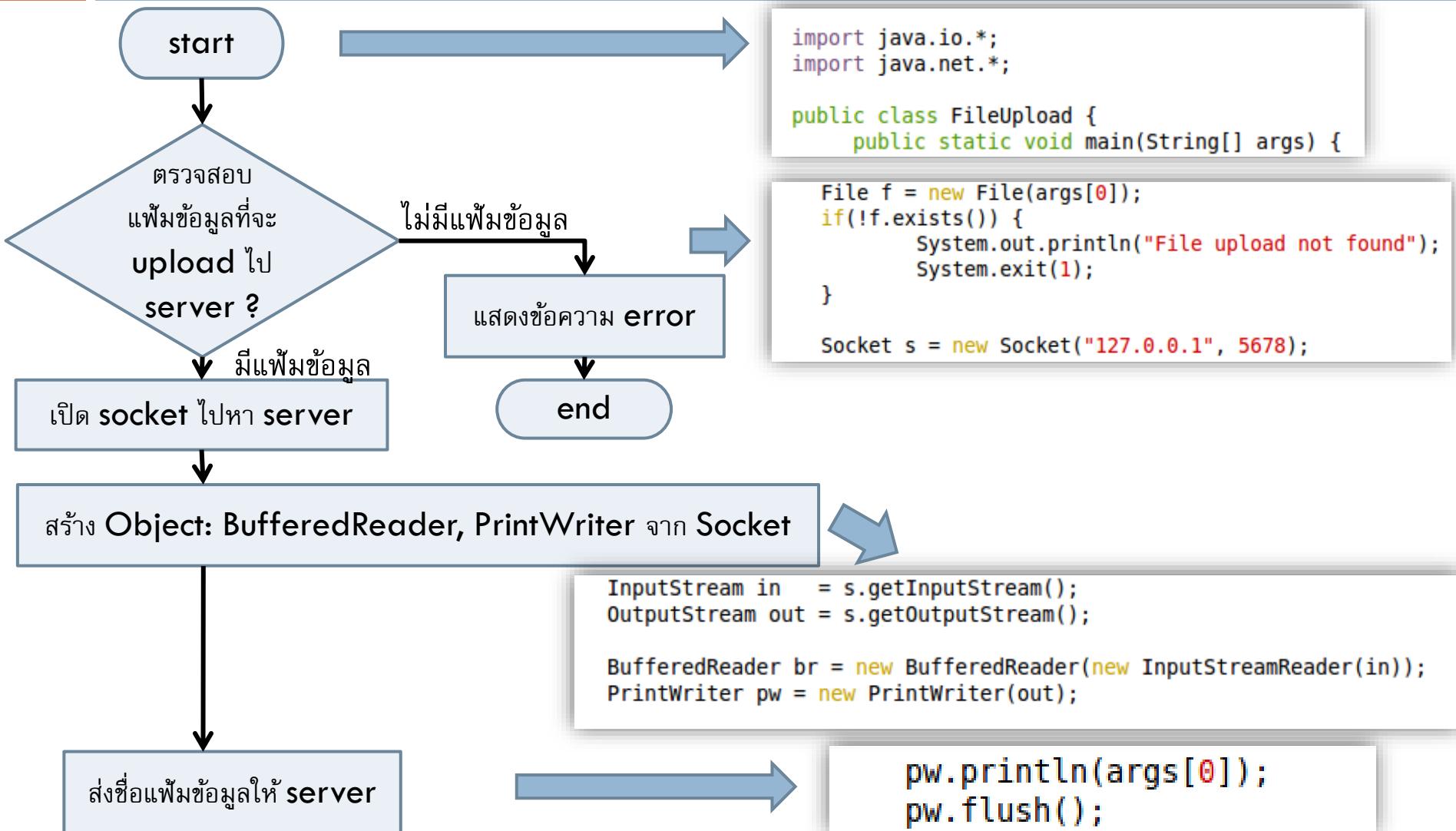
การ Upload แฟ้มข้อมูลเข้าสู่ Server



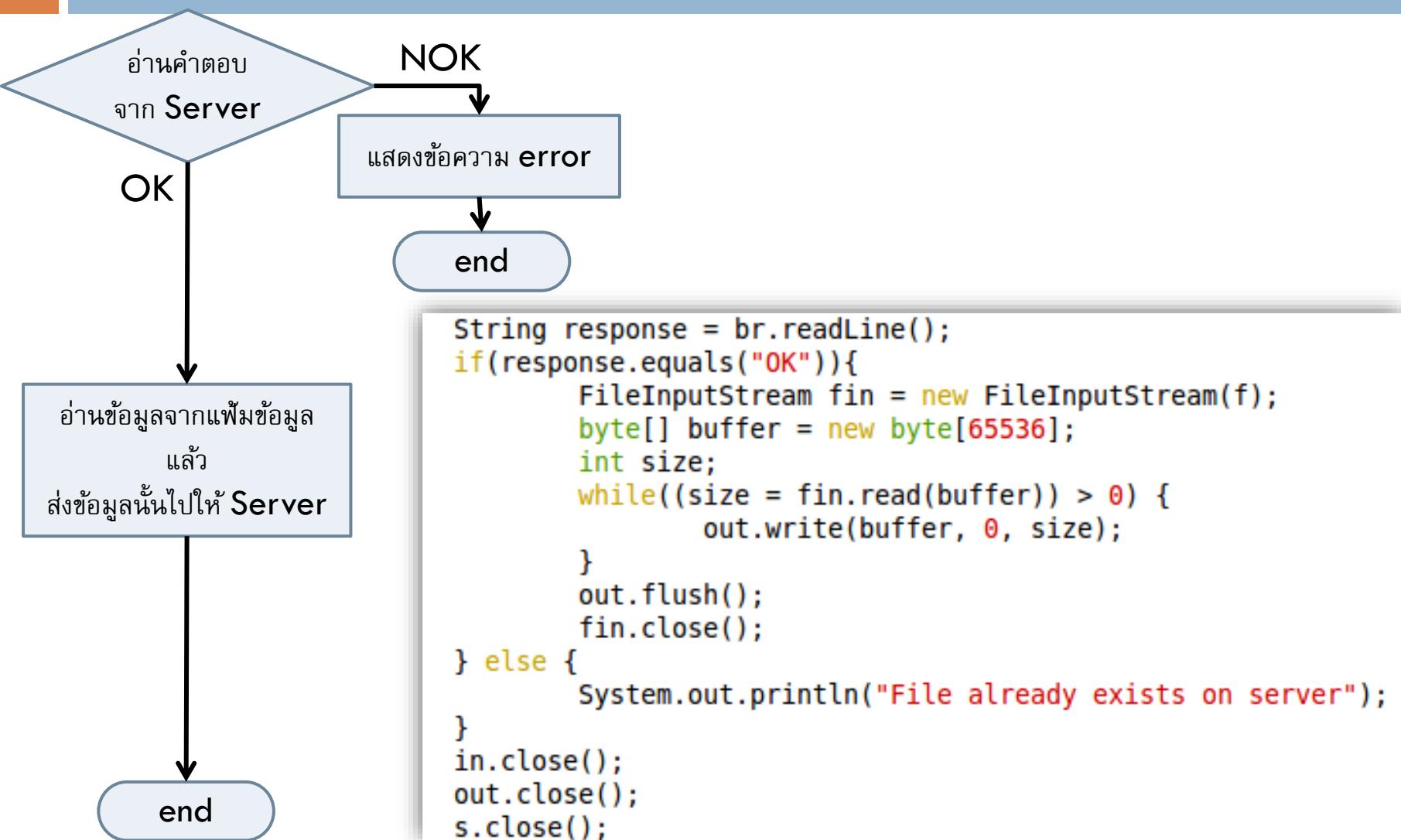
Flow Chart : Client (Upload)



Flow Chart to Code (1)



Flow Chart to Code (2)



Source Code : FileUpload.java

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

public class FileUpload {
    public static void main(String[] args) {
        try {
            File f = new File(args[0]);
            if(!f.exists()) {
                System.out.println("File upload not found");
                System.exit(1);
            }

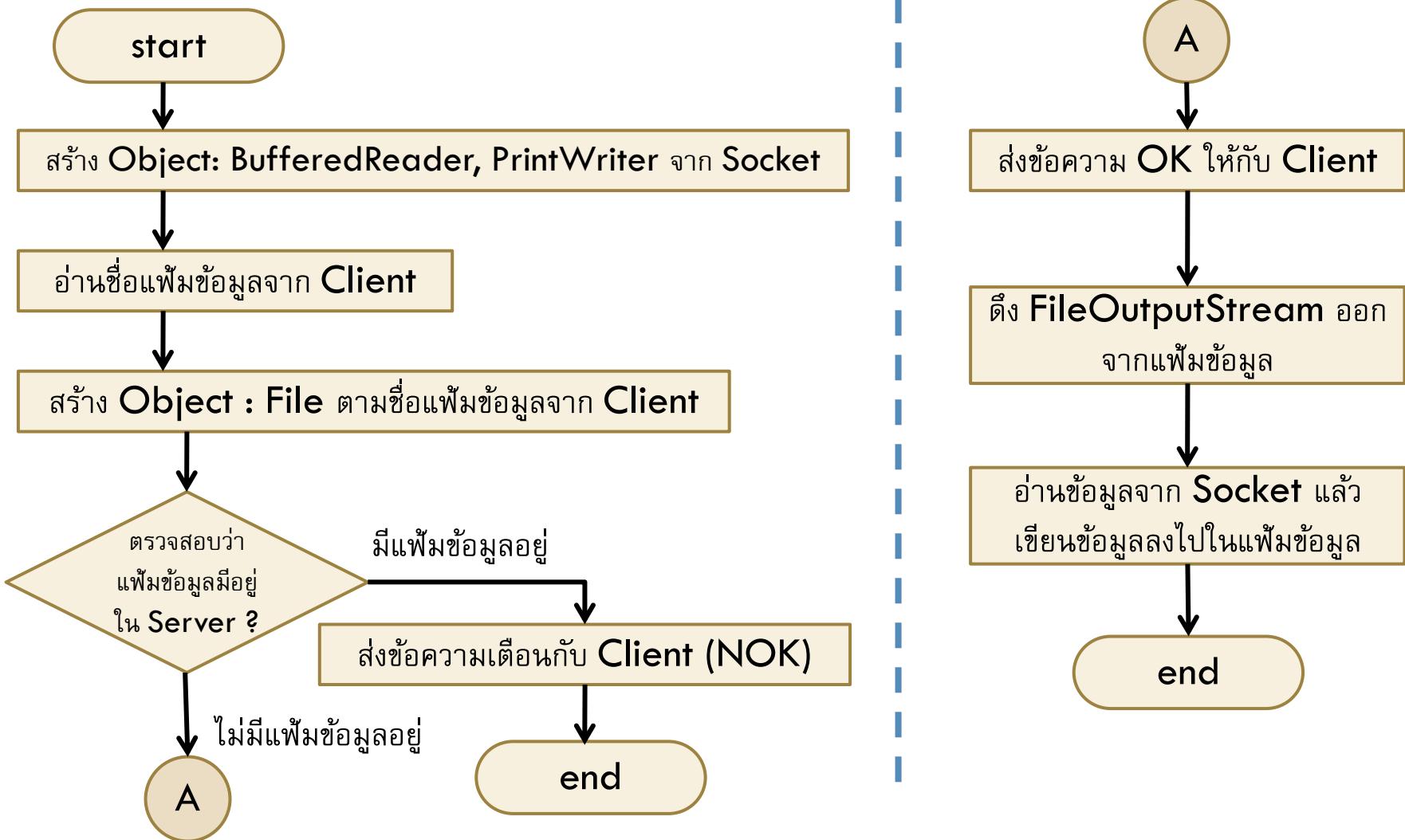
            Socket s = new Socket("127.0.0.1", 5678);
            InputStream in = s.getInputStream();
            OutputStream out = s.getOutputStream();

            BufferedReader br = new BufferedReader(new InputStreamReader(in));
            PrintWriter pw = new PrintWriter(out);

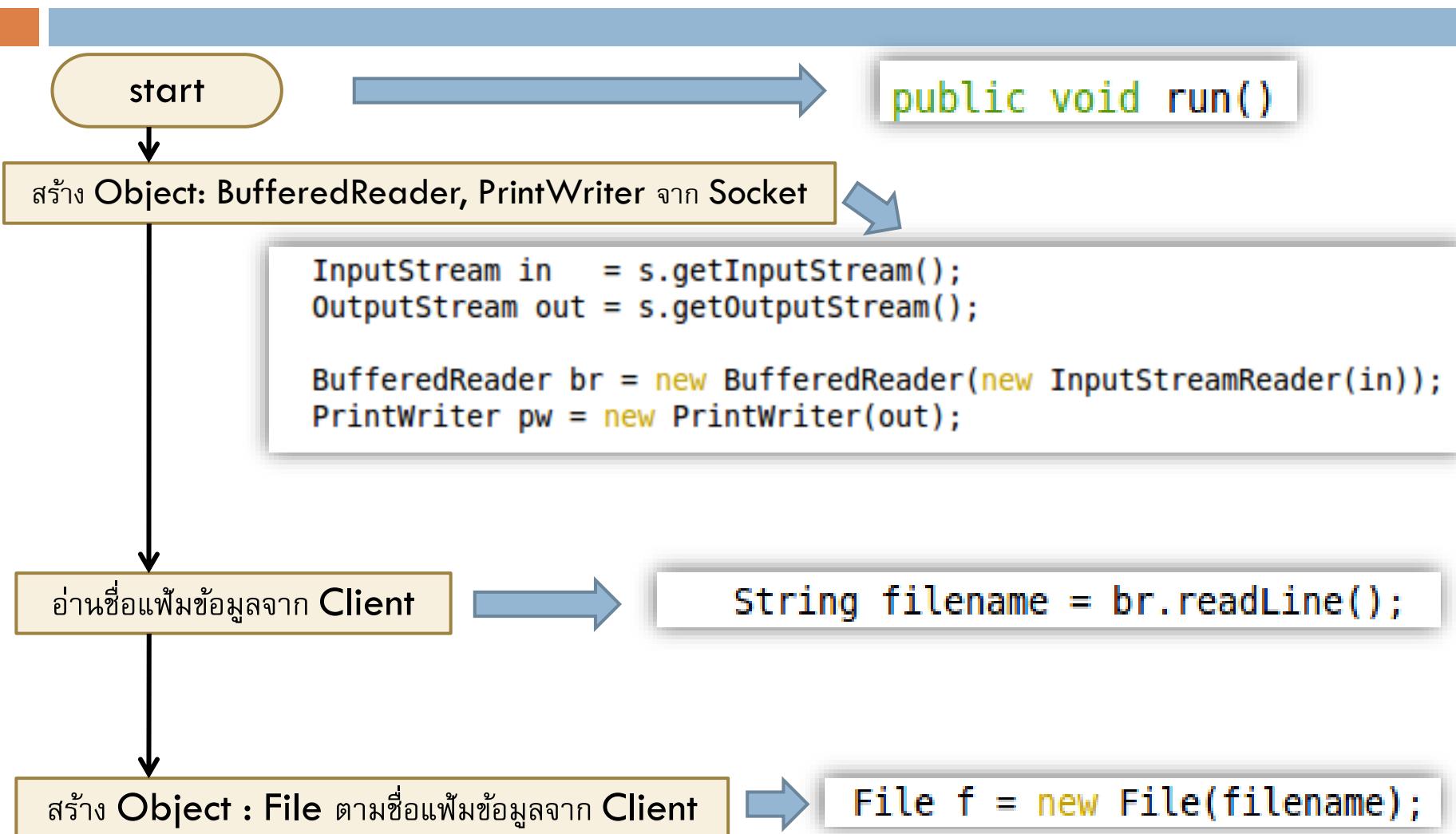
            pw.println(args[0]);
            pw.flush();
            String response = br.readLine();
            if(response.equals("OK")){
                FileInputStream fin = new FileInputStream(f);
                byte[] buffer = new byte[65536];
                int size;
                while((size = fin.read(buffer)) > 0) {
                    out.write(buffer, 0, size);
                }
                out.flush();
                fin.close();
            } else {
                System.out.println("File already exists on server");
            }
            in.close();
            out.close();
            s.close();
        } catch(Exception e) { e.printStackTrace(); }

    }
}
```

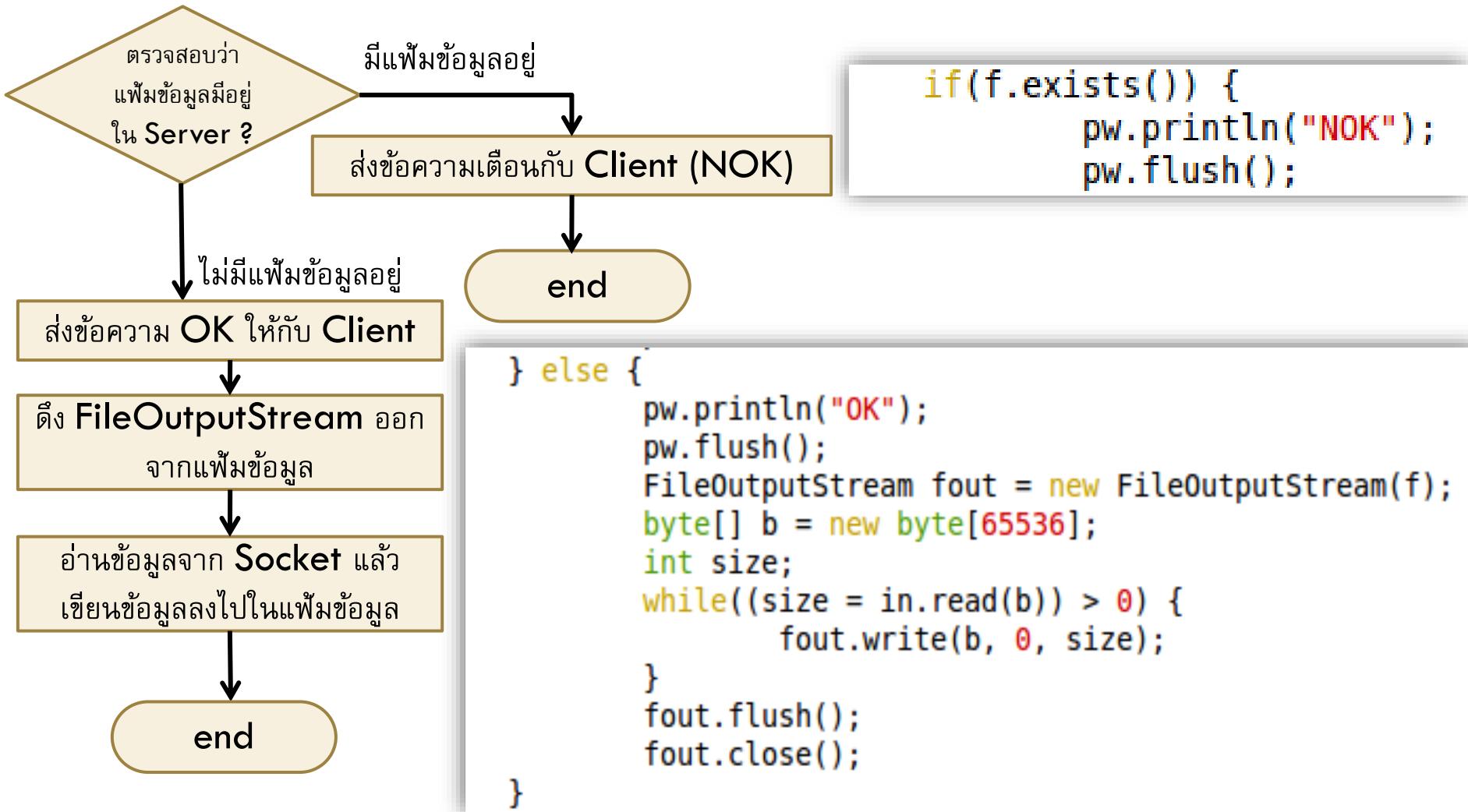
Flow Chart : Server (Upload)



Flow Chart to Code (1)



Flow Chard to Code (2)



Source Code : FileServerUpload.java

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

public class FileServerUpload implements Runnable {
    Socket s = null;

    public FileServerUpload(Socket s) {
        this.s = s;
    }

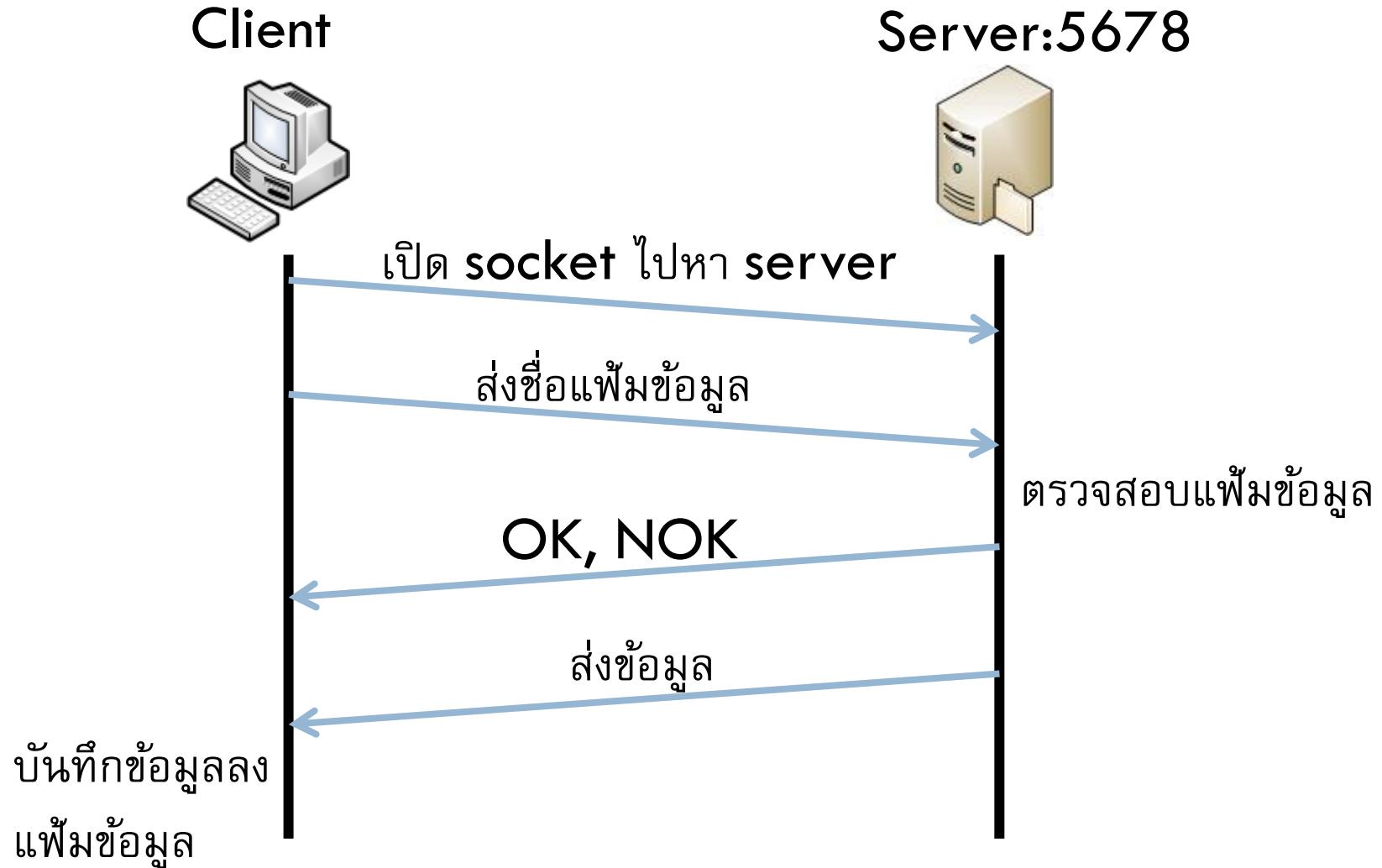
    public void run() {
        try {
            InputStream in = s.getInputStream();
            OutputStream out = s.getOutputStream();

            BufferedReader br = new BufferedReader(new InputStreamReader(in));
            PrintWriter pw = new PrintWriter(out);

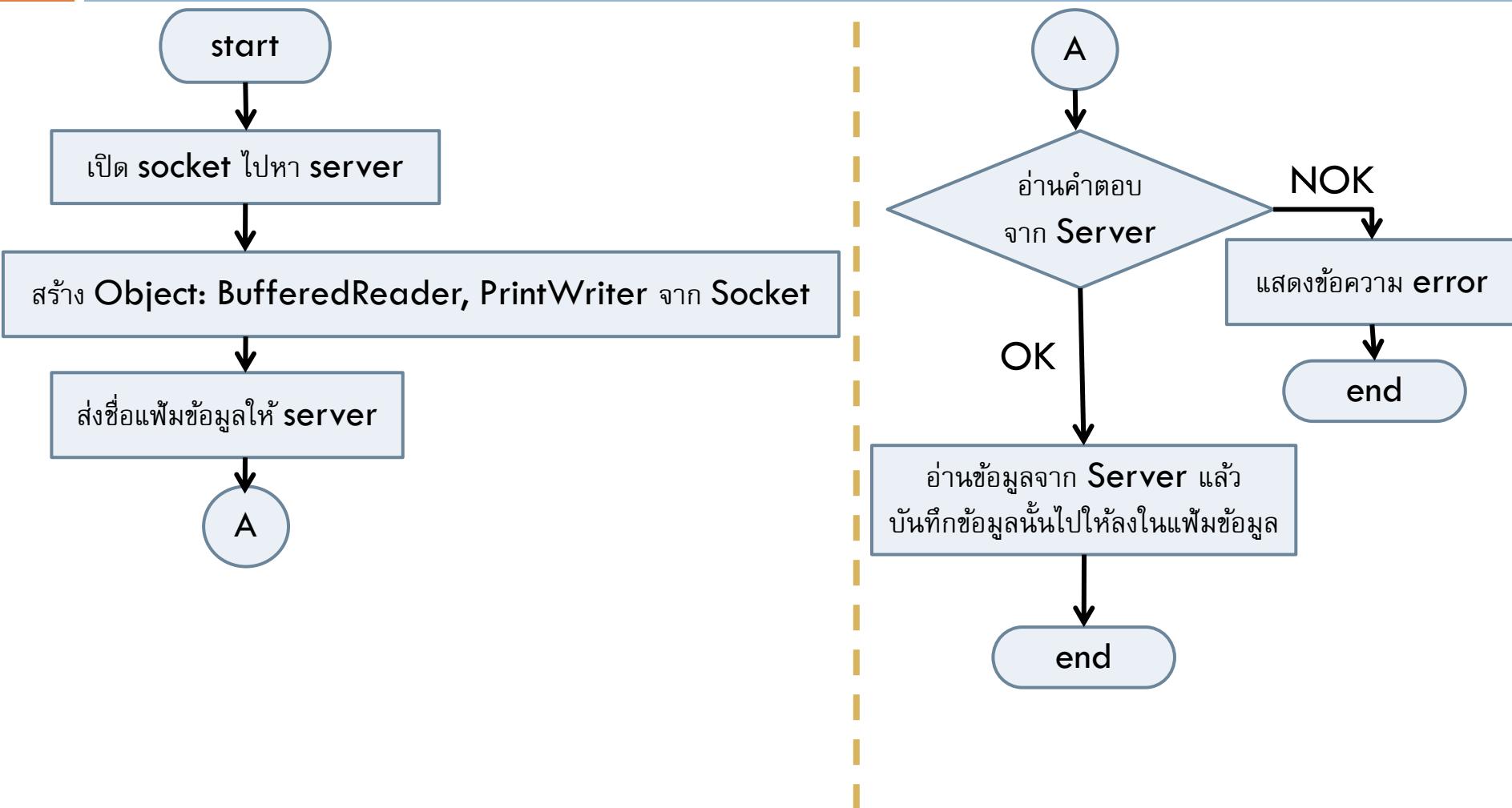
            String filename = br.readLine();
            File f = new File(filename);
            if(f.exists()) {
                pw.println("NOK");
                pw.flush();
            } else {
                pw.println("OK");
                pw.flush();
                FileOutputStream fout = new FileOutputStream(f);
                byte[] b = new byte[65536];
                int size;
                while((size = in.read(b)) > 0) {
                    fout.write(b, 0, size);
                }
                fout.flush();
                fout.close();
            }
            in.close();
            out.close();
            s.close();
        } catch (Exception e) { e.printStackTrace(); }
    }
}

public static void main(String[] args) {
    try {
        ServerSocket serv = new ServerSocket(5678);
        while(true) {
            Socket s = serv.accept();
            FileServerUpload fs = new FileServerUpload(s);
            Thread t = new Thread(fs);
            t.start();
        }
    } catch(Exception e) { e.printStackTrace(); }
}
```

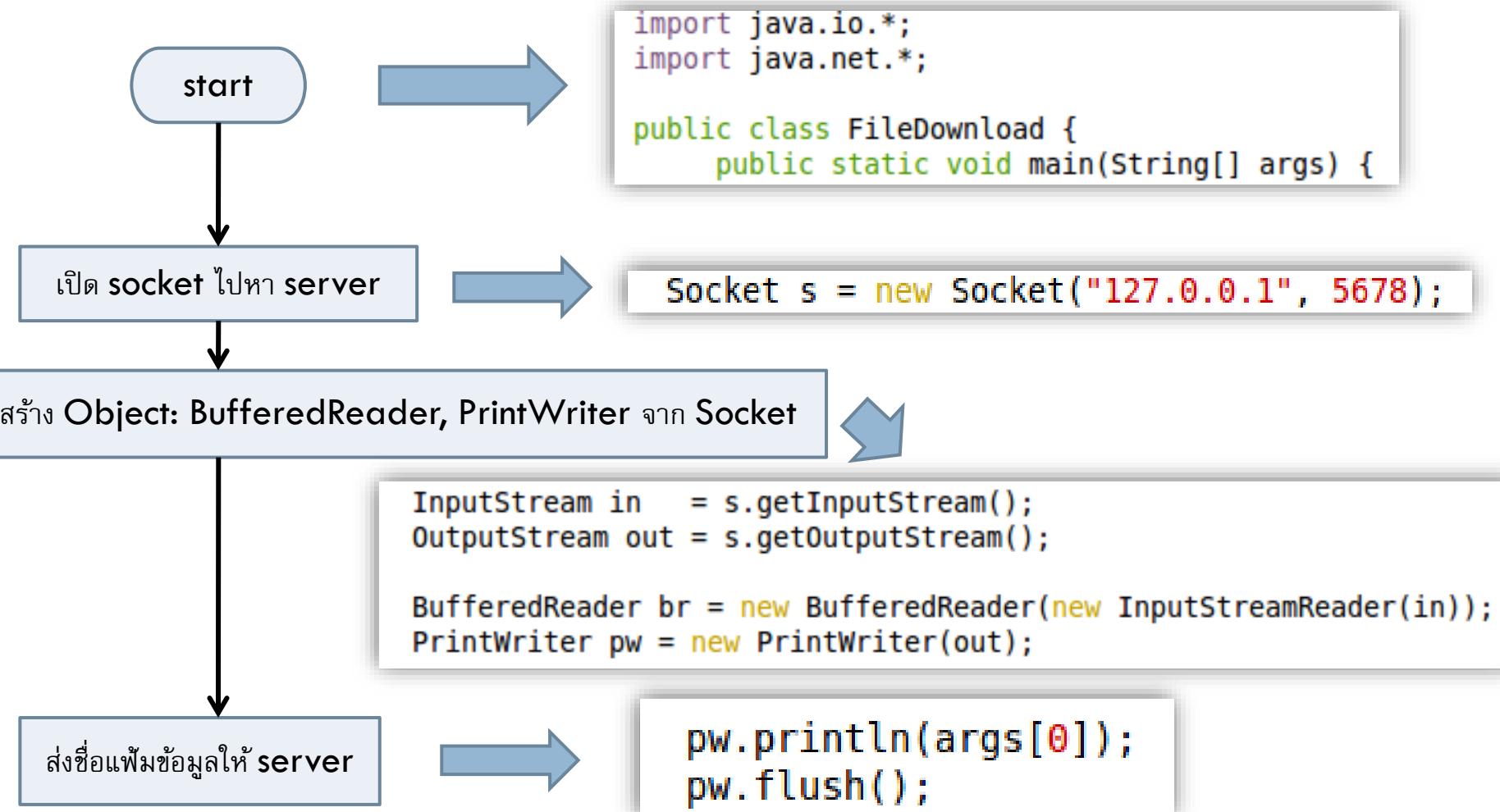
การ Download แฟ้มข้อมูลจาก Server



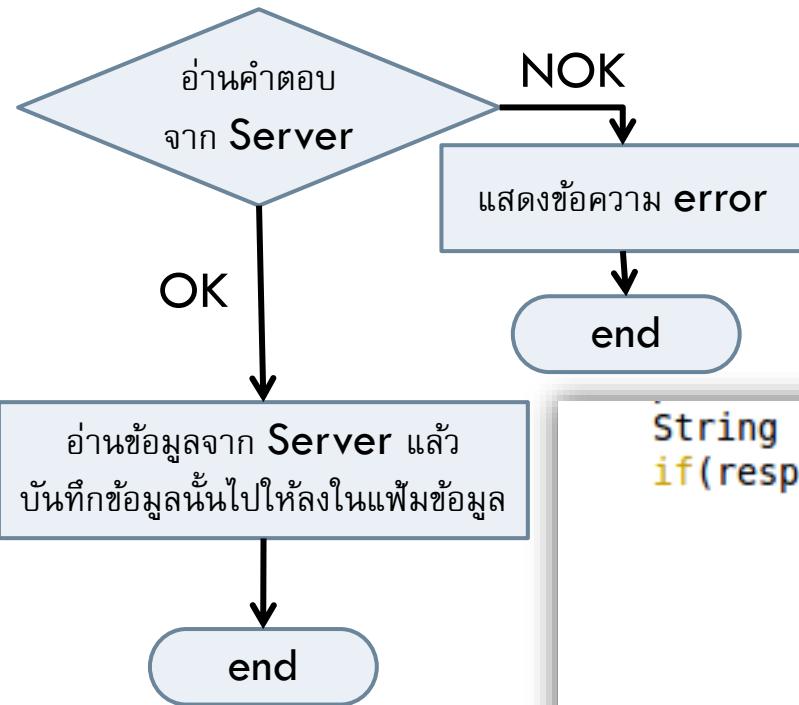
Flow Chart : Client (Download)



Flow Chart to Code (1)



Flow Chart to Code (2)



```
String response = br.readLine();
if(response.equals("OK")){
    File f = new File(args[0]);
    FileOutputStream fout = new FileOutputStream(f);
    byte[] b = new byte[65536];
    int size;
    while((size = in.read(b)) > 0) {
        fout.write(b, 0, size);
    }
    in.close();
} else {
    System.out.println("No file found on server");
}
```

Source Code : FileDownload.java

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

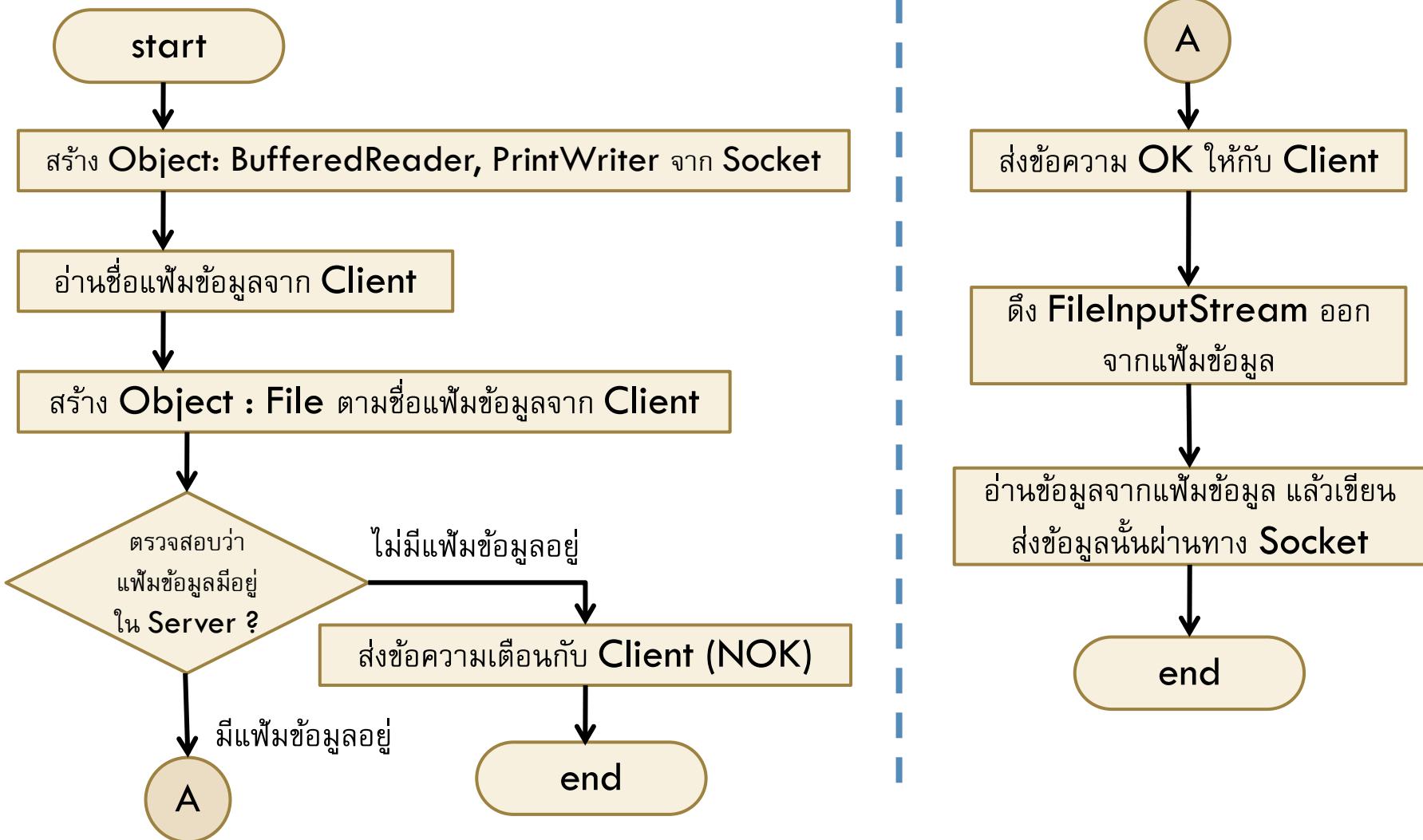
public class FileDownload {
    public static void main(String[] args) {
        try {
            Socket s = new Socket("127.0.0.1", 5678);
            InputStream in = s.getInputStream();
            OutputStream out = s.getOutputStream();

            BufferedReader br = new BufferedReader(new InputStreamReader(in));
            PrintWriter pw = new PrintWriter(out);

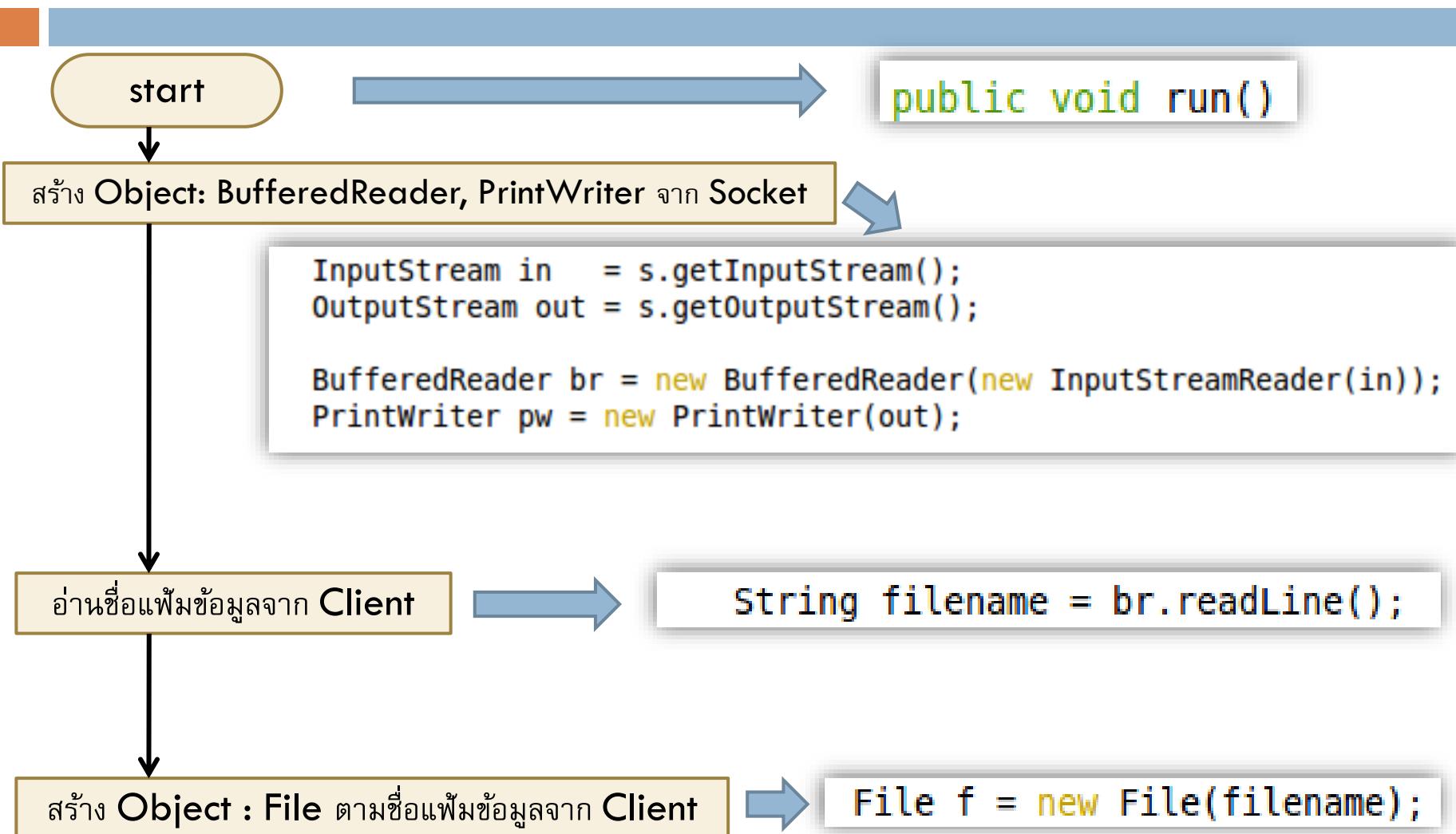
            pw.println(args[0]);
            pw.flush();
            String response = br.readLine();
            if(response.equals("OK")){
                File f = new File(args[0]);
                FileOutputStream fout = new FileOutputStream(f);
                byte[] b = new byte[65536];
                int size;
                while((size = in.read(b)) > 0) {
                    fout.write(b, 0, size);
                }
                in.close();
            } else {
                System.out.println("No file found on server");
            }
            s.close();
        } catch(Exception e) { e.printStackTrace(); }

    }
}
```

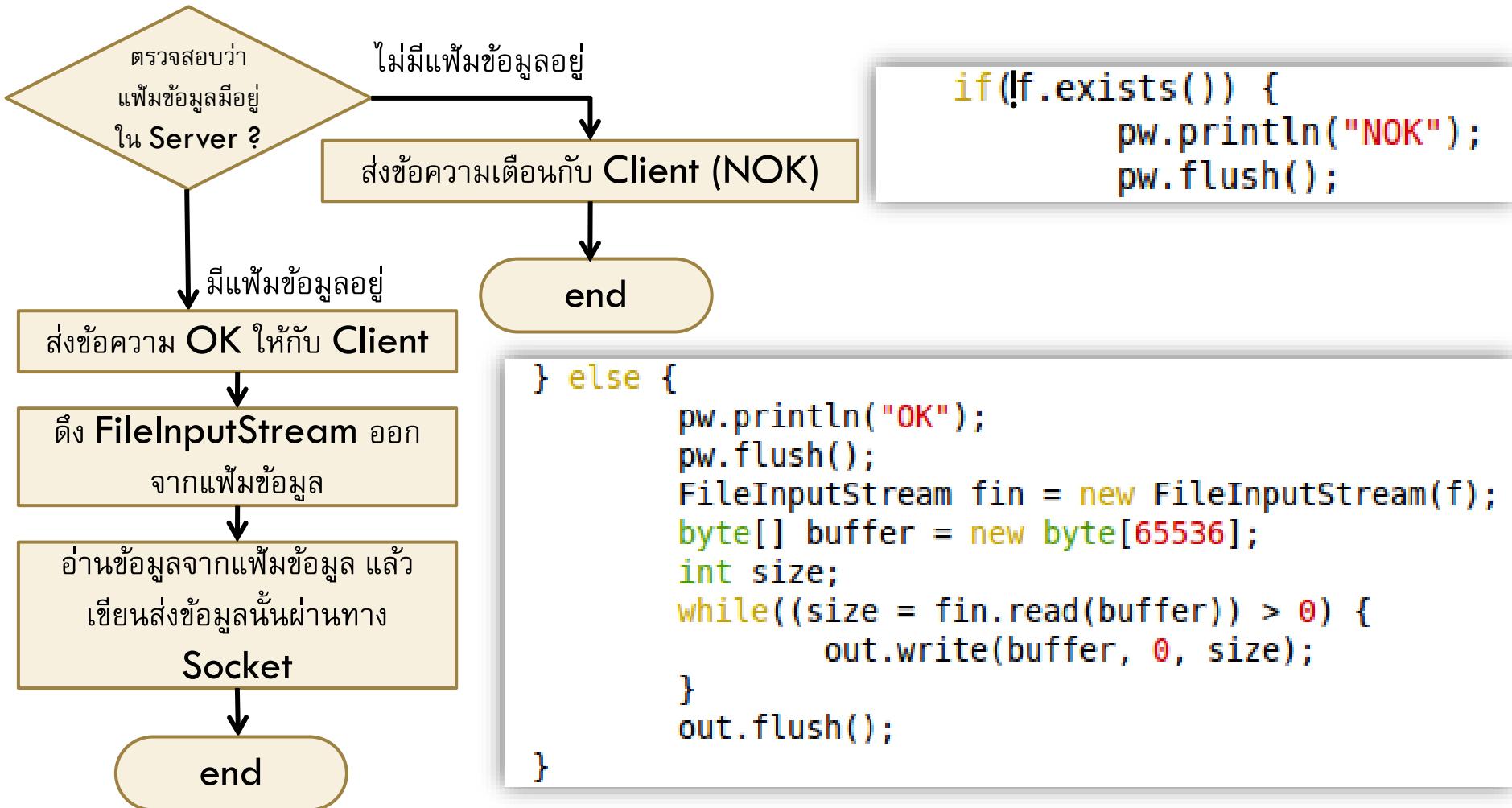
Flow Chart : Server (Download)



Flow Chart to Code (1)



Flow Chard to Code (2)



Source Code : FileServerDownload.java

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

public class FileServerDownload implements Runnable {
    Socket s = null;

    public FileServerDownload(Socket s) {
        this.s = s;
    }

    public void run() {
        try {
            InputStream in = s.getInputStream();
            OutputStream out = s.getOutputStream();

            BufferedReader br = new BufferedReader(new InputStreamReader(in));
            PrintWriter pw = new PrintWriter(out);

            String filename = br.readLine();
            File f = new File(filename);
            if(!f.exists()) {
                pw.println("NOK");
                pw.flush();
            } else {
                pw.println("OK");
                pw.flush();
                FileInputStream fin = new FileInputStream(f);
                byte[] buffer = new byte[65536];
                int size;
                while((size = fin.read(buffer)) > 0) {
                    out.write(buffer, 0, size);
                }
                out.flush();
            }
            out.close();
            s.close();
        } catch (Exception e) { e.printStackTrace(); }
    }
}

public static void main(String[] args) {
    try {
        ServerSocket serv = new ServerSocket(5678);
        while(true) {
            Socket s = serv.accept();
            FileServerDownload fs = new FileServerDownload(s);
            Thread t = new Thread(fs);
            t.start();
        }
    } catch(Exception e) { e.printStackTrace(); }
}
```

JAVA DATAGRAM

030523313 - Network programming
Asst. Prof. Dr. Choopan Rattanapoka

UDP (User Datagram Protocol)

- นอกจากการสื่อสารระหว่างเครื่องคอมพิวเตอร์ในเครือข่ายด้วย TCP แล้ว ยังมีอีก protocol ที่ใช้กันคือ UDP
- UDP จะมี **header** ที่เล็กกว่า TCP ทำให้ส่งข้อมูลขนาดเล็กได้เร็วกว่า
- แต่ UDP จะไม่มีการสถาปนาการเชื่อมต่อ และ ไม่รับประกันข้อมูลซึ่งข้อมูลที่ถูกส่งอาจจะสูญหายระหว่างทางได้
- การส่งข้อมูลผ่าน TCP จะรับประกันการรับข้อมูลตามลำดับ (**seq number + ack number**) ในการส่ง ในขณะที่ UDP ไม่รับประกันการรับข้อมูลตามลำดับ

ทำไมถึงยังมีการใช้ UDP

- ทำไมถึงมี **network application** ที่ใช้ UDP ซึ่งไม่มีการรับประกันข้อมูลที่อาจจะสูญหายระหว่างทาง
- UDP ไม่เหมาะสมอย่างยิ่งกับงานที่ต้องการความถูกต้องของข้อมูล เช่น **FTP(File Transfer Protocol)**
- แต่ UDP เหมาะกับงานที่ต้องการความเร็วในการรับส่งข้อมูล โดยจำเป็นต้องได้รับข้อมูลที่ถูกต้อง
 - Real-time audio, video
 - NTP (Network Time Protocol) → ถ้าไม่ได้เวลาที่จะมา synchronize ก็ไม่เป็นไรมาก
 - DNS → ถ้า resolve IP ไม่ได้ก็ติดต่อใหม่ได้
- การจัดการกับลำดับข้อมูลที่ต้องการรับสามารถจัดการในชั้นของ **application** ได้

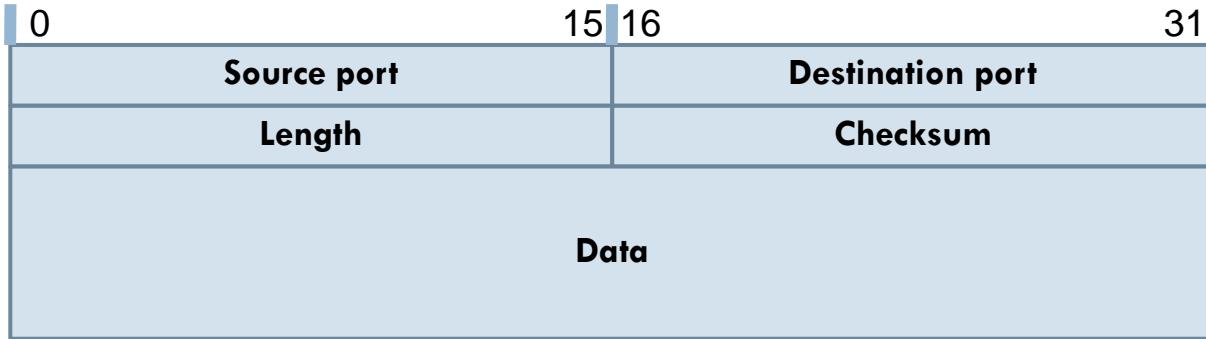
UDP ใน Java

- การใช้งาน UDP ใน `java` จะแบ่งออกเป็น 2 **classes**
 - **DatagramPacket**
 - เป็น **class** ที่ใช้เพื่อ ใส่ข้อมูลที่ต้องการจะส่ง และ ดึงข้อมูลที่ต้องการจะรับ
 - **DatagramSocket**
 - เป็น **class** ที่ใช้เพื่อ ส่งและรับ ข้อมูลของ UDP
- สรุป ก็คือ
 - การส่งข้อมูล บรรจุข้อมูลลงใน **DatagramPacket** และใช้ **DatagramSocket** เป็นตัวส่ง **DatagramPacket** ที่มีข้อมูล ไปยังเครื่องปลายทาง
 - การรับข้อมูล อ่านข้อมูล **DatagramPacket** จาก **DatagramSocket** และ ดึงข้อมูลออกจาก **DatagramPacket** ที่ได้รับ

ข้อแตกต่างระหว่าง UDP และ TCP ใน Java

- ข้อ 1
 - TCP จะมี class : **Socket** และ **ServerSocket**
 - UDP จะไม่มี **ServerSocket** เนื่องจากจะใช้ **DatagramSocket** ทำงาน
เหมือน **Socket** และ **ServerSocket** ในตัว
- ข้อ 2
 - TCP มี **InputStream + OutputStream (Streaming)**
 - UDP ไม่มี **InputStream + OutputStream** จะจัดการข้อมูลในแต่ละ **DatagramPacket**
- ข้อ 3
 - TCP จะติดต่อกันเป็นคู่ **Client+Server** เมื่อเปิด **Socket**
 - UDP สามารถจะทำงานระหว่าง host ไหนก็ได้เนื่องจากไม่มีการสถาปนาการเชื่อมต่อ

ขนาดของข้อมูลที่ส่งผ่าน UDP



- ถ้าดูที่ UDP header จะเห็นว่า Length ใน header มีขนาด 16 บิต
- ความยาวนี้คือความยาวทั้ง datagram ซึ่งหมายความว่า Data + UDP header และ จะมีขนาดมากสุดคือ 65536
- ดังนั้น UDP header มีขนาด 64 บิต ทำให้ ข้อมูลที่ UDP จะส่งได้จริงต่อ datagram จะต้องไม่เกิน 65742 บิต (ประมาณ 8 kB)
- แต่อย่างไรก็ตาม UDP จะทำงานผ่าน IP ซึ่งมีการกำหนดขนาดของข้อมูลแค่ 65536 บิต รวม IP header ด้วย ดังนั้นพยายามอย่าให้ ข้อมูลที่ส่งผ่าน UDP เกิน 8kB
- แต่ในการใช้งานส่วนใหญ่ application จะพยายามกันไม่รับ UDP datagram ที่มีขนาดใหญ่กว่า 576 bytes รวม data + UDP header

Class : DatagramPacket

- DatagramPacket มี constructor ที่สำคัญแบ่งออกเป็น 2 ส่วนคือ
 - ▢ Constructor สำหรับรับข้อมูล
 - public DatagramPacket(byte[] buffer, int length)
 - public DatagramPacket(byte[] buffer, int offset, int length)
 - ▢ Constructor สำหรับส่งข้อมูล
 - public DatagramPacket(byte[] data, int length,
InetAddress destination, int port)
 - public DatagramPacket(byte[] data, int offset, int length,
InetAddress destination, int port)

Get Method ใน Class DatagramPacket

- ใน DatagramPacket จะมีเมธอดเกี่ยวกับการดึงข้อมูลออกที่สำคัญอยู่ 5 เมธอด
 - **public InetAddress getAddress()**
 - ดึงค่า InetAddress ที่เป็นเครื่องปลายทาง
 - **public int getPort()**
 - ดึงค่าหมายเลข port ของเครื่องปลายทาง
 - **public int getLength()**
 - คืนจำนวน byte ของข้อมูลใน datagram
 - **public int getOffset()**
 - คืนค่า offset ที่เป็นจุดเริ่มต้นของข้อมูล
 - **public byte[] getData()**
 - ดึงข้อมูลออกจาก datagrampacket ปกติแล้วควรจะแปลงข้อมูลให้อยู่ในรูปที่นำไปใช้ได้เลย เช่น (กำหนด dp เป็น object ของ DatagramPacket)
 - `String s = new String(dp.getData(), "ASCII");`
 - `InputStream in = new ByteArrayInputStream(dp.getData(), dp.getOffset(), dp.getLength());`

ตัวอย่าง 1

```
import java.net.*;

public class DatagramExample {
    public static void main(String[] args) {
        String s = "This is a test";

        byte[] data = s.getBytes();
        try {
            InetAddress ia = InetAddress.getByName("cit.kmutnb.ac.th");
            int port = 7;
            DatagramPacket dp = new DatagramPacket(data, data.length, ia, port);

            System.out.println("This packet will send to " + dp.getAddress());
            System.out.println("Port = " + dp.getPort());
            System.out.println("Length of data = " + dp.getLength());
            System.out.println("Data --> " +
                               new String(dp.getData(), dp.getOffset(), dp.getLength()));
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

This packet will send to cit.kmutnb.ac.th/202.44.36.4
Port = 7
Length of data = 14
Data --> This is a test

Set Method ใน Class DatagramPacket

- ปกติการสร้าง **DatagramPacket** ด้วย **constructor** เพียงพออยู่แล้ว ยกเว้นในกรณีที่ต้องการแก้ไขค่าโดยไม่สร้าง **object** ใหม่เพื่อความเร็วในการใช้งาน มีเมธอดสำคัญอยู่ 4 เมธอด
 - **public void setData(byte[] data)**
 - **public void setData(byte[] data, int offset, int length)**
 - สำหรับต้องการเปลี่ยนค่า **data** ที่ต้องการจะส่ง
 - **public void setAddress(InetAddress remote)**
 - **public void setPort(int port)**
 - **public void setLength(int length)**

Class : DatagramSocket

- Socket ใน **datagram** จะถูกผูกอยู่กับ **port** ซึ่งจะใช้ทั้งรับและส่งข้อมูล
 - ในการนี้ที่เขียน **client** ไม่จำเป็นจะต้องระบุ **port** สามารถปล่อยให้ระบบหา **port** ที่ว่างให้เอง (**anonymous port**)
 - ในการนี้ของ **Server** จะต้องกำหนด **port** เพื่อให้ **Client** สามารถติดต่อเข้ามาได้
 - ดังนั้นเมื่อต้องการเขียนโปรแกรมที่เป็น **Server** จะต้องเลือกใช้ **constructor** ที่สามารถระบุหมายเลข **port** ได้

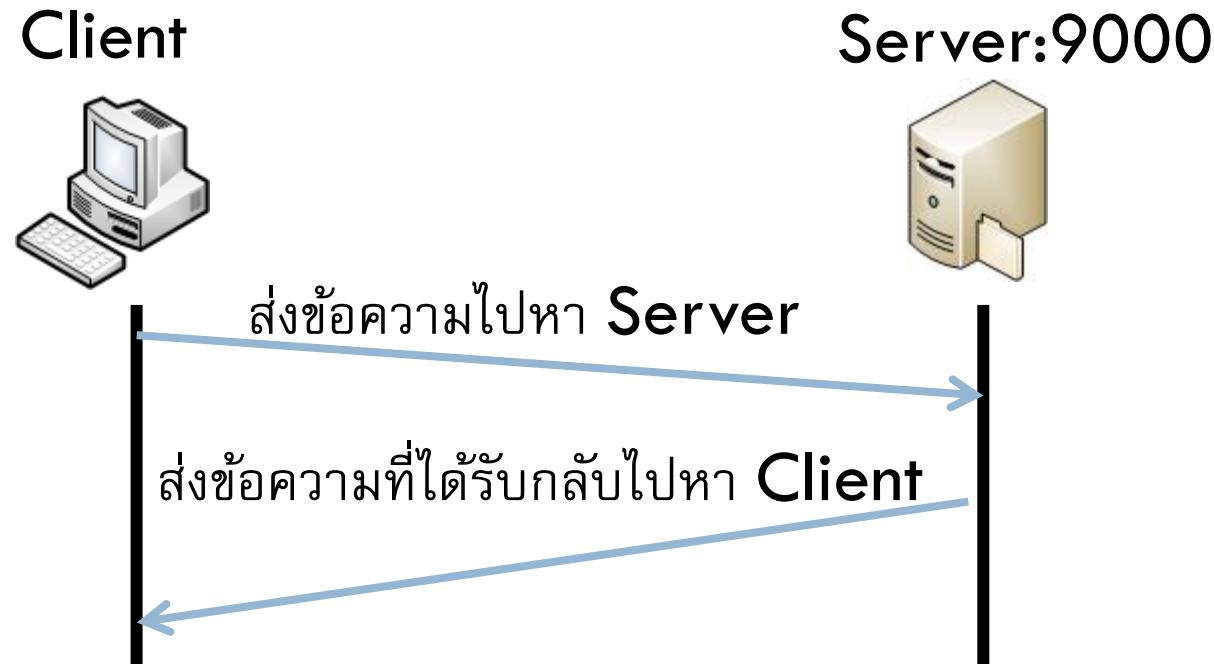
Constructor : DatagramSocket

- ใน DatagramSocket มี constructor อよู่ 3 แบบ ซึ่งจะถูกเรียกใช้ในเหตุการณ์ที่ต่างกัน
 - เปิด socket แบบ anonymous port
 - public DatagramSocket() throws SocketException
 - เปิด socket บน port ที่จะจะไว้บนทุก network interface
 - public DatagramSocket(int port) throws SocketException
 - เปิด socket บน port ที่จะจะไว้บน network interface ที่จะจะ
 - public DatagramSocket(int port, InetAddress address) throws SocketException
- Port ที่กล่าวถึงคือ local port หรือเครื่องตัวเอง
- เตือน : remote IP, remote port จะตั้งค่าใน DatagramPacket

การส่งและรับข้อมูล datagram

- เป้าหมายหลักของ **DatagramSocket** คือ การส่งและรับข้อมูล
- **public void send(DatagramPacket dp)**
 - ใช้สำหรับส่งข้อมูล **datagram** ไปยังเครื่องปลายทาง
- **public void receive(DatagramPacket dp)**
 - ใช้สำหรับรับข้อมูล **datagram** จากเครื่องปลายทาง
 - การรอข้อมูลจะ **block** ที่คำสั่งนี้จนกว่าจะได้รับข้อมูล
- **public void close()**
 - ใช้สำหรับคืน **port** ที่ใช้งาน
- **public int getLocalPort()**
 - คืนค่าหมายเลข **port** ซึ่งทำให้รู้ว่าระบบจัดสรร **port** หมายเลขอะไรมาให้เรา

Example : Echo Server / Echo Client



Example : EchoClient

```
import java.net.*;

public class EchoClient {
    public static void main(String[] args) {

        try {
            byte[] sendBuffer;
            byte[] recvBuffer = new byte[512];
            InetAddress ia = InetAddress.getByName("127.0.0.1");

            DatagramSocket socket = new DatagramSocket();

            String msg = "Hello";
            sendBuffer = msg.getBytes();

            DatagramPacket sendDP = new DatagramPacket(sendBuffer, sendBuffer.length, ia, 9000);
            socket.send(sendDP);

            DatagramPacket recvDP = new DatagramPacket(recvBuffer, recvBuffer.length);
            socket.receive(recvDP);
            String s = new String(recvDP.getData(), 0, recvDP.getLength());
            System.out.println("Server --> " + s);
            socket.close();

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

Example : EchoServer

```
import java.net.*;

public class EchoServer {
    public static void main(String[] args) {

        try {
            DatagramSocket socket = new DatagramSocket(9000);
            while(true) {
                byte[] recvBuffer = new byte[8000];

                DatagramPacket dp = new DatagramPacket(recvBuffer, recvBuffer.length);
                socket.receive(dp);
                DatagramPacket dp2 = new DatagramPacket(dp.getData(),
                                              dp.getLength(),
                                              dp.getAddress(),
                                              dp.getPort());
                socket.send(dp2);
            }
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Class : Date

- ในภาษา Java สามารถดึงเวลาของเครื่องคอมพิวเตอร์ออกมายได้โดยใช้ Class ที่ชื่อว่า Date
- Date มี constructor หลายแบบแต่ถ้าต้องการเวลาปัจจุบันสามารถใช้ constructor ที่ง่ายที่สุดคือ public Date()
 - Date now = new Date();
- เมธอดใน Class Date สามารถดึงค่า วัน เดือน ปี ชั่วโมง นาที ได้ แต่ถ้าต้องการทั้งหมด สามารถใช้เมธอด **toString()**;
 - String date = now.toString();

ตัวอย่าง

```
import java.util.*;  
  
public class DateTime {  
    public static void main(String[] args) {  
        Date now = new Date();  
        System.out.println("Current Time = " + now.toString());  
    }  
}
```

Current Time = Fri Jan 28 09:35:29 ICT 2011