

Agenda



❖ Day 2

- Chapter 4 : Exception Handling
- Chapter 5 : Using Java Classes
 - 5.1 String
 - 5.2 Wrapper Class
 - 5.3 Collection and Map
- Chapter 6 : Advance Topics
 - Networking Programming
 - Stream and Serialization
 - JDBC



Chapter 1 Introduction to Java



By Mongkol Puengpipattrakul
MFEC Public Co., Ltd.



Chapter 1

Introduction to Java



- ประวัติของภาษา Java
- ข้อดีของภาษา Java
- Java Platform
- Java Development
- Lab 1 : First Java Program



ประวัติของภาษา Java



- C/C++ Problem
 - ต้อง Compile ใหม่เมื่อทำงานบนหน่วยประมวลผลที่แตกต่างกัน
 - เป็นภาษาที่ไม่ปลอดภัย Pointer สามารถใช้งานได้โดยไม่ต้องจำกัด
- Green Group โดย Sun Microsystem พัฒนาภาษาใหม่ขึ้นใช้งานชื่อว่า Oak
 - ง่ายต่อการเรียนรู้
 - เป็นภาษาที่มีความปลอดภัย
 - เป็น OOP
 - ทำงานผ่าน interpreter -> Platform Independent



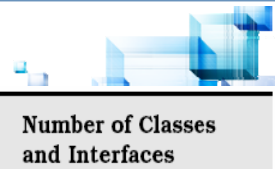
ประวัติของภาษา Java



- Oak ไม่ประสบความสำเร็จในตลาด
- ปี 1993 HTML และ Browser เริ่มเกิดขึ้น
- Sun ต้องการภาษาที่เขียนโปรแกรมที่สามารถทำงานได้บนเครื่อง Computer ใดๆ
- ปัดฝุ่น Oak → Java
- สร้าง WebRunner : Web Browser ที่สามารถรันโปรแกรมภาษา Java ได้
- Java ได้รับความนิยมน Microsoft, IBM ประกาศสนับสนุน Java
- Netscape ออก Netscape 2.0 สามารถรันโปรแกรมภาษา Java ได้
- ปี 1995 Sun เผยแพร่ JDK 1.0 ทาง Internet



ประวัติของภาษา Java



Version	Year	New Language Features	Number of Classes and Interfaces
1.0	1996	The language itself	211
1.1	1997	Inner classes	477
1.2	1998	None	1,524
1.3	2000	None	1,840
1.4	2004	Assertions	2,723
5.0	2004	Generic classes, "for each" loop, varargs, autoboxing, metadata, enumerations, static import	3,279
6	2006	None	3,777



ข้อดีของภาษา Java



- เป็นภาษาเชิงวัตถุ (**Object-Oriented Programming**)
 - ง่ายต่อการพัฒนาระบบขนาดใหญ่ที่มีความซับซ้อน
- **Platform Independent**
 - Write Once – Run Any Where
- ความปลอดภัยสูง
 - No Pointer(ไม่มีการใช้ตัวแปรชนิด pointer)
- ❖ การป้องกันการผิดพลาด (Robust)
 - มีการตรวจสอบความผิดพลาดต่าง ๆ เช่น ขั้นตอนคอมไพล์, ขั้นตอนการรัน



ข้อดีของภาษา Java



- สนับสนุนงานหลายระดับ
 - J2EE / J2SE / J2ME
- มี **Opensource Library** ให้ใช้มากมาย
- กลไกในการคืนพื้นที่ในหน่วยความจำอัตโนมัติ (garbage collection)
- ฟรี



Java Platform



❖ Java 2 Platform, Standard Edition (J2SE)

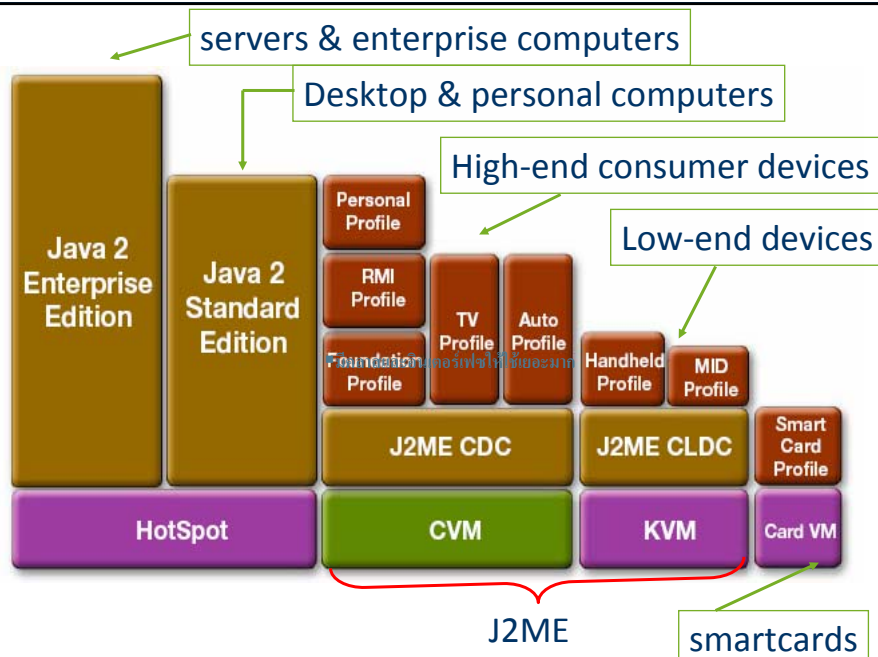
- Javaแอปพลิเคชัน (Java application)
- แอปเพลต (Java applet)

❖ Java 2 Platform, Enterprise Edition (J2EE)

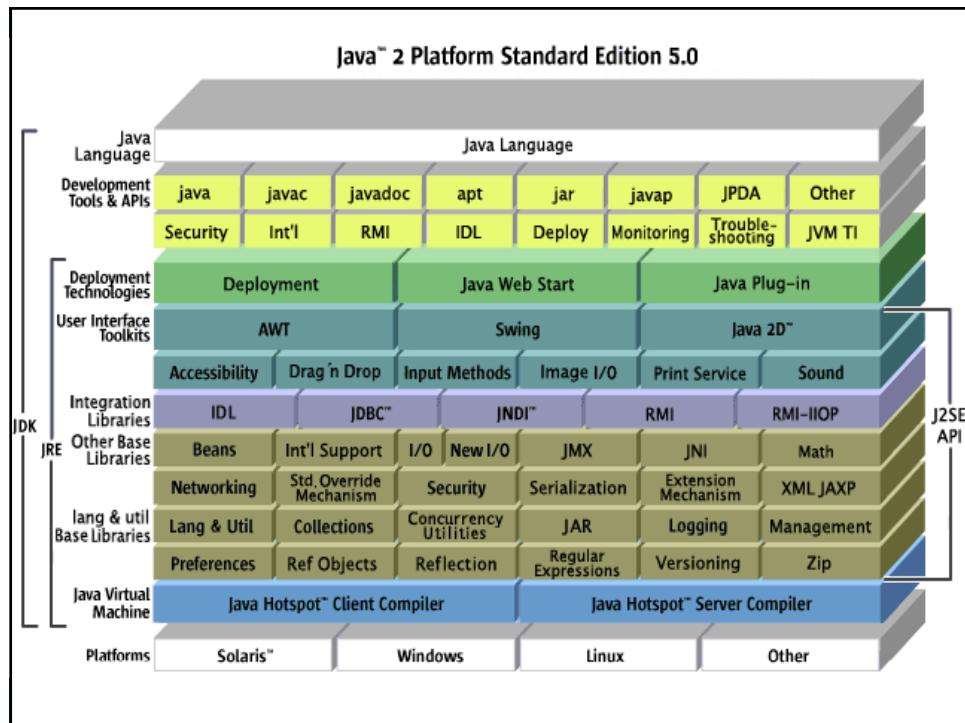
- โปรแกรมแบบมัลติ-tier (multitiered) สำหรับการพัฒนาโปรแกรมในระดับองค์กร

❖ Java 2 Platform, Micro Edition (J2ME)

- สินค้าอิเล็กทรอนิกส์ เช่น โทรศัพท์มือถือ พีดีเอ และกล้องเว็บแคม



Java 2 Platform editions and their target markets



Compiling and Running

❖ ชุดพัฒนาภาษาJava (Java Development Kit - JDK)

- Javaคอมไพเลอร์ (javac.exe)



- สภาพแวดล้อมการรันโปรแกรมJava (Java Runtime Environment - JRE) (java.exe)



❖ Download <http://java.sun.com>

Development Tool

❖ Notepad

- มาพร้อมกับ Windows

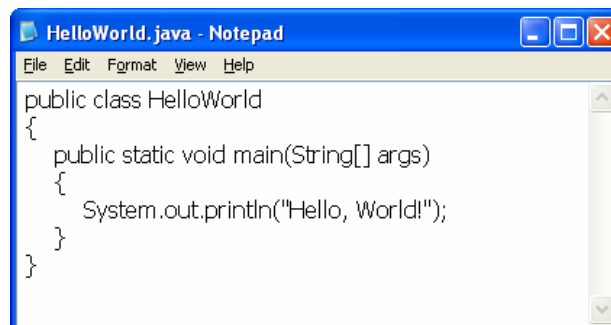
❖ Netbeans

- <http://www.netbeans.org/>

❖ Eclipse

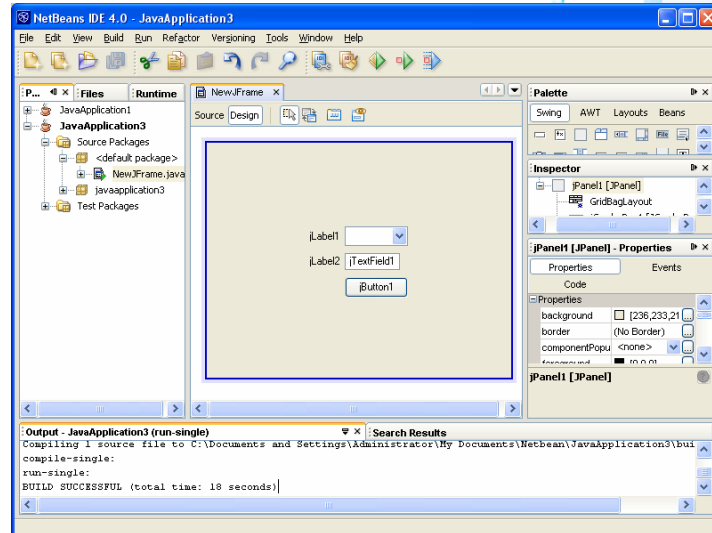
- <http://www.eclipse.org/>

Notepad

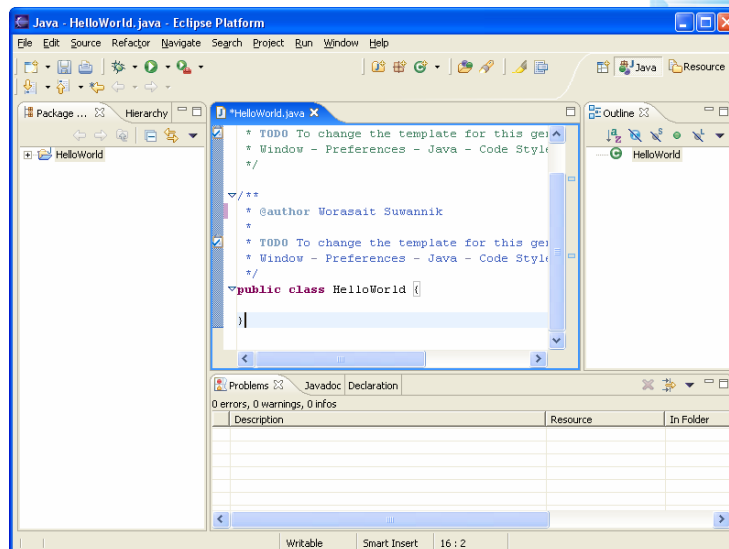


```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```


Netbeans



Eclipse



First Java Program

1. Setup Environment

■ Install JDK 6.0



- เพิ่ม `/bin` ของ JDK(ex:C:\Program Files\Java\jdk1.6.0_14\bin)
ในตัวแปร `PATH` ของ `System Variable`
 - เพื่อให้สามารถเรียกใช้ `javac.exe` ที่ command line ได้

First Java Program

2. สร้าง Student.java

```
public class Student {  
  
    public void sayHi(){  
        System.out.println("Hello Everybody!!!");  
    }  
  
    public static void main(String[] args) {  
  
        Student student = new Student();  
        student.sayHi();  
  
    }  
}
```

First Java Program

3. Compile and Run

- ที่ command line
- Compile ด้วย javac.exe
 - javac Student.java
- Run ด้วย java.exe
 - java Student
- ปรากฏข้อความ
 - Hello Everybody!!!



Summary

- เมื่อ Compile Student.java ด้วย javac จะได้ Student.class



- เมื่อ Run Student.class ด้วย java โปรแกรมจะทำงาน



Summary



- โดยปกติ 1 File ควรมี Class เดียว และ ชื่อ File ควรเหมือนกับชื่อ Class
- Method main() จะเป็นจุดเริ่มต้นของการทำงาน
 - `public static void main (String args[])`



Chapter 2 Java Programming Language



By Mongkol Puengpipattrakul
MFEC Public Co., Ltd.



Chapter 2

Java Programming Language



- 2.1 Declaration
- 2.2 Operation
- 2.3 Flow Control



Chapter 2

Java Programming Language



2.1 Declaration



Identifier



■ Identifier

- ชื่อที่ผู้เขียนสามารถตั้งขึ้นได้ เพื่อใช้เป็นชื่อของ
 - Class
 - Interface
 - Variable
 - Method
 - Label
 - File



Legal Identifier



- **Java** กำหนดเงื่อนไขในการตั้งชื่อ **identifier** ดังนี้
 - สามารถประกอบด้วย
 - ตัวอักษรตัวเล็ก ตัวใหญ่ (เช่น a, z, A, Z)
 - ตัวเลข
 - เครื่องหมาย Under Score (_)
 - เครื่องหมาย Dollar Sign (\$)
 - แต่ต้องขึ้นต้นด้วยตัวอักษรเท่านั้น
 - ห้ามตั้งชื่อด้วยคำสงวน (reserved word)



Reserved Words

abstract, assert, boolean, break, byte,
case, catch, char, class, const, continue,
default, do, double, else, enum, extends,
final, finally, float, for, goto, if,
implements, import, instanceof, int, interface,
long, native, new, package, private, protected,
public, return, short, static, strictfp, super,
switch, synchronized, this, throw, throws,
transient, try, void, volatile, while



Type

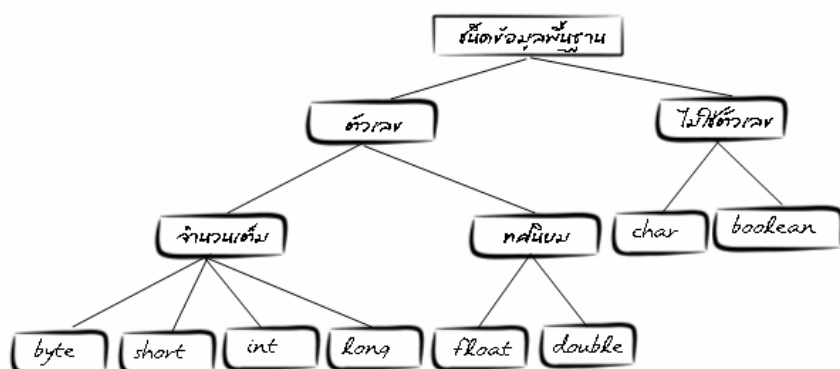
- **Type**
 - Primitive Type
 - Class
 - Interface
 - Array



Primitive Type

ชนิดข้อมูล	จำนวนบิต	ช่วงของค่าที่เก็บได้
boolean	แล้วแต่ JVM จะกำหนด	true หรือ false
char	16 บิต	ใช้เก็บอักขระที่มีรหัสตั้งแต่ 0 ถึง 65535
byte	8 บิต	-128 ถึง 127
short	16 บิต	-32768 ถึง 32767
int	32 บิต	-2147483648 ถึง 2147483647
long	64 บิต	เลขลบเยอะมาก ถึง เลขบวกเยอะมาก
float	32 บิต	เก็บเลขทศนิยมและเลขยกกำลัง
double	64 บิต	เก็บเลขทศนิยมและเลขยกกำลังได้ละเอียดกว่า float

Primitive Type



Primitive Type

```
public class PrimitiveType {  
    public static void main(String args[]) {  
        boolean testBoolean = true;  
        char testChar = 'A';  
        byte testByte = 127;  
        short testShort = 32767;  
        int testInt = 2147483467;  
        long testLong = 9999999999999999991;  
        float testFloat = 3.1415926535897896F;  
        double testDouble = 3.1415926535897896D;  
    }  
}
```

Class

- See Chapter 3

Interface

- See Chapter 3

Array

- ชุดของตัวแปร **Type** เดียวกัน
- ใช้เพื่อรวมตัวแปร **Type** เดียวกันจำนวนมากเป็นหนึ่งหน่วย
- Syntax

```
<type>[] <array name> = new <type>[<size>]
```

OR

```
<type> <array name>[] = new <type>[<size>]
```

Array



```
class ArrayInit {  
    public static void main(String args[]) {  
        int[] i = {1, 2, 3, 5, 7, 7+4};  
        char[] c = {'H', 'e', 'l', 'l', 'o'};  
        String[] m = {"How", "do", "you", "do?"};  
  
        Student[] studentArr1 =  
            { new Student(), new Student(), new Student()};  
  
        Student[] studentArr2 = new Student[3];  
        studentArr2[0] = new Student();  
        studentArr2[1] = new Student();  
        studentArr2[2] = new Student();  
    }  
}
```



Chapter 2 Java Programming Language



2.2 Operator



Assignment Operator



❖ รูปแบบ

- ตัวแปร = ค่าที่ต้องการกำหนด;

❖ ตัวอย่าง

- `count = 3;`
- `benefit = income - outcome`



Arithmetic Operator



❖ Integer / Floating Point Arithmetic Operator

- +
- -
- *
- /
- % (mod)

```
System.out.println(1 + 2);  
System.out.println(2 - 3);  
System.out.println(3 * 4);  
System.out.println(4 / 5);  
System.out.println(6 % 7);
```



Arithmetic Operator



❖ Arithmetic Assignment Operator

- +=
- -=
- *=
- /=

```
int x = 1;
x += 1; System.out.println(x);
x *= 2; System.out.println(x);
x -= 3; System.out.println(x);
x /= 4; System.out.println(x);
float y = 1f;
y += 1; System.out.print(y);
y *= 2; System.out.print(y);
y -= 3; System.out.print(y);
y /= 4; System.out.println(y);
```

Arithmetic Operator



❖ Increment and Decrement

- ++
 - x++
 - ++x
- --
 - x--
 - --x

```
int x = 1, y = 1;
System.out.println(x++ + ", " + ++y);
System.out.println(x-- + ", " + --y);
```

Bitwise Operator



❖ Arithmetic Assignment Operator

- ~ (bitwise unary NOT)
- & (bitwise AND)
- | (bitwise OR)
- ^ (bitwise EXECUSIVE OR)
- >> (bitwise SHIFT RIGHT)
- >>> (bitwise SHIFT RIGHT ZERO FILL)
- << (bitwise SHIFT LEFT)



Bitwise Operator



```
int a = 9;
int b = 8;

System.out.println("a      = " + Integer.toBinaryString(a) );
System.out.println("b      = " + Integer.toBinaryString(b) );
System.out.println("a&b    = " + Integer.toBinaryString(a&b) );
System.out.println("a|b    = " + Integer.toBinaryString(a|b) );
System.out.println("a^b    = " + Integer.toBinaryString(a^b) );
System.out.println("a >> 4 = " + Integer.toBinaryString(a>>4));
System.out.println("a << 4 = " + Integer.toBinaryString(a<<4));
System.out.println("~a     = " + Integer.toBinaryString(~a));
System.out.println("-a     = " + Integer.toBinaryString(-a));
```



Relational Operator



❖ Relational Operator

- == (equal to)
- != (not equal to)
- > (greater than)
- < (less than)
- >= (greater than or equal to)
- <= (less than or equal to)

```
int a = 0, b = 1;
System.out.println(a == b);
System.out.println(a != b);
System.out.println(a < b);
System.out.println(a <= b);
System.out.println(a > b);
System.out.println(a >= b);
```



Logical Operator



❖ Boolean Logical Operator

- & (logical AND)
- | (logical OR)
- ^ (logical EXCLUSIVE OR)
- ! (logical unary NOT)
- NOTE : ใช้ operator เหมือน Bitwise
 - ถ้า Operand เป็น boolean → compiler ตีความเป็น logical operator
 - ถ้า Operand เป็น number → compiler ตีความเป็น bitwise operator

```
boolean t = true, f = false;
System.out.println(t & f);
System.out.println(t | f);
System.out.println(t ^ f);
System.out.println(!t);
```



Logical Operator

❖ Short-circuit Logical Operator

- && (logical AND)
- || (logical OR)
- NOTE : เหมือน & และ | แต่จะหยุดคำนวณเมื่อผลลัพธ์ของ expression สามารถกำหนดค่าได้แล้ว

```
int x = 0;  
System.out.println(x != 0 && (10 / x) > 1);  
System.out.println(x == 0 || (10 / x) > 1);
```

Conditional Operator

❖ Syntax

- (<condition> ? <expression1> : <expression2>)
- ถ้า condition เป็น true ได้ค่าตาม expression1
- ถ้า condition เป็น false ได้ค่าตาม expression2

```
int a = 1;  
int b = 2;  
System.out.println((a > b)? "a is greater" : "b is greater")
```




2.3 Flow Control

Block

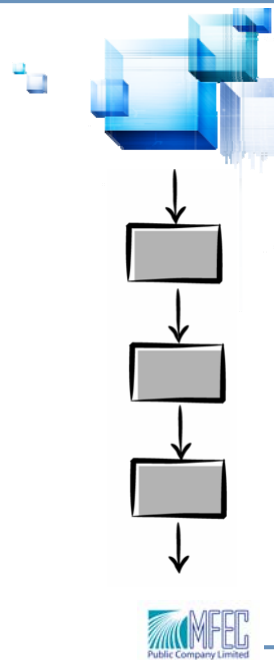


- ขอบเขตระหว่าง { }

```
public class HelloWorld {  
  
    public static void main(String[] args) {  
        System.out.println("Hello World !!!");  
    }  
}
```

การทำงานแบบเรียงลำดับ

```
public class Shopping
{
    public static void main(String[] args)
    {
        int cash    = 500;
        int bookPrice = 180;
        int foodPrice = 20;
        int sodaPrice = 7;
        cash -= bookPrice;
        cash -= foodPrice;
        cash -= sodaPrice;
        System.out.println("Cash = " + cash);
    }
}
```



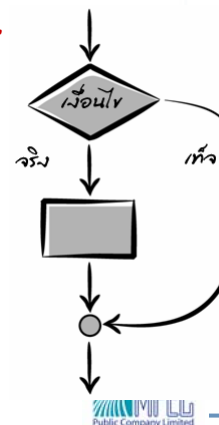
if statement

❖ รูปแบบ

```
if (< เงื่อนไข boolean expression >)
{
    < statements ที่จะทำงานเมื่อเงื่อนไขเป็นจริง >;
}
```

❖ ตัวอย่าง

```
if (cash >= bookPrice) {
    System.out.println("Buy book.");
}
```



Example

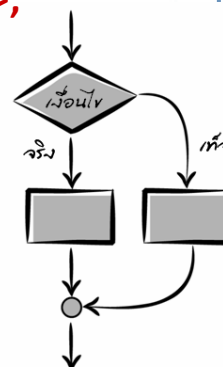
```
public class If {  
    public static void main(String[] args) {  
        int cash    = 200;  
        int bookPrice = 180;  
  
        if (cash >= bookPrice) {  
            cash -= bookPrice;  
            System.out.println("Buy book.");  
        }  
  
        System.out.println("Cash = " + cash);  
    }  
}
```



If - else statement

❖ รูปแบบ

```
if (< เงื่อนไข boolean expression >) {  
    < statements ที่จะทำงานเมื่อเงื่อนไขเป็นจริง >;  
}  
else {  
    < statements ที่จะทำงานเมื่อเงื่อนไขเป็นเท็จ >;  
}
```



Example

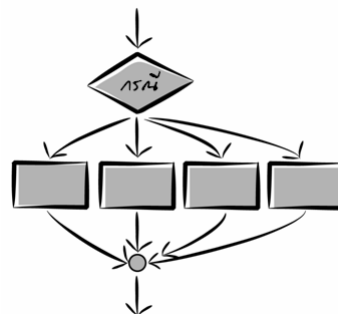
```
int cash = 200;
int price = 380;

if (cash >= price) {
    cash -= price;
    System.out.println("Use cash");
}
else{
    System.out.println("Use card");
}
```

switch - case statement

❖ รูปแบบ

```
switch (<integer expression>) {
    case <value>:
        <statement>;
        break;
    case <value>:
        <statement>;
        break;
    default:
        <statement>;
}
```



Example

```
char grade = 'B';
switch (grade) {
case 'A':
    System.out.println("Average");
    break;
case 'B':
    System.out.println("Boring");
    break;
case 'C':
    System.out.println("Cool!");
    break;
default:
    System.out.println("Fabulous");
}
```



switch - case statement

ข้อควรระวังในการใช้ switch – case statement

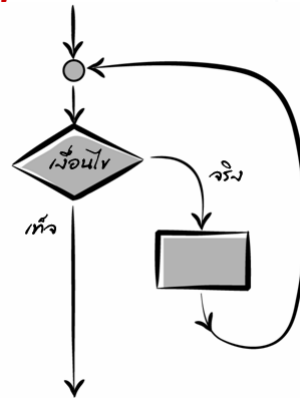
- อย่าลืม break;

```
class Switch3 {
    public static void main(String args[]) {
        char g = args[0].charAt(0);
        switch(g) {
            case 'A':
            case 'B':
            case 'C': System.out.println("Passes"); break;
            case 'D':
            case 'F': System.out.println("Fails"); break;
            default : System.out.println("Invalid");
        }
    }
}
```



while statement

while (<เงื่อนไข เป็น *boolean expression*>){
 <statements ที่ทำซ้ำขณะที่เงื่อนไขเป็นจริง>·
}

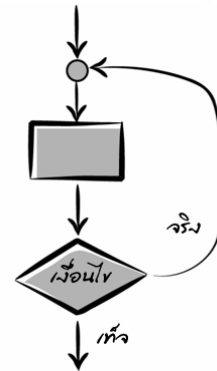


Example

```
int year = 0;  
  
double balance = 100;  
while (balance <= 10000)  
{  
    year++;  
    balance *= 1.05;  
}
```

do – while statement

```
do {  
    <statements>;  
} while( <boolean expression> );
```



MFEC
Public Company Limited

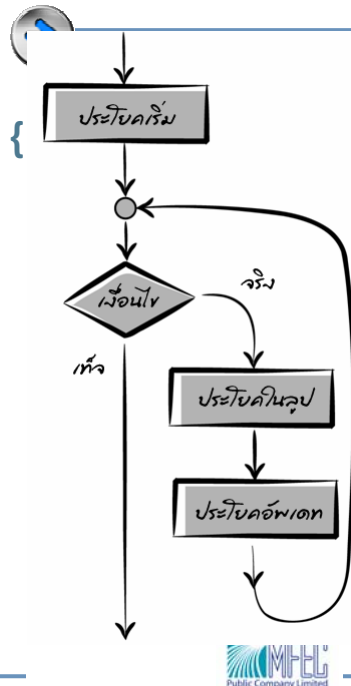
Example

```
int n = 100;  
int i = 1, f = 1;  
do {  
    f *= i;  
} while (i++ < n);  
System.out.println(n + "! = " + f);
```

MFEC
Public Company Limited

for statement

```
for(ประโยคเริ่ม; เงื่อนไข; ประโยคอัปเดต) {  
    ประโยคในลูป;  
}
```



Example

```
double balance = 100;
```

```
for (int i = 0; i < 95; i++)  
{  
    balance *= 1.05;  
}
```

```
System.out.println(balance);
```


for(each) statement



ใช้สำหรับวนเอดำจาก **Array** หรือ **Iterable**

ใช้ได้กับ **JDK 5.0** ขึ้นไป

```
for ( <type> j : <array or iterable> ) {  
    <statements>  
}
```



Example



```
class ForEach {  
    public static void main(String args[]) {  
        int[] i = new int[]{ 1,2,3,4,5 };  
  
        for (int j : i) {  
            System.out.println( j );  
        }  
    }  
}
```



break statement

- ใช้หยุดการทำงาน และออกจาก **Block** ปัจจุบันทันที
- ใช้ได้กับ **switch, while, do** หรือ **for** เท่านั้น

```
class Break1 {  
    public static void main(String args[]) {  
        for (int i = 0; i < 5; i++) {  
            if (i == 3)  
                break;  
            System.out.println(i);  
        }  
        System.out.println("End");  
    }  
}
```



continue statement

- ใช้หยุดการทำงานของ **loop** รอบปัจจุบัน และทำการทดสอบเงื่อนไขเพื่อขึ้นรอบใหม่ทันที
- ใช้ได้กับ **while, do** หรือ **for** เท่านั้น

```
class Continue1 {  
    public static void main(String args[]) {  
        for (int i = 0; i < 5; i++) {  
            if (i == 3)  
                continue;  
            System.out.println(i);  
        }  
        System.out.println("End");  
    }  
}
```



return statement

- ทำการคืนค่าและหยุดการทำงานของ method ทันที
- รูปแบบ
 - `return [<expression>;`

```
class Return {  
    public static void main(String args[]) {  
        for (int i = 0; i < 5; i++) {  
            if (i == 3)  
                return;  
            System.out.println(i);  
        }  
        System.out.println("End");  
    }  
}
```

Chapter 3

Java

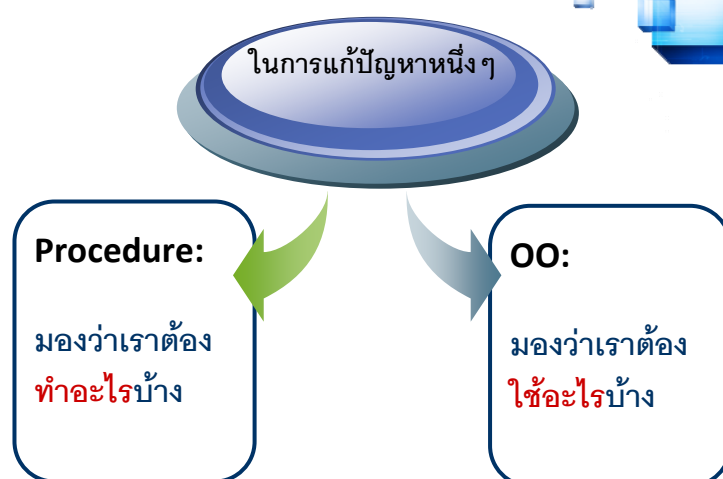
Object-Oriented Programming

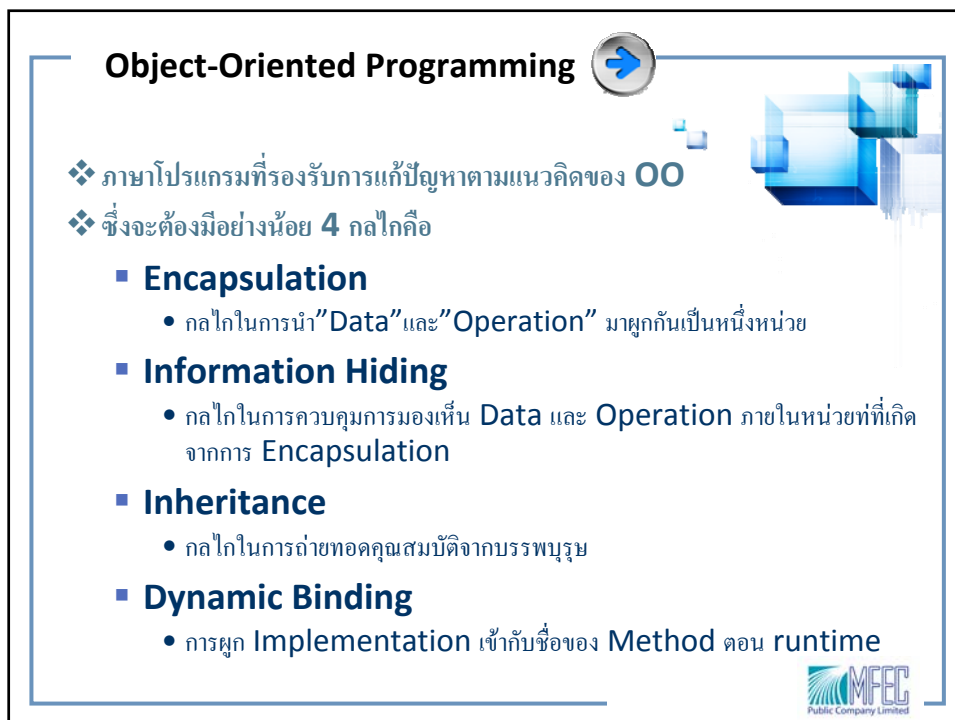
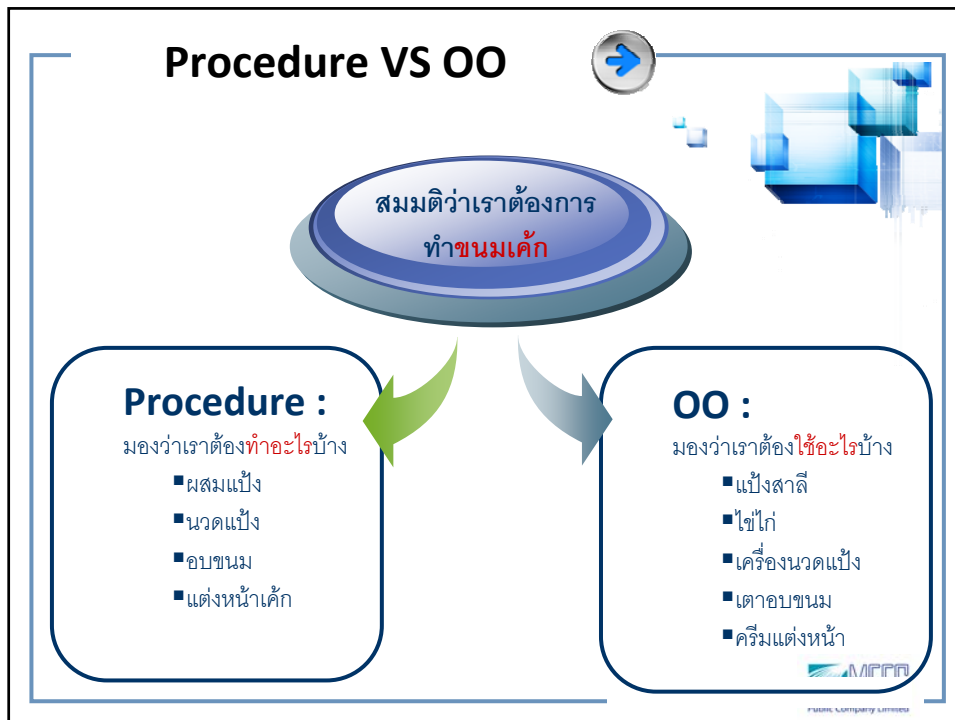
By Mongkol Puengpipattrakul
MFEC Public Co., Ltd.



3.1 OO Concept

Procedure VS OO







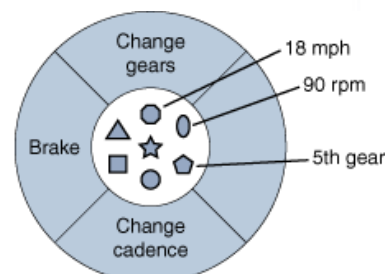
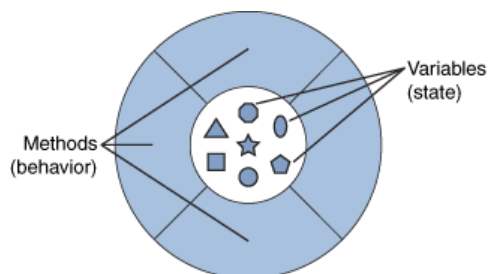
3.2 Java Class

Class

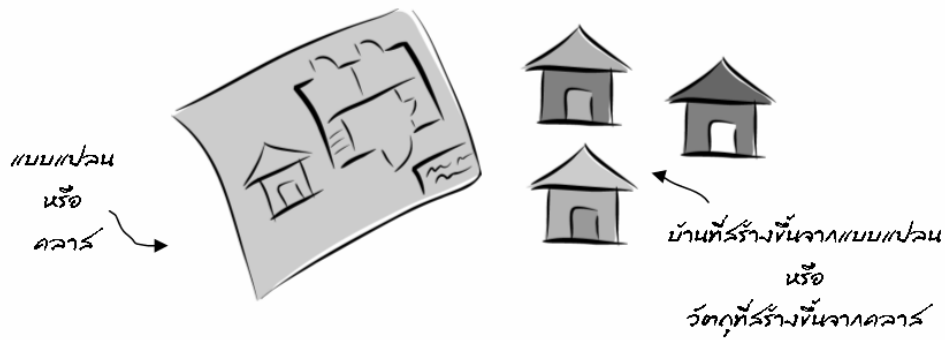


❖ วัตถุต่างๆ ในธรรมชาติสามารถอธิบายได้ด้วย 2 องค์ประกอบ

- **State** : ลักษณะของวัตถุ
- **Behavior** : พฤติกรรมของวัตถุ หรือกิจกรรมที่วัตถุสามารถทำได้



Class and Instance



Level of Abstraction

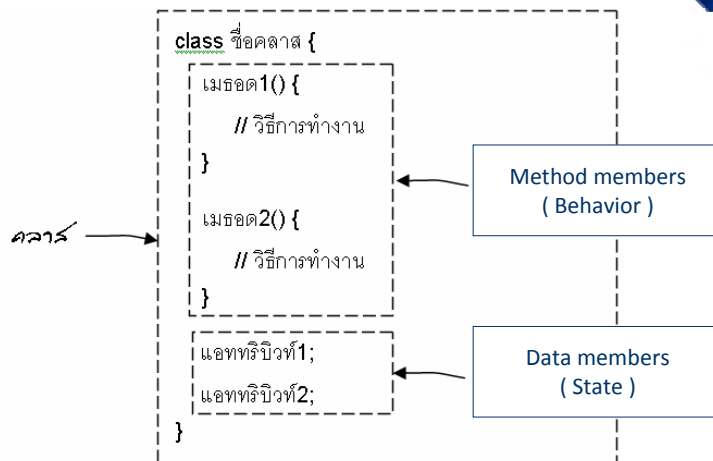
❖ เลือกจำลองลักษณะที่สำคัญ

- จำลองเฉพาะความสามารถของวัตถุที่เราสนใจหรือเกี่ยวข้องกับปัญหาที่เราต้องการจะแก้ไข
- เช่น ถ้าเราต้องการคำนวณหาน้ำหนักเฉลี่ยของแมว เราก็อาจจะให้แมวจำลองมี "น้ำหนัก" เป็นลักษณะที่สำคัญ

❖ ไม่จำลองลักษณะทั้งหมดของวัตถุ

- การจำลองแมวจะไม่จำลองหมวดทุกเส้นของแมว

Java Class



Java Class

```
public class Notebook {  
    String color = "Black";  
    String brand = "HP";  
  
    void start(){  
        System.out.println("Notebook is started");  
    }  
    void shutdown(){  
        System.out.println("Notebook is shutdown");  
    }  
}
```


package

- เป้าหมาย
 - เพื่อป้องกันปัญหาเรื่องชื่อ Class ที่อาจซ้ำกัน
 - การจัด Class แยกออกเป็นหมวดหมู่ จัดเก็บ class ต่าง ๆ ที่เกี่ยวข้องกันไว้ที่เดียวกัน
- ใน package หนึ่งสามารถมี class ไม่จำกัด
- ใน package สามารถมี subpackage อีกเท่าใดก็ได้

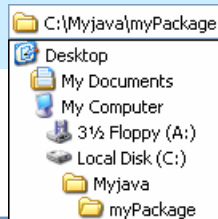
Keyword

- Define โดยการประกาศที่หัว Class โดยใช้ Keyword “package”
- Folder ของ .java file จะต้องสอดคล้องกับชื่อ package ของ Class

```
package myPackage;  
public class A {  
    public void print() {  
        System.out.println("This's class A");  
    }  
}
```

จัดเก็บไฟล์ไว้ที่

C:\Myjava\myPackage



Package

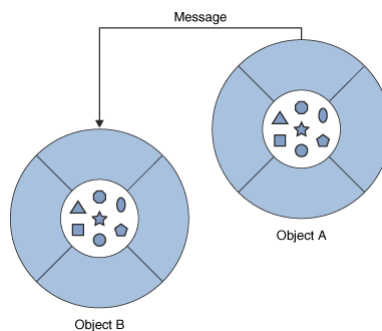
❖ สามารถเรียกใช้ class A ได้ 2 แบบ คือ

```
class PackageTest1{  
    public static void main(String args[]){  
        new myPackage.A().print();  
    }  
}
```

```
import myPackage.A;  
class PackageTest2{  
    public static void main(String args[]){  
        new A().print();  
    }  
}
```

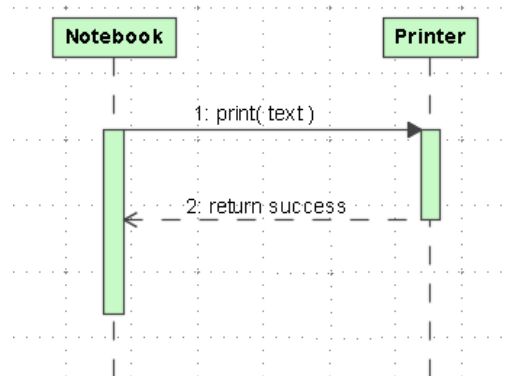
Method & Message

❖ Objects interact (communicate) by passing messages



Method & Message

- ❖ Request Message : ส่งผ่าน Parameters ของ method
- ❖ Response Message : ค่า return ของ Method



Defining Methods

- ❖ With a return value (type)

```
String print( String text )
{
    System.out.println( text );
    return "success"; //return is required
}
```

A green arrow points to the **return** statement in the code block above.

- ❖ Without a return value (type)

- void return type

```
String print( String text )
{
    System.out.println( text );
    return; // return is not required
}
```

Method Overloading



- ❖ **Method** ชื่อเดียวกัน แต่ละ **Parameter** แตกต่างกัน
- ❖ ใช้ในการเพิ่มความสะดวกในการใช้งาน **Method** ของ **Class**

Method Overloading



```
public class Printer {  
  
    public String print( String text ){  
        System.out.println( text );  
        return "success";  
    }  
  
    public String print( int number ){  
        System.out.println( number );  
        return "success";  
    }  
}
```

Constructor

❖ ใช้ในการสร้าง **Instance** ใหม่ของ **Class**
ผ่านคำสั่ง **new**

- `Printer myPrinter = new Printer();`

❖ ประกาศเหมือน **method** แต่ต้องชื่อเหมือนกับ **Class**

❖ เงื่อนไข

- ไม่มี **return type**
- ไม่สามารถประกาศเป็น **static** , **final**, **abstract**
- สามารถทำ **Overload** ได้เหมือน **Method**



Constructor Example

```
public class Printer {  
  
    String brand;  
    String colorMode;  
  
    public Printer() {  
        super();  
    }  
  
    public Printer(String brand, String colorMode) {  
        super();  
        this.brand = brand;  
        this.colorMode = colorMode;  
    }  
}
```



this Reference

❖ ใช้ keyword ว่า “this”

- ถูกใช้ได้จากภายใน Class เท่านั้น
- ใช้อ้างถึง instance ที่สร้างขึ้นจาก Class

❖ มักใช้ประโยชน์ในการแยก **data member** ของ **Class** ออกจาก **parameter** ที่อาจมีชื่อซ้ำกันได้

```
public class Printer {  
    String brand;  
    String colorMode;  
  
    public Printer(String brand, String colorMode) {  
        super();  
        this.brand = brand;  
        this.colorMode = colorMode;  
    }  
}
```

Chapter 3 : Java OOP

3.3 Modifiers

Access Modifier

❖ ประกาศหน้า **data member** หรือ **method member**

เพื่อกำหนดขอบเขตการเข้าถึงโดย **Class** อื่น

- **private** String brand;
- **public** String print(int number)

	Public	protected	default	private
Same Class - Same Package	O	O	O	O
Sub Class - Same Package	O	O	O	X
Other Class - Same Package				
Sub Class - Other Package	O	O	X	X
Other Class - Other Package	O	X	X	X

Information Hiding

❖ ความเป็นส่วนตัวสามารถป้องกันวัตถุอื่นๆเข้ามาทำลายคุณสมบัติบางอย่าง

- เลขบัญชีธนาคารติดลบแทนที่จะเป็นเลขศูนย์หรือเลขบวก
- ความกว้างและความสูงของสี่เหลี่ยมน้อยกว่าหรือเท่ากับศูนย์

❖ ทำให้เราสามารถเปลี่ยนแปลงชนิดของตัวแปรเหล่านั้นได้ โดยไม่กระทบกระเทือนคลาสอื่นที่ใช้คลาสของเรา

Information Hiding



❖ สามารถเปลี่ยนแปลงวิธีการทำงานของวัตถุได้โดยไม่กระทบต่อผู้ใช้วัตถุ

❖ เกิดความยืดหยุ่น (**Loose Coupling**)

- เช่น ถ้าเปรียบวัตถุเหมือนกับรถ และผู้ใช้วัตถุคือคนขับรถ บริษัทผู้ผลิตรถสามารถเปลี่ยนเครื่องยนต์รุ่นใหม่ เปลี่ยนระบบเบรก ฯลฯ โดยที่คนขับยังสามารถขับรถได้เหมือนเดิม



Non-Access Modifier : static



❖ **static method** : ทำให้ **Method** สามารถเรียกใช้ได้จาก **Class** โดยไม่จำเป็นต้องทำการสร้าง **instance** ของ **Class**

❖ รูปแบบการประกาศ

- **public static** ชนิดข้อมูลที่ส่งกลับ **Method** (พารามิเตอร์) { ... }

❖ รูปแบบการเรียกใช้

- คลาส. **Static Method** (พารามิเตอร์)

❖ ตัวอย่างการเรียกใช้

- `int m = Math.round(12.5);`
- `int n = String.valueOf(1234);`



Non-Access Modifier : static



❖ static data member :

ทำให้ **Data Member** นั้นใช้ค่าเดียวกัน
ในทุก **instance** ของ **Class** นั้น

❖ รูปแบบ

- **private static** ชนิดข้อมูล ตัวแปร;

❖ ตัวอย่าง

- **private static** int count;



Non-Access Modifier : final



❖ final data member :

ทำให้ไม่สามารถถูกแก้ไขค่าได้อีก
สามารถนำไปใช้กับเก็บค่าคงที่ (**Constant**)

❖ รูปแบบ

- **public static final** ชนิดข้อมูลที่ส่งกลับ *Method* (พารามิเตอร์)
{ ... }

❖ ตัวอย่าง

- **public static final** String print(String text);



Non-Access Modifier : final



❖ final method :

ทำให้ **method** ไม่สามารถ **Override** ได้

❖ รูปแบบ

- `public static final` ชนิดข้อมูล ตัวแปร=ค่า;

❖ ตัวอย่าง

- `public static final int TOTAL_SIDES = 4;`



Chapter 3 : Java OOP



3.4 Inheritance



พิจารณา Class ของสัตว์ต่างๆ



❖ Class แมว

- Attribute : อายุ และ ความหิว
- Method : กิน() และ นอน()

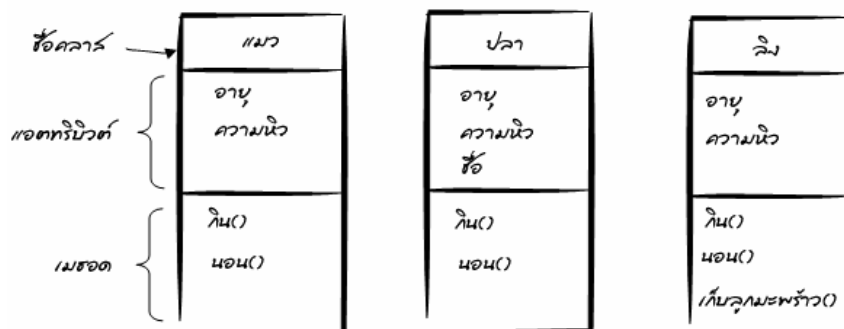
❖ Class ปลา

- Attribute : อายุ ความหิว และ ชื่อ
- Method : กิน() และ นอน()

❖ Class ลิง

- Attribute : อายุ และ ความหิว
- Method : กิน() นอน() และเก็บลูกมะพร้าว()

พิจารณา Class ของสัตว์ต่างๆ



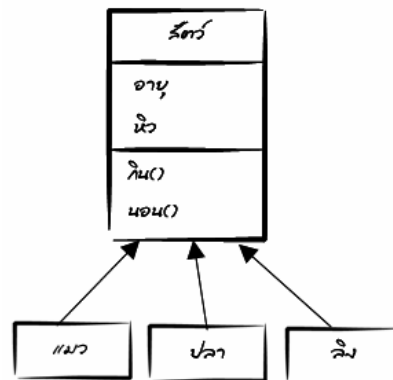
Inheritance

❖ รูปแบบ

- class คลาสลูก **extends** คลาสแม่

❖ ตัวอย่าง

```
class แมว extends สัตว์ {
}
class ปลา extends สัตว์ {
}
class ลิง extends สัตว์ {
}
```



Inheritance



Attribute และ Method ของ Class แม่

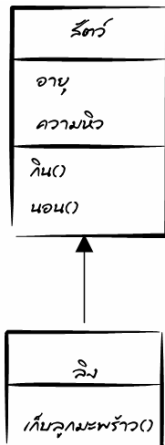
จะได้รับการถ่ายทอดสู่ Class ลูก

(เฉพาะที่มี Access Modifier เป็น Public และ Protected)

```
ลิง ล = new ลิง();
```

```
ล.กิน();
```

เพิ่ม Attribute / Method ที่ Class ลูก



```

class ปลา extends สัตว์ {
    // แอตทริบิวต์ที่เพิ่มเข้ามา
    ชื่อ;
}
    
```

```

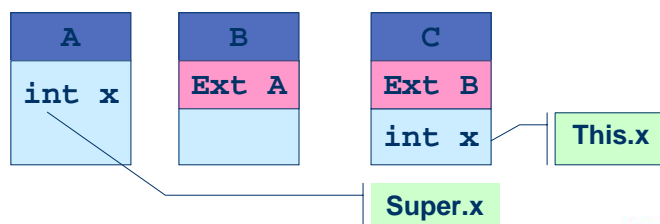
class ลิง extends สัตว์ {
    // Method ที่เพิ่มเข้ามา
    เก็บลูกมะพร้าว() { ... }
}
    
```

Super Reference

❖ ใช้ keyword ว่า “super”

แทน class ที่ inherit เพื่อใช้ในการอ้างถึง member ของ super class

❖ ในการอ้าง super จะหมายถึงตัว data member ตัวแรกที่อยู่ในสายของบรรพบุรุษ เช่น



Super Reference

```
class A {
    int a;
    void print() { System.out.println(a);}
}
class B extends A {
    int a;
    B(int x, int y){super.a = x; this.a = y;}
    void print() {
        super.print(); System.out.println(a);}
}
class Super1{
    public static void main(String args[]){
        B b = new B(1,2);
        b.print();
    }
}
```

1
2



Chapter 3 : Java OOP

3.5 Polymorphism

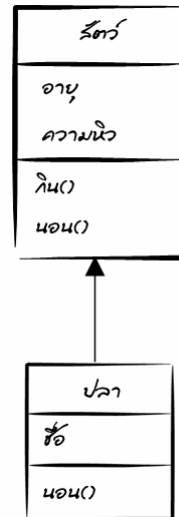


Method Overriding

Class ลูกสามารถทำการประกาศ method ทับ method ที่สืบทอดมาจาก Class แม่ได้ เมื่อต้องการพฤติกรรมการทำงานที่แตกต่างไปจาก Class แม่

```
class ปลา extends สัตว์ {
    // แอตทริบิวต์
    ชื่อ;

    // Method
    นอน() {
        // วิธีการนอนของปลา
        ...
    }
}
```



MFEC
Public Company Limited

Polymorphism

สัตว์ ส;

ส = new สัตว์();

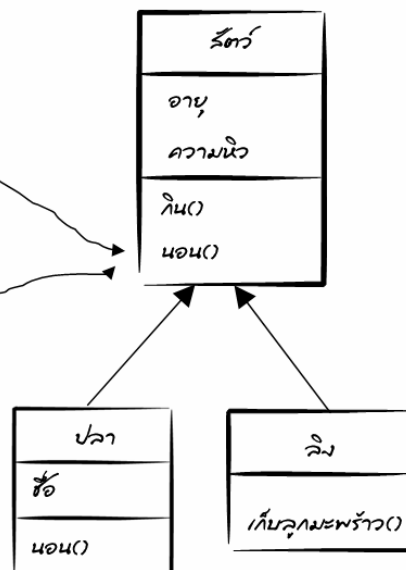
ส.นอน();

ส = new ลิง();

ส.นอน();

ส = new ปลา();

ส.นอน();



MFEC
Public Company Limited

Polymorphism



- ❖ **poly** แปลว่า หลายหรือมาก
- ❖ **morphism** นั้นมาจากคำว่า **morph** ซึ่งแปลว่ารูปร่าง
- ❖ รวมกันแล้ว หมายถึง ความสามารถที่สิ่งหนึ่งจะมีได้หลายรูปร่าง
- ❖ ซึ่งเมื่อใช้คำนี้กับการโปรแกรมเชิงวัตถุ ก็จะหมายถึงการที่คำสั่งแบบเดียวกันสามารถถูกเปลี่ยนได้หลายแบบ

- ❖ จะเห็นได้ว่า **Class** สัตว์เหมือนกัน เมื่อเรียก **Method** นอน() แล้วอาจมีการทำงานแตกต่างกันออกไป ขึ้นกับว่าสัตว์นั้นเป็นสัตว์อะไร
- ❖ ดังนั้น กลไกในการผูกชื่อ **Method** กับการทำงานของ **Method** จึงต้องทำตอน **Runtime** เท่านั้น
 - **Dynamic Binding**



Abstract Class



- ❖ **Class** ที่เป็นนามธรรม ไม่สามารถทำการ **instantiate** ได้
 - ใส่ keyword “abstract” ที่หน้า class

```
abstract class A {  
    abstract public void f();  
    public void g() { System.out.println("g"); }  
}
```



Abstract Method

- ❖ **Abstract Class** สามารถประกาศ **abstract method** ได้
 - ใส่ keyword “abstract” ที่หน้า method
 - เป็น method ที่ไม่มี implementation (รอ class ลูกมา override)
- ❖ Class ลูกที่ **extends abstract class** จะต้องทำการ **implement abstract method** ให้ครบ จึงจะสามารถเป็น **concrete class** ที่สามารถทำการ **instantiate** ได้

```
abstract class A {  
    abstract public void f();  
    public void g() { System.out.println("g"); }  
}  
class B extends A {  
    public void f() { System.out.println("f"); }  
}
```



Interface

- ❖ **Abstract class** ที่มี member เป็น
 - ค่าคงที่ (static final) หรือ
 - abstract method เท่านั้น
- ❖ ใช้ keyword “interface” แทน Class

```
public interface USBInterface {  
    static final String code = "USB";  
    abstract void execute();  
}
```



Using Interface

- ❖ สมมติว่ามีบริษัทคิด ผลิต **Notebook** ที่มี **Printer** ติดมาด้วย
- ❖ อาจเกิดปัญหาว่าลูกค้าไม่พอใจ **Printer** ยี่ห้อที่เรา **bundle** ไปให้
- ❖ ??? แก้ปัญหายังไง

```
public class NotebookWithPrinter {  
    String color = "Black";  
    String brand = "HP";  
  
    Printer printer;  
    ...  
}
```



Implements Interface

- ❖ ลองคิดเทียบกับในโลกความเป็นจริง
 - Notebook มี USB Interface
 - ก็เอา USB Printer ที่ไหนมาเสียบใช้งานได้

```
public class PrinterUSBImpl  
    implements USBInterface {  
    public String execute(String input) {  
        return print(input);  
    }  
}
```



Implements Interface

```
public class NotebookWithUSB {  
    USBInterface usb;  
  
    public void setUsb(USBInterface usb) {  
        this.usb = usb;  
    }  
  
    void execute( String text ){  
        usb.execute(text);  
    }  
    ....  
}
```



Java Naming Convention

❖ ชื่อ Class

- ตั้งเป็นคำนาม
- Camel Case ขึ้นต้นด้วยอักษรตัวใหญ่
- Example : Student

❖ ชื่อ Method

- เป็นคำกริยา
- Camel Case ขึ้นต้นด้วยอักษรตัวเล็ก
- Example : getScore()



Java Naming Convention



❖ ชื่อ Data Member

- เป็นคำนาม
- Camel Case เริ่มต้นด้วยอักษรตัวเล็ก
- Example : `int examScore;`

❖ ชื่อ Package

- ใช้ตัวอักษรตัวเล็กทั้งหมด
- Example : `training.java.lab1`

❖ ชื่อ Class ที่ implement Interface

- ตั้งชื่อลงท้ายด้วย `Impl`
- Example : `CommonDAOImpl`



Chapter 4 Exception Handling



By Mongkol Puengpipattrakul
MFEC Public Co., Ltd.



Error

❖ Syntax Error

- เขียนขึ้นผิดจากข้อกำหนดในภาษานั้น
- สามารถตรวจพบได้ง่าย ตั้งแต่ตอนคอมไพล์โปรแกรม

❖ Logical Error

- เกิดจากความเข้าใจผิดของผู้เขียนโปรแกรมเอง
- ความผิดพลาดลักษณะนี้จะตรวจพบได้ยาก เนื่องจากตัวภาษาจะไม่แจ้งความผิดพลาดนั้นออกมา



Syntax Error

```
public class SyntaxError {  
    public static void main(String[] args) {  
        int i;  
        System.out.println(i);  
    }  
}
```

Error @ Compile Time

```
C:\> C:\WINDOWS\system32\cmd.exe  
C:\Java\07> javac SyntaxError.java  
SyntaxError.java:5: Variable i may not have been initialized.  
    System.out.println(i);  
                        ^  
1 error  
C:\Java\07> _
```

Logical Error

```
public class LogicalError {  
    public static void main(String[] args) {  
        int i = 2000000000;  
  
        System.out.println(i*2);  
    }  
}
```

-294967296



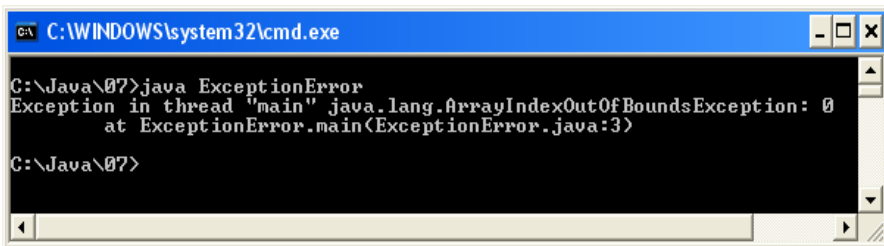
ปัจจัยที่ไม่สามารถควบคุมได้

- ❖ หน่วยความจำที่มีจำกัด
- ❖ อุปกรณ์การรับและส่งข้อมูลที่อาจขัดข้อง
- ❖ การกรอกข้อมูลที่ผิดพลาด



ตัวอย่างการเกิด Exception

```
public class ExceptionError {  
    public static void main(String[] args) {  
        int[] array = new int[3];  
  
        System.out.println(array[3]);  
    }  
}
```



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The command entered is "C:\Java\07>java ExceptionError". The output shows an exception: "Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0 at ExceptionError.main(ExceptionError.java:3)". The prompt then shows "C:\Java\07>".

Exception Handling in C

❖ ตรวจสอบค่าที่ส่งคืน

```
FILE *fp;  
fp = fopen("mypic.jpg", "r");  
  
if (fp == NULL) {  
    printf("Cannot Open Image\n");  
    exit(0);  
}
```

ข้อเสีย

❖ อาจละเลยการตรวจสอบได้

- ภาษา C ไม่ได้บังคับว่าจะต้องมีการตรวจสอบความผิดปกติเช่นนี้ทุกครั้ง
- โปรแกรมเมอร์ต้องตรวจสอบเอง

❖ คำสั่งหลักอยู่บนกับคำสั่งจัดการความผิดปกติ

- ใช้ if เพื่อดักจับความผิดปกติทุกครั้ง

❖ ไม่สามารถระบุสาเหตุของความผิดปกติ

- ค่าที่คืนอาจเป็น NULL หรือตัวเลขติดลบ
- ไม่เพียงพอที่จะอธิบายถึงสาเหตุหลักของความผิดปกติได้



Exception Handling in VB

❖ มีส่วนที่ใช้จัดการความผิดปกติ

On Error GoTo LoadPicError

MyImage.Picture = LoadPicture("mypic.jpg")

Exit Sub

LoadPicError:

MsgBox(Err.Description)



ข้อเสีย



❖ ตรวจสอบลำดับการทำงานตรวจสอบลำดับการทำงานได้ยาก

- การใช้คำสั่ง **GoTo** เพื่อให้โปรแกรมข้ามการทำงานไปยังบริเวณที่จัดการความผิดปกติ ทำให้โครงสร้างของโปรแกรมไม่เป็นระเบียบ มีทางออกจากฟังก์ชันได้หลายทาง

❖ จัดการความผิดปกติได้ยาก

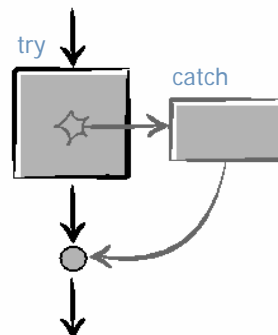
- การจัดการความผิดปกติจะต้องระบุอยู่เฉพาะในฟังก์ชันนั้นๆ เท่านั้น ไม่สามารถส่งให้ผู้เรียกใช้ฟังก์ชันนั้นตัดสินใจได้เองว่าจะจัดการกับความผิดปกติอย่างไร

Exception Handling in Java



❖ รูปแบบ

```
try {  
    ประโยคทำงานที่อาจเกิดความผิดปกติ;  
}  
catch (วัตถุException) {  
    ประโยคที่ใช้จัดการความผิดปกติ;  
}
```



No Exception Handling

```
import java.util.Scanner;

public class NoTryCatch {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String str = sc.next();
        int i = Integer.parseInt(str);
        System.out.println(i);
    }
}
```



ผลการทำงาน

❖ เมื่อป้อนค่า 10

10

❖ เมื่อป้อนค่า 10.5

```
Exception in thread "main" java.lang.NumberFormatException:
    For input string: "10.5"
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
    at java.lang.Integer.parseInt(Integer.java:456)
    at java.lang.Integer.parseInt(Integer.java:497)
    at NoTryCatch.main(NoTryCatch.java:8)
```



Handle with try-catch

```
Scanner sc = new Scanner(System.in);
String str = sc.next();
int i = 0;
try {
    i = Integer.parseInt(str);
} catch (Exception e) {
    System.out.println("Catch exception");
}
System.out.println(i);
```



ผลการทำงาน

❖ เมื่อป้อนค่า 10

10

❖ เมื่อป้อนค่า 10.5

Catch exception

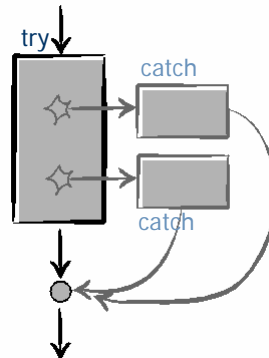
0



Handle by Exception Type

❖ รูปแบบ

```
try {  
    ประโยคทำงานที่อาจเกิดความผิดปกติ;  
}  
catch (วัตถุ Exception ชนิดที่ 1) {  
    ประโยคที่ใช้จัดการความผิดปกติชนิดที่ 1;  
}  
catch (วัตถุ Exception ชนิดที่ 2) {  
    ประโยคที่ใช้จัดการความผิดปกติชนิดที่ 2;  
}
```



Handle by Exception Type

```
try {  
    i = Integer.parseInt(str);  
} catch (NumberFormatException e) {  
    System.out.println("Catch number format exception");  
} catch (Exception e) {  
    System.out.println("Catch exception");  
}  
System.out.println(i);
```

ผลการทำงาน

❖ เมื่อป้อนค่า 10

10

❖ เมื่อป้อนค่า 10.5

Catch number format exception

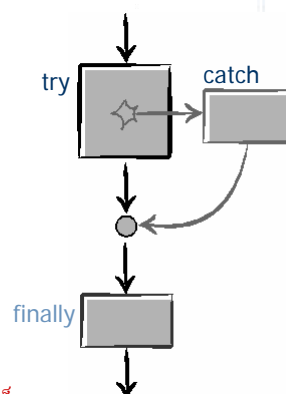
0



finally

❖ รูปแบบ

```
try {  
    ประโยคทำงานที่อาจเกิดความผิดปกติ;  
}  
catch (วัตถุ Exception1) {  
    ประโยคที่ใช้จัดการความผิดปกติ;  
}  
catch (วัตถุ Exception2) {  
    ประโยคที่ใช้จัดการความผิดปกติ;  
}  
finally {  
    ประโยคที่ทำท้ายสุดในทุกกรณี;  
    (ไม่ว่าจะเกิด Exception ใดๆ หรือ ไม่เกิด Exception ก็ตาม )  
}
```



Using finally

```
try {
    i = Integer.parseInt(str);
}
catch (NumberFormatException e) {
    System.out.println("Catch number format exception");
}
catch (Exception e) {
    System.out.println("Catch exception");
}
finally{
    System.out.println(i);
}
```



Exception Type

❖ Uncheck Exception

- Inherit จาก Runtime Exception
- Unchecked exceptions represent error conditions that are considered “fatal” to program execution.
- You do not have to do anything with an unchecked exception. Your program will terminate with an appropriate error message.
- เช่น
 - NumberFormatException
 - IndexOutOfBoundsException
 - NullPointerException
 - IllegalArgumentException



Exception Type

❖ Checked Exception

- Checked exceptions are inherited from the core Java class Exception. They represent exceptions that are frequently considered “non fatal” to program execution
- Checked exceptions must be handled in your code, or passed to parent classes for handling.
- เช่น
 - IOException
 - FileNotFoundException



Handle Checked Exception

ดักด้วย **try-catch**

```
public void parseNumber( String str ){  
  
    try {  
        i = Integer.parseInt(str);  
    } catch (NumberFormatException e) {  
        System.out.println("Catch number format exception");  
    }  
  
}
```



Handle Checked Exception



ส่ง Exception ขึ้นไปให้ Method Caller

```
public void parseNumber( String str )  
    throws NumberFormatException{  
  
    i = Integer.parseInt(str);  
  
}
```



Chapter 5 Using Java Classes



By Mongkol Puengpipattrakul
MFEC Public Co., Ltd.





5.1 String

Method ที่น่าสนใจ



- ❖ **length()** ความยาวของ **String**
- ❖ **charAt()** ตัวอักษรในตำแหน่งที่กำหนด
 - ตัวอักษรตัวแรกคือตำแหน่งที่ 0
 - ตัวอักษรสุดท้ายคือ **length() - 1**
- ❖ **indexOf()** ตำแหน่งของสายอักขระใน **String**
- ❖ **substring()** **String** ที่อยู่ในช่วงที่กำหนด

ตัวอย่างการใช้ Method



```
public class StringMessage {  
    public static void main(String[] args) {  
        String name = "Smith";  
  
        System.out.println(name.length());  
        System.out.println(name.charAt(2));  
        System.out.println(name.indexOf("t"));  
        System.out.println(name.substring(1, 3));  
    }  
}
```



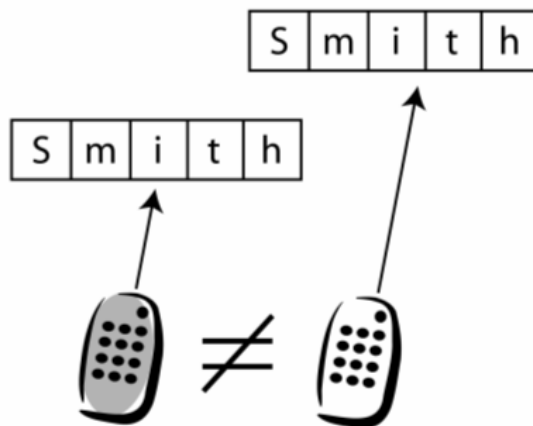
การเปรียบเทียบ String (แบบผิดๆ)



```
public class StringCompareInCorrect {  
    public static void main(String[] args)  
    {  
        String name1 = new String("Smith");  
        String name2 = new String("Smith");  
        System.out.println(name1 == name2);  
    }  
}
```



การเปรียบเทียบ String (แบบผิดๆ)



การเปรียบเทียบ String



```
public class StringCompare {  
    public static void main(String[] args)  
    {  
        String name1 = new String("Smith");  
        String name2 = new String("Smith");  
  
        System.out.println(name1.equals(name2));  
    }  
}
```

การต่อ String

❖ Method concat()

❖ เครื่องหมาย +

❖ เครื่องหมาย +=

Method concat()

```
public class StringConcat1 {  
    public static void main(String[] args) {  
        String name = "Smith";  
        String lastName = " Brown";  
        String fullName = name.concat(lastname);  
        System.out.println(fullName);  
    }  
}
```

เครื่องหมาย +



```
public class StringConcat2 {  
    public static void main(String[] args) {  
        String name = "Smith";  
        String lastName = " Brown";  
        String fullName = name + lastName;  
        System.out.println(fullName);  
    }  
}
```



เครื่องหมาย +=



```
public class StringAppend {  
    public static void main(String[] args) {  
        String name = "Smith";  
        String lastName = " Brown";  
        name += lastName;  
        System.out.println(name);  
    }  
}
```



Immutability

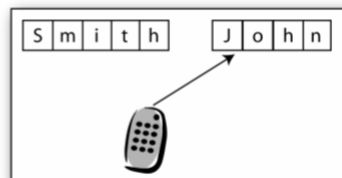
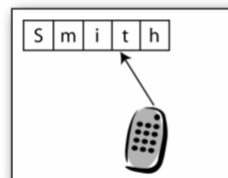


- ❖ไม่สามารถเปลี่ยนแปลงค่าของ **String** ได้เลยหลังจากที่เราได้สร้างมันขึ้นมา
- ❖ไม่มี **method** ใดเลยที่ทำให้ **String** เปลี่ยนแปลงได้
 - length()
 - substring()

String is immutable

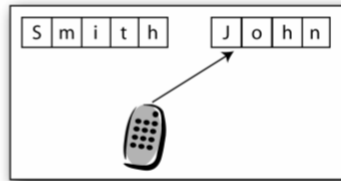
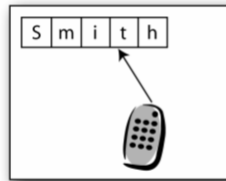


```
String str;  
str = "Smith";  
str = "John";
```



String is immutable

```
String str;  
str = new String("Smith");  
str = new String("John");
```

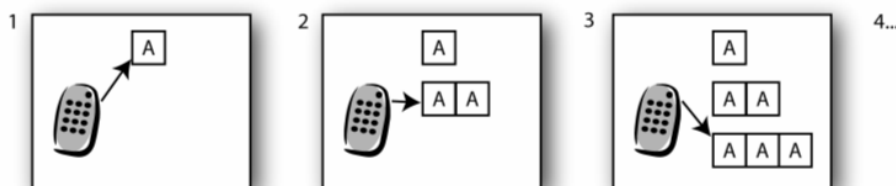


String is immutable

```
String str = "Smith";  
System.out.println(str.toUpperCase());  
System.out.println(str);
```

String ไม่เปลี่ยนแปลง

```
public class StringDeficiency {  
    public static void main(String[] args)  
    {  
        String str = "A";  
        for (int i = 0; i < 100; i++)  
            str += "A";  
        System.out.println(str);  
    }  
}
```



StringBuffer

- ❖ ข้อดีของ **StringBuffer**
- ❖ ประหยัดทั้งหน่วยความจำและเวลาในการประมวลผล
 - วัตถุ **StringBuffer** สามารถเปลี่ยนแปลงค่าในสายอักขระที่มันนำเสนอได้เอง โดยไม่ต้องสร้างวัตถุขึ้นมาใหม่

Method ที่น่าสนใจ

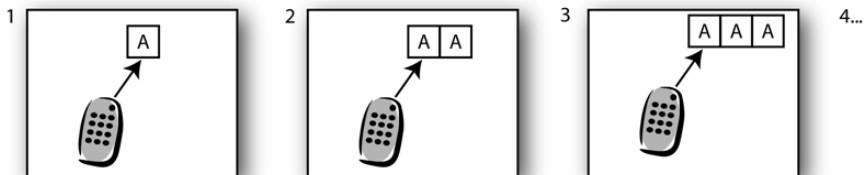
- ❖ **append()** การต่อสายอักขระ
- ❖ **insert()** การแทรกสายอักขระ
- ❖ **delete()** การลบสายอักขระย่อย

ตัวอย่างการใช้ Method

```
StringBuffer sb = new StringBuffer("John");  
System.out.println(sb);  
  
sb.append(" Hunter");  
System.out.println(sb);  
sb.insert(4, "y");  
System.out.println(sb);  
  
sb.delete(2, 4);  
System.out.println(sb);
```

ทดลองเปลี่ยนแปลง StringBuffer

```
public class StringBufferAppend {  
    public static void main(String[] args) {  
        StringBuffer sb;  
        sb = new StringBuffer("A");  
  
        for (int i = 0; i < 100; i++)  
            sb.append("A");  
  
        System.out.println(sb);  
    }  
}
```



StringBuilder

- ❖ มี Method ที่เหมือนกับ StringBuffer
- ❖ ทำงานได้เร็วกว่า StringBuffer
- ❖ StringBuffer ปลอดภัยสำหรับการทำงานในแบบ multiple threads



5.2 Wrapper Class

Wrapper Class



- ❖ ห่อชนิดข้อมูลพื้นฐาน
 - เพื่อใส่ใน Collection
- ❖ แปลง **String** เป็นชนิดข้อมูลพื้นฐาน
- ❖ แปลงจากชนิดข้อมูลพื้นฐานเป็น **String**
- ❖ มีค่าคงที่ของค่าที่มากที่สุดและน้อยของชนิดข้อมูลพื้นฐาน

Wrapper Class

ชนิดข้อมูลพื้นฐาน	คลาสในกลุ่ม Wrapper
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

การสร้างวัตถุ Wrapper

```
Boolean    bo = new Boolean(true);  
Boolean    bo = new Boolean("true");  
  
Character   c  = new Character('c');  
  
Byte        by = new Byte((byte)20);  
Byte        by = new Byte("20");  
  
Short       s  = new Short((short)20);  
Short       s  = new Short("20");
```

การแปลง **Wrapper** กลับเป็นชนิด ข้อมูลพื้นฐาน



❖ รูปแบบ

- ตัวแปรชนิดข้อมูลพื้นฐาน = `Wrapper.ชนิดข้อมูลพื้นฐานValue()`;

❖ ตัวอย่าง

```
String s = "20";  
Integer i = new Integer(s);  
int number = i.intValue();
```



ตัวอย่าง



```
Scanner sc = new Scanner(System.in);  
  
System.out.print("Please enter a number : ");  
String s = sc.next();  
  
Integer i = new Integer(s);  
int number = i.intValue();  
  
System.out.print("Your number plus 10 equals ");  
System.out.println(number + 10);
```



การแปลง **Wrapper** เป็น **String**



❖ รูปแบบ

- *Ref. String* = *Wrapper Class.toString()*;

❖ ตัวอย่าง

```
int number = 20;  
Integer i = new Integer(number);  
String s = i.toString();
```

การแปลง **String** เป็นชนิดข้อมูลพื้นฐาน



❖ รูปแบบ

- *ตัวแปรชนิดข้อมูลพื้นฐาน* = *คลาสห่อหุ้ม.parseInt()*;

❖ ตัวอย่าง

```
Scanner sc = new Scanner(System.in);  
System.out.print("Please enter a number : ");  
int number = Integer.parseInt(sc.next());
```

การแปลงชนิดข้อมูลพื้นฐาน เป็น String



❖ รูปแบบ

- *Reference String* = คลาสห่อหุ้ม.toString(ข้อมูลพื้นฐาน);

❖ ตัวอย่าง

```
Scanner sc = new Scanner(System.in);  
System.out.print("Please enter a number : ");  
String s = Integer.toString(sc.nextInt());
```



ค่าคงที่ในคลาส Wrapper



❖ Integer

- Integer.MIN_VALUE = -2147483648
- Integer.MAX_VALUE = 2147483647

❖ Double

- Double.MIN_VALUE = 4.9E-324
- Double.MAX_VALUE = 1.7976931348623157E308





5.3 collections and maps

Topics



- ❖ 5.3.1 Array
- ❖ 5.3.2 ArrayList
- ❖ 5.3.3 Collection
- ❖ 5.3.4 List
- ❖ 5.3.5 Set
- ❖ 5.3.6 Map



5.3.1 Array

Array



- ❖ ตัวแปรที่เก็บข้อมูลได้หลายค่า
- ❖ เก็บข้อมูลชนิดพื้นฐานประเภทเดียวกัน หรือ
- ❖ เก็บ **Reference** ของวัตถุในลำดับชั้นการสืบทอด
 - เช่น Shape[] สามารถเก็บ Shape หรือ Rectangle ได้ แต่ไม่สามารถเก็บ Object หรือ String ได้
- ❖ เป็นวัตถุ

ข้อจำกัดของ **Array**



- ❖ ไม่สามารถเปลี่ยนแปลงขนาดของ **Array** หลังจากที่ถูกสร้างขึ้นได้
- ❖ ต้องกะประมาณขนาดของที่เราต้องการใช้ไว้ล่วงหน้า
 - อาจจะจองพื้นที่เผื่อไว้เยอะๆ แต่ก็ไม่มีประสิทธิภาพ

แก้ไขโดย



- ❖ สร้างคลาสเพื่อเก็บข้อมูลได้หลายๆค่า
- ❖ ไม่ต้องกำหนดขนาดไว้ล่วงหน้า

คลาส **ArrayList**

- ❖ หนึ่งในกลุ่มของ **Collection**
- ❖ แก้ไขข้อจำกัดของ **Array**
- ❖ เก็บได้เฉพาะ **Reference**
- ❖ อยู่ในแพ็คเกจ **java.util**

เปรียบเทียบการใช้งาน

```
String[] array;  
array = new String[3];
```

```
ArrayList<String> arrayList;  
arrayList = new ArrayList<String>();  
หรือ  
arrayList = new ArrayList();
```

การใส่ค่า



```
String[] array;  
array = new String[3];  
  
array[0] = "Somchai";  
array[1] = "Somying";  
array[2] = "Somporn";  
  
for (String s : array)  
    System.out.println(s);
```

```
ArrayList<String> arrayList;  
arrayList = new ArrayList<String>();  
  
arrayList.add("Somchai");  
arrayList.add("Somying");  
arrayList.add("Somporn");  
  
for (String s : arrayList)  
    System.out.println(s);
```



เพิ่มข้อมูลเข้าไปอีกหนึ่ง



```
String[] array;  
array = new String[3];  
  
array[0] = "Somchai";  
array[1] = "Somying";  
array[2] = "Somporn";  
  
array[3] = "Somwang";  
  
for (String s : array)  
    System.out.println(s);
```

```
ArrayList<String> arrayList;  
arrayList = new ArrayList<String>();  
  
arrayList.add("Somchai");  
arrayList.add("Somying");  
arrayList.add("Somporn");  
  
arrayList.add("Somwang");  
  
for (String s : arrayList)  
    System.out.println(s);
```



ขนาดของ **Array** และ **ArrayList**



```
String[] array;  
array = new String[3];
```

```
System.out.println(array.length);
```

```
array[0] = "Somchai";  
array[1] = "Somying";  
array[2] = "Somporn";
```

```
System.out.println(array.length);
```

```
ArrayList<String> arrayList;  
arrayList = new ArrayList<String>();
```

```
System.out.println(arrayList.size());
```

```
arrayList.add("Somchai");  
arrayList.add("Somying");  
arrayList.add("Somporn");
```

```
System.out.println(arrayList.size());
```



การนำวัตถุออกจาก **ArrayList**



```
ArrayList<String> arrayList = new ArrayList<String>();
```

```
arrayList.add("Somchai");  
arrayList.add("Somying");  
arrayList.add("Somporn");
```

```
arrayList.remove("Somying");
```

```
for (String s : arrayList) {  
    System.out.println(s);  
}
```



การใส่วัตถุต่างคลาสลงใน ArrayList

```
// สร้าง ArrayList ของวัตถุ Object
ArrayList arrayList = new ArrayList();

arrayList.add("Hello");
arrayList.add(new Integer(1));
arrayList.add(new Double(1.0));
```

ระบุให้เก็บเฉพาะวัตถุในคลาสเดียวกัน

❖ รูปแบบ

- ArrayList<คลาสที่ต้องการจะเก็บ> *Reference* ;

❖ ตัวอย่าง

- ArrayList<String> strings;

generics

- ❖ คลาสที่ถูกประกาศโดยมีพารามิเตอร์ที่เป็นคลาส
 - `ArrayList<String> strings;`
- ❖ **Method** ต่างๆที่เกี่ยวข้องกับพารามิเตอร์นี้ก็จะถูกกำหนดให้รับหรือส่งข้อมูลที่เป็นของคลาสนั้นเท่านั้น
 - เช่น Method `add()` ก็สามารถรับเฉพาะข้อมูลที่เป็น `String` เท่านั้น

Chapter 5 : Using Java Classes

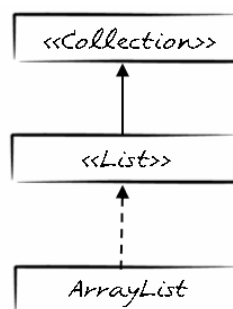
5.3.2 Collection

อินเทอร์เฟซ Collection

❖ วัตถุที่สามารถใช้เก็บรวบรวมวัตถุต่างๆ

❖ Collection ใน Java เป็นอินเทอร์เฟซ
ที่ชื่อ Collection

- Method add(), remove() และ size()
 - `Collection<String> collection = new ArrayList<String>();`



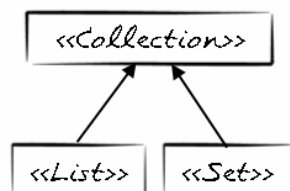
ประเภทของ Collection

❖ List (List)

- เก็บวัตถุเรียงกันเป็นรายการและมีลำดับ

❖ Set (Set)

- ไม่เก็บวัตถุที่ซ้ำกัน



การใช้งาน List

```
Collection<String> list;  
list = new ArrayList<String>();  
list.add("Somchai");  
list.add("Somying");  
list.add("Somchai");  
System.out.println(list);
```

[Somchai, Somying, Somchai]

การใช้งาน Set

```
Collection<String> set;  
set = new HashSet<String>();  
set.add("Somchai");  
set.add("Somying");  
set.add("Somchai");  
System.out.println(set);
```

[Somying, Somchai]

การใช้ **Collection** เก็บชนิดข้อมูลพื้นฐาน



- ❖ **Collection** สามารถใช้เก็บ **Reference** ได้เพียงอย่างเดียว
- ❖ นำชนิดข้อมูลพื้นฐานมาห่อด้วยวัตถุ **wrapper** ก่อน แล้วจึงนำไปใส่ใน **Collection**
 - เช่น นำ **int** มาห่อด้วยวัตถุ **Integer**
- ❖ แต่ **Java 5.0** ห่อให้โดยอัตโนมัติ
 - เรียกว่า **autoboxing**
 - `collection.add(10);`



ทดสอบ **autoboxing and unboxing**



```
Collection<Integer> col = new ArrayList<Integer>();  
col.add(10);  
col.add(20);  
col.add(30);  
  
for (int n : col)  
    System.out.println(n);
```



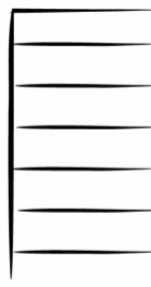


5.3.3 List

อินเทอร์เฟซ List



- ❖ อินเทอร์เฟซ **List** สืบทอดจาก
อินเทอร์เฟซ **Collection**
- ❖ มีลักษณะเฉพาะคือมีการเรียงลำดับของ
วัตถุใน **List Method** ในอินเทอร์เฟซนี้จึงมีพารามิเตอร์เกี่ยวกับลำดับหรือ
index
 - void add(int **index**, E element)
 - E remove(int **index**)
 - E set(int **index**, E element)
 - E get(int **index**)



คลาสที่อิมพลิเมนต์อินเตอร์เฟซ List



❖ คลาส ArrayList

- List ที่ถูกสร้างขึ้นด้วย Array
- อ้างถึงวัตถุใน List ทำได้อย่างรวดเร็ว
- แต่การแทรกและการเอาวัตถุออกจาก List ทำได้ช้า

❖ คลาส LinkedList

- List ที่ถูกสร้างขึ้นด้วยโครงสร้างข้อมูลแบบ linked list
- การลบและการแทรกวัตถุทำได้อย่างรวดเร็ว
- การเข้าถึงข้อมูลแบบ random access ทำได้ช้ากว่า

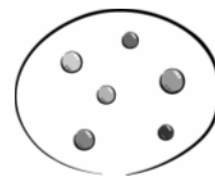
Chapter 5 : Using Java Classes



5.3.4 Set

อินเทอร์เน็ตเซต Set

- ❖ สืบทอดจากอินเทอร์เน็ตเซต Collection
- ❖ ไม่มีวัตถุที่เหมือนกันอยู่ใน Set



คลาสที่อิมพลีเมนต์อินเทอร์เน็ตเซต Set

- ❖ HashSet
 - Set ที่ถูกสร้างขึ้นจากตารางแฮช (hash table)
 - การเปรียบเทียบความเหมือนกันของวัตถุที่จะเพิ่มเข้ามาใน Set ทำได้อย่างรวดเร็ว
 - ไม่มีการเรียงตามลำดับที่ถูกเพิ่มเข้ามา
- ❖ LinkedHashSet
 - Set ที่ถูกสร้างขึ้นจากตารางแฮชและโครงสร้างข้อมูลแบบ linked-list
 - สมาชิกของ Set นี้จะเรียงตามลำดับที่ถูกเพิ่มเข้ามา
- ❖ TreeSet
 - Set ที่ถูกสร้างขึ้นจากโครงสร้างข้อมูลที่เรียกว่าต้นไม้ Red-Black
 - ข้อมูลใน Set ชนิดนี้จะถูกเรียงตามลำดับ

ทดสอบการทำงานของ HashSet



```
Set<String> set1 = new HashSet<String>();  
set1.add("Sompong");  
set1.add("Somying");  
set1.add("Somchai");  
System.out.println(set1);
```

[Somying, Somchai, Sompong]



ทดสอบการทำงานของ LinkedHashSet



```
Set<String> set2 = new LinkedHashSet<String>();  
set2.add("Sompong");  
set2.add("Somying");  
set2.add("Somchai");  
System.out.println(set2);
```

[Sompong, Somying, Somchai]



ทดสอบการทำงานของ TreeSet



```
Set<String> set3 = new TreeSet<String>();  
set3.add("Sompong");  
set3.add("Somying");  
set3.add("Somchai");  
System.out.println(set3);
```

[Somchai, Sompong, Somying]



Chapter 5 : Using Java Classes



5.3.5 Map



Map



- ❖ **Array** แบบหนึ่ง
- ❖ การอ้างถึงค่าใน **Array** ประเภทนี้จะไม่ใช่ดัชนีที่เป็นตัวเลข
- ❖ แต่จะใช้วัตถุเป็นอินเด็กซ์ (หรือในที่นี้เรียกว่าคีย์ - **key**) เพื่อดึงเอาค่าออกมา (เรียกว่า **value**)

Mapของที่อยู่



คีย์ (key)	ค่า (value)
สมชาย	11 ถนนเพชรเกษม อำเภอหาดใหญ่ จังหวัดสงขลา
สมหญิง	4 ถนนพริเว็ท ลิตเติ้ลวิงกิ้ง เซอร์เรย์
สมพร	11 ถนนเพชรเกษม อำเภอหาดใหญ่ จังหวัดสงขลา

เปรียบเทียบระหว่าง Array กับ Map



Array

ที่อยู่[0] = "0 ถนนหนทาง";
ที่อยู่[1] = "4 ถนนพริเวณ";
ที่อยู่[2] = "0 ถนนหนทาง";

Map

ที่อยู่[สมชาย] = "0 ถนนหนทาง";
ที่อยู่[สมหญิง] = "4 ถนนพริเวณ";
ที่อยู่[สมพร] = "0 ถนนหนทาง";

อินเตอร์เฟซ Map



- ❖ Method **put(คีย์, ค่า)** ใช้สำหรับใส่ค่าลงใน Map
- ❖ Method **get(คีย์)** ใช้สำหรับเรียกดูค่าใน Map

การใช้งาน Map



```
Map address = new HashMap();
```

```
address.put("Somchai", "0 Hontang Rd.");  
address.put("Somying", "4 Privet Drive");  
address.put("Somporn", "0 Hontang Rd.");
```

```
System.out.println(address);
```

```
System.out.println(address.get("Somporn"));
```

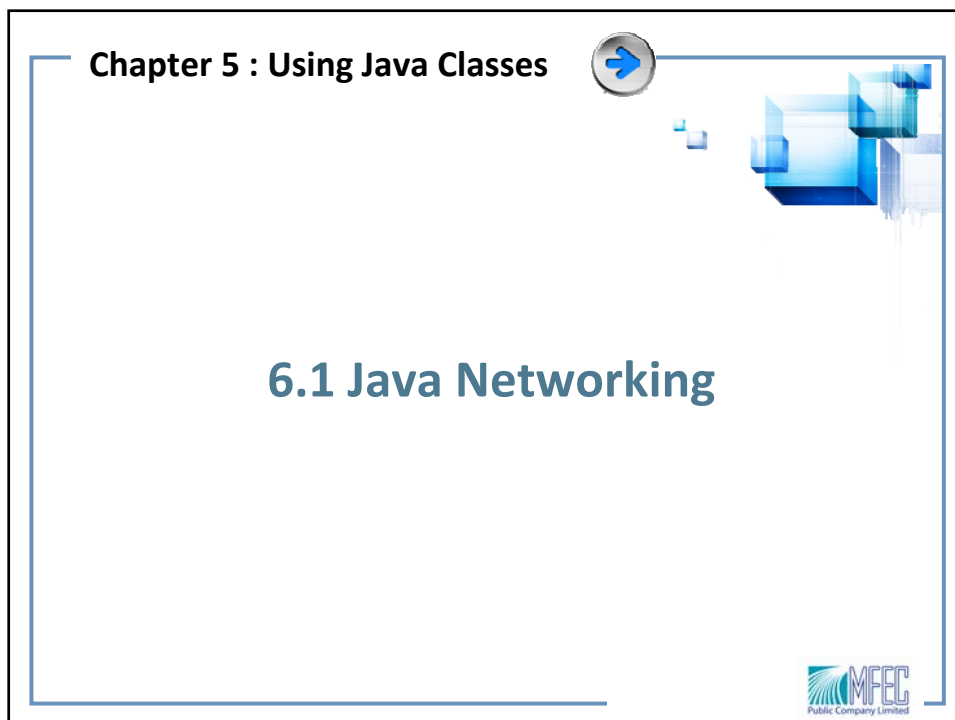
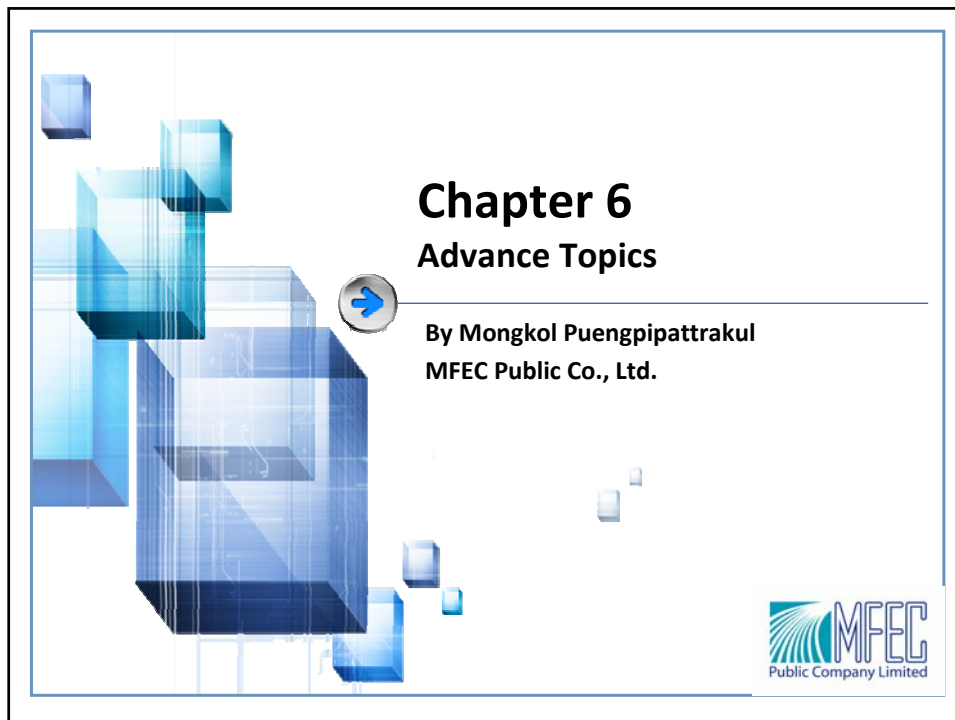


Method อื่นๆของ Map



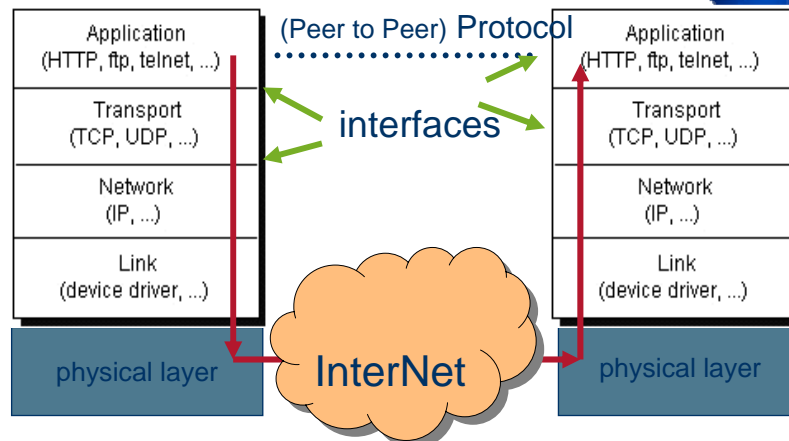
- ❖ int size()
- ❖ boolean containsKey(Object key)
- ❖ boolean containsValue(Object value)
- ❖ Set<K> keySet()
- ❖ Collection<V> values()





Networking Basics

❖ TCP/IP Network Protocol



TCP and UDP

- ❖ TCP (Transmission Control Protocol) is a **connection-based** protocol that provides a **reliable** flow of data between two computers.
- ❖ UDP (User Datagram Protocol) is a protocol that sends independent packets of data, called **datagrams**, from one computer to another **with no guarantees about arrival**. UDP is not connection-based like TCP.

Class InetAddress



- ❖ This class represents an Internet Protocol (IP) address.
- ❖ "localhost": the "local loopback" IP address for testing without a network
`InetAddress addr = InetAddress.getByName(null);`
- ❖ Equivalently:
`InetAddress.getByName("localhost");`
- ❖ Or using the reserved IP number for the loopback:
`InetAddress.getByName("127.0.0.1");`



Class InetAddress



```
// Finds out your network address
import java.net.*;
public class WhoAmI {
    public static void main(String[] args) throws Exception {
        if(args.length != 1) {
            System.err.println("Usage: WhoAmI MachineName");
            System.exit(1);
        }
        InetAddress a = InetAddress.getByName(args[0]);
        System.out.println(a);
    }
}
```



Class URL

❖ `InputStream openStream();` // return a stream for reading

❖ EX:

```
URL yahoo = new URL("http://www.yahoo.com/");
BufferedReader in = new BufferedReader( new
    InputStreamReader( yahoo.openStream() ) );
String inputLine;
while ((inputLine = in.readLine()) != null)
    System.out.println(inputLine);
in.close();
```



Chapter 5 : Using Java Classes

6.2 Stream & Serialization



Streams and I/O



❖ **Streams** คือ วัตถุสมมติที่มีลักษณะคล้ายท่อบรรจุข้อมูล มีข้อมูลต่อกันเป็นแถวเรียงหนึ่ง เราสามารถนำ **Stream** มาใช้ต่อระหว่าง

- โปรแกรม — หน่วยความจำ — อุปกรณ์สื่อสาร — File - ...
- โดยผู้ที่เขียนหรืออ่านข้อมูลจาก **Stream** ไม่ต้องสนใจว่าปลายอีกข้างของ **Stream** ต่ออยู่กับอะไร

❖ ตัวอย่าง **Stream**

- **InputStream**
 - ByteArrayInputStream, FileInputStream, ObjectInputStream
- **OutputStream**
 - ByteArrayOutputStream, FileOutputStream, ObjectOutputStream

221

Serialization



❖ เนื่องจาก **Stream** มีลักษณะเป็นท่อเรียงหนึ่ง

❖ ดังนั้น **Object** ที่จะส่งผ่านไปกับ **Stream** นั้น จะต้องทำการบีบให้เป็นเส้น (**Serialization**) ก่อน

❖ โดย **Object** จะสามารถทำการ **Serialize** ได้ เมื่อ **Class** ของ **Object** นั้น **implement Serializable Interface**

```
import java.io.*;
class A implements Serializable {
    int x = 1;
}
```

222

Serialization

```
import java.io.*;
class Serial {
    public static void main(String[] args) throws Exception {
        FileOutputStream fos = new FileOutputStream("c:/tmp");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(new A());
        oos.close();
        fos.close();
    }
}
```

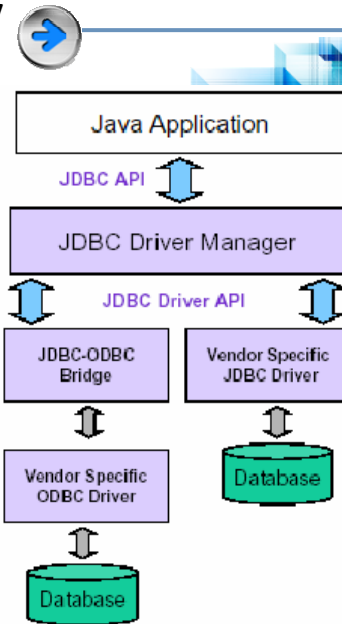
223

Chapter 5 : Using Java Classes

6.3 JDBC

Java Database Connectivity (JDBC)

- ❖ An **interface** to communicate with a relational database
 - Allows database agnostic Java code
 - Treat database tables/rows/columns as Java objects
- ❖ **JDBC driver**
 - An implementation of the JDBC interface
 - Communicates with a particular database



JDBC steps

1. Connect to database
2. Query database (or insert/update/delete)
3. Process results
4. Close connection to database

1. Connect to database



❖ Load JDBC driver

- `Class.forName("com.mysql.jdbc.Driver").newInstance();`

❖ Make connection

- `Connection conn = DriverManager.getConnection(url);`

❖ URL

- **Format: "jdbc:<subprotocol>:<subname>"**
- `jdbc:mysql://128.100.53.33/GROUPNUMBER?user=USER&password=PASSWORD`



2. Query database



a. Create statement

- `Statement stmt = conn.createStatement();`
- `stmt` object sends SQL commands to database
- Methods
 - `executeQuery()` for SELECT statements
 - `executeUpdate()` for INSERT, UPDATE, DELETE, statements

b. Send SQL statements

- `stmt.executeQuery("SELECT ...");`
- `stmt.executeUpdate("INSERT ...");`



3. Process results



❖ Result of a SELECT statement (rows/columns) returned as a **ResultSet** object

```
ResultSet rs =  
    stmt.executeQuery("SELECT * FROM student");
```

❖ Step through each row in the result

```
rs.next()
```

❖ Get column values in a row

```
String studentName = rs.getString("name");  
int studentId = rs.getInt("id");
```

student table			
<u>id</u>	name	dept	gpa
123	John Rambo	Comp Sci	1.1
666	Jack Ripper	Biology	4.0



Print the student table



```
ResultSet rs = stmt.executeQuery("SELECT * FROM student");  
  
while (rs.next()) {  
    String id = rs.getInt(1);  
    String name = rs.getString("name");  
    String dept = rs.getString("dept");  
    double gpa = rs.getDouble("gpa");  
    System.out.println(id + " " + name + " " + dept + " " + gpa );  
}
```

student table			
<u>id</u>	name	dept	gpa
123	John Rambo	Comp Sci	1.1
666	Jack Ripper	Biology	4.0



Add a row to the users table

```
String str =  
    "INSERT INTO student  
    VALUES(123, 'John Rambo', 'Comp Sci', 1.1)";
```

```
// Returns number of rows in table  
int rows = stmt.executeUpdate(str);
```

student table			
<u>id</u>	name	dept	gpa
123	John Rambo	Comp Sci	1.1

4. Close connection to database

❖ Close the ResultSet object


- `rs.close();`

❖ Close the Statement object

- `stmt.close();`

❖ Close the connection

- `conn.close();`



```

import java.sql.*;
class MySqlTest {
    public static void main(String argv[]) throws Exception {

        Class.forName("com.mysql.jdbc.Driver");


        Connection c =
        DriverManager.getConnection("jdbc:mysql://localhost/mysql", "", "");

        Statement s = c.createStatement();
        ResultSet r = s.executeQuery("SELECT * FROM student");



        while (r.next())
            System.out.println(r.getInt(1) + "," + r.getString(2) +
            "," + r.getString(3) + "," + r.getDouble(4));

        s.close();
        c.close();
    }
}

```



Transactions

- 
- ❖ Currently every executeUpdate() is “finalized” right away
 - ❖ Sometimes want a set of updates to all fail or all succeed
 - E.g. add to Appointments and Bookings tables
 - Treat both inserts as one transaction
 - ❖ Transaction
 - Used to group several SQL statements together
 - Either all succeed or all fail
- 

Transactions

❖ Commit

- Execute all statements as one unit
- “Finalize” updates

❖ Rollback

- Abort transaction
- All uncommitted statements are discarded
- Revert database to original state

Transactions in JDBC

❖ Disable auto-commit for the connection

- `conn.setAutoCommit(false);`

❖ Call necessary `executeUpdate()` statements

❖ Commit or rollback

- `conn.commit();`
- `conn.rollback();`

```

Class.forName("com.mysql.jdbc.Driver");
Connection c =
DriverManager.getConnection("jdbc:mysql://localhost/mysql", "", "");
c.setAutoCommit(false);
try{
PreparedStatement p = c.prepareStatement(
"CREATE TABLE Student(id INTEGER, name VARCHAR(20), dept VARCHAR(20),
gpa REAL)");
p.executeUpdate();
p = c.prepareStatement("INSERT INTO Student VALUES (?, ?, ?, ?)");
BufferedReader br = new BufferedReader(new FileReader("data.txt"));
for (int i = 0; i < 4; i++) {
    p.setString(1, br.readLine());
    p.setString(2, br.readLine());
    p.setString(3, br.readLine());
    p.setString(4, br.readLine());
    p.executeUpdate();
}
c.commit();
br.close(); p.close();
}
catch( Exception e ){
    c.rollback();
}

```



Summary



❖ Day 1

- Chapter 1 : Introduction to Java
- Chapter 2 : Java Programming Language
 - 2.1 Declaration
 - 2.2 Operator
 - 2.3 Flow Control
- Chapter 3 : Java Object-Oriented Programming
 - 3.1 OO Concept
 - 3.2 Java Class
 - 3.3 Modifier
 - 3.4 Inheritance
 - 3.5 Polymorphism



Summary



❖ Day 2

- Chapter 4 : Exception Handling
- Chapter 5 : Using Java Classes
 - 5.1 String
 - 5.2 Wrapper Class
 - 5.3 Collection and Map
- Chapter 6 : Advance Topics
 - Networking Programming
 - Stream and Serialization
 - JDBC



