



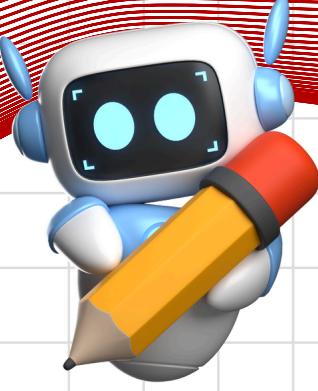
# JAVA & OOP

V1 (Update: 28/12/25)

Book javaRecab =  
new Book("Free");

JAVA & การប្រព័ន្ធឌីជីថល

BY CHAQUI.RL



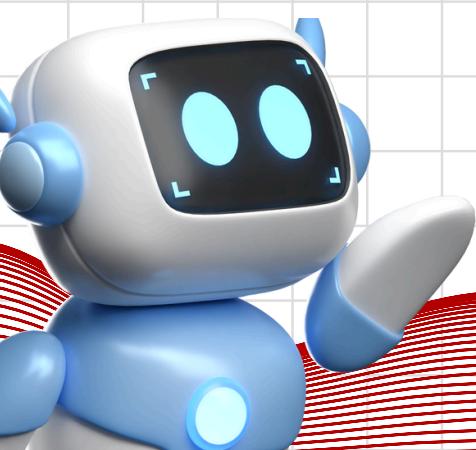
## - Preface -

เอกสารสรุปฉบับนี้เนื้อหาเกี่ยวกับภาษา Java และการโปรแกรมเชิงวัตถุ (OBJECT-ORIENTED PROGRAMMING) ซึ่งมีการนำเสนอเนื้อหาต่าง ๆ เช่น ไวยากรณ์, ชนิดข้อมูล รวมถึงตอนเชปต์ต่าง ๆ ในภาษา Java

สรุปฉบับนี้จัดทำโดย CHAQUI.RL โดยเป็นการแจกฟรีไม่มีค่าใช้จ่าย และ อนุญาตให้แชร์ต่อหรือนำไปใช้ประกอบสื่อการสอนได้โดยไม่ต้องขออนุญาตจากผู้จัดทำ ทั้งนี้ผู้จัดทำขอความร่วมมือไม่ลบทิ้งของผู้จัดทำออก / นำไปเป็นของตนเอง / จำหน่ายต่อ หากพบเห็นจะมีการดำเนินคดีตามกฎหมายท่อไป

หมายเหตุการใช้งาน AI: เนื้อหาบางส่วน, ตัวอย่างโค้ด และภาพประกอบในเอกสารนี้ มีการใช้งาน AI (GEMINI, CHATGPT) เข้ามาช่วยในการสรุปและจัดทำเพื่อความสะดวกและถูกต้องทางลิขสิทธิ์ อย่างไรก็ตาม เนื้อหาทั้งหมดได้รับการเรียบเรียงและตรวจสอบความถูกต้องโดยผู้เขียนเรียบร้อยแล้ว

ขอบคุณผู้อ่านทุกท่านที่สนใจ หวังว่าเอกสารนี้จะช่วยให้การเรียนรู้ภาษา Java ง่ายขึ้น หากมีข้อผิดพลาดประการใดต้องขออภัยมา ณ ที่นี้ด้วยครับ





## ສັແຈງເກີ່ມວິທີບຸນປະບາດການເຂົ້ານໂຄດໃນສຽບເລ່ອນນີ້



ໃນສຽບເລ່ອນນີ້ໄດ້ສຳການເຂົ້ານຕົວອ່າງໂຄດໂດຍຮວມຫລາຍຄລາສໄວ້ໃນໄຟລ໌ເຕືອນ (Main.java) ເພື່ອໃຫ້ຜູ້ອ່ານເຫຼົາໄອຄອນເຫັນເປັນໄດ້ຈໍາຍ ອ່າງໄຣກໍ່ຕາງໃນການເຂົ້ານໂປຣແກຣມຊີ້ງ ຄວາມແນກເປົ້ນພື້ນໆ class ຕ່ອ 1 ໄຟລ໌! ຕົວອ່າງການແນ່ດາໄຟລ໌:

```
class Car {  
    String color;  
    public Car(String color) {  
        this.color = color;  
    }  
    public void honk() {  
        System.out.println(color + " Beep! Beep!");  
    }  
    public void accelerate(int speed) {  
        System.out.println(color + " car is accelerating to " + speed +  
            "km/h.");  
    }  
    public void brake() {  
        System.out.println(color + " car is braking.");  
    }  
}
```

ຄວາມແນກອອກມາເຂົ້ານເປົ້ນ Car.java  
(ອ່ານື້ນເຕີ້ມ public ພໍ້າ class)



```
public class Main {  
    public static void main(String[] args) {  
        Car carA = new Car("red");  
        Car carB = new Car("blue");  
        carA.accelerate(45);  
        carA.honk();  
        carB.brake();  
    }  
}
```

Main.java

# ພຣີວແນ້ອໜາ

01.

## ແນະໜໍາລາຍາ Java



ບານີ້ຈະເປັນກາຮແນະສິ່ງທີ່ຄວຮຽ້ມບັນດາຕົ້ນເກື່ອງຈົບ Java

02.

## ໂປຣແກຣມແຮກຊອງສິ່ນ



ບານີ້ຈະສອນເຂົ້າໃຫຍ່ໂປຣແກຣມລາຍາ Java ຢູ່ໂປຣແກຣມແຮກ (HelloWorld)

03.

## ຮັບຄ່າຈາກ USER ດ້ວຍ Scanner



ບານີ້ຈະສອນເຂົ້າໃຫຍ່ໂປຣແກຣມລາຍາ Java ທີ່ສຳກັນຮັບຄ່າຈາກ Keyboard ຈອງຜູ້ໃຊ້

04.

## ຄລາສແລະວັດຖຸ



ບານີ້ຈະແນະໜໍາເກື່ອງຈົບພື້ນສູານ Object-Oriented programming ໃນລາຍາ Java ໂດຍເຮັດວຽກກາຮແນະໜໍາຄວາມໝາຍຂອງຄລາສແລະວັດຖຸກ່ອນ

05.

## ໄວຍາກຮົດພື້ນສູານ </>

ບານີ້ຈະແນະໜໍາໄວຍາກຮົດຕ່າງໆ ໃນລາຍາ Java ທີ່ຄວຮຽ້ມເຊື່ອ ກາຮສ້າງ object ເຊື່ອນໄຂ ກາຮທຳຊ້າ ແລະອື່ນ ແລະ

# ພຣີວເນ້ອຂາ

06.

ຈົດຂໍ້ມູນ



ສີ 3 ນາບຂອບ!!

ບານີ້ຈະແນະນຳເກືອງກັນຈົດຂໍ້ມູນຕ່າງ ຫຼັກາຊາ ປອນ ຮວ່າລົງຈາກ  
ແປລົງຈົດຂໍ້ມູນ

07.

ຕົວຮະບຸສິຫຼືເຂົ້າສົ່ງ



ບານີ້ຈະແນະນຳຕົວຮະບຸສິຫຼືການເຂົ້າສົ່ງໃນລາຍາ ປອນ ທີ່ຈະສືບຜົດຕ່າງ  
ການເຂົ້າສົ່ງ/ເຮັດໄຟ ຕົວແປຣ ຄລາສ ແລະ ເນືອດ

08.

ຕົວຄຳເນີນການ



ບານີ້ຈະແນະນຳຕົວຄຳເນີນການປະເທາຕ່າງ ຮວ່າລົງແສດງຕົວອໜ່າງ  
ປະດອບ

09.

ຕົວແປຣປະເທາຕ່າງ ໃຫ້



ບານີ້ຈະແນະນຳຕົວແປຣປະເທາຕ່າງ ໃຫ້ ແລະ ຂອບເຂົ້າມີການໃຊ້ງານ  
ຂອງໜີ້ນ

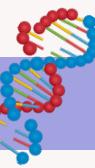
10.

ກາຮ້ອ່ານຸມຂໍ້ມູນ



ບານີ້ຈະແນະນຳເນືອດທີ່ກຳຈານກັນ Variable ທີ່ເປັນ private  
ໃນ OOP (Encapsulation)

# ພຣີວເໜືອຂາ



11.

## ກາຮສືບທອດຄຸດສະບັບຕິຮະກວາງຄລາສ

ບານີ້ຈະແນະນຳກາຮສືບທອດຕົວແປຣແລະເນືດອົດຈາກຄລາສແພ່ໄປບັງ  
ຄລາສສູງ

12.

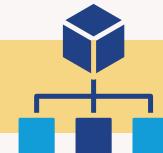
## ຄວາມເປົ້ານໍາມຕຮ່ານ



ບານີ້ຈະແນະນຳເກື່ອງກັບນໍາມຕຮ່ານ ທີ່ຈະເກື່ອງຈົບຈັນ Interface  
ແລະ Abstract class ໃນລາຍາ ຝູນດ

13.

## ກາຮພ້ອງຮູບ / ມື້ຈລາຍຮູບແບບ



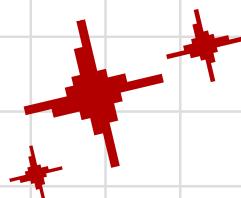
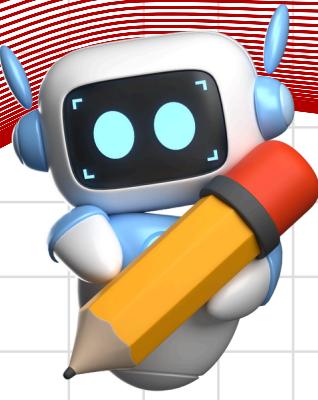
ບານີ້ຈະແນະນຳເກື່ອງກັບກາຮພ້ອງຮູບໃນລາຍາ ຝູນດ ທີ່ຈະ  
ເກື່ອງຈົບຈັນ Overloading ແລະ overriding

14.

## ກາຮຈັດກາຮຂ້ອຜິດພລາດໃນໂປຣແກຣມ



ບານີ້ຈະແນະນຳເກື່ອງກັບປະເທດຂ້ອຜິດພລາດຕ່າງ ຣ ແລະ ວິທີກາຮ  
ຈັດກາຮອ່າງເຫັນ try catch exception



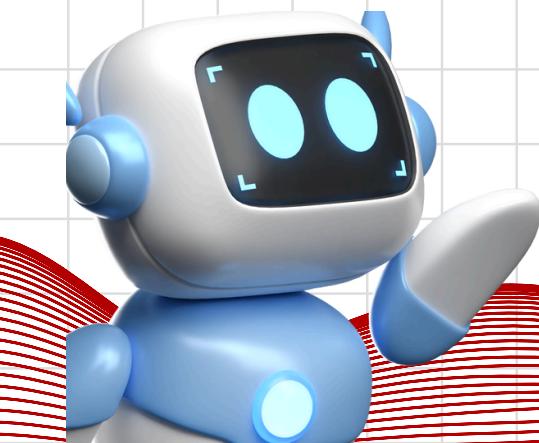
- Chapter 1 -

# INTRODUCTION แนะนำภาษา JAVA

เกี่ยวกับบทนี้



“บทนี้จะเป็นการแนะนำสิ่งที่ควรรู้เบื้องต้นเกี่ยวกับ JAVA”



# - ການໂພນ Java -

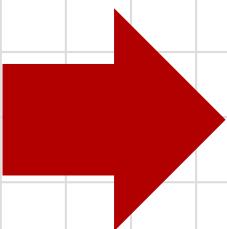
## Java គឺអេឡិចត្រូនុយ៉ាវា



ภาษา Java เป็นภาษาโปรแกรมเชิงวัตถุ (Object-Oriented Programming) พัฒนาโดย เจมส์ กอสลิ่ง และพีทเวิล์ดท์ที่บริษัท Sun Microsystems ในปี พ.ศ. 2534 ภาษาต่อไปนี้เป็นภาษาที่ใช้ในการพัฒนาซอฟต์แวร์เชิงโครงสร้าง เช่น Java, C++, C# และ Python ซึ่งมีความสำคัญอย่างมากในโลกปัจจุบัน

## Fun Fact!!

เดิมชื่อภาษา "ໂອກ" (Oak) ตั้งชื่อตามต้นไม้ชนิดที่อยู่ในพื้นที่ แต่ต้องเปลี่ยนชื่อเนื่องจากติดปัญหาลิขสิทธิ์ จึงเปลี่ยนเป็น "ຈາວາ" (Java) ซึ่งได้แรงบันดาลใจจากการแพร

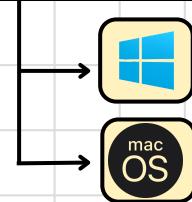
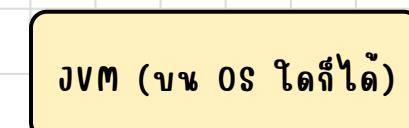
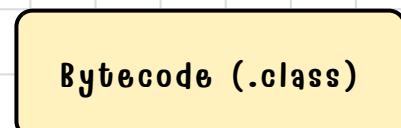
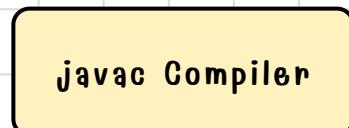
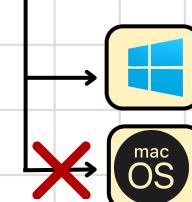


ກຳໄໄມ ກອງລ ຄື່ງເປັນທີ່ນິຍມ

ຈາກ ໂດົງຕັ້ງເພຣະນັນ "ເສົ່າຍຣ, ຂ້າມແພລົດພອຮ່າງໄດ້, ໄຊ້ງານໄດ້ຂລາຂລາຍ, ສີ່ຈຸ່ມຈຸນ  
ແລະເຄື່ອງສືອຮອງຮັບ" ເພີ້ມທີ່ກັບສືອໃໝ່ແລະອອກຄໍາຮະຕັບໂລກ



ตอนเชปต์เขียนครั้งเดียว รันໄດ້ທຸກທີ (Write Once, Run Anywhere)



JVM គົດວາງແລ້ວ???

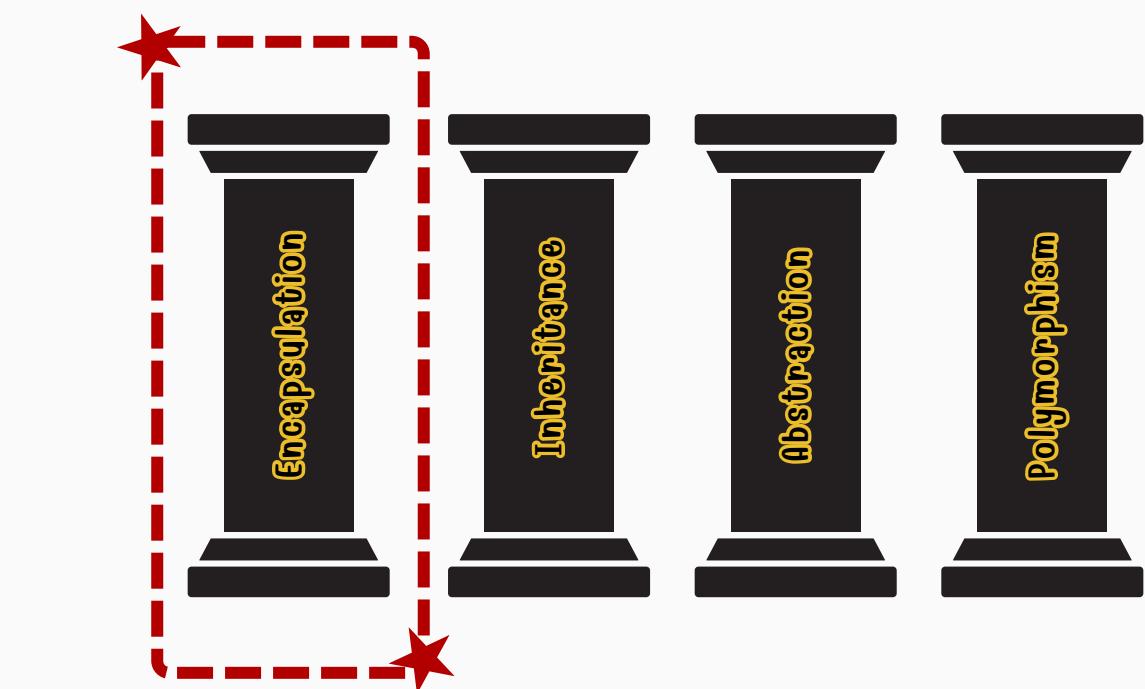
JVM (Java Virtual Machine) ຕົວເຄີຍແສ່ງໂສນທີ່ໃຫ້ສໍາຜັນ ຮັນໂປຣແກຣມ Java ໂດຍໜັນທຳໜ້າທີ່ແປລອງ Bytecode (.class) ໃຫ້ທຳຈານບ່ນຮະບນປົງປົງຕິກາຣອິງ ເຊິ່ງ Windows, macOS ຈີ່ອ Linux

Note: ຂາຍື່ງໄໝ່ເຂົ້າໃຈແພນກາພດ້ານບ່ນໜາກໜ້າ ໄໝ່ເປັນໄວ!!!! ເພີ່ງແມ່ຈຳຄອນເຫັນຈ່າຍ ຖ້າໄປວ່າ ປົນຈຸ ເຈີນຄັ້ງເດືອນ ເອາໄປໃຫ້ໄດ້ທຸກທີ່ກີ່ພອ ແພນກາພດ້ານບ່ນເປັນເພີ່ງກາຮ່າຍໃຫ້ເຫັນກາພຊັດເຈັນແບນລະເອີ້ນທຶນໝາກ ໆ ເທົ່າໜັນຄົບນີ້





## ตอนที่ 4 เสาหลักของ OOP (The 4 Pillars of OOP)



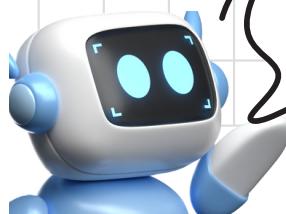
### Encapsulation (การห่อซุกซ่อน)

การห่อซ่อน ข้อมูล (Attributes) และ เมธอด (Methods) ที่ทำงานกับข้อมูลนั้นไว้ ในชั้นนอกเดียวกันที่เรียกว่า "คลาส" (Class) รวมถึงการจำกัดการเข้าถึงข้อมูลภายใต้ บางอย่างจากภายนอก ซึ่งเราเรียกว่า "Data Hiding"

#### ประโยชน์

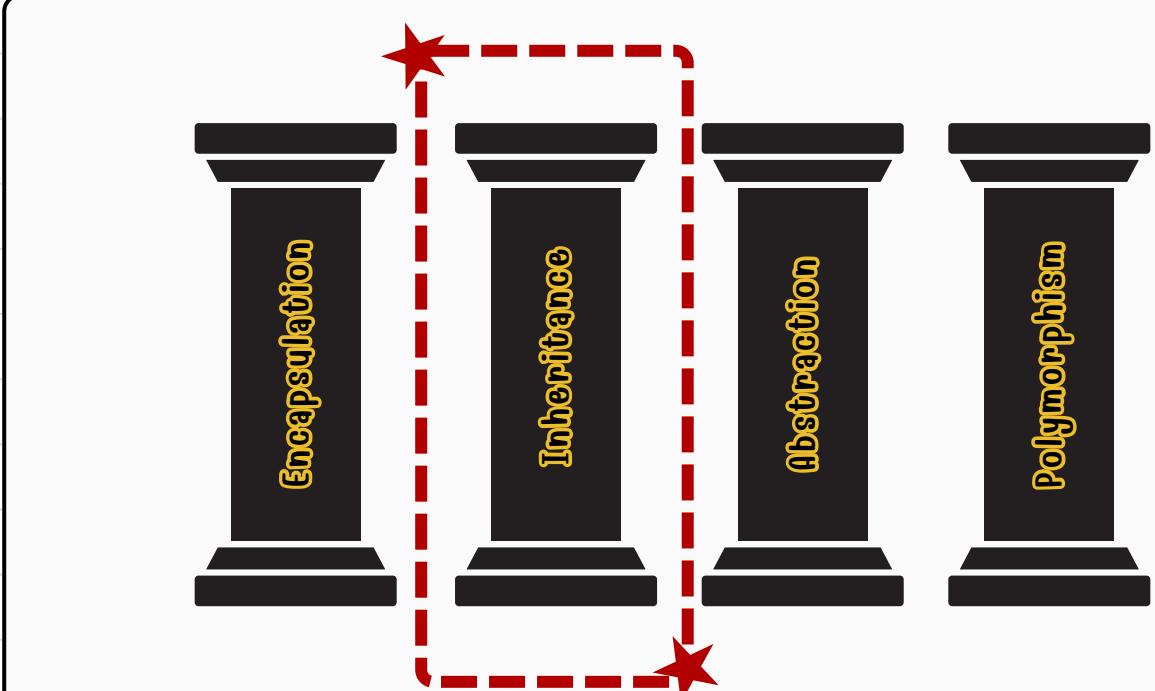
ช่วยปกป้องความปลอดภัยของข้อมูลโดยการควบคุมการเข้าถึงผ่าน เมธอดสาธารณะ (Public methods) และช่องรายการและอีกด้านการทำงานที่ซ่อนอยู่ในไฟล์ไฟล์ ภายนอกเพื่อน

Concept นี้จะถูกพูดถึง<sup>ใน</sup> Chapter 10:  
Encapsulation





## ตอนเชปต์ 4 เสาหลักของ OOP (The 4 Pillars of OOP)



### Inheritance (การสืบทอด)

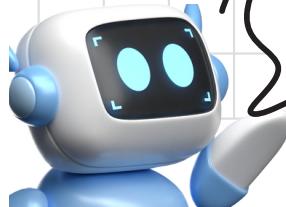
กลไกที่คลาส子辈 (Subclass หรือคลาஸลูก) สามารถรับการจากคลาสแม่และพูดคุยร่วมกับคลาสที่มีอยู่แล้ว (Parent class หรือคลาสแม่)



ประโยชน์

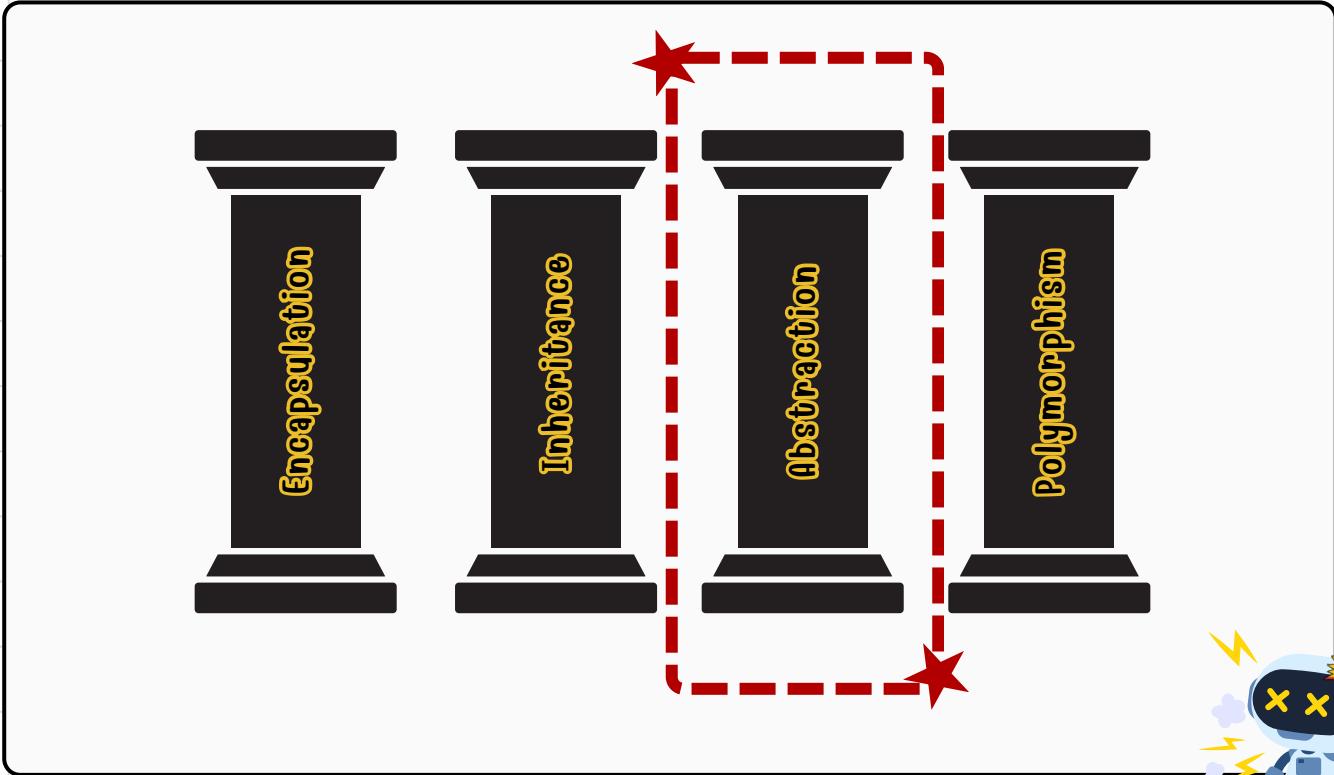
ช่วยลดการเขียนโค้ดซ้ำ (Code Reusability) โดยใช้เราต้องอ่านจากโค้ดเดิม แทนที่จะต้องปั๊วะ

Concept นี้จะถูกพูดถึง  
ยกใน Chapter 11:  
Inheritance





## ตอนเชปต์ 4 เสาหลักของ OOP (The 4 Pillars of OOP)



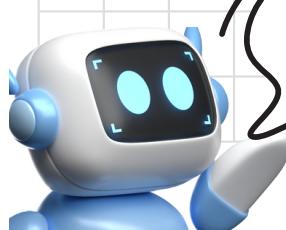
### Abstraction (นามธรรม)

การซ่อนรายละเอียดการทำงานที่ซับซ้อนไว้ข้างหลัง (เช่น เวลาเรียกใช้เมธอดบางอย่าง แค่รู้ชื่อ ก็เพียงพอ ไม่จำเป็นต้องไปรับรู้ว่าหลังบ้านมีงานยังไงก็ได้เป็นผลลัพธ์ที่เราต้องการออกมากได้) และใช้เฉพาะ "สิ่งที่จำเป็น" ให้ผู้ใช้เห็น

ประโยชน์

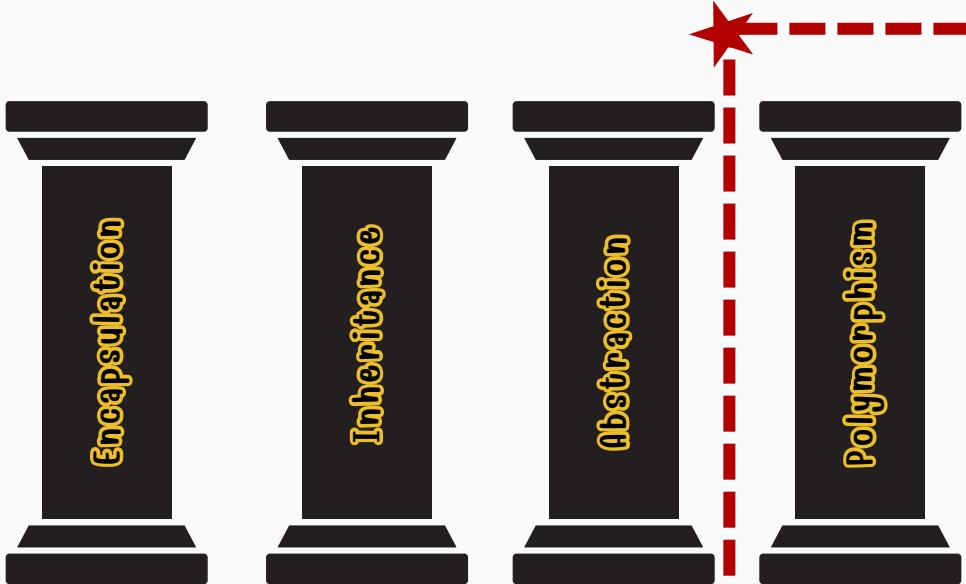
ช่วยลดความซับซ้อนของระบบให้ญี่บุนเดินได้เรียบง่ายขึ้น ผู้ใช้สามารถใช้งานได้โดยไม่ต้องเข้าใจภายใน

Concept นี้จะถูกพูดถึงใน Chapter 12:  
Abstraction





## ตอนที่ 4 เสาหลักของ OOP (The 4 Pillars of OOP)



### Polymorphism (การพ้องรูป)

ความสามารถที่วัตถุจะเปลี่ยนรูปได้หลายแบบ หรืออธิบายง่ายๆ คือ "เมื่อคลิ๊กเดียว ก็ แต่พุติกรรรมต่างกัน" ซึ่งอยู่กับว่าเรียกใช้ฟังก์ชันวัตถุอะไร สำหรับวัตถุคนละประเภท ทำได้ 2 วิธีคือ

- **Method Overriding:** คลาสลูกเขียนทับเมธอดของแม่เพื่อทำงานแบบของตัวเอง
- **Method Overloading:** สร้างเมธอดซึ่งเดิมแต่รับค่า (Parameters) ต่างกัน



ประโยชน์

ช่วยให้เขียนโค้ดที่เป็นกลาง (Generic) ได้มากขึ้น ไม่ต้องก้อนปี้โค้ดเดิมซ้ำๆ

Concept นี้จะถูกพูดถึง

อีกใน Chapter 13:

Polymorphism



ตัวอย่างโปรแกรม / เกมที่พัฒนาด้วยภาษา Java



Minecraft (Java Edition)



เกมแนว Sandbox / Survival ที่สร้างขึ้นด้วย ภาษา Java โดย Mojang Studios เปิดตัวในปี 2009 อุดเด่นของเวอร์ชันนี้คือ รองรับการติดตั้ง Mod, สร้างเซิร์ฟเวอร์เองได้ และรันได้บน Windows, macOS, Linux ผ่าน Java Virtual Machine (JVM)

MCEexample.java X



```
...
public class Block extends BlockBehaviour {
    public Block(BlockBehaviour.Properties properties) {
        super(properties);
    }
    public void stepOn(Level level, BlockPos pos, BlockState state, Entity entity) {
        super.stepOn(level, pos, state, entity);
    }
}
```



ກອນຈະເຮັ່ງພື້ນາໂປຣແກຣມ Java ຕ້ອງໂໜລດ JDK ຕິດເຄີ່ງດ້ວຍ!



JDK ສັນຄືອ ອີ່ຫຼັງ?????



JDK (Java Development Kit) ຄືອ .. ຊຸດເຄີ່ງມືອສໍາຫັນ ພື້ນາໂປຣແກຣມລາຍາ Java ລາຍໃນ JDK ດະວີ 3 ມີຫຼັກ ຖ້າ:

## 1. JRE (Java Runtime Environment)

- ສ່ວນທີ່ທຳໄໝຮັນໂປຣແກຣມ Java ໄດ້ (ຮອງ JVM)

## 2. javac

- ຄອນໄພເລອຮົ່ງທີ່ໃຊ້ແບບໂຄດ .java ໃຫ້ເປັນ .class (bytecode)

## 3. ເຄີ່ງມືອອິ່ນ ບໍ່

- ເຊື່ອ javadoc, jar, debugger ເປັນຕົ້ນ



## ໂໜລດ JDK

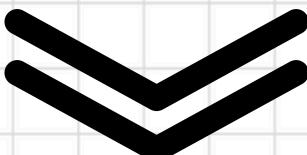
ແນະນຳໃໝ່ໂໜລດຈາກ Oracle JDK ຜົ່ງວິສິດາຮໂໜລດສາມາຮຄັນຈາໄດ້ຕາງ Internet ເພື່ອໂໜລດເສື້ອແລ້ວສາມາຮອໃຫ້ຄຳສັ່ງ java --version ເພື່ອຕຽວຊອບໄດ້!



ຕົວຢ່າງທີ່ຕິດຕັ້ງສໍາເລັກ:

```
C:\Users\ADMIN>java --version
java 17.0.9 2023-10-17 LTS
Java(TM) SE Runtime Environment (build 17.0.9+11-LTS-201)
Java HotSpot(TM) 64-Bit Server VM (build 17.0.9+11-LTS-201, mixed mode, sharing)
```

ຈົ້ນຕອນຕ່ອໄປກໍ່ຄືອກາຮເລືອກ IDE ທ້ອງການໃໝ່



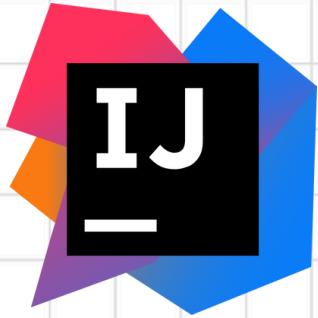
# Java Programming



## INTRO TO JAVA



IDE ທີ່ເພັນມະດັບການເສື້ອງໂປຣແກຣມ ຈອງ



# IntelliJ IDEA

- ຮອງຮັບພາສາ Java ຕີ່ທີ່ສຸດ!!!
  - ໄຈ້ານຈ່າຍແລະເຮືອງ
  - ຮອງຮັບ Spring, Maven, Gradle, Android, Kotlin



# NetBeans

- ໃຈ່ງານຈ່າຍ
  - ເຊັມກະກົບນີກເຮືອນ / ສົ່ວໂລມ່ເຮືອນໂຄດ



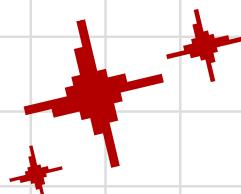
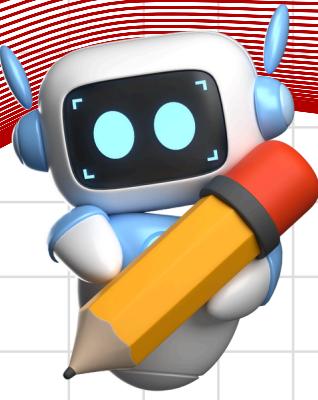
## Eclipse

- Փրկ 100% և բերածություններ
  - Հիմնական գործությունները կազմակերպված են ուղարկություններում՝ ուղարկությունների համար և առաջնահատիքների համար:
  - Առաջնահատիքը կազմակերպված է ուղարկությունների համար և առաջնահատիքների համար:

VS Code

- UI ที่นักเขียนสามารถกำหนดเอง!
  - ใช้งานง่ายผ่าน Extension สำหรับ Java





- Chapter 2 -

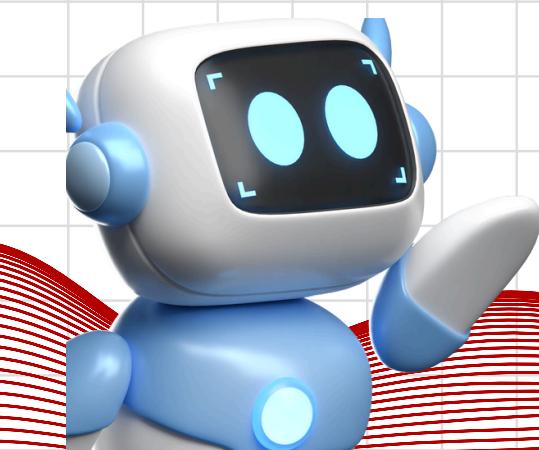
# MY 1ST PROGRAM

## เริ่มเขียนโปรแกรมแรก

เกี่ยวกับบทนี้



“บทนี้จะสอนเขียนโปรแกรมภาษา JAVA โปรแกรมแรก  
(HELLOWORLD)”





## -My First Program-

ขึ้นตีต้อนรับเข้าสู่พาร์ทที่จะได้เขียนโปรแกรมไงที่สุด!! เราเริ่มจากการเขียนตามไปก่อนแล้วค่อยๆ เรียนรู้ส่วนประกอบของโปรแกรมที่เราเขียน เริ่มด้วยโปรแกรมชื่อคลาสสิกอย่าง Helloworld !

Helloworld.java X



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

Output:

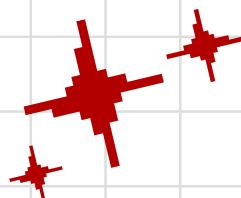
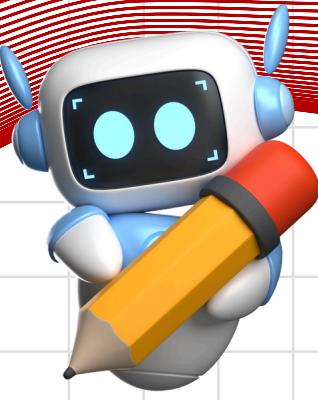
HelloWord

ข้อควรระวัง!

ตัวอักษรพิมพ์เล็กพิมพ์ใหญ่เข้ม!  
ต้องบันทึก! (Case Sensitive)



โคตรധา!



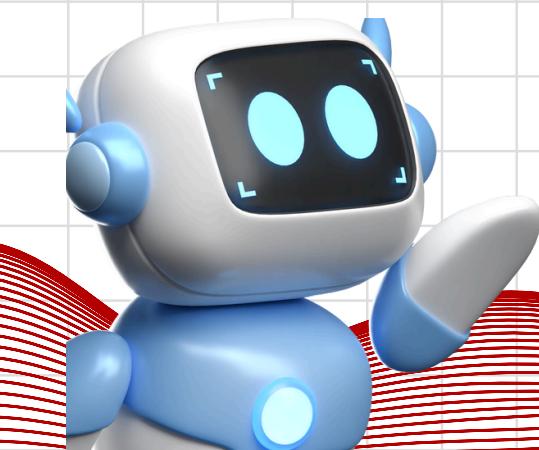
- Chapter 3 -

# JAVA SCANNER

## รับค่าจาก USER

เกี่ยวกับบทนี้ 

“บทนี้จะสอนเขียนโปรแกรมภาษา JAVA ที่มีการรับค่า  
จาก KEYBOARD ของผู้ใช้”





## ปุ๊ลจุ๊บ รีบค่าด้วย Scanner~

เมื่อเราพ่อไปประมวลผลกันไปแล้วก็เข้าสู่ประมวลที่ 2 กัน!

พิขพ์ข้อความปกติขึ้นมาไปแล้ว!! じゃเราต้องการรับค่าจากคีย์บอร์ดผู้ใช้ด้วยหล่ะ..  
เรื่องนี้เลย!



การที่เราจะรับค่าจากคีย์บอร์ดผู้ใช้ในภาษา Java ปุ๊บ นั้นสามารถทำได้โดยการสร้าง Object จาก Class Scanner และเรียกใช้ Method ของ Class Scanner เพื่อรับค่าจากคีย์บอร์ด (คงจะเข้าใจ Object และ Class มากขึ้นอ่างๆนะครับ)

Helloworld.java X

ต้อง import ก่อนใช้งานด้วยเด้อสู!



```
import java.util.Scanner;
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Please enter your name: ");
        Scanner scan = new Scanner(System.in);
        String name = scan.nextLine();
        System.out.println("Hello " + name);
    }
}
```

สร้าง Object ชื่อ scan  
จาก Class Scanner

ใช้ Object ชื่อ scan  
เรียก Method จาก Class Scanner

Output:

Please enter your name:

ChaurrelZ

Hello ChaurrelZ



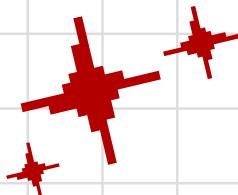
## Method ที่ใช้รับค่าประเภทอื่น ๆ ใน คลาส Scanner



เรารู้กันไปแล้วครึ่งว่า Method `nextLine()`; ที่ใช้รับค่า `String` 1 แต่ถ้า  
อยากรับค่าอื่นๆ กันบ้างล่ะ? `int float` เราสามารถใช้ `nextLine()` ไม่ใช่?  
คำตอบคือ... ไม่ได้!!! เราต้องใช้ method เดพาะของ `Data type` นั้นนะ!



ชื่อ Method	Datatype	คำอธิบาย
<code>nextByte()</code>	<code>Byte</code>	ใช้รับค่า <code>Byte</code> จาก <code>Input</code>
<code>nextBoolean()</code>	<code>Boolean</code>	ใช้รับค่า <code>Boolean</code> จาก <code>Input</code>
<code>nextInt()</code>	<code>Int</code>	ใช้รับค่า <code>Int</code> จาก <code>Input</code>
<code>nextFloat()</code>	<code>Float</code>	ใช้รับค่า <code>Float</code> จาก <code>Input</code>
<code>nextDouble()</code>	<code>Double</code>	ใช้รับค่า <code>Double</code> จาก <code>Input</code>
<code>nextLong()</code>	<code>Long</code>	ใช้รับค่า <code>Long</code> จาก <code>Input</code>
<code>nextShort()</code>	<code>Short</code>	ใช้รับค่า <code>Short</code> จาก <code>Input</code>



- Chapter 4 -

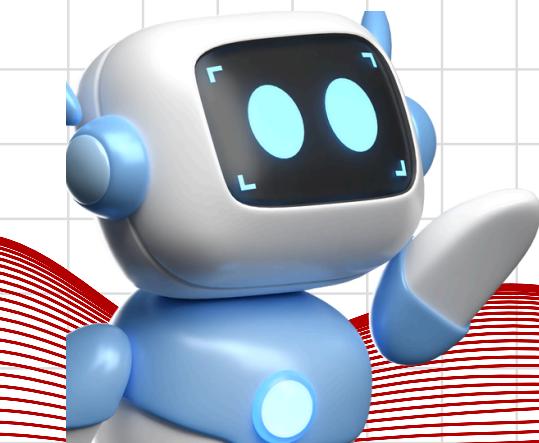
# CLASS & OBJECT

## คลาสและวัตถุ

เกี่ยวกับบทนี้



“บทนี้จะแนะนำเกี่ยวกับพื้นฐาน OBJECT-ORIENTED  
PROGRAMMING ในภาษา JAVA โดยเริ่มจากการแนะนำ  
ความหมายของคลาสและวัตถุก่อน”



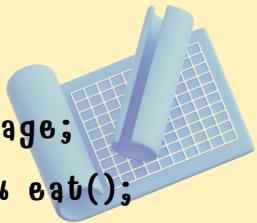


## -Class & Object-

### Class

เป็นโครงสร้างแบบสำหรับสำหรับสร้าง Object กำหนดค่าไว้

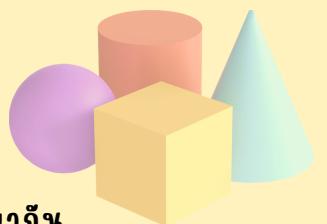
- คุณสมบัติ (Attribute): ข้อมูล/สถานะของ Object เช่น Int age;
- พฤติกรรม (Method): พฤติกรรมที่ Object สามารถทำได้ เช่น eat();



### Object

เป็นสิ่งที่สร้างจาก Class ที่กำหนด

- มีค่า Attribute จริง ๆ ในตัว
- สามารถเรียกใช้ Method ของ Class เพื่อทำงานได้
- Object แต่ละตัวมีข้อมูลของตัวเองแยกสร้างจาก Class เดียวกัน



### Class และแบบ



### Class Car

```
accelerate();
honk();
brake();
```



### Object เรียกใช้ Method



```
carA.accelerate(45);
```



```
carA.honk();
```



### Object เรียกใช้ Method



```
carB.brake();
```

# Java Programming



CLASS & OBJECT

ຕົວອອນໄປຮັດຮັບ myCar ຈາກໜ້າທີ່ແລ້ວ

class Car { Class ແຂ່ງມານ

String color;

public Car(String color) {  
 this.color = color;  
}

Constructor method

public void honk() {

System.out.println(color + " Beep! Beep!");

}

public void accelerate(int speed) {

Method ຢອງ Class

System.out.println(color + " car is accelerating to " + speed +  
 "km/h.");

}

public void brake() {

System.out.println(color + " car is braking.");

}

}

public class Main {

public static void main(String[] args) {

Car carA = new Car("red");

สรໍາງ Object

Car carB = new Car("blue");

carA.accelerate(45);

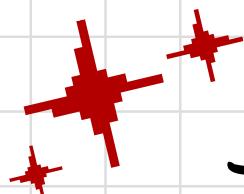
Object ເຮັດວຽກ Method

carA.honk();

carB.brake();

}

}



- Chapter 5 -

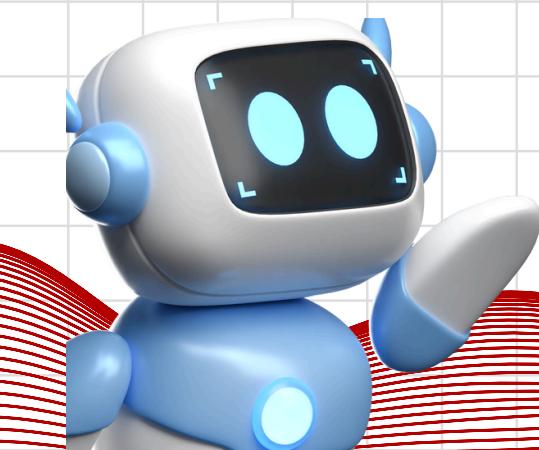
# SYNTAX

# ไวยากรณ์พื้นฐาน

เกี่ยวกับบทนี้



“บทนี้จะแนะนำไวยากรณ์ต่าง ๆ ในภาษา JAVA ที่ควรรู้ เช่น การสร้าง OBJECT เนื่องไป การทำซ้ำ และอื่น ๆ ”





## -Syntax พื้นฐาน-

### การประกาศ Class

- Class คือ ชื่อใจของ จ Ava เลยนะ!
- Class คือ แม่พิมพ์ของวัตถุทั้งปวง
- Java ประกอบด้วย Class 1+ ชิ้น ไป

//syntax

```
public class ชื่อคลาส {
//bodyๆๆๆๆ
//Method และ Variable
}
```



### การประกาศ Variable

- Aka. Attribute หรือ Field !
- Variable คือ ตัวที่บอกลักษณะของ Object แต่ละตัว (แต่ละ Obj. จะ แยกกัน)

//syntax

```
modifiers datatype ชื่อตัวแปร ;
// Ex: public int x ;
// Ex: public static int y ;
```



### การประกาศ Method

- Method คือ สิ่งที่กำหนดให้ Object ในคลาสนั้น สามารถทำได้

//syntax

```
modifiers returnType ชื่อเมธอด(parameter){
//ส่วน Body ของเมธอด
} //Ex. private int methodA(int x) {...}
//Ex. private static int methodB(int x) {...}
```



### การเขียน Comment

- Comment สำหรับเพื่อการอธิบายโค้ด
- คอมเม้นท์จะไม่ประมวลผล ข้อความเหล่านี้
- javadoc คือ เอกสารประกอบโค้ด

//This is a comment

```
/*This is also a comment
hi hi hi hi hi */
/** {javadoc} */
```





## -Syntax พื้นฐาน-

### การสร้าง Object

- การสร้าง Object สามารถสร้างได้ผ่าน Constructor

//syntax

ชื่อคลาส ชื่อobjexec = กดพ ชื่อคลาส();  
//Ex. Car carA = new Car();



### ประโยคเงื่อนไข if else switch

- if - else:** ตรวจสอบเงื่อนไข ถ้าจริง ทำสิ่งหนึ่ง ถ้าไม่จริงทำอีกอย่าง
- switch:** เลือกทำงานตามค่าที่ตรงกับ case ที่กำหนด

//if-else

```
if (เงื่อนไข) { //คำสั่ง}
else if (เงื่อนไขอื่น) { //คำสั่ง}
else { //คำสั่ง}
```

//switch case

```
switch (ตัวแปร) {
    case ค่าที่1://คำสั่ง break;
    case ค่าที่2://คำสั่ง break;
    default://ไม่ตรง case ไหน
}
```



### การทำซ้ำ (Iteration)

- for:** วนซ้ำจำนวนรอบที่ระบุไว้ในวงเล็บ
- while:** วนซ้ำตราบใดที่เงื่อนไขเป็นจริง
- do - while:** ทำงานอย่างน้อย 1 รอบแล้วค่อยตรวจสอบเงื่อนไข
- for-each:** วนอ่านค่าทุกตัวใน array หรือ collection

//for loop

```
for (int i = 0; i < 5; i++) { //คำสั่ง}
```

//while loop

```
while (เงื่อนไข) { // คำสั่ง}
```

//do-while loop

```
do { // คำสั่ง} while (เงื่อนไข);
```

//for-each loop

```
for (ชนิดตัวแปร item : อาร์เรย์หรือ
collection) { // คำสั่งแต่ละ item}
```





## -Syntax พื้นฐาน -

### Break & Continue

- ใช้ร่วมกับการวนลูป (iteration) เพื่อควบคุมพฤติกรรมของลูป
- break:** จะออกจากลูปทันที
- continue:** ไม่ได้ออกจากลูปทันที เพียงแค่ข้ามไปทำตัวถัดไป
- หากໂຄດด้านขวาเลขที่จะถูกพิมพ์ออก อาทิ 1 3 และ 5 (เมื่อ i เท่ากับ 2, 4, 6 จะถูกข้าม(continue) และ เมื่อ i เท่ากับ 7 จะออกจากลูปทันที (break) 8, 9 ไม่ถูกทำ)

### //syntax (ตัวอย่าง)

```
for (int i = 0; i < 10; i++) {
    if (i % 2 == 0){
        continue;
    }
    if (i == 7){
        break;
    }
    System.out.println(i);
}
```



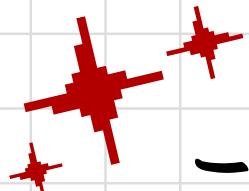
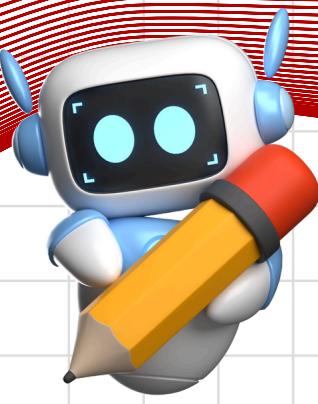
### ฟังก์ชันเรียกตัวเอง (Recursion)

- เมื่อต้องการให้ทำงานตัวเองซ้ำๆ เพื่อแยกปัญหาโดยแบ่งเป็นปัญหา ข้อๆ จนกว่าจะถึงจุดสิ้นสุด และส่งค่าคืนกลับมาเป็นท่าตอนๆ
- Base Case (จุดหยุด):** เมื่อนำมาสู่ "จุดหยุดเรียกตัวเอง" และเริ่มคืนค่ากลับ (Return)
- Recursive Case (จุดเรียกซ้ำ):** ส่วนที่ "เรียกตัวเองอีกครั้ง" และเปลี่ยนค่าพารามิเตอร์ให้ "ขั้นเช้าใจลึก Base Case มากขึ้น"

### //syntax (ตัวอย่าง)

```
int factorial(int n) {
    // Base Case:
    if (n <= 1) {
        return 1;
    }
    // Recursive:
    else {
        return n * factorial(n - 1);
    }
}
```

เครื่องหมาย  
ฟังก์ชันเดิม



- Chapter 6.1 -

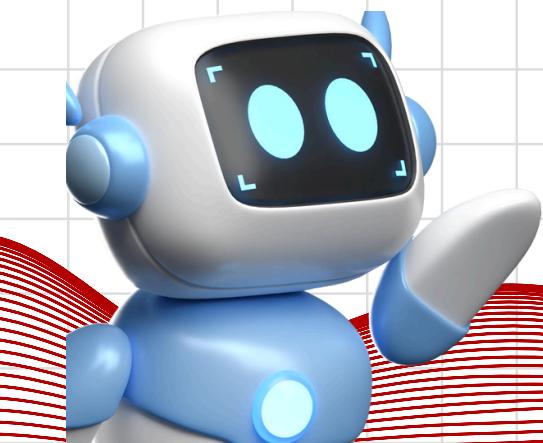
# DATATYPE:PRIMITIVE

## ชนิดข้อมูล

เกี่ยวกับบทนี้



“บทนี้จะแนะนำเกี่ยวกับชนิดข้อมูลต่าง ๆ ในภาษา  
JAVA รวมถึงการแปลงชนิดข้อมูล”





## -Datatype-

Primitive Datatype หรือ ชนิดของข้อมูลในภาษา Java ป้องกันมีทั้งหมด 8 ชนิด

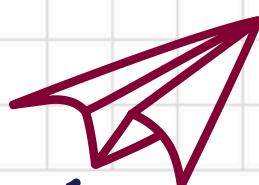


### --- 6 ชนิดตัวเลข ---

ชนิดข้อมูล	ขนาด (Bit)	ค่าต่ำสุด	ค่าสูงสุด	ใช้งาน
Byte	8	-128	127	จำนวนเต็มขนาดเล็ก
short	16	-32,768	32,767	จำนวนเต็มขนาดกลาง
int	32	-2,147,483,648	2,147,483,647	จำนวนเต็มทั่วไป
long	64	-9,223,372,036,854,775,808	9,223,372,036,854,775,807	จำนวนเต็มขนาดใหญ่
float	32	~1.4E-45	~3.4E38	ทศนิยมความแม่นยำเดียว
double	64	~4.9E-324	~1.8E308	ทศนิยมความแม่นยำมาก

### --- 2 ชนิดพิเศษ ---

ชนิดข้อมูล	ขนาด (Bit)	ค่าต่ำสุด	ค่าสูงสุด	ใช้งาน
char	16	\u0000	\uffff	เก็บตัวอักษรแบบ Unicode 1 ตัว
boolean	1	false	true	ค่าความจริง





## Suffix ในภาษา Java

Suffix ในภาษา Java คือ ตัวอักษรที่เขียนต่อท้ายคำคงที่ (Literal)

เพื่อระบุชนิดข้อมูลที่ต้องการใช้โดยตรง แนวค่าเดิมจะถูกตีความเป็นชนิดอื่นโดย  
บริขาร ตัวอย่างเช่น จ้าเขียน 3.14 โดยไส้ Suffix ค่าจะเป็น double  
อันนี้มีตัวต่อๆ กันไส้ 3.14F ค่าจะถูกมองเป็น float หันที่

Suffix	ใช้กับชนิดข้อมูล	ตัวอย่าง	หมายเหตุ
L หรือ L	long	100L	ใช้ L ตัวใหญ่ดังชุดเดนกว่า เพราะ L เสียอางสีบน หัวเลข 1
F หรือ F	float	3.14F	จ้าไส้ Suffix จะเป็น double โดยอัตโนมัติ
D หรือ D	double	3.14D	ปกติไม่จำเป็นต้องไส้ เพราะค่าทางคณิตเป็น double อยู่แล้ว
ไส้ Suffix	int (จำนวนเต็ม), double (จำนวน)	100, 3.14	เบื้องค่าเดียวไส้ จ้าไส้

SuffixEx.java

X



```
public class SuffixEx {
    public static void main(String[] args) {
        long population = 7800000000L;
        float pi = 3.14159F; ——————
        double g = 9.81D;
    }
}
```

แล้วจ้าเราหางึงไส้ F ตาม  
หลัง เพราะซึ่เดียจหล่อ!?  
จะได้ไหม?????  
(คำตอบหน้าจัดไบ)



ເຊື່ອຈາກພາກອ່ນຫ້າ..... float pi = 3.14159; ພັງ!!!!!!

ເພຣະໃນ Java ຕົວເລີຂາທຄນິຍາມທີ່ເຈີບນຕຮງໆ (ເຊື່ນ 3.14159) ຈະ ຖຸກຂອງເປົ້ນ double ທີ່ເອງ!! float ໄສ້ 32 ບົດ ແຕ່ double ໄສ້ 64 ບົດ ຕັ້ງນີ້ ກາຣເອາ double ມາໂສ່ໃນ float ດືອກາຮ ແປລອຈາກພິໂສູ່ ເປົ້ນ ອົບດເລີກ ທີ່ Java ໄວ່  
ຂອມທຳໄໝອັຕໂນໜັດ ຕົ້ນບອກໄໝຈົດວ່າຈະຂອມໄໝແປລອງ

## Type casting ในภาษา Java

### Implicit Casting (Auto)

- ແປລອໂດຍອັຕໂນໜັດ ເສື່ອແປລອຈາກ  
ອົບດເລີກໄປໄໝຈົດ
- ຕົວອ່າງ: int → long → double

```
int a = 10;
double b = a; /* ທຳມານໄດ້
(Implicit) (int → double) */
```



### ໂພຍ! Primitive Datatype ເລີກ → ໄໝຈົດ

byte → short → int → long → float → double

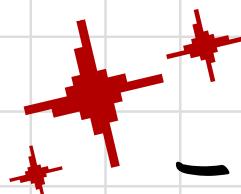
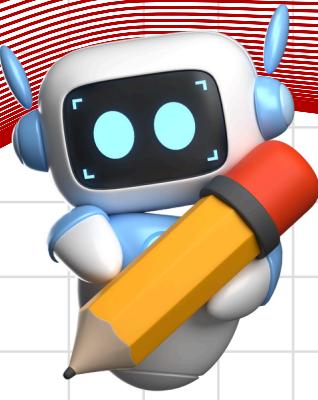


### Explicit Casting (Manual)

- ຕອງຮຽນ type ປາຍຫາງໃນວົງເລີກ  
ເສື່ອແປລອຈາກພິໂສູ່ໄປເລີກ (ອາຈ  
ສູງເສີ່ງຂໍ້ມູນ)
- ຕົວອ່າງ: double → int

```
double x = 10.5;
int y = (int) x; /* y = 10 (ດີດ
ຖານທີ່ຈິງ) */
```





- Chapter 6.2 -

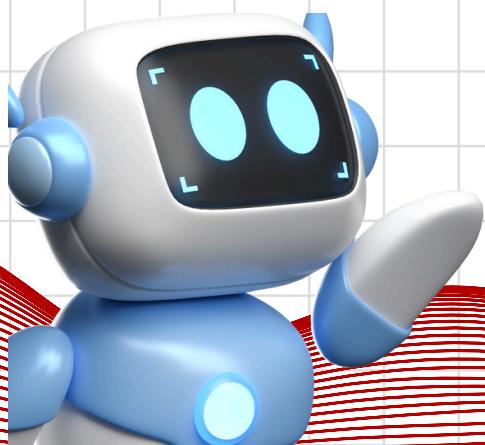
# DATATYPE:STRING

## ข้อมูลแบบสตริง

เกี่ยวกับบทนี้



“บทนี้จะแนะนำเกี่ยวกับข้อมูลแบบสตริง [ชุดของตัวอักษรหลาย ๆ ตัว] หรือที่เรียกว่า “ก็อต” ข้อความนั้นเอง”





S T R

-String-

String ในภาษา Java ไม่ใช่ Primitive Type!! (ไม่ใช่ชื่อ int, boolean) แต่เป็น Class (Reference Type) จึงมั่นใจว่าจะใช้ง่ายเมื่อตอน Primitive Datatype ถ้าตาม อย่างสับสน!!

I N G

## การประกาศใช้ String

- ถึง String จะเป็น Class แต่เวลาจะใช้ ไม่ต้อง Import เพราะ String อยู่ใน Package java.lang
- แม้เป็น Class แต่ Java ของใช้เราสร้างขึ้นจากๆ ด้วยเครื่องหมาย "" (Double Quote) ได้เลขโดยไม่ต้องกอน ตลอดเวลา

```
String a = new String("Hello");
//ใช้ "" ได้ ไม่ต้อง กอน เช่น
String b = "Hello";
```

## String Immutability

- เมื่อสร้าง String ขึ้นมาแล้ว แก้ไขค่าซึ่งในไม่ได้
- แต่ถ้าทำการแก้ค่า มันจะ ไม่ Error แต่ป้อมๆ จะสร้าง Memory ก้อนใหม่ๆ เลย ไม่ได้แก้ที่ก้อนเดิม

```
String s = "Java";
s = "Deluxe";
//แก้ค่า = สร้างเข็มก้อนใหม่!
```

String s = "Java";

s → Java

s = "Deluxe";

s → **New!**  
Java  
Deluxe



## การต่อความ

- เราสามารถนำ String มาต่อกันได้โดยใช้ เครื่องหมาย +
- จ้าเรา ตัวเลข + String ผลลัพธ์จะกลายเป็น String เช่น

```
String s = "Java" + "Deluxe";
//JavaDeluxe
String intS = 10+20+"Baht";
//30Baht
String sInt = "Baht" + 10 + 20;
//Baht1020
```



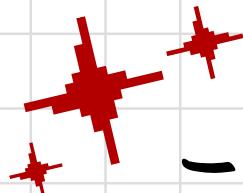
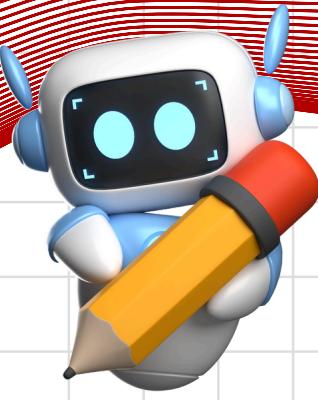
## การเปรียบเทียบ String

- == == ในการเปรียบเทียบ!!
- == : เช็คว่า "เป็นวัตถุก่อนเดี๋ยวกันในเมฆอะไรรึไม่รึ"
- .equals() : เช็คว่า "ต้องอักษรซ้ำกันใน เชือกกันไม่จะ"

```
String a = "Hello";
String b = "Hello";
if (a == b) { ... } //ผิด!!
if (a.equals(b)) { ... } //ถูก!!
```



Method ของนี้ข	功用	ตัวอย่าง	ผลลัพธ์
.length()	จำนวนตัวอักษร	"Hi".length()	2
.toUpperCase()	แปลงเป็นตัวพิมพ์ใหญ่	"java".toUpperCase()	"JAVA"
.toLowerCase()	แปลงเป็นตัวพิมพ์เล็ก	"JAVA".toLowerCase()	"java"
.trim()	ตัดช่องว่างหน้า-หลัง	" ok ".trim()	"ok"
.indexOf(str)	หาตำแหน่งค่า	"Cat".indexOf("a")	1
.charAt(index)	ดึงตัวอักษรตามตำแหน่ง	"Cat".charAt(0)	0



- Chapter 6.3 -

# DATATYPE:COLLECTION

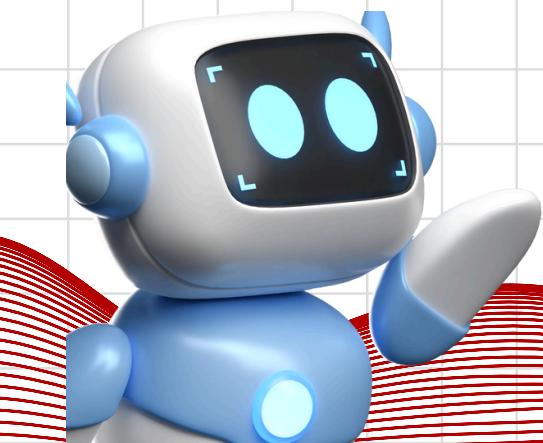
## ข้อมูลแบบคอลเลคชัน

เกี่ยวกับบทนี้



“บทนี้จะแนะนำเกี่ยวกับข้อมูลแบบคอลเลคชันได้แก่

**ARRAY, ARRAYLIST, HASHMAP**





## -Array-

Array ในภาษา Java ไม่ใช่ Primitive Type!! เช่นเดียวกับ String และ Array เป็นเชิงมืออาชีพมากของที่มีหลาย ๆ อย่าง แต่ละอย่างจะมีข�性และ操作 ว่าเป็นช่องที่เท่าไหร่ (index) และทั้งตู้เก็บของประเภทเดียวกันหมด

### การประกาศใช้ Array

- Array ไม่จำเป็นต้อง import อะไรไว้เพื่อใช้ได้เลย
- เวลาประกาศเราอาจจะต้องระบุขนาดให้ชัดเจนตั้งแต่แรกเลย ไม่สามารถซึ่งกันและกันได้
- สามารถใส่ข้อมูลลงไปตั้งแต่ตอนประกาศ หรือ ค่อยอพเดตหลังจากจบการพิมพ์ที่ไว้แล้วก็ได้

```
int[] scores = new int[5];
scores[0] = 80;
scores[4] = 95;

int[] ages = {18,0,0,0,21};
```

Array: scores

index	0	1	2	3	4
value	80	0	0	0	95

Array: ages

index	0	1	2	3	4
value	18	0	0	0	21



ค่า Default ของ Array แต่ละชนิด แล้วจ้าไม่กำหนดค่าให้ขึ้นนะพี่!!

- ตัวเลข (int, double): 0 หรือ 0.0
- ตรรกะ (boolean): false
- ว่าง (String, Object): null

การเข้าถึงและแก้ไขข้อมูลใน Array

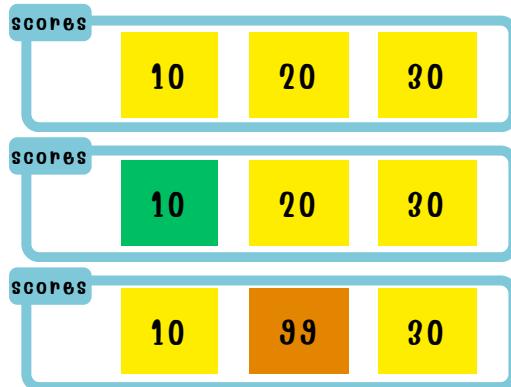
```
int[] scores = {10, 20, 30};  

// อ่านค่า (ใช้ Index)  

int x = scores[0]; // 10  

// แก้ไขค่า (ใช้ Index)  

scores[1] = 99; // 20 -> 99
```



การหาความยาวของ Array (Length)

```
int size = scores.length;
```

คำเตือน : length ห้ามส่องเงื่อน () นะ !!



การวนลูปเข้าถึงสมาชิกใน Array (Iteration)

```
for (int i = 0; i < scores.length; i++) {  

    System.out.println("Index " + i + ": " + scores[i]);  

}
```

For loop

```
for (int score : scores) {  

    System.out.println(score);  

} // ใช้ได้กับ Array scores ด้วย!!
```

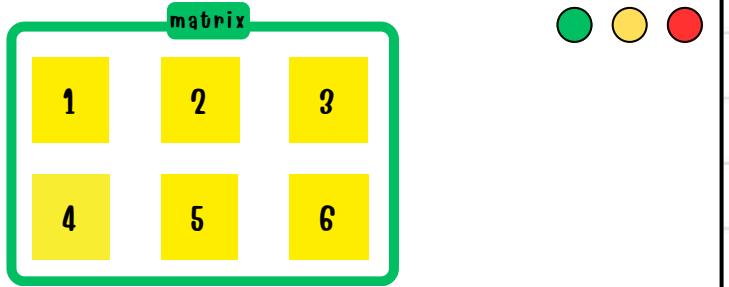
For each



## Array หลายมิติ (Multidimensional Array)

- เวลาสร้าง Array 2 มิติให้เราใช้จีบภาพ ตารางที่มีแถวและคอลัมน์

```
int[][] matrix = { {1, 2, 3},  
                   {4, 5, 6} };
```



- การเข้าถึงข้อมูล ใช้ Index เชื่อมเดิม โดยเราจะเริ่มจาก Index ของแถว ตามด้วย Index ของคอลัมน์ ของข้อมูลที่เราต้องการ
- เช่น ต้องการเข้าถึงข้อมูลเลข 4 จากภาพด้านบนเรารู้ว่า เลข 4 อยู่แถวที่ 2 (Index = 1) และ คอลัมน์ที่ 1 (Index = 0) จะได้ว่า int four = matrix[1][0]; นี่เอง

## Class สำหรับสุดโลกที่ช่วยจัดการ Array

```
import java.util.Arrays;
```

ตัวอย่าง method ที่ทำให้ชีวิตง่ายขึ้นล้านเท่า!!

- `.sort()` : ใช้เรียงลำดับจากน้อยไปมาก
- `.toString(array)` : ปรินต์ค่าใน Array จา `print(data)` เฉพุ ฉะนี้เลขที่อยู่ใน Memory อ่านໄ้ร์รูเรื่อง ต้องแปลงเป็น String ก่อน
- `.binarySearch(array, x)` : ค้นหาข้อมูล (ค่า x) (คือค่าเป็น Index)

```
int[] data = {5, 1, 9, 3};  
  
Arrays.sort(data);  
  
// ผล: {1, 3, 5, 9}  
System.out.println(Arrays.toString(data));  
  
// ผล: "[1, 3, 5, 9]"  
int index = Arrays.binarySearch(data, 5);  
  
// ผล: 0
```



## ArrayList

ເບື່ອໄຫວ? ການທີ່ຈະສ້າງ Array ແລ້ວຕ້ອງພາວິດວ່າຈະສ້າງຂາດເທົ່າໄຫວ່ດີ? ແລ້ວຈ້າເຮົາໄຟ່ສາມາດຄັດເຕົາໄດ້ລ່ວງໜ້າວ່າຈະອອງພື້ນທີ່ເທົ່າໄຫວ່ດີ? ລອງເປັນຢ່າງໃຊ້ ArrayList ສີ!! ເບື່ອ ArrayList ທີ່ "ເົືດຈົດຂາດໄດ້ເອງອັດໂນໜັດ" (Dynamic Array) ໄນມີກຳນົດຂາດລ່ວງໜ້າ

### ການປະກາດໃຊ້ ArrayList

- ການຈະໃຊ້ ArrayList ຈໍາເປັນຕ້ອງ  
`import java.util.ArrayList;`  
 ເພີ່ມກ່ອນໃຊ້ງານ
- ເວລາປະກາດເຮົາຈະໄຟ່ຕອງຮຽບຊາດໃຫ້ຈົດເຈັນຕິ່ງແຕ່ແຮກເພຣະ  
 ArrayList ສາມາດຮັບ/ຮັດຂາດໄດ້
- ArrayList ເກີນ Primitive Type  
 ໂດຍຕຽບໄຟ່ໄດ້ ຕ້ອງເກີນເບື້ນ  
 Wrapper Class ເທົ່ານີ້ ເຊິ່ງ  
`Integer, Double, String`

```
import java.util.ArrayList;
```

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

```
ArrayList<Integer> scores =  
new ArrayList<>();
```

### Wrapper Class

- Class ທີ່ກຳນົດທີ່ "ຫ່ອງຊຸມ" ຂ້ອງມູລພື້ນຖານ (Primitive Type) → ວັດຖາ (Object)
- ນີ້ໃຊ້ກັບ Collection (ເຊົ່ານ ArrayList) ທີ່ຮອງຮັບເຂົາພາວ Object

Primitive Type	int	char	double	boolean	byte	long	float
Wrapper Class	<code>Integer</code>	<code>Character</code>	<code>Double</code>	<code>Boolean</code>	<code>Byte</code>	<code>Long</code>	<code>Floating</code>

(ຈົ່ອສັງເກດ: ເປັນຢ່າງແລ້ວຈົ່ງຕັ້ງຕົວໃຫຍ່ ຂາເວົ້າ int ຈັນ char ທີ່ເປັນຢ່າງເປົ້າເຕີມ)



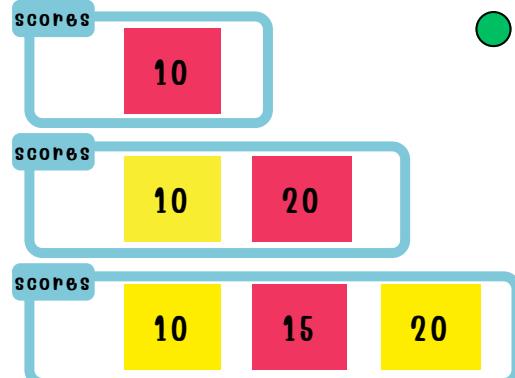
## การเพิ่มข้อมูลเข้า ArrayList

- ใช้ `.add(x)` เพื่อเพิ่มข้อมูล (`x`) ที่ต้องการไปที่ท้าย `ArrayList`
- หากต้องการจะดูว่าจะแทรก (`x`) ลงตำแหน่งไหนใน `ArrayList`  
สามารถทำได้โดยการระบุ `Index` ต่อไปด้วย `.add(index, x)`

```
scores.add(10);
// ต่อท้าย: [10]

scores.add(20);
// ต่อท้าย: [10, 20]

scores.add(1, 15);
// แทรกที่ index 1: [10, 15, 20]
```

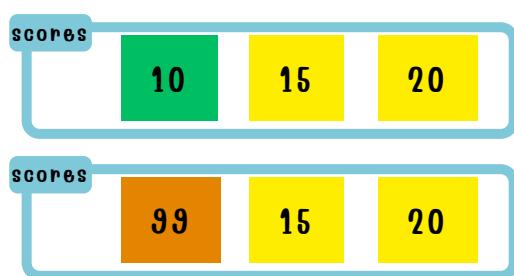


## การเข้าถึงและแก้ไขข้อมูลใน ArrayList

- ใช้ `.get(index)` ในการเข้าถึงค่าในตำแหน่งที่ต้องการ
- ใช้ `.set(index, x)` ในการเปลี่ยนค่าในตำแหน่งที่ต้องการ

```
int x = scores.get(0);
// ได้ค่า 10

scores.set(0, 99);
// เปลี่ยนค่าที่ index 0 เป็น 99
```



## การหาความยาวของ ArrayList

```
int size = scores.size();
```

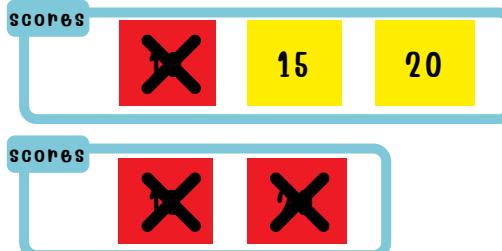
คำเตือน : รอบสีวงเล็บ () และงะ !!  
 เพราะเราจำลังใช้ Method



## การลบข้อมูลใน ArrayList

- ใช้ `.remove(index)` เพื่อลบข้อมูลในตำแหน่งที่ต้องการ
- ใช้ `.clear()` เพื่อล้างข้อมูลทั้งหมดออกจาก ArrayList

```
scores.remove(0);
// ลบตัวที่ index 0
scores.clear();
// ลบเกลี้ยงทั้ง List
```



## การวนลูปเข้าจิ้งษากิจใน ArrayList (Iteration)

```
for (int i = 0; i < scores.length; i++) {
    System.out.println("Index " + i + ": " + scores.get(i));
}
```



For loop

```
for (Integer score : scores) {
    System.out.println(score);
} //Wrapper Class ที่ใช้ต้องตรงกับของ scores ด้วยนะ!!
```



For each

## ARRAY VS ARRAYLIST (คุณสมบัติเบื้องต้น)

คุณสมบัติ	Array []	ArrayList
ขนาด	คงที่ (Fixed)	มีขนาดได้ (Dynamic)
ประเภทข้อมูล	ไม่ซ้ำกัน	เป็น Object
การคำนวณขนาด	<code>.length (property)</code>	<code>.size() (method)</code>



## ARRAY VS ARRAYLIST (ការខ្សោយចិត្តខ្លួន)

គុលសមប័ត្រ	Array []	ArrayList
ការទិន្នន័យខ្លួន (Get)	var = arr[index]	var = list.get(index)
ការແណ្ឌីឡើខ្លួន (Set)	arr[index] = value	list.set(index, value)
ការធើរឹងខ្លួន (Add)	កំណើនឯង (ឧបាទគម្ពី ពីរ ស្រាវជ្រាវភាពពិស់)	list.add(value)
ការសូបខ្លួន (Remove)	កំណើនឯង (ពីរគឺមិនគេគោល ដើម្បីបានលើខ្លួន)	list.remove(index)

### Common Utilities សំវារ៉ាប់ ArrayList

import java.util.Collections;

តើវាបាន method អ្វីខ្លះទីនឹងទៅ!!

- `.contains(x)` : ទរវតសុចរាប់ថា ArrayList មីខ្លួន(x) ឲ្យមិន
- `.indexOf(x)` : មាត្រាំងខ្លួន (Index) ខ្លួន (x) (ឲ្យផ្លូវគីឡូគាត់ -1)
- `.sort(arrayList)` : រៀនការណិតខ្លួនរាយការណ៍ខ្លួន
- `.toArray(new array)` : ផលូលបើន Array នៃរាយការណ៍



```
ArrayList<Integer> nums = new ArrayList<>(List.of(5, 1, 9));
boolean hasFive = nums.contains(5);
// true

int index = nums.indexOf(9);
// ឲ្យ 2 (តាមឲ្យផ្លូវគីឡូគាត់ -1)

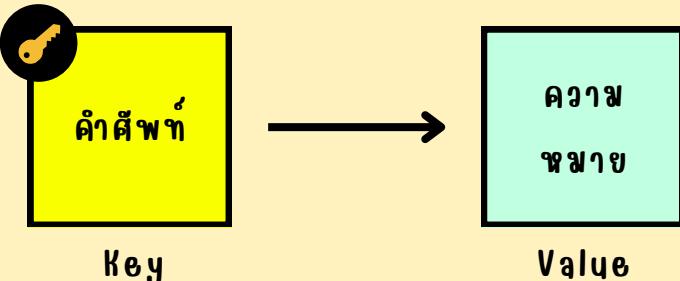
Collections.sort(nums);
// ឈូ: [1, 5, 9]

Integer[] arr = nums.toArray(new Integer[0]);
```



## -HashMap-

แล้วจ้าไม่ขาดใช้ Index ในการเข้าถึงข้อมูลล่ะ? วันนี้เราเปลี่ยนมาใช้ HashMap ดัน! ซึ่ง HashMap เป็น Data Structure ที่เก็บข้อมูลแบบ "คู่" (Key-Value Pair) เช่น "พจนานุกรม" ที่เราใช้ คำศัพท์ (Key) เพื่อหาความหมาย (Value)



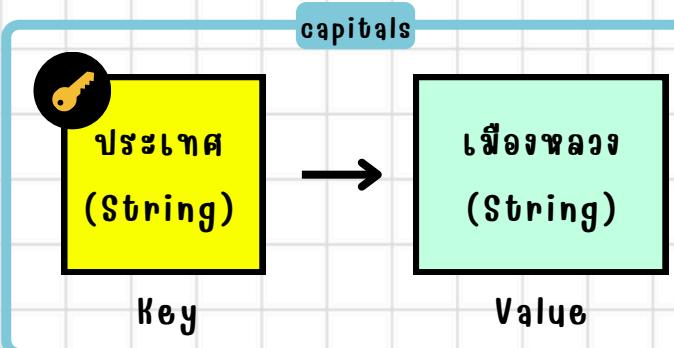
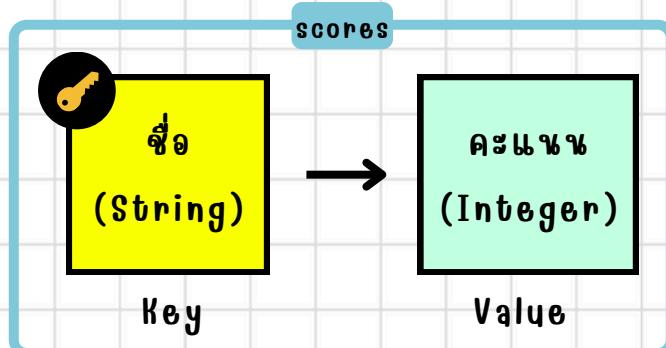
### การประกาศใช้ HashMap

- การจะใช้ HashMap จะเป็นต้อง `import java.util.HashMap;`
- `import java.util.HashMap;`  
เพื่อก่อนใช้งาน
- เวลาประกาศเราจะ **ไม่ต้องระบุขนาด** เช่นเดียวกับ `ArrayList`
- HashMap ใช้ **Primitive Type** ต้องเก็บเป็น `Wrapper Class` ทั้ง `Key` และ `Value` เท่านั้น (ถ้าปั้นไม่เข้าใจว่า `Wrapper Class` คืออะไร ข้อมูลในอ่าที่ `ArrayList` ก่อนนะ!!)

```
import java.util.HashMap;
```

```
HashMap<String, Integer>
scores = new HashMap<>();
```

```
HashMap<String, String>
capitals = new HashMap<>();
```



# Java Programming

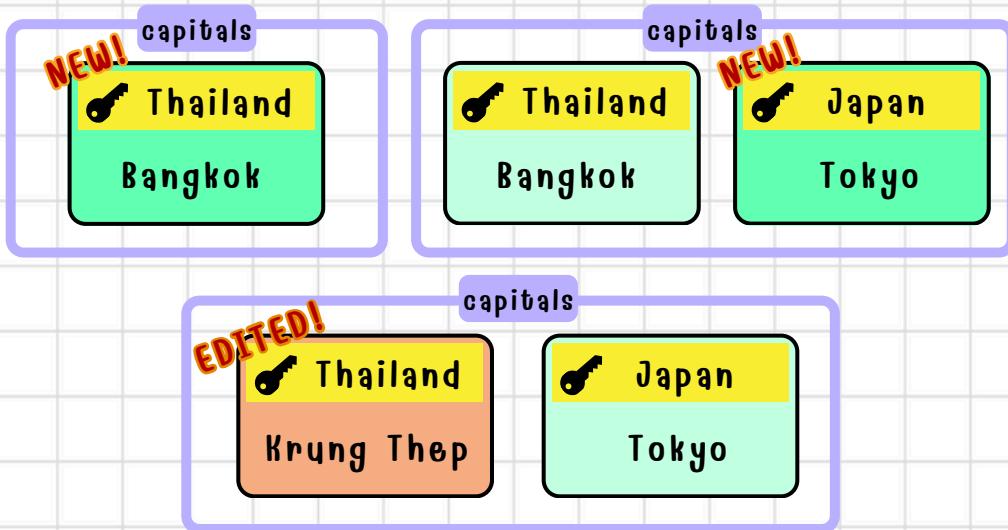


DATA TYPE: COLLECTIONS

การเพิ่มข้อมูลเข้า Hashmap / การแก้ไขข้อมูล

- ใช้ `.put(key, value)` ในที่ของการเพิ่ม/แก้ไขใน Hashmap

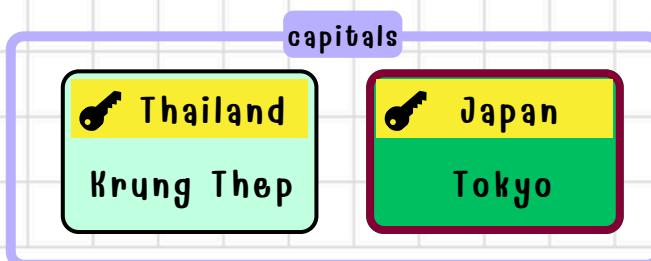
```
capitals.put("Thailand", "Bangkok");
// เพิ่มใหม่
capitals.put("Japan", "Tokyo");
// เพิ่มใหม่
capitals.put("Thailand", "Krung Thep");
// ขับค่าเดิมของ Key "Thailand"
```



การเข้าถึงข้อมูลใน Hashmap

- ใช้ `.get(key)` ในการเข้าถึงค่าของ key ที่ต้องการใน Hashmap (key ไม่ตรง = null)

```
String city = capitals.get("Japan");
// ได้ "Tokyo"
String unknown = capitals.get("Mars");
// จ้าไม่รู้ Key นี้ จะได้ null
```





## การหาความยาวของ HashMap

```
int count = capitals.size();
```

ข้อสังเกต : ใช้เหมือน ArrayList

## การลบข้อมูลใน HashMap

- ใช้ `.remove(key)` เพื่อลบข้อมูลโดยใช้ key เพื่อลบค่าที่ต้องการ
- ใช้ `.clear()` เพื่อล้างข้อมูลทั้งหมดออกจาก HashMap

```
capitals.remove("Japan");
```



```
// ลบค่าของตัวที่มี key เป็น Japan  
capitals.clear();  
// ลบเกลี้ยงทั้ง List
```



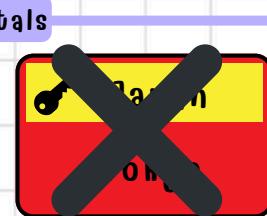
**Remove!**

capitals



Thailand  
Krung Thep

**Clear!**



## การวนลูปเข้าจิ้งหมายจิกใน HashMap (Iteration)

- HashMap ไม่สามารถเข้าจิ้งข้อมูลผ่าน Index จึงไม่สามารถใช้ For loop ปกติได้
- เริ่มต้นจากการใช้ `.keySet()` เพื่อตั้ง Collection ของ Keys ทุกตัวออกจากอุปกรณ์ จากนั้นใช้ for each วนเพื่อใช้ key ทีละตัวเพื่อเข้าจิ้งค่า Value และละ key ในแต่ละรอบ

```
for (String country : capitals.keySet()) {
```



```
    System.out.println(country + ":" + capitals.get(country));
```

} //Wrapper Class ที่ใช้ต้องทรงดับเบิล capitals ด้วยนะ!!



## การวนลูปเข้าจิ้งสาขาจิ๊กใน Hashmap (Iteration) (ต่อ)

- อีกหนึ่งลูป for each ที่มีประสิทธิภาพดีกว่าจ้าต้องการใช้ทั้ง Key และ Value พร้อมกัน
- วิธีนี้จะใช้ `.entrySet()` เพื่อดึง Collection ของ "คู่" Keys และ Values ทุกตัวออกมาก่อน จากนั้นใช้ for each วนเท็บค่า entry ทีละคู่ จากนั้น เข้าจิ๊ก key ด้วย `.getKey()` และ เข้าจิ๊ก value ด้วย `.getValue()`



```
for (Map.Entry<String, String> entry : capitals.entrySet()) {
    System.out.println(entry.getKey() + " -> " + entry.getValue());
}
```

## Common Utilities สำหรับ Hashmap #ไม่ต้อง Import อะไรเพิ่ม

ตัวอย่าง method ที่ทำให้ชีวิตง่ายขึ้นล้านเท่า!!

- `.containsKey(key)` : เช็คว่ามี Key นี้ไหม? (True/False)
- `.containsValue(value)` : เช็คว่ามี Value นี้ไหม? (True/False)
- `.getOrDefault(key, defaultValue)` : ถ้าค่าอุดมมา จ้าไม่รู้ใช่ใช่ค่า Default



```
boolean hasKey = capitals.containsKey("Thailand");
//ได้ค่า True เพราะมี Key Thailand อยู่จริง
boolean hasValue = capitals.containsValue("London");
//ได้ค่า False เพราะไม่มี Value London อยู่จริง
String capital = capitals.getOrDefault("Laos", "Unknown");
//ได้ค่า "Unknown" หา Key Laos ไม่เจอ (ไม่มี)
```



Collections ยังมีอีกเพียบ

จึงขอยกตัวอย่างแค่ที่

เรียนบ่อยๆ จะ

(ไม่ได้ยก. นะว้อว><)



- Chapter 7 -

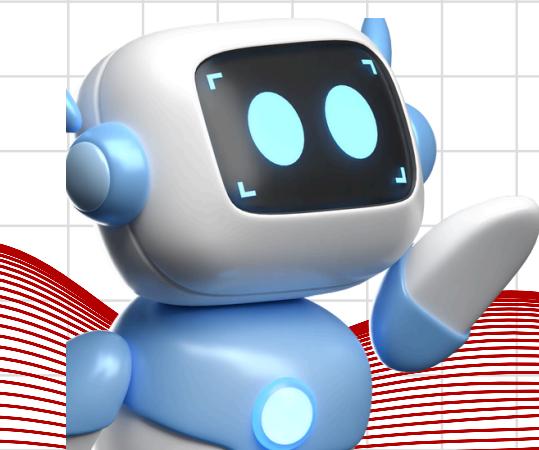
# ACCESS MODIFIERS

## ตัวระบุสิทธิ์การเข้าถึง

เกี่ยวกับบทนี้



“บทนี้จะแนะนำตัวระบุสิทธิ์การเข้าถึงในภาษา Java ที่จะมีผลต่อระดับการเข้าถึง/เรียกใช้ ตัวแปร คลาส เมธอด”





## Access Modifiers

Modifiers ใน Java สามารถแบ่งได้ 2 ประเภท

Access Modifiers - ควบคุณการเข้าถึง

- public:** ใช้ได้จากทุกที่
- protected:** ใช้ได้ในคลาสเดียวกัน, แพ็คเกจเดียวกัน และคลาสลูก
- private:** ใช้ได้ในคลาสเดียวกันเท่านั้น

Non-Access Modifiers - เป็นชนิดอื่นๆ เช่น static / final / synchronized

- static:** ใช้กับ Method สิ่ง / ตัวแปร สิ่ง
- final:** ค่าห้ามเมธอดที่เปลี่ยนไม่ได้ / คลาสที่สืบทอดไม่ได้
- abstract:** คลาสหรือเมธอดที่เป็นนามธรรม ต้องใช้คลาsslูกมาสืบทอด/ เรียนรู้
- synchronized:** ใช้กับเมธอด/บล็อกให้ทำงานทีละเฟรด ป้องกันปัญหาการเข้าถึงพร้อมกัน
- native:** เมธอดที่เขียนด้วยภาษาอื่น (เช่น C/C++) และเขียนกับ Java
- strictfp:** บังคับให้การคำนวณเลขทางคณิตใช้มาตรฐาน IEEE 754 เดียวกันทุกแพลตฟอร์ม

Static Method VS Static Variable

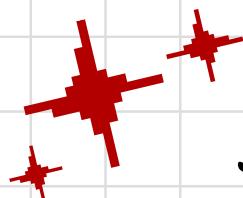
VS

เมธอดสิ่ง (Static Method)

- เป็นเมธอดที่เป็นของ คลาส ไม่ใช่ของออบเจกต์
- เรียกใช้ได้โดยตรงผ่านชื่อคลาส โดยไม่ต้องสร้างออบเจกต์

ตัวแปรสิ่ง (Static Variable)

- เป็นตัวแปรที่เป็นของ คลาส เช่น กัน
- มีค่าเดียวกันทุกออบเจกต์ ที่สร้างจากคลาสนั้น
- เช่น สำหรับเก็บข้อมูลที่ใช้ร่วมกัน เช่น ตัวนับ (counter)



- Chapter 8 -

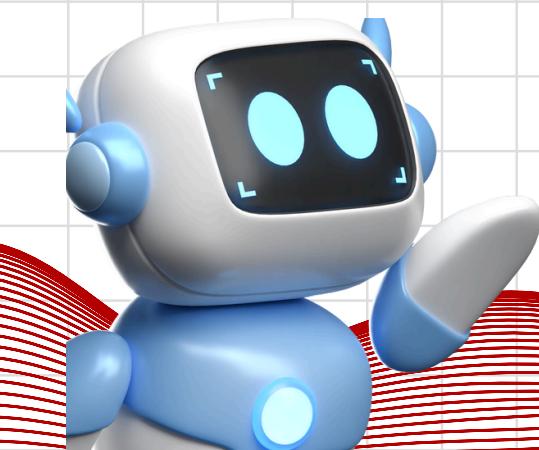
# OPERATORS

## ตัวดำเนินการ

เกี่ยวกับบทนี้



“บทนี้จะแนะนำตัวดำเนินการประเภทต่าง ๆ รวมถึง  
แสดงตัวอย่างประกอบ”





## -Operators-

### 1. Arithmetic Operators (คณิตศาสตร์)

สัญลักษณ์	ความหมาย	ตัวอย่าง ( $a=10, b=3$ )	ผลลัพธ์	ข้อควรระวัง
+	บวก	$a + b$	13	ใช้กับ String ได้ ("A" + "B" -> "AB")
-	ลบ	$a - b$	7	
*	คูณ	$a * b$	30	
/	หาร	$a / b$	3	Integer หารด้วย 0 ไม่ได้!
%	หารเอาเศษ (Modulo)	$a \% b$	1	

### 2. Logical Operators (ตรรกะ)

สัญลักษณ์	ความหมาย	เงื่อนไข	ตัวอย่าง
&&	AND (และ)	ถ้าเงื่อนไขทั้งสอง是真的	(คะแนน > 50) && (ไม่เรียนครับ)
	OR (หรือ)	มีอันใดอันหนึ่ง是真的	(มีบัตรสมาชิก)    (อายุเงินสด)
!	NOT (ไม่)	กลับค่า (真假->真假, 真假->真假)	isValid

# Java Programming



OPERATORS

## 2. (ទី២) Logical Operators (ទទរកនេ) (with Truth table)

X	Y	X && Y
T	T	T
T	F	F
F	T	F
F	F	F

X	Y	X    Y
T	T	T
T	F	T
F	T	T
F	F	F

X	!X
T	F
F	T

## 3. Relational Operators (បែវីមុខពើឱ្យ)

សញ្ញាណការណ៍	គរាមខ្សោយ	ពិនិត្យ (x=5)	ផលតឹម្បី
<code>==</code>	ទោរកីឃុំ	<code>x == 5</code>	true
<code>!=</code>	ឬផ្សោរទោរកីឃុំ	<code>x != 5</code>	false
<code>&gt;</code>	ខាតកវា	<code>x &gt; 3</code>	true
<code>&lt;</code>	ឯកចាកវា	<code>x &lt; 5</code>	false
<code>&gt;=</code>	ខាតកវាចិនទោរកីឃុំ	<code>x &gt;= 5</code>	true
<code>&lt;=</code>	ឯកចាកវាចិនទោរកីឃុំ	<code>x &lt;= 10</code>	true

# Java Programming



OPERATORS

## 4. Unary Operators (เพิ่ม/ลดค่า)

โค๊ด	ประเภท	ความหมาย	วิธีใช้
<b><math>++i</math></b>	Prefix	เพิ่มค่าก่อน แล้วค่อยใช้	นำก่อนใช้
<b><math>i++</math></b>	Postfix	เอาค่าไปใช้ก่อน แล้วค่อยนำ	ใช้ก่อนนำ

```
int a = 10;  
int b = ++a;  
// a เป็น 11 ก่อน, และ b ได้ 11 (a=11, b=11)
```



```
int x = 10;  
int y = x++;  
// y ได้ 10 ก่อน, และ x ค่อยเป็น 11 (x=11, y=10)
```

## 5. Unary Operators (ติดลบ(-)/ติดบวก(+))

โค๊ด	ความหมาย	ชนิด	โนํต
<b><math>+X</math></b>	ติดบวก	ระบุว่าค่านี้เป็น "ค่าวา"	แบบไม่มีผลอะไรมากมายค่าตัวเดียว เพราะปกติเวลาเขียน เช่น <code>int x = 5;</code> ซึ่งที่เป็นหน่วยก็แล้ว
<b><math>-X</math></b>	ติดลบ	ใช้ "กลับเครื่องหมาย" จากน้ำเป็น ลบ / ลบเป็นบวก	



## 6. Assignment Operators (จำนวนต่อค่าแบบข้อ)

แบบขอ	แบบเท็จ	ความหมาย
$x += 5$	$x = x + 5$	เพิ่มค่า x ไปอีก 5
$x -= 2$	$x = x - 2$	ลดค่า x ลง 2
$x *= 3$	$x = x * 3$	เอา x คูณ 3 และเก็บค่าเดิม
$x /= 2$	$x = x / 2$	เอา x หาร 2 และเก็บค่าเดิม
$x %= 2$	$x = x \% 2$	เอา x หารเอาเศษ 2 และเก็บค่าเดิม

## 7. Ternary Operator (จำนวนต่อค่าแบบมีเงื่อนไข)

- รูปข้อของ if-else เหมาะกับการจำนวนต่อค่าง่ายๆ
- โครงสร้าง: (เงื่อนไข) ? ค่าถ้าจริง : ค่าถ้าเท็จ;

```
// แบบปกติ
int score;
if (point > 50) {
    score = 100;
} else {
    score = 0;
}

// แบบ Ternary (บรรทัดเดียวจบ)
int score = (point > 50) ? 100 : 0;
```



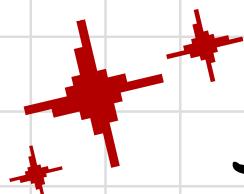


## 8. ลำดับความสำคัญของตัวดำเนินการ (Operator Precedence)

ลำดับ	กลุ่ม	ตัวดำเนินการ	รายละเอียด
1 (สูงสุด)	Parentheses	( )	ทำให้วงเล็บก่อนเสมอ (ใช้บังคับลำดับ)
2	Unary	++, --, !, + (Unary), - (Unary)	เพิ่ม/ลดค่าอน, ติดลบ
3	คูณ/หาร	*, /, %	คูณ, หาร, หารเอาเศษ
4	บวก/ลบ	+, -	บวก, ลบ
5	เปรียบเทียบ	<, <=, >, >=	มากกว่า, น้อยกว่า
6	ความเท่ากัน	==, !=	เท่ากัน, ไม่เท่ากัน
7	Logical AND	&&	และ (ใช้เชื่อมเงื่อนไข)
8	Logical OR		หรือ (ใช้เชื่อมเงื่อนไข)
9	Ternary	? :	กำหนดค่าแบบมีเงื่อนไข
10 (ต่ำสุด)	กำหนดค่า	=, +=, -=, *=, /=, %=	กำหนดค่า (ทำทีหลังสุดเสมอ)

เช่น  $x = 6 + (1+2) * 3$  เราจะเริ่มทำ  $(1+2)$  ก่อน เพราะ () มี Precedence สูงสุด  
 ได้เป็น  $x = 6 + 3 * 3$  ต่อมาจะทำ  $3 * 3$  ต่อ เพราะ \* มี Precedence สูงกว่า + ได้เป็น  $x = 6 + 9$   
 แล้วก็ทำ + ต่อได้เป็น  $x = 15$  และปิดท้ายด้วย = (กำหนดค่า)

ที่มี Precedence ต่ำสุด



- Chapter 9 -

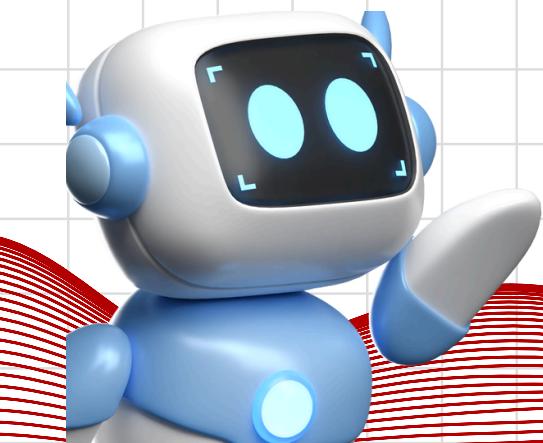
# VARIABLE TYPE

## ตัวแปรประเภทต่าง ๆ

เกี่ยวกับบทนี้



“บทนี้จะแนะนำตัวแปรประเภทต่าง ๆ และขอบเขตการ  
ใช้งานของมัน”





## -Variable type-



VariableEx.java X



```
class VariableEx {
    static int objCounter = 0;      static variable (ใช้ร่วมกันทุกอブเจค)
    int x = 1;                     instance variable (แต่ละอับเจคจะเป็นของตัวเอง)
}
```

```
VariableEx() {
```

```
    objCounter++;
}
```

```
public void addNumber(int y, int z) { Parameter ค่าที่รับเข้า method
```

```
    int a = 2; local variable (ใช้ได้แค่ในเขตเดียวกัน)
```

```
    System.out.println(a + x + y + z);
}
```

```
}
```

```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        VariableEx varEx = new VariableEx();
```

```
        varEx.addNumber(1, 2); Actual argument ค่าที่ส่งเข้า Method
```

```
        System.out.println("Instance variable: " + varEx.x);
```

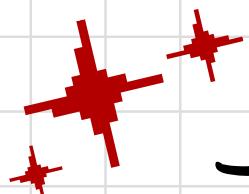
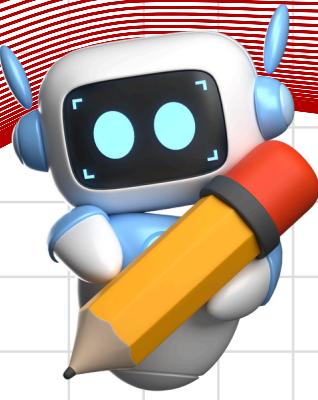
```
        System.out.println("Total object created: " + varEx.objCounter);
    }
}
```



## ประเภทของ Variable ใน Java



Variable	ขอบเขตการใช้งาน	คำอธิบาย
static variable	ใช้ได้ทั่วทั้งโปรแกรม โดยเรียกผ่านชื่อคลาส	<ul style="list-style-type: none"> <li>ถูกแชร์ระหว่างทุกอีบเจตของคลาสนั้น</li> <li>ถูกสร้างเมื่อโปรแกรมเริ่มทำงาน และถูกทำลายเมื่อโปรแกรมดับ</li> <li>สำคัญในการคำนวณที่ต้องร่วมกัน เช่น ตัวนับอีบเจต (object counter)</li> </ul>
instance variable	ใช้ได้ทั่วทั้งคลาส และต้องเข้าถึงผ่าน อีบเดียว	<ul style="list-style-type: none"> <li>ประกาศ ภายในคลาส แต่ถูกนัดเมื่อต้องการ</li> <li>ไม่ต้องกำหนดค่าเริ่มต้น (เช่น int = 0, boolean = false)</li> <li>ถูกสร้างเมื่ออีบเจตถูกสร้าง และถูกทำลายเมื่ออีบเจตถูกทำลาย</li> <li>แต่ละอีบเจตจะมี instance variable เป็นของตัวเอง</li> <li>สามารถใช้ access modifier (public, private, protected) ได้</li> </ul>
local variable	ใช้ได้เฉพาะ ภายในเขตอพาร์ทีมีลักษณะที่ประกาศเท่านั้น	<ul style="list-style-type: none"> <li>ประกาศ ภายในเขตอพาร์ทีมีลักษณะที่ (เช่น for, if, while)</li> <li>ต้องกำหนดค่าเริ่มต้นก่อนใช้งาน (ถ้าไม่กำหนดจะเกิด compile error)</li> <li>ถูกสร้างเมื่อเข้าสู่/ออกจากเริ่มทำงาน และถูกทำลายเมื่อเข้าสู่/ออกจากการทำงาน</li> <li>ไม่สามารถใช้ static หรือ access modifier (public, private, protected) ได้</li> </ul>
Parameter	ใช้ได้เฉพาะ ภายในเขตอพาร์ทีมีลักษณะที่ประกาศ	<ul style="list-style-type: none"> <li>ประกาศ 之内จะเลือกของเมื่อต้องการ เพื่อรับค่าจากผู้เรียกใช้</li> <li>ซึ่งเป็น local variable ประจำชั้นเรียน</li> <li>ถูกสร้างเมื่อเข้าสู่/ออกจากเริ่มทำงาน และถูกทำลายเมื่อเข้าสู่/ออกจากงาน</li> </ul>
Actual Argument	ใช้ เอกสารตอนเรียกเมื่อต้องการ และค่าของถูกส่งไปยัง parameter	<ul style="list-style-type: none"> <li>คือ ค่าที่ส่งเข้าไปในเมื่อต้องการ เพื่อทำการเรียกใช้</li> <li>ต้องตรงกับ ประจำชั้นเรียนของ parameter</li> </ul>



- Chapter 10 -

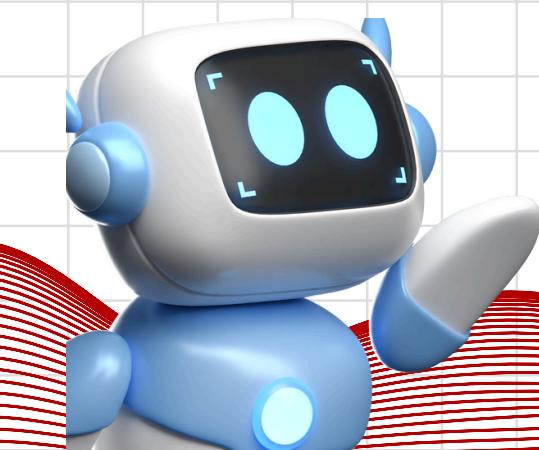
# ENCAPSULATION

## การห่อหุ้มข้อมูล

เกี่ยวกับบทนี้



“บทนี้จะแนะนำเนื้อหาที่ทำงานกับ VARIABLE ที่เป็น PRIVATE ใน OOP ซึ่งเกี่ยวของโดยตรงกับสาหลักเรื่อง ENCAPSULATION”





## -Getter & Setter Method-

ຈາເຮາດອງພະຍານາມແກ້ໄຂ / ເຊົ້າຈີ່ງຄ່າຕົວແປຣທີ່ເປັນ private ດະເດືອນໄຮ້ຈິ່ນ



myPet.java X

```
class myPet {
    private String petName = "Prayuth";
}

public class Main {
    public static void main(String[] args) {
        myPet dog = new myPet();
        System.out.println("petName is: " + dog.petName);
    }
}
```



ແສດງວ່າຕົວແປຣ private ກີ່ເຊົ້າຈີ່ງຈະຄລາສໜ້າງນອກໄໝໄລ້ເລີ່ມສີ ແລ້ວຈາກພິບພໍ່ສ່ອນອ່ານຫາ  
ປະບຸກົດ ຕ້ອງກຳໄໝງງງງງງງງ...?

A). ປຶ້ມພາເດີດທີ່ private ໄໃຈໜີ້!  
**private → public ອຸບ!!!**  
ແຕ່ເປັນວິທີທີ່ຄຸງຮິງ ມ ຊຮອ?

```
class myPet {
    public String petName = "Prayuth";
}
```

B). ຈົດຈັດຂອງຕົວແປຣ private  
ຄືອ “ເຊົ້າຈີ່ງໄດ້ໃໝ່ຄລາສເທົ່ານີ້”  
ຮັ້ນເຮົາກີ່ ສ່າງ method ໃນຄລາສນີ້  
ແລ້ວເຮືອງ method ພ່ານ Object  
ໃນຄລາສ main ສີ

```
class myPet {
    private String petName = "Prayuth";
    public String getPetName(){....}
    public void setPetName(String name){
        ....
    }
}
```



Main.java

X



```
import java.util.Scanner;  
class myPet {  
    private String petName = "Prayuth";
```

public String getPetName() { ประจักษ์ getter & setter method

```
    return petName;  
}
```

```
public void setPetName(String petName) {  
    this.petName = petName;  
}
```

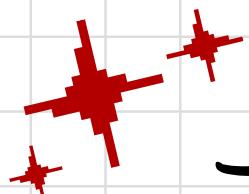
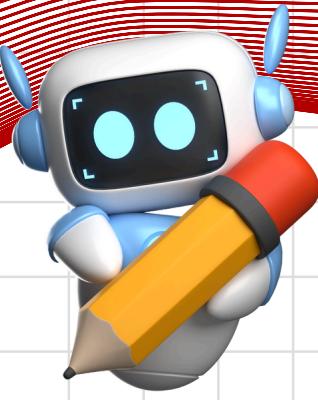
}

public class Main { เรียนรู้ getter & setter method พาก Object

```
    public static void main(String[] args) {  
        myPet dog = new myPet();  
        System.out.println("petName is: " + dog.getPetName());  
        Scanner scan = new Scanner(System.in);  
        String newPetName = scan.nextLine();  
        dog.setPetName(newPetName);  
        System.out.println("NEW petName is: " + dog.getPetName());  
    }  
}
```

เขียนโปรแกรมแบบนี้ทั้งตีและปอกดลัด!!!





- Chapter 11 -

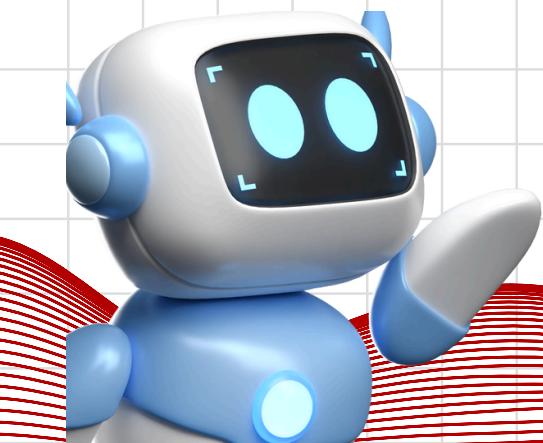
# INHERITANCE

การสืบทอดคุณสมบัติ  
ระหว่างคลาส

เกี่ยวกับบทนี้



“บทนี้จะแนะนำการสืบทอดตัวแปรและเมธอดจาก  
คลาสแม่ไปยังคลาสลูก”





## -Inheritance-

### Class แม่

- คุณสมบัติ (Attribute) และพฤติกรรม (Method) ที่อยู่ในคลาสแม่ สามารถถูกアクセスไปสู่คลาสลูกได้ (ยกเว้นตัวที่มี Access Modifier เป็น private)

### Class ลูก

- คลาสลูกต้องประกาศใช้ Extends คลาสแม่ เช่น (มีแม่ได้สูงสุด 1 คลาส)
- Object ในคลาสลูกสามารถเข้าถึงตัวแปร / เรียกใช้ Method ที่อยู่ในคลาสแม่ได้ (ยกเว้นตัวที่มี Access Modifier เป็น private ในคลาสแม่)
- Constructor ในคลาสลูกต้องส่งต่อให้คลาสแม่ด้วยการเรียก Constructor คลาสแม่ผ่าน super()

### Class แม่



### Class Vehicle

String vColor  
honk()

### Class ลูก



### Class Car

accelerate(int);

### สร้าง Object

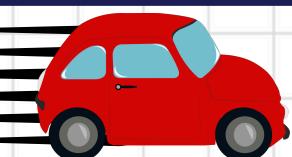
Object carA  
color = "red";

### Object เข้าถึงตัวแปรคลาสแม่



String color =  
carA.vColor;

### Object เรียกใช้ Method



carA.accelerate(45);



carA.honk();

# Java Programming



CONCEPT: INHERITANCE

Main.java

X



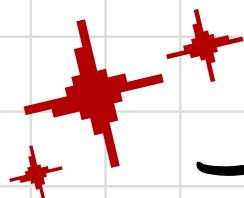
```
class Vehicle {  
    public String vColor; ตัวแปรของคลาสแม่  
  
    public Vehicle(String color) {  
        vColor = color; Constructor ของคลาสแม่  
    }  
  
    public void honk() {  
        System.out.println("Beep! Beep!"); method honk ของคลาสแม่  
    }  
}
```

คลาสแม่

```
class Car extends Vehicle { กำหนดให้คลาสลูก繼承 คลาสแม่  
    public Car(String color) {  
        super(color); Constructor ของคลาสลูก  
#ใช้ super() เพื่อเรียก Constructor ของ  
แม่ พร้อมส่ง color เป็น Argument  
    }  
  
    public void accelerate(int speed) {  
        System.out.println("Accelerating to " + speed); method accelerate  
ของคลาสลูก  
    }  
}
```

คลาสลูก

```
public class Main {  
  
    public static void main(String[] args) {  
        Car carA = new Car("Red"); สร้าง Object ของคลาสลูก  
        System.out.println("Color: " + carA.vColor); เข้าถึงตัวแปรของคลาสแม่ได้  
        carA.honk(); เรียกใช้ Method ของคลาสแม่ได้  
        carA.accelerate(45); เรียกใช้ method คลาสลูกเอง  
    }  
}
```



- Chapter 12 -

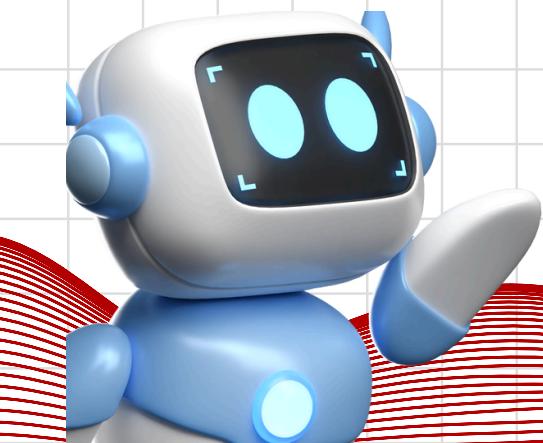
# ABSTRACTION

## ความเป็นนามธรรม

เกี่ยวกับบทนี้



“บทนี้จะแนะนำเกี่ยวกับนามธรรม ซึ่งจะเกี่ยวข้องกับ  
INTERFACE หรือ ABSTRACT CLASS ในภาษา JAVA”





## Library

Library เปรียบเสมือนกล่องเครื่องมือวิเศษ เราเพียงแค่ "ชื่อ Class" และ "ชื่อ Method" (API) ที่เข้าเป็นใช้ (มาจากเอกสารของเจ้าของ Library) เพียงเท่านั้นก็สามารถใช้งานตามความต้องการของเราได้เลย โดยไม่ต้องรู้ว่า Method จ้างในทำงานอย่างไร

### ตัวอย่างการใช้ Library ใน Abstraction

- Import Library ที่ต้องการ (ในตัวอย่างนี้จะใช้ Math ซึ่งมีอยู่แล้วในแพ็คเกจ `java.lang` จึงไม่ต้อง Import เพิ่ม)
- เขียนโปรแกรมของเราให้เรียกใช้ Public Method ที่เข้าเ泰สท์มไว้ใช้ (หาดูได้จากเอกสารของ Library ที่เราต้องการใช้ เขาจะมีข้อมูลว่า ต้อง Import 什么 นี้ Method อะไรมาก่อนและรับ Parameter อย่างไรบ้าง) ในกรณีนี้เราจะรู้ว่าคลาส Math มี Method ชื่อ `sqrt()` ที่ทำหน้าที่หาค่ารากของตัวเลขที่ส่งเข้าไป
- หน้าที่เดียวของเราคือ เรียกใช้ Method `sqrt` ผ่านคลาส `Math` ที่เรา Import เข้ามา แค่นี้ เรา ก็จะได้ค่าตอบที่ต้องการโดยไม่ต้องรู้เลขว่าภายใน Method `sqrt` ใส่อะไรเป็นขึ้นไป

```
public class Main {  
    public static void main(String[] args) {  
        double number = 25.0;  
        double result = Math.sqrt(number);  
        System.out.println("Result: " + result);  
    }  
}
```



\*ฉะนั้นเราไม่ใช่เรื่อง Abstraction เข้ามาช่วย เราจะต้องเขียน Method `sqrt` เอง!!

เอกสาร Math Class: <https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/Math.html>



## Interface

Interface คือหัวใจหลักของ Abstraction ในภาษา Java เลย!! โดยมีนี่จะเป็นการที่เราสร้างข้อตกลงไว้ว่าจะใช้ชื่อนำมาระไรได้บ้าง และเรื่องทำอย่างไร คลาสต้องเอาไปเขียนด้วยตัวเอง!! #Interface จำเป็นต่อคลาสจะต้องมี Method อะไรบ้าง และจะเป็น Method ที่ไม่ใช้ได้ใน (Abstract Method) และคลาสที่ประกาศใช้ Implement Interface นี้จะต้องเขียนเอาเองว่าจะใช้ชื่อนำมาระไร

### การประกาศ Interface

- ประกาศตัวบัญญัติ **Interface** และ **Class**
- ตัวแปรใน **Interface** จะเป็น **Constants (public static final)** เช่น (แม้จะไม่เขียนขอก)
   
(**public static final int MAX\_COPIES = 100;**)
- Abstract Method:** จัดตั้งชื่อ ไม่ใช่ชื่อปกติ {} และจบด้วย ; (บริษัท มีชื่อต้องเขียนว่า "public abstract void print();") และ Java จะตีความ **public abstract** ให้เป็นอัตโนมัติ)

```
public interface Printable {
    int MAX_COPIES = 100;
    void print();
}
```

### -- ข้อตกลง --

คลาสใดก็ตามที่จะ Implement Interface **Printable** จะต้องมี **Method print()** และต้องเป็น **public** ในคลาสนั้น ๆ ด้วยเช่นกัน



**Class** ที่ Implement จะต้องมี **Method** ตาม **Interface** ให้ครบถ้วน!!

ถ้า **Interface** มี 5 **Method** คลาสที่นำไปใช้ (**implements**) ต้องเขียนโค้ดให้ครบถ้วน 5 **Method** ข้างขาดแคลนแต่ลืมเดียว

จะขาดไปแม้แต่ตัวเดียว ก็จะแสดงข้อผิดพลาด Error ทันที (Compile ไม่ผ่าน)

### -- คำเตือน --





## การนำไปใช้ใน Class

- ประกาศใช้ `class .. implements interface` ที่ต้องการ
- สร้าง `method` ตามที่ได้ตกลงกันไว้ใน `Interface` (อย่าลืม ว่า `method` ต้องเป็น `public!!`) และระบุตัวชี้ว่าจะใช้ฟังก์ชันทำอะไร
- `@Override` คือ `Annotation` (ป้ายกำกับ) เพื่อบอก `Compiler` ว่า "เมื่อตอนนี้ ตั้งใจจะเขียนที่นี่ ของ `Interface` (`Abstract Method`) ประ祐ชน์คือ เอาไว้เขียนค่าว่าพิมพ์ซึ่งถูกเบ็งๆ ให้ ถ้าไม่ได้ แล้วเราพิมพ์ซึ่งผิด คอมพิวเตอร์จะหักว่า เราสร้าง `Method` ให้ไม่ลง!!! (Note: เราจะเจอตัวนี้อีกครั้งในบทถัดไป (`Polymorphism`) ตอนที่เราต้องการสร้าง `Method` ซึ่งเดี๋ยวนี้แต่ทำงานคนละอย่างที่นี่ method ของคลาสแม้')

```
public class Printer implements Printable {
    @Override
    public void print() {
        System.out.println("Printing document");
    }
}
```



## ประโยชน์ของ Interface

- ทำ `Multiple Inheritance` ได้: ช่วยให้ `Class` นี้รับความสามารถได้หลายอย่าง (แยกข้อจำกัดที่ `extends` ได้แค่ 1 `Class`)
- สร้างมาตรฐาน: เป็น "ข้อตกลง" บังคับให้ทุก `Class` ตั้งชื่อ `Method` ตรงกันเบ็ง (ทำงานเป็นทีมงาน)
- ลดการซ้ำเติบ: ทำให้โค้ดมีความซ้ำซ้อนน้อยลง เมื่อเปลี่ยนแปลงไปใน `(Class)` ได้ง่ายโดยไม่ต้องรีอ่อนบนใหม่

# Java Programming



CONCEPT: ABSTRACTION

## ตัวอย่างใช้งาน Interface

Main.java

X



```
interface Printable {  
    void print();  
}
```

Interface

```
class Printer implements Printable {  
    @Override  
    public void print() {  
        System.out.println("Printing document... Success!");  
    }  
}
```

ไฟล์ class Printer implements  
Interface Printable

สร้าง method print() ตามที่สืบใน Interface และใส่ไว้ใน (クラス  
พิมพ์) ใช้สีน้ำเงิน!! อย่างเช่น PUBLIC!!!!!!!

```
public class Main {  
    public static void main(String[] args) {  
        Printable myPrinter = new Printer();  
        myPrinter.print();  
    }  
}
```



## -Abstract Class-

Abstract Class เป็นอีกหนึ่งเรื่องสำคัญของการ Abstraction ขั้นคือคลาสที่ขึ้นสร้างไม่เสร็จ บางส่วนเขียนไปแล้ว บางส่วนให้คลาสลูกเอาไปเขียนต่อเอง เป็นคล้าย ๆ Interface ที่กำหนดวิธีการทำงานไปบังบາบังส่วน ไม่ใช่ “ไม่กำหนด” อะไรเลย เหมาะกับกรณีที่เราต้องการสร้าง Object จากคลาส 2 คลาส ที่แตกต่างกัน และอยากให้การทำงานของย่าง เชื่อมกันเป็น! แต่ยังอย่างให้ต่างกันตามประเภทของคลาส

### การประดิษฐ์ Abstract Class

- เพิ่ง **abstract** ที่พื้นที่ Class
- ตัวแปรใน Abstract Class ไม่จำเป็นต้องเป็น Constant!
- Concrete Method:** เป็น method ที่เขียนมาเรียบร้อยแล้ว #เป็นสิ่งที่ต้องการให้คลาสลูกทำแบบเดียวกันเป็น! เช่น สัตว์ทุกตัวจะนอนหลับ เชื่อมกัน (เขียนให้ใน Method ไปเลย)
- Abstract Method:** จัดซื้อ ไม่มีปีกๆ {}, จบด้วย ; และต้องเพิ่ง **abstract** ไว้ด้านหน้า method ด้วย #เป็นสิ่งที่คลาสลูกต้องการให้คลาสลูกทำ แต่ไม่จำเป็นต้องเชื่อมกัน เช่น สัตว์แต่ละตัวร้องไม่เชื่อมกัน (เว้นไว้ให้ลูกเขียน)

```
abstract class Animal {
    String name;
    public void sleep() {
        System.out.println("Zzz");
    }
    abstract void makeSound();
}
```

```
Animal ant = new Animal();
```

-- คำเตือน --

ข้ามสร้าง Object จาก Abstract Class เด็ดขาด!! ต้องสร้างจาก Class ลูกเท่านั้น!!





## การสืบทอดไปสร้าง Class ลูก

- ใช้ `extends` (ทำให้มีอันเรื่อง Inheritance) เพราะ Animal คือ Class (แค่เป็นแบบ `abstract`)
- สร้าง method ตาม `Abstract method` ใน `Abstract Class` ไม่ต้องสร้าง `Concrete Method` เพิ่ม (รอบนี้ไม่จำเป็นต้องใส่ Access modifier แล้ว!!) และระบุด้วยว่าจะใช้ชื่อที่อะไร
- @Override คือ Annotation (ป้ายกำกับ) เพื่อบอก Compiler ว่า "เมธอดนี้ ตั้งใจจะเขียนทับ ของ Interface (Abstract Method) ประ祐ชน์คือ เอาไว้เขียนว่า พิมพ์ชื่อถูกไปใช่ จ้าไม่ใช่ แล้วเราพิมพ์ชื่อผิด คอมพิวเตอร์จะนึกว่าเราสร้าง Method ใช่แล้ว!!! (Note: เราจะเห็นตัวนี้หากรังสีบทลักษณ์ใน Java ไป (Polymorphism) ตอนที่เราต้องการสร้าง Method ซึ่งเดียวจัดแต่ทำงานคนละอย่างทัน method ของคลาสแม้)

```
public class Dog extends Animal {  
    @Override  
    void makeSound() {  
        System.out.println("Woof! Woof!");  
    }  
}
```



## ประ祐ชน์ของ Abstract Class

- คลาสเดียว: สามารถเขียนโค้ดส่วนที่เหลือรักษาไว้ได้เลย (Concrete Method) ลูกเอาไปใช้ได้ทันทีโดยไม่ต้องเขียนใหม่
- บังคับโครงสร้าง: ใช้ `abstract method` บังคับให้ลูกต้องมีฟังก์ชันนี้แน่นอน (เหมือน Interface) และซึ่งบอกให้รักษาโค้ดส่วนกลางได้ด้วย
- อัตราจ่าย: ทำให้คลาสเป็นระเบียบและอัตราจ่ายในส่วนของ Superclass เดียวทัน

# Java Programming



CONCEPT: ABSTRACTION

## ពួកខ្សោយថាគារ Abstract Class

Main.java

X



```
abstract class Animal {  
    public void sleep() {  
        System.out.println("Zzz... Sleeping");  
    }  
  
    abstract void makeSound();  
}
```

គម្រោង

Concrete Method

#នូវ ឬ ការងារដែលរាយ

Abstract Method

#នូវ ឬ បញ្ជីការណ៍ដែលត្រូវ

class Dog extends Animal {

គម្រោង (1)

```
@Override  
void makeSound() {  
    System.out.println("Woof! Woof!");  
}
```

Class នូវបានគេបង្កើតឡើងទៅផ្ទាល់ជាដំឡើង  
ដូច #ម្រាម៉ាម៉ា Woof! Woof!

class Cat extends Animal {

គម្រោង (2)

```
@Override  
void makeSound() {  
    System.out.println("Meow~");  
}
```

Class នូវបានគេបង្កើតឡើងទៅផ្ទាល់ជាដំឡើង  
ដូច #មេវវិចិថុន Meow~

public class Main {

public static void main(String[] args) {

```
    Animal myDog = new Dog();  
    Animal myCat = new Cat();
```

សរាប់ Object ទៅបានគម្រោង ឱ្យបានការងារ  
ត្រូវបានបង្កើតឡើង

```
    myDog.sleep(); //myCat.sleep();
```

ការងារដែលរាយ

```
    myDog.makeSound(); //myCat.makeSound();
```

សេចក្តីថ្លែងការណ៍ដែលត្រូវបានការងារ

}

}



- Chapter 13 -

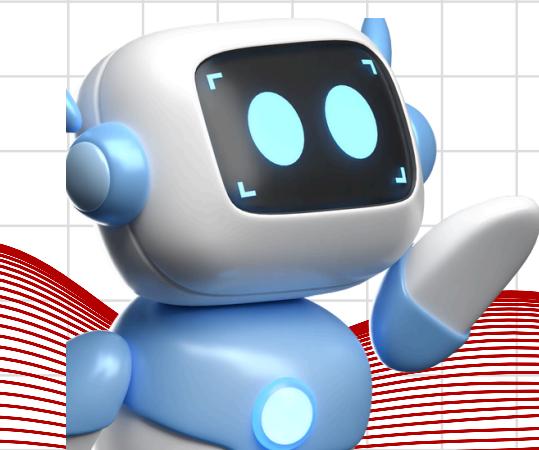
# POLYMORPHISM

## การผ้องรูป/มีหลายรูปแบบ

เกี่ยวกับบทนี้



“บทนี้จะแนะนำเกี่ยวกับการผ้องรูปในภาษา JAVA ซึ่ง  
จะเกี่ยวข้องกับ OVERLOADING และ OVERRIDING”





## -Overloading-

Overloading คือการที่เราใช้ชื่อเมธ็อดกันและอยู่ใน Class เดียวกัน (แยกกันด้วย "จำนวน/ชนิดของตัวแปร") ซึ่งเป็นส่วนหนึ่งของคอนเซปต์ จาร์พ้องรูป (Polymorphism) ในภาษา Java #ใช้ชื่อเมธ็อดกันแต่มีรายละเอียดที่แตกต่างกัน

### Method Overloading

- สร้าง Method ซึ่งมีชื่อเดียวกันในคลาส เดียวกันแต่ต้องมี Parameter ต่างกัน (จำนวนไม่เท่ากัน หรือ ชนิดตัวแปร ต่างกัน)
- จากโคลาจด้านขวาจะเห็นว่าเรามี Method ชื่อ add เมธ็อดกันตั้ง 3 ตัว ในคลาสเดียวกัน
  - add ตัวที่ 1: รับ int 2 ตัว
  - add ตัวที่ 2: รับ int 3 ตัว ซึ่งจำนวนต่างจาก Method ตัวที่ 1
  - add ตัวที่ 3: รับ double 2 ตัว ซึ่งประเภทต่างจาก Method ตัวที่ 1

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }

    double add(double a, double b) {
        return a + b;
    }
}
```

### ประโยชน์ของ Method Overloading

- คงใช้คำชี้อเดียวพอง (add) ไม่ต้องจำชื่อ เมธ็อด (addTwoInt, addThreeInt, addDouble) ส่ง Argument ชิ้นๆ
- คงเขียนกีฬาขยันะ ไม่ต้องคิดชื่อให้ซ้ำ เช่นแบบ :

```
int add(int a, int b) {...}
double add(int a, int b) {...}
```

-- คำเตือน --
 ห้ามเปลี่ยนแคร์ Return Type!!
 แล้วบอกว่าเป็น Overloading เพราะจะเกิด Error เนื่องจาก Compiler แยกไม่ออก

# Java Programming



CONCEPT: POLYMORPHISM

## ตัวอย่างใช้งาน Overloading Method

Main.java

X



```
class Calculator {
```

```
    int add(int a, int b) {  
        return a + b;  
    }
```

add #1

บวกเลขจำนวนเต็ม 2 ตัว

```
    int add(int a, int b, int c) {  
        return a + b + c;  
    }
```

add #2

บวกเลขจำนวนเต็ม 3 ตัว

```
    double add(double a, double b) {  
        return a + b;  
    }
```

add #3

บวกหาผลนิมมข 2 ตัว

```
}
```

```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        Calculator calc = new Calculator();
```

```
        System.out.println(calc.add(10, 20));
```

```
        System.out.println(calc.add(10, 20, 30));
```

```
        System.out.println(calc.add(5.5, 2.5));
```

```
}
```

```
}
```

จงอ ฉลาก! สิ่งจะเรียก Method ที่  
ตรงกับค่าที่เราส่งไปใช้เอง



## -Overriding-

Overriding คือการที่เราใช้ชื่อ Method และพารามิเตอร์เดียวกันในคลาสแม่-ลูก และต้องการเขียนใหม่ Method คลาสลูกทำงานทับคลาสแม่ ซึ่งเรื่องนี้เราได้เจอนไปก่อนหน้านี้แล้วในหัวข้อ Interface/Abstract Class บทก่อนหน้าที่เราได้เขียนใหม่ Method ในคลาสลูก ทับ Abstract Method ในคลาสแม่ที่มีชื่อและพารามิเตอร์เดียวกันเบื้องต้น!!

### Method Overriding

- สร้าง Class แม่และลูก จากนั้นสร้าง Method ที่มีชื่อเดียวกันและพารามิเตอร์เดียวกันในทั้งคลาสแม่และลูก
- ใน Method ของคลาสลูกต้องเขียน @Override ป้ายกำกับ (Annotation) เพื่อระบุว่า Method นี้จะเขียนทับของคลาสแม่
- จากโคลด์ดาวน์ว่าเด็นไกด์ว่าคลาส Bird ที่เป็นลูกของคลาส Animal มีการเขียนทับ Method move ของแม่ ดังนั้นเมื่อเรียก move ด้วย Object ของคลาส Bird ก็จะพิมพ์ flying ออกหน้าจอ

```
class Animal {
    void move() {
        System.out.println("moving");
    }
}

class Bird extends Animal {
    @Override
    void move() {
        System.out.println("flying");
    }
}
```



### ประโยชน์ของ Method Overloading

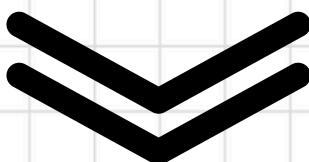
- กำหนดการทำงานเฉพาะตัวสำหรับคลาสลูก: ช่วยให้คลาสลูกมีพฤติกรรมที่เหมาะสมกับตัวเองโดยไม่ต้องจำใจใช้ชื่อของคลาสแม่ตลอดเวลา
- สั่งงานผ่านตัวแปรแม่ แต่ได้ผลลัพธ์แบบลูก: ช่วยให้เขียนโค้ดจัดการ Object หลากหลายชนิดได้ในคำสั่งเดียว (เช่น วนลูปสั่งสั่งทุกตัวให้ขึ้น โดยไม่ต้องมาหั่งเชื่อมว่าตัวไหนเป็น哪 ตัวไหนเป็น哪 ปลา)



## Method Overloading VS Method Overriding

พื้นที่	Overloading	Overriding
สถานที่	เกิดใน Class เดียวเดียว	เกิดระหว่าง แม่ - ลูก
อุปประสงค์	เพิ่มทางเลือก (รับค่าได้หลายแบบ)	เปลี่ยนการทำงานเดิม (ของแม่)
Parameter	ต้องต่างกัน	ต้องเชื่อมโยง
Return Type	เป็นอะไรก็ได้	ต้องเชื่อมโยง

จบแล้วววว ตัวอย่างไปรассмотримๆ กดไป



# Java Programming



CONCEPT: POLYMORPHISM

## ตัวอย่างใช้งาน Overriding Method

Main.java

X

```
class Animal {  
    void move() {  
        System.out.println("Animal is moving...");  
    }  
}
```

คลาสแม่

Method move  
ของคลาสแม่

```
class Bird extends Animal {  
  
    @Override  
    void move() {  
        System.out.println("Bird is flying!");  
    }  
}
```

คลาสลูก (1)

Method move ของลูก  
เขียนทับ (Override)  
ของคลาสแม่

```
class Fish extends Animal {  
  
    @Override  
    void move() {  
        System.out.println("Fish is swimming!");  
    }  
}
```

คลาสลูก (2)

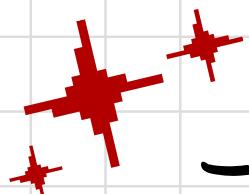
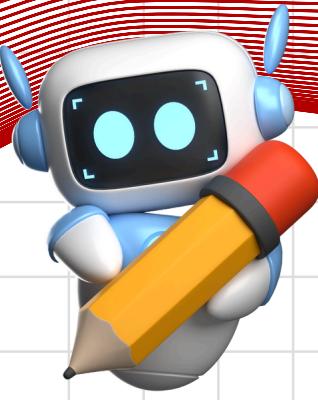
Method move ของ  
ลูกเขียนทับ  
(Override) ของ  
คลาสแม่

```
public class Main {  
  
    public static void main(String[] args) {  
        Animal myPet = new Bird();  
        myPet.move();  
        myPet = new Fish();  
        myPet.move();  
    }  
}
```

อีกหนึ่งความเจ๋งของ Polymorphism  
สังเกตว่าเราประมวลผลแบบปรับเปลี่ยน  
Animal (แม่) และ Object เป็น  
Bird (ลูก) หลังจากนั้นเปลี่ยนไปสู่ใน  
ตัวแปรเดิม เป็น Fish ได้เลย!!

}

BY CHAQUI.RL



- Chapter 14 -

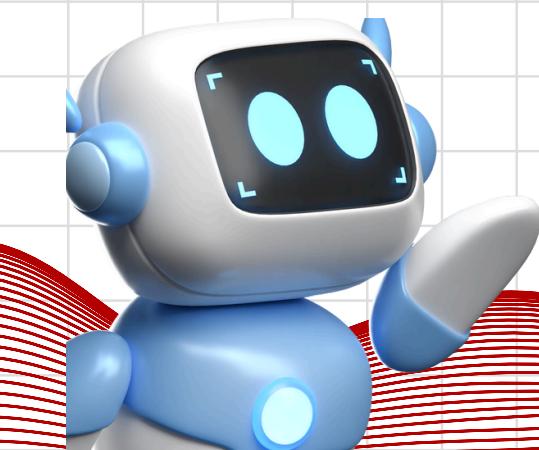
# ERROR HANDLING

## การจัดการกับข้อผิดพลาด ในโปรแกรม

เกี่ยวกับบทนี้



“บทนี้จะแนะนำเกี่ยวกับประเภทของข้อผิดพลาดต่าง ๆ  
และวิธีการจัดการอย่างเช่น TRY CATCH EXCEPTION”





## -Error & Exception-

รู้จักมือไร... ใน ปุ่งๆ คำว่า "Error" กับ "Exception" ไม่ใช่มีชื่อกันนะ!!

- **Error:** ปัญหาที่ระบบต้องการแก้ไขโดยโปรแกรมเมอร์ เช่น การแบ่งด้วยศูนย์ หารด้วยศูนย์ เป็นต้น
- **Exception:** ปัญหาที่เกิดจากโค้ดหรือปัจจัยภายนอก เช่น การอ่านไฟล์ไม่สำเร็จ ค่าตัวแปรไม่ถูกต้อง เป็นต้น

ใน ปุ่งๆ สามารถแบ่งประเภทของผิดพลาดได้ 3 ประเภท

### 1. กรณี Error (ระดับวิกฤต - ขั้น Catch)

- เกิดจาก JVM หรือ Hardware ซึ่งปัญหา โปรแกรมจะปิดตัวทันที
- ไม่ควรเขียน try-catch ล็อก

ชื่อ Error	สาเหตุ	วิธีแก้ไขเบื้องต้น
StackOverflowError	เรียก Method ซ้อนกันล้าเกินไป (มักเกิดจาก Recursive Loop ที่ไม่สิ้นสุด)	เช็คเรื่องไใช้ชุดของ Recursive / ลดการวนลูป
OutOfMemoryError	RAM ไม่พอ (สร้าง Object เข้าเกินไป หรือ ไฟล์ใหญ่เกิน)	อุณห Memory (Xmx) หรือแก้ค่าใช้คืนธรรม (Garbage Collection)
NoClassDefFoundError	ตอน Compile ไม่ไฟล์ Class อญ แต่ตอน Run ชา ไฟล์ Class ไม่เจอ	เช็ค Classpath หรือไฟล์ .jar ว่าครับไฟล์
UnknownError	ข้อผิดพลาดร้ายแรงที่ไม่ระบุสาเหตุใน JVM	กว้างๆ... เอ๊ะ! เช็ค System Log อย่างละเอียด



## 2. คลุ่ม Runtime Exception (Unchecked - ข้าที่คนเขียนทำเอง)

- Compiler ไม่บังคับให้ตัด (ไม่ต้อง try-catch ถ้า compile ผ่าน)
- แต่จำเป็นต้องทราบและระวัง ลดความผิดพลาดของ Programmer

ชื่อ Error	สาเหตุ	วิธีแก้ไขเบื้องต้น
NullPointerException (NPE)	พิษจากใช้ตัวแปรที่เป็นค่า null (เรียก method หรือตั้งค่า)	เช็ค if ( <code>x != null</code> ) ก่อนใช้เสมอ
ArrayIndexOutOfBoundsException	เรียก Index ของ Array เกินขนาดจริง (เช่น มี 5 ของ เรียกของที่ 10)	เช็ค <code>array.length</code> หรือ Loop ให้ถูก
ArithmetricException	คำนวณเลขผิดพลาดในส่วนตัว (เช่น หารด้วย 0)	เช็คตัวหารต้องไม่เป็น 0
IllegalArgumentException	ส่งค่า Argument ผิดประเภท/ผิดเงื่อนไข เข้าไปใน Method	ตรวจสอบค่าที่รับเข้า method

## 3. คลุ่ม Checked Exception (บังคับตัด)

- เกิดจาก ปัจจัยภายนอก (ไฟล์, เน็ต, ฐานข้อมูล)
- ปุ่ม "บังคับ" ให้เราต้องเขียน try-catch หรือ throws ไม่ใช่ Compile ไม่ผ่าน

ชื่อ Error	สาเหตุ	วิธีแก้ไขเบื้องต้น
IOException	ปัญหาการอ่าน/เขียน ข้อมูลทั่วไป (Input/Output)	ใช้ try-catch ตัด และแจ้งเตือนผู้ใช้
FileNotFoundException	หาไฟล์ตาม Path ที่ระบุไม่เจอ	เช็ค Path / สร้างไฟล์ default รองรับ
SQLException	เชื่อมต่อ Database ไม่ได้ หรือ SQL Query ผิด	เช็ค Connection String / เช็ค SQL Command
ClassNotFoundException	พิษจากไม่โหลด Class ด้วยชื่อ String และหาไม่เจอ	เช็คชื่อ Class / Library ใน Classpath

\*Error ที่เขียนไว้ตารางทั้งหมดเป็นเพียงตัวอย่างที่พบบ่อยเท่านั้น!!



## -Try-Catch-

Try-Catch គឺជាដែលទិន្នន័យនៃបញ្ហាសម្រាប់ការបង្រាក់ការបង្រាក់នៅក្នុងកិច្ចការបង្រាក់ការបង្រាក់ (Exception Handling) នៅក្នុង Java ដែលធានាទីផ្លូវការនៃការបង្រាក់ការបង្រាក់។ នៅក្នុងកិច្ចការបង្រាក់ការបង្រាក់ ត្រូវបានរាយការណ៍ដើម្បីបង្កើតការបង្រាក់ការបង្រាក់ដែលមិនអាចបង្រាក់បាន និងបង្កើតការបង្រាក់ការបង្រាក់ដែលមិនអាចបង្រាក់បាន។ នៅក្នុងកិច្ចការបង្រាក់ការបង្រាក់ ត្រូវបានរាយការណ៍ដើម្បីបង្កើតការបង្រាក់ការបង្រាក់ដែលមិនអាចបង្រាក់បាន និងបង្កើតការបង្រាក់ការបង្រាក់ដែលមិនអាចបង្រាក់បាន។

### Try-Catch

```
try {  
    int data = 50 / 0;  
    int x = 30 / 0; //<-skip this line  
} catch (ExceptionType e) {  
    System.out.println("Error Laew Bro: " + e.getMessage());  
}
```



- **try:** បើជាបុគ្គលិកដែលបានបង្កើតឡើងដើម្បីបង្កើតការបង្រាក់ការបង្រាក់។
- **ការចាយនៃ Block try:** នៅក្នុងកិច្ចការបង្រាក់ការបង្រាក់ ត្រូវបានរាយការណ៍ដើម្បីបង្កើតការបង្រាក់ការបង្រាក់។
- **catch:** បើជាបុគ្គលិកដែលបានបង្កើតឡើងដើម្បីបង្កើតការបង្រាក់ការបង្រាក់។
- **បញ្ជីការបង្រាក់ការបង្រាក់:** នៅក្នុងកិច្ចការបង្រាក់ការបង្រាក់ ត្រូវបានរាយការណ៍ដើម្បីបង្កើតការបង្រាក់ការបង្រាក់។



## Try-Catch-Finally

- **finally:** เป็น Block ที่จารนตีว่า จะทำงานแน่ ๆ 1000% ไม่ว่าจะเกิด Error บน Block try หรือไม่
- เผื่องสิ่งที่จะรับ: ใส่คำสั่ง "เก็บข้อมูล" (Cleanup) เช่น ปิดไฟล์, ปิดการเชื่อมต่อฐานข้อมูล (Close Connection) เพื่อคืน RAM ให้ระบบ

```
try {
    openFile();
} catch (Exception e) {
    handleError();
} finally {
    closeFile();
}
```



## Multiple Catch

```
try {...}
catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("อย่าเรียกช่องที่ไม่มีอยู่จริง!");
}

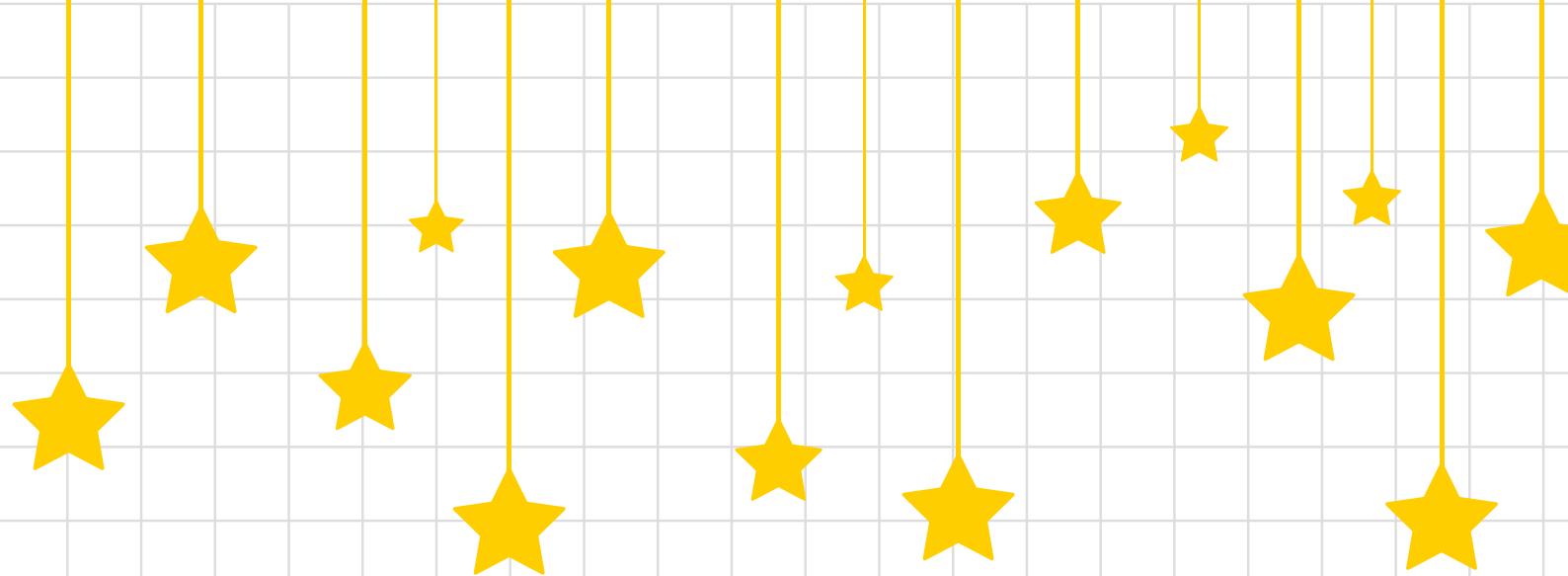
catch (ArithmetricException e) {
    System.out.println("ห้ามหารด้วย 0 นะ!");

}

catch (Exception e) {
    System.out.println("Error อืบๆ: " + e);
}
```



- **ตัดแยกแคลส:** หากใน try มีโอกาสเกิด Error หลายรูปแบบ เราสามารถเขียน catch หลาย Block ต่อ กัน เพื่อแยกจัดการแต่ละปัญหาได้
- **ตัวปิดท้าย:** ควรปิด catch (Exception e) ไว้เป็น Block สุดท้ายเสมอ เพื่อตัด Error ที่ไม่ไปที่เรา "คาดไม่ถูก" (กันเห็นช่วงเวลาเพื่อไม่ทรงกับแคลสข้างบน)



# THANK YOU FOR READING

ขอบคุณที่อ่านสรุปของเรานะครับ! เนื้อหา Java ขั้งสีอีกมาก

ถ้ามีโอกาสจะทำเพิ่มนะครับ :)))

