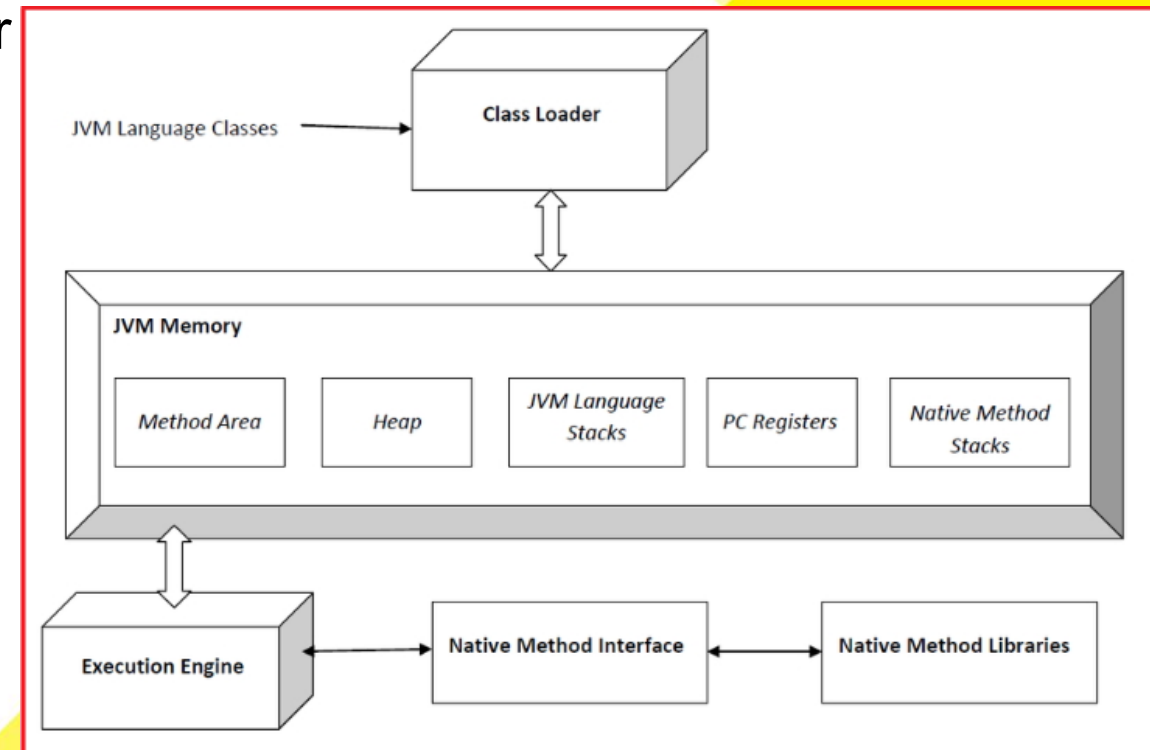


# Java Fundamentals

# Introduction To Java

Java เป็นภาษาโปรแกรมเชิงวัตถุที่มีความนิยมแพร่หลาย ในการเขียน Web Application หรือระบบขนาดใหญ่ที่เราเรียกกันว่า Enterprise Web Application ก็ตาม โดยโปรแกรมที่เขียนจาก Java จะต้องถูก Run บน **Java Virtual Machine(JVM)** เพื่อให้การทำงานนั้นไม่ต้องสนใจว่าอยู่บน Hardware หรือ OS ใดๆก็ตาม ขอให้เรามี JVM อยู่บนเครื่องนั้นก็พอ โดยจะมี **Class Loader** ทำหน้าที่ในการ Load Class เข้าสู่ JVM เพื่อใช้งาน กระบวนการทำงานของ Java จะมีการเก็บข้อมูลที่เป็น Object ไว้ใน Memory ที่เรียกว่า Heap เมื่อ Object ถูกใช้งานจนเสร็จก็就会被ทิ้งเป็นขยะอยู่ใน Heap เมื่อมีจำนวนมากขึ้นก็จะทำให้ Heap เต็มได้ ดังนั้น Java จึงมี **Garbage Collection(GC)** มาช่วยจัดการในส่วนนี้ โดยทำการเก็บ Object ที่ไม่ได้ใช้งานออกจาก Heap เพื่อให้มีพื้นที่เหลือสำหรับทำงานต่อไป



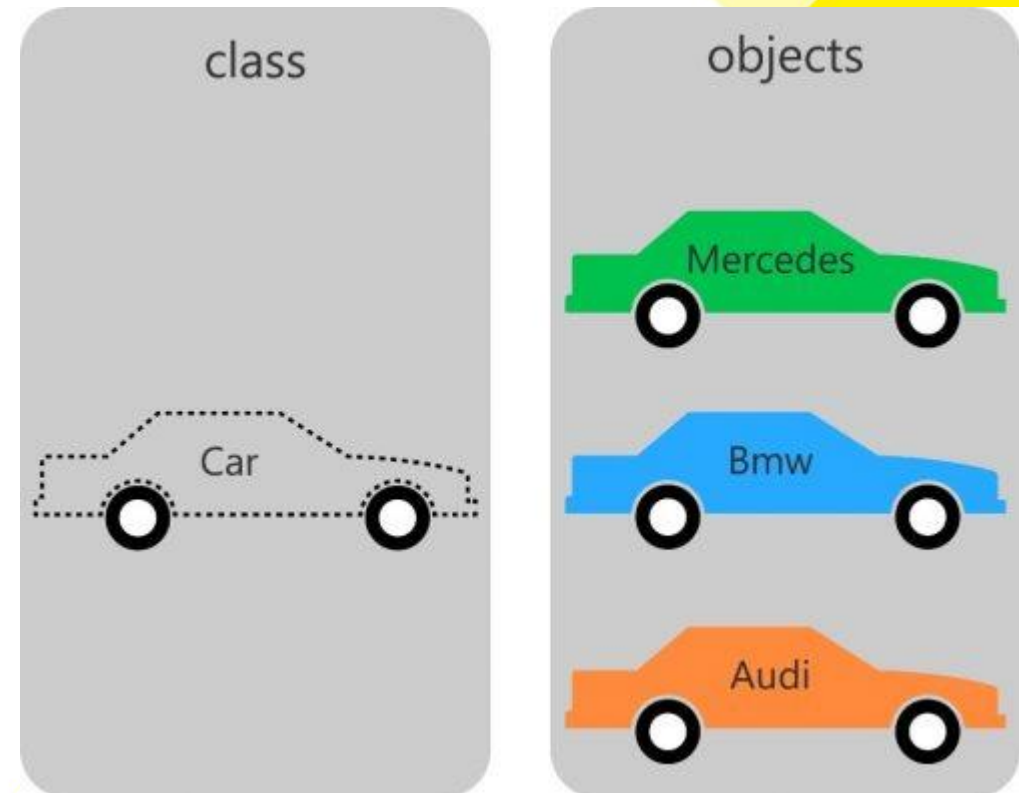
# Object Oriented Programming

เพื่อให้การพัฒนา Software มีส่วนประกอบที่ชัดเจน โครงสร้างที่ยืดหยุ่นและง่ายต่อการแก้ไข เราจึงใช้แนวคิดเชิงวัตถุในการออกแบบ Software โดยแนวคิดดังกล่าวมีส่วนประกอบหลักๆดังนี้

1. **Encapsulation** เป็นความสามารถในการปิดบังหรือซ่อน กระบวนการทำงานที่ซับซ้อนหรือข้อมูลที่ไม่ต้องการเปิดเผยให้ภายนอกทราบ
2. **Inheritance** เป็นการสืบทอดคุณสมบัติจากแม่มาสู่ลูก โดยที่ลูกไม่จำเป็นต้องเขียนโปรแกรมการทำงานใหม่ หากคุณสมบัติอื่นๆ เหมือนการทำงานของแม่ ยกเว้นแต่หากการทำงานของลูกมีการปรับปรุงหรือเพิ่มเติม จึงจะทำการเขียนโปรแกรมการทำงานนั้นๆแทนที่ของแม่ หรือเพิ่มเติมเข้าไป ทำให้ช่วยลดความซ้ำซ้อนไปได้มาก
3. **Polymorphism** เป็นความสามารถในการแปรเปลี่ยนการกระทำไปตามชนิดของวัตถุ โดยวัตถุที่อยู่ในตระกูลเดียวกัน แต่การทำงานแตกต่างกัน

# Object Oriented Programming

การใช้ภาษา Java กับการทำงานแบบ OOP นั้นจะอยู่ในรูปแบบของ Class ซึ่ง Class เปรียบเสมือน Blueprints หรือพิมพ์เขียวที่ใช้ในการสร้าง Object ขึ้นมา ตัวอย่าง เช่น Class Car เป็นต้นแบบในการสร้าง Object ของรถต่างๆ โดยจากรูป จะเห็นตัวอย่าง ในการ สร้างรถ Mercedes, Bmw, Audi จาก Class Car



# Object Oriented Programming

## Syntax พื้นฐานของ Java Class

```
<modifiers> class <class_name> {  
    [<attribute_declarations>]  
    [<constructor_declarations>]  
    [<method_declarations>]  
}
```

```
public class Car {  
    private String color;  
    public Car() {  
    }  
    public String getColor() {  
        return color;  
    }  
    public void setColor(String color) {  
        this.color = color;  
    }  
}
```

## Syntax พื้นฐานของ Attribute

```
<modifiers> <type> <name>;
```

```
2  
3 public class Car {  
4     private String color;  
5     private String name;  
6     private int wheel;  
7 }  
8
```

# Object Oriented Programming

## Syntax พื้นฐานของ Method

```
<modifiers> <return_type> <name>([<argument_list>]) {  
    [<statements>]  
}
```

## Syntax พื้นฐานของ Constructor

```
<modifier> <class_name> ([<argument_list>]) {  
    [<statements>]  
}
```

```
public class Car {  
    private String color;  
    private String name;  
    private int wheel;  
  
    public Car() {  
    }  
  
    public String getColor() {  
        return color;  
    }  
  
    public void setColor(String color) {  
        this.color = color;  
    }  
}
```

# Object Oriented Programming

- **Default Constructor** เนื่องจากทุกๆ Class จำเป็นต้องมี Constructor ที่ใช้ในการสร้าง Object อย่างน้อย 1 Constructor โดย Class ใดๆที่ไม่กำหนด Constructor แสดงว่า Class นั้นๆจะมี Default Constructor เพื่อใช้ในการสร้าง Object
  - Default Constructor ไม่รับ arguments
  - Default Constructor ไม่มี body
  - Default Constructor ไม่ปรากฏอยู่ใน Code
  - สร้าง Object จาก Default Constructor โดยใช้คำสั่ง `new ClassName();`

```
public class MyDate {
    private int day;
    private int month;
    private int year;
}
```

ใช้ Default Constructor

```
public class MyDate {
    private int day;
    private int month;
    private int year;

    public MyDate() {
    }
}
```

ไม่ใช้ Default Constructor

```
MyDate date = new MyDate();
```

การสร้าง Object จาก Default Constructor



# Object Oriented Programming

- การเข้าถึง Object Members ด้วยการใช้นotation dot notation

<object>.<member>

โดยเป็นการเข้าถึง attribute หรือ method ของ object นั้นๆ

```
public static void main(String[] args) {
    Car car = new Car();
    car.brand = "Honda";
    car.setColor("Red");
}
```

```
public class Car {
    private String color;
    private String name;
    private int wheel;
    public String brand;

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }
}
```

- Information Hiding ซ่อนการเข้าถึงข้อมูลโดยตรง เพื่อป้องกันการแก้ไขข้อมูลที่ผิดพลาด หรือเห็นข้อมูลที่ไม่ต้องการให้เห็น

```
public class MyDate {
    public int day;
    public int month;
    public int year;
}
```

เปิดข้อมูลให้ภายนอกเข้าถึงได้

```
public class MyDate {
    private int day;
    private int month;
    private int year;
}
```

ปิดไม่ให้ภายนอกเข้าถึงข้อมูลได้



# Object Oriented Programming

จะเห็นว่าการเปิดเผยข้อมูลให้ภายนอกเข้าถึงได้โดยตรงนั้น อาจทำให้ข้อมูลภายใน Object มีความผิดพลาดได้ หรือภายนอกสามารถดึงข้อมูลไปใช้ได้โดยตรง

```
public static void main(String[] args) {  
  
    MyDate date = new MyDate();  
    date.day = 32;  
  
    date.month = 2;  
    date.day = 31;  
  
    date.month = 12;  
    date.month = date.month + 1;  
  
}
```

กำหนดวันที่ ที่ไม่เหมาะสมเข้าไปได้

กำหนดวันที่ ไม่เหมาะสมกับเดือน

ดึงข้อมูลเดือนออกมาแก้ไขได้โดยตรง  
ทำให้เดือนที่ได้ผิดจากความเป็นจริง

# Object Oriented Programming

แนวทางการแก้ไขคือการใช้ Information Hiding เข้ามาช่วยเพื่อปกปิดข้อมูลที่ไม่ต้องการให้ภายนอกเข้าถึงได้โดยตรง หากต้องการเข้าถึงข้อมูลให้กระทำผ่านกระบวนการ (method) ที่เราเตรียมไว้ให้เท่านั้น

```
public static void main(String[] args) {
    MyDate date = new MyDate();
    date.day = 32;
    date.setDay(32);

    date.month = 2;
    date.setMonth(2);

    date.day = 31;
    date.setDay(31);

    date.month = 12;
    date.setMonth(12);

    date.month = date.month + 1;
    date.setMonth(date.getMonth() + 1);
}
```

```
public class MyDate {
    private int day = 1;
    private int month = 1;
    private int year = 2562;

    public int getDay() {
        return day;
    }

    public void setDay(int day) {
        if(day < 1 || day > 31) return;
        this.day = day;
    }

    public int getMonth() {
        return month;
    }

    public void setMonth(int month) {
        if(month < 1 || month > 12) return;
        this.month = month;
    }

    public int getYear() {
        return year;
    }

    public void setYear(int year) {
        this.year = year;
    }
}
```

# Object Oriented Programming

**Inheritance** เป็นการสืบทอดคุณสมบัติจาก Class แม่ ไปสู่ Class ลูก จากตัวอย่างคือ Employee ไปสู่ Manager โดยการสืบทอดนั้นจะใช้คำสั่ง `extends` ในการกำหนดการสืบทอด โดยการสืบทอดมีการกำหนดข้อยกเว้นในส่วน ของ constructor

Class Manager สืบทอดจาก Employee

```
public class Employee {
    public String name;
    public int salary;

    public String getDetails() {
        this.name = "Amarin Sangkarat";
        this.salary = 10000;
        return "Name [" + name + "] Salary [" + salary + "]";
    }

    public void setDetails(String name, int salary) {
        this.name = name;
        this.salary = salary;
    }
}
```

Override Method

Overload Method

```
public class Manager extends Employee {
    public String department;

    public String getDetails() {
        this.name = "Jojo Jotaro";
        this.salary = 50000;
        this.department = "Application Development";
        return department + " Manager Name [" + name + "] Salary [" + salary + "]";
    }

    public void setDetails(String name, int salary, String department) {
        this.name = name;
        this.salary = salary;
        this.department = department;
    }
}
```

# Object Oriented Programming

จากตัวอย่างก่อนหน้านี้เราจะเห็นว่าในส่วนของ Class Manager ที่สืบทอดมาจาก Class Employee นั้น จะได้รับคุณสมบัติต่างๆมาจาก Class Employee ทั้งหมด ยกเว้นเพียงแต่ constructor เท่านั้น เนื่องจาก member ทั้งหมดของ Employee มี Access Modifiers เป็น public สิ่งที่เราเห็นจาก Class ตัวอย่างคือ

1. **Override method** คือ การเขียนทับกระบวนการทำงานของ method จาก class แม่ โดยชื่อ method , parameter และ return type จำเป็นต้องเหมือนกันทั้งหมด แต่ final method และ private method ไม่สามารถ Override ได้ ตัวอย่าง method getDetails() จะเห็นว่า Class Manager มีการเขียน Override method เพื่อเพิ่มในส่วนของการแสดง department เข้าไป
2. **Overload method** ไม่เกี่ยวกับการสืบทอด แต่แสดงเพื่อไม่ให้เกิดความสับสนในส่วนของการชื่อ method ที่เหมือนกัน แต่ต่างกันที่จำนวน parameter หรือ parameter type ซึ่งหากเป็นเช่นนี้จะเรียกว่า Overload method จากตัวอย่างคือ method setDetails ที่มีจำนวน parameter ไม่เท่ากัน โดย Overload method ใน 1 Class จะมีจำนวนเท่าใดก็ได้ จะอยู่ใน Class ลูกกับ Class แม่ก็ได้ หรืออยู่ใน Class เดียวกันก็ได้

# Object Oriented Programming

**Access Modifiers** คือระดับในการเข้าถึง member ของ Class เนื่องจากการทำงานแบบ OOP นั้นจำเป็นต้องมีการปิดบังการทำงาน หรือข้อมูลภายในไว้บางกรณี จึงจำเป็นต้องแบ่งระดับในการเข้าถึงข้อมูลต่างๆ ดังนี้

Modifier	Class	Package	Sub Class	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

# Object Oriented Programming

**Polymorphism** จากที่บอกไว้ก่อนหน้านี้เป็นความสามารถในการเปลี่ยนการกระทำไปตามชนิดของวัตถุ โดยวัตถุที่อยู่ในตระกูลเดียวกัน แต่การทำงานแตกต่างกัน มาดูตัวอย่างของ Object ที่อยู่ในตระกูลเดียวกันแต่ทำงานแตกต่างกัน โดยเราจะมี Interface Printer เป็นต้นตระกูล และมี Class Canon, Epson, Brother เป็น Class ที่ Implement Printer ไป โดยคำสั่งที่ใช้คือคำว่า implements

```
public interface Printer {
    public void print();
}
```

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

    Printer canon = new Canon();
    Printer epon = new Epson();
    Printer brother = new Brother();

    canon.print();
    epon.print();
    brother.print();

}
```

```
public class Canon implements Printer {
    public void print() {
        System.out.println("Printer Canon");
    }
}
```

```
public class Epson implements Printer {
    public void print() {
        System.out.println("Printer Epson");
    }
}
```

```
public class Brother implements Printer {
    public void print() {
        System.out.println("Printer Brother");
    }
}
```



# Data Types In Java

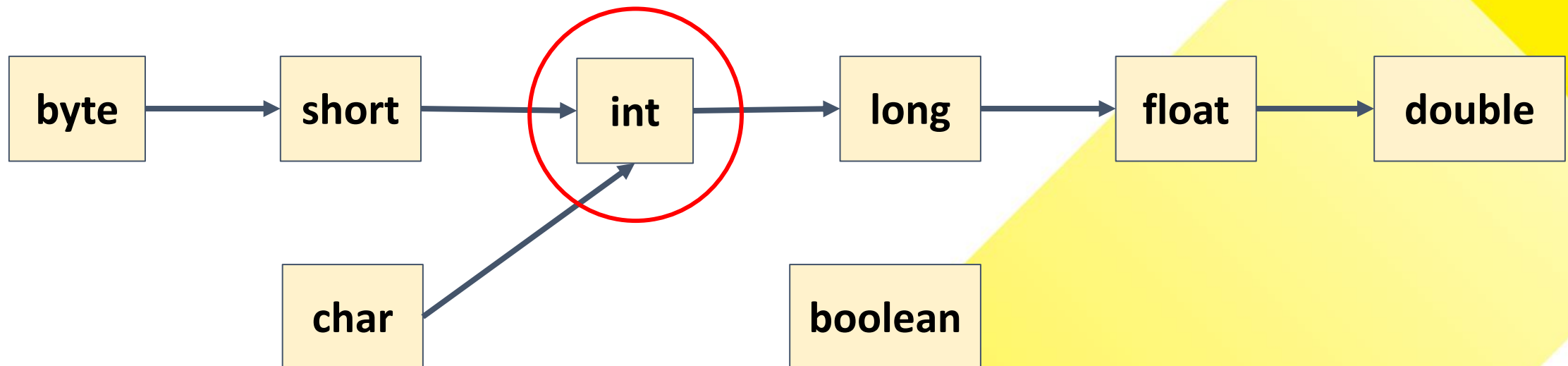
- Primitive Data Types เป็นชนิดข้อมูลพื้นฐานของ Java ที่ได้จัดเตรียมไว้ให้ โดยแบ่งประเภทตามนี้
  - Logical : boolean
  - Textual : char
  - Integral : byte, short, int, long
  - Floating : float, double

```
public class PrimitiveType {  
    byte byteValue = 1;  
    short shortValue = 1;  
    int intValue = 1;  
    long longValue = 11;  
    float floatValue = 1.00f;  
    double doubleValue = 1.00d;  
    char charValue = 'A';  
    boolean booleanValue = true;  
}
```



# Data Types In Java

- Automatic Type Promotion and Casting ในส่วนของ Primitive Data Types นั้นจะมีกระบวนการทำงานที่เกี่ยวข้องคือ การ Promote จาก Size ที่เล็กกว่าเป็น Type ที่ใหญ่ขึ้น และการ Casting คือกำหนดให้เป็นขนาดที่เราต้องการ ในส่วนของการ Promote นั้น สามารถดูลำดับการ Promote จากรูปด้านล่าง โดยจะเห็นว่า boolean ซึ่งเป็นประเภท Logical นั้นจะไม่สามารถ Promote ได้



# Data Types In Java

- ตัวอย่าง กระบวนการ Automatic Promotion และ Casting

```
public static void main(String[] args) {  
  
    byte byteValue1 = 1;  
    byte byteValue2 = 1;  
    byte plusByteValue = byteValue1 + byteValue2;  
  
}
```

Type mismatch: cannot convert from int to byte  
2 quick fixes available:  
1. Add cast to 'byte'  
2. Change type of 'plusByteValue' to 'int'

```
public static void main(String[] args) {  
  
    byte byteValue1 = 1;  
    byte byteValue2 = 1;  
    byte plusByteValueToByte = (byte)(byteValue1 + byteValue2);  
    int plusByteValue = byteValue1 + byteValue2;  
  
}
```

ไม่สามารถกำหนด data type byte  
ได้เนื่องจากเกิดกระบวนการ  
automatic promotion

## วิธีการแก้ไข

1. หากต้องการ data type byte ให้ใช้กระบวนการ Casting ให้เป็น byte
2. ให้เปลี่ยน data type ไปตาม type ที่ได้จากการ promote

# Data Types In Java

- **Reference Types** ประเภทข้อมูลอื่นๆที่ไม่ใช่ Primitive Types ซึ่งจะเป็นประเภทข้อมูลที่อยู่ในรูปแบบของ Object เช่น Wrapper Class , String และ Class อื่นๆที่สร้างขึ้นมา

Wrapper Class

- อยู่ในรูปแบบของ Object
- ห่อหุ้มค่า Primitive อยู่
- เปลี่ยนแปลงค่าภายใน Object ไม่ได้
- ซ่อมักคล้าย Primitive Types แต่ขึ้นต้นด้วยตัวใหญ่ ตามรูปแบบของการตั้งชื่อ Class

Primitive Types	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

# Data Types In Java

## Autoboxing and Unboxing

จากที่เราได้เรียนรู้เรื่อง Data Types ที่เป็น Primitive Types และ Wrapper Class ไปแล้วนั้นเนื่องจาก Wrapper Class เป็น Class ที่ใช้ในการห่อหุ้ม Primitive Types ไว้ Java จึงอำนวยความสะดวกในการแปลงค่าของทั้งสองประเภทให้ดังนี้

```
/*
 * 1.กระบวนการ Autoboxing คือการเปลี่ยนจาก
 * Primitive Types เป็น Wrapper Class
 */
public void autoboxing() {

    Integer intValue = 1;
    Long longValue = 1L;

}
```

```
/*
 * 2.กระบวนการ Unboxing คือการเปลี่ยนจาก
 * Wrapper Class เป็น Primitive Types
 */
public void unboxing() {

    Integer intValue = new Integer(1);
    Long longValue = new Long(1);

    int intValue2 = intValue;
    long longValue2 = longValue;

}
```

# Data Types In Java

String เป็น Type ที่ใช้กันบ่อยมาก โดย Java จะมีกระบวนการจัดการ String เป็นพิเศษเรียกว่า String Pool เพื่อให้เกิดการใช้งาน Memory ให้เกิดประโยชน์สูงสุด การสร้าง Object String มีด้วยกัน 2 แบบคือ

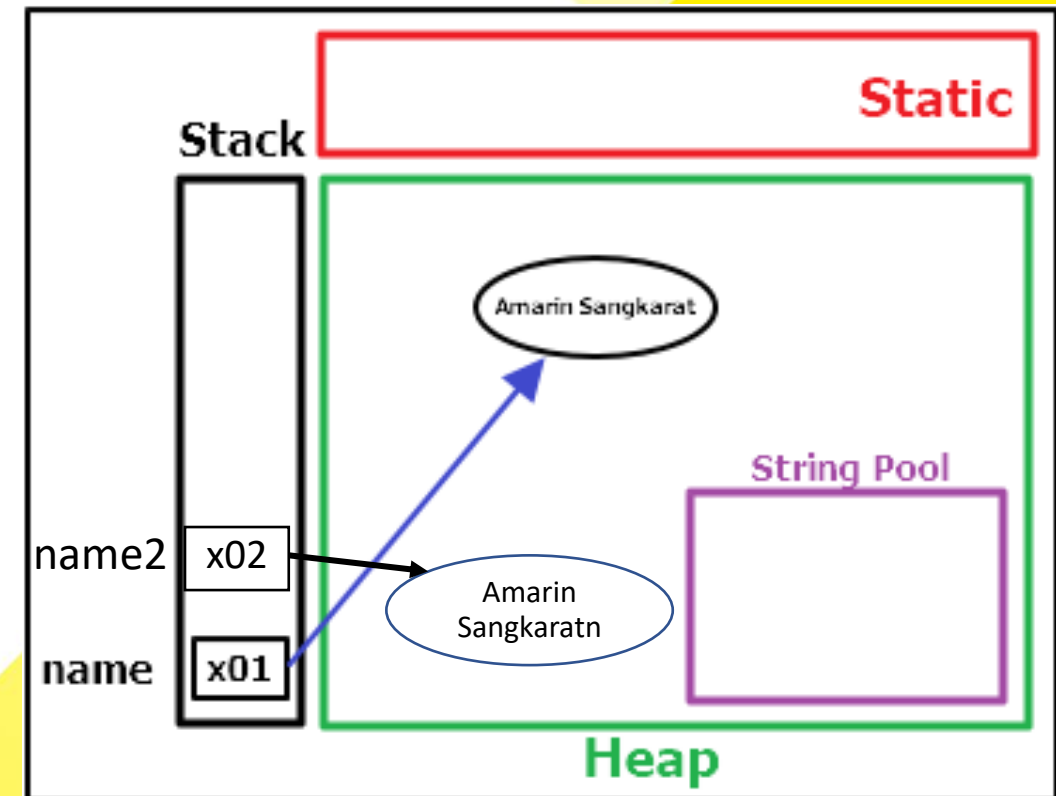
```
public void newString() {
    String name = new String("Amarin Sangkarat");
}
```

1. การสร้างผ่าน Constructor

การสร้าง String โดยการ new Object นั้นจะทำการสร้าง String เข้าไปในส่วนของ Memory ส่วน Heap หลัก หากมีคำสั่ง

`String name2 = new String("Amarin Sangkarat")`

อีกครั้ง ก็จะเป็นการสร้าง Object ใหม่เสมอ



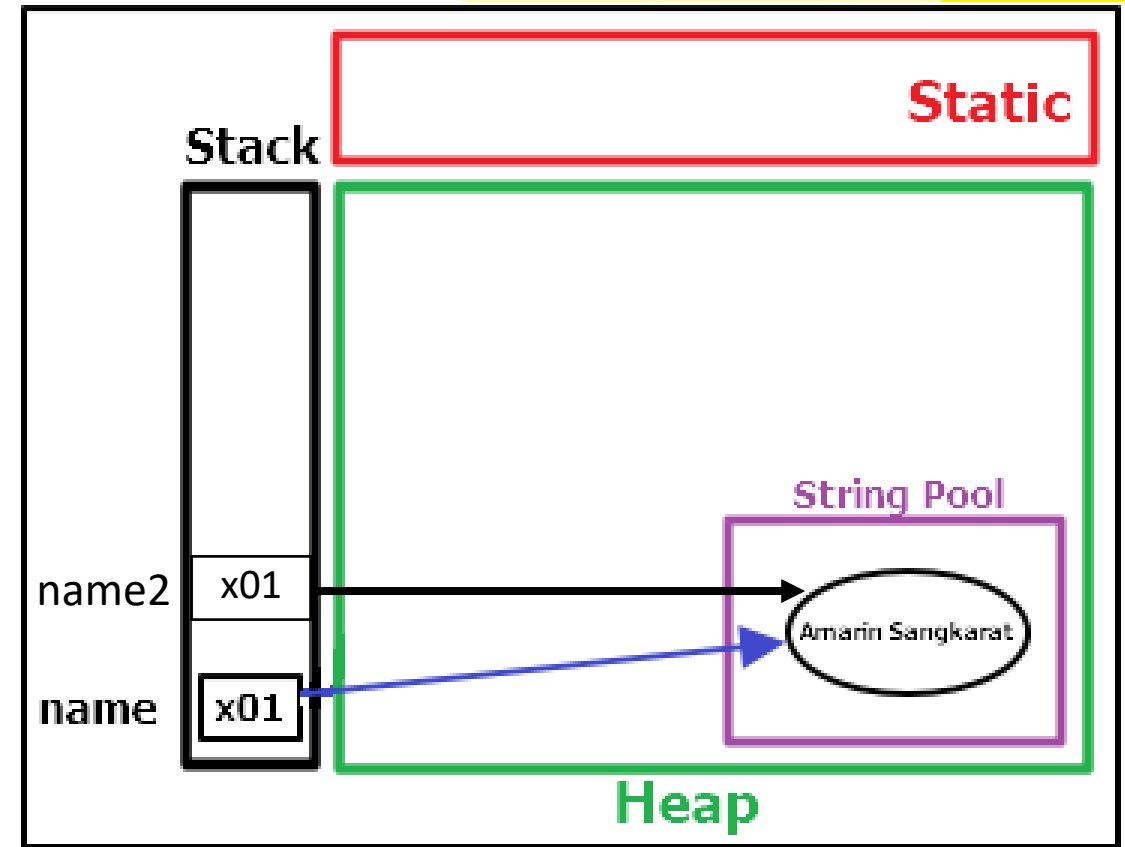
# Data Types In Java

```
public void literalString() {  
    String name = "Amarin Sangkarat";  
}
```

2. การสร้างผ่าน Literal ""

การสร้าง String โดยการ Literal "" นั้น Java จะทำการตรวจสอบก่อนเสมอว่ามี String คำนี้อยู่ใน String Pool แล้วหรือไม่หากมีแล้วจะไม่สร้าง String ขึ้นมาใหม่แต่จะให้ใช้ String ตัวเดิมเพื่อไม่ให้เกิดการสร้าง String มากจนเกินไป

String name2 = "Amarin Sangkarat";



# Scope Of Variables In Java

Scope ของตัวแปร ในภาษา Java นั้นหลักๆแบ่งออกเป็น

1. **Member Variable(Class Level Scope)** หมายถึงตัวแปรที่อยู่ในระดับของ Class ซึ่งสามารถที่จะนำไปใช้ได้ในทุกที่ภายใน Class ข้อมูลจะถูกเก็บอยู่ใน Object ที่สร้างอยู่ใน Heap
2. **Local Variable(Method Scope)** หมายถึงตัวแปรที่อยู่ในระดับของ Method สามารถที่จะนำไปใช้ได้ทุกที่ภายใน Method ซึ่ง Scope จะแคบกว่า Member Variable ข้อมูลของตัวแปรจะถูกเก็บอยู่ใน Stack เมื่อหมด Scope ก็จะถูกนำออกจาก Stack
3. **Loop Variable(Loop Scope)** หมายถึงตัวแปรที่อยู่ในระดับของ Loop สามารถที่จะนำไปใช้ได้ทุกที่ภายใน Loop ซึ่ง Scope จะแคบกว่า Member Variable และ Method Variable ข้อมูลของตัวแปรจะถูกเก็บอยู่ใน Stack เมื่อหมด Scope ก็จะถูกนำออกจาก Stack



# Scope Of Variables In Java

```
public class ScopeOfVariable {
```

```
    private int sum;  
    private int max;
```

1. Member Variable(Class Level Scope)

```
    public void process() {
```

```
        int newMax = 50;
```

2. Local Variable(Method Level Scope)

```
        this.max = newMax;
```

```
        for (int i = 0; i < max; i++) {
```

```
            sum = sum + i;
```

```
            String detail = "No.";  
            detail = detail + i;
```

3. Loop Variable(Loop Level Scope)

```
            System.out.println("Loop " + detail);
```

```
        }
```

```
    }
```

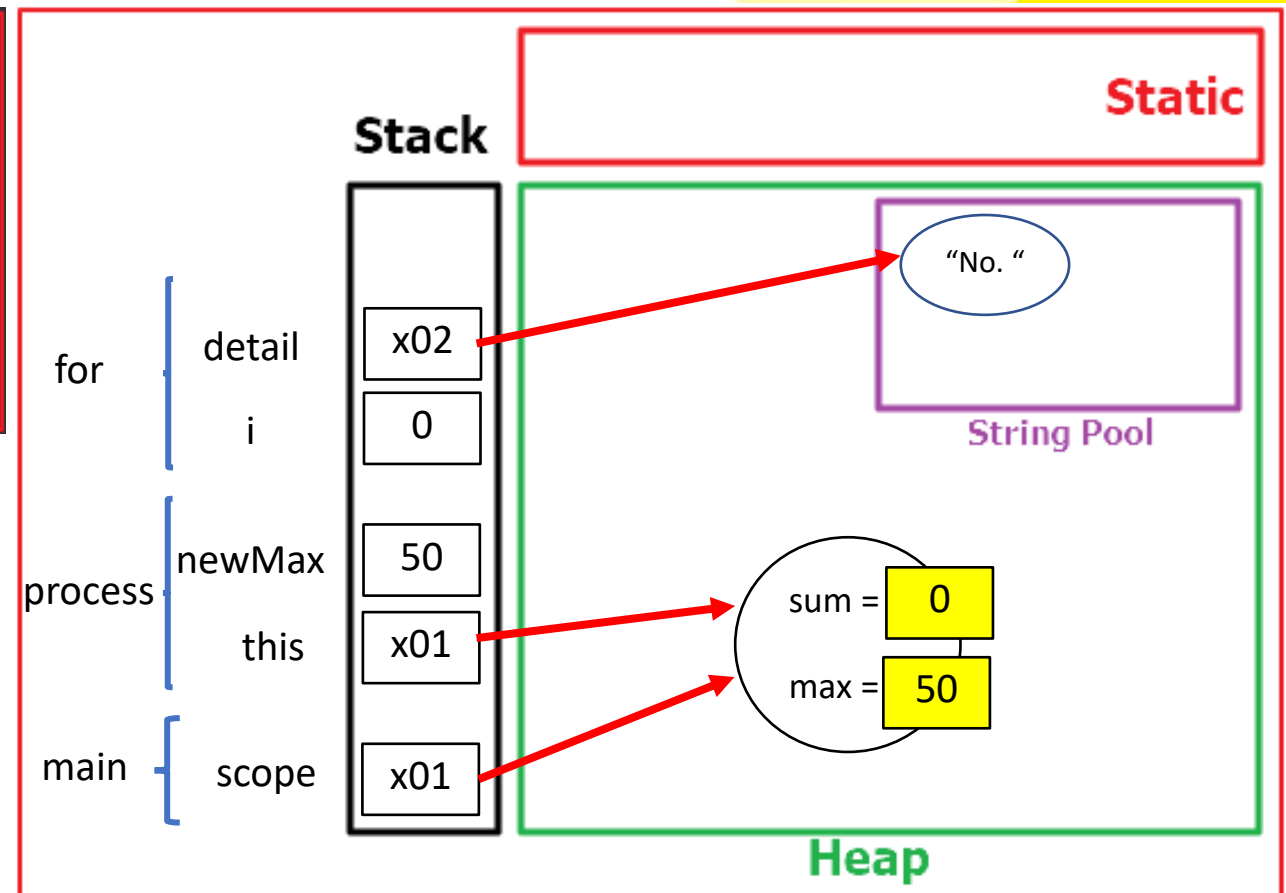
```
}
```

# Scope Of Variables In Java

หลังจากทราบแล้วว่า Scope ของตัวแปรเป็นอย่างไร มาดูการนำค่าของตัวแปรของแต่ละ Scope เข้าสู่ Memory ว่ากระบวนการนำเข้ามีลำดับการทำงานอย่างไร

```
public class TestScopeOfVariable {
    public static void main(String[] args) {
        ScopeOfVariable scope = new ScopeOfVariable();
        scope.process();
    }
}
```

this Reference ใช้สำหรับอ้างอิงถึง member ต่างๆ ภายใน Object ของ Class เช่น Constructor, Method, Instance Variable



# Control Statement

คำสั่งที่ใช้ในการควบคุมการทำงานภายในโปรแกรมของ Java แบ่งออกเป็นหลักๆ ดังนี้

- Decision Making Statements คือประเภทคำสั่งที่ใช้ในการตัดสินใจ
  - if statement(if , else)
  - switch statement

## if statement(if , else)

```
if(condition) //statement of block;

if(condition) {
    //statement of block;
}

if(condition) //statement of block;
else //statement of block;

if(condition) {
    //statement of block;
} else if(condition) {
    //statement of block;
} else {
    //statement of block;
}
```

```
public void ifThenElse() {
    int score = 79;

    if(score >= 80) System.out.println("Grade A");

    if(score >= 80) {
        System.out.println("Grade A");
    }

    if(score >= 80) System.out.println("Grade A");
    else System.out.println("Grade F");

    if(score >= 80) {
        System.out.println("Grade A");
    } else if(score >= 70) {
        System.out.println("Grade B");
    } else if(score >= 60) {
        System.out.println("Grade C");
    } else {
        System.out.println("Grade D");
    }
}
```

# Control Statement

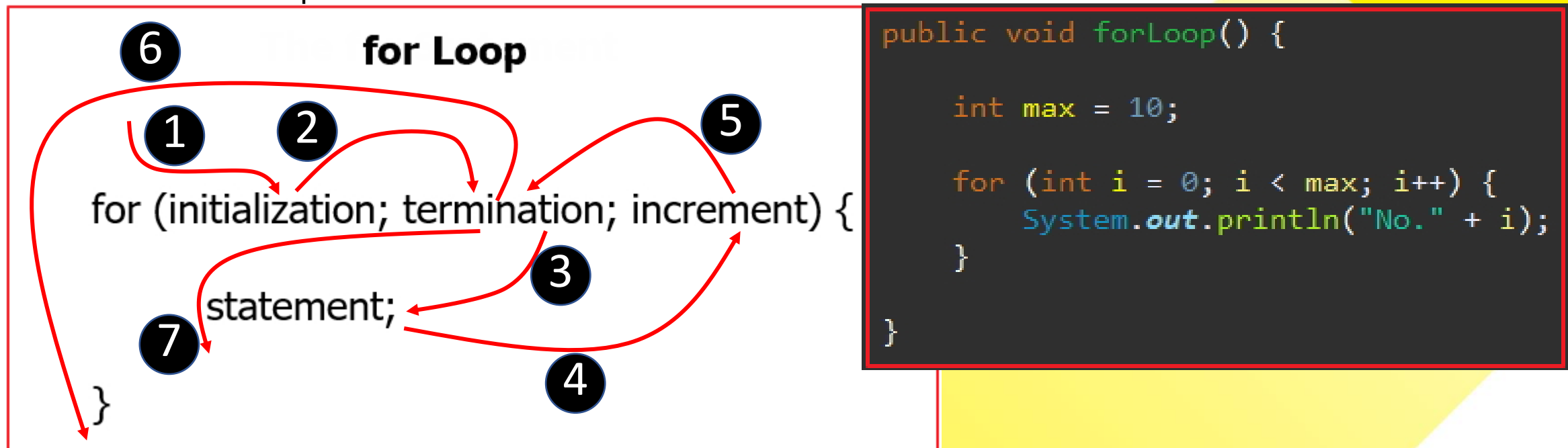
## Switch Statement

```
switch(expression) {
    case constant1 :
        statement;
        break;
    case constant2 :
        statement;
        break;
    .
    .
    .
    default :
        statement;
        break;
}
```

```
public void switchCase() {
    int i = 2;
    switch(i) {
        case 1 :
            System.out.println("Value 1");
            break;
        case 2 :
            System.out.println("Value 2");
            break;
        case 3 :
            System.out.println("Value 3");
            break;
        default :
            System.out.println("Value > 3");
            break;
    }
}
```

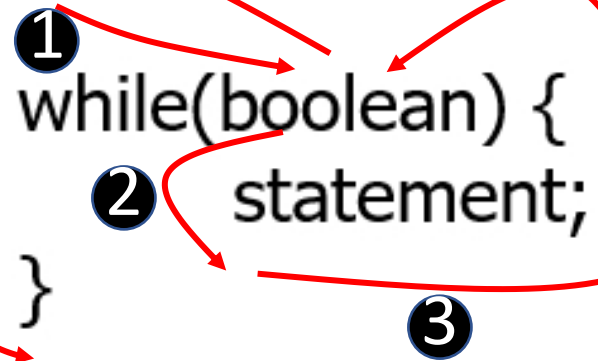
# Control Statement

- Looping Statements คือประเภทคำสั่งที่ใช้วนซ้ำการทำงานตามเงื่อนไขที่กำหนด
  - for loop
  - while loop
  - do...while loop
  - for-each loop



# Control Statement

## ④ while Loop



```

1 while(boolean) {
2   statement;
3 }
4

```

The diagram illustrates the flow of a while loop. It starts at point 1, enters the loop body at point 2, executes the statement, and then reaches point 3. From point 3, the flow loops back to point 1, checking the condition again. Point 4 is located below the closing brace, representing the exit point of the loop.

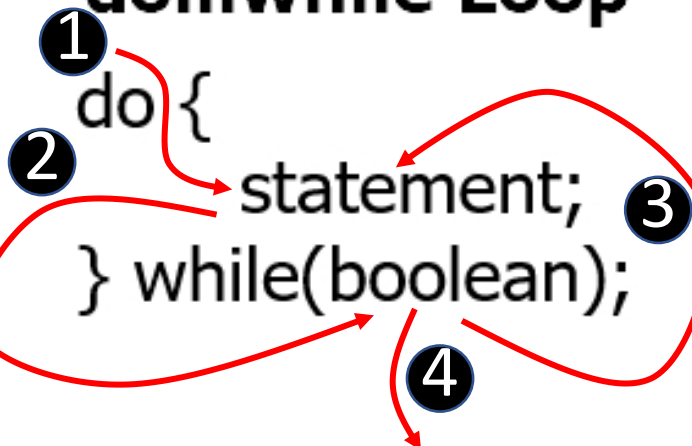
```

public void whileLoop() {
    int i = 1;

    while(i <= 10) {
        System.out.println("No." + i);
        i++;
    }
}

```

## do...while Loop



```

1 do {
2   statement;
3 } while(boolean);
4

```

The diagram illustrates the flow of a do...while loop. It starts at point 1, enters the loop body at point 2, executes the statement, and then reaches point 3. From point 3, the flow goes to point 4, which checks the condition. If the condition is true, the flow loops back to point 1. If the condition is false, the flow exits the loop.

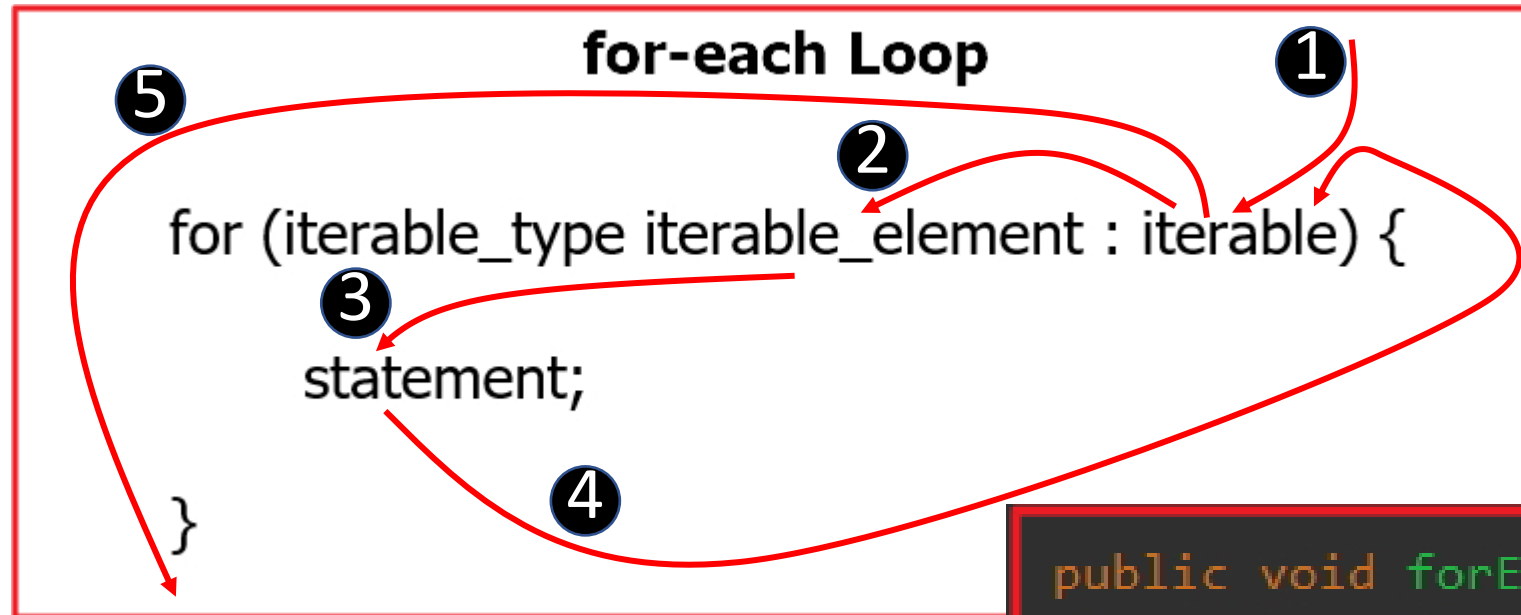
```

public void doWhileLoop() {
    int i = 1;

    do {
        System.out.println("No." + i);
        i++;
    } while(i <= 10);
}

```

# Control Statement



```
public void forEachLoop() {
    int[] values = {1 , 2 , 3 , 4, 5};

    for (int value : values) {
        System.out.println("No." + value);
    }
}
```



# Control Statement

- Branching Statements คือคำสั่งที่ใช้กระโดดข้ามจากคำสั่งหนึ่งไปอีกคำสั่งหนึ่ง
  - **break** คำสั่งที่ใช้ในการหยุดการทำงานเพื่อออกจาก loop
  - **continue** คำสั่งที่ให้โปรแกรมกลับไปทำงานที่ต้น loop โดยไม่ทำคำสั่งที่เหลือ
  - **return** คำสั่งที่ใช้ในการหยุดการทำงานเพื่อกลับไปตำแหน่งที่เรียกใช้งาน method

```
public void branching() {
    int max = 10;

    for (int i = 0; i < max; i++) {
        if(i == 7) {
            max = i;
            break;
        }

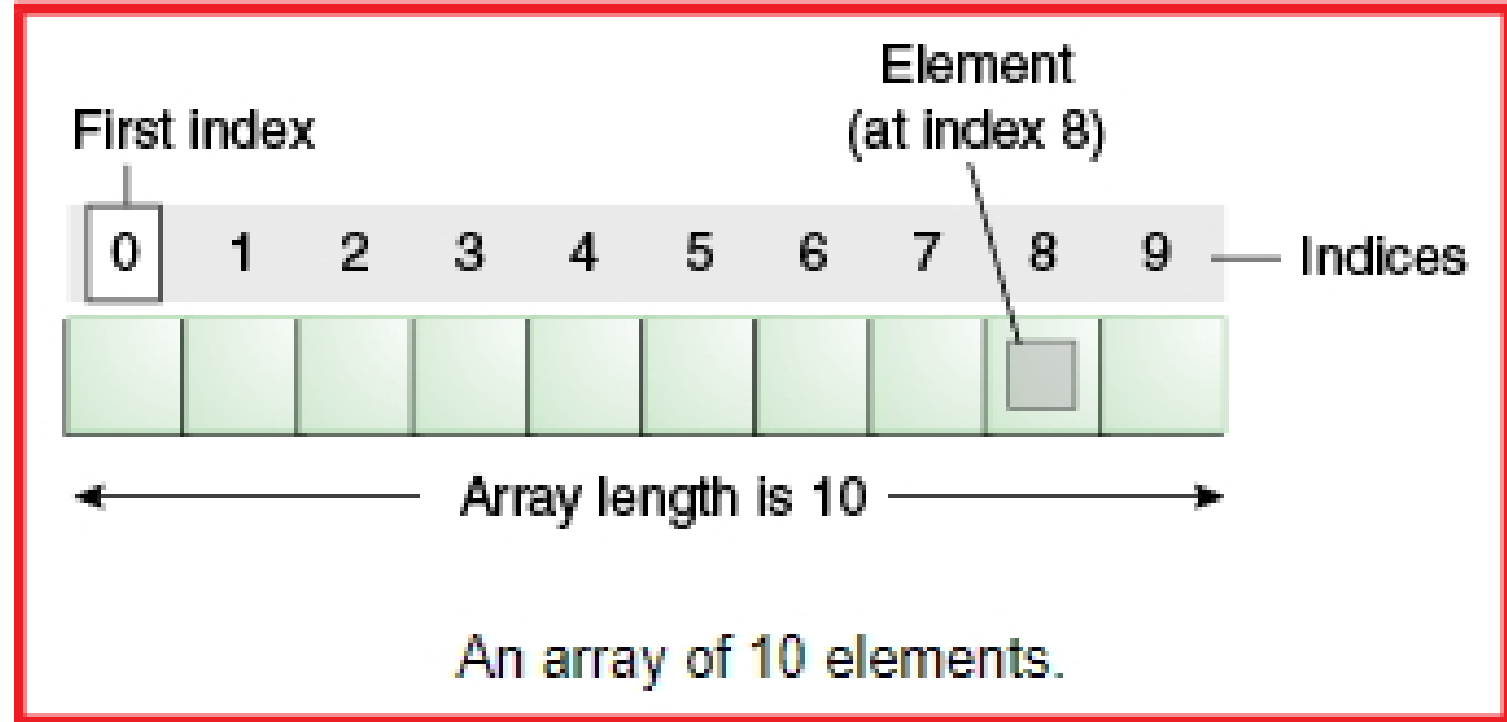
        if(i == 3) {
            continue;
        }

        System.out.println("No." + i);
    }

    if(max == 7) {
        return;
    }
}
```

# Array and Collections Framework

- Array เป็นชุดของกลุ่มข้อมูลประเภทเดียวกัน สามารถเก็บได้ทั้ง Primitive Types และ Object โดยไม่สามารถเปลี่ยนแปลงขนาดได้ โดยมี 2 รูปแบบคือ
  1. Array 1 มิติ
  2. Array หลายมิติ



# Array and Collections Framework

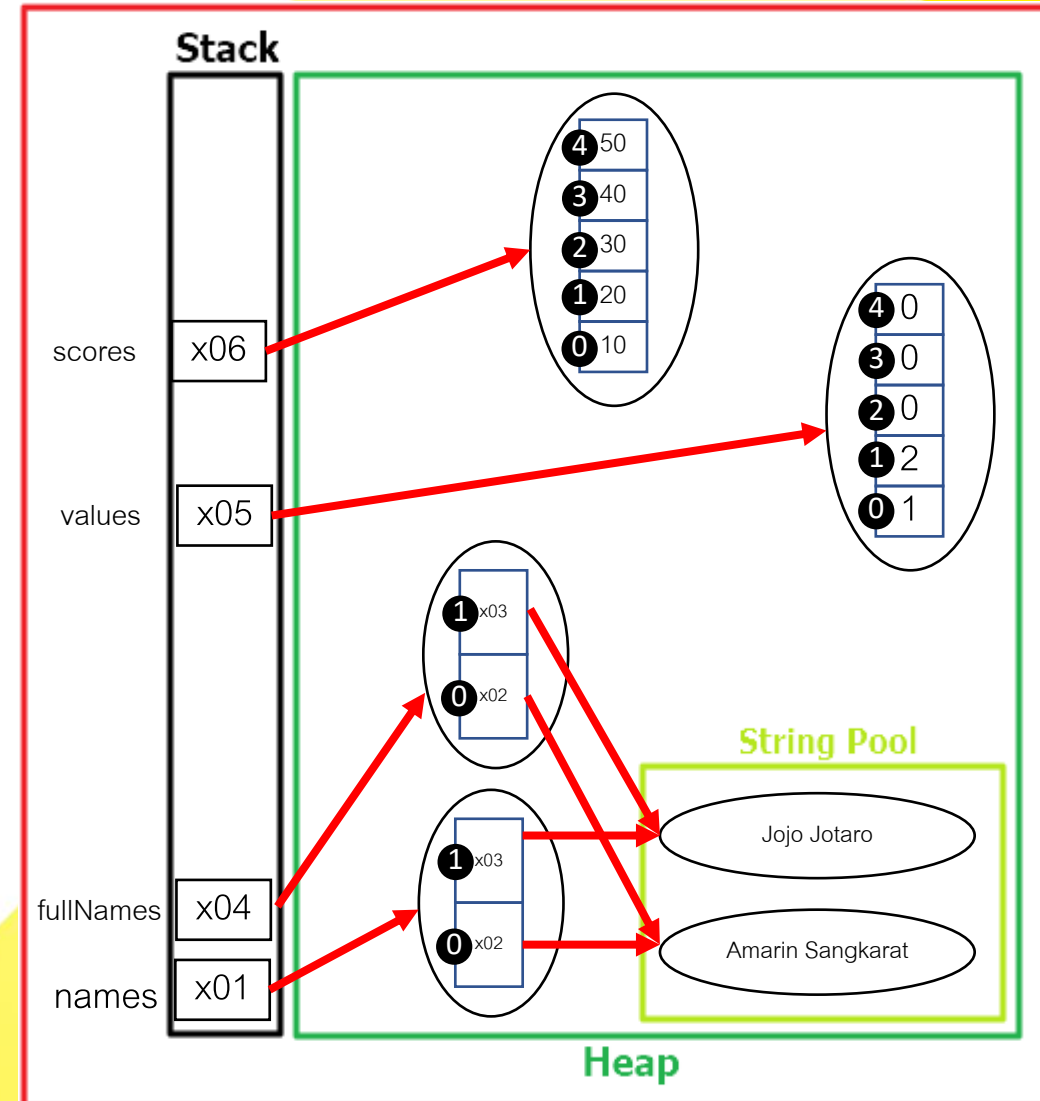
## Array One Dimension

```
public void demoArrayOneDimension() {
    String[] names = new String[10];
    names[0] = "Amarin Sangkarat";
    names[1] = "Jojo Jotaro";
    for (int i = 0; i < names.length; i++) {
        System.out.println("name [" + i + "] " + names[i]);
    }

    String[] fullNames = {"Amarin Sangkarat", "Jojo Jotaro"};
    for (int i = 0; i < fullNames.length; i++) {
        System.out.println("fullName [" + i + "] " + fullNames[i]);
    }

    int[] values = new int[5];
    values[0] = 1;
    values[1] = 2;
    for (int i = 0; i < values.length; i++) {
        System.out.println("value [" + i + "] " + values[i]);
    }

    int[] scores = {10, 20, 30, 40, 50};
    for (int i = 0; i < scores.length; i++) {
        System.out.println("score [" + i + "] " + scores[i]);
    }
}
```



# Array and Collections Framework

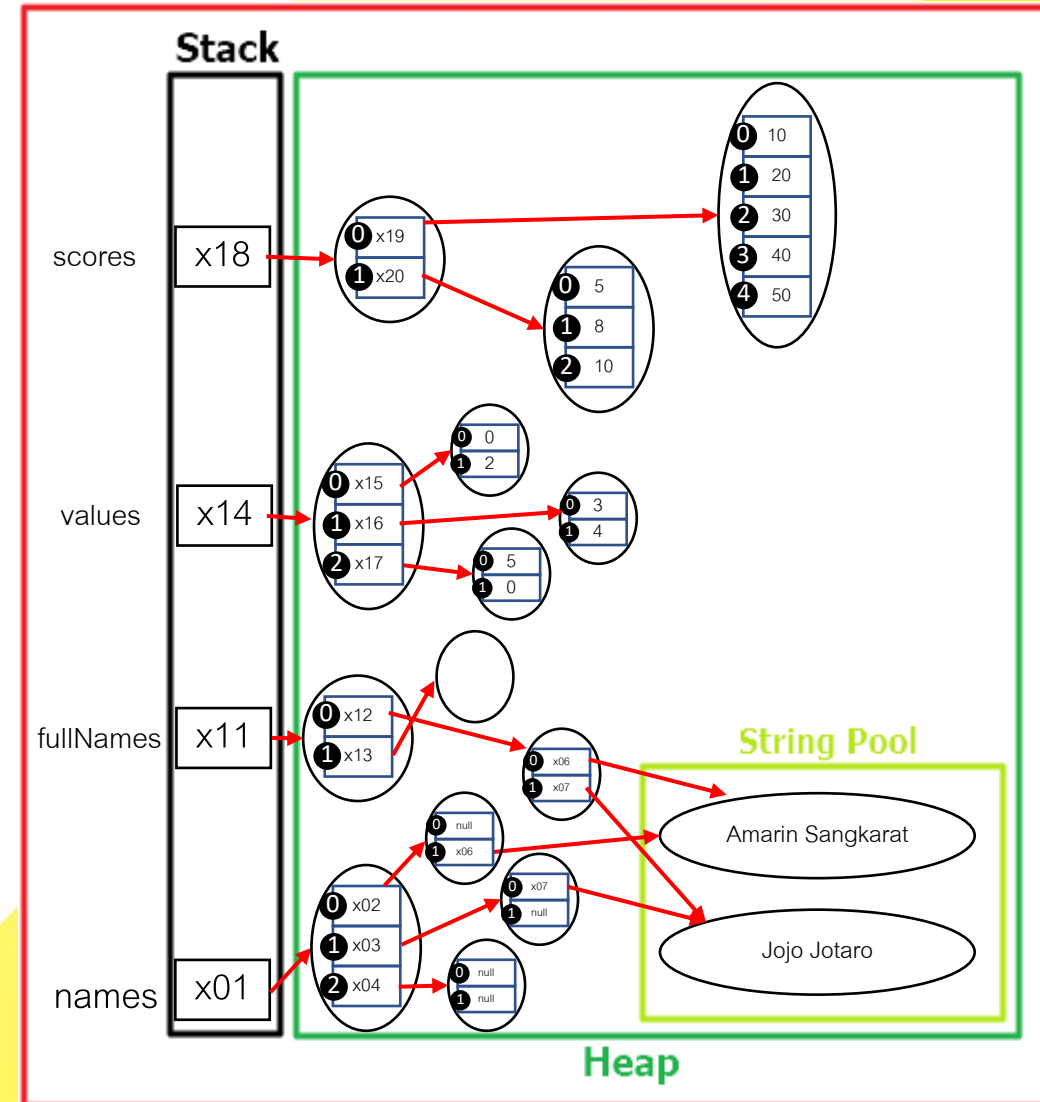
## Array Multi Dimension

```
public void demoArrayMultiDimension() {
    String[][] names = new String[3][2];
    names[0][1] = "Amarin Sangkarat";
    names[1][0] = "Jojo Jotaro";
    for (int i = 0; i < names.length; i++) {
        for (int j = 0; j < names[i].length; j++) {
            System.out.println "[" + i + "]" + j + " " + names[i][j]);
        }
    }

    String[][] fullNames = {"Amarin Sangkarat", "Jojo Jotaro"}, {};
    for (int i = 0; i < fullNames.length; i++) {
        for (int j = 0; j < fullNames[i].length; j++) {
            System.out.println "[" + i + "]" + j + " " + fullNames[i][j]);
        }
    }

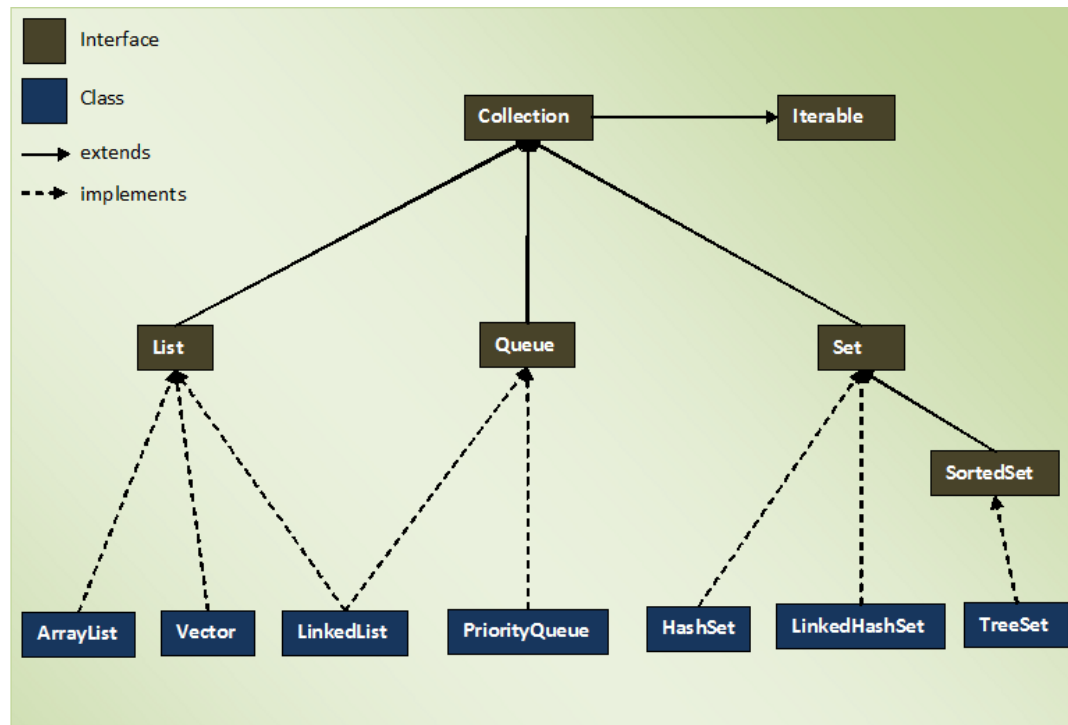
    int[][] values = new int[3][2];
    values[0][1] = 2;
    values[1][0] = 3;
    values[1][1] = 4;
    values[2][0] = 5;
    for (int i = 0; i < values.length; i++) {
        for (int j = 0; j < values[i].length; j++) {
            System.out.println "[" + i + "]" + j + " " + values[i][j]);
        }
    }

    int[][] scores = {{10,20,30,40,50},{5,8,10}};
    for (int i = 0; i < scores.length; i++) {
        for (int j = 0; j < scores[i].length; j++) {
            System.out.println "[" + i + "]" + j + " " + scores[i][j]);
        }
    }
}
```



# Array and Collections Framework

- **Collections Framework** เป็น Object ที่แสดงถึงกลุ่มของ Objects และสามารถขยายขนาดได้ เนื่องจากเป็นการเก็บ Object ดังนั้นจึงไม่สามารถเก็บข้อมูลแบบ Primitive Types ได้ แบ่งกลุ่มการทำงานออกเป็น 4 กลุ่ม คือ Set, List, Queue, Map โดยเราจะพูดถึงหลักๆ 3 กลุ่มคือ Set, List, Map และเฉพาะบาง Class ที่ใช้บ่อย



# Array and Collections Framework

## 1. Set เก็บข้อมูลแบบไม่เรียงลำดับ เก็บข้อมูลซ้ำไม่ได้

### Set

```
public void demoHashSet() {  
  
    HashSet<MyDate> myDates = new HashSet<MyDate>();  
    MyDate date1 = new MyDate();  
    date1.setYear(2018);  
  
    MyDate date2 = new MyDate();  
    date2.setYear(2019);  
    MyDate date3 = date2;  
  
    myDates.add(date1);  
    myDates.add(date2);  
    myDates.add(date3);  
  
    for (MyDate myDate : myDates) {  
        System.out.println(myDate.getYear());  
    }  
}
```

ใส่ date3 เข้าไปใน Set ได้หรือไม่.....

ใส่เข้าไปไม่ได้ เนื่องจาก date3 ซ้ำไปที่ MyDate เดียวกับ date2

### การนำข้อมูลออกจาก Set

```
myDates.remove(date2);  
for (MyDate myDate : myDates) {  
    System.out.println(myDate.getYear());  
}
```



# Array and Collections Framework

## 2. List เก็บข้อมูลแบบเรียงลำดับ เก็บข้อมูลซ้ำได้

```

List
public void demoArrayList() {

    ArrayList<MyDate> myDates = new ArrayList<MyDate>();
    MyDate date1 = new MyDate();
    date1.setYear(2018);

    MyDate date2 = new MyDate();
    date2.setYear(2019);
    MyDate date3 = date2;

    myDates.add(date1);
    myDates.add(date2);
    myDates.add(date3);

    for (MyDate myDate : myDates) {
        System.out.println(myDate.getYear());
    }
}

```

ใส่ date3 เข้าไปใน List ได้หรือไม่.....

ใส่เข้าไปได้ เนื่องจาก List สามารถเก็บข้อมูลซ้ำได้

### การนำข้อมูลออก List

```

myDates.remove(date3);
myDates.remove(1);

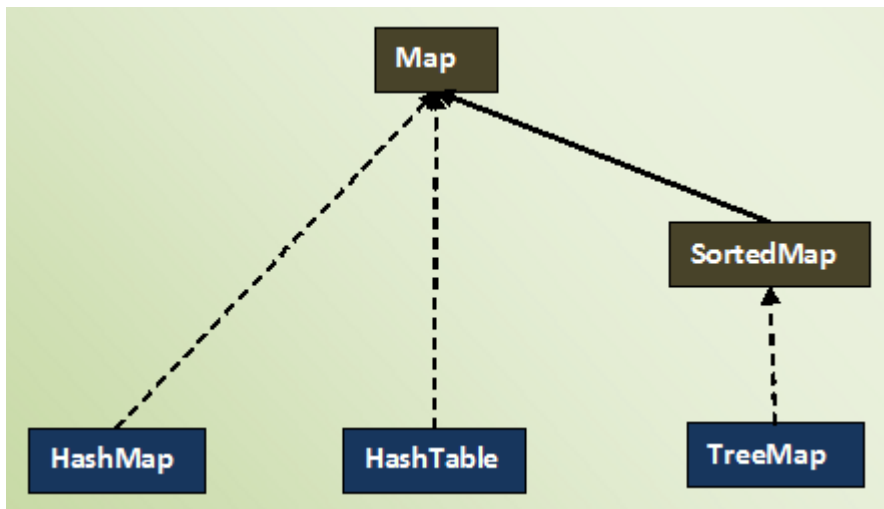
for (MyDate myDate : myDates) {
    System.out.println(myDate.getYear());
}

```



# Array and Collections Framework

## 3. Map เป็นการเก็บข้อมูลในรูปแบบของ Key คู่กับ Value



Map

```

public void demoMap() {

    HashMap<Integer,MyDate> myDates = new HashMap<Integer,MyDate>();
    MyDate date1 = new MyDate();
    date1.setYear(2018);

    MyDate date2 = new MyDate();
    date2.setYear(2019);
    MyDate date3 = date2;

    myDates.put(1,date1);
    myDates.put(2,date2);
    myDates.put(3,date3);
    myDates.put(4,date1);
    for(Entry<Integer, MyDate> myDate : myDates.entrySet()) {
        System.out.println("key [" + myDate.getKey() + "] year [" + myDate.getValue().getYear() + "]");
    }

    MyDate myDate = myDates.get(2);
    System.out.println("year [" + myDate.getYear() + "]");
}
  
```

# Exception Handling

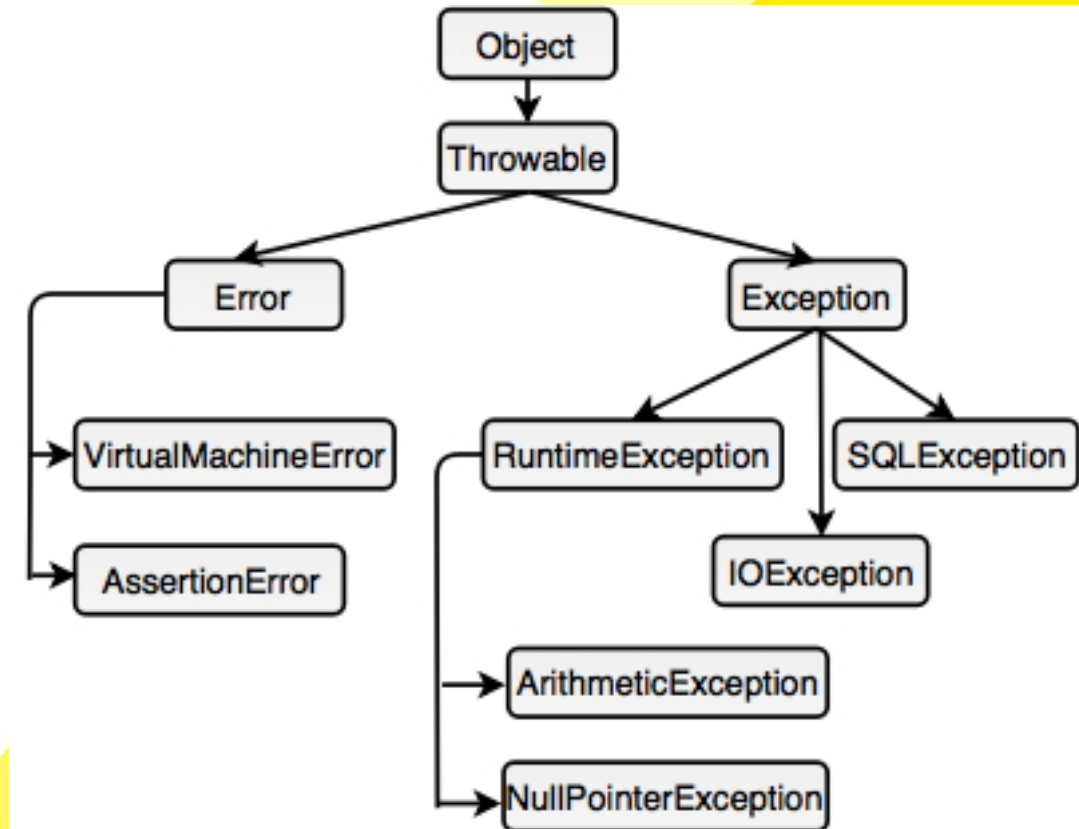
- **Exception Handling** เป็นการจัดการข้อผิดพลาดที่เกิดขึ้น ซึ่งเรามักจะต้องดักจับข้อผิดพลาดที่เกิดขึ้นนั้นไว้ และกระทำอะไรบางอย่างกับความผิดพลาดนั้นๆ โดยข้อผิดพลาดนั้นจะมีทั้งแบบร้ายแรง และ แบบที่พอจะจัดการได้

Class หลักของข้อผิดพลาดคือ Throwable โดย จะมี

Class ที่สืบทอดโดยตรง 2 Class คือ

1. Error คือข้อผิดพลาดร้ายแรงซึ่งอาจจะทำให้ JVM หยุดทำงานลงได้เราจึงไม่ดักจับ Error
2. Exception คือข้อผิดพลาดแบบที่พอจะจัดการได้ ในส่วนของโปรแกรมเราจึงมีการดักจับข้อผิดพลาดประเภทนี้ และทำการบางอย่างหลังจากดักจับข้อผิดพลาดนั้นๆได้

จาก Diagram เป็นเพียงตัวอย่างบางส่วนของข้อผิดพลาด



# Exception Handling

Exception นั้นแบ่งออกเป็น 2 ประเภทคือ

1. **Un-checked Exception** คือ exception ที่จะจัดการหรือไม่จัดการก็ได้ อยู่ในตระกูล RuntimeException
2. **Checked Exception** คือ exception ที่จะต้องจัดการ หากไม่จัดการ compiler จะไม่ยอมให้ compile ผ่าน โดยเป็น exception อื่นๆที่ไม่ได้อยู่ในตระกูล RuntimeException

การกระทำกับ Exception นั้นจะมี statement blocks หลักๆอยู่ 3 statement

1. **try block** คือขั้นแรกในการกำหนด scope ในการจัดการข้อผิดพลาด
2. **catch blocks** เมื่อเกิดข้อผิดพลาดจาก try block แล้วโปรแกรมก็จะมาทำงานต่อที่ catch block ที่กำหนด โดยใน 1 try block สามารถมี catch block ได้มากกว่า 1 block
3. **finally block** เป็น block สุดท้ายที่ต้องถูกทำเสมอ เพื่อกันความผิดพลาดเนื่องจากบาง statement จำเป็นจะต้องถูกทำเสมอไม่ว่ากรณีใดๆก็ตาม เช่นกรณีที่เกิด Exception และไม่มีการกำหนด finally block ไว้ statement ต่างๆที่อยู่ต่อบรรทัดที่เกิด Exception จะไม่ถูกทำไปด้วย ดังนั้นบาง statement จึงจำเป็นต้องนำมาไว้ใน finally block

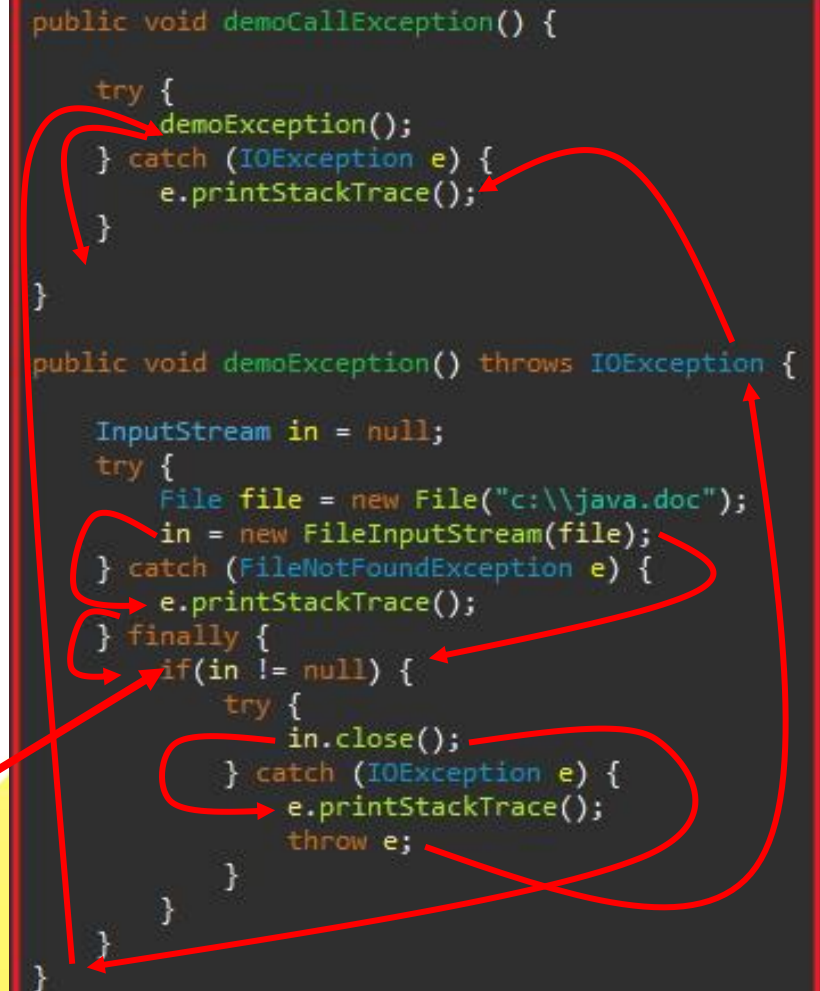
# Exception Handling

นอกจาก try catch finally แล้วเรายังมีคำสั่ง throw และ throws ที่ใช้ในการโยน Exception ที่เกิดขึ้นไปให้ method ที่เรียกใช้งาน นำ Exception ที่เกิดขึ้นไปดำเนินงานต่อด้วย จากตัวอย่างด้านล่างเป็นการดักจับ Exception โดยหากไม่มีการดักจับ compiler จะไม่ยอมให้เรา compile ผ่าน ในตัวอย่างนี้ไม่มีการดักจับ RuntimeException เนื่องจากว่าในการเขียนโปรแกรมนั้นมีโอกาสเกิด RuntimeException ได้มากมาย ดังนั้นทางที่ดีที่สุดคือ ต้องเขียนโปรแกรมให้ถูกต้องและรัดกุม

ตัวอย่างการเขียนโปรแกรม เพื่อไม่ให้มีโอกาสเกิด Exception ที่ชื่อว่า NullPointerException

```
public void demoCallException() {
    try {
        demoException();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public void demoException() throws IOException {
    InputStream in = null;
    try {
        File file = new File("c:\\java.doc");
        in = new FileInputStream(file);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } finally {
        if(in != null) {
            try {
                in.close();
            } catch (IOException e) {
                e.printStackTrace();
                throw e;
            }
        }
    }
}
```





The diagram illustrates the flow of exception handling between two methods. Red arrows show the following sequence: 1. An arrow from the `demoCallException()` method to the `try` block of `demoException()`. 2. An arrow from the `catch (IOException e)` block of `demoException()` back to the `demoCallException()` method. 3. An arrow from the `try` block of `demoException()` to the `finally` block. 4. An arrow from the `finally` block back to the `try` block. 5. An arrow from the `throw e;` statement in the nested `catch` block of the `finally` block back to the `catch (IOException e)` block of the `demoException()` method.



# Java Database Connectivity (JDBC)

JDBC คือ ชุดคำสั่งที่ใช้ในการกระทำระหว่าง Java กับ Database โดยผู้ผลิต Software Database ของแต่ละค่ายต้องทำการ implement JDBC Driver ตามข้อกำหนด เพื่อให้สามารถทำงานกับ Database ของตัวเองได้ ดังนั้นหากต้องการ เชื่อมต่อกับ Oracle ก็จำเป็นต้องใช้ Oracle JDBC Driver หากเชื่อมต่อกับ MySQL ก็จำเป็นต้องใช้ MySQL JDBC Driver โดย JDBC Driver จะอยู่ในรูปแบบของ Jar File

## ตัวอย่าง MySQL JDBC Driver

▼  mysql-connector  
 mysql-connector-java-5.1.40-bin.jar

## ตัวอย่าง Oracle JDBC Driver

▼  lib  
 ojdbc6-11.2.0.4.jar

การกระทำใดๆ กับ Resource ของ Database จำเป็นต้องคืน Resource นั้นๆกลับให้ Database เสมอ เพื่อป้องกันไม่ให้เกิดการใช้ Resource จนเต็มและระบบงานอื่นๆไม่มี Resource ในการใช้งาน เช่น การ Close Connection , การ Close Statement , การ Close Resultset เป็นต้น

# Java Database Connectivity (JDBC)

## ตัวอย่างการเปิด Connection ไป Database MySQL

```
public Connection getConnect() {
    Connection con = null;
    try {
        Class.forName("com.mysql.jdbc.Driver");
        con = DriverManager.getConnection("jdbc:mysql://localhost:3306/mysql?user=root&password=root");
    } catch (Exception e) {
        e.printStackTrace();
    }
    return con;
}
```

Load Driver Class ของ MySQL

Database connection URL

เปิด Connection ไปที่ database

รูปแบบของ database connection URL

`jdbc:mysql://[host][:port]/[database][?propertyName1][=propertyValue1][&propertyName2][=propertyValue2]`



# Java Database Connectivity (JDBC)

ตัวอย่างการ Insert ข้อมูลลง Database  
 ในส่วนของการ Update ข้อมูล หรือ  
 Delete ข้อมูล ออกจาก Database ก็จะมี  
 กระบวนการทำงานเหมือนกับ  
 กระบวนการ Insert นี้เพียงแค่เปลี่ยน  
 SQL Command

UPDATE table\_name  
 SET column1 = value1, column2 = value2,  
 ...  
 WHERE condition;  
  
 DELETE FROM table\_name WHERE condition;

```
public void insertCompany() {
    Connection con = getConnect();
    String sqlInsert = "insert into company(name, open_date, emp_total) values(?,?,?)";
    PreparedStatement pstInsert = null;

    try {
        pstInsert = con.prepareStatement(sqlInsert);
        pstInsert.setString(1, "CDGS");

        long time = new Date().getTime();
        pstInsert.setDate(2, new java.sql.Date(time));

        pstInsert.setInt(3, 350);

        int result = pstInsert.executeUpdate();
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try {
            if(pstInsert != null) {
                pstInsert.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

        try {
            if(con != null) {
                con.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

เปิด Connection

สร้าง Statement Insert

เตรียม Statement สำหรับใส่ข้อมูล

ใส่ข้อมูลใน Statement ตามลำดับ

สั่งให้ทำงานตาม Statement

ปิด Resource ที่มีการใช้งานตามลำดับ



# Java Database Connectivity (JDBC)

ตัวอย่างการค้นหาข้อมูลจาก Database

```
try {
    if(rs != null) {
        rs.close();
    }
} catch (SQLException e) {
    e.printStackTrace();
}

try {
    if(st != null) {
        st.close();
    }
} catch (SQLException e) {
    e.printStackTrace();
}

try {
    if(con != null) {
        con.close();
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```

```
public List<String> getCompanyNames() {

    Connection con = getConnect(); เปิด Connection
    String sqlQuery = "select name from company"; สร้าง Statement Select
    Statement st = null;
    ResultSet rs = null;

    List<String> companyNames = new ArrayList<String>();
    try {
        st = con.createStatement(); สร้าง Statement สำหรับใช้งาน
        rs = st.executeQuery(sqlQuery); สั่งให้ Statement ทำงานตาม SQL Command
        while(rs.next()) {
            companyNames.add(rs.getString("name")); ตรวจสอบและดึงผลลัพธ์ที่ได้จากการ
            ทำงาน โดยดึงข้อมูลด้วยชื่อ Column
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {

        }

    }
    return companyNames;
}
```

ปิด Resource ที่มีการใช้งานตามลำดับ

# Reference

ขอขอบคุณ ข้อมูลและรูปภาพจากหนังสือและ website ต่างๆดังนี้

- <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html>
- <https://i0.wp.com/javaconceptoftheday.com/wp-content/uploads/2014/11/CollectionHierarchy.png?w=95%25>
- <https://www.tutorialride.com/images/core-java/exception-hierarchy.jpg>
- [https://en.wikipedia.org/wiki/Java\\_virtual\\_machine#/media/File:JvmSpec7.png](https://en.wikipedia.org/wiki/Java_virtual_machine#/media/File:JvmSpec7.png)
- <https://www.parthpatel.net/wp-content/uploads/2017/05/class-and-objects-php.jpg>
- <https://www.w3schools.com/sql/>
- หนังสือ JAVA PROGRAMMING Volume I ผู้แต่ง ดร.วีระศักดิ์ ชิงฉาวร



**THANK YOU**