PROGRAMMING FUNDAMENTAL

KMITL

Prachya Sangkharat (!CE)
Microsoft Learn Student Ambassadors
Department of Computer Science, KMITL

Pakorn Tantiwutthiphat (Prite)
Super AI Engineer SS3
Department of Computer Science, KMITL

KMITL

Method (2)

Method

KMITL

- กลุ่มของคำสั่ง จะทำงานก็ต่อเมื่อเรียกใช้กลุ่มคำสั่งนั้น
- Method ที่เห็นกันบ่อย ๆ ➔ main()
- ทุกโปรแกรมจะมองหา Method main ก่อนเพื่อรันโปรแกรม
- ส่วนประกอบของ Method

Access Modifier | Return Type | Name | Parameter

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

}
```

Return Type

KMITL

- ชนิดของข้อมูลที่จะคืนค่าเมื่อจบการทำงาน Method
- หากไม่มีการคืนค่า จะใช้เป็น void

ตัวอย่างการใช้ Return Type

KMITL

```
public static void main(String[] args) {
        System.out.println(sayPika()); //Pika!
}

static String sayPika(){
        return "Pika!";
}
```

- Return Type เป็น String ➔ ภายใน Method จำเป็นต้องมี return แล้วตามด้วย String ตามที่ประกาศใน return type
- return ถือเป็นการจบการทำงานของ method

Parameter

KMITL

- ตัวแปรที่จะรับเข้ามาทำงานใน Method
- ประกาศตัวแปร Parameter เหมือนประกาศตัวแปรทั่วไป
- เป็นแบบ Primitive หรือ Non-primitive ก็ได้
- มีกี่ตัวก็ได้

Parameter

KMITL

```
public static void main(String[] args) {
    System.out.println(plus(2, 3, 4));
}

static int plus(int a, int b, int c){
    int result = a + b + c;
    return result;
}
```

Argument (ตัวส่ง)

Parameter (ตัวรับ)

- ถ้าต้องการบวกเลข 2 ตัว?
- ถ้าต้องการบวกเลข 5 ตัว?
- ถ้าต้องการบวกเลขแค่ 4 ตัว?
- ...

Method Overloading

Method Overloading

KMITL

- Method ชื่อเดียวกัน แต่มีหลายรูปแบบ

เช่น ถ้าต้องการบวกเลข 2 ตัว?

ถ้าต้องการบวกเลข 5 ตัว?

ถ้าต้องการบวกเลขแค่ 4 ตัว?

```
static int plus(int a, int b){
        int result = a + b;
        return result;
}

static int plus(int a, int b, int c){
        int result = a + b + c;
        return result;
}

static int plus(int a, int b, int c, int d){
        int result = a + b + c + d;
        return result;
}
```

Local Variable

Local Variable

KMITL

- โปรแกรมจะรู้จักตัวแปรภายใน Block Scope เดียวกันเท่านั้น

```
public static void main(String[] args) {
        System.out.println(pikachuAttack); //error
}

static void pokemonBattle(){
        int pikachuAttack = 12;
        System.out.println(pikachuAttack);
}
```

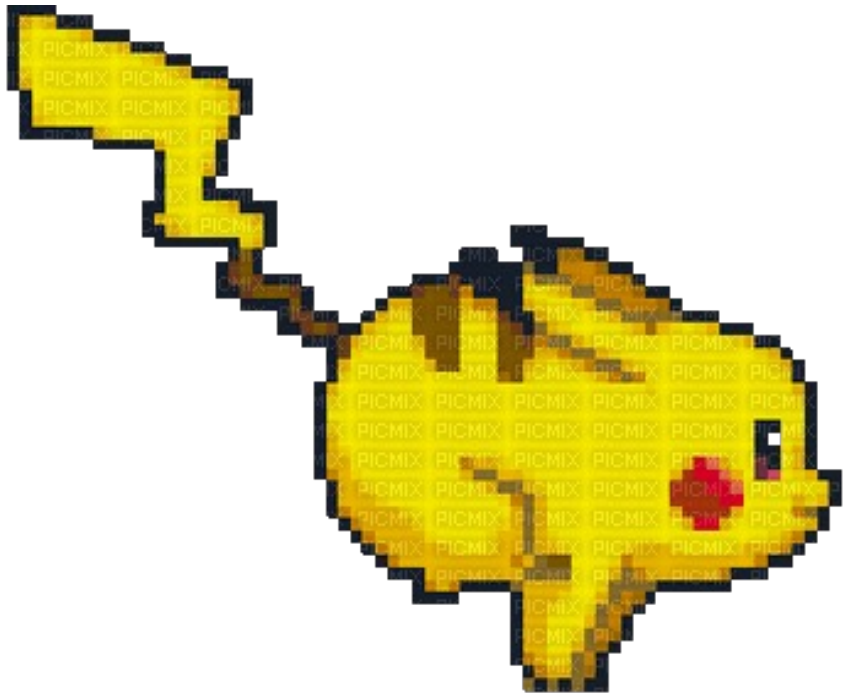
- ตัวแปร pikachuAttack รู้จักกันแค่ภายในพื้นที่สีเหลือง
- หากออกไปจาก Block Scope จะไม่รู้จักตัวแปรนี้

Object-Oriented Programming

OOP

KMITL

- มองทุกอย่างเป็นวัตถุก้อนหนึ่ง
- วัตถุจะมี**คุณสมบัติ(Attribute)** และ**พฤติกรรม(Method)**



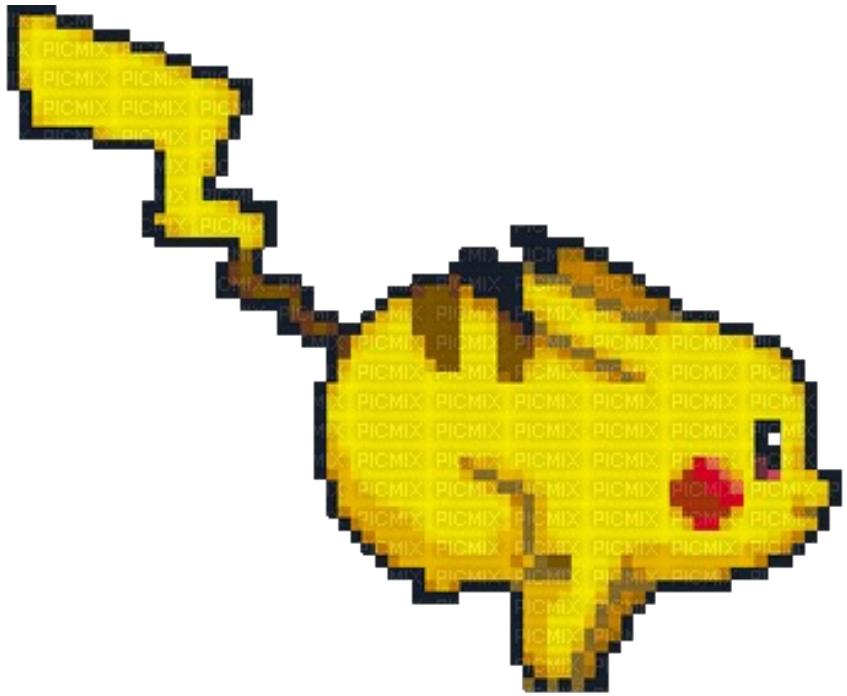
Pikachu

- คุณสมบัติ : มีสีเหลือง ตัวเล็ก ธาตุไฟฟ้า
- พฤติกรรม : เดิน วิ่ง ปล่อยไฟฟ้าแสนโวลต์ ใช้ท่า Iron Tail

OOP

KMITL

- มองทุกอย่างเป็นวัตถุก้อนหนึ่ง
- วัตถุจะมี**คุณสมบัติ(Attribute)** และ**พฤติกรรม(Method)**



Pikachu

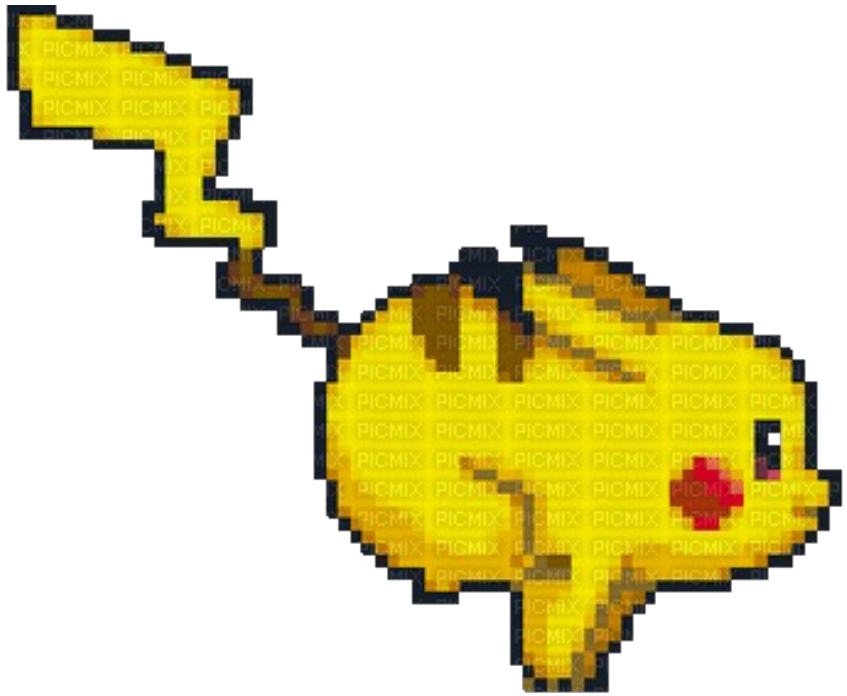
- คุณสมบัติ : มีสีเหลือง ตัวเล็ก ธาตุไฟฟ้า

```
{
        String color = “Yellow”;
        char size = ‘M’;
        String type = “Electric”
}
```

OOP

KMITL

- มองทุกอย่างเป็นวัตถุก้อนหนึ่ง
- วัตถุจะมี**คุณสมบัติ(Attribute)** และ**พฤติกรรม(Method)**



Pikachu

- พฤติกรรม : เดิน วิ่ง ปล่อยไฟฟ้าแสนโวลต์
  ใช้ท่า Iron Tail

```
{
        void move( );
        void run( );
        void thunderbolt( );
        void ironTail( );                }
```

Class

Class

- Blueprint ของวัตถุ ซึ่งจะกำหนดว่าวัตถุใดที่สร้างจาก Blueprint นี้ ต้องมีคุณสมบัติ และพฤติกรรมอย่างไรบ้าง
- วิธีประกาศ Class

ชื่อ Class

```
class Pokemon{
}
```

Class

KMITL

- Class ต้องมี Attribute และ Method

```
class Pokemon{
    String name;
    String color = "Yellow";
    char size = 'M';
    String type = "Electric";

    void move(){
        System.out.println("Let's move, " + name);
    }

    void run(){
        System.out.println("Let's run," + name);
    }

    void thunderbolt(){
        System.out.println(name + ", 100,000 Volt Thunderbolt!!!");
    }
    void ironTail(){
        System.out.println(name + "Iron Tail!!!");
    }
}
```

Attribute

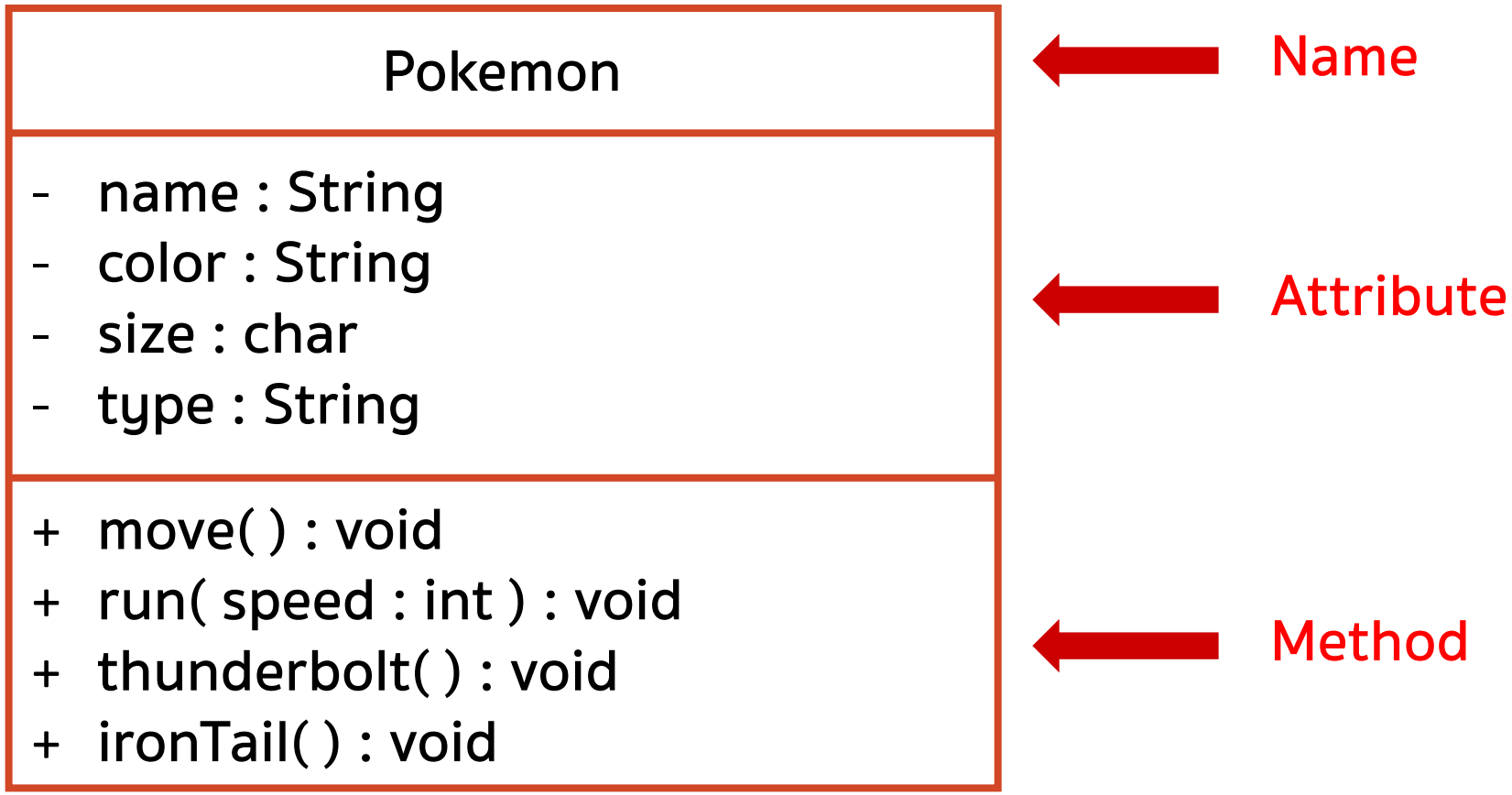
Method

Class Diagram

KMITL

- แผนภาพที่เป็นตัวแทนทำให้รู้ว่า Class นี้มี Attribute หรือ Method อย่างไรบ้าง และแต่ละ Class มีความสัมพันธ์กันอย่างไร

- \+ คือ public
- \- คือ private



Instance

- เมื่อมี Class แล้ว ต้องสร้าง Object ขึ้นมา

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

    Pokemon pikachu = new Pokemon();

}
```

ชื่อ Class — ชื่อ Object — ชื่อ Class

ในเรื่องนี้ จะเป็นชื่อ Class

แต่ในเรื่องต่อไป

อาจเป็นอย่างอื่น

การเข้าถึง Attribute

KMITL

- object_name.attribute ➔ อ่าน attribute
- object_name.attribute = value ➔ เขียน attribute

```
public static void main(String[] args) {
    Pokemon pikachu = new Pokemon();
    pikachu.name = "Pikachu";
    pikachu.move();
}
```

Constructor

- เบื่อ Pikachu แล้วอยากสร้างตัวอื่น?
- ทำไมไม่ใช้วิธีเดิม?

```
public static void main(String[] args) {
   Pokemon arceus = new Pokemon(“Arceus”,
   “White”, ‘L’, “Normal”);
}
```

Constructor

KMITL

- ที่เขียนไปก่อนหน้านี้คืออะไร?
- ได้ Pokemon ตัวใหม่แล้ว!

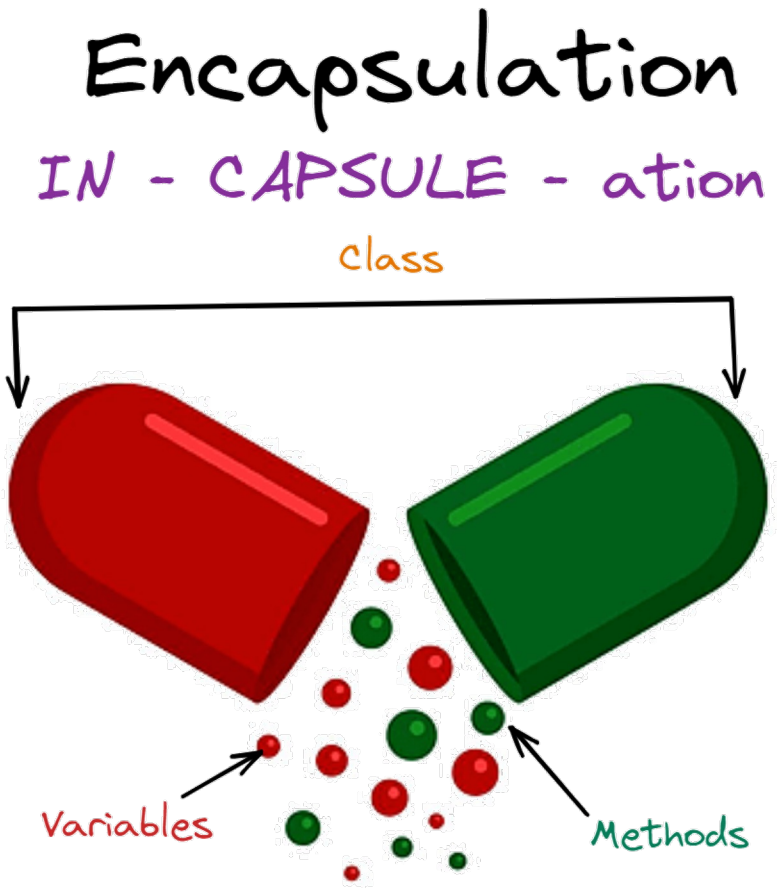
```
Pokemon (String inputName, String inputColor, Char inputSize,
String inputType){
        name = inputName;
        color = inputColor;
        size = inputSize;
        type = inputType;
}
```

```
String name;
String color
char size;
String type;
```

Encapsulation

Encapsulation

- เป็นการห่อหุ้มเพื่อซ่อนข้อมูล
- ป้องกันการเข้าถึงข้อมูลจากภายนอก



KMITL

private

- ใช้ private เพื่อห่อหุ้ม Attribute หรือ Method
- ทำให้สามารถเข้าถึงได้ภายใน Class เดียวกันเท่านั้น
- Attribute ทุกตัวควรห่อหุ้มไว้ เป็นการสร้างความปลอดภัยโดยเบื้องต้น

```
private String type = "Electric"; //private attribute

private void move(){
      System.out.println("Let's move, " + name);
} //private method
```

การเข้าถึงตัวแปรที่ถูกห่อหุ้มจาก Class อื่น

KMITL

`pokemon.type` ✗

- เข้าถึงโดยตรงไม่ได้
- ต้องพึ่งพา getter และ setter

getter / setter

KMITL

- getter เป็น Method สำหรับเข้าถึง Attribute เพื่อ**อ่าน**ข้อมูล

  ชื่อ Method : get ตามด้วยชื่อ Attribute ที่ขึ้นต้นตัวพิมพ์ใหญ่

- setter เป็น Method สำหรับเข้าถึง Attribute เพื่อ**เขียน**ข้อมูล

  ชื่อ Method : set ตามด้วยชื่อ Attribute ที่ขึ้นต้นตัวพิมพ์ใหญ่

getter / setter ของ name

KMITL

```
public String getName() {
      return name;
}

public void setName(String name) {
      this.name = name;
}
```

getter / setter ของ type


```
public String getType() {
      return type;
}

public void setType(String type) {
      this.type = type;
}
```

getter / setter ของ type


```
public String getType() {
      return type;
}

public void setType(String type) {
      this.type = type;
}
```

this

- เป็น Keyword ที่อ้างถึง attribute หรือ method ของ Class

```
public void setType(String type) {
    this.type = type;
}
```

- this.type หมายถึง type ที่เป็น attribute
- type หมายถึง type ที่เป็น parameter
- ถ้ากำหนด type = type มันจะ error เพราะไม่รู้ว่า type ตัวใด

Inheritance

Inheritance

- ต้องการสร้าง Class Pikachu, Lizadon, Nidoran
- ทั้ง 3 ล้วนเป็นโปเกมอน
- มีคุณสมบัติ และพฤติกรรม ตามแบบโปเกมอนพื้นฐาน

Inheritance

KMITL

- ต้องการสร้าง Class Pikachu, Lizadon, Nidoran
- ทั้ง 3 ล้วนเป็นโปเกมอน
- มีคุณสมบัติ และพฤติกรรม ตามแบบโปเกมอนพื้นฐาน

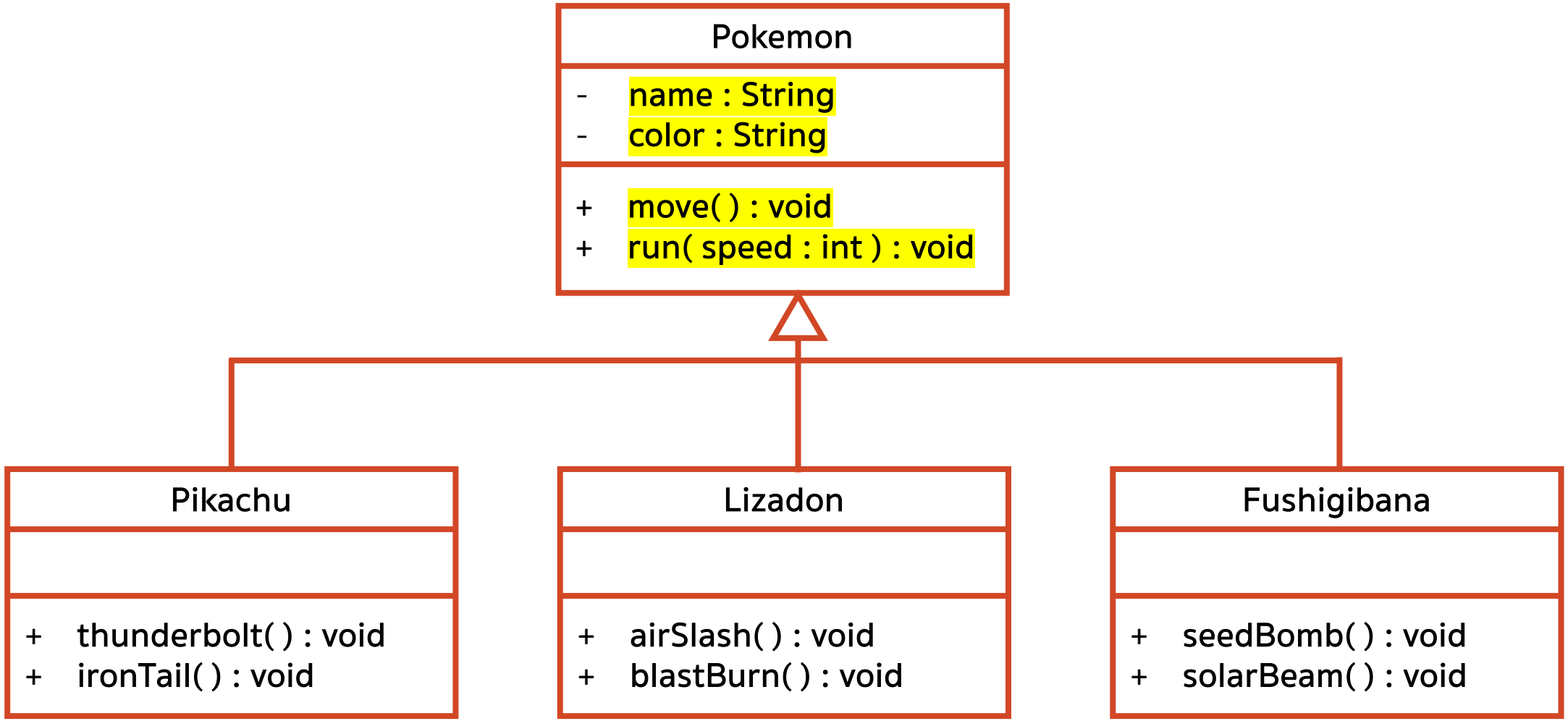
Pikachu
- name : String - color : String
+ move( ) : void + run(speed : int) : void + thunderbolt( ) : void + ironTail( ) : void

Lizadon
- name : String - color : String
+ move( ) : void + run(speed : int) : void + airSlash( ) : void + blastBurn( ) : void

Fushigibana
- name : String - color : String
+ move( ) : void + run(speed : int) : void + seedBomb( ) : void + solarBeam( ) : void

Inheritance

KMITL



Inheritance

KMITL

- ใช้ Keyword : **extend**

```
class Pikachu extends Pokemon{

}
```

Inheritance

- อยากให้ Pikachu รู้จัก Attribute name ด้วย
- name อยู่ที่คลาสแม่ ต้องกำหนด name เป็น protected เพื่อให้คลาสลูกเห็น

```
class PokemonMain{
        protected String name;
        …
}

class Pikachu extends PokemonMain{
    public static void main(String[] args) {
        Pikachu pikachu = new Pikachu();
        pikachu.name = "Pikachuuuu";
        System.out.println(pikachu.name); //Pikachuuuu
    }
}
```

Polymorphism

Polymorphism

- Pokemon แต่ละตัว มีท่าการโจมตีที่ต่างกัน
- ถ้าคลาสแม่มี attack( ) แบบหนึ่ง ลูกก็มี attack( ) เหมือนกัน แต่อาจไม่เหมือนแม่
- Polymorphism คือการมีหลายรูปแบบ
- จากตัวอย่างข้างบนก็จะมี attack( ) หลายรูปแบบ

Method Overriding

- การที่ class ลูก เอา Method แม่มาเขียนคำสั่งใหม่ โดยใช้ชื่อเดิม และ Parameter เหมือนเดิม

```
class PokemonMain2{
    public void attack(int damage){
        System.out.println("Punch");
    }
}

class Pikachu2 extends PokemonMain2{

    @Override
    public void attack(int damage){
        System.out.println("100,000 Volt Thunderbolt!!!");
    }
}
```

Method Overriding

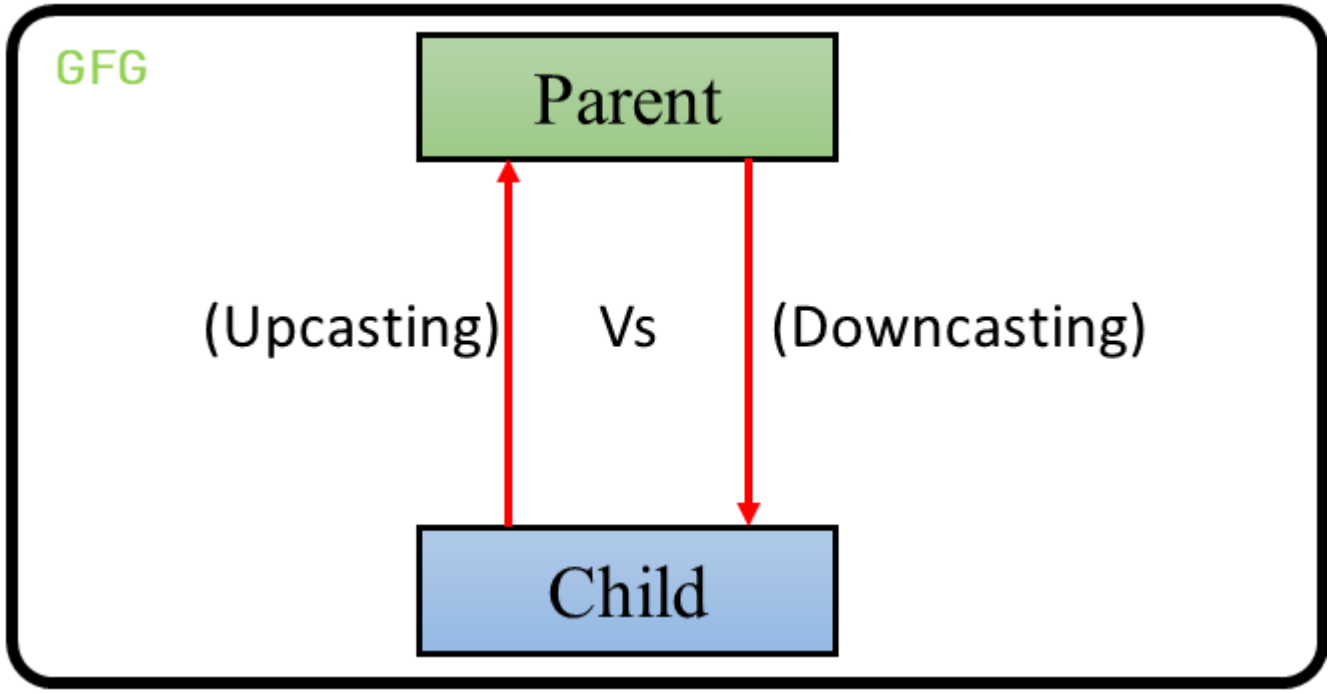
- การที่ class ลูก เอา Method แม่มาเขียนคำสั่งใหม่ โดยใช้ชื่อเดิม และ Parameter เหมือนเดิม

```
class Fushigibana extends PokemonMain2{

    @Override
    public void attack(int damage){
        System.out.println("solarBeam");
    }
}

class Lizadon extends PokemonMain2{

    @Override
    public void attack(int damageLizadon){
        System.out.println("blastBurn");
    }
}
```

Upcasting / Downcasting

KMITL

- Upcasting คือ การที่ cast จากลูกเป็นแม่
- Downcasting คือ การที่ cast จากแม่เป็นลูก



https://media.geeksforgeeks.org/wp-content/uploads/20200505231745/Upcasting-Vs-Downcasting.png

Upcasting / Downcasting


```java
class PokemonMain2 {
    int hp = 100;
    public void attack(int damage) {
        System.out.println("Punch");
    }
}

class Pikachu2 extends PokemonMain2 {
    int hp = 50;

    @Override
    public void attack(int damage) {
        System.out.println("100,000 Volt Thunderbolt!!!");
    }

}
```

Upcasting / Downcasting


```java
// Upcasting
PokemonMain2 pikachu = new Pikachu2();
pikachu.attack(3); // 100,000 Volt Thunderbolt!!!
System.out.println(pikachu.hp);


//Downcasting
Pikachu2 pikachu2 = (Pikachu2)pikachu;
System.out.println("HP"+pikachu2.hp); // HP50
```

Abstraction

Abstraction

- นามธรรม คือการที่เรารู้ว่าสิ่งนี้มีไว้ทำอะไร มีคุณลักษณะอย่างไร ไม่จำเป็นต้องรู้กระบวนการทำงานด้านใน
- เราสามารถกำหนดได้ว่า Class นี้จะมีคุณสมบัติอย่างไรบ้าง และทำอะไรได้บ้าง โดยที่ไม่บอกว่า **ทำอย่างไร**

KMITL

To be continued in Lab 3

Question? Problem?