เขียนโปรแกรมเชิงวัตถุ
(Object Oriented Programming)

https://www.youtube.com/c/KongRuksiamOfficial/

https://www.facebook.com/KongRuksiamTutorial/

เขียนโปรแกรมภาษา C# สำหรับผู้เริ่มต้น [2021]

74 videos • 21,905 views • Last updated on Dec 16, 2021

KongRuksiam Official

SUBSCRIBE

1 

เขียนโปรแกรมภาษา C# | สำหรับผู้เริ่มต้น [Phase1]
KongRuksiam Official

2 

เขียนโปรแกรมภาษา C# | สำหรับผู้เริ่มต้น [Phase2]
KongRuksiam Official

3 

สอน C# เบื้องต้น [2021] ตอนที่ 1 - แนะนำเนื้อหา
KongRuksiam Official

4 

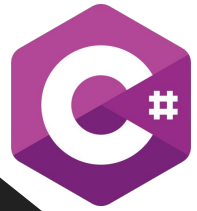
สอน C# เบื้องต้น [2021] ตอนที่ 2 - ติดตั้ง Visual Studio
KongRuksiam Official

5 

สอน C# เบื้องต้น [2021] ตอนที่ 3 - สร้างโปรเจกต์
KongRuksiam Official

https://www.youtube.com/c/KongRuksiamOfficial/

https://www.facebook.com/KongRuksiamTutorial/



OOP

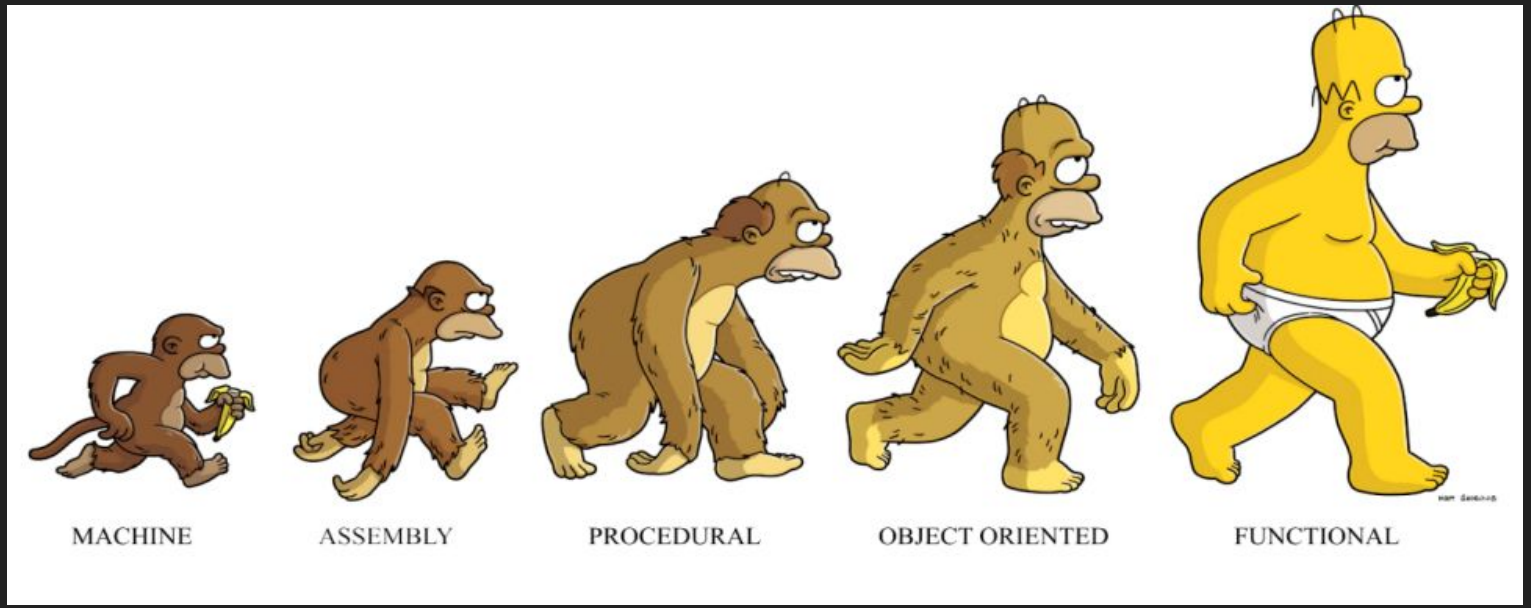
=

Object Oriented Programming

 https://www.youtube.com/c/KongRuksiamOfficial/

 https://www.facebook.com/KongRuksiamTutorial/

วิวัฒนาการของภาษาเชิงวัตถุ

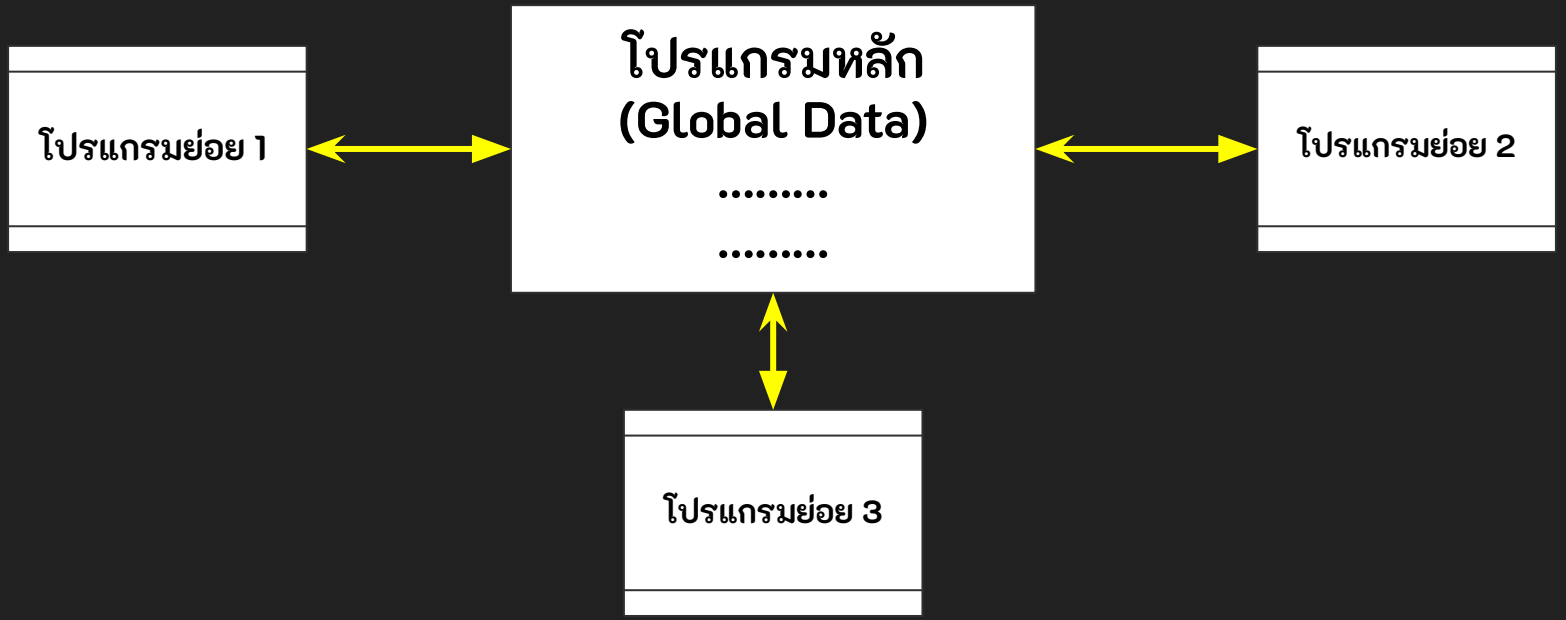


Procedural Programming

โครงสร้างโปรแกรมแบ่งออกเป็น 2 ส่วน คือ

1. โปรแกรมหลัก ที่มี Data เป็นส่วนประกอบ
2. โปรแกรมย่อย โดย Data ที่ประกาศอยู่ในโปรแกรมหลัก จะถูกเรียกใช้งานในโปรแกรมย่อยต่างๆภายในโปรแกรม ลักษณะของ Data ที่มีการประกาศใช้งานทั่วโปรแกรม ทั้งหมดจะเรียกว่า “Global Data”

Procedural Programming



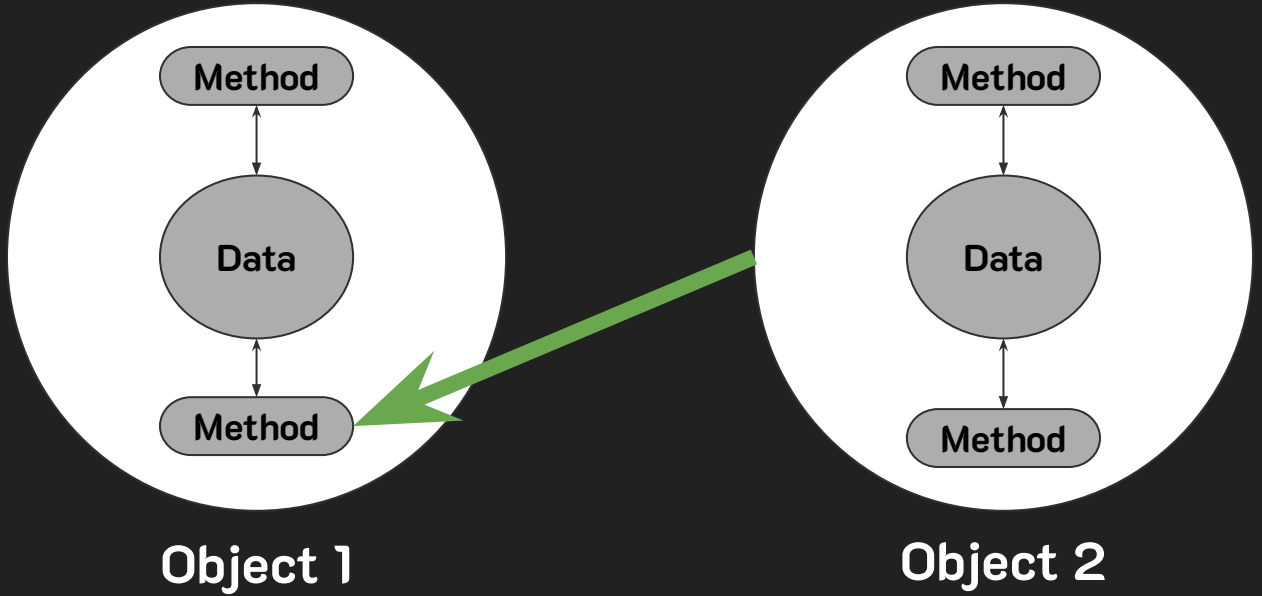
Procedural Programming

มีข้อจำกัด คือ โปรแกรมย่อยต่างเรียกใช้งานข้อมูลจากโปรแกรมหลักเดียวกัน อาจทำให้เกิดปัญหาจากการเปลี่ยนแปลงค่าของข้อมูล และส่งผลเสียต่อการควบคุมการเปลี่ยนแปลงข้อมูลของโปรแกรมซึ่งยากต่อการแก้ไขโปรแกรมในภายหลัง

Procedural Programming

สำหรับแนวทางการเขียนโปรแกรมเชิงวัตถุนั้นจะต่างออกไป Data หรือข้อมูลที่ถูกประกาศขึ้นมานั้นจะถูกใช้งานเฉพาะภายในแต่ละ Object กล่าวคือ 1 Object จะมี Data และการทำงานของโปรแกรม (Method) รวมอยู่ด้วยกัน

Object Oriented Programming



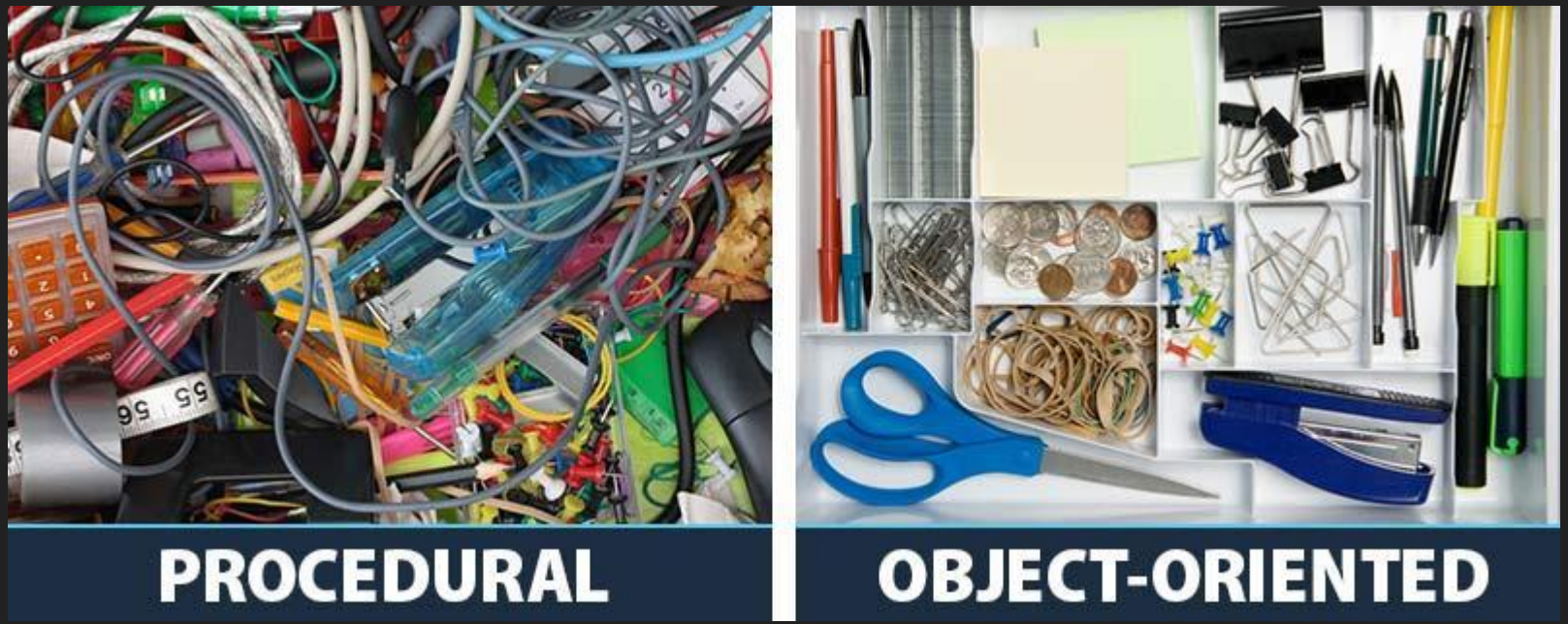
https://www.youtube.com/c/KongRuksiamOfficial/
https://www.facebook.com/KongRuksiamTutorial/

Object Oriented Programming

แนวคิดการเขียนโปรแกรมเชิงวัตถุนั้นจะจัด Data ไว้ในแต่ละ Object เพื่อปกป้องข้อมูลภายใน Object และลดปัญหาการเปลี่ยนแปลงข้อมูลภายใน Object โดยไม่ได้รับอนุญาต

Object หนึ่งจะสามารถเข้าถึงข้อมูลในอีก Object หนึ่งได้ก็ต่อเมื่อมีการใช้ Method ของ Object ที่เป็นเจ้าของข้อมูลเท่านั้น จึงส่งผลให้การแก้ไขโปรแกรมในภายหลังทำได้สะดวกยิ่งขึ้น

Procedural VS Object Oriented



https://www.youtube.com/c/KongRuksiamOfficial/

https://www.facebook.com/KongRuksiamTutorial/

Procedural VS Object Oriented

ภาษาเชิงกระบวนการ (Procedural Programming Language)

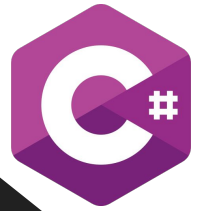
- โปรแกรมจะแบ่งออกเป็นส่วนย่อยๆ เรียกว่าโมดูล (Module)
- แต่ละโมดูลควรออกแบบให้มีการทำงานเพียง 1 งานเท่านั้น
- การออกแบบให้แต่ละโมดูลมีความเป็นอิสระต่อกันนั้นทำได้ยาก

ภาษาเชิงวัตถุ (Object Oriented Programming Language)

- การพัฒนาโปรแกรมเป็นการเลียนแบบการทำงานเชิงวัตถุ
- ออกแบบให้วัตถุมีความเป็นอิสระต่อกันทำได้ง่ายด้วยคุณสมบัติเชิงวัตถุ
- สามารถนำโปรแกรมกลับมาใช้ใหม่ (Reuse) ได้ดีกว่าภาษาเชิงกระบวนการ

Procedural VS Object Oriented

ภาษาเชิงกระบวนการ	ภาษาเชิงวัตถุ
กำหนดขั้นตอนการแก้ปัญหา	กำหนดปัญหาเป็นองค์ประกอบ (วัตถุ)
โปรแกรมและข้อมูลอยู่คนละส่วนกัน	เอาส่วนโปรแกรมและข้อมูลไว้ด้วยกัน
ออกแบบจากล่างขึ้นบน	ออกแบบเป็นวัตถุ
แก้ไขง่ายเพราะแต่ละส่วนไม่มีความสัมพันธ์กัน	การแก้ไขไม่กระทบส่วนอื่นๆของโปรแกรม เพราะวัตถุจะมีความสมบูรณ์ในตัวเอง



การเขียนโปรแกรมเชิงวัตถุ
(Object Oriented Programming)

เขียนโปรแกรมเชิงวัตถุ

คือ การเขียนโปรแกรมอีกรูปแบบหนึ่ง โดยมองสิ่งต่างๆ เป็นวัตถุ โดยในวัตถุจะมีคุณสมบัติและพฤติกรรมซึ่งมีมุมมองจากพื้นฐานความจริงในชีวิตประจำวัน

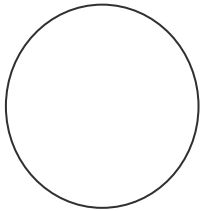
องค์ประกอบพื้นฐาน

คลาส (class) คือ ต้นแบบของวัตถุการจะสร้างวัตถุขึ้นมาได้จะต้องสร้างคลาสขึ้นมาเป็นโครงสร้างต้นแบบสำหรับวัตถุก่อนเสมอ

วัตถุหรือออบเจ็ค (object) คือ สิ่งที่ถูกสร้างจากคลาส ประกอบด้วยคุณสมบัติ 2 ประการ คือ คุณลักษณะ และ พฤติกรรม

องค์ประกอบพื้นฐาน

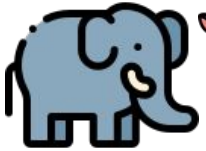
Class

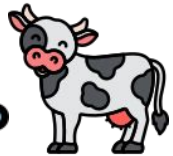


Animal

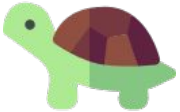
Object


สิงโต


ช้าง


วัว

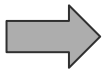

ไก่


เต่า

องค์ประกอบพื้นฐาน

Class

Employee



Accounting

Object

Programmer

Sale

องค์ประกอบพื้นฐาน

- **คุณลักษณะ (Field)** สิ่งที่บ่งบอกลักษณะทั่วไปของวัตถุ
- **พฤติกรรม (Behavior หรือ Method)** คือ พฤติกรรมทั่วไปของวัตถุที่สามารถกระทำได้

Animal

คุณลักษณะ (Field)

ชื่อ : ช้าง

สี : ฟ้าอ่อน

ประเภท : สัตว์บก

น้ำหนัก : 6 ตัน

จำนวนเท้า : 4 เท้า

พฤติกรรม (Method/Behavior)

- วิ่ง
- นอน
- ส่งเสียงร้อง

Animal

คุณลักษณะ (Field)

ชื่อ : นก

สี : เหลือง

ประเภท : สัตว์ปีก

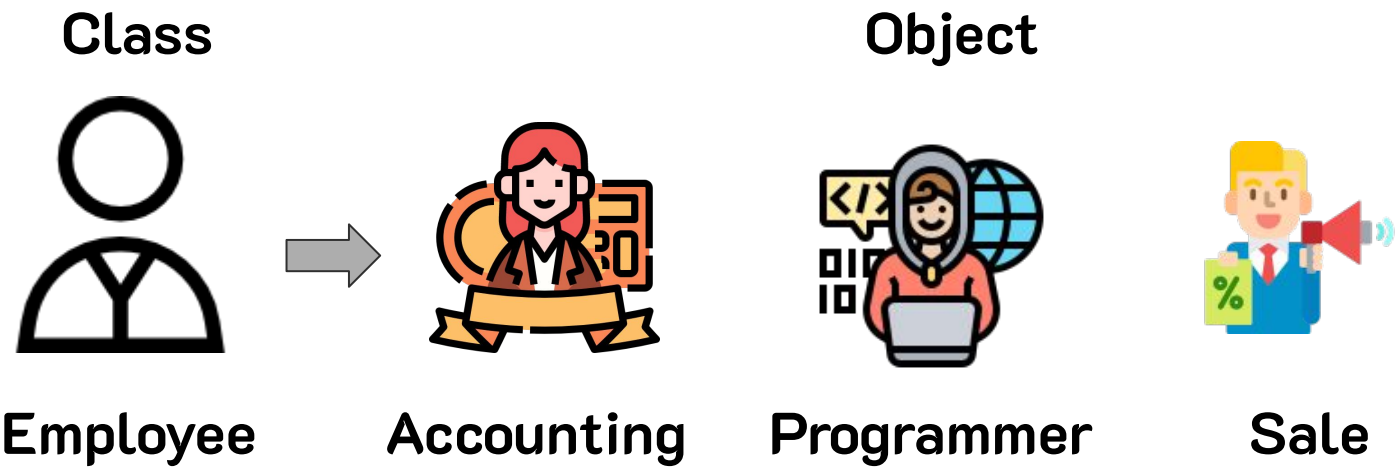
น้ำหนัก : 0.8 กิโลกรัม

จำนวนเท้า : 2 เท้า

พฤติกรรม (Method/Behavior)

- บิน
- เดิน
- ส่งเสียงร้อง

องค์ประกอบพื้นฐาน



Employee (พนักงาน)

Accounting	Programmer	Sale
Field - ชื่อ - เงินเดือน	**Field** - ชื่อ - เงินเดือน - ประสบการณ์ทำงาน	**Field** - ชื่อ - เงินเดือน - เขตพื้นที่รับผิดชอบ
Method - คำนวณเงินเดือน - แสดงรายละเอียด	**Method** - คำนวณเงินเดือน - ค่าล่วงเวลา - แสดงรายละเอียด	**Method** - คำนวณเงินเดือน - ค่าคอมมิสชั่น - แสดงรายละเอียด

สรุป

- Class - ต้นแบบของวัตถุ
- Object - สิ่งที่ถูกสร้างขึ้นมาจาก Class ประกอบด้วย
    - คุณลักษณะ (Field)
    - พฤติกรรม (Method)
- คุณสมบัติของการเขียนโปรแกรมเชิงวัตถุ
    - การห่อหุ้ม (Encapsulation)
    - การสืบทอด (Inheritance)
    - การพ้องรูป (Polymorphism)

ติดตั้งเครื่องมือพื้นฐาน

การสร้าง Class & Object

การสร้าง Class

```
class class_name{

    Field & Method

}
```

```
class Employee{

    Field & Method

}
```

ตัวอย่างการสร้าง Object

```
class_name obj = new class_name();
```

ตัวอย่างการสร้าง Object

```
Employee empl = new Employee();
```

กฎการตั้งชื่อ

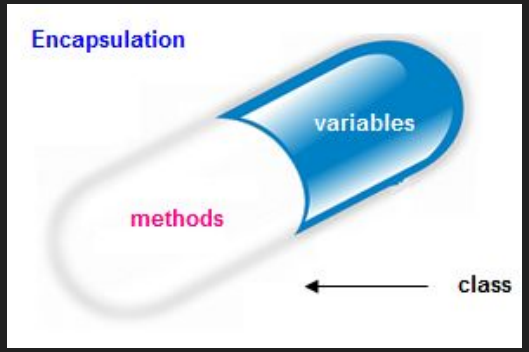
1. ชื่อไฟล์กับชื่อคลาสต้องเป็นชื่อเดียวกัน (.cs)
2. Class กำหนดให้ตัวอักษรตัวแรกเป็นตัวพิมพ์ใหญ่ที่เหลือเป็นพิมพ์เล็ก
   เช่น MyClass , Employee เป็นต้น
3. ชื่อ Object กำหนดเป็นตัวพิมพ์เล็กทั้งหมด
4. Field กำหนดเป็นตัวพิมพ์เล็ก เช่น name , age เป็นต้น
5. Property กำหนดให้ตัวอักษรตัวแรกเป็นตัวพิมพ์ใหญ่ที่เหลือเป็นพิมพ์เล็ก
   เช่น Name , Age เป็นต้น

การห่อหุ้ม (Encapsulation)

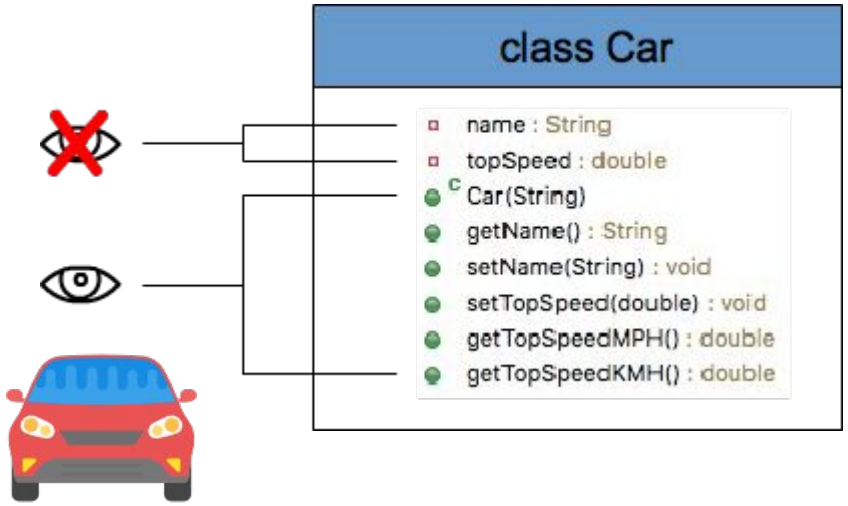
การห่อหุ้ม (Encapsulation)

1. เป็นกระบวนการซ่อนรายละเอียดการทำงานและข้อมูลไว้ภายในไม่ให้ภายนอกสามารถมองเห็นและสามารถทำการเปลี่ยนแปลงแก้ไขข้อมูลภายในได้ ซึ่งเป็นผลทำให้เกิดความเสียหายแก่ข้อมูล

2. สามารถสร้างความปลอดภัยให้แก่ข้อมูลได้ เนื่องจากข้อมูลจะถูกเข้าถึงจากผู้มีสิทธิ์เท่านั้น

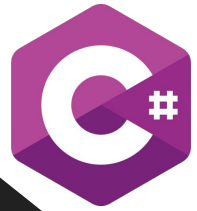


การห่อหุ้ม (Encapsulation)



https://www.youtube.com/c/KongRuksiamOfficial/

https://www.facebook.com/KongRuksiamTutorial/



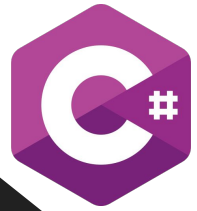
Access Modifier

Access Modifier

คือ ระดับในการเข้าถึง Class, Field, Method และอื่น ๆ ในภาษาเชิงวัตถุ มีประโยชน์อย่างมากในเรื่องของการกำหนดสิทธิในการเข้าใช้งาน การซ่อนข้อมูลภายในคลาส และอื่น ๆ

Access Modifier

- **Public** เป็นการประกาศระดับการเข้าถึงที่เข้มงวดน้อยที่สุดหรือกล่าวได้ว่าใครๆ ก็สามารถเข้าถึงและเรียกใช้งานได้
- **Protected** เป็นการประกาศระดับการเข้าถึงที่เกี่ยวข้องกับเรื่องการสืบทอด (Inheritance) ทำให้คลาสนั้นๆ สามารถเรียกใช้งานสมาชิกของคลาสที่ถูกกำหนดเป็น Protected ได้
- **Private** เป็นการประกาศระดับการเข้าถึงที่เข้มงวดที่สุด กล่าวคือ จะมีแต่คลาสของตัวมันเองเท่านั้นที่มีสิทธิ์ใช้งานได้



การสร้าง Field

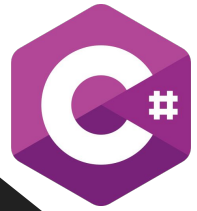
โครงสร้างคำสั่ง

```
class className {

    modifier type fieldName = value;

}
```

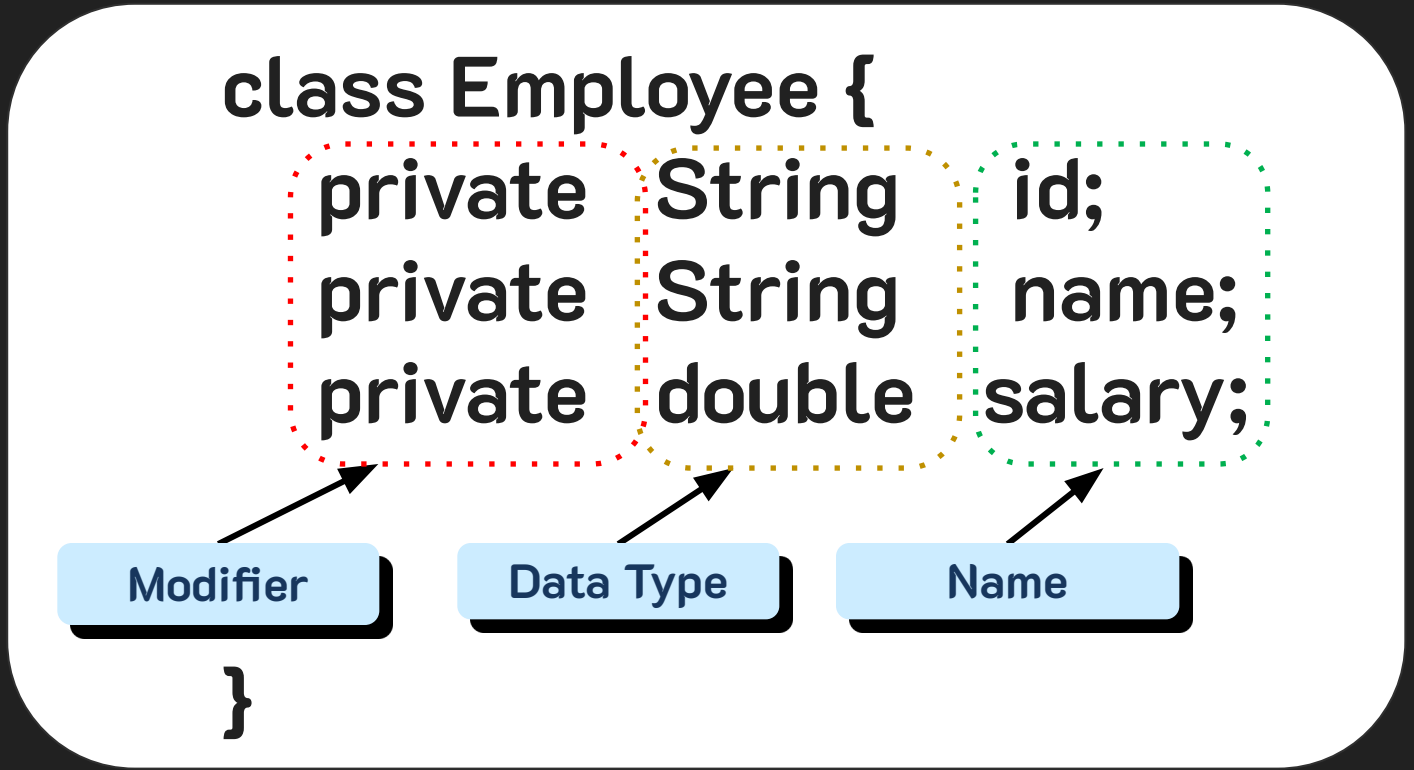
ตัวอย่าง



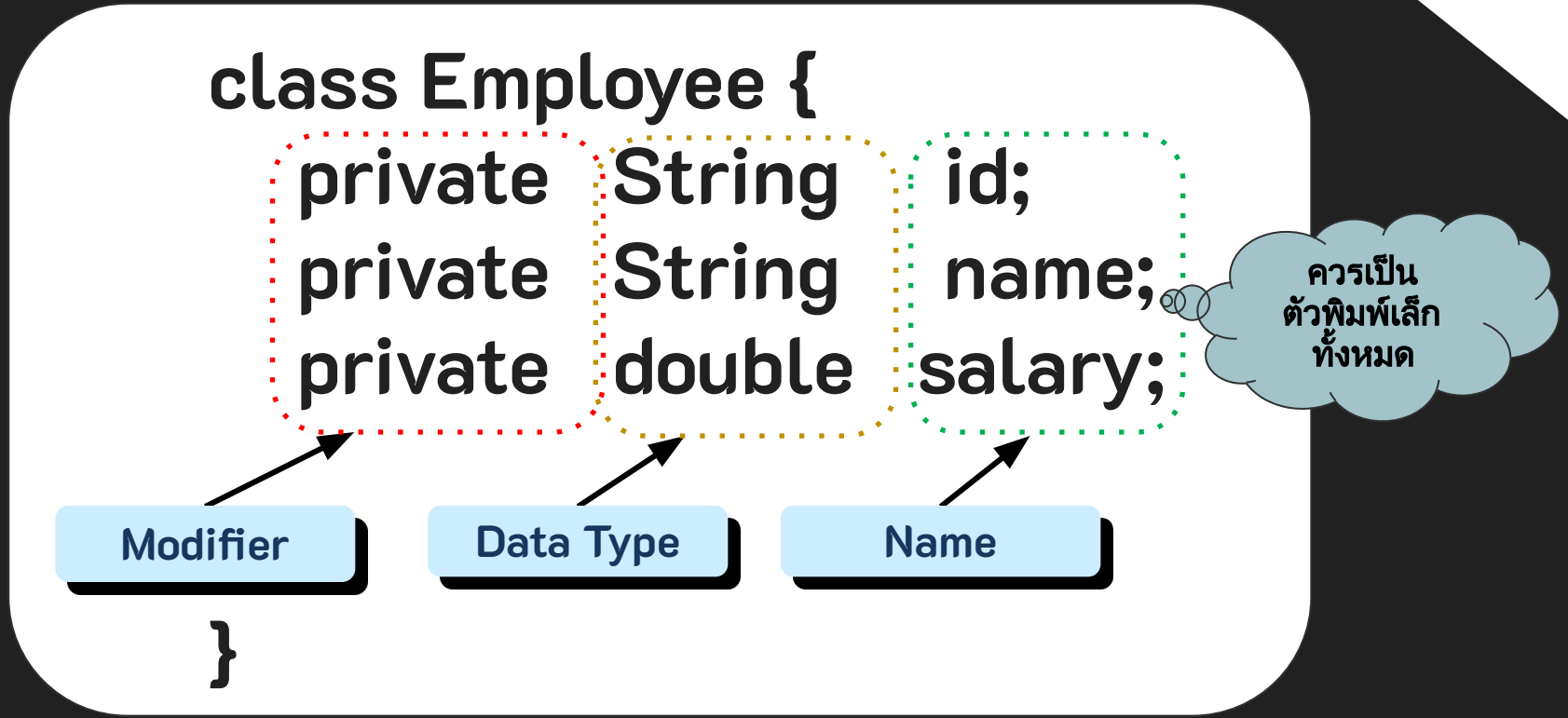
```
class Employee {
    private   String    id;
    private   String    name;
    private   double   salary;


}
```

ตัวอย่าง


class Employee {
private String id;
private String name;
private double salary;
Modifier
Data Type
Name
}

ตัวอย่าง


class Employee {
private String id;
private String name;
private double salary;
}
Modifier
Data Type
Name
ควรเป็นตัวพิมพ์เล็กทั้งหมด

การเรียกใช้งาน Field

เรียกใช้งานภายใน Class

- this.fieldName

เรียกใช้งานภายนอก Class

- obj_name.fieldName

Constructor

Constructor

เป็นฟังก์ชั่นพิเศษที่จะถูกใช้เมื่อสร้างวัตถุและทำงานอัตโนมัติเพียงครั้งเดียวในตอนเริ่มต้น

โครงสร้าง Constructor

```
public className([parameter]){
    //การทำงานด้านใน
}
```

รูปแบบ Constructor

- Default Constructor คือ Constructor เริ่มต้นที่มีอยู่ในทุกคลาส

- Parameter Constructor คือ Constructor ที่สามารถส่งพารามิเตอร์เข้าไปทำงานได้

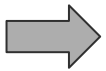
คีย์เวิร์ด this

การใช้คีย์เวิร์ด this จะเป็นตัวชี้หรือตัวที่บ่งบอกว่าตอนนี้เราทำงานกับวัตถุใด ให้บอกตัวตนของวัตถุนั้นๆ เช่น การกำหนดคุณสมบัติต่างๆ ในวัตถุ เป็นต้น

กำหนดค่าเริ่มต้น

Class

Employee

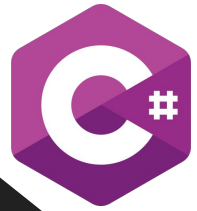


ชื่อ : เจน
เงินเดือน : xx,xxx

Object

ชื่อ : ก้อง
เงินเดือน : xx,xxx

ชื่อ : โจ้
เงินเดือน : xx,xxx



เมธอด (Method)

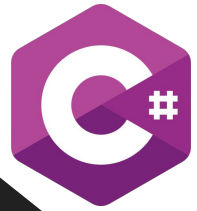
การสร้าง Method

โครงสร้างคำสั่ง

```
modifier return_type method_name (type parameter){
    // statement
}
```

การเรียกใช้งาน

```
obj.method_name()
```



Getter , Setter Method

การสร้าง Property

การสร้าง Property

Property คือ เมธอดพิเศษที่ช่วยให้เราเข้าถึงฟิลด์ได้ง่าย ผ่าน Accessor ที่ชื่อว่า get กับ set โดยสามารถกำหนดได้ว่า Property ที่สร้างขึ้นมานั้นทำงานกับ Field ใด โดยมีองค์ประกอบ 3 ส่วน คือ

- Get คือ ตัวช่วยสำหรับเรียกดูข้อมูล
- Set คือ ตัวช่วยสำหรับเขียนข้อมูล
- value คือ ค่าที่กำหนดตอนเรียกใช้ property

โครงสร้างคำสั่ง

```
public return_type PropertyName
  {
    get { return fieldName; }
    set { fieldName = value; }
  }
```

ตัวอย่างการสร้าง Property

```
public string Name {

    get { return name; }

    set { name = value; }

}
```

เขียนแบบลดรูป (Auto-Implemented Property)

```
    public string Name { get; set; }
```

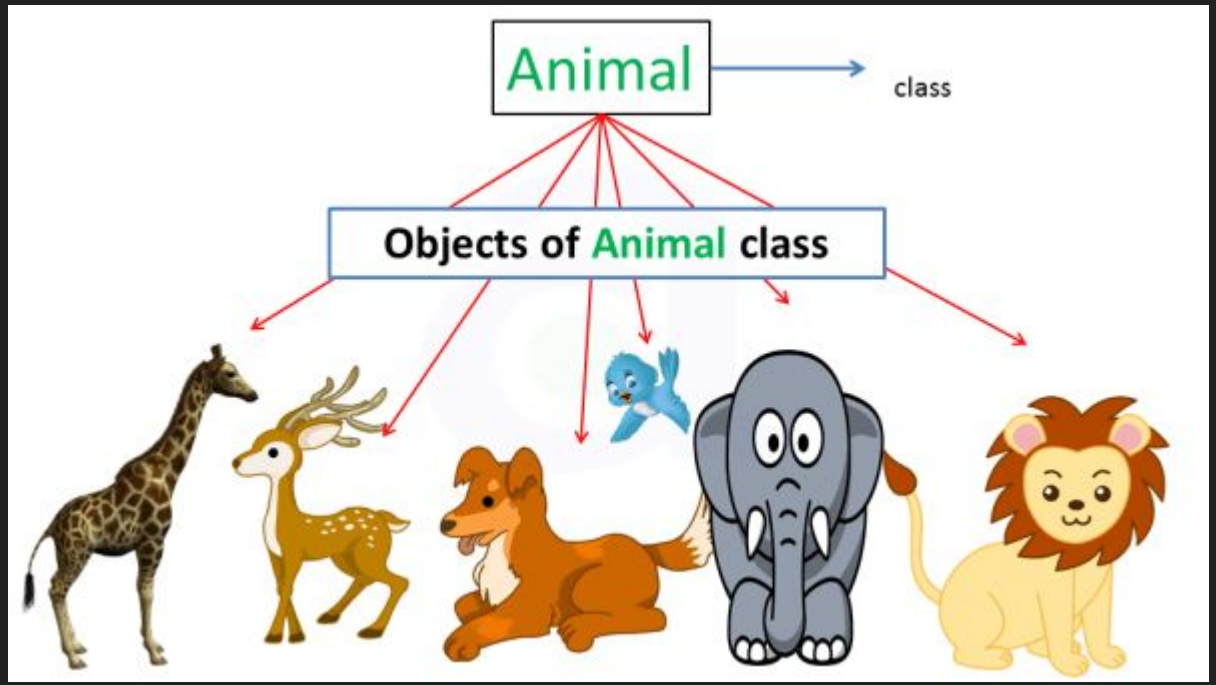
การสืบทอดคุณสมบัติ (Inheritance)

การสืบทอดคุณสมบัติ (Inheritance)

คือ ทำการสร้างสิ่งใหม่ขึ้นด้วยการสืบทอด หรือรับเอา (inherit) คุณสมบัติบางอย่างมาจากสิ่งเดิมที่มีอยู่แล้ว

ข้อดีของการ inheritance คือ สามารถนำสิ่งที่เคยสร้างขึ้นแล้วนำกลับมาใช้ใหม่ (re-use) ได้ ทำให้ช่วยประหยัดเวลาการทำงานลง เนื่องจากไม่ต้องเสียเวลาพัฒนาใหม่หมด

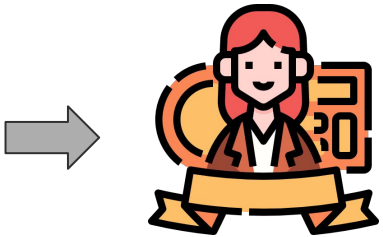
คลาสแม่ (Base class) คลาสลูก (Derived Class)



คุณสมบัติต่างๆจากแม่จะถูกถ่ายทอดไปยังลูก

Class

ยกเว้น Private Property & Private Method



Employee

Accounting

Programmer

Sale

การสืบทอดคุณสมบัติ (Inheritance)

คลาสแม่

```
class BaseClass{


// Property & Method
}
```

คลาสลูก

```
class DerivedClass : BaseClass{


// Property & Method
}
```

ตัวอย่าง

คลาสแม่

```
class Employee{


// Property & Method
}
```

คลาสลูก

```
class Accounting : Employee{


// Property & Method
}
```

Base

Base คืออะไร

เป็นคำสั่งสำหรับเรียกใช้งานเมื่อต้องการคุณสมบัติต่างๆที่ทำงานอยู่ในคลาสแม่ เช่น Constructor , Method , Property

Employee (พนักงาน)

Accounting	Programmer	Sale
Field - ชื่อ - เงินเดือน - เพศ	**Field** - ชื่อ - เงินเดือน - ประสบการณ์ทำงาน	**Field** - ชื่อ - เงินเดือน - เขตพื้นที่รับผิดชอบ
Method - คำนวณเงินเดือน - แสดงรายละเอียด	**Method** - คำนวณเงินเดือน - ค่าล่วงเวลา - แสดงรายละเอียด	**Method** - คำนวณเงินเดือน - ค่าคอมมิสชั่น - แสดงรายละเอียด

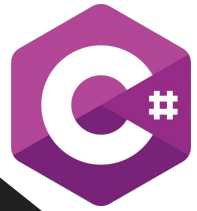
Employee (พนักงาน)

Accounting	Programmer	Sale
Field - ชื่อ - เงินเดือน - เพศ	**Field** - ชื่อ - เงินเดือน - ประสบการณ์ทำงาน	**Field** - ชื่อ - เงินเดือน - เขตพื้นที่รับผิดชอบ
Method - คำนวณเงินเดือน - แสดงรายละเอียด	**Method** - คำนวณเงินเดือน - ค่าล่วงเวลา - แสดงรายละเอียด	**Method** - คำนวณเงินเดือน - ค่าคอมมิสชั่น - แสดงรายละเอียด

Protected Access Modifier

Access Modifier

- **Public** เป็นการประกาศระดับการเข้าถึงที่เข้มงวดน้อยที่สุดหรือกล่าวได้ว่าใครๆ ก็สามารถเข้าถึงและเรียกใช้งานได้
- **Protected** เป็นการประกาศระดับการเข้าถึงที่เกี่ยวข้องกับเรื่องการสืบทอด (Inheritance) ทำให้คลาสนั้นๆ สามารถเรียกใช้งานสมาชิกของคลาสที่ถูกกำหนดเป็น Protected ได้
- **Private** เป็นการประกาศระดับการเข้าถึงที่เข้มงวดที่สุด กล่าวคือ จะมีแต่คลาสของตัวมันเองเท่านั้นที่มีสิทธิ์ใช้งานได้

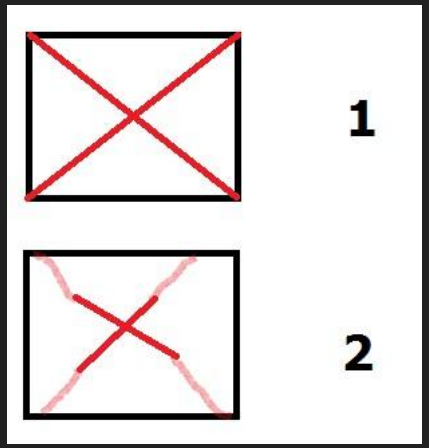


การพ้องรูป (Polymorphism)

การพ้องรูป (Polymorphism)

“ ข้อความเดียวกันแต่กระบวนการทำงานภายในแตกต่างกันนั้น เรียกว่า การพ้องรูป หรือ polymorphism ”

กา



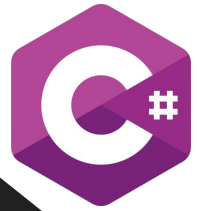
การพ้องรูป (Polymorphism)

คุณสมบัติการพ้องรูป คือ สามารถตอบสนองต่อ Method เดียวกันด้วยวิธีการที่ต่างกันและสามารถกำหนด object ได้หลายรูปแบบมีข้อดี คือ ทำให้โปรแกรมสามารถปรับเปลี่ยนหรือเพิ่มเติมได้ง่ายขึ้น

ดำ

https://www.youtube.com/c/KongRuksiamOfficial/

https://www.facebook.com/KongRuksiamTutorial/



Virtual & Override

Virtual & Override

จากคุณสมบัติการพ้องรูป (Polymorphism) สามารถกำหนดให้เมธอดของคลาสลูก (Derived Class) ที่มีชื่อเหมือนกับเมธอดของคลาสแม่ (Base class) ได้ แต่มีกระบวนการทำงานที่แตกต่างกัน หมายถึงเมธอดในคลาสลูก สามารถเขียนทับการทำงานของคลาสแม่ เพื่ออธิบายถึงความสัมพันธ์ของการสืบทอดคุณสมบัติ

Virtual & Override

คลาสแม่

```
class BaseClass{
    virtual method(){
        //การทำงานของคลาสแม่
    }
}
```

คลาสลูก

```
class DerivedClass : BaseClass{
    override method(){
        //การทำงานของคลาสลูก
    }
}
```

Virtual & Override

คลาสแม่

```
class BaseClass{
    virtual method(){
        //การทำงานของคลาสแม่
    }
}
```

คลาสลูก

```
class DerivedClass : BaseClass{
    override method(){
        //การทำงานของคลาสลูก
    }
}
```

Virtual & Override

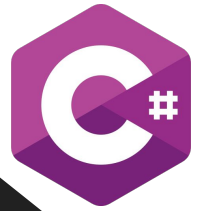
คลาสแม่

```
clas
    virtual method(){
        //การทำงานของคลาสแม่
    }
}
```

อนุญาตให้แก้ไขได้

คลาสลูก

```
class DerivedClass : BaseClass{
    override method(){
        //การทำงานของคลาสลูก
    }
}
```



Abstract
Class & Method

https://www.youtube.com/c/KongRuksiamOfficial/

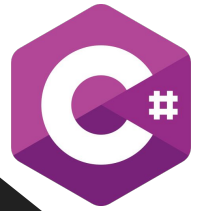
https://www.facebook.com/KongRuksiamTutorial/

Abstract คืออะไร

- คีย์เวิร์ด abstract ใช้กำหนดให้กับ คลาส หรือเมธอด สำหรับกำหนดโครงสร้างโดยไม่ระบุรายละเอียดการทำงานด้านใน
- กฎของ abstract คือ หากคลาสใดสืบทอดมาจาก abstract class คลาสนั้นจะต้องทำการระบุเมธอดทุกเมธอดที่เป็น abstract method ใน abstract class ไว้เสมอ **(ไม่กำหนดรายละเอียดก็ได้แต่จะต้องมีการเขียน abstract method ทุกเมธอดลงไปในคลาสนั้นด้วย)**

Employee (พนักงาน)

Accounting	Programmer	Sale
Field - ชื่อ - เงินเดือน - เพศ	**Field** - ชื่อ - เงินเดือน - ประสบการณ์ทำงาน	**Field** - ชื่อ - เงินเดือน - เขตพื้นที่รับผิดชอบ
Method - รายได้ต่อปี - แสดงรายละเอียด	**Method** - รายได้ต่อปี + ประสบการณ์ทำงาน - แสดงรายละเอียด	**Method** - รายได้ต่อปี + ค่าคอม - แสดงรายละเอียด



Static Field

Static Field

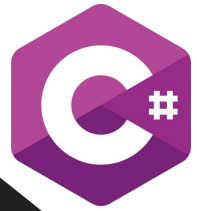
คือ Field ที่สามารถเรียกใช้งานได้โดยตรง ไม่ต้องเรียกผ่าน Object การสร้าง Static Field จะเหมือนกับการสร้าง Field โดยทั่วไปเพียงแค่เติม static นำหน้า Field

โครงสร้างคำสั่ง

```
modifier static type field = defaultvalue;
```

การเรียกใช้งาน

```
className.field
```



Static Method

Static Method

คือ Method ที่สามารถเรียกใช้งานได้โดยตรง ไม่ต้องเรียกผ่าน Object การสร้าง Static Method จะเหมือนกับการสร้าง Method โดยทั่วไปเพียงแค่เติม static นำหน้า

Static Method

โครงสร้างคำสั่ง

```
modifier static returntype method_name (parameter){
    //statement
}
```

การเรียกใช้งาน

```
className.method()
```

Sealed Class

Sealed คืออะไร

- sealed เป็นคีย์เวิร์ดหนึ่งในภาษา C# ทำให้คลาสนั้นไม่สามารถสืบทอดหรือมีคลาสลูกได้

อินเทอร์เฟซ (Interface)

อินเทอร์เฟซ (Interface)

อินเทอร์เฟซ (interface) มีหลักการคล้ายกับ abstract class คือ สร้างอินเทอร์เฟซขึ้นมาก็เพื่อกำหนดโครงสร้างของเมธอดที่จำเป็นใช้งานขึ้นมา แต่ยังไม่ได้กำหนดรายละเอียดการทำงานใดๆ ลงไปให้กับเมธอดนั้น (abstract method) เมธอดในอินเทอร์เฟซจึงเป็นเมธอดที่ว่างเปล่า ซึ่งในภายหลังจึงมีการกำหนดรายละเอียดของเมธอดเหล่านั้นลงไป โดยถูกกำหนดโดยคลาสที่เรียกใช้อินเทอร์เฟซนั้นๆ

การใช้งาน Interface

```
class name : baseClass , interface_N{

    methodN(){

        //รายละเอียดการทำงาน

    }

}
```

Interface กับ Abstract Class ต่างกันอย่างไร

- เมธอดใน abstract class ไม่เป็น abstract method ก็ได้ แต่เมธอดทุกเมธอดใน interface เป็น abstract method
- คลาสที่จะเรียกใช้งานเมธอดในอินเทอร์เฟซไม่จำเป็นต้องมีความสัมพันธ์ใดๆ กับอินเทอร์เฟซทั้งสิ้น

Interface กับ Abstract Class ต่างกันอย่างไร

- คลาสที่จะเรียกใช้งาน abstract method ใน abstract class จะต้องสืบทอดคุณสมบัติไปจาก abstract class นั้น แล้วจึงทำการสร้างเมธอดของตัวเองขึ้นมาให้มีชื่อเดียวกับ abstract method ใน abstract class โดยกำหนดรายละเอียดการทำงานให้กับ abstract method เหล่านั้นตามต้องการ