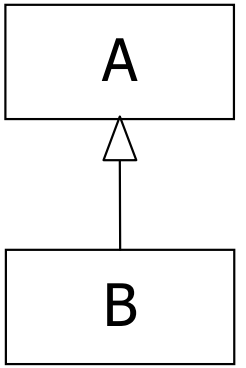
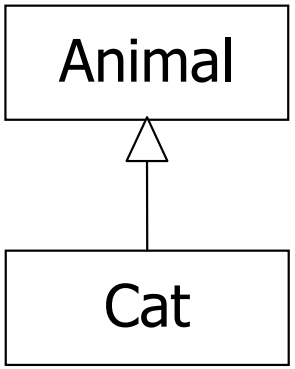
จาวา – Inheritance

สมชาย ประสิทธิ์จูตระกูล

Inheritance

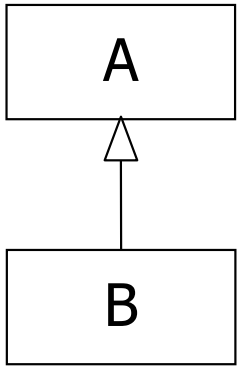
- เขียนคลาสใหม่ซึ่ง "ขยาย" ลักษณะจากคลาสเก่า (class B extends A)
- เรียกคลาส B ว่า inherits จากคลาส A
  - เรียก B ว่าเป็น subclass ของ A
  - เรียก A ว่าเป็น superclass ของ B
- ตีความได้ว่า B คือคลาสที่เป็นกรณีพิเศษของ A
  - ออบเจกต์ของ B “เป็น” A
  - แต่ออบเจกต์ของ A ไม่จำเป็นต้อง “เป็น” B
  - ตัวอย่างเช่น class Cat extends Animal
    - แมวทุกตัวเป็นสัตว์
    - แต่สัตว์ทุกตัวไม่ใช่แมว





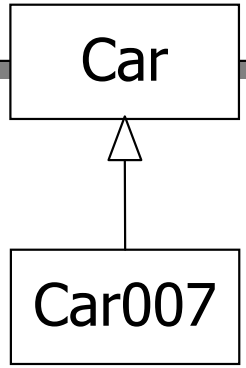
การรับมรดก

- class B extends A
  - B มีทุกๆ fields ของ A (โดยอัตโนมัติ)
  - B มีทุกๆ methods ของ A (โดยอัตโนมัติ)
  - B ไม่รับ constructors ใดๆ จาก A

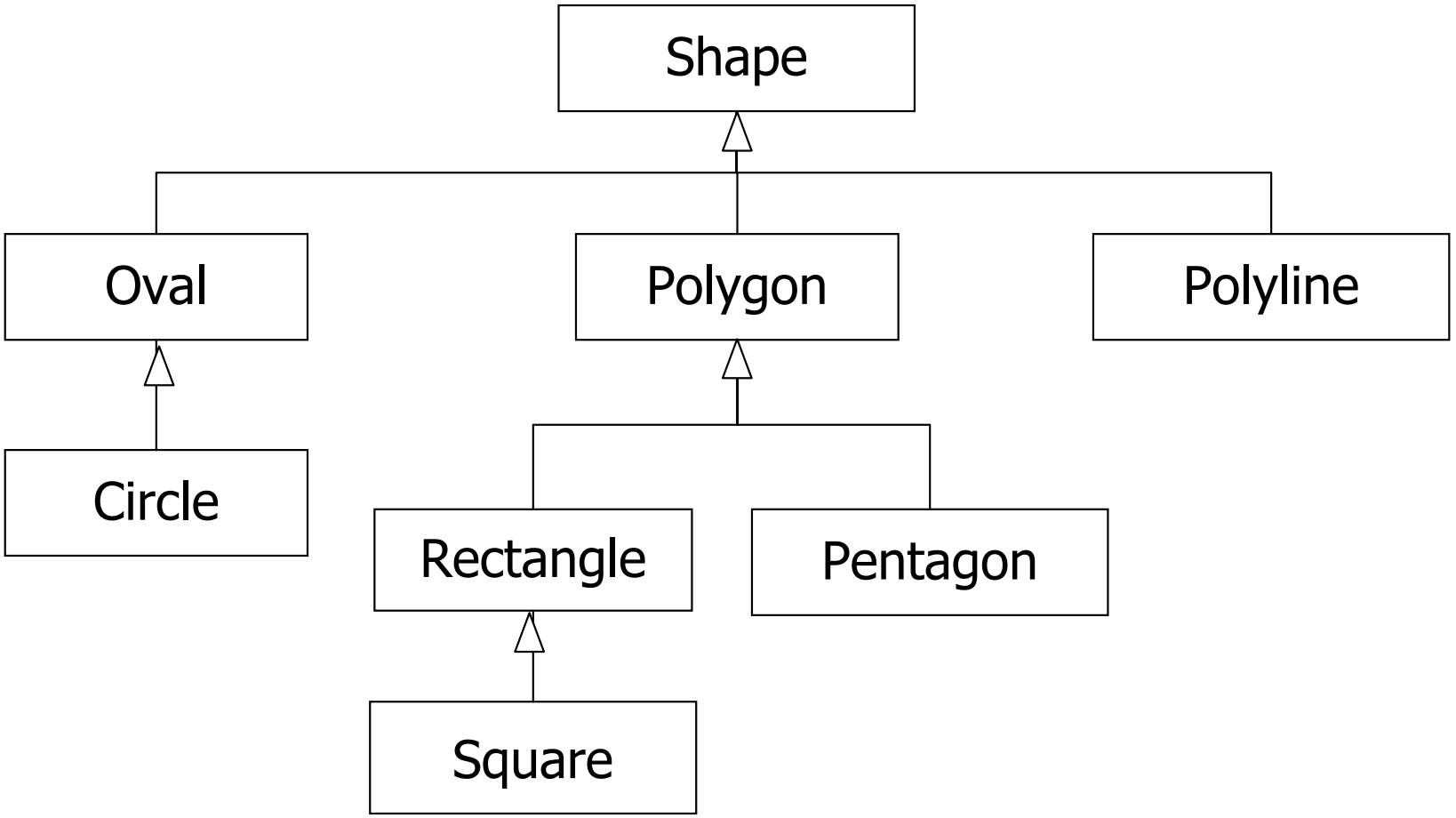



```
class Car {
  int speed;
  Engine engine;
  Car() {...}
  void forward(int speed) {...}
  void reverse(int speed) {...}
  void turnLeft(int angle) {...}
  void turnRight(int angle) {...}
}
```

```
class Car007 extends Car {
  Missile[] missiles;
  void forward(int speed) {...}
  void setTurbo(boolean enabled) {...}
  void fire() {...}
}
```



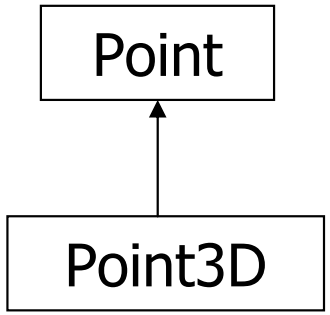
Inheritance Hierarchy



การอ้างอิง field ของ superclass

```
class Point  {
  int x, y;
  Point(int x, int y) {
    this.x = x; this.y = y;
  }
  int getX() {
    return this.x;
  }
  int getY() {
    return this.y;
  }
  ...
}
```

super. บอกว่าเป็นของ superclass




```
class Point3D extends Point {
  int z;
  Point3D(int x, int y, int z) {
    super.x = x;
    super.y = y;
    this.z = z;
  }
  int getZ() {
    return this.z;
  }
  ...
}
```

ใช้ this. แทน super. ได้ ถ้าไม่กำกวม

- ของ “พ่อ” ก็เหมือนของ “ลูก”

```
class Point3D extends Point {
  int z;
  Point3D(int x, int y, int z) {
    super.x = x;
    super.y = y;
    this.z = z;
  }
  ...
}
```

```
class Point3D extends Point {
  int z;
  Point3D(int x, int y, int z) {
    this.x = x;
    this.y = y;
    this.z = z;
  }
  ...
}
```

ใช้ super. เรียกเมท็อดของพ่อก็ได้

```
class Point3D extends Point {
  int z;
  Point3D(int x, int y, int z) {
    super.x = x;
    super.y = y;
    this.z = z;
  }
  ...
}
```

```
class Point3D extends Point {
  int z;
  Point3D(int x, int y, int z) {
    super.setXY(x, y);
    this.z = z;
  }
  ...
```

```
class Point3D extends Point {
  int z;
  Point3D(int x, int y, int z) {
    this.setXY(x, y);
    this.z = z;
  }
  ...
}
```

ใช้ super(...) เรียก constructors ของพ่อก็ได้

```
class Point3D extends Point {
  int z;
  Point3D(int x, int y, int z) {
    super.x = x;
    super.y = y;
    this.z = z;
  }
  ...
}
```

```
class Point  {
  int x, y;
  Point(int x, int y) {
    this.x = x; this.y = y;
  }
```

```
class Point3D extends Point {
  int z;
  Point3D(int x, int y, int z) {
    super(x, y);
    this.z = z;
  }
```

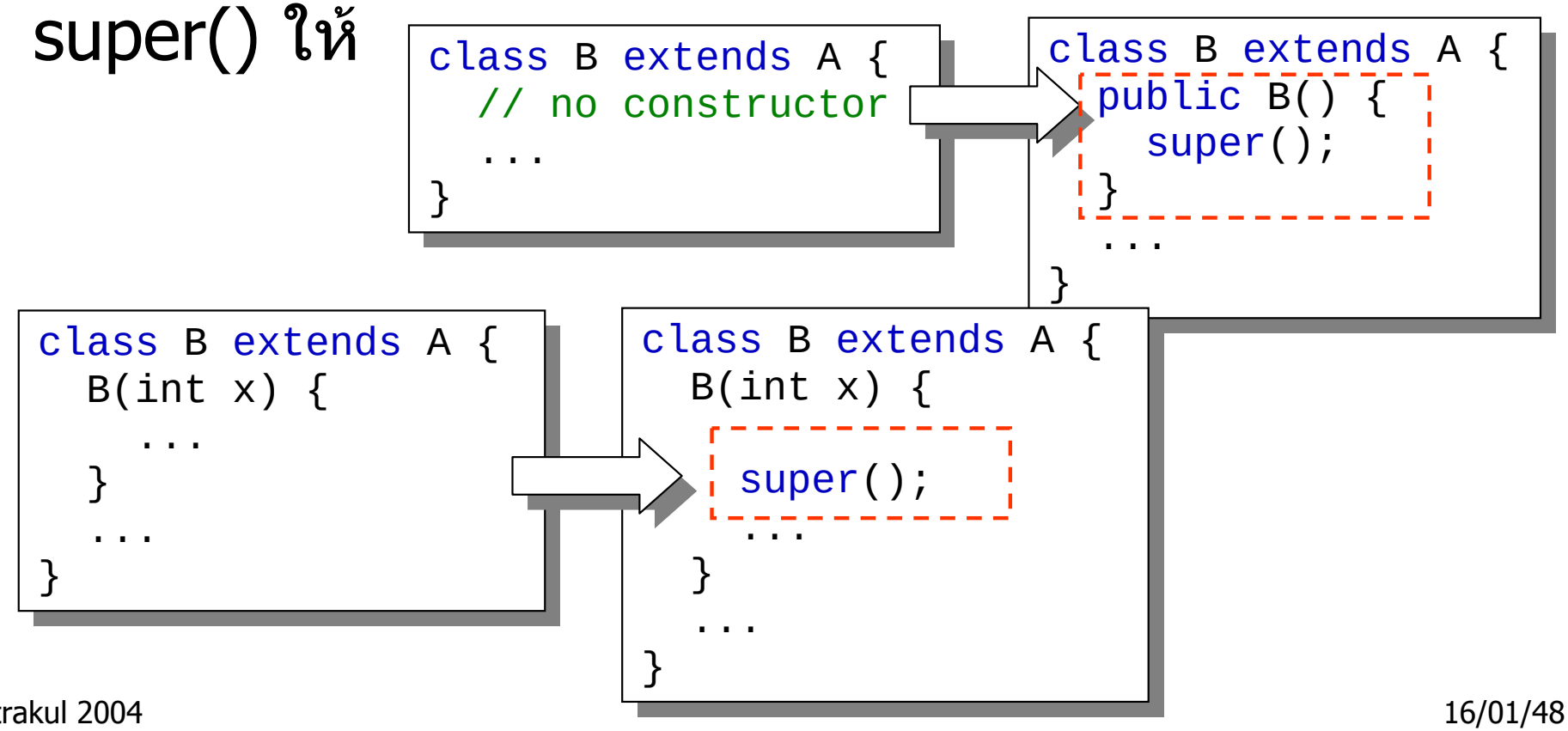
ต้องเป็นคำสั่งแรก

```
class Point3D extends Point {
  int z;
  Point3D(int x, int y, int z) {
    this(x, y);
    this.z = z;
  }
  ...
}
```

ผิด : พ่อไม่ให้ constructors ลูก !!

Constructors

- ถ้าเขียนคลาสไม่มี constructor ตัว compiler จะเพิ่ม no-arg constructor ให้ที่ไม่ได้ทำอะไร
- ถ้าบรรทัดแรกของ constructor ไม่ใช่คำสั่ง this(...) หรือ super(...) ตัว compiler จะเติมคำสั่ง super() ให้



เปลี่ยนพฤติกรรมของ “พ่อ” ได้

```
class Point  {
   ...
   void draw() {
      System.out.println("I'm a point");
   }
}
class Point3D extends Point {
   ...
   void draw() {
      System.out.println("I'm a 3d-point");
   }
}
```

เหมือนกัน

draw( ) ของ Point3D overrides draw( ) ของ Point

```
Point p = new Point();
Point3D p3 = new Point3D();
p.draw();
p3.draw();
```

เรียก draw( ) ของ Point

เรียก draw( ) ของ Point3D

แบบทดสอบ : ผิดที่ไหน ?

```
class A {
  int a, b;
  void m1() {}
  void m2() {}
}
```

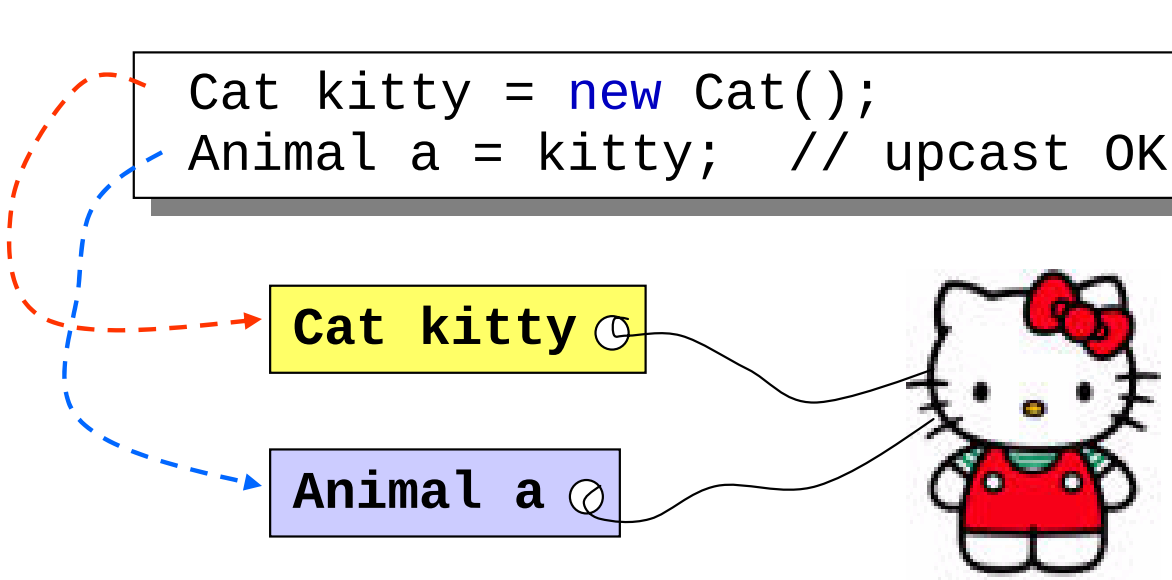
```
class B extends A {
  int a, c;
  void m2() {
    c = a;
    c = this.a;
    c = super.a;
    c = this.c;
    c = super.c;
    m2();
    this.m2();
    super.m2();
    m1();
    this.m1();
    super.m1();
  }
}
```

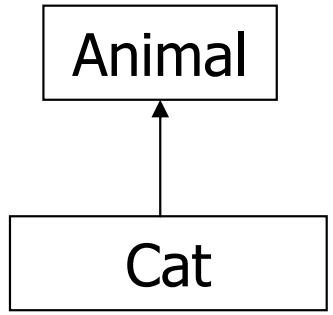
```
class C extends A {
  int a, c;
  static void m3() {
    c = a;
    c = this.a;
    c = super.a;
    c = this.c;
    c = super.c;
    m3();
    this.m3();
    super.m3();
    m1();
    this.m1();
    super.m1();
    new C().m1();
    new C().m3();
  }
}
```

Upcast

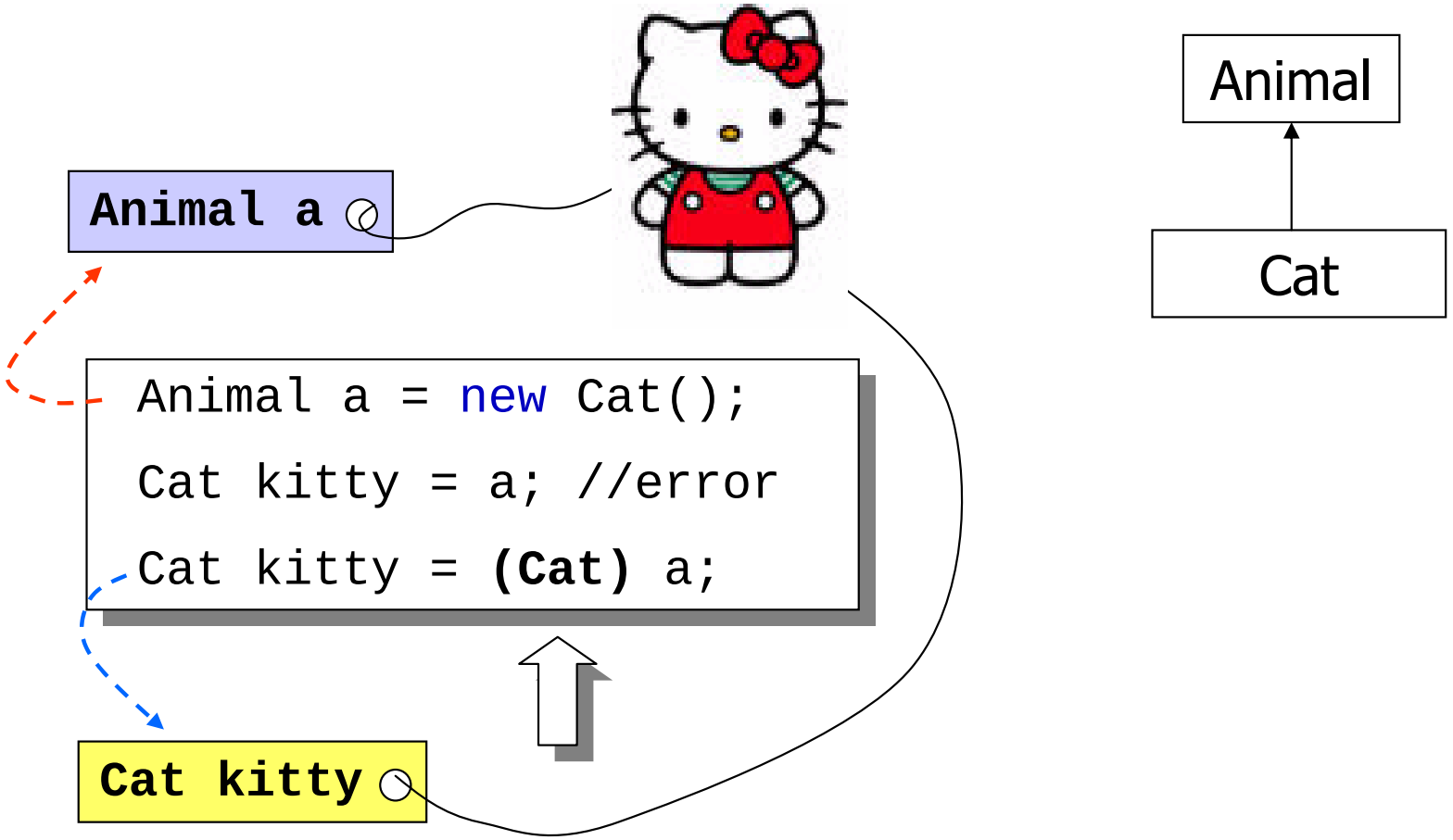
- ทุก ๆ ออปเจกต์ต้องมีตัวแปรอ้างอิง
- ถ้า Cat extends Animal
  - ออปเจกต์ของ Cat ก็ "เป็น" Animal เพราะออปเจกต์ของ Cat ต้องมีทุก field และทุก method ที่ Animal มี
  - สามารถอ้างอิงออปเจกต์ของ Cat ผ่านตัวแปรแบบ Animal ได้

```
Cat kitty = new Cat();
Animal a = kitty;  // upcast OK
```

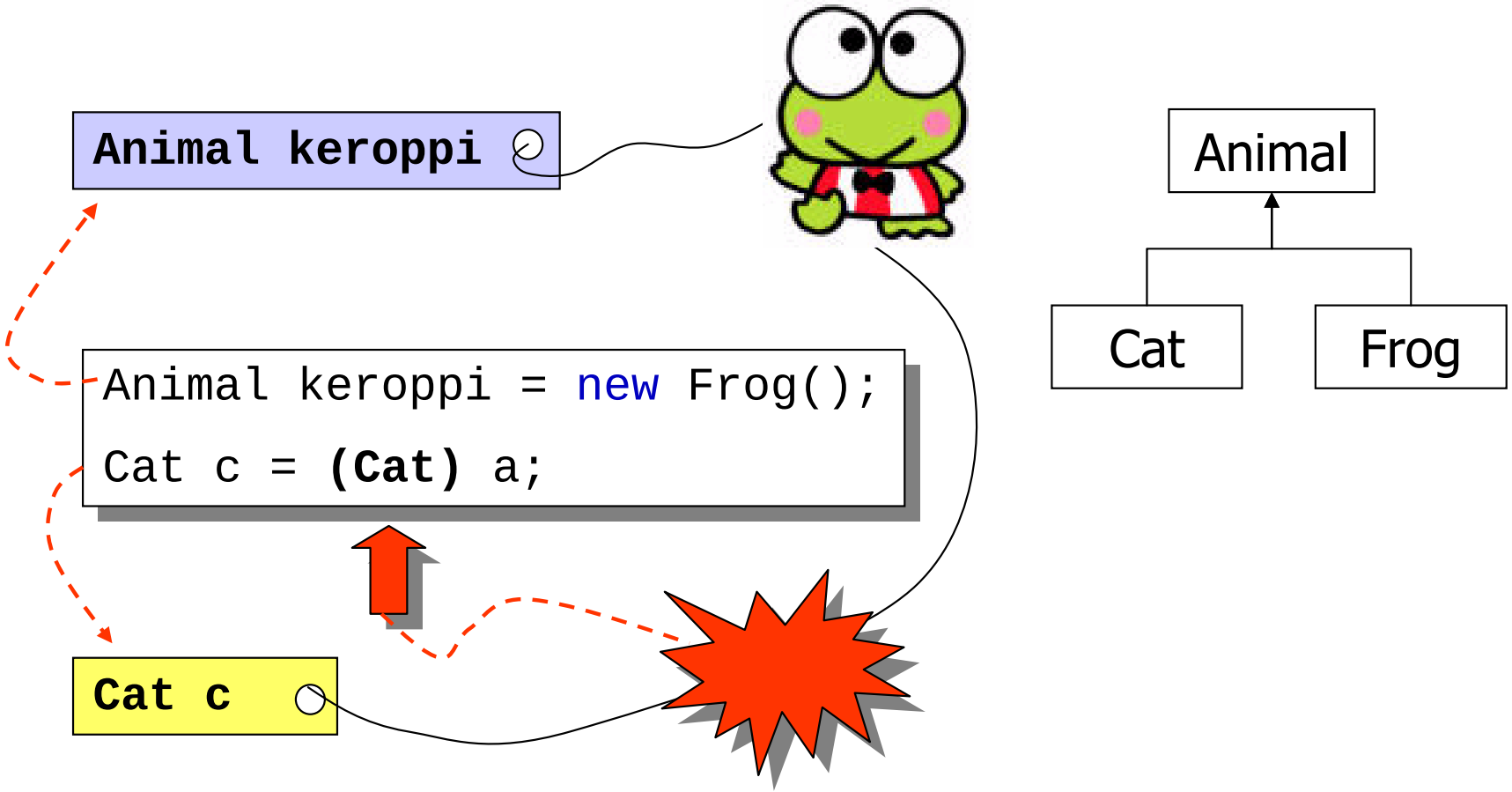




Downcast

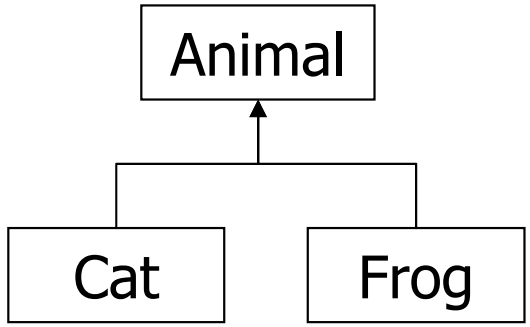

Animal a
Animal a = new Cat();
Cat kitty = a; //error
Cat kitty = (Cat) a;
Cat kitty
Animal
Cat

Run-time Error : ClassCastException



เกิด **ClassCastException** ตอน run-time

Inconvertible : Illegal Casting

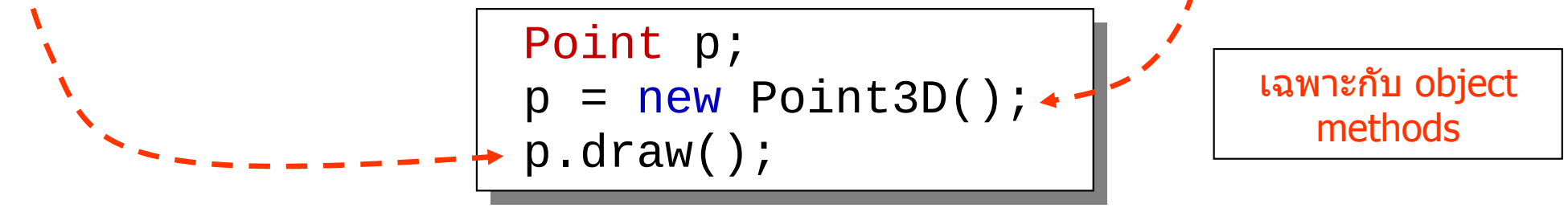



```
Cat c = new Cat();
Frog f = (Cat) c;
```

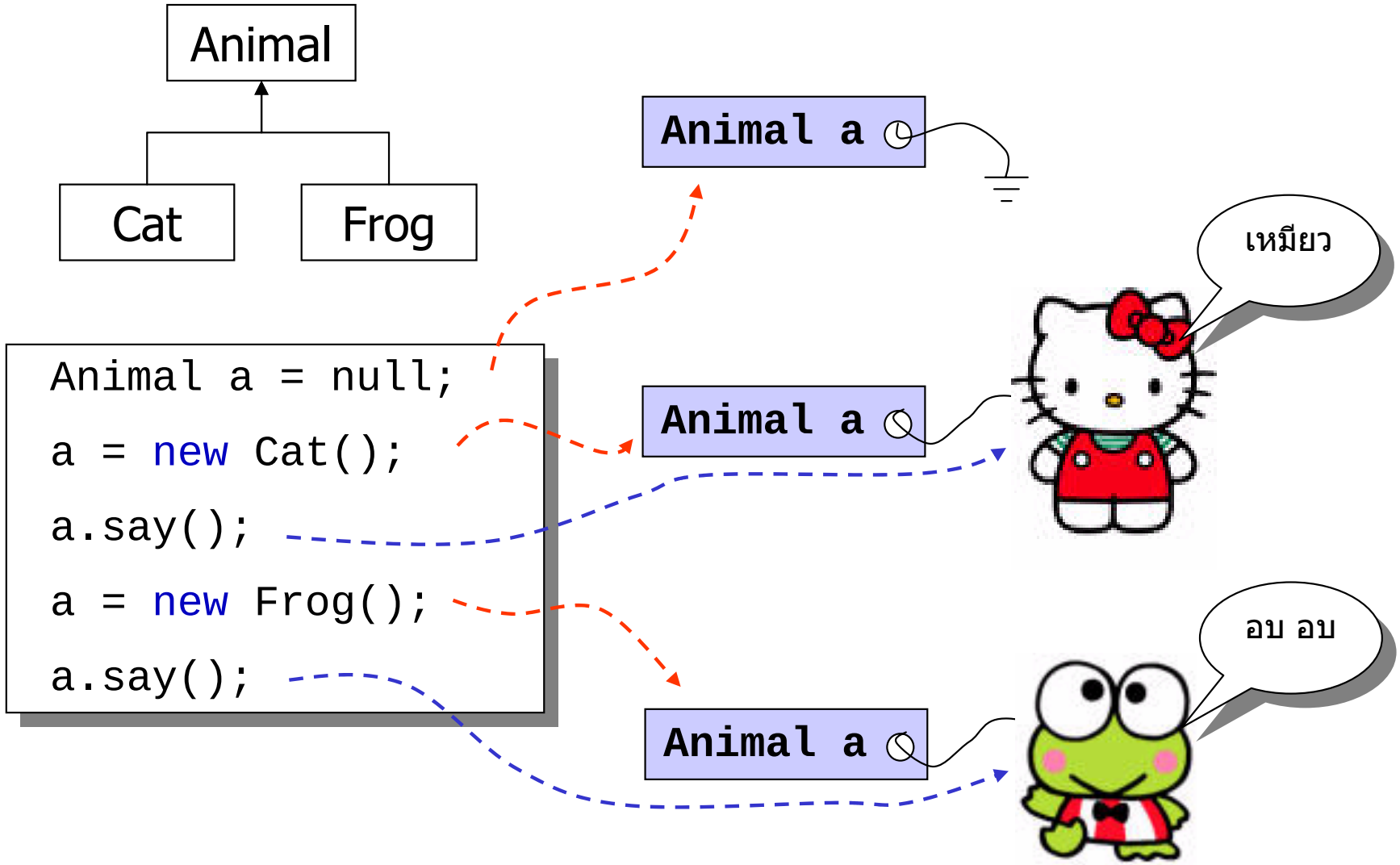
จาก inheritance hierarchy เราไม่สามารถ cast Cat เป็น Frog, compilation error

การเรียกใช้ object method

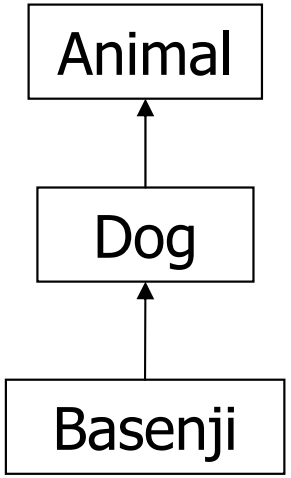
- เมื่อมีการเรียกใช้ method เช่น p.draw()
  - ตอน compile : ดูแค่ว่าคลาสที่ p ประกาศไว้ มี draw( ) ?
  - ตอน run-time : เรียก draw ของออปเจกต์ที่ p อ้างอิง
- ตัวอย่าง
  - p เป็น Point ซึ่งมี draw( ) : compile ผ่าน
  - รับรองว่า subclass ของ Point ก็ต้องมี draw( )
  - ตอน run-time, p อ้างอิงออปเจกต์ของ Point3D ซึ่ง override draw( ) ของ Point
  - p.draw( ) จึงเรียก draw( ) ของ Point3D ตอนทำงานจริง



Virtual Method Invocation



Virtual Method Invocation



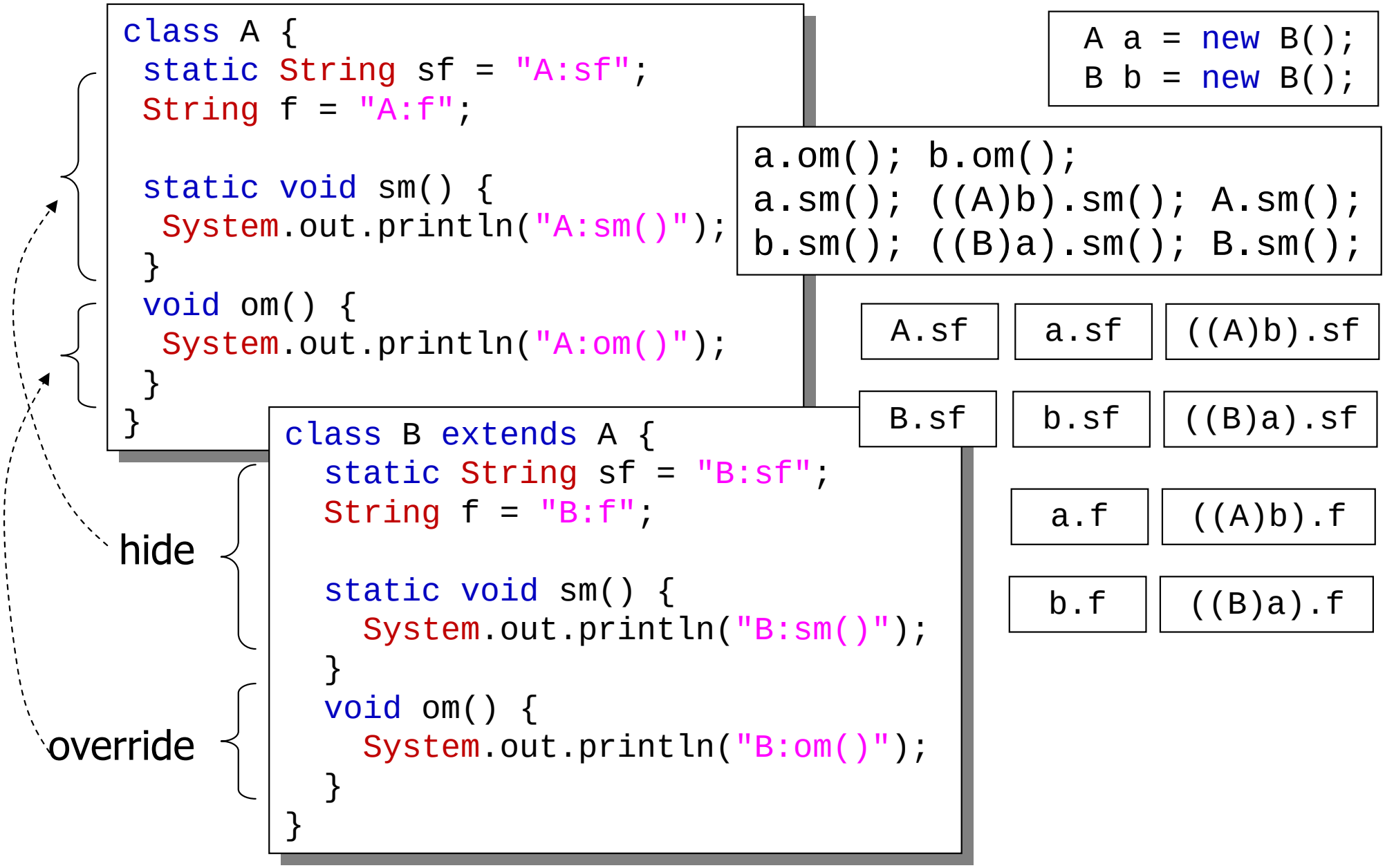

```
class Animal {
  void eat(Food f) {...}
  void say() {...}
  ...
}
```

```
class Dog extends Animal {
  void say() { bark(); }
  void bark() { ... }
  void bite(Animal a) {...}
}
```

```
class Basenji extends Dog {
  void bark() {} // barkless dog
}
```


```
Dog d = new Basenji();
d.bark();
Animal a = d;
a.bark(); // WRONG
a.say();
```

Overriding and Hiding



Overriding ไม่เหมือน Overloading

- Overriding เป็นการเขียนเมท็อดใน subclass ซึ่งมีชื่อรายการพารามิเตอร์ และ return type เหมือนใน superclass (เป็นการ "ลบล้าง" ของเก่าที่มีอยู่)
- Overloading เป็นการเขียนเมท็อดใหม่ ซึ่งมีชื่อซ้ำ แต่รายการพารามิเตอร์ไม่เหมือนกับของเดิม (เป็นการ "เพิ่มภาระ" ให้กับชื่อเมท็อดว่าทำงานหลายแบบขึ้นกับรายการพารามิเตอร์ที่ได้รับ)

```
class A {
  void f(int x, double y) {}
  void f(int x, long y) {}
  void f(int x, String s) {}
}

class B extends A {
  void f(int x, String s) {}
  void f(int x, int y) {}
}
```

```
A a = new B();
B b = (B) a;
a.f(2, 3);
b.f(2, 3);
a.f(2, "3");
```

การสร้างคลาสใหม่ด้วย Inheritance

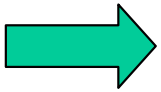
- สร้างคลาสใหม่ให้ extends จากคลาสเก่าที่มี
  - เขียน constructors ที่ต้องการให้บริการ
  - เพิ่ม fields ที่จำเป็น
  - เพิ่มเมท็อดใหม่ๆ ที่ superclass ไม่มี
  - override เมท็อดของ superclass ที่อยากเปลี่ยน
- จาวาไม่อนุญาตให้ลบ methods ของ superclass ที่ subclass ไม่อยากได้
- หลีกเลี่ยง hiding : สับสน

```
/**
  * clear is not supported in this class
  */
 void clear() {
   throw new UnsupportedOperationException();
 }
```

คลาส Object

- การเขียนคลาสใหม่ที่ไม่ได้ extends จากคลาสใด ถือได้ว่า extends จากคลาส Object
- Object นี้เป็นชื่อคลาส ขึ้นต้นด้วยโอใหญ่
- คลาส Object เป็น "บรรพบุรุษ" ของทุกคลาสในจาวา
- อะไรที่ Object มี จะตกทอดให้ทุก ๆ คลาสในจาวา

```
class Mammal {
   ...
}
```



```
class Mammal extends Object {
   ...
}
```

เมท็อดของคลาส Object

- **`boolean equals(Object obj)`**
  - indicates whether some other object is "equal to" this one.
- **`String toString()`**
  - returns a string representation of the object.
- **`int hashCode()`**
  - returns a hash code value for the object.
- **`Object clone()`**
  - creates and returns a copy of this object.
- **`Class getClass()`**
  - returns the runtime class of an object.
- **`void finalize()`**
- **`void notify()`** **`void notifyAll()`**
- **`void wait()`** **`void wait(long timeout)`**
- **`void wait(long timeout, int nanos)`**

ควร override เมท็อด toString( )

```
class Point {
  int x, y;
  public String toString() {
    return "Point[x=" + this.x + ",y=" + this.y + "]";
  }
  ...
}
```

```
...
Point p = new Point(3, 4);
String s = "" + p;
System.out.println(s);
System.out.println(p);
```

toString() ถูกเรียกโดยอัตโนมัติ เมื่อนำออปเจกต์ไป + กับสตริง

toString() ถูกเรียกโดยอัตโนมัติ เมื่อส่งออปเจกต์ไป print

```
Point p = new Point(3, 4);
String s = (String) p; // error: inconvertible
```

ควร override เมท็อด equals( )

```
class Object {
  ...
  public boolean equals(Object obj) {
    return (this == obj);
  }
  ...
}
```

ออปเจกต์สองออปเจกต์จะ “เท่ากัน” เมื่อเป็นออปเจกต์เดียวกัน

```
...
Point p = new Point(2, 3);
Point q = new Point(2, 3);
System.out.println(p.equals(q));
```

```
class Point {
  ...
  public boolean equals(Object obj) {
    if (!(obj instanceof Point)) return false;
    Point pt = (Point) obj;
    return (x == pt.x) && (y == pt.y);
  }
}
```

ออปเจกต์สองออปเจกต์จะ “เท่ากัน” เมื่อมี “เนื้อใน” เหมือนกัน

final

- final class คือคลาสที่ไม่ต้องการให้ใครมา subclass
- final method คือเมท็อดที่ subclass override ไม่ได้
- final variable คือตัวแปรที่ให้ค่าได้เพียงครั้งเดียว

```
final class String { ... }
class FastString extends String {
   ...
}
```

ระบบไม่ให้
extends String

```
class A {
  final void goo() {...}
}
class B extends A {
  void goo() {...}
}
```

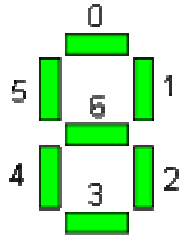
ระบบไม่ให้
override goo( )

```
class B {
  final int ONE = 1;

  void goo() {
    ONE = 2;
  }
}
```

ระบบไม่ให้ใส่ค่าใหม่

Lab 3 : SevenSegmentLED



สิ่งที่มีอยู่แล้ว : คลาส SevenSegmentLED

ออปเจกต์ของคลาส SevenSegmentLED มีไว้ใช้เป็นหน่วยแสดงผลซึ่งประกอบไปด้วยขีดสั้น ๆ 7 ขีด เราสามารถสั่งให้แต่ละขีดสว่างหรือดับได้ด้วยการเรียกเมท็อด setSegments(s) โดยที่ s คืออาเรย์ของ boolean ขนาด 7 ช่อง

s[k] มีค่า true เมื่อต้องการให้ขีด k สว่าง ถ้าเป็น false ก็คือให้ดับ เช่น

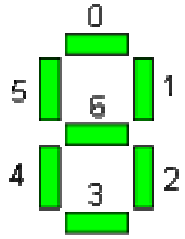
```
SevenSegmentLED led = new SevenSegmentLED();
boolean[] s = {true,false,false,true,false,false,true};
led.setSegment( s );
```

ก็จะทำให้ขีดหมาย 0 3 และ 6 (ซึ่งคือขีดแนวนอนสามขีด) สว่าง

คลาส SevenSegmentLED มี constructor ให้หนึ่งแบบ กับอีกสองเมท็อด ที่จะต้องสนใจใช้ก็คือ setSegments

```
public void setSegments(boolean[] s) {
}
```

Lab 3 : DecimalLED



สิ่งที่ต้องการ : คลาส DecimalLED

DecimalLED มี object methods ให้บริการดังนี้

`public void setDigit( int d )`

เมท็อดนี้สั่งให้แสดงตัวเลขซึ่งมีค่า d โดยที่ d มีค่าได้ตั้งแต่ 0 ถึง 9
ถ้า d เป็นค่าอื่นให้แสดง "มืดๆ"

`public int getDigit( )`

เมท็อดนี้คืนจำนวนเต็มซึ่งมีค่าเหมือนกับที่แสดงอยู่
(ถ้าแสดง "มืดๆ" ให้คืนค่า -1)

ต้องการทดสอบให้ Run คลาส Main