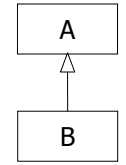
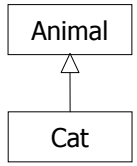
จาวา – Inheritance

สมชาย ประสิทธิ์จูตระกูล

Inheritance

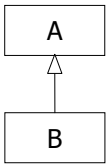
- เขียนคลาสใหม่ซึ่ง "ขยาย" ลักษณะจากคลาสเก่า (class B extends A)
- เรียกคลาส B ว่า inherits จากคลาส A
  - เรียก B ว่าเป็น subclass ของ A
  - เรียก A ว่าเป็น superclass ของ B
- ตีความได้ว่า B คือคลาสที่เป็นกรณีพิเศษของ A
  - ออบเจกต์ของ B "เป็น" A
  - แต่ออบเจกต์ของ A ไม่จำเป็นต้อง "เป็น" B
  - ตัวอย่างเช่น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





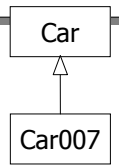
การรับมรดก

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

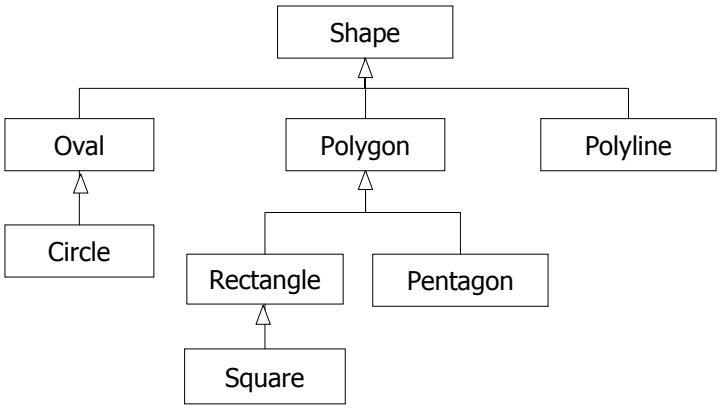



```
class Car {
  int speed;
  Engine engine;
  Car() {...}
  void forward(int speed) {...}
  void reverse(int speed) {...}
  void turnLeft(int angle) {...}
  void turnRight(int angle) {...}
}
```

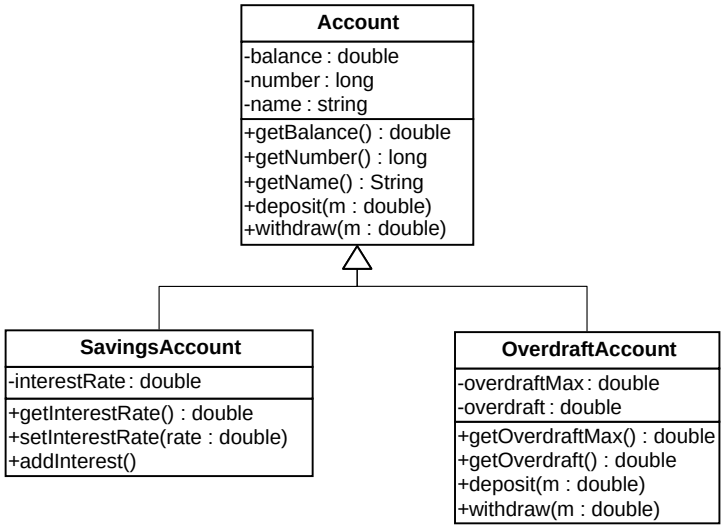
```
class Car007 extends Car {
  Missile[] missiles;
  void forward(int speed) {...}
  void setTurbo(boolean enabled) {...}
  void fire() {...}
}
```



Inheritance Hierarchy



Accounts




```
class Account {
  double balance;
  long number;
  String name;
  //-----------------------------------------
  Account(long number, String name) {
    this.number = number;
    this.name = name;
  }
  //-----------------------------------------
  double getBalance() { return balance; }
  long getNumber() { return number; }
  String getName() { return name; }
  void deposit(double amount) { balance += amount; }
  void withdraw(double amount) {
    if (amount <= balance) {
      balance -= amount;
    }
  }
}
```


ใช้ super. เพื่อใช้ member ของ superclass

```
class Account {
  double balance;
  ...
  double getBalance() { return balance; }
  void deposit(double amount) { balance += amount; }
  ...
}
```

```
class SavingsAccount extends Account {
  double interestRate;
  ...
  void addInterest() {
    super.balance +=
       super.balance * interestRate;
  }
```

```
void addInterest() {
  super.deposit(super.getBalance()*interestRate);
}
```


ละ super. ได้ ถ้าไม่มีใน class ตัวเอง

```
class Account {
  double balance;
  ...
  double getBalance() { return balance; }
  void deposit(double amount) { balance += amount; }
  ...
}
```

```
class SavingsAccount extends Account {
  double interestRate;
  ...
  void addInterest() {
    balance += balance * interestRate;
  }
  ...
}
```

```
void addInterest() {
  deposit(getBalance()*interestRate);
}
```

ใช้ super(...) เรียก constructors ของพ่อ

```
class Account {
  double balance;
  long number;
  String name;
  Account(long number, String name) {
    this.number = number;
    this.name = name;
  }
```

```
class SavingsAccount extends Account {
  double interestRate;

  SavingsAccount(long number, String name, double ir) {
    super(number, name);
    interestRate = ir;
  }
  ...
}
```

ต้องเป็นคำสั่งแรก

super( )

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

```
class B extends A {
  // no constructor
  ...
}
```

```
class B extends A {
  public B() {
    super();
  }
  ...
}
```

```
class B extends A {
  B(int x) {
    ...
  }
  ...
}
```

```
class B extends A {
  B(int x) {
    super();
    ...
  }
  ...
}
```


เปลี่ยนพฤติกรรมของ "พ่อ" ได้

```
class Account {
  double balance;
  long number;
  String name;
  String toString() {
    return "Account:" + number + ":" + balance;
  }
```

เหมือนกัน

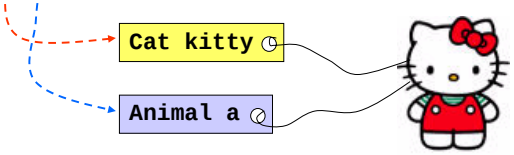
```
class SavingsAccount extends Account {
  double interestRate;
  String toString() {
    return "SavingsAccount:" + number +
           "(" + (interestRate * 100) +
           "%):" + balance;
  }
```

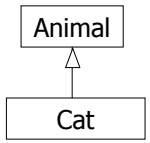
```
Account a1 = new Account(123, "Kid");
SavingsAccount a2 = new Account(234, "Nat", 0.01);
System.out.println(a1.toString());
System.out.println(a2.toString());
```


Upcast

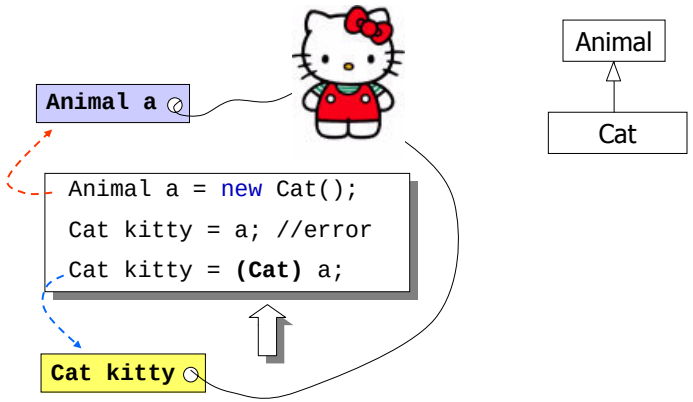
- ทุก ๆ ออปเจกต์ต้องมีตัวแปรอ้างอิง
- ถ้า Cat extends Animal
  - ออปเจกต์ของ Cat ก็ "เป็น" Animal เพราะออปเจกต์ของ Cat ต้องมีทุก field และทุก method ที่ Animal มี
  - สามารถอ้างอิงออปเจกต์ของ Cat ผ่านตัวแปรแบบ Animal ได้

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

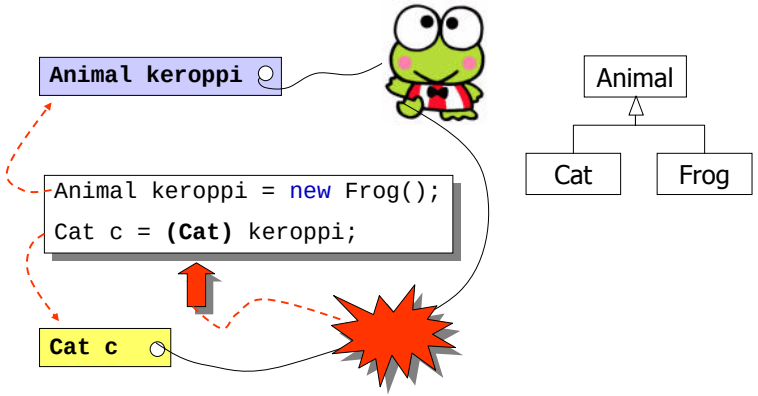




Downcast

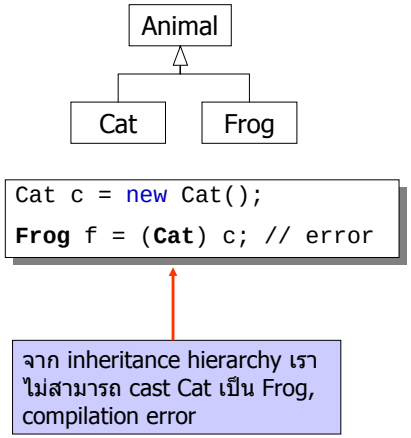


Run-time Error : ClassCastException

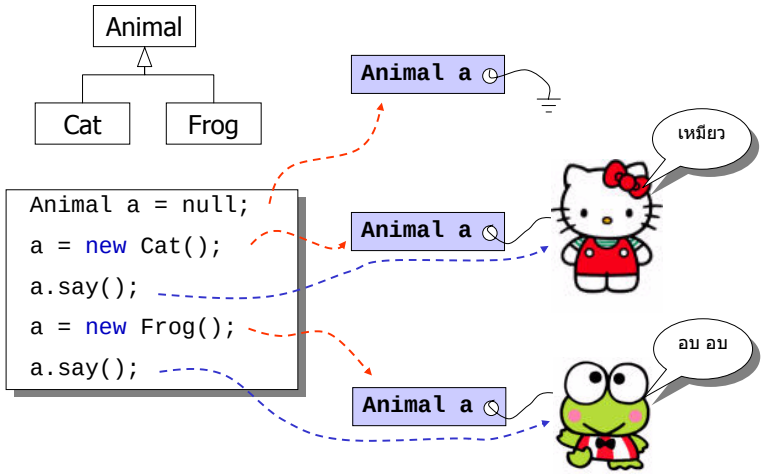


เกิด **ClassCastException** ตอน run-time

Inconvertible : Illegal Casting



การเรียกใช้ object method



การเรียกใช้ object method


```
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();
```


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

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 Account {
  double balance;
  long number;
  String name;
  public String toString() {
    return "Account:" + number + ":" + balance;
  }
  ...
```

```
Account a = new Account(234, "Na
String s = "" + a;
System.out.println(s);
System.out.println(a);
```

toString() ถูกเรียกโดยอัตโนมัติ เมื่อนำออปเจกต์ไป + กับสตริง

toString() ถูกเรียกโดยอัตโนมัติ เมื่อส่งออปเจกต์ไป print

```
Account a = new Account(234, "Nat");
String s = (String) a; // error: inconvertible
```


ควร override เมท็อด equals( )

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

ออปเจกต์สองตัว "เท่ากัน" เมื่อเป็นออปเจกต์เดียวกัน

```
Account a1 = new Account(234, "Nat");
Account a2 = new Account(234, "Nat");
System.out.println(a1.equals(a2));
```

```
class Account {
  ...
  public boolean equals(Object obj) {
    if (!(obj instanceof Account) return false;
    Account arg = (Account) obj;
    return(this.balance == arg.balance);
  }
  ...
```

สองบัญชีเท่ากันเมื่อมียอดเงินเท่ากัน

equals( ) ของ SavingsAccount

```
class SavingsAccount {
  double interestRate;
  ...
  public boolean equals(Object obj) {

    if (!super.equals(obj)) return false;

    SavingsAccount arg = (SavingsAccount) obj;
    return (this.interestRate == arg.interestRate);
  }
  ...
}
```