# Recitation 1

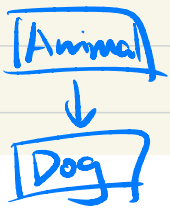
- [x] Introduction — Charlie. CDS.
  - [x] office hour: Tues 1-3 PM 60FA Room 402
  - [x] contact: charlie.chen@nyu.edu
  - [x] recitation logistics: 60th 5th Avenue
- [ ] A quick recap of OOP
  - [ ] Inheritance
  - [ ] Interface
  - [ ] Polymorphism
- [ ] Non-coding exercise
- [ ] Eclipse IDE
- [ ] Coding exercise

# Recitation Logistics

1. Attendance and participation are required. No recordings
2. Practice problems.
3. Quizzes (last 20 minutes of recit)

# A quick recap of OOP

## Inheritance




```java
public class Animal {
    private String my_name;
    public Animal(String name) {
        my_name = name;
    }
    public void introduce() {
        System.out.println("My name is " + my_name);
    }
}
public class Dog extends Animal {
    private int height;
    public Dog(String name, int height) {
        super(name); // Use *super* to call the method in the parent class
        this.height = height; // Use *this* to distinguish the name from th
    }
    public void introduce() {
        super.introduce();
        System.out.println("My height is " + height);
    }
    public static void main(String[] args) {
        Dog dog_max = new Dog("Max", 10);
        dog_max.introduce();
    }
}
```

overriding

overloading
introduce(String msg)

# Interface

```java
public interface Communictable {
    public void talk(String input);
}
public class Dog implements Communictable {
    public Dog() {}
    public void talk(String input) {
        System.out.println("Woof woof");
    }
    public static void main(String[] args) {
        Dog my_dog = new Dog();
        my_dog.talk("Hello");
    }
}
// Anything that is communictable can implement the interface
public class RadioStation implements Communictable {
    public RadioStation() {}
    public void talk(String input) {
        System.out.println("Received msg: " + input);
    }
    public static void main(String[] args) {
        RadioStation my_station = new RadioStation();
        my_station.talk("Hello");
    }
}
```

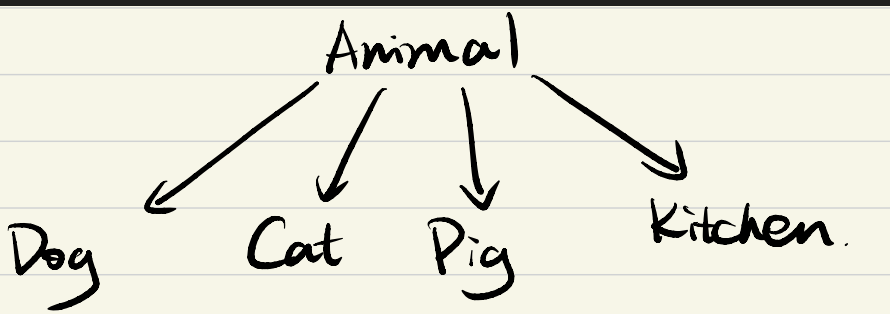
# Polymorphism

```
Animal ani = new Dog();

ani.introduced();
```

↑ has access to the attribute height. which is not in Animal class.

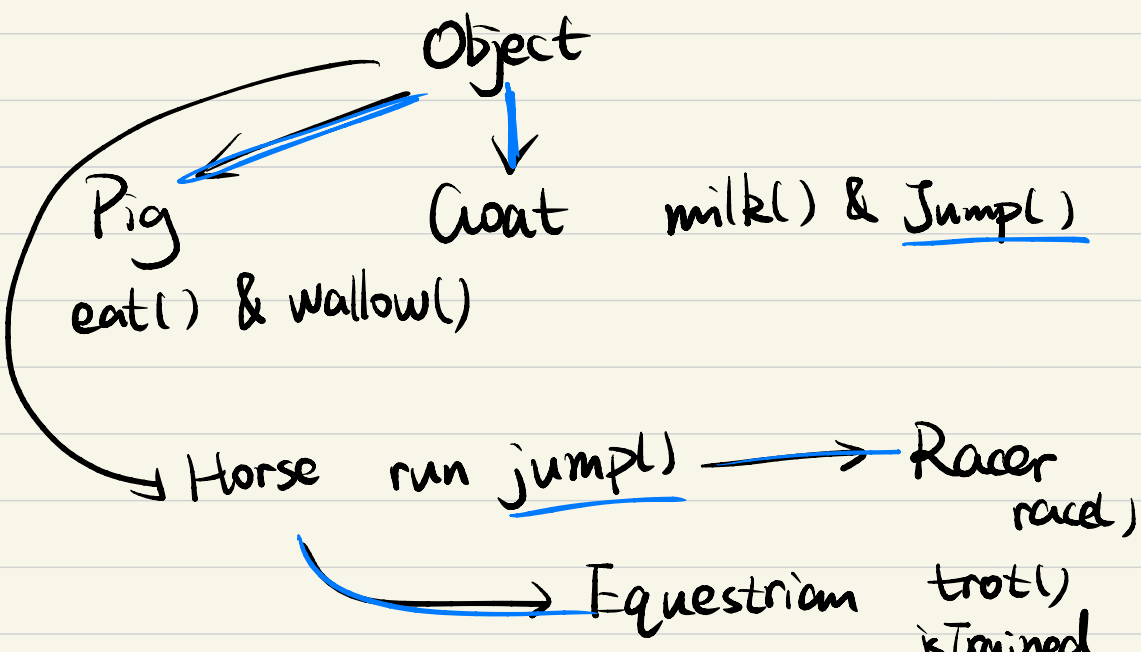
1. What are some potential efficiency disadvantages of having very shallow inheritance trees, that is, a large set of classes, A, B, C, and so on, such that all of these classes extend a single class, Z?




1. Write repetitive codes for A, B, C.
2. The parent class needs to be generic.
3. Not much abstraction.

2. Draw a class inheritance diagram for the following set of classes:

- Class Goat extends Object and adds an instance variable tail and methods milk() and jump().
- Class Pig extends Object and adds an instance variable nose and methods eat(food) and wallow().
- Class Horse extends Object and adds instance variables height and color, and methods run() and jump().
- Class Racer extends Horse and adds a method race().
- Class Equestrian extends Horse and adds instance variable weight and isTrained, and methods trot() and isTrained().



3. Consider the inheritance of classes from the previous exercise, and let d be an object variable of type Horse. If d refers to an actual object of type Equestrian, can it be cast to the class Racer? Why or why not?

```
Horse d;
d = new Equestrian()
((Racer) d).race()
```

compile error