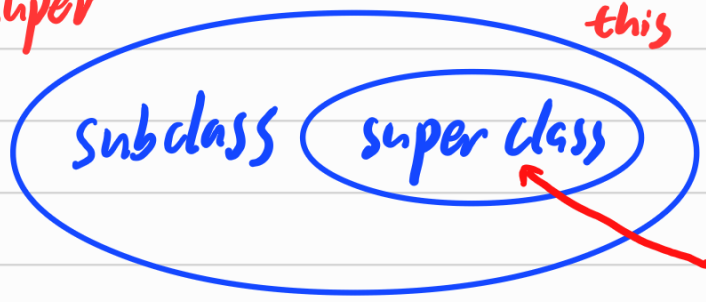
Super class —extends→ Child class (Code re-use)
—extends→ Child class
—extends→ Child class

super
this

→ a subclass automatically has all non-private instance variables & methods
→ a subclass can have additional instance variables & methods.



override
(non-static)

<modifier> class <class name> extends <super class name> {

① instance variables

② Constructor(s)

③ methods

}

Constructors: super(params); ← refers to the super class constructor

this (params); ← refers to the current class constructor

<modifiers>: public / null / protected / private

abstract { method: incomplete method
class: contains incomplete methods
interface: default

static { variable: only one
method: an independent object

final { variable: cannot be changed

method: cannot be overriden
class: cannot have subclass

'Object' class: every Java class is a decendent of the class 'object'.
→ automatically extends

→ methods: toString() ← 返回一段自定义的 string

equals() ← { primitive data type: 比较值
object: 比较 object 内容 (instance vars...)

Subtype polymorphism: a parameter of type 'object' can be replaced by an object of any class
因为所有 class 都 extends 'object',
所以任何一个 object 都能使用/生成
定义为使用一个 method.

'instanceof': check whether a object is an instanceof class

Dynamic binding: 从下到上使用 method
(searching bottom-up)

{ Overriding: different class, same signature
Overloading: same class, different signature

→ method name + parameter list

implicit
↓
Type casting: ClassName1 c = s;

```
ClassName1 C = (ClassName1) S;
```

↑ explicit

☆ upcast always work

downcast must be explicit

↳ valid only if superclass has all methods in the child class.

↓

Abstract methods / class

```
<modifier> abstract class <class name>{
  <modifier> abstract <return type> <method name>(param);
}
```

↓ valid downcasting

```
<modifier> class <class name> extends <super class>{
  @Override ← Jvm check error
  <modifier> <return type> <method name>(param){
  }
}
```

Interfaces : a specification of what some classes can do

(API, 提供定义方法)

```
public interface <interface name>{
```

<modifier> <return type> <method name> (params);

:

}

use: 'implements' <interface1>, <interface2>, ...

eg: public class Dog extends Pet implements Swimmable{

}

interface vs abstract class:

Same: ① no code

② cannot be used to create objects

③ provide subtyping polymorphism

	Abstract class	Interface
Is it a Class?	Yes	No
Create Ojbect	No	No
Relationship with Class	Inheritance	A class can implement multiple interfaces
Members	May have private memebers	Default to pulic static final，any class implement the interface has the access to these constant.
Methods	May be private，non-abstract which has implementation	Default to public，abstract
Design	"is-a" relationship	"like-a"relationship
Implementation	extends	implements
Derived/Concrete Methods	Implement abstract methods or still be abstract	Must implement all methods in the interface