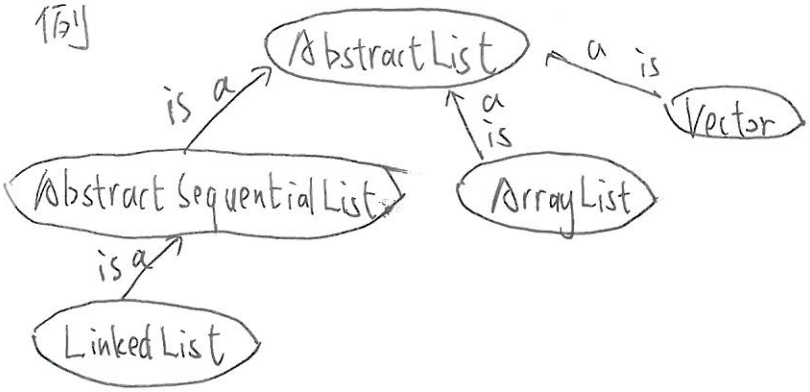
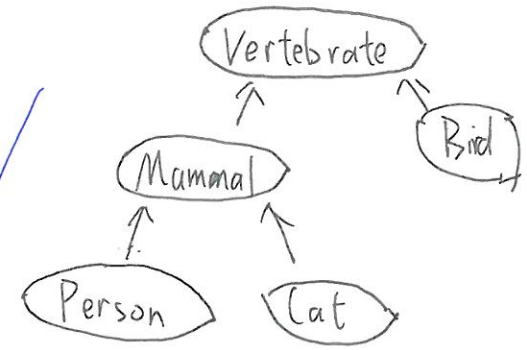
Lecture 16

Inheritance (继承): class的属性传给subclass

- subclass "is a" superclass object
  - 一层层着次变来吐兽加

例





例: Person "is a" Mammal

Person has extra attributes

↑ more general; ↓ more specific

specific "is a" general

"is a" 是一种单向关系

Superclass (父类) 只有大略的功能

扩展父类获得新的子类 (该子类也与父类兼容)

⇓

继承由 extend a class 来实现

步骤: public class NewClass extends OldClass

(此时的NewClass隐含着OldClass所有的数据与操作)

protected declaration 介于 public 与 private 之间

protected data & method 在父类本及其子类中可直接访问 (其它地方不能直接)

另: private declarations 是子类的部分, 但子类不能直接访问

```
public class Foo
{
   private int X;
   protected int Y;
   public int Z;

   // rest of class
   // not shown
```

```
public class SubFoo
             extends Foo
{
   public SubFoo()
   {
      X =    // ILLEGAL
      Y =    // LEGAL
      Z =    // LEGAL

   // rest of class
   // not shown
```

```
// main program
Foo F = new Foo();
SubFoo S = new SubFoo();

F.X = // ILLEGAL
F.Y = // ILLEGAL
F.Z = // LEGAL

S.X = // ILLEGAL
S.Y = // ILLEGAL
S.Z = // LEGAL
```

⇓ private 在一个类内，就只能在那个类中访问（子类也不行）

protected 在一个类内，可以在该类及其子类中访问

public 在一个类内，啥地方都能访问

仔细考虑使用以上哪个 declaration

subclass "is a" superclass

例 ✓ Foo F = new Foo();

✓ SubFoo S = new SubFoo();

✓ Foo F = new subFoo(); "is a"

✗ SubFoo S = new Foo();
子类 子对象名 父类名

We can access a subclass object using a superclass reference

↘父类可以直接访问子类

//怎么创建对象的方法：类名 对象名 = new 类名();

如果我们使用SubFoo对象，那只能用SubFoo引用，即 SubFoo S = new SubFoo();

?当混合对象类型

例：Foo [] data = new Foo[3]; //创建数组对象

data[0] = new Foo();

data[1] = new SubFoo();

data[2] = new ···· ();

数组中元素放类及一些子类

⇒ array reference is Foo

另例

```
public RationalNumber add (RationalNumber op2)
{
    int commonDenominator = denominator * op2.getDenominator();
    int numerator1 = numerator * op2.getDenominator();
    int numerator2 = op2.getNumerator() * denominator;
    int sum = numerator1 + numerator2;

    return new RationalNumber (sum, commonDenominator);
}
```

//获得一个通用分母

//分子归一化

//添加归一化分子

//创建并返回新有理数，包含分子和公共分母

新数据类型：Mixed Number（有理数，由整数和分数组合而成）

例：有理数 $\frac{11}{3}$，Mixed Number为 $3\frac{2}{3}$

有两种创建方法 { Composition 组合 - MixedNumber
Inheritance 继承 - MixedNumber2

有理数

RationalNumber
int numerator int denominator --- add(), subtract(), multiply(), divide(), etc.

Composition 组合

MixedNumber
int whole RationalNumber frac --- add(), subtract(), multiply(), divide(), etc.

MixedNumber类用的是 RationalNumber 对象

以其中的方法将 RationalNumber 对象当作client处理

两者间没有特殊关系

Inheritance 继承

MixedNumber2 extends RationalNumber	RationalNumber
add(), subtract(), multiply(), divide(), etc.	int numerator int denominator --- add(), subtract(), multiply(), divide(), etc.

MixedNumber 2类是 RationalNumber类的子类

"is a"

例：MixedNumber2中的numerator变量在RationalNumber中就已被定义，RN中的方法可以直接使用

只有新的特性（返回值为MN2）才是必须

以上两种情况使用 add() method

1° composition 组合

实质重写了整个方法.

```
public MixedNumber add(MixedNumber rightOp)
{
        int newWhole = whole + rightOp.whole;
        RationalNumber newFrac = frac.add(rightOp.frac);
        MixedNumber temp = new MixedNumber(newFrac);
        temp.whole += newWhole;
        return temp;
}
```

类名 对象

// 两者所有数相加
// 所有分数相加，获得新的有理数文
// 从分数中获得一个新的MixNumber
// 把整数文部份加给 MixNumber

temp.

2° inheritance 继承

工作通过继承的 add() 方法实现

don't change

```
public MixedNumber2 add(MixedNumber2 m)
{
        return new MixedNumber2(super.add(m));
}
```

把m加到父类

继承的另一个优势：两个方法的对象可以同时被使用

例

```
RationalNumber R1 = new MixedNumber2(3, 4, 5);
RationalNumber R2 = new RationalNumber(7, 8);
RationalNumber sum = R1.add(R2);
```

所有变量都是父类(R/N)下的变量

/忘了：调用对象 对象名.( ) ↑数据

重点：当父类引用用于访问子类对象时

⚠ 一旦一可以调用的方法是最初在父类中被定义的方法

(子类中最初定义的不能调用)

即：只能用通用的，

Lecture 17

Object 类是所有类的父类

- 当没有明确说一个类继承于另一个类时，该类 extends Object
- 扩展自Object及其所有子类的类本对被称为 class hierarchy
- 所有类最终返回 Object

⚠⚠ Polymorphism 多态

- 以多种形式出现的能力
- 根据对象的数对应类型或类，以不同的方式处理对象

[我们能够以一致的方式混合不能类型的方法和对象]

?方法签名

方法名称+形参类型

例中MS为 max(float,float)

ad-hoc多态 / method overloading

同一个类(层次结构)中不同的方法 有同样的名字 却不同的 method signature (name + parameter)

例：public static float max(float a, float b)

```
int M = 10, N = 5;
float X = (float) 3.5, Y = (float) 8.8;

int max1 = max(M, N);

float max2 = max(X, Y);
```

大多用两个实参调用 max( )

编译器评估实参类型选取正确版本

static method 和 instance method 可用

区别在于有无 void

instance method 要先建立实例变量才能调用

若无法精确匹配，使用 widening ⇒ smaller type → larger type

例：int to float

Subclassing Polymorphism 子类多态

1° Method overriding 方法覆盖

— 在父类中被定义的方法在具有相同方法签名的子类中被重新定义.

— 由于签名相同，并不重载方法，而是 overrid the method

（相当于子类的定义中替换更新父类的）

例

```
public boolean add(String val)
{
    if (size == data.length)
        resize(2 * size);
    data[size] = val;
    size++;
    return true;
}
```

add() 添加了一个 boolean 变量到 String

二⇒ 考虑子类 SortedList

public class SortedList extends SimpleSortedList

add的不同用法（不能将其直接放在末尾）

```
public boolean add(String val)
{
    if (size == data.length)
        resize(2 * size);
    int loc = size;
    for (int i = size-1; i >= 0; i--)
    {
        int result = val.compareTo(data[i]);
        if (result >= 0)
        {
            break;
        }
        else
        {
            data[i+1] = data[i];
            loc--;
        }
    }
    data[loc] = val;
    size++;
    return true;
}
```

如何使方法在子类消失？

throw an expectation

2° Dynamic Binding 动态绑定

— 运行时，方法调用与对应的代码相关联

— 实际执行的方法由对象的种类确定

⇒ 用于访问混合数据类型的集合

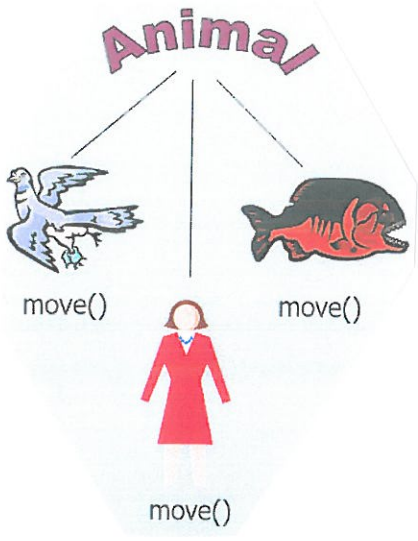
例：每个子类以自己的方式覆写方法

```
Animal [] A = new Animal[3];
A[0] = new Bird();
A[1] = new Person();
A[2] = new Fish();
for (int i = 0; i < A.length; i++)
    A[i].move();
```

↓

三个对象分别调用 move 方法.

move() 与 OBJECT 相联



父类引用可以访问子类对象，反之不行

Animal A = new Bird(); ✓　　Bird B = new Animal(); ✗.

给定Class C的Reference R，R只能访问 class C or Above 中定义的变量.

之前例：move()通过数组A可以访问，因为move()在上级类中被定义.

但在子类中可以重新定义（方法覆写）.

动态绑定，我们从Animal Reference调用move()

→我们获得由被访问的对象定义的版本

A[i].move() 实际指向3个不同方法.

/ class Fish包含新实例变量 waterType 和新方法 getWaterType()

```
Fish F = new Fish();
Animal A = new Fish();
System.out.println(F.getWaterType()); // ok
System.out.println(A.getWaterType()); // NO!
```

↓A is Animal Reference，只能看见 Animal 中的 data & method.

而该方法在Fish.

↓改 ((Fish).A).get ... ());

↓overridding → 间接访问

例：Class Animal有 method characteristics(). [Fish class overrides characteristics.

```
public void characteristics()
{
    super.characteristics(); 对象super调用.
    System.out.println("I have fins");
    System.out.println("I like " + myWater);
}
```

→不存在

A.characteristics();

↓一个fish object, animal reference.

（子类对象，父引用）

用A调用.

cast a reference to a type（遵循 is a）.

cast Fish Object to Class Fish.

SUM

Superclass references CAN BE used to reference subclass objects

Subclass references CANNOT BE used to reference superclass objects

The type of the reference determines what public data and methods are ACCESSIBLE / can be seen

The type of the object determines what data and methods EXIST

- Methods and data initially defined within a subclass CANNOT BE accessed via a superclass reference
- The type of the object also determines which VERSION of an overridden method is called

父引用 ⟶ 子对象

子引用 X 父对象

引用类型 ⇒ 能看见 public data & method

对象类型 ⇒ 能存在 data & method.

Lecture 18

abstract class

- 在声明时使用关键字 abstract : public abstract class XX{.
- 可能存在 abstract method (不实现)
- 对象不能实例化

abstract class的子类 ⟶ 子类本身被命名为抽象

⟶ 子类应当实现所有抽象方法

其优势

- 父类引用仍可以以多态访问所有子类对象
- 不必为子类单独定义一继承的
- 类结构良好.

single inheritance 单一继承

- 一个新类只能是一个父类的子类 (只能继承自一个)

一个抽象类的例子

```
public abstract class SimpleShape
{
    protected String color;
    public SimpleShape(String col)
    {
        color = new String(col);
    }
    public abstract double area();
}
```

对象

```
public class Circle extends SimpleShape
{
    private double radius;
    public Circle(String col, double r)
    {
        super(col);
        radius = r;
    }
    public double area()
    {
        return (Math.PI * radius * radius);
    }
}
```

算面积

super() 子类重写父类.

另一个子类 Square

```
public class Square extends SimpleShape
{
    private double side;
    public Square(String col, double s)
    {
        super(col);
        side = s;
    }
    public double area()
    {
        return side * side;
    }
}
```

创建数组储存这些信息

```
// Main
SimpleShape [] shapes = new SimpleShape[2];
shapes[0] = new Circle("Red", 2.0);
shapes[1] = new Square("Blue", 4.0);
for (SimpleShape s: shapes)
    System.out.println("Area: " + s.area());
```

以多种方式识别对象

— 基于其继承的固有性质 (他是一个人/一条狗)

Classes are used to 识别对象

— 或基于其能做什么 (他是一个作家/运动员)

Interface (接口) are used to 识别对象

接口是方法的集合

— 没有实例方法的主体部分

— 可以使用常量, 但不可以使用 instance variables

关于 class

- 接口说明了该类的能力
- 类能实现任意数量的接口

↓

```
public interface Laughable
{
    public void laugh();
}
```

① ②

```
public interface Booable
{
    public void boo();
}
```

Java 类通过实现方法②来实现①.

①: 类名 ②: 方法名.

在 class header, 声明方法被实现了

```
public class Comedian implements Laughable, Booable
{
   // various methods here (constructor, etc.)
   public void laugh()
   {
       System.out.println("Ha ha ha");
   }
   public void boo()
   {
       System.out.println("You stink!");
   }
}
```

一个接口变量可以用于引用实现该接口的任何对象.

—(方法名相同,不同类中代表不同代码)

只有interface reference才能访问 interface method

```
Laughable [] funny = new Laughable[3];
funny[0] = new Comedian();
funny[1] = new SitCom(); // implements Laughable
funny[2] = new Clown();  // implements Laughable
for (int i = 0; i < funny.length; i++)
    funny[i].laugh();

funny[0].boo();    // illegal even though Comedian
                   // has the boo() method
```

声明数组
调用内部对象

错, 类 Laughable, 对应方法 laugh

越权,别的方法的接口不能调用另一方法的对象.

通过接口访问对象的好处

—只关注一类的部分属性

例: Sorting —每个东西都不同,但存在sth.使其能被排序

根据接口定义的单个方法排序

```
public static void selectionSort(int [] array)
  {
     int startScan, index, minIndex, minValue;

     for (startScan = 0; startScan < (array.length-1); startScan++)
     {
        minIndex = startScan;
        minValue = array[startScan];
        for(index = startScan + 1; index < array.length; index++)
        {
           if (array[index] < minValue)
           {
              minValue = array[index];
              minIndex = index;
           }
        }
        array[minIndex] = array[startScan];
        array[startScan] = minValue;
     }
  }
```

Parameter is an array of int

minValue is an int

Primitive values are compared using relational ops

红色代码是此生-特定用于int数组排序的

Comparable 接口：是 method int compareTo(Object r);

当前对象 < r，返回负值.

当前对象 = r，返回 0

当前对象 > r，返回正值.

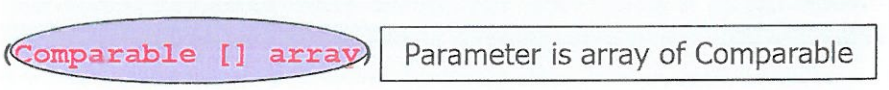
该规则通用

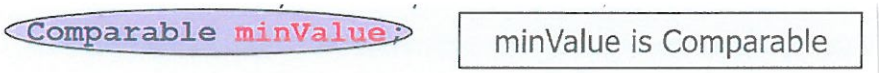
```
String S1 = new String("zebra");
String S2 = new String("aardvark");
String S3 = new String("abacus");
int check1 = S1.compareTo(S2);
int check2 = S2.compareTo(S3);
int check3 = S3.compareTo(S3);
System.out.println("check1 is " + check1);
System.out.println("check2 is " + check2);
System.out.println("check3 is " + check3);
```

(这里的值确的)

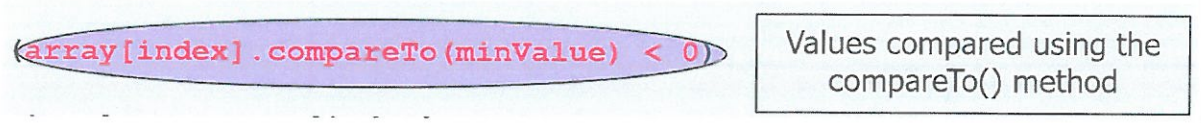
2
-1
0

ADT 同样适用，用此方法改写.





查询结果是否<0判断大小.



这是个黑匣子，不需知道很细

Single sort method 适用于 Comparable 类下的任意 array

Lecture 19

generic method (泛型方法) 排序数据 (上个 lec)

?限制数据的同质性与特定性 — parameterized type (参数化类型)

想法：An array list of (T) → T 是特定的 Java 类型

(从而使集合同质)

以 SimpleList 为例，其可用类型为 string，形参和返回值都是 String.

如果用 string 外的类型，得定义新类

用泛型接口，来实现想法

例

```
public interface SimpleListInterface<T>
{
    public boolean add(T val);

    public boolean add(int loc, T val);

    public T remove(int loc);

    public T get(int loc);

    public T set(int loc, T val);

    public int size();
}
```

见旁：T 可以是任意类型

我们可以在制作实现接口的对象

时再定义接口

使接口下的方法与具体的实现相分离.