阶乘
递归调用

```
//Fibonacci 数列的打印
//
public class Fibonacci{
    public static void main(String[] args){
        int n = 10;
        T fabe = new T();
        int fabe1 = fabe.f(n);
        System.out.println("n= "+n+"对应的斐波拉契数为："+fabe1);
    }
}

class T{
    public int f(int n ){
        if( n==1 || n==2 ){
            return 1;
        }else if(n>2){
            return f(n-1) + f(n-2);
        }else{
            System.out.println("应该输入大于 0 的数");
            return -1;
        }
    }
}

//迷宫问题
//
public class MiGong{
    public static void main(String[] args){
        //    思路
        //1. 先创建迷宫，用二维数组表示：  int[][] map = new int[8][9]
        //2. 再规定  map  数组的元素值：0  表示可以走; 1  表示障碍物

        int[][] map = new int[8][7];
        //3. 将最上面的一行和最下面的一行全部设置为 1
        for(int i = 0; i<7 ; i++){
            map[0][i] = 1;
            map[7][i] = 1;
        }
        //4. 将最右边的一列和最左边的一类全部设置为 1
        for(int i=0;i<8;i++){
            map[i][0]=1;
```

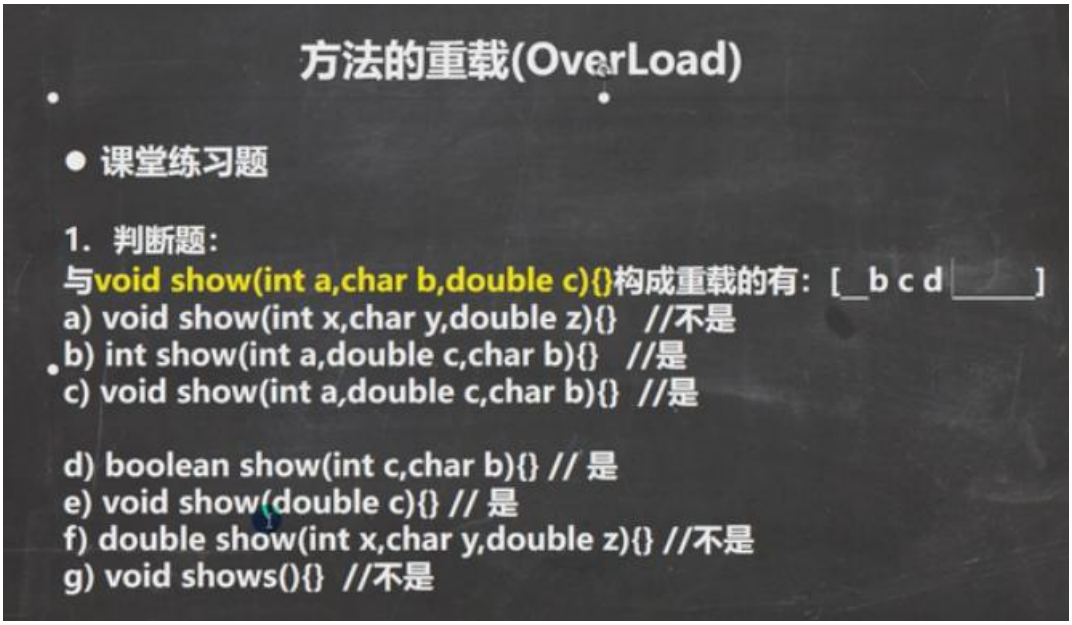
```
            map[i][6]=1;
        }
        //障碍物的设置
        map[3][1] = 1;
        map[3][2] = 1;
        map[2][2] = 1;
        //输出当前地图
        System.out.println("====当前地图情况====");
        for(int i=0; i<map.length ; i++){
            for(int j=0 ; j<map[i].length;j++){
                System.out.print(map[i][j] + " ");//输出一行
            }
            System.out.println();
        }

        //使用 findWay 给老鼠找路
        T t1 = new T();
        t1.findWay(map,1,1);

        System.out.println("====找路的情况如下====");
        for(int i=0; i<map.length ; i++){
            for(int j=0 ; j<map[i].length;j++){
                System.out.print(map[i][j] + " ");//输出一行
            }
            System.out.println();
        }
    }
}

class T{
    //使用递归回溯的思想来解决老鼠出迷宫问题
    //1. findWay 方法是用来专门找出迷宫的路径
    //2. 如果找到就返回 true，没有找到就返回 false
    //3. map 代表二维数组即迷宫
    //4. i 和 j 代表老鼠的位置，初始化的位置为(1,1)点
    //5. 由于使用递归找路，所以规定 map 数组每一个值的含义
    //    0 表示可以走;1 表示障碍物;2 表示可以走;3 表示走过但是是死路
    //6. 当 map[6][5] = 2 就说明找到通路,就可以结束,否则就继续找
    //7. 先确定老鼠的找路策略 下->右->上->左
    public boolean findWay(int[][] map, int i, int j){
        if(map[6][5] == 2){
            return true;
        }else {
            if(map[i][j] == 0){//当前这个位置为 0，表示可以走
```

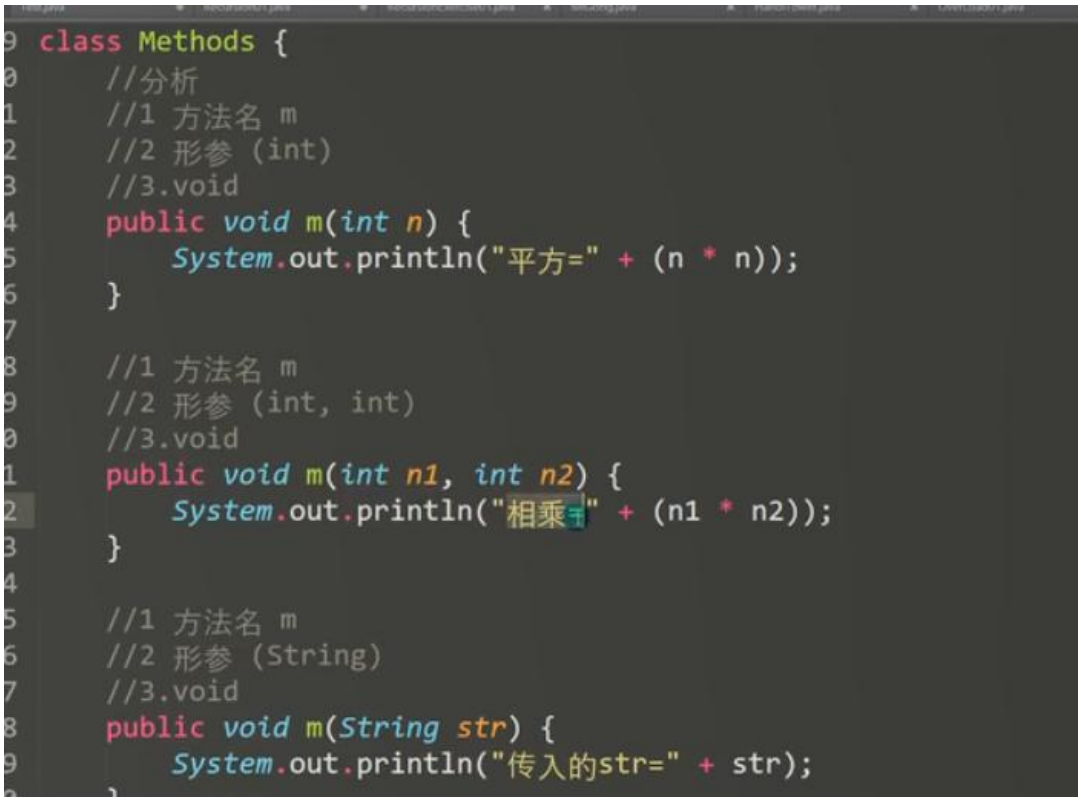
```
                //假定可以走通
                map[i][j] = 2;
                //使用找路策略类确定该位置是否真的可以走通
                //下->右->上->左
                if(findWay(map,i+1,j)){//先走下
                    return true;
                }else if(findWay(map,i,j+1)){//右边
                    return true;
                }else if(findWay(map,i-1,j)){//上
                    return true;
                }else if(findWay(map,i,j-1)){//左
                    return true;
                }else{
                    map[i][j] = 3;
                    return false;
                }
            }else{ //map[i][j] = 1,2,3
                return false;
            }
        }

    }
    //修改找路策略，看看路径是否变化
    //下->右->上->左  ===>  上->右->下->左
        public boolean findWay2(int[][] map, int i, int j){
        if(map[6][5] == 2){
            return true;
        }else {
            if(map[i][j] == 0){//当前这个位置为 0，表示可以走
                //假定可以走通
                map[i][j] = 2;
                //使用找路策略类确定该位置是否真的可以走通
                //上->右->下->左
                if(findWay2(map,i-1,j)){//先走上
                    return true;
                }else if(findWay2(map,i,j+1)){//右边
                    return true;
                }else if(findWay2(map,i+1,j)){//下
                    return true;
                }else if(findWay2(map,i,j+1)){//左
                    return true;
                }else{
                    map[i][j] = 3;
                    return false;
```

```
                }
            }else{ //map[i][j] = 1,2,3
                return false;
            }
        }

    }
}
```

方法重载（OverLoad）
java 中允许同一个类中，有多个同名方法的存在，但是要求形参列表不一致！
重载减轻了命名的麻烦

注意：（1）方法名必须相同；（2）形参列表必须不同（形参的类型或者个数或者顺序，至少有一样不同，但是参数名无要求）；（3）返回类型无要求。





```java
class Methods {
    //分析
    //1 方法名 m
    //2 形参 (int)
    //3.void
    public void m(int n) {
        System.out.println("平方=" + (n * n));
    }

    //1 方法名 m
    //2 形参 (int, int)
    //3.void
    public void m(int n1, int n2) {
        System.out.println("相乘=" + (n1 * n2));
    }

    //1 方法名 m
    //2 形参 (String)
    //3.void
    public void m(String str) {
        System.out.println("传入的str=" + str);
```

可变参数：java 允许将同一个类中多个同名同功能但参数个数不同的方法，封装成一个方法。
其基本语法为：
访问修饰符 返回类型 方法名(数据类型... 形参名){
}

注意事项：
（1）可变参数的实参可以为 0 个或任意多个；
（2）可变参数的实参可以为数组；
（3）可变参数的本质就是数组；
（4）可变参数可以和普通类型的参数一起放在形参列表，但是必须保证可变参数在最后；
（5）一个形参列表中只能出现一个可变参数。

作用域：
1. 在 java 编程中，主要的变量就是属性（成员变量=全局变量）和局部变量；
2. 我们说的局部变量一般是指在成员方法中定义的变量；
3. java 中作用域的分类：全局变量：也就是属性，作用域为整个整体；局部变量：也就是除了属性之外的其他变量，作用域为定义它的代码块中；
4. 全局变量（属性）可以不赋值，可以直接使用。因为有默认值，局部变量必须赋值后才能使用，因为没有默认值。

注意事项：
1. 属性（全局变量）和局部变量可以重名，访问时遵循就近原则；

2. 在同一个作用域中，比如在同一个成员方法中，两个局部变量不能重名；
3. 属性的生命周期较长，伴随着对象的创建而创建，伴随着对象的销毁而销毁。局部变量，生命周期较短，伴随着它的代码块的执行而创建，伴随着代码块的结束而死亡；
4. 作用域范围不同：全局变量（属性）：可以被本类使用或其他类使用（通过对象调用）；局部变量：只能在本类中对应的方法中使用；
5. 修饰符不同：全局变量（属性）：可以加修饰符；局部变量：不可以加修饰符。

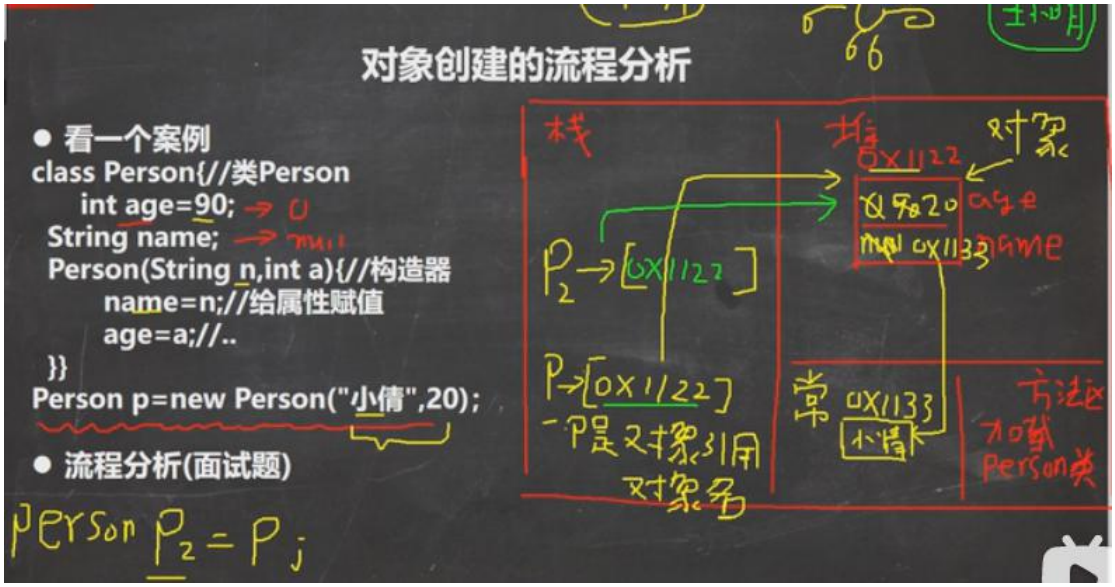
构造方法/构造器：
构造方法又叫构造器（constructor），是类的一种特殊的方法，它的主要作用是完成对新对象的初始化，有几个特点：
1）方法和类名相同；
2）没有返回值；
3）在创建对象时，系统会自动的调用该类的构造器完成对对象的初始化。

基本语法：
[修饰符] 方法名（形参列表）{
　　方法体;
}
说明：
（1）构造器的修饰符可以默认；
（2）构造器没有返回值；
（3）方法名和类名字必须一样；
（4）参数列表和成员方法一样的规则；
（5）构造器的调用由系统完成。

注意事项：
1. 一个类可以定义多个不同的构造器，即构造器重载；
2. 构造器名和类名要相同；
3. 构造器没有返回值；
4. 构造器是完成对象的初始化，并不是创建对象；
5. 在创建对象时，系统自动的调用该类的构造方法。
6. 如果程序员没有定义构造器，系统会自动给类生成一个默认无参构造器（也叫默认构造器），比如 Person(){};可以使用 javap 反编译.class 文件
7. 一旦定义了自己的构造器，默认的无参数构造器就被覆盖了，就不能再使用默认的无参构造器了，除非显式地定义一下，即 Dog(){}

```
//构造器
//
public class Constructor01{
    public static void main(String[] args){
        //当我们 new  一个对象时，直接通过构造器
        Person p1 = new Person("smith", 80);
```

```
        System.out.println("p1 的信息如下");
        System.out.println("p1 对象  name= " + p1.name);
        System.out.println("p1 对象  age = " + p1.age);
    }
}

class Person{
    String name;
    int age;
    //构造器
    //1. 构造器没有返回值，也不能写 void
    //2. 构造器的名称和类 Person 一样
    //3. (String pName, int pAge) 是构造器形参列表，规则和成员方法一样
    public Person(String pName, int pAge){
        System.out.println("构造器被调用~~ 完成对象的属性初始化");
        name = pName;
        age = pAge;
    }
    public Person(String pName){
        name = pName;
    }
}
```



流程分析：

1. 加载 person 类信息（Person.class），只会加载一次；
2. 在堆中分配空间（地址）；
3. 完成对象初始化[(1)默认初始化 age = 0 name = null，(2)再进行显式初始化 age = 90 name = null ，(3)构造器的初始化 age = 20 name = 小倩]；
4. 把对象在堆中的地址返回给 p（p 为对象名或叫做对象的引用）。

this 关键字

```java
class Dog{ //类

    String name;
    int age;
    // public Dog(String dName, int  dAge){//构造器
    //   name = dName;
    //   age = dAge;
    // }
    //如果我们构造器的形参，能够直接写成属性名，就更好了
    //但是出现了一个问题，根据变量的作用域原则
    //构造器的name 是局部变量，而不是属性
    //构造器的age  是局部变量，而不是属性
    public Dog(String name, int  age){//构造器
        name = name;
        age = age;
    }

    public void info(){//成员方法,输出属性x信息
        System.out.println(name + "\t" + age + "\t");
    }
}
```

局部变量和属性会进行重叠，进而导致混乱

Java 虚拟机会给每个对象分配 this，代表当前对象。

```java
    //如果我们构造器的形参，能够直接写成属性名，就更好了
    //但是出现了一个问题，根据变量的作用域原则
    //构造器的name 是局部变量，而不是属性
    //构造器的age  是局部变量，而不是属性
    //==> 引出this关键字来解决
    public Dog(String name, int  age){//构造器
        //this.name 就是当前对象的属性name
        this.name = name;
        //this.age 就是当前对象的属性age
        this.age = age;
    }

    public void info(){//成员方法,输出属性x信息
        System.out.println(name + "\t" + age + "\t");
    }
}
```

简单来说：哪个对象调用，this 就代表哪个对象

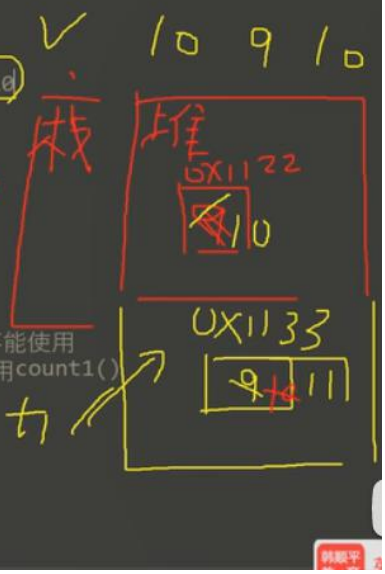
This 使用注意事项：

1. this 关键字可以用来访问本类的属性、方法、构造器；
2. this 用于区分当前类的属性和局部变量；
3. 访问成员方法的语法：this.方法名(参数列表)；

4. 访问构造器语法：this(参数列表);注意只能在构造器中使用；
5. this 不能在类定义的外部使用，只能在类定义的方法中使用。

```java
class T {

    /*
    细节: 访问构造器语法：this(参数列表);
    注意只能在构造器中使用(即只能在构造器中访问另外一个构造器)

    注意： 访问构造器语法：this(参数列表); 必须放置第一条语句
     */

    public T() {
        //这里去访问 T(String name, int age) 构造器
        this("jack", 100);
        System.out.println("T() 构造器");

    }

    public T(String name, int age) {

        System.out.println("T(String name, int age) 构造器");
    }
```

```java
public class Test {   //公有类
    int count = 9; //属性
    public void count1() { //Test类的成员方法
            count=10;//这个count就是属性  改成 10
            System.out.println("count1=" + count); //10
    }
    public void count2() {  //Test类的成员方法
        System.out.println("count1=" + count++); 9->10
    }

    //这是Test类的main方法，任何一个类，都可有main
    public static void main(String args[]) {
        //老韩解读
        //1.  new Test() 是匿名对象，  匿名对象使用后，就不能使用
        //2.  new Test().count1() 创建好匿名对象后，就调用count1()
        new Test().count1();

        Test t1= new Test();
        t1.count2();
        t1.count2();
    }
}
```



IDEA 的使用
IDEA 常用的快捷键：
1. 删除当前行，ctrl+y；
2.复制当前行，自己配置 ctrl+d；
3.补全代码 alt + /；
4.添加注释和取消注释 ctrl + /[第一次是添加注释，第二次是取消注释]。
5.导入改行需要的类 先配置 auto import，然后使用 alt+enter；
6.快速格式化代码 ctrl+shift+L
7.快速运行程序 alt+R
8.生成构造方法等 alt+insert【提高开发效率】
9.查看一个类的层级关系 ctrl+H【学习继承】
10.将光标放在一个方法上，输入 ctrl+B，可以选择定位到那个方法类的方法
11.自动分配变量名，通过在后面加.var

IDE（集成开发环境） IDEA 的使用
模版：file -> setting -> editor -> Live templates -> 查看有哪些模板快捷键/自己增加模板
Sout -> System.out.println()
Main -> public static void main(String[] args)
Fori -> for(int i=0; i< ; i++){}

包
包的三大作用：
1. 区分相同名字的类；
2. 当类很多时，可以很好的管理类[看 Java API 文档]
3. 控制访问范围

包的基本语法
package com.ganedu;
1. package 关键字，表示打包；
2. com.ganedu：表示包名。

包的本质：实际上就是创建不同文件夹（目录）来保存文件

包的命名：
命名规则：只能包含数字、字母、下划线、小圆点，但是不能用数字开头，不能是关键字或保留字
demo.class.exec1 // 错误 -> class 是关键字
demo.12a // 错误 -> 不能以数字开头 12a
demo.ab12.oa // 正确

命名规范
一般是小写字母+小圆点
com.公司名.项目名.业务模块名

比如：com.wgedu.oa.model; com.wgedu.oa.controller;
举例：
com.sina.crm.user //用户模块
com.sina.crm.order //订单模块
com.sina.crm.utils //工具类

Java 中常用的包：
一个包下，包含有很多的类，java 中常用的包有：
Java.lang.*　　//lang 包是基本包，默认引入，不需要再引入
Java.util.*　　//util 包，系统提供的工具包，工具类，使用 Scanner
Java.net.*　　//网络包，网络开发
Java.awt.*　　//做 java 的界面开发，GUI

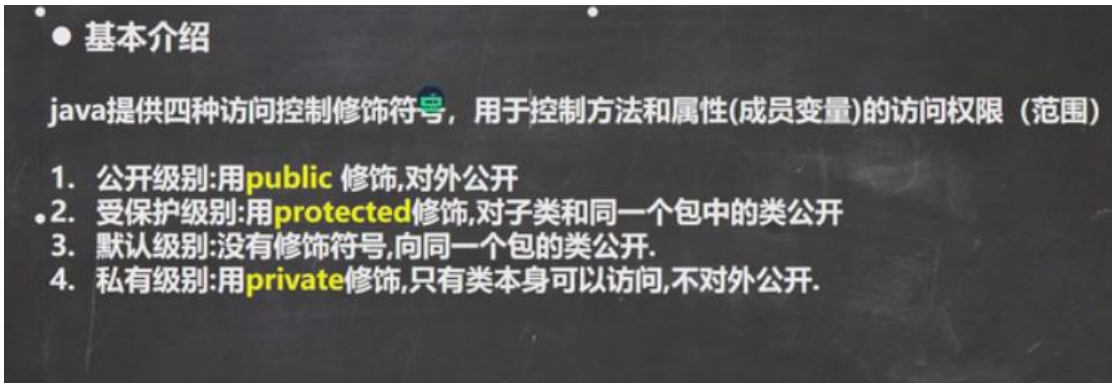
如何导入包：
import java.util.Scanner;　　//表示只会引入 java.util 包下的 Scanner
import java.util.*;　　//表示将 java.util 包下的所有类都引入（导入）
建议需要哪个类就导入哪个类（即第一种方式）

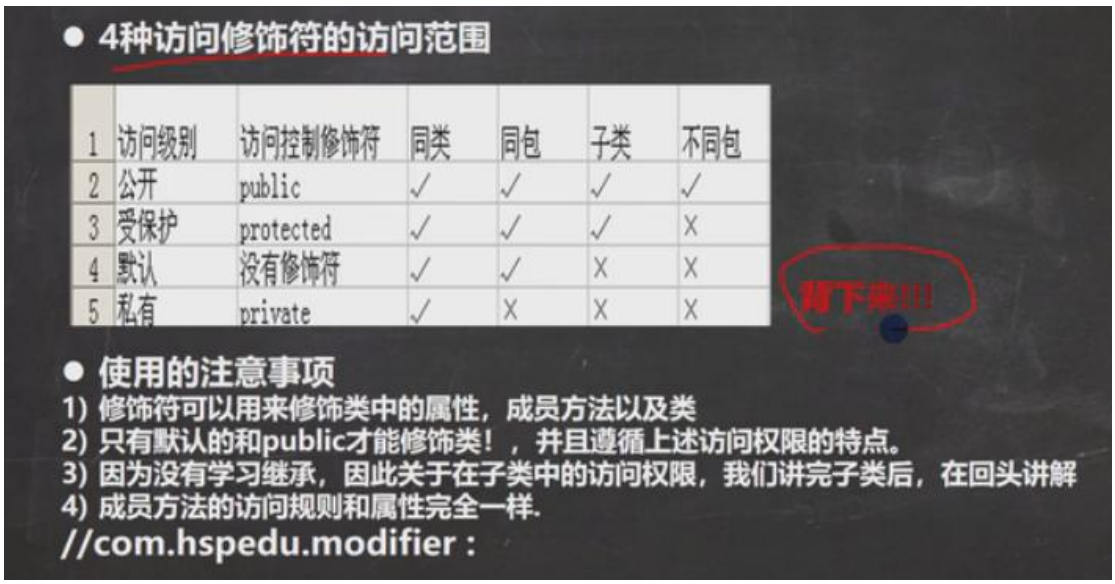
包的三大作用：1.区分相同名字的类；2.当类很多时，可以很好的管理类[看 java API 文档]；3.控制访问范围。

包的基本语法：
package com.wenganedu;
说明：1.package 关键字，表示打包；2.com.wenganedu:表示打包

注意事项和使用细节：
1. package 的作用是声明当前类所在的包，需要放在类的最上面，一个类中最多只有一句 package；
2. import 指令位置放在 package 的下面，在类定义前面，可以有多句且没有顺序要求。

访问修饰符：





成员方法的访问规则和属性完全一样

面向对象编程的三大特征
封装、继承和多态

封装（encapsulation）就是把抽象出的数据（属性）和对数据的操作（方法）封装在一起，数据被保护在内部，程序的其他部分只有通过被授权的操作（方法），才能对数据进行操作。

封装的好处：
1. 隐藏实现细节 方法（连接数据库）<--调用（传入参数）；
2. 可以对数进行验证，保证安全合理

封装的实现步骤：
1）将属性进行私有化 private [不能直接修改属性]；
2）提供一个公共的(public) set 方法，用于对属性判断并且赋值

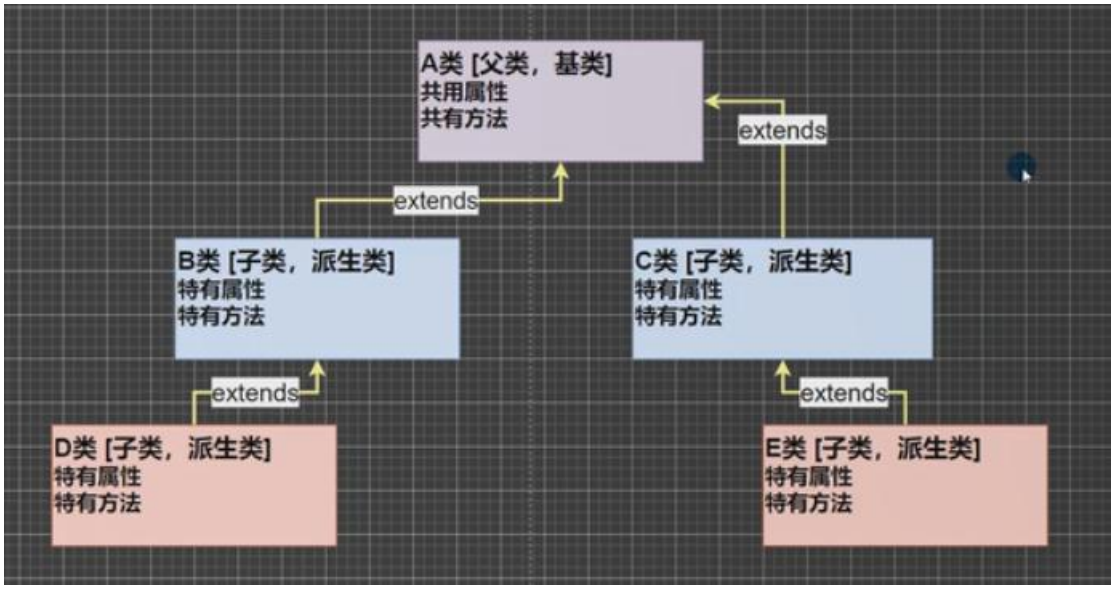
```
public void setXxx(类型 参数名){//Xxx 表示某个属性
        //加入数据验证的业务逻辑
        属性 = 参数名；
}
```

3）提供一个公共的(public)get 方法，用于获取属性的值

```
public 数据类型 getXxx(){//权限判断
    return xx;
}
```

继承->增加代码复用性

继承：当多个类存在相同的属性（变量）和方法时，可以从这些类中抽象出父类，在父类中定义这些相同的属性和方法，所有的子类不需要重新定义这些方法，只需要通过 extends 来声明继承父类即可。



继承的基本语法：
class 子类 extends 父类{

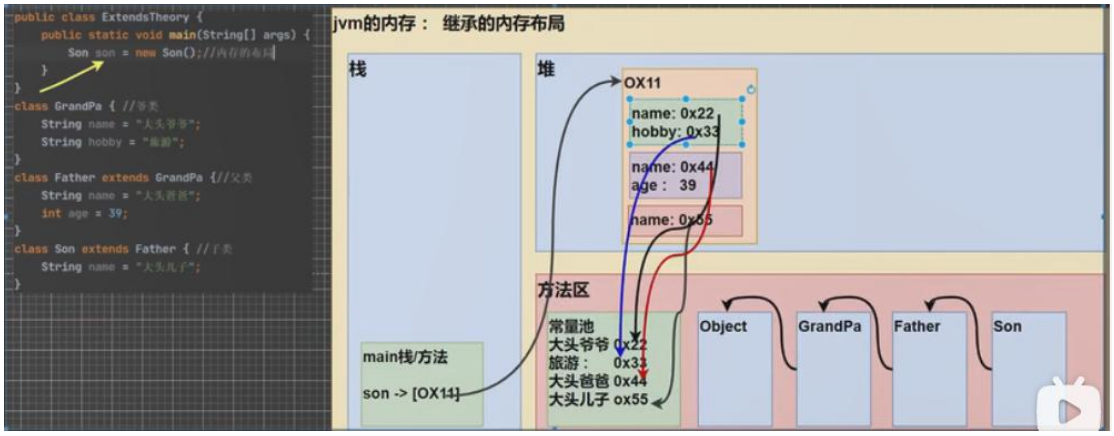
}
（1）子类会自动拥有父类定义的属性和方法
（2）父类又叫做超类、基类
（3）子类又叫做派生类

继承细节：
1. 子类继承了所有的属性和方法，非私有的属性和方法可以在子类直接访问，但是私有属性不能在子类直接访问，要通过公共的方法区访问
2. 子类必须调用父类的构造器，完成父类的初始化
3. 当创建子类对象时，不管使用子类的哪个构造器，默认情况下总会去调用父类的**无参构造器**，如果父类没有提供无参构造器，则必须在子类的构造器中用 super 去指定使用父类的哪个构造器完成对父类的初始化工作。
4. 如果希望指定去调用父类的某个构造器，则显式地调用一下 super(参数列表)
5. super 在使用时，必须放在构造器第一行，super 关键字只能在构造器中使用，不能在成员方法中使用（用了 this 就不能再用 super，反之亦然）
6. super()和 this()都只能放在构造器第一行，因此这两个方法不能共存在一个构造器
7. java 所有类都是 Object 类的子类，Object 类是所有类的基类
8. 父类构造器的调用不限于直接父类！将一直往上追溯直到 Object 类（顶级父类）
9. 子类最多只能继承一个父类（指直接继承），即 java 中是单继承机制

10.不能滥用继承，子类和父类之间必须满足 is-a 的逻辑关系

```java
package com.hspedu.extend_;

public class Sub extends Base { //子类

    public Sub(String name, int age) {
        //1. 老师要调用父类的无参构造器
        super();//父类的无参构造器
        System.out.println("子类Sub(String name, int age)构造器被调用....");
    }

    public Sub() {//无参构造器
        //super(); //默认调用父类的无参构造器
        super("smith", 10);
        System.out.println("子类Sub()构造器被调用....");
    }
```

继承的本质分析：当子类对象创建好后，建立查找关系。



只要是构造器就会默认有 super()，有 this()就不会有 super()

super 代表父类的引用，用于访问父类的属性、方法、构造器
（1）访问父类的属性，但不能访问父类的 private 属性--super.方法名（参数列表）；
（2）访问父类的方法，不能访问父类的 private 方法--super.方法名（参数列表）；
（3）访问父类的构造器--super（参数列表）；只能放在构造器的第一句且只能出现一句。

super 给编程带来的便利和细节
（1）调用父类的构造器的好处（分工明确，父类属性由父类初始化，子类的属性由子类初始化）
（2）当子类中有和父类中的成员（属性和方法）重名时，为了访问父类的成员，必须通过 super。如果没有重名，使用 super、this、直接访问是一样的效果
（3）super 访问不限于直接父类，如果爷爷类和本类中有同名的成员，也可以使用 super 去访问爷爷类的成员；如果多个基类（上级类）中都有同名的成员，使用 super 访问遵循就近原则。A->B->C

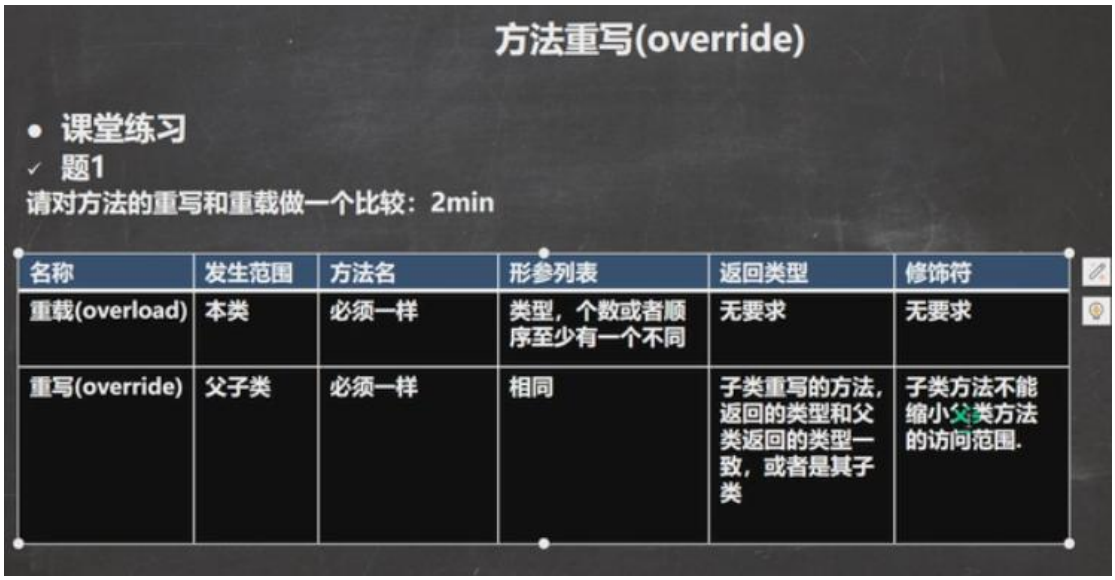
super关键字

- super和this的比较

No.	区别点	this	super
1	访问属性	访问本类中的属性，如果本类没有此属性则从父类中继续查找	从父类开始查找属性
2	调用方法	访问本类中的方法,如果本类没有此方法则从父类继续查找.	从父类开始查找方法
3	调用构造器	调用本类构造器，必须放在构造器的首行	调用父类构造器，必须放在子类构造器的首行
4	特殊	表示当前对象	子类中访问父类对象

方法重写/覆盖（override）：
简单来讲，方法重写（覆盖）就是子类有一个方法，和父类的某个方法的名称、返回类型、参数都一样，那么就说子类的这个方法覆盖了父类的方法。

方法重写需要满足以下的条件：
（1）子类的方法的参数、方法名称，要和父类方法的参数、方法名完全一样；
（2）子类方法的返回类型和父类方法返回类型一样，或者是父类返回类型的子类，比如：父类返回类型的 Object，子类方法返回类型是 String；
（3）子类方法不能缩小父类方法的访问权限。（比如父类的 public，子类就不能是 protected 等比 public 小的）



名称	发生范围	方法名	形参列表	返回类型	修饰符
重载(overload)	本类	必须一样	类型，个数或者顺序至少有一个不同	无要求	无要求
重写(override)	父子类	必须一样	相同	子类重写的方法，返回的类型和父类返回的类型一致，或者是其子类	子类方法不能缩小父类方法的访问范围.

面向对象编程--多态
基本介绍：方法或对象具有多种形态，是面向对象的第三大特征，多态是建立在封装和继承的基础之上的。

多态的具体体现：
1. 方法的多态：重写和重载就体现多态
2. 对象的多态：（1）一个对象的编译类型和运行类型可以不一致（2）编译类型在定义对象时就确定了，不能改变（3）运行类型是可以变化的（4）编译类型看定义时等号的左边，运行类型看等号的右边

多态的注意事项以及细节：
多态的前提是：两个对象（类）存在继承关系

多态的向上转型：
1）本质：父类的引用指向了子类的对象
2）语法：父类类型 引用名 =new 子类类型（）
3）特点：编译类型看左边，运行类型看右边，可以调用父类中所有的成员（需要遵守访问权限）；不能调用子类中特有成员；最终运行效果看子类的具体体现。

```java
//向上转型：父类的引用指向了子类的对象
//语法：父类类型引用名 = new 子类类型();
Animal animal = new Cat();
Object obj = new Cat();//可以吗？可以 Object 也是 Cat的父类

//向上转型调用方法的规则如下：
//(1)可以调用父类中的所有成员(需遵守访问权限)
//(2)但是不能调用子类的特有的成员
//(#)因为在编译阶段，能调用哪些成员，是由编译类型来决定的
//animal.catchMouse();错误
//(4)最终运行效果看子类(运行类型)的具体实现，即调用方法时，按照从子类(运行类型)开始查找方法
//，然后调用，规则我前面我们讲的方法调用规则一致。
animal.eat();//猫吃鱼..
animal.run();//跑
animal.show();//hello,你好
```

多态的向下转型：

1）语法：子类类型 引用名 = （子类类型） 父类引用

2）只能强转父类的引用，不能强转父类的对象

3）要求父类的引用必须指向的是当前目标类型的对象

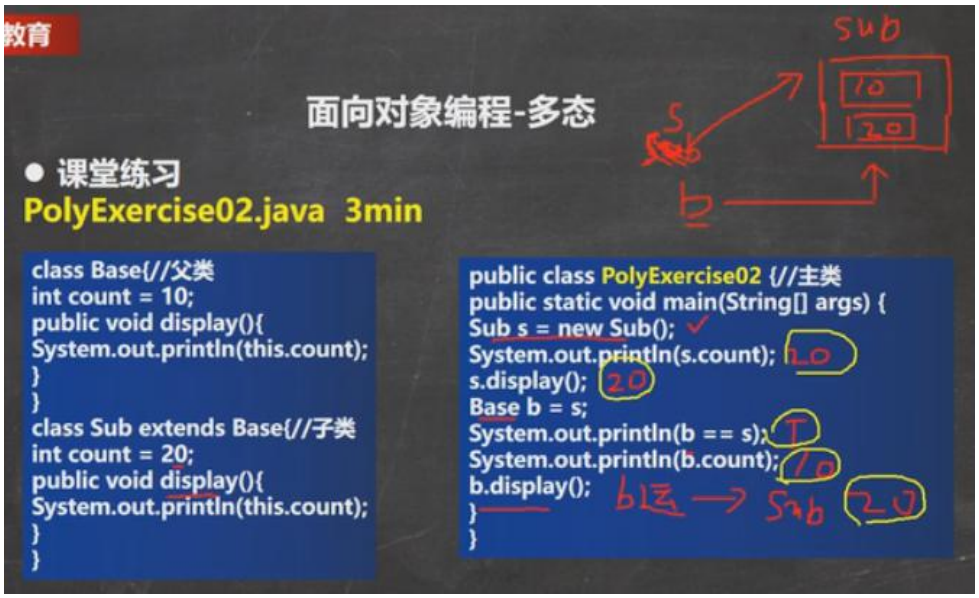
4）当向下转型后，可以调用子类类型中所有的成员

```java
    //老师希望，可以调用Cat的 catchMouse方法
    //多态的向下转型
    //(1)语法：子类类型 引用名 =（子类类型）父类引用;
    //问一个问题？ cat 的编译类型 Cat,运行类型是 Cat
    Cat cat = (Cat) animal;
    cat.catchMouse();//猫抓老鼠
    //(2)要求父类的引用必须指向的是当前目标类型的对象
    Dog dog = (Dog) animal; //可以吗？

    System.out.println("ok~~");
}
```

属性没有重写一说，属性的值看编译类型

instanceOf 比较操作符，用于判断对象的类型是否为 XX 类型或 XX 类型的子类型

```java
请说出下面的每条语言，哪些是正确的，哪些是错误的，为什么？2min后老师评讲
public class PolyExercise01{
        public static void main(String[] args) {
                double d = 13.4; //ok
                long l = (long)d; //ok
                System.out.println(l);  //13
                int in = 5; //ok
                boolean b = (boolean)in; //不对，boolean -> int
                Object obj = "Hello"; //可以，向上转型
                String objStr = (String)obj;  //可以,向下转型
                System.out.println(objStr); // hello

                Object objPri = new Integer(5);//可以，向上转型    String str =
(String)objPri; //错误ClassCastExcetpion，指向Integer的父类引用，转成String
                Integer str1 = (Integer)objPri; //可以，向下转型
        }}
```



Java 的**动态绑定机制**（非常重要!!!）

1. 当调用对象方法的时候，该方法会和该对象的内存地址/运行类型绑定
2. 当调用对象属性时，没有动态绑定机制，如果哪里声明了就在哪里使用。（属性是没有动态绑定机制的!!!）

instanceof 语句可以用于判断某一个变量的类型：比如，worker instanceof Person 即代表判断 worker 变量是不是 Person 类型

属性看编译，方法看运行

多态的应用：

1）多态数组：数组的定义类型为父类类型，里面保存的实际元素类型为子类类型；

2）多态参数：方法定义的形参类型为父类类型，实参类型允许为子类类型；

Object 类详解：

（1）equals 方法

==和 equals 方法的对比：

1）==：既可以判断基本类型，又可以判断引用类型

2）==：如果判断基本类型，判断的是值是否相等，示例：int i=10;double d-10.0；

3）==：如果判断引用类型，判断的是地址是否相等，即判断是不是同一个对象

4）equals：equals 是 Object 类中的方法，只能判断引用类型

5）默认判断的是地址是否相等，子类中往往重写该方法，用于判断内容是否相等，比如：Integer,String

Object 的 equals()方法判断的是两个对象是不是同一个对象，和==一个意思（即比较的是地址，地址不一样）
要改变的话需要对 equals()方法进行重写，重写后就不和==一样了

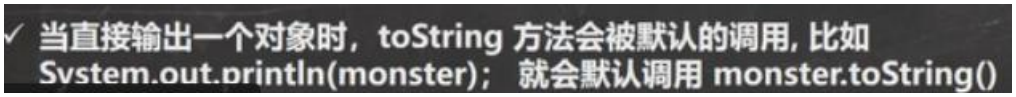
hashCode 方法：
1）提高具有哈希结构的容器的效率；
2）两个引用，如果指向的是同一个对象，则哈希值肯定是一样的；
3）两个引用，如果指向的是不同对象，则哈希值是不一样的；
4）哈希值主要是根据地址号来确定的，不能完全将哈希值等价于地址

toString 方法:
默认返回：全类名（包名+类名） + @ + 哈希值的十六进制 【查看 Object 的 toString 方法】
子类往往重写 toString 方法，用于返回对象的属性信息

重写 toString 方法，打印对象或拼接对象时，都会自动调用该对象的 toString 形式

当直接输出一个对象时，toString 方法会被默认的调用

```
System.out.println("==当直接输出一个对象时，toString 方法会被默认的调用==");
System.out.println(monster); //等价 monster.toString()
```




```
/*
Object的toString() 源码
(1)getClass().getName() 类的全类名(包名+类名 )
(2)Integer.toHexString(hashCode()) 将对象的hashCode值转成16进制字符串
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
 */
```