Java基础代码笔记

一、新版本特性

> java XXX.java

直接编译源文件，跳过javac，不生成class文件，是为了简化开发

二、入门

类、方法

```
public class XXX  //这是类体
{
 ///
    public static void main(String[] args) //这是方法
    {

    }
}
```

> 类体里面只能放方法，放语句会编译错误

打印输出

```
System.out.println("")  //换行输出
System.out.print("")    //不换行输出
```

如果输出英文字符串 需要**英文双引号**

System是一个类名，那么说明out是一个静态变量（可以是个类new出来一个静态对象），System.out返回一个对象。

```
Account act = new Account();
System.out.println(pro); //Account@1b083826
System.out.println(pro.tostring); //Account@1b083826
```

如果println（pro）；pro参数是一个引用的话，会直接调用pro的toStringg()方法

tips

1.一个Java源文件可以生成多个class

2.public的类不是必须的，可以没有

3.在源文件中只要有一个class的定义，那必然对应生成一个class的文件

4.public可以没有，但是如果有的话，必须和源文件名保持一致

三、标识符

定义

Java源代码中，editplus中显示高亮颜色是黑色时，这个单词属于标识符，红色是SUN写好的一个类

标识符可以标识：类名，方法名，变量名，接口名，常量名。。。。

一句话搞定：凡是程序员自己有权利命名的单词都是标识符（main是标识符，但不能更改）

命名规则

属于语法机制，**必须遵守**，不遵守命名规则则编译器会报错

1. 规则i标识符只能由**数字**、**字母**（包括中文）、**下划线**、**美元符号$**组成，不能含有其他符号
2. 标识符不能以数组开头，中不能有空格
3. 关键词不能做标识符，蓝色字体 public static 等都是关键字
4. 标识符严格区分大小写，大写A和小写a是不一样的
5. 标识符理论上没有长度限制

命名规范

不必须遵守，不遵守命名规范编译器不会报错

1. 见名知意
2. 遵循**驼峰**命名（一高一低），如HelloWorld
3. 类名，接口名有特殊要求：首字母大写，后面每个单词首字母大写
4. 变量名，方法名有特殊要求：首字母小写，后面每个单词首字母大写
5. 所有常量名全部大写，单词直接用下划线衔接

关键词

sun公司开发的特殊含义的单词，**全部小写**，不可做标识符，蓝色字体 public static 等都是关键字

Java严格区分大小写，public和Public不一样，后者不是关键字

四、变量

怎么在语法级别上区分字符型和字符串型?

主要看是双引号还是单引号

单引号一定是字符型

双引号一定是字符串型

变量大小

在Java中任何数据都是有数据类型的，根据不同的数据类型，**分配不同大小的数据空间**

1字节 = 8 比特位

1比特位 = 0或1 二进制位

变量可以重新赋值，但不能重新命名，且不能同名（不管数据类型是否相同）

变量分类

局部变量：方法体之内的变量，main函数结束，该变量的内存就释放了

成员变量：方法体之外的变量

变量作用域

作用域：变量的有效范围

就近原则：选择最近的方法的那一个域

```
public class Test01
{
    public static void main(String[] args)
    {
        for(int i = 0;i<10;i++)
        {
            //
        }
        System.out.println(i) //不可以,和JavaScript不同 for循环结束，i会丢弃
    }
    public static std(String[] args)
    {
        int i
        for(i = 0;i<10;i++)
        {
            //
        }
        System.out.println(i) //可以
    }
}
//出了大括号就不认识了
```

五、数据类型

作用

数据类型用来声明变量，程序在运行过程中根据不同的数据类型分配不同大小的空间

基本数据类型

4大类，8小种：

```
//第一类：数值型
byte  short  int  long
//第二类：浮点型
float  double
//第三类：布尔型 只有两个值 true、false
boolean
//第四类：字符型 必须单引号括起来
char
```

数值型

byte　　1个字节　　最大值127

short　2个字节　　最大值32767

int　　4个字节　　2147483647是int最大值，超了这个范围可以使用long类型。

long　　8个字节

1个字节-8个二进制位　1byte = 8bit

对于以上的四个类型来说，最常用的是int.

开发的时候不用斤斤计较，直接选择使用int就行了.

tips:

在任何时候，整数型的字面量或数据默认被当成int类型处理

如果希望该字面量被当错long类型来处理，需要在字面量后面加上L/l

```
long y= 12122222221;
System.out.println(y); //错误: 整数太大，默认被当成int类型处理
long y= 12122222221L;
System.out.println(y); //打印: 12122222221
```

浮点型

float　4个字节　单精度　　10.0/3 = 3.33333

double　8个字节　双精度　　更精确　　10.0/3 = 3.3333333333

java.math.BigDecimal :专门用于财务软件（不是基本数据类型，属于引用数据类型）

哪个容量大?

　注意：任何一个浮点型都比整数型空间大

　float容量 > long容量

Java规定，任何一个浮点型数据**默认被当成double**来处理，如果向被当成float可以加上F/f

　1.0　默认double

　1.0F这才是float类型

布尔类型

在java语言中boolean类型只有两个值，没有其他值: 只有**true和false**。不像C或者C++，c语言中1和0也可以表示布尔类型。

```
boolean x = 1 //错误，int无法转换为boolean
```

字符型

char

1. char占用2个字节
2. char的取值范围:[0-65535]
3. char采用unicode的编码方式
4. char类型的字面量使用单引号括起来

5. char可以存储一个汉字（汉字占用2个字节，Java中的char类型占用2个字节）

基础数据类型的区别

整数型：byte、short、int、long

浮点型：float、double

区别：占用的空间大小不同

类型	占用字节数量	取值范围
byte	1	-128~127
short	2	-32768~32867
int	4	-2147483648 ~ 2147483647
long	8	0~65535
float	4	
double	8	
boolean	1	
char	2	

引用数据类型

String属于引用数据类型，不属于基本数据类型范畴

Java中除了基本数据类型之外，剩下的都是引用数据类型

引用数据类型后期面向对象的时候才会接触

六、字符编码

认为定义的一套转换表，规定一系列的文字对应的二进制码。涉及编码和解码两个过程，编码和解码的时候必须采用同一套字符编码，不然会出现乱码。

ASCII码采用byte进行存储

‘a’————————->97

‘A’————————->65

‘0’————————->48

‘a’——(采用ASCII进行编码)——>01100001

01100001————(采用ASCII码进行编码)————>‘a’

七、转义字符

Java中“\”负责转义，会将后面紧挨着的字符进行转义

```
char s= 't'; //普通t字符
char s= '\t'; //制表符 ，只是一个字符
```

如果只是想单纯打印单引号’或者斜杠\

```
System.out.println('\''); // 打印 ’
System.out.println('\'); // 报错，因为会转义右边的’
System.out.println('\\')// 打印 \
System.out.println("\"test\""); // 打印"test"
```

特别tips:

```
char x = '\u4e2d'
system.out.println(x) // \u为显示后面的是一个字符的unicdoe编码，编码是16进制的
```

八、基本数据类型转换

转换规则

1.八种基本数据类型，除了boolean剩下的七种都可以进行转换

```
boolean  t = 1;
System.out.println(t) // int无法转换为boolean
int  t = (int)true;
System.out.println(t); //boolean无法转换为int
```

2.如果这个整数型字面量没有超出byte/short/char的取值范围，那么照这个整数型字面量可以直接赋值给byte/short/char类型的变量

3.小容量向大容量转换称为自动类型转换，容量从小到大的排序为：byte<short(shar)<int<long<float<double，其中short和char都占用两个字节，但是char可以表示更大的正整数

```
long a = 120; //是自动类型转换    int转成long类型
long b = 120L;//不是自动类型转换 120L联合起来就是一个long类型的字面量
```

4.大容量转换成小容量，称为强制类型转换，编写时必须添加“强制类型转换符”，但运行时可能出现精度损失，谨慎使用

```
long a = 1231231323L;
int b= a;
System.out.println(b); //错误: 不兼容的类型: 从long转换到int可能会有损失
int b= (int)a;
System.out.println(b)  //打印：1231231323 编译虽然过了，还是可能会有损失精度
```

5.char+ short+byte 运算时会先转换成int再做运算

```
char a = ‘a’;
byte b = 1;
System.out.println(a+1);//错误:不兼容的类型:从int转换到short可能会有损失
//编译器不知道这个加法最后的结果是多少。只知道是int类型。
short c = (short)(a +b);。
int a = 1;
short b =a;
system.out.println(x);//不可以，编译器只知道a是int类型，不知道a中存储的是哪个值。
```

6.多种数据类型做混合运算时，最终结果时最大容量对于的类型

```
long a = 10L;
char b = 'a';
short c = 100;
int d = 30;
System.out.println(a+b+c+d); //237

int x = a+b+c+d;
System.out.println(x); // 错误: 不兼容的类型: 从long转换到int可能会有损失

int x = (int)(a+b+c+d);
System.out.println(x); //237
```

精度损失的原理

long类型100L: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 01100100

转换从int类型，会将前面的4个字节砍掉： 00000000 00000000 00000000 01100100

```
byte c = (byte)300; //300这个int类型对应的二进制:00000000 00000000 00000001 00101100
System.out.println(c);//byte占用1个字节，砍掉前3个字节，结果是:00101100 (44)
```

计算机存储时二进制补码，补码 = 反码+1

char的类型转换

1. 当一个整数赋值给char类型变量的时候，会自动转换成char字符型，最终的结果是一个字符
2. 当一个整数没有超出byte、short、char的取值范围时，这个整数可以直接赋值过去

浮点型的类型转换

Java规定，任何一个浮点型数据**默认被当成double**来处理，如果向被当成float可以加上F/f

1.0 默认double

1.0F这才是float类型

```
float e = 3.14;
System.out.println(e); //错误: 不兼容的类型: 从double转换到float可能会有损失
float e = 3.14f;
System.out.println(e); //3.14
float e = (float)3.14;
System.out.println(e); //3.14  会有精度损失
```

不一定时精确

```
int temp = 10/3;

System.out.println(temp); //3    Java规定，int类型和int类型的结果还是int类型

int temp1 =1/2;

System.out.println(temp1);//0
```

九、运算符

算数运算符

关系运算符

所有关系运算符的运算结果都是布尔类型

逻辑运算符

& 逻辑与（并且）

| 逻辑或（或者）

！ 逻辑非（取反）

&& 短路与

|| 短路或

短路与&& 和逻辑& 有什么区别

首先着两个运算符的运算结果没有任何区别，完全相同。

只不过前者会发生**短路现象**

短路现象

使用短路与&&的时候，当左边的表达式为false的时候，右边的表达式不执行，这种现象被称为短路。

使用短路或||的时候，当左边的表达式为true的时候，右边的表达式不执行，这种现象被称为短路。

```
int x = 10;
int y = 11;
System.out.println(x>y & x>y++); //false
System.out.println(y) // 12 通过这个测试，我们得知x>y++执行了

int m = 10;
int n = 11;
System.out.println(m>n && m>n++); //false
System.out.println(n) // 11  通过这个测试，我们得知x>y++没执行
```

非常重要

逻辑元素符两个要求都是布尔类型，并且最终运算结果也是布尔类型。

```
100 & true //不行，语法错误
100 & 200  //不行，没有这种语法
true & false //可以
```

赋值运算符

使用扩展运算符时+=，不会改变运算结果类型

```
i += 10和i = i +10真的是完全一样吗? 不一样，并不完全相同

byte x = 100;

x  = x +1; ////错误:不兼容的类型:从int转换到byte可能会有损失

x +=1; //可以 相当于 x = (byte)(x+1) 强转会损失精度
```

三目运算符

字符串连接运算符

+号在Java中的作用

1.求和： 当＋运算符两边都是数字类型的时候

2.字符串拼接： ＋运算符两边的"任意一边"是字符串，结果是字符串

遵循自左向右的原则，依次执行

```
int x = 10;
int a = x+x++;
System.out.println(a); //20
System.out.println(x); //11
int b = x+++x;
System.out.println(b); //23
System.out.println(x); //12
int c = x+x--;
System.out.println(c); //24
System.out.println(x); //11
int d = x + --x;
System.out.println(d); //21
System.out.println(x); //10
```

十、输入

```
java.util.Scanner s = new java.util.Scanner(System.in);//创建一个键盘扫描器对象

int i = s.nextInt();//接受用户的输入,从键盘上接受一个int类型的数据
String str = s.next()//接受用户的输入,从键盘上接受一个的数据
```

另一种写法

```
import java.util.Scanner;
.....
Scanner i = new Scanner(System.in);
```

十一、控制语句

分类：

1. **选择语句**if..else
2. **循环语句**for、while、do...while
3. **转向语句**break、continue、return

选择语句

if...else

if和else if括号里面一定得是布尔语句，值必须是布尔类型，int类型的会报错

```
if(布尔表达式){
    //java语句
}else{
    //java语句
}
```

if条件语句的大括号省略 相当于自动加了一个{}但是必须只能用于**一条语句**

```
if(sex)
    System.out.println("男");
else
    System.out.println("女");
```

简化改进操作

流沙式条件，逐步递进，减少条件限制

```
//原始操作
if(age<0&&age>=150)
    System.out.println("对不起,年龄值不合法");
else if(age>=0&&age<=5)
    System.out.println("婴幼儿");
else if(age>6&&age<=10)
    System.out.println("少儿");
else if(age>11&&age<=18)
    System.out.println("少年");
else if(age>19&&age<=35)
    System.out.println("青年");
else if(age>35&&age<=55)
    System.out.println("中年");
else
    System.out.println("老年");

//简化操作 -1
if(age<0||age>=150)
    System.out.println("对不起,年龄值不合法");
else if(age<=5)
    System.out.println("婴幼儿");
else if(age<=10)
    System.out.println("少儿");
else if(age<=18)
    System.out.println("少年");
else if(age<=35)
    System.out.println("青年");
else if(age<=55)
    System.out.println("中年");
```

```
else
    System.out.println("老年");

//简化操作 -2
String ages = "";
if(age<0||age>=150)
    ages="对不起,年龄值不合法";
else if(age<=5)
    ages="婴幼儿";
else if(age<=10)
    ages="少儿";
else if(age<=18)
    ages="少年";
else if(age<=35)
    ages="青年";
else if(age<=55)
    ages="中年";
else
    ages="老年";
System.out.println(ages);
```

switch语句

switch可以叫做选择语句，也可以叫做分支语句

switch(值){

case 值1：

java语句；

break；

case 值2：

java语句；

break；

default:

java语句；

}

以上是一个完整的switch语句：其中break并不是必须的，default也不是必须的

switch支持的值有哪些：**int类型**和**String类型**（JDK8支持不支持String类型），short、byte、long会自动进行类型转换

当case语句都不满足时，执行default语句

合并

```
case 值1:case 值2:case 值3:
```

执行原理

拿值与值1进行比较，如果相同，则执行该分支的Java语句，然后遇到break语句，swtich语句就结束了，如果没有break，就会出现穿透现象

循环语句

for循环

语法机制

for(初始化表达式：条件表达式：更新表达式){

java语句循环体；

}

初始表达式最先执行，且只执行一次

条件表达式必须是一个布尔类型，true或false

while循环

语法机制

while(布尔表达式){

循环体;

}

执行原理

判断布尔表达式的结果，如果为true就执行循环体，循环体结束之后，再次判断布尔表达式的结果，如果
还是true，继续执行循坏体，直到布尔表这式结果为false，while循环结束

本质上，与for循环执行原理相同

do...while循环

语法机制

do{

循环体;

}while();

注意别漏掉分号;

执行原理

先执行循环体当中的代码，执行一次循环体之后，判断布尔表达式的结果，如果为true，则继续执行
循环体，如果为false循环结束

对于do..while循环来说，循环体至少执行1次。循环体的执行次数是:1~n次

对于while循坏来说，循坏体执行次数是:0~n次

转向语句

break

一个单词成为一个完整的java语句。continue也是这样

第一个位置: switch语句当中，用来终止switch语句的执行

第二个位置: break;语句用在循环语句当中，用来终止循坏的执行

会让离它最近的循环终止，不是针对if语句的

终止指定外层循环

```
a:for(int i = 0;i<5;i++){
    b:for(int j = 0;j<5;j++){
        System.out.println(i*j);
        if(i==3){
            break a;
        }
    }
}
```

continue

终止当前“本次”循环，直接进入下一次循环继续执行。

当continue语句执行时，continue下面的代码不执行，更新表达式。

十二、方法

语法机制

[修饰符列表] 返回值类型 方法名 （实行参数列表）{

方法体;

}

注意:

[]符号叫做中括号，以上中括号的内容不是必须的，可选

方法体由Java语句构成

1. 关于修饰符列表

   修饰符列表不是必选项，是可选的。统一写成：public static

2. 返回值类型返回值类型可以是任何类型，只要是java中合法的数据类型就行，数据类型包括基本数据类型和引用数据类型，也就是说返回值类型可以是:byte short int long float double boolean char String

3. 如果返回值类型“不是void”，那么你在方法体执行结束的时候**必须使用“return值**;”这样的语句来完成“值”的返回，如果没有”return值;“这样的语句那么编译器会报错。

4. 只要有“return”关键字的语句执行，**当前方法**必然结束。

5. 形式参数列表中每一个参数都是**局部变量**，方法结束内存释放

6. 返回类型不是void **必须是100%的return语句**，在if语句中不行的，会报错，除非有else里也有return

调用：类名.方法名(实际参数列表)

方法名要注意首单词字母小写，后面单词首字母大写

方法有static,这种方法被称为：**静态方法**

类名.方法名（）；

方法没有static,这种方法被称为：**实例方法**

需要先new一个对象，再用对象调用这个方法：**对象.方法名（）；**

return和break的区别

return：终止当前最近的方法

break：终止当前switch和循环

执行时内存变化

方法只有调用的时候才会在栈内**分配空间**，并且调用就是**压栈**

方法执行结束之后，该方法所需要的**空间就会释放**，此时发生**出栈**

所以调用方法所传的参数和方法内用的参数并不在同一个内存

栈中存储

重载机制

使用重载机制，可以使功能相似的方法同名，更易于以后的代码编写

如果方法名相同的情况下，编译器会通过方法的参数类型进行方法的区分

```
public static void mian(String[] args){
    sum(1,2);    //int求和
    sum(1L,2L);  //long求和
    sum(1.0,2.0);//double求和
}
public static int sum(int a,int b){
    System.out.println(“int求和”);
    return a+b;
}
public static int sum(long a,long b){
    System.out.println(“int求和”);
    return a+b;
}
public static int sum(double a,double b){
    System.out.println(“int求和”);
    return a+b;
}
```

这样代码既美观，又便于后期代码编写

注意：代码功能必须相似

发生条件

1. 同一类中
2. 方法名相同
3. 参数列表不同 （个数、类型、顺序）

与返回值类型无关，主要看方法名和形参

```
public static int sum(){
   return 1;
}
public static long sum(){
    return 1L;
}
// 错误: 已在类 MethodTest1中定义了方法 sum()
```

参数传递

java中规定:参数传递的时候，和类型无关，不管是基本数据类型还是引用数据类型统一都是将盒子中保存的那个“值”复制一份，传递下去。

也就是说，传过去只是值，不是内存啥的，源数据不会改变

十三、面向对象

面向过程、对象的区别

从语言角度

1. 对于C语言来说，是完全面向过程的
2. 对于C++来说，是一半面向过程，一般面向对象
3. 对于Java来说，是完全面向对象

面向过程的开发方式

主要特点：注重步骤，注重的是实现这个功能的步骤、实现功能的因果关系。没有对象的概念

缺点：面向过程最主要是每一步与每一步的因果关系，其中A步骤因果关系到B步骤，A和B联合起来形成一个子模块，子模块和子模块之间文因为因果关系结合在一起，假设其中任何一个因果关系出现问题（错误），此时整个系统的运转都会出现向题。(代码和代码之间的耦合度太高，扩展力太差。）

耦合度与扩展力成反比

优点：对于小型项目，采用面向过程的方式进行开发，效率教高，不需要前期进行对象的提取，模型的建立

面向对象的开发方式

更符合人类的思维方式，耦合度低，扩展力强

术语

OOA：面向对象分析

OOD：面向对象设计

OOP：面向对象编程

整个软件开发过程中，都是采用oo进行贯穿的

实现一个软件的过程：

分析（A）—> 设计（D）-->编程（P）

类和对象的概念

类：实际上不存在，抽象的概念

对象：实际存在的个体

类的语法格式

[修饰符列表] class 类名{

//类体 = 属性+方法

//属性子在代码上以“变量”形式存在，也就是成员变量

//方法描述动作和行为

}

三大特征

封装、继承、多态

封装

优点

保证内部结构的安全

屏蔽复杂，暴露简单

在代码级别上，调用代码人员不需要关心代码的复杂实现，只需要通过一个简单的入口就可以访问。另外，类体中安全级别高的数据封装起来，外部人员不可随意访问，来保证数据的安全性

怎么实现

1. 属性私有化（使用private关键字进行修饰）
2. 对外提供简单的操作入口，set、get

继承

基本作用：子类继承父类，代码可以得到复用

重要作用：有了继承，才有了方法的覆盖和多态

相关特性

1. B类继承A类，则称A类为超类、父类、基类，B类称为子类、派生类、扩展类
2. Java只支持单继承，不支持多继承（C++支持），可以有间接继承
3. Java中规定，子类继承父类，除**构造方法**不能被继承均可被继承。但是私有的属性无法**直接**在子类访问，可以通过间接的方法
4. Java中类没有显示的继承任何类，则默认继承Object类，也就是Java语言的**根**类

```
public class ClassExtendTest
{
    public static void main(String[] args){
        Account act = new Account("1",1.0);
        System.out.println(act.getActno());
        System.out.println(act.getBalance());
```

```
        CreditAccount acts = new CreditAccount();
        acts.setActno("2");
        acts.setBalance(2.0);
        System.out.println(acts.getActno());
        System.out.println(acts.getBalance());
    }
}
class Account  //父类
{
    private String actno;
    private double balance;

    //构造方法
    public Account(){
    }
    public Account(String actno,double balance){
        this.actno = actno;
        this.balance = balance;
    }
    //实例方法
    public void setActno(String actno){
        this.actno = actno;
    }
    public void setBalance(double balance){
        this.balance = balance;
    }
    public String getActno(){
        return this.actno;
    }
    public double getBalance(){
        return this.balance;
    }
}
class CreditAccount extends Account//子类
{
//  private String actno;
//  private double balance;  继承父类 不需要再单独申明
}
```

多态

优点：降低程序耦合度，提高程序扩展力

父类型的引用 指向子类型的对象

编译阶段（**绑定父类方法**）：对于编译器来说，编译器只知道c的类型是Animal，所有编译器在检查语法的时候，会去Animal.class字节码文件中去找move（）方法，找到了绑定move（）方法，编译通过，静态绑定成功！

运行阶段（**动态绑定子类型对象的方法**）：运行阶段的时候，实际在堆内存创建的Java对象是Cat对象，所有move的时候，真正参与move对象的是一只猫，运行阶段会动态执行Cat对象的move方法。这是动态绑定！

```
public class duotai
{
    public static void main(String[] args){
```

```
        Animal a = new Animal();
        a.move(); //动物在移动

        Animal c = new Cat();
        c.move(); //猫在爬

        Animal d = new Dog();
        d.move(); //狗在跑
    }
}
class Animal  //父类
{
    public void move(){
        System.out.println("动物在移动");
    }
}
class Cat extends Animal//子类
{
    public void move(){
        System.out.println("猫在爬");
    }
}
class Dog extends Animal//子类
{
    public void move(){
        System.out.println("狗在跑");
    }
}
```

向上转型

子--------》父　自动转

向下转型

父--------》子 强转

如果你想访问的方法是子类特有的方法，需要向下转型，负责报错找不到该方法

```
Animal d = new Dog();
d.shit(); //找不到符号
Dog x = (Dog)d; //qian
d.shit()  //狗在叫
```

避免ClassCastExceptio异常的发生

instanceof

1. instanceof可以在运行阶段动态判断引用指向的对象的类型
2. instanceof运算结果只能是布尔类型

static关键字

所有static关键字修饰的都是类相关的，类级别的

都采用“类名 . ”的方式访问

修饰静态变量，修饰静态方法

空指针对象可以调用静态变量

静态代码块

语法:

static{

java语句;

}

static可以定义静态代码块，**类加载时执行，并且只执行一次。**静态代码块有这样的特征。

```
public class Person2
{
    static{
        System.out.println("一个类可以有多个代码块，但都只执行一次");
    }
    static{
        System.out.println("静态代码块在类加载时执行，自上而下");
    }
    //入口
    public static void main(String[] args){
        System.out.println("main函数");
    }
    static{
        System.out.println("并且在main方法之前执行");
    }
    //一个类可以有多个代码块,但都只执行一次
    //静态代码块在类加载时执行，自上而下
    //并且在main方法之前执行
    //main函数
}
```

静态代码块执行顺序

```
public class Person2
{
    static int j = 0;
    static{
        System.out.println(j); //可以打印，都是静态，都是在类加载时执行
    }

    static{
        System.out.println(k); //报错，非法前向引用
    }
    static int k = 0;

    int i = 0;
```

```
    static{
        System.out.println(i);//报错，静态代码块在类加载时执行，此时i的变量空间还没开
    }
    public static void main(String[] args){

    }
}
```

实例代码块

构造方法执行一次，实例方法块就执行一次，且在这之前

```
public class Person3
{
    //入口
    public static void main(String[] args){
        new Person3();
        new Person3(3);
    }
    //实例代码块
    {
        System.out.println("实例代码块");
    }
    public Person3(){
        System.out.println("无参数构造方法");
    }
    public Person3(int i){
        System.out.println("有参数构造方法"+i);
    }
    //实例代码块
    //无参数构造方法
    //实例代码块
    //有参数构造方法3
}
```

实例对象、实例方法、实例变量

关键在于static有没有加

```
class User{
    int age;//这是实例变量
    static int sex; //这是静态变量
    public static void getAge(){ //这是静态方法
        //
    }
    public  void getSex(){ //这是实例方法
        //
    }
}
User u1 = new User();
u1.getAge();//对象可以调用静态方法 ✓
User.getAge();//类可以调用静态方法 ✓
u1.getSex();//对象可以调用实例方法 ✓
User.getSex();//类不可以调用实例方法 ×

u1.age();//对象可以调用实例变量 ✓
```

```
User.age();//类不可以调用实例变量 ×
u1.sex();//对象可以调用静态变量 ✓
User.sex();//类可以调用静态变量 ✓

切记 静态方法不可调用实例变量！！！！！！！！ 访问实例变量的方法一定只能是实例方法
```

方法覆盖

子类继承父类时，当继承过来的方法无法满足当前子类业务时，子类有权利进行重新编写，即“方法覆盖”

方法覆盖又称方法重写，Ove·rride，**和方法重载不同！！！**

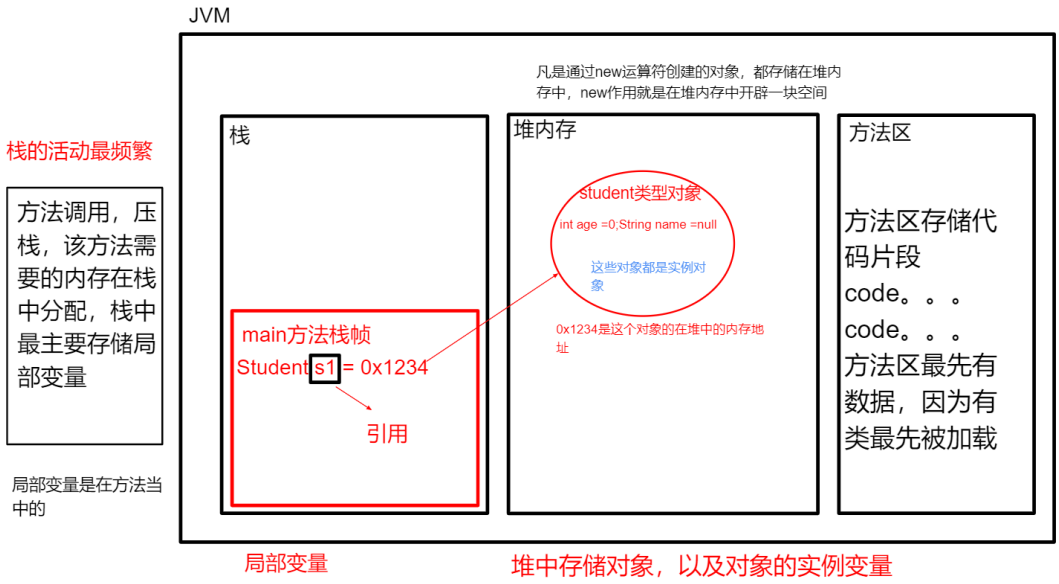
```
public class OverrideTest
{
    public static void main(String[] args)
    {
        Cat c = new Cat();
        c.move();
    }
}

class Animals
{
    public void move(){
        System.out.println("动物在移动");
    }
}
class Cat extends Animals
{
    public void move(){
        System.out.println("猫在爬");
    }
}
```

覆盖之后，一定调用覆盖之后的方法！

要求：

1. 之间要有继承关系
2. 重写与之前的方法之间具有： **相同的返回值类型、相同方法名、相同的形式参数列表**
3. 访问权限不能更低，只能更高（protected 方法不能覆盖public 方法）
4. 重写方法不能比原先方法抛出更多异常
5. **构造方法和私有方法**不能被覆盖 静态方法也不存在覆盖
6. 基本数据类型的返回值类型必须一致，但是引用数据类型可以（大变小可以，但小变大不行）

※※创建对象的JVM内存结构※※



对象：通过new出来的，在堆内存中存储

引用：但凡是变量，并且该变量中保存了内存地址指向了堆内存当中的对象的

静态变量存储在方法区

静态变量在类加载时初始化，不需要new对象，静态变量空间就开始出来了

实例变量存储在堆内存

局部变量存储在栈中

构造方法

定义

通过构造的方法完成对象的创建，以及实例变量的初始化

当一个类没有提供任何构造方法，系统回默认提供一个无参数的构造方法，这种构造方法被称为**缺省构造器**

当一个类中提供构造方法，系统将不会提供无参数构造方法

语法结构

普通方法的语法结构

[修饰符列表] 返回值类型 方法名（形式参数列表）{

    方法体;

}

构造方法的语法结构

[修饰符列表] 构造方法名（形式参数列表）{

    构造方法体

}

注意：

1. 修饰符列表目前统一写：public，**不要写public static**
2. 构造方法名和类名必须一致

3. 构造方法不需要指定返回类型，也不能写void

十三、this、super关键字

this关键字

一个对象一个this

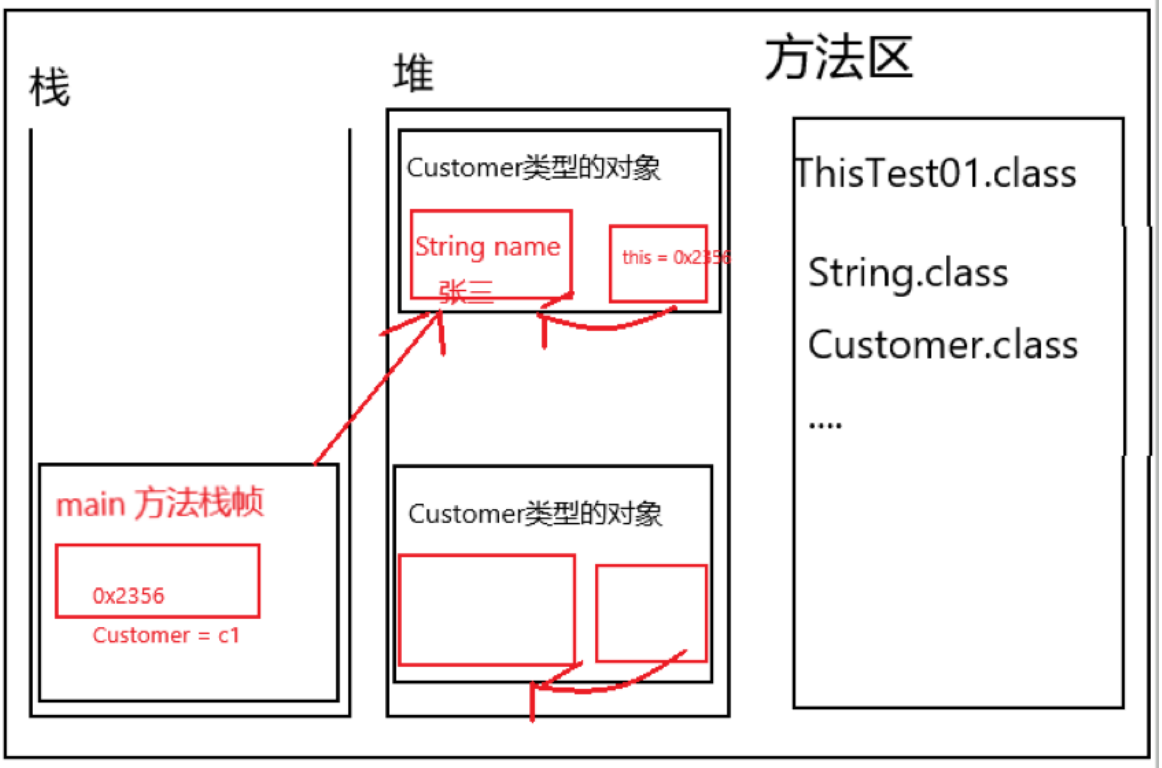
1. **this是一个关键字，是一个引用，保存内存地址指向自身**
2. **this可以使用在实例方法中，也可以使用在构造方法中**
3. **this出现在实例方法中其实代表的是当前对象**
4. **this不能使用静态方法**
5. **this. 大多情况可以省略，但是用来区分局部变量和实例变量的时候不能省略**
6. **this（）这种语法只能出现在构造方法的第一行**
7. **在实例方法中，调用实例变量可省略this. 在main中调用实例变量，不可省略this.** 也不可用this！！

代码复用 this()

this除了调用对象，还可以用作调用构造函数--—**this(.....)** 》》

```
public class  Man
{
    int id;
    String name;

    public Man(){
        //this.id =2;
        //this.name = "刘强";
        this(2,"刘强"); //代码复用,只能是第一行
    }
    public Man(int id,String name){
        this.id =id;
        this.name = name;
    }
    public static void main(String[] args)
    {
        Man n = new Man();
        System.out.println(n.id);
        System.out.println(n.name);
    }
}
```

this的内存地址图



super关键字

super() **只能出现在构造方法第一行**，通过当前的构造方法去调用父类中的构造方法，

目的:创建子类对象的时候，先初始化父类型特征。

super关键字代表的时当前对象的部分父类型特征

System.out.println(this)可以输出当前对象的toString方法;

System.out.println(super)报错，**super不可单独使用**

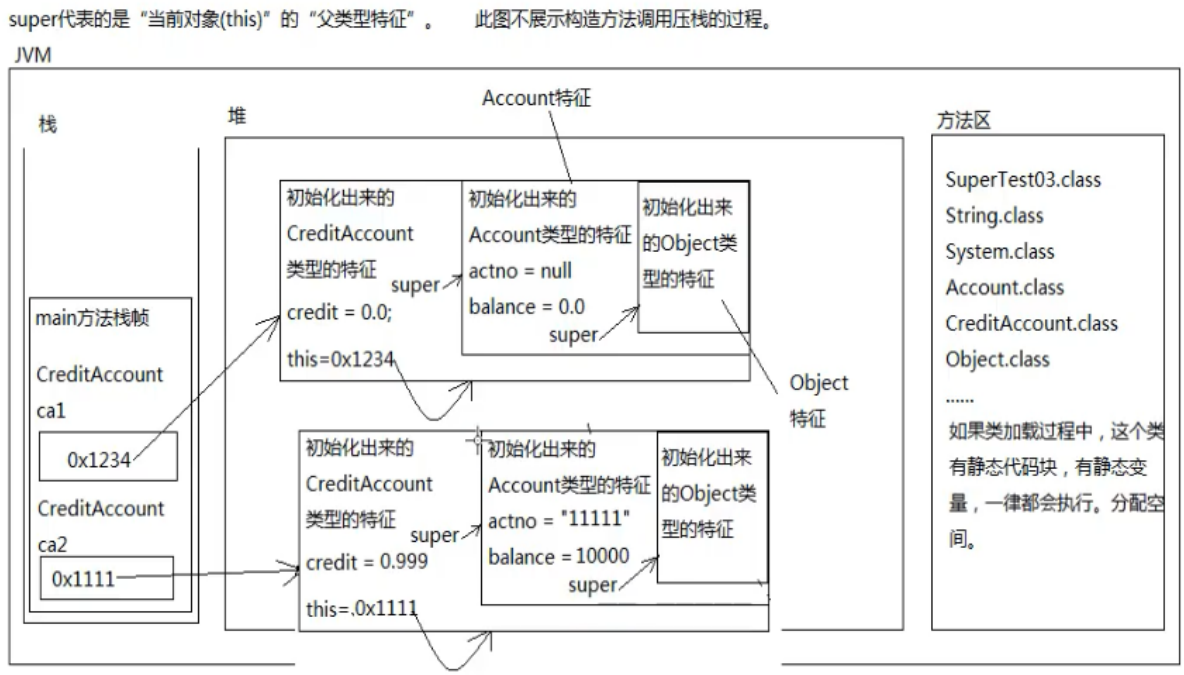
```
class superTest
{
    public static void main(String[] args)
    {
        new B();
    }
}

class A
{
//  public A(){
//      System.out.println("A的无参构造方法");
//  }
    public A(int i){
        System.out.println("A的无参构造方法"+i);
    }
}

class B extends A
```

```
{
    public B(){
        //super()    //默认存在的
        super(1);    //当子类继承父类时，子类的构造函数里面有一个默认super()会调用父类的构造函数
        System.out.println("B的无参构造方法");
    }
}

//当一个构造函数没有提供super(),this()时，默认提供super()!!!!!

//如果手动提供一个带参的构造函数，默认super()调用没有代参，执行会报错
```

小练习-判断输出结果

```
public class superTest2
{
    public static void main(String[] args)
    {
        new C();
    }
}

class A
{
    public A(){
        System.out.println("1");
    }
}
class B extends A
{
    public B(){
        System.out.println("2");
    }
    public B(int i){
        System.out.println("3");
    }
}
class C extends B
{
    public C(){
        this(1);
        System.out.println("4");
    }
    public C(int i){
        this(1,2);
        System.out.println("5");
    }
    public C(int i,int j){
        super(1);
        System.out.println("6");
    }
}


//13654
```

super内存图



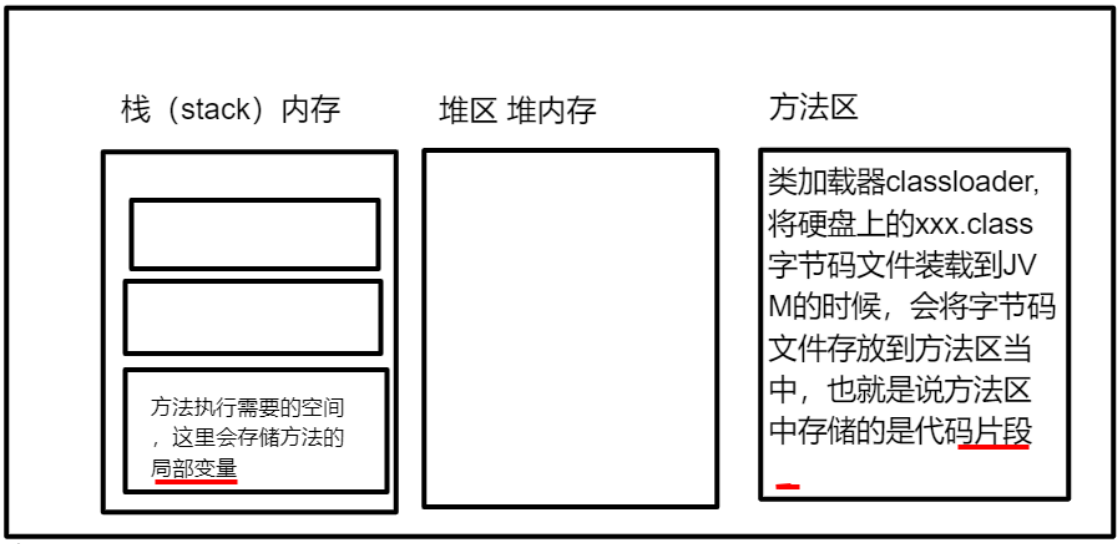
super与this的对比

	this	super
能否出现在实例方法和构造方法中	✓	✓
语法	this. 、this()	super. 、super()
能否出现在静态方法中	✕	✕
可以省略么	this.大多情况可以省略	super.大多情况可以省略
什么时候不能省略	区分局部变量和实例变量不能省略	当子类和父类都用同名属性时，this指向当前对象属性，super指向父类对象的属性
第一行要求	this()构造器第一行，与super()不能共存	super()构造器第一行，与this()不能共存
单独使用println()	✓	✕

JVM

JVM主要三块内存空间：**栈、堆、方法区**

JVM(java虚拟机)



创建于 2020-10-09 09:12

异常

空指针异常

NULLPointExecption

```
public class NULLPointTest{
    public static void main(String[] args){
        Customer c = new Customer();
        c.id = 9527;
        System.out.println(c.id);
        c.age = null;
        System.out.println(c.age);
    }
}
class Customer{
    int id; //成员变量中的实例变量，应该先创建对象，然后通过“引用.”的方式引用
    int age;
}
```

Java中垃圾回收机制GC主要针对回收的是**堆内存当中**的垃圾数据，当一个Java对象没有任何引用指向该对象的时候，GC会考虑将该垃圾数据释放回收掉

出现空指针异常的前提条件

“空引用”访问实力【对象相关】相关的数据时，都会出现空指针异常

Javadoc

```
javadoc -d XXX xxxx.java
```

XXX为生成文件夹名，xxxx.java是你想要生成Java文档的Java源文件