

java基础笔记-重点内容

语法：

枚举

关键字：

访问控制（开放程度从上到下）

常量

java基础类

java.lang包

java中类型转换与自动类型转换

自动类型转换：

强制类型转换：

java中的变量类型（成员变量，静态变量等）

Static

static方法

static变量

构建顺序：

静态导入包

java面向对象

实例化对象

java重写

java重载

重写与重载之间的区别

抽象类

三大特性

多态：

继承

规范：

1. java对大小写的敏感的

2. 类名的每个单词的首字母都应该大写
3. 方法名首个单词小写，后面单词的首字母应该大写

语法：

枚举

在类或方法中定义一个枚举，在其他地方调用时， 枚举限制变量只能在预先设定好的值中挑选

Java | 复制代码

```
1 class Juice{
2     enum JuiceSize{small,medium,large} //枚举的选项， JuiceSize的类型的枚举
3     JuiceSize size; //定义JuiceSize类型的变量size
4 }
5
6 public class Main {
7     public static void main(String[] args){
8         Juice juice = new Juice(); //new一个Juice对象
9         juice.size = Juice.JuiceSize.small;//为对象的size赋值为Juice类中枚举的
10        small
11    }
12 }
```

关键字：

访问控制（开放程度从上到下）

访问控制关键字	描述	适用于
public	公共的，对所有类可见	类、接口、成员变量、方法
protected	受保护的，只能在同一包内或子类中访问	成员变量、方法

default	默认/包访问，只能在同一包内访问	类、接口、成员变量、方法、构造函数，仅限于当前包中的类和接口
private	私有的，只能在定义他们的内部使用（作为内部类）	成员变量、方法、内部类

▼

例子

Java

复制代码

```

1  public class Main { //外部类
2      public static void main(String[] args){
3          app myApp = new app(); //new一个app对象
4          System.out.print(myApp.a); //输出里面的变量a
5      }
6
7      private static class app{ //定义一个私有类app，因为是内部类，所以能调用
8          int a=1;
9      }
10
11 }

```

常量

在java中使用final关键字定义常量，

tip：常量的方法不能被继承

java基础类

包名	内容概述
Java.applet	提供创建applet小程序所需要的类
Java.awt	包含用于创建用户界面和绘制图形图像的所有类
Java.io	提供与输入输出相关的类
Java.beans	包含与开发javaBeans相关的类

Java.lang	提供java语言程序设计的基础类
Java.net	提供实现网络操作相关的类
Java.nio	为输入输出提供缓冲区的类
Java.text	提供处理文本、日期、数字和消息的类和接口
Java.util	提供处理日期、时间、随机数生成等各种使用工具的类
Javax.net	提供用于网络应用程序的类、网络应用扩展类
Java.swing	提供一组与AWT功能相同的纯java的组件类

java.lang包

Java.lang包是java语言体系中其他所有类库的基础，已经内嵌到java虚拟机中，而且以对象的形式创建好了，所以，我们在使用java.lang包时不用导入，直接使用里面的函数。

java中类型转换与自动类型转换

自动类型转换：

指直接赋值给一个其他类型的变量

```

1
2 public class ZiDongLeiZhuan{
3     public static void main(String[] args){
4         char c1='a';//定义一个char类型
5         int i1 = c1;//char自动类型转换为int
6         System.out.println("char自动类型转换为int后的值等于"+i1);
7         char c2 = 'A';//定义一个char类型
8         int i2 = c2+1;//char 类型和 int 类型计算
9         System.out.println("char类型和int计算后的值等于"+i2);
10    }
11 }
12

```

自动类型转换

```

1 低 -----> 高
2
3 byte,short,char-> int -> long-> float -> double
4 转换从低级到高级。

```

数据类型转换必须满足如下规则：

- 1. 不能对boolean类型进行类型转换。
- 2. 不能把对象类型转换成不相关类的对象。
- 3. 在把容量大的类型转换为容量小的类型时必须使用强制类型转换。
- 4. 转换过程中可能导致溢出或损失精度。

强制类型转换：

格式：(type)value type是要强制类型转换后的数据类型

实例：

```
1 public class Main {  
2     public static void main(String[] args){  
3         int i =1;  
4         byte a = (byte)i;  
5         System.out.print(a);  
6     }  
7 }
```

java中的变量类型（成员变量，静态变量等）

```
1 public class Main {  
2     private int instanceVar; //成员变量  
3     private static int staticVar; //静态变量  
4     public void method(int paramVar){ //paramVar为参数变量  
5         int localVar = 10; //局部变量  
6         instanceVar = localVar;  
7         staticVar = paramVar;  
8         System.out.println("localVar = " + localVar);  
9         System.out.println("paramVar = " + paramVar);  
10        System.out.println("instanceVar = " + instanceVar);  
11        System.out.println("staticVar = " + staticVar);  
12    }  
13    public static void main(String[] args){  
14        Main a = new Main();  
15        a.method(20);  
16    }  
17 }
```

Static

static方法

static静态方法与非静态方法的区别：

1. static方法可以通过类名直接调用方法，如果是非静态方法的话，需要先实例化再调用方法
2. 和类一起加载的，比非静态方法快

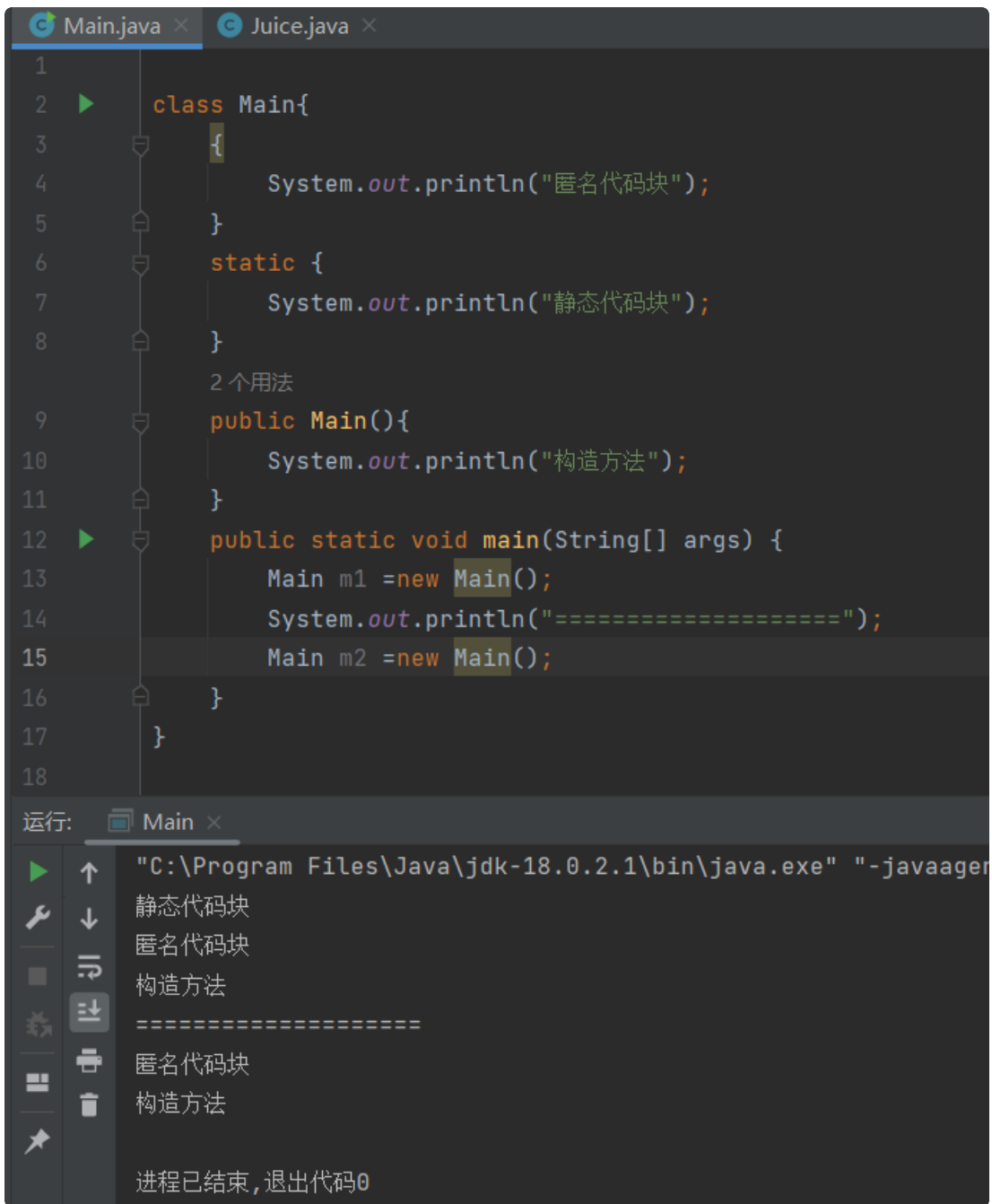
static变量

静态变量对于类中而言在内存中只有一个，在其他类中也可以使用，一般在多线程中去使用

Python | [复制代码](#)

```
1  class Main{
2      private static int age; //静态的变量
3      private double score; //非静态的变量
4      public static void main(String[] args) {
5          Main m = new Main();
6          System.out.println(Main.age);
7      //      System.out.println(Main.score); //不能直接使用类名去访问
8          System.out.println(m.score);
9          System.out.println(m.age);
10     }
11 }
```

构建顺序：



The screenshot shows an IDE with two tabs: Main.java and Juice.java. The Main.java file contains the following code:

```
1  
2 class Main{  
3     {  
4         System.out.println("匿名代码块");  
5     }  
6     static {  
7         System.out.println("静态代码块");  
8     }  
9     2 个用法  
10    public Main(){  
11        System.out.println("构造方法");  
12    }  
13    public static void main(String[] args) {  
14        Main m1 =new Main();  
15        System.out.println("=====");  
16        Main m2 =new Main();  
17    }  
18 }
```

The execution output at the bottom shows the following sequence of events:

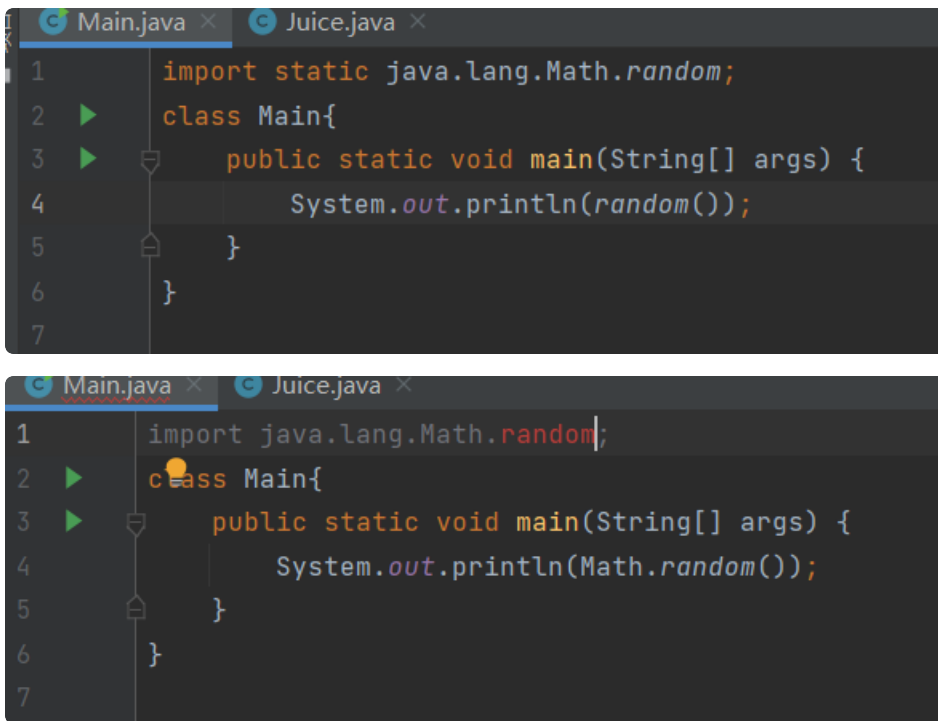
- 运行: Main x
- "C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" "-javaagen
- 静态代码块
- 匿名代码块
- 构造方法
- =====
匿名代码块
- 构造方法
- 进程已结束,退出代码0

而且可以看到，静态代码块只加载一次

静态导入包

静态导入和非静态导入的区别：

注：random是包（类） `Math` 中的一个方法，不能直接使用导入，需要使用静态导入



```
1 import static java.lang.Math.random;
2 class Main{
3     public static void main(String[] args) {
4         System.out.println(random());
5     }
6 }
7
```

```
1 import java.lang.Math.random;
2 class Main{
3     public static void main(String[] args) {
4         System.out.println(Math.random());
5     }
6 }
7
```

java面向对象

```
Student a = new Student();
```

```
Main b = new Student();
```

对象能执行哪些方法主要看左边的类型，和右边的关系不大

实例化对象

在Java中，类的实例化（使用 new 关键字创建对象）时，会自动调用与类同名的构造函数。

java重写

重写是子类对父类的允许访问的方法的实现过程进行重新编写，返回值和形参都不能改变。即外壳不变，核心重写！

```
1 class Animal{
2     public void move(){
3         System.out.println("动物可以移动");
4     }
5 }
6
7 class Dog extends Animal{
8     public void move(){
9         System.out.println("狗可以跑和走");
10    }
11    public void bark(){
12        System.out.println("狗可以吠叫");
13    }
14 }
15
16 public class TestDog{
17     public static void main(String args[]){
18         Animal a = new Animal(); // Animal 对象
19         Animal b = new Dog(); // Dog 对象
20
21         a.move();// 执行 Animal 类的方法
22         b.move();//执行 Dog 类的方法
23         b.bark();
24     }
25 }
```

这样的写法是错的，因为在19行，b的类型被定义为 `Animal`，而 `Animal` 类型中没有 `bark` 方法，所以不能调用 `bark` 方法。

可以将19行改为：`Dog b = new Dog();`

这样就可以正常运行

不能重写：

1. static方法，静态方法属于类，它不属于实例
2. final常量
3. private方法，私有类

super关键字

当需要在子类中使用父类的被重写方法时，可以使用super关键字

```
1 class Animal{
2     public void move(){
3         System.out.println("动物可以移动");
4     }
5 }
6
7 class Dog extends Animal{
8     public void move(){
9         super.move(); // 应用super类的方法
10        System.out.println("狗可以跑和走");
11    }
12 }
13
14 public class TestDog{
15     public static void main(String args[]){
16
17         Animal b = new Dog(); // Dog 对象
18         b.move(); //执行 Dog类的方法
19
20     }
21 }
```

java重载

重载(overloading) 是在一个类里面，方法名字相同，而参数不同。

```
1 public class Main{
2     public int test(){
3         System.out.println("test1");
4         return 1;
5     }
6     public void test(int a){
7         System.out.println("test2");
8     }
9     public String test(String s,int a){
10        System.out.println("test3");
11        return "returntest3";
12    }
13    public String test(int a,String s){
14        System.out.println("test4");
15        return "returntest4";
16    }
17
18    public static void main(String[] args) {
19        Main m = new Main();
20        System.out.println(m.test());
21        m.test(1);
22        System.out.println(m.test(1,"test3"));
23        System.out.println(m.test("test4",1));
24    }
25 }
```

重写与重载之间的区别

区别点	重载方法	重写方法
参数列表	必须修改	一定不能修改
返回类型	可以修改	一定不能修改
访问	可以修改	一定不能做更严格的限制（可以降低限制）

抽象类

特性：

1. 抽象类不能实例化对象，只能被继承使用，而且继承它的子类（除了抽象类），要实现其抽象父类的所有方法。（相对于给了子类一个约束）
2. 抽象类中可以写普通的方法
3. 抽象方法必须在抽象类中

使用 `abstract` 关键字定义抽象类。

```
Java | 复制代码
1 //抽象类
2 public abstract class Main{
3     //抽象方法，只有方法名字，没有方法的实现
4     public abstract void doSomething();
5 }
```

三大特性

- 封装
- 继承
- 多态

多态：

1. 多态是方法的多态，属性没有多态
2. 存在条件：
 - 继承关系
 - 子类重写父类方法（如果不能重写就没有多态，这样不同引用就可以调用不同方法）
 - 父类引用指向子类对象

```
1 class Main{
2     public static void main(String[] args) {
3         Student a = new Student();
4         Main b = new Student(); //父类的引用指向子类
5         Object c = new Student(); //Object也是父类
6         // 对象能执行那些方法主要看左边的类型，和右边关系不大。
7         a.draw();
8         //b.draw(); 错误的，因为Main类下没有draw()方法
9         b.run();
10        a.run(); //Student继承了Main， 所以有run方法
11    }
12    void run(){
13        System.out.println("run");
14    }
15 }
16
17 class Student extends Main {
18     void draw(){
19         System.out.println("draw");
20     }
21 }
```

继承

格式

```
1 public class Main{
2     public static void main(String[] args) {
3     }
4 }
5
6 class LittleMain extends Main{
7 }
```

使用extends关键词进行继承

```
1 // 父类
2 class Vehicle {
3     private String brand;
4
5     public Vehicle(String brand) {
6         this.brand = brand;
7     }
8
9     public void drive() {
10         System.out.println("Driving the vehicle");
11     }
12
13     public void stop() {
14         System.out.println("Stopping the vehicle");
15     }
16 }
17
18 // 子类
19 class Car extends Vehicle {
20     private int numOfSeats;
21
22     public Car(String brand, int numOfSeats) { //在Java中，类的实例化（使用 new
    关键字创建对象）时，会自动调用与类同名的构造函数。
23         super(brand); // 使用super调用父类的构造函数
24         this.numOfSeats = numOfSeats; //前面为成员变量，后者为参数变量
25     }
26
27     public void accelerate() {
28         System.out.println("Accelerating the car");
29     }
30 }
31
32 // 使用子类
33 public class Main {
34     public static void main(String[] args) {
35         Car car = new Car("Toyota", 5);
36         car.drive(); // 继承自父类的方法
37         car.stop(); // 继承自父类的方法
38         car.accelerate(); // 子类特有的方法
39     }
40 }
41
```

继承的特性：

- 子类拥有父类非 private 的属性、方法。
- 一个子类不能同时有两个父类

super 与 this 关键字：

- super关键字：我们可以通过super关键字来实现对父类成员的访问，用来引用当前对象的父类。
- this关键字：指向自己的引用。

使用final修饰类/方法：

- 使用 final 关键字声明类，就是把类定义定义为最终类，不能被继承，或者用于修饰方法，该方法不能被子类重写：

构造器：

如果父类的构造器带有参数，则必须在子类的构造器中显式地通过 super 关键字调用父类的构造器并配以适当的参数列表。

如果父类构造器没有参数，则在子类的构造器中不需要使用 super 关键字调用父类构造器，系统会自动调用父类的无参构造器。


```
1
2 class SuperClass {
3     private int n;
4     SuperClass(){
5         System.out.println("SuperClass()");
6     }
7     SuperClass(int n) {
8         System.out.println("SuperClass(int n)");
9         this.n = n;
10    }
11 }
12 // SubClass 类继承
13 class SubClass extends SuperClass{
14     private int n;
15
16     SubClass(){ // 自动调用父类的无参数构造器
17         System.out.println("SubClass");
18     }
19
20     public SubClass(int n){
21         super(300); // 调用父类中带有参数的构造器
22         System.out.println("SubClass(int n):"+n);
23         this.n = n;
24     }
25 }
26 // SubClass2 类继承
27 class SubClass2 extends SuperClass{
28     private int n;
29
30     SubClass2(){
31         super(300); // 调用父类中带有参数的构造器
32         System.out.println("SubClass2");
33     }
34
35     public SubClass2(int n){ // 自动调用父类的无参数构造器
36         System.out.println("SubClass2(int n):"+n);
37         this.n = n;
38     }
39 }
40 public class TestSuperSub{
41     public static void main (String args[]){
42         System.out.println("-----SubClass 类继承-----");
43         SubClass sc1 = new SubClass();
44         SubClass sc2 = new SubClass(100);
45         System.out.println("-----SubClass2 类继承-----");
```

```
46     SubClass2 sc3 = new SubClass2();
47     SubClass2 sc4 = new SubClass2(200);
48 }
49 }
```

输出:

Java | 复制代码

```
1  -----SubClass 类继承-----
2  SuperClass()
3  SubClass
4  SuperClass(int n)
5  SubClass(int n):100
6  -----SubClass2 类继承-----
7  SuperClass(int n)
8  SubClass2
9  SuperClass()
10 SubClass2(int n):200
```

父类构造器无参数=》调用子类的构造器时自动调用父类构造器

父类构造器有参数=》需要使用super(参数)进行调用父类构造器