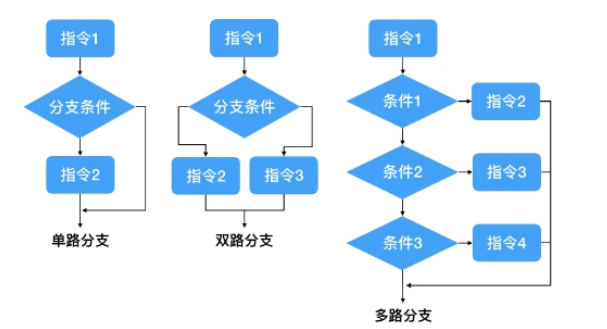
# 分支结构



## 1.1 单路分支

if(分支条件){

指令 //为true则执行，为false则跳过

}

## 1.2 双路分支

if(分支条件){ //为true则执行指令1，为false则执行指令2

指令1

}else{

指令2

}

## 1.3 多路分支

if(分支条件1){

指令1

}else if(分支条件2){

指令2

} else if(分支条件3){

指令3

} else{

指令4

}

## 1.4 switch case结构

switch只能用于整数的选择

int a = 2;

switch(a){ //数据类型为byte short int char string

case 1:

指令1;

break;

case 2:

指令2; //匹配到2，则从此处进入，若无break则执行以下所有指令

break;

case 3:

指令3;

break;

default:

指令4;

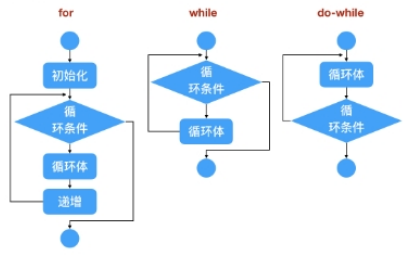
}

## 1.5条件运算符和嵌套

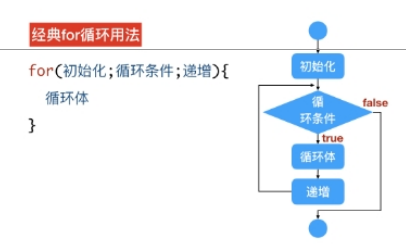
int b = a>0?1:-1

string score = a>=90? “A”:(a>=60 && a< 90?”B”:”C”);

# 2 循环结构



## 2.1 for循环



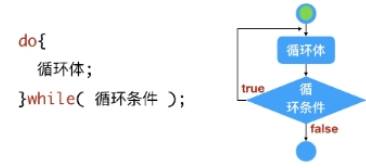
初始化：只执行一次

循环条件：true则继续循环，false则退出循环

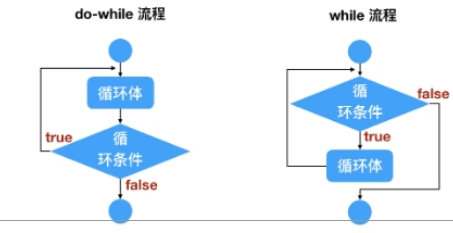
## 2.2 while结构



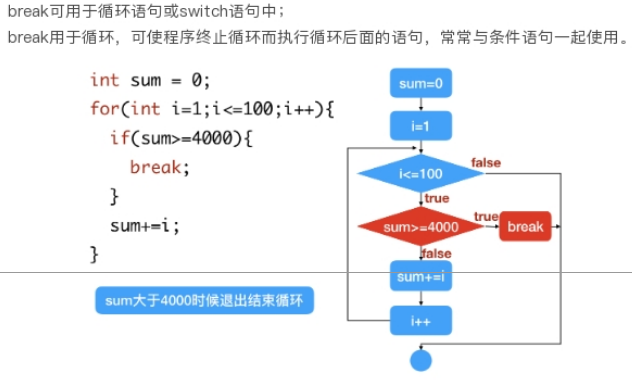
## 2.3 do...while结构



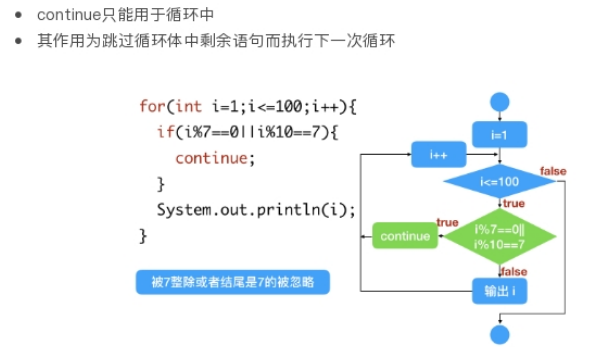
## 2.4 比较while和do...while结构



## 2.5 break语句



## 2.6 continue语句



# 3 数组 Array

## 3.1 数组的声明及初始化

**int**[] a = **new** **int**[10]; //声明一个整形数组

**double**[] b = **new** **double**[5]; //声明一个浮点数组

**boolean**[] c = **new** **boolean**[7];//声明一个布尔数组

System.***out***.println(a[0]);//默认值为0

System.***out***.println(b[0]);//默认值为0.0

System.***out***.println(c[0]);//默认值为false

//初始化方法

// int[] arr = new int[3];

// int[] arr = {2,5,8};

// int[] arr = new int[] {2,5,8};

## 3.2 数组的复制

（1）方法一

System.*arraycopy*(src, srcPos, dest, destPos, length);

/\*src:源数组

\*srcPos:源数组的起始下标

\*dest:目标数组

\*destpos:目标数组的起始下标

\*length :要复制的长度

\*/

（2）方法二

**int**[] a = Arrays.copyOf(arg0, arg1)

// arg0:源数组

// a:目标数组

// arg1:目标数组的长度

// 若源数组长度大于目标数组长度，则截取掉末尾;

// 若源数组长度小于目标数组长度，则末尾取默认值;

此方法可用于改变数组的长度，如

a = Arrays.*copyOf*(a, a.length+1);

## 3.2 数组的排序

Arrays.*sort*(a);// 给数组排序

# 4 方法 Method

## 3.1方法的声明

// 修饰词 返回值类型 方法名(参数列表) {

// 方法体

// }

**public** **static** **void** sayHi() { //无返回值无参数的方法;

System.***out***.println("你好!");

}

//无返回值有参数的方法

**public** **static** **void** sayHello(String name, **int** age) {

System.***out***.println("大家好，我叫"+name+"今年"+age+"岁了");

}

**public** **static** **int** quadraticSum(**int** x,**int** y){ //有返回值的方法

**int** z=x\*x+y\*y;

**return** z;

}

## 3.2方法的调用

方法名(参数列表);

*sayHi*();

*sayHello*("HXK",26);

**int** z = *quadraticSum*(5,12);

# 5.对象 Object

## 4.1类和对象的创建，对象的属性初始化，对象方法的调用

**public** **class** Practice07 {

**public** **static** **void** main(String[] args) {

Student zs = **new** Student(); //创建一个对象

zs.name = "张三"; //给对象的各个属性赋值

zs.age = 30;

zs.address = "河北廊坊";

zs.study(); //调用对象中的方法

zs.sayHi();

}

}

**class** Student{ //创建类

String name;

**int** age;

String address;

**void** study() { //创建类中的方法

System.***out***.println(name+"在学习...");

}

**void** sayHi() {

System.***out***.println("我叫"+name+"，今年"+age+"岁了，家住"+address);

}

}

## 4.2初始化

方法一

**public** **class** ClassInitialize {

**public** **static** **void** main(String[] args) {

Student student1 = **new** Student ("hxk",26,"湖北武汉");

}

}

**class** Student {

String name;

**int** age;

String address;

**public** Student (String aaa, **int** aa, String aaaa) {

name = aaa;

age = aa;

address = aaaa;

}

}

方法二

**class** Bubble{

**int** d;

**double** x;

**double** y;

**int** r;

**int** g;

**int** b;

Color color;

**double** offsetX;

**double** offsetY;

**public** Bubble() { //初始化

d = (**int**)(Math.*random*()\*40+20);

x = (**int**)(Math.*random*()\*(790-d));

y = (**int**)(Math.*random*()\*(590-d));

r = (**int**)(Math.*random*()\*256);

g = (**int**)(Math.*random*()\*256);

b = (**int**)(Math.*random*()\*256);

color = **new** Color(r,g,b);

offsetX = Math.*random*()\*(6 - 1) + 1;// 随机生成方向

offsetY = Math.*random*()\*(6 - 1) + 1;

{// 语句块

offsetX = Math.*random*() < 0.5 ? -offsetX : offsetX;

offsetY = Math.*random*() < 0.5 ? -offsetY : offsetY;

}

}

}

# 6. 继承 extends

## 6.2向上造型

将子类创建的对象装到父类变量里，便于集中管理

**public** **static** **void** main(String[] args) {

Aoo b1 = **new** Boo();

}

}

**class** Aoo {// a

**int** a;

}

**class** Boo **extends** Aoo {// a,b

**double** b;

}

## 6.3方法重写

子类中的方法名与父类中相同，调用时访问哪个看对象是哪个。

## 6.4抽象方法和抽象类

为什么需要抽象类：

抽象方法和抽象类看上去是多余的，对于抽象方法，不知道如何实现，定义一个空方法体不就行了吗，而抽象类不让创建对象，看上去只是增加了一个不必要的限制。

引入抽象方法和抽象类，是Java提供的一种语法工具，对于一些类和方法，引导使用者正确使用它们，减少被误用。

使用抽象方法，而非空方法体，子类就知道他必须要实现该方法，而不可能忽略。

使用抽象类，类的使用者创建对象的时候，就知道他必须要使用某个具体子类，而不可能误用不完整的父类。

无论是写程序，还是平时做任何别的事情的时候，每个人都可能会犯错，减少错误不能只依赖人的优秀素质，还需要一些机制，使得一个普通人都容易把事情做对，而难以把事情做错。抽象类就是Java提供的这样一种机制。

# 7. 修饰符

## 7.1 访问控制修饰符

（1）成员变量前的修饰符

**public** **class** Aoo {

**public** **int** a; //任何类

**protected** **int** b; //本类、派生类、同包类

**int** c; //本类、同包类

**private** **int** d; //本类

}

方法一般都是public

类前一般都是public，不推荐默认（本类，同包类）

## 7.2 final

（1）修饰变量：变量不能被改变

（2）修饰方法：方法不能被重写

（3）修饰类：类不能被继承

## 7.2 static

**（1）静态变量：**

1）由static修饰

2）属于类，储存在方法区中，只有一份

3）常常通过类名点来访问

4）所有对象共享数据时使用

**（2）静态方法**

static，类，方法区，一份，类名点来访问没有隐式this传递，不能直接访问实例成员方法的操作与对象无关

**（3）静态块**

1）由static修饰

2）归属于类，在类被加载期间自动执行，因类只被加载一次，所以静态块只被执行一次(在main方法开始前执行)

3）加载/初始化静态资源

**（4）static final常量: 应用率高**

1)必须声明同时初始化

2)通过类名点来访问，不能被改变

3)建议:常量名所有字母都大写，多个单词之间用下划线\_分隔

4)编译器在编译时会将常量直接替换为具体的值，效率高

5)何时用: 数据永远不变，并且经常使用

# 8.内部类

## 1.成员内部类:应用率低，了解即可

1)类中套类，外面的称为外部类，里面的称为内部类

2)内部类通常只服务于外部类，对外不具备可见性

3)内部类对象通常在外部类中创建

4)内部类中可以直接访问外部类的成员(包括私有的)

内部类中有个隐式的引用指向了创建它的外部类对象----外部类名.this

## 2.匿名内部类:--------------大大简化代码

1)若想创建一个类(派生类)的对象，并且对象只创建一次，

此时该类不必命名，称为匿名内部类

2)在匿名内部类中不能修改外面变量的值，因为在此处外面的变量会默认为final的

## 3做功能的套路:

1)先写行为/方法:

1.1)若行为为对象所特有的行为，则将行为设计在特定的类中

1.2)若行为为所有对象所共有的行为，则将行为设计在超类中

2)窗口调用:

2.1)若为定时发生的，就在定时器中调用

2.2)若为事件触发的，就在侦听器中调用