# 数组定义格式

## 1.1数组的定义格式

* 格式1： 数据类型[ ] 变量名
* 例如：int [] arr
* 定义了一个int类型的数组，数组名为arr
* 格式2：数据类型 变量名[ ]
* 例如：int arr [ ]
* 定义了一个int类型的变量，变量名为arr数组

# 数组初始化

## 2.1动态初始化：初始化时只指定数组长度，由系统为数组分配初始值

* 格式：数据类型 [ ] 变量名 = new 数据类型[数组长度];
* 例如：int [ ] arr = new arr[3];

int[ ] arr = new int[3];

/\*

\* 左边：

\* int:说明数组中的元素类型是int类型

\* []:数组

\* arr:数组名称

\* 右边：

\* new:为数组申请内存空间

\* int:说明数组中的元素类型是int型

\* 3:数组长度，其实就是数组中的元素个数

\* \*/

## 2.2静态初始化：初始化时指定每个数组元素的初始值，由系统决定数组长度

* 格式：数据类型 [ ] 变量名 = new 数据类型[ ]{数据1，数据2，数据3，……}
* 例如：int [] arr = new int []{1,2,3….}
* 简化格式：数据类型[] 变量名 = {数据1，数据2，数据3，……}
* 例如： int [] arr = {1,2,3,….}

## 2.3两个常见小问题

* 索引越界：访问了数组中不存在的索引对应的元素，造成索引越界
* 空指针异常：访问的数组已经不再指向堆内存的数据，造成空指针异常
* null：空值，引用数据类型的默认值，表示不指向任何有效对象

# 3.数组元素访问

## 3.1数组元素访问

* 数组变量访问方式：数组名
* 数组内部保存的数据的访问方式：数组名[索引]
* 索引是数组中数据的编号方式
* 作用：索引用于访问数组中的数据使用，数组名[索引]等于变量名，是一种特殊的变量名
* 特征：索引从0开始，索引是连续的，逐一增加，每次加1

# 4.内存分配

## 4.1Java中内存分配

* 栈内存：存储局部变量，定义在方法中的变量，例如：arr，使用完毕，立即消失
* 堆内存：存储new出来的内容（实体，对象）数组在初始化时，会为存储空间添加默认值。整数：0 浮点数：0.0 布尔：false 字符：空字符 引用数据类型：null。每一个new出来的东西都有一个地址值，使用完毕，会在垃圾回收空闲时被回收。

001 new int[3]

Int [ ] arr 001(数组名)

|  |  |
| --- | --- |
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |

堆内存

栈内存

# 5.数组常见操作

## 5.1遍历

## 5.2获取数组元素数量

* 格式：数组名.length
* 例如：arr.length

## 5.3获取最值

思路：

1：定义一个变量，用于保存最大值或最小值

2：取数组中第一个数据作为变量的初始值

3：与数组中剩余的数据逐个对比，每次比最大值保存到变量中

4：循环结束后打印变量值

# 6方法概述

## 6.1什么是方法

* 方法：是将具有独立功能的代码块组织成为一个整体，使其具有特殊功能的代码集
* 方法必须先创建才能使用，该过程称为方法定义
* 方法创建后并不是直接运行，需手动使用后执行，该过程称为方法调用

## 6.2方法的定义和调用

* 格式：public static void 方法名（）{

//方法体

}

* 范例：public static void isEvenNumber(){

//方法体

}

* 方法调用 格式：方法名（）； 例如：isEvenNumber();
* 方法必须先定义再调用

# 7带参数方法的定义和调用

## 7..1.带参数方法定义

* 格式： public static void 方法名(参数){。。。。。}
* 格式（单个参数） public static void 方法名(数据类型 变量名){……}
* 格式（多个参数） public static void 方法名(数据类型 变量名1，数据类型 变量名2，……){……}
* 注意：方法定义时，参数中的数据类型与变量名都不能缺少，缺少任意一个程序将会报错，多个参数之间使用逗号隔开。

## 7.2 带参数方法调用

* 格式：方法名(参数)；
* 格式（单个参数）方法名(变量名/常量值)；
* 格式（多个参数）方法名(变量名1/常量值1，变量名2/常量值2)；
* 注意：方法定义时，参数中的数量与类型必须与方法定义中的设置相匹配，否则程序将会报错

## 6.3.3 形参和实参

* 形参：方法定义中的参数，等同于变量定义格式，例如int number
* 实参：方法调用中的函数，等同于使用变量或常量，例如：isEven Number(10);

# 8带返回值方法的定义和调用

## 8.1.带返回值方法定义

* 格式： public static 数据类型 方法名(参数){

return 数据

}

* 范例：public static boolean isEvenNumber(int number){

return true;

}

* 注意：方法定义时，return后面的返回值与方法定义上的数据类型要匹配，否则程序将会报错

## 8.2.带返回值方法调用

* 格式1：方法名(参数)；
* 范例：isEvenNumber(5)；
* 格式2：数据类型 变量名 = 方法名(参数)；
* 范例：boolean flag = isEvenNumber(5)；
* 注意：方法的返回值通常会使用变量接收，否则该返回值将无意义

# 9.方法的注意事项

* 方法不能嵌套定义
* Void表示无返回值，可以省略return，也可单独书写return，后面不加数据。
* 方法通用格式：

public static 返回值类型 方法名（参数）{

方法体；

return 数据；

}

* 定义方法时，要做到两个明确：

明确返回函数值：主要是明确方法操作完毕之后是否有数据返回，如果没有，写void；如果有，写对应数据类型。

明确参数：主要是明确参数的类型和数量。

* 调用方法时

void类型的方法，直接调用即可

非void类型的方法，推荐用变量接收调用

# 10.方法重载

## 10.1方法重载概述：指同一个类中定义的多个方法之间的关系，满足下列条件的多个方法相互构成重载

* 多个方法在同一个类中
* 多个方法具有相同的方法名
* 多个方法的参数不相同，类型不同或者数量不同

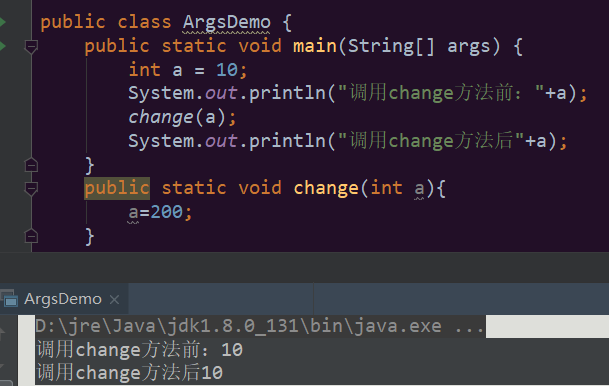
## 10.2方法重载特点

* 重载仅对应方法的定义，与方法的调用无关，调用方式参照标准格式
* 重载仅针对同一个类中方法的名称与参数进行识别，与返回值无关，换句话说不能通过返回值来判断两个方法是否相互构成重载。

# 11.方法的参数传递

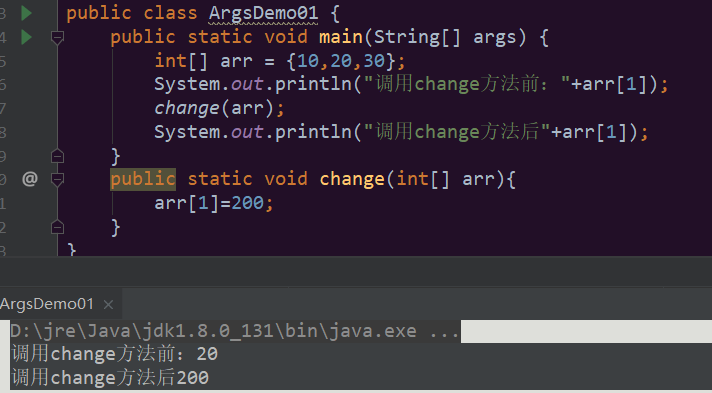
## 11.1方法参数传递（基本类型）

* 对于基本数据类型的参数，形式参数的改变，不影响实际参数的值。



## 11.2方法参数传递（引用类型）

* 对于引用数据类型的参数，形式参数的改变，影响实际参数的值



# 12.Debug

## 12.1Debug操作流程

* Debug：是供程序员使用的程序调试工具，它可以用于查看程序的执行流程，也可用于追踪程序执行过程来调试程序。

Debug调试：又称断点调试，断点其实是一个标记，告诉我们从哪里开始查看。

1. 如何加断点：选择要设置断点的代码行，在行号的区域的后面单击鼠标左键。
2. 如何运行加了断点的程序：在代码区域右键Debug执行。
3. 看哪里：看Debugger窗口和Console窗口。
4. 点哪里：点step into(F7)这个箭头。点stop结束。
5. 如何删除断点：选择要删除的断点，单击鼠标左键即可。

* 注意事项：如果数据来自键盘输入，一定要记得输入数据，不然就不能继续往下查看了。

类和对象

# 类和对象

## 1.1什么是对象

万物皆对象，客观存在的事务皆为对象。

## 1.2什么是类

* 类是对现实生活中一类具有共同属性和行为的事务的抽象。
* 类的特点：

1. 类是对象的数据类型
2. 类是具有相同属性和行为的一组对象的集合

## 1.3什么是对象的属性

* 属性：对象具有的各种特征，每个对象的每个属性都有特定的值。

## 1.4什么是对象的行为

* 行为：对象能够执行的行为。

## 1.5类和对象的关系

* 类是对现实生活中一类具有共同属性和行为的事务的抽象。
* 对象是能够看得到摸得着的真是存在的实体
* 类是对象的抽象。对象是类的实体。

## 1.6类的定义

* 类的重要性：是Java程序的基本组成单位。
* 类的组成：属性和行为

属性：在类中通过成员变量来体现（类中方法外的变量）

行为：在类中通过成员方法来体现（和前面的方法相比去掉static关键字即可）

* 类的定义步骤：

1. 定义类
2. 编写类的成员变量
3. 编写类的成员方法
4. public class 类名{

//成员变量

变量的数据类型 变量；

……

//成员方法

方法1；

方法2；

}

## 1.7对象的使用

* 创建对象：类名 对象名 = new 类名();
* 使用对象：

1. 使用成员变量：对象名.变量名 例如：p.brand
2. 使用成员方法：对象模.方法名 例如：p.call()

# 2.成员变量和局部变量

|  |  |  |
| --- | --- | --- |
| 区别 | 成员变量 | 局部变量 |
| 类中位置不同 | 类中方法外 | 方法内或方法声明上 |
| 内存中的位置不同 | 堆内存 | 栈内存 |
| 生命周期不同 | 随着对象的存在而存在，随着对象的消失而消失 | 随着方法的调用而存在，随着方法的调用完毕而消失 |
| 初始化值不同 | 有默认初始值 | 没有默认初始值，必须先定义，赋值才能使用 |

# 封装

## 3.1 private 关键字

* 是一个权限修饰符，可以修饰成员（成员变量和成员方法）
* 作用是保护成员不被别的类使用，被private修饰的成员只在本类中才能访问
* 针对private修饰的成员变量，如果需要被其它类使用，提供相应的操作
* 提供“get变量名（）“方法，用于获取成员变量的值，方法用public修饰
* 提供“set变量名（）“方法，用于设置成员变量的值，方法用public修饰

## 3.2 private 关键字的使用

一个标准类的编写：

* 把成员变量用private修饰
* 提供对应的getXxxx()/setXxx()方法

## 3.3 this关键字的使用

1. this修饰的变量用于指代成员变量

* 方法的形参如果与成员变量同名，不带this修饰的变量指的是形参，而不是成员变量
* 方法的形参如果没有与成员变量同名，不带this修饰的变量指的是成员变量

1. 什么时候用this？解决局部变量隐藏成员变量时
2. this：代表所在类的对象引用

* 方法被那个对象调用，this就代表哪个对象

## 3.4 封装

1. 封装概述

* 是面向对象三大特征之一（封装，继承，多态）
* 是面向对象编程语言对客观世界的模拟，客观世界里成员变量都是隐藏在对象内部的，外界是无法直接操作的。

2.封装原则

* 将类的某些信息隐藏在类内部，不允许外部程序直接访问，而是通过该类提供的方法来实现对隐藏信息的操作和访问成员变量private,提供对应的getXxx()/setXxx()方法

3.封装好处

* 通过方法来控制成员变量的操作，提高了代码的安全性。把代码用方法进行封装，提高了代码的复用性。

# 构造方法

## 4.1 构造方法描述

* 构造方法是一种特殊的方法，作用：创建对象
* 格式：public class 类名{

修饰符 类名（参数）{

}

}

* 功能：主要是完成对象的数据初始化

## 4.2 构造方法的注意事项

1. 构造方法的创建

* 如果没有定义构造方法，系统将给出一个默认的无参构造方法
* 如果定义了构造方法，系统将不再提供默认的构造方法

1. 构造方法的重载

* 如果自定义了带参构造方法，仍使用无参构造方法，就必须再写一个无参构造方法

3. 推荐方式：无论是否使用，都手工书写无参构造方法

## 4.3 标准类制作

1. 成员变量用private修饰

2. 构造方法：

* 提供一个无参构造方法
* 提供一个带有多个参数的构造方法

3.﷒成员方法

* 提供每一个成员变量对应的setXxx()/getXxx()
* 提供一个显示对象show()

4. 创建对象并为其成员变量赋值的两种方式

* 无参构造方法创建对象后使用setXxx()赋值
* 使用带参构造方法直接创建带有属性值的对象

# API

## 5.1 API（应用程序编程接口）概述

# 6. String

* String类在java.lang包下，所以使用的时候不需要导包
* String类代表字符串，Java程序中的所有字符串文字都被实现为此类的实例，也就是说。Java程序中所有的双引号字符串，都是String类的对象
* 字符串的特点

1. 字符串不可变，值在创建后不能被更改
2. 虽然String的值不可改变，但可以被共享
3. 字符串效果上相当于字符数组(char[])，但底层原理是字节数组（byte[]）

## 6.1 String对象的特点

1. 通过new创建的字符对象，每一次new都会申请一个内存空间，虽然内容相同，但地址值不同

2. 以“ ”方式给出的字符串，只要字符序列相同（顺序和大小写），无论在程序代码中出现几次，JVM都只会创建一个String对象，并在字符串池中维护。

## 6.2字符串的比较

使用 == 作比较

* 基本类型：比较的是数据值是否相同
* 引用类型：比较的是地址值是否相同

字符串是对象，它比较内容是否相同的，是通过一个方法来实现的，这个方法叫：equals（）

* public Boolean equals(Object anObject):将此字符串与指定对象进行比较。由于我们比较的是字符串对象，所以参数直接传递一个字符串。

## 6.3通过帮助文档查看String中的方法

|  |  |
| --- | --- |
| 方法名 | 说明 |
| public Boolean equals(Object anObject) | 比较字符串的内容，严格区分大小写（用户名，密码） |
| public char charAt(int index) | 返回指定索引处的char值 |
| public int length() | 返回此字符串长度 |

# 7. StringBuilder

## 7.1StringBuilder概述

* StringBuilder是一个可变的字符串类，我们可以把它看成是一个容器，这里的可变指的是StringBuilder对象中的内容是可变的。String内容是不可变的。

## 7.2StringBuilder构造方法

|  |  |
| --- | --- |
| 方法名 | 说明 |
| public StringBuilder() | 创建一个空白可变的字符串对象，不含有任何内容 |
| public StringBuilder(String str) | 根据字符串的内容，来创建可变字符串对象 |

## 7.3StringBuilder的添加和反转方法

|  |  |
| --- | --- |
| 方法名 | 说明 |
| public StringBuilder append(任意类型) | 添加数据，并返回对象本身 |
| public StringBuilder reverse() | 返回相反的字符序列 |
| Public String toString() | 通过toString()就可以实现把StringBuilder转换为String |

## 7.4StringBuilder和String的相互转换

1. StringBuilder转换为String:

public String toString():通过toString()就可以实现把StringBuilder转换为String

1. String转换为StringBuilder

public StringBuilder(String s):通过构造方法就可以实现把String转换为StringBuilder

# 8. 集合基础

## 8.1集合概述

集合类的特点：提供一种存储空间可变的存储模型，存储的数据容量可以发生改变

集合类有很多，目前先学习:ArrayList

ArrayList<E>:

* 可调整大小的数组实现
* <E>:是一种特殊的数据类型，泛型

怎么用？

在出现E的地方我们使用引用数据类型替换即可

举例：ArrayList<String>, ArrayList<Student>

## 8.2ArrayList构造方法和添加方法

|  |  |
| --- | --- |
| 方法名 | 说明 |
| public ArrayList() | 创建一个空的集合对象 |
| public Boolean add(E e) | 将指定的元素追加到此集合的结尾 |
| public void add(int index,E element) | 在此集合的指定位置插入指定的元素 |

## 8.3ArrayList集合常用方法

|  |  |
| --- | --- |
| 方法名 | 说明 |
| public Boolean remove(Object o) | 删除指定的元素，返回删除是否成功 |
| public E remove(int index) | 删除指定索引处的元素，返回被删除的元素 |
| public E set(int index,E element) | 修改指定索引处的元素，返回被修改的元素 |
| public E get(int index) | 返回指定索引处元素 |
| public int size() | 返回集合中元素的个数 |

# 9. 继承

## 9.1继承概述

继承是面向对象三大特征之一。可以是的子类具有父类的属性和方法，还可以在子类中重新定义，追加属性和方法。

继承格式：

* public class 子类名 extends 父类名{}
* 范例：public class ZI extends FU{}
* FU：是父类，也被称为基类，超类
* ZI：是子类，也被称为派生类

继承中子类的特点：

* 子类可以有父类的内容和子类特有的内容

## 9.2继承的好处和弊端

继承好处

* 提高了代码的复用性（多个类相同的成员可以放到同一个类中）
* 提高了代码的维护性（如果方法需要修改，修改一处即可）

继承弊端

* 继承让类与类之间产生了关系，类的耦合性增强了，当父类发生变化时，子类实现也不得不跟着变化，削弱了子类的独立性。

什么时候使用继承？

* 继承体现的关系：is a(什么是什么的一种)
* 假设法：有A和B两个类，如果满足A是B的一种，或者B是A的一种，就说明她们存在继承关系，这个时候就可以考虑使用继承来体现，否则就不能滥用继承

## 9.2继承中变量的访问特点

在子类中访问一个变量

* 子类局部范围找
* 子类成员范围找
* 父类成员范围找
* 如果没有就报错

## 9.3 super

super关键字的用法和this关键字的用法相似

* this：代表本类对象的引用
* super：代表父类存储空间的标识（可以理解为父类对象引用）

|  |  |  |  |
| --- | --- | --- | --- |
| 关键字 | 访问成员变量 | 访问构造方法 | 访问成员方法 |
| this | this.成员变量 | this(……)访问本类构造方法 | this.成员方法（…）  访问本类常用方法 |
| super | super.成员变量  访问父类成员变量 | super（…）  访问父类构造方法 | super.成员方法（…）  访问父类成员方法 |

## 9.4继承中构造方法的访问特点

子类中所有的构造方法默认都会访问父类中的无参构造方法。

为什么？

* 因为子类会继承父类中的数据，可能还会使用父类的数据。所以，子类初始化之前，一定要先完成父类数据的初始化。
* 每一子类构造方法的第一条语句默认都是：super()

如果父类中没有无参构造方法，只有带参构造方法，该怎么办呢？

* 通过使用super关键字去显示调用父类的带参构造方法
* 在父类中自己提供一个无参构造方法（推荐此方法）

## 9.5继承中成员方法的访问特点

通过子类对象访问一个方法

* 子类成员范围找
* 父类成员范围找
* 如果没有就报错

## 9.6方法重写

方法重写概述

* 子类中出现和父类一摸一样的方法声明

方法重写的应用

* 当子类需要父类的功能，而功能主题子类有自己的特定内容时，可以重写父类中的方法，这样，即沿袭了父类的功能，又定义了子类特有的内容

@override

* 是一个注解
* 可以帮助我们检查重写方法的方法声明的正确性

## 9.7方法重写注意事项

* 私有方法不能被重写（父类私有成员，子类不能继承）
* 子类方法访问权限不能更低（public>默认>私有）

## 9.8Java中继承的注意事项

* Java中类只支持单继承，不支持多继承（如：public class Son extends Father,Mother{}这是错误的）
* Java中类支持多层继承

修饰符

# 包

## 1.1包的概述和使用

其实就是文件夹。作用是对类进行分类管理

报的定义格式：package 包名； //多级包用.分开

带包的Java类编译和执行

* 手动建包：

1.按照以前的格式编译Java文件 javac HelloWorld.java

2.手动创建包 在E盘建立文件夹com,然后com下建文件夹study

3.带包执行 java com.study.HelloWorld

* 自动建包：javac -d . HelloWorld java com.study.HelloWorld

## 1.2导包

格式：import 包名；

# 修饰符

## 2.1修饰符的分类

* 权限修饰符
* 状态修饰符

## 2.2 权限修饰符

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 修饰符 | 同一个类中 | 同一个包中子类/无关类 | 不同包中的子类 | 不同包的无关类 |
| private | √ |  |  |  |
| 默认 | √ | √ |  |  |
| protected | √ | √ | √ |  |
| public | √ | √ | √ | √ |

## 2.3 状态修饰符

* final(最终态)
* static(静态)

## final

final关键字时最终的意思，可以修饰成员方法，成员变量，类

final修饰的特点

* 修饰方法：表明该方法是最终方法，不能被重写
* 修饰变量：表明该变量是常量，不能被再赋值
* 修饰类：表明该类是最终类，不能被继承

## final修饰局部变量

* 变量是基本类型：final修饰指的是基本类型的数据值不能发生改变
* 变量是引用类型：final修饰指的是引用类型的地址值不能发生改变，但地址里面的内容是可以改变的

## static

static关键字是静态的意思，可以修饰成员方法，成员变量

static修饰的特点

* 被类的所有对象共享

这也是我们判断是否使用静态关键字的条件

* 可以通过类名调用，也可以通过对象名调用（推荐使用类名调用）

## static访问特点

非静态的成员方法

* 能访问静态的成员变量
* 能访问非静态的成员变量
* 能访问静态的成员方法
* 能访问非静态的成员方法

静态的成员方法

* 能访问静态的成员变量
* 能访问静态的成员方法

总结：静态成员的只能访问静态成员

多态

# 1.多态

## 1.1多态概述

同一对象，在不同时刻表现出来的不同形态

多态的前提和引用

* 有继承/实现关系
* 有方法重写
* 有父类引用指向子类对象

## 1.2多态中成员访问变量特点

* 成员变量：编译看左边，执行看左边
* 成员方法：编译看左边，执行看右边

为什么成员变量和成员方法的访问不一样？

* 因为成员方法有重写，而成员变量没有

## 1.3多态的好处和弊端

* 好处：提高了程序的扩展性

具体表现：定义方法的时候，使用父类型作为参数，将来在使用的时候，使用具体的子类型参与操作

* 弊端：不能使用子类特有的功能

## 1.4多态中的转型

* 向上转型

从子到父

父类引用指向子类对象

* 向下转型

从父到子

父类引用转为子类对象

抽象类

# 1.抽象类

## 1.1抽象类概述

在Java中，一个没有方法体的方法应该定义为抽象方法，而类中如果有抽象方法，该类必须定义为抽象类

## 1.2抽象类的特点

* 抽象类和抽象方法必须使用abstract关键字来修饰

public abstract class 类名{}

public abstract void eat（）；

* 抽象类中不一定有抽象方法，有抽象方法的类一定是抽象类
* 抽象类不能实例化，参照多态的方式，通过对子类对象实例化，这叫抽象类多态
* 抽象类的子类

要么重写抽象类中的所有抽象方法

要么是抽象类

## 1.3抽象类的成员特点

* 成员变量：可以是变量，也可以是常量
* 有构造方法，但不能实例化。构造方法的作用是用于子类访问父类数据的初始化
* 成员方法“

可以有抽象方法：限定子类必须完成某些动作

也可以有非抽象方法：提高代码复用性

## 1.4抽象类名作为形参和返回值

* 方法的形参是抽象类名，其实需要的是该抽象类的子类对象
* 方法的返回值是抽象类名，其实返回的是该抽象类的子类对象

接口

# 1.接口

## 1.1接口描述

接口就是一种公共的规范标准，只要符合标准规范，大家都可以通用

Java中的接口更多的体现在对行为的抽象

## 1.2接口特点

* 接口用关键字interface修饰

public interface 接口名{}

* 类实现接口用implements表示

public class 类名 implements 接口名{}

* 接口不能实例化，参照多态的方式，通过实现类对象实例化，这叫接口多态

多态的形式：具体类多态，抽象类多态，接口多态

多态前提：有继承或实现关系；有方法重写；有父（类/接口）引用指向（子/实现）对象

* 接口的实现：要么重写接口中的所有抽象方法，要么是抽象类

## 1.3接口特点

* 成员变量只能是常量，默认修饰符：public static final
* 构造方法

接口没有构造方法，因为接口主要是对行为进行抽象的，是没有具体存在

一个类如果没有父类，默认继承Object类

* 成员方法只能是抽象方法，默认修饰符：public abstract

## 1.4接口和类的关系

* 类和类的关系

继承关系，只能单继承，但可以多层继承

* 类和接口的关系

实现关系，可以单实现，也可以多实现，还可以在继承一个类的同时实现多接口

* 接口和接口的关系

继承关系，可单继承，也可以多继承

## 1.5抽象类和接口的区别

* 成员区别

抽象类：变量，常量，有构造方法，有抽象方法，也有非抽象方法

接口： 常量，抽象方法

* 关系区别

类与类：继承，单继承

类与接口：实现，可以单实现，也可以多实现

接口与接口：继承，单继承，多继承

* 设计理念区别

抽象类：对类抽象，包括属性、行为

接口： 对行为抽象，主要对行为

# 1.形参和返回值

## 1.1接口名作为形参和返回值

* 方法的形参是接口名，其实需要的是该接口实现类对象
* 方法的返回值是接口名，其返回的是该接口的实现类对象

内部类

# 1.内部类

## 1.1内部类概述

内部类：就是在一个类中定义一个类。举例：在一个类A的内部定义一个类B，类B就被称为内部类

内部类的定义格式

* 格式：

public class 类名{

修饰符 class 类名{

}

}

内部类的访问特点

* 内部类可以直接访问外部类的成员，包括私有
* 外部类要访问内部类的成员，必须创建对象

## 1.2成员内部类

按照内部类在定义的位置不同，可以分为如下两种形式

* 在类的成员位置：成员内部类
* 在类的局部位置：局部内部成员

## 1.3局部内部类

局部内部类是方法中定义的类，所以外界是无法直接使用，需要在方法内部创建对象并使用

该类可以直接访问外部类的成员，也可以访问方法内的局部变量

## 1.4匿名内部类

前提：存在一个类或者接口，这里的类可以是具体类也可以是抽象类

* 格式：

new 类名或者接口名（）{

重写方法；

};

* 范例：new Inter(){

public void show(){}

};

## 1.5匿名内部类在开发中的使用

常用API

# 1.Math

## 1.1Math概述

Math包括执行基本数字运算的方法

没有构造方法，如何使用类中的成员呢？

看类的成员是否是静态的，如果是，通过类名就可以直接调用

## 1.2Math类的常用方法

|  |  |  |
| --- | --- | --- |
| 方法名 | | 说明 |
| public static int abs(int a) | 返回参数的绝对值 | |
| public static double ceil(double a) | 返回大于或等于参数的最小double值，整数 | |
| public static double floor(double a) | 返回小于或等于参数的最大double值，整数 | |
| public static int round(float a) | 按照四舍五入返回最接近参数的int | |
| public static int max(int a,int b) | 返回两个int值中的较大值 | |
| public static int min(int a,int b) | 返回两个int值中的较小值 | |
| public static double pow(double a,double b) | 返回a的b次幂的值 | |
| public static double random() | 返回值为double的正值，[0.0,1.0) | |

# 2. System

## 2.1 System类概述

System包含几个有用的类字段和方法，不能被是实例化

## System类的常用方法

|  |  |
| --- | --- |
| 方法名 | 说明 |
| public static void exit(int status) | 终止当前运行的Java虚拟机，非零表示异常终止 |
| public static long currentTimeMillia() | 返回当前时间（以毫秒为单位） |

# 3. Object

## 3.1 Object类概述

Object是类层次结构的根，每个类都可以将Object作为超类。所有类都直接或者间接继承自该类。构造方法：public Object()

回想面向对象中，为什么说子类的构造方法默认访问的是父类的无参构造方法？因为他们的顶级父类只有无参构造方法。

## 3.2 Object类的常用方法

|  |  |
| --- | --- |
| 方法名 | 说明 |
| public String toString() | 返回对象的字符串表现形式。建议所有子类重写该方法，自动生成 |
| public Boolean equals(Object o) | 比较对象是否相等。默认比较地址，重写可以比较内容，自动生成44 |

# 4. Arrays

## 4.1 冒泡排序

排序：将一组数据按照固定的规则进行排序

冒泡排序：对要进行排序的数据中相邻的数据进行两两比较，将较大的放在后面，一次对所有的数据进行操作，直至所有数据按要求完成排序

* 如果有n个数据进行排序，总共需要比较n-1次
* 每一次比较完毕，下一次的就会少一个数据参与

## 4.2 Arrays类的概述和常用方法

排序：将一组数据按照固定的规则进行排序

Arrays类包含用于操作数组的各种方法

|  |  |
| --- | --- |
| 方法名 | 说明 |
| public static String toString(int[] a) | 返回指定数组的内容的字符串表示形式 |
| public static void sort(int[] a) | 按照数字顺序排列指定的数组 |

工具类的设计思想：

* 构造方法用private修饰
* 成员用public static 修饰，方便通过类名访问

# 5.基本类型包装类

## 5.1 基本类型包装类概述

将基本数据类型封装成对象的好处在于可以在对象中定义更多的功能方法操作该数据

常用的操作之一：用于基本数据类型与字符间的转换

## 5.2 Integer类的概述和使用

Interger:包装一个对象中的原始类型int的值

|  |  |
| --- | --- |
| 方法名 | 说明 |
| public Integer(int value) | 根据int值创建Integer对象（已过时） |
| public Interger(String s) | 根据String值创建Integer对象（已过时） |
| public static Integer valueOf(int i) | 返回表示指定int值的Integer实例 |
| public static Integer valueOf(String s) | 返回一个保存指定值的Integer对象String |

## 5.3 int和String的相互转换

基本类型包装类的常见操作就是：用于基本类型和字符串之间的相互转换

1. int转String: public static String valueOf(int i):返回int参数的字符串表示形式。该方法是String类中的方法。
2. String转int： public static int parseInt(String s):将字符串解析为int类型。该方法是Integer类中的方法。

## 5.4 自动装箱和拆箱

* 装箱：把基本数据类型转换为对应的包装类类型
* 拆箱：把包装类类型转换为对应的基本数据类型
* 注意：在使用包装类类型的时候，如果做操作，最好先判断是否为null，推荐：只要是对象，在使用前必须进行不为null的判断。

# 6.日期类

## 6.1 Date类概述和构造方法

Date代表了一个特定的时间，精确到毫秒

|  |  |
| --- | --- |
| 方法名 | 说明 |
| public Date() | 分配一个Date对象，并初始化，以便它代表它被分配的时间，精确到毫秒 |
| public Date(long date) | 分配一个Date对象，并将其初始化为表示从标准基准时间起指定的毫秒数 |

## 6.2 Date类的常用方法

|  |  |
| --- | --- |
| 方法名 | 说明 |
| public long getTime() | 获取的是日期对象从1970年1月1日00：00：00到现在的毫秒值 |
| public void setTime(long time) | 设置时间，给的是毫秒值 |

## 6.3 SimpleDateFormat类概述

SimpleDateFormat是一个具体类，用于以区域设置敏感的方式格式化和解析日期。重点学习日期格式化和解析。

日期和时间格式由日期和时间模式字符串指定，在日期和时间模式字符串中，从‘A’到‘Z’以及从‘a’到’z‘引导的字母被解释为表示日期或时间字符串的组件的模式字母

常用的模式字母以及对应关系如下：

* y 年
* M 月
* d 日
* H 时
* m 分
* s 秒

## 6.4 SimpleDateFormat的构造方法

|  |  |
| --- | --- |
| 方法名 | 说明 |
| public SimpleDateFormat() | 构造一个SimpleDateFormat,使用默认模式和日期格式 |
| public SimpleDateFormat(String pattern) | 构造一个SimpleDateFormat,使用给定的模式和默认日期格式 |

## 6.5SimpleDateFormat格式化和解析日期

1.格式化（从Date到String）

public final String format(Date date):将日期格式化成日期/时间字符串

2.解析（从String和Date）

public Date parse(String source):从给定字符串的开始解析文本以生成日期

## 6.6Calendar类概述

Calendar为某一时刻和一组日历字段之间的转换提供了一些方法，并为操作日历字段提供了、一些方法。

Calendar提供了一个类方法getlnstance用于获取Calendar对象，其日历已使用当前日期和时间初始化：Calendar rightnow =Calendar.getInstance();

## 6.7 Calendar的常用方法

|  |  |
| --- | --- |
| 方法名 | 说明 |
| public int get(int filed) | 返回给定字段的值 |
| public abstract void add(int field,int amount) | 根据日历的规则，将指定的时间量添加或减去给定的日历字段 |
| public final void set(int year,int month,int date) | 设置当前日历的年月日 |

异常

# 1.异常

## 1.1异常概述

异常：就是程序出现了不正常的情况

**异常体系**

Throwable : Error和Exception , Exception包括RuntimeException和非RuntimeException

Error:严重问题，不需要处理

Exception:称为异常类，它表示程序本身可以处理的问题

* RuntimeException ：在编译期是不检查的，出现问题后，需要我们回来修改代码
* 非RuntimeException：编译期就必须处理的，否则程序不能通过编译，就更不能正常运行了

## 1.2JVM的默认处理方案

如果程序出现了问题，我们没有做任何处理，最终JVM会做默认的处理

* 把异常的名称，异常原因及异常出现的位置等信息输出在控制台
* 把程序停止执行

## 1.3 异常处理

如果程序出现了问题，我们需要自己来处理，有两种方案：

* try…catch…
* throws

## 1.4异常处理之try…catch…

* 格式：

try{

可能出现异常的代码；

}catch(异常类名 变量名){

异常的处理代码；

}

执行流程：程序从try里面的代码开始执行，出现异常，会自动生成一个异常类对象，该异常对象将被提交给Java运行时系统。当Java运行时系统接收到异常对象时，会到catch中去找匹配的异常类，找到后进行异常的处理。执行完毕之后，程序还可以继续执行下去。

## 编译时异常和运行时异常的区别

Java中的异常被分为两大类：编译时异常和运行时异常，也被称为受检异常和非受检异常

所有的RuntimeException类及其子类被运行时异常，其他的异常都是编译时异常

* 编译时异常：必须显示处理，否则程序就会发生错误，无法通过编译
* 运行时异常：无需显示处理，也可以和编译时异常一样处理

## 异常处理之throws

格式：throws 异常类名；

注意：这个格式是跟在方法的括号后面

* 编译时异常必须要进行处理，两种处理方案：try…catch…或者throws。如果采用throws这种方案，将来谁调用谁负责
* 运行时异常可以不处理，出现问题后，需要回来修改代码