# Java类和类的成员

## 属性

* + 1. 类的属性的赋值

按照顺序：

默认初始化

显式初始化/代码块中初始化 [这两个看声明的先后顺序]

构造器中初始化

“对象·方法”或“对象·属性”

## 方法

* + 1. 方法的重写

override或者overwrite

子类中对继承自父类的方法的重写

子类中的叫做重写的方法，父类中的叫做被重写的方法

重写的规定：

1. 子类重写的方法和父类被重写的方法的方法名和形参列表相同
2. 子类重写的方法的权限修饰符要不小于被重写的方法的权限修饰符
3. 子类不能重写父类中声明为private的方法
4. 返回值类型

如果父类被重写的方法返回值类型为void，重写返回也只能是void

如果被重写方法返回类型为x，重写的返回类型可以是x类和x的子类

如果被重写方法返回类型是基本类型，重写的返回类型必须一致

1. 子类重写的方法抛出的异常类型不大于父类被重写的方法的异常类型
2. 子类和父类中同名同参数的方法，要么都声明为非static的[考虑重写]，要么就声明为static的

## 构造器

* + 1. 构造器的作用

1. 创建对象
2. 初始化对象的信息
   * 1. 构造器的说明
3. 只要创建对象，就要用构造器
4. 一个类一定会有构造器

## 代码块

Block，又称初始化代码块。

代码块的作用是，用来初始化类，对象。

代码块如果有修饰的话，只能是static。

分为静态代码块和非静态代码块。

* + 1. 静态代码块

// static的代码块  
static {  
 System.*out*.println("hello static block.");  
}

静态代码块内部可以由输出语句。

随着类的加载而执行，而且只执行一次，就是在类加载的时候。

静态代码块的执行优先于非静态代码块的执行，和位置无关。

静态代码块之间按照位置的先后顺序执行。

静态代码块中可以调用静态的属性和方法，但是不可以调用非静态的。

* + 1. 非静态代码块

// 非static的代码块  
{  
 System.*out*.println("hello not-static block.");  
}

非静态代码块内部可以输出语句。

随着对象的创建而执行，每创建一个对象，就执行一次非静态代码块。

作用是在创建对象时，对对象的属性等进行初始化。

非静态代码块之间按照位置的先后顺序执行。

非静态代码块中可以调用静态和非静态的属性和方法。

* + 1. 执行顺序的总结

“由父及子，静态先行”

## 内部类

Java中允许将一个类A声明在一个类B中，此时，A称为内部类，B称为外部类。

* + 1. 内部类的分类：

1. 成员内部类
2. 局部内部类 [方法内，代码块内，构造器内]
   * 1. 成员内部类
3. 作为一个类

可以订立属性，方法，构造器，代码块，内部类等。

可以被final修饰，表示不可继承；否则可以继承。

可以被abstract修饰，表示不可实例化；否则可以实例化。

1. 作为外部类的成员

可以调用外部类的结构。

可以用static修饰。

可以被四种不同的权限修饰。

* + 1. 关注的问题

1. 如何实例化成员内部类的对象
2. 如何在成员内部类中区分调用外部类的结构
3. 在开发中，局部内部类的使用

# Java面向对象的三大特性

## 封装性

高内聚，低耦合的原则

隐藏对象内部的复杂性，只向外展示简单的接口

* + 1. 四种权限

**四种权限可以等价为封装性**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 类内 | 包内 | 不同包的子类 | 同一个工程 |
| private | 是 |  |  |  |
| 缺省 | 是 | 是 |  |  |
| protected | 是 | 是 | 是 |  |
| public | 是 | 是 | 是 | 是 |

* + 1. 封装性的体现

A．将类的属性私有化，提供get和set方法

B．隐藏一些私有的方法

C．单例模式，将构造器私有化

D．不希望类在package外被调用

## 继承性

* + 1. 继承性的好处

1. 减少代码冗余，提高了代码复用性
2. 便于功能的扩展
3. 为之后的多态性的使用提供了前提
   * 1. 继承性的格式

class A extends B { }

A 子类，派生类，subclass

B 父类，超类，superclass

一旦子类A继承了父类B，子类A就获取了B中声明的所有结构[属性和方法]，父类中声明的private的也可以继承，但是不能直接调用[封装性的影响]。

子类继承父类之后，还可以声明自己特有的属性和方法。

子类和父类的关系，不能理解为子集和集合的关系，extends是延展和扩展。

* + 1. 关于继承性的规定

一个父类可以有多个子类

一个子类只能有一个父类，即Java中类的单继承性

类可以多层继承

子类直接继承的类称为直接父类，间接继承的类称为间接父类

子类继承父类之后，就获取了直接父类和所有间接父类的属性和方法

如果没有显式地声明一个类的父类，那么这个类节继承于java.lang.Object，所有的java类，都直接或间接地继承于java.lang.Object，都具有Object中声明的功能。

## 多态性

* + 1. 理解多态性

可以理解为一个事物的多种形态

* + 1. 何为多态性

父类的引用指向子类的对象，或称为子类的对象赋给父类引用。

Person p = new Man();

* + 1. 多态性的使用

“虚拟方法调用”

有了对象的多态性以后，我们在编译期，只能调用父类中声明的方法，但是在执行期，执行的是子类重写的方法。编译看左边，执行看右边。

1. 当对象调用子类和父类同名同参数的方法时，执行的是子类重写的方法。
2. 但是此对象不能调用子类所特有的方法。
   * 1. 多态性使用的前提
3. 要有类的继承关系
4. 要有方法的重写
5. 对象的多态性只适用于方法，不适用于属性，属性的编译和执行都看左边
   * 1. 虚拟方法调用的再理解

子类中定义了和父类中同名同参数的方法，在多态情况下，将此时父类的方法称为“虚拟方法”或“虚方法”，父类根据赋给它的不同子类对象，动态地调用属于子类的该方法。这样的方法调用在编译时时无法确定的。

**多态是一个运行时行为**。只用到运行时才能确定。

虚拟方法运行的过程可称为“动态绑定”或“晚绑定”。

* + 1. 向下转型

有了多态性以后，内存中实际上是已经加载了子类特有的属性和方法的，但是由于变量声明为父类类型，导致在编译时，只能调用父类中声明的属性和方法，子类特有的属性和方法无法调用。

如何调用子类中特有的属性和方法？

使用强制类型转换符，如，Man m = (Man) p;

这种方法称为“向下转型”。

“向上转型”就是多态。

使用强转时，可能出现ClassCastException异常，转型失败。

为了不在强转时出现异常，通常使用instanceof进行判断。

一些例子：

1. 编译通过，运行不过

Person p = new Woman();

Man m = (Man) p;

或者

Person p = new Person();

Man m = (Man) p;

1. 编译通过，运行也通过

Object obj = new Woman();

Person p = (Person) obj;

1. 正确用法示例

Person p = new Man;

Man m = (Man) p;

## 抽象性——第四大特性

也可以视为abstract关键字的使用。

abstract可以用来修饰类，方法。

* + 1. abstract修饰类

abstract修饰的类不可实例化，即不可造对象了，这样的类被称为抽象类。

抽象类中也有构造器，便于子类实例化时调用。

开发中，都会提供抽象类的子类，让子类对象实例化，完成相关的实例化。

* + 1. abstract修饰方法

abstract修饰的方法称为抽象方法，只有方法的声明，没有方法体。表现为没有大括号，直接以分号结束。

包含抽象方法的类，一定是抽象类。只有抽象类，才能包含抽象方法。

抽象类中可以没有抽象方法。

若子类重写了父类中的抽象方法后，此子类方可实例化。若子类没有重写父类中的所有抽象方法，则此子类也是一个抽象类。

* + 1. abstract的使用注意点

abstract不能修饰属性，构造器。

abstract不能修饰私有方法，静态方法，final修饰的方法。

abstract不能修饰final修饰的类。

* + 1. 抽象类的匿名子类

创建匿名子类时会重写方法。

# Java关键字

## this

* + 1. 可以调用的机构

属性和方法

构造器

* + 1. this调用属性和方法

此时this可以理解为当前正在创建的对象或方法

* + 1. this调用构造器

显式地使用“this(形参列表)”的方式，调用本类中其他的构造器

## import

在源文件中显式地导入指定包下的类和接口

声明在包和类之间

如果需要导入多个结构，都写出即可

可以使用XXX. \*的方式导入XXX包下的所有结构

Java.lang包下和本包下定义的类和接口可以不用import

## super

子类重写的方法覆盖了父类中被重写的方法

想调用父类中被重写的方法

super理解为父类的…

super可以调用属性，方法，构造器

* + 1. super调用属性和方法

可以在子类的方法和构造器中，使用“super·属性”或“super·方法”的方式，显式地调用父类中声明的属性和方法。但是习惯省略“super”。

特殊情况[属性或方法同名]：

当子类和父类中定义了同名的属性时，要想在子类中调用父类中的属性，必须显式地使用“super·属性”的方式调用父类中的属性。

当子类重写了父类的方法以后，要想在子类中调用父类中被重写的方法，必须显式地使用“super·方法”的方式。

* + 1. super调用构造器

可以在子类的构造器中显式地使用“super（形参列表）”的方式调用父类中指定的构造器，且必须声明在子类构造器中的首行。

我们在类的构造器中，“super（形参列表）”和“this（形参列表）”只能二选一。当在构造器的首行既没有显式调用“super（形参列表）”或“this（形参列表）”，默认使用“super（形参列表）”。

## instanceof

为了不在强转时出现异常，通常使用instanceof进行判断。

a instanceof A

判断对象a是否是类A的实例。如果是，返回true；否则，返回false。

如果a instanceof A 返回true，则a instanceof A的父类，以一定返回true

## static

当我们编写一个类时，其实是在描述对象的属性和方法，而没有产生实质上的对象，只有new出来以后才能产生对象，这时系统才会给对象分配内存空间，方法才能被外部调用。

我们有时候希望，属性等不为某一个特定的对象所有。例如每个中国人的对象属性都是中国，而不必在每一个中国人的实例中声明。

static意为静态的，用来修饰属性，方法，代码块，内部类。

* + 1. static修饰属性

属性被static修饰的，为静态属性，或称为静态变量，类变量；否则，为非静态属性，或称为实例变量。

1. 实例变量：

每个对象，独立地拥有一套实例变量。当修改一个对象的实例变量时，不会导致其他对象中的修改。

1. 静态变量：

所有对象，共享静态属性，当通过某一个对象去修改静态属性时，会导致其他对象的属性也是修改过的。

静态变量，随着类的加载而加载，早于对象的创建，可以直接以“类·静态变量”的方式调用。

由于类只加载一次，则静态变量在内存中也只会加载一份。

例如，System·out，Math·PI

* + 1. static修饰方法

使用static修饰的方法：静态方法

随着类的加载而加载，可以“类·静态方法”的方式来调用；也可以以“对象·静态方法”的方式来调用。

静态方法中只能调用静态方法或静态属性。

在静态方法中，不能使用this和super关键字。因为this和super都是基于对象的。

类中的常量，常常声明为static。

* + 1. 属性和方法是否应该static的经验

属性可以被多个对象所共享的，不会随着对象不同而不同的。

工具类中的方法，习惯上声明为static的。

一个属性时static的，通常，操作这个属性的方法也是static的。

## final

final可以用来修饰类，方法，变量。意为最终的。

* + 1. final修饰类

final修饰的类，不可再有子类，即不能被继承。

例如，String类，System类，StringBuffer类等。

* + 1. final修饰方法

final修饰的方法，不可再被重写。

例如，Object类中的getClass()等。

* + 1. final修饰变量

final修饰的变量，不可再变，成为一个常量。

1. final修饰属性

可以考虑赋值的位置，显式初始化，代码块中初始化，在构造器中初始化。

1. final修饰局部变量

尤其是用final修饰形参时，表明形参时一个常量。当我们调用此方法时，给常量形参赋一个实参，一旦赋值，方法中只能用这个形参，不能再重新赋值。

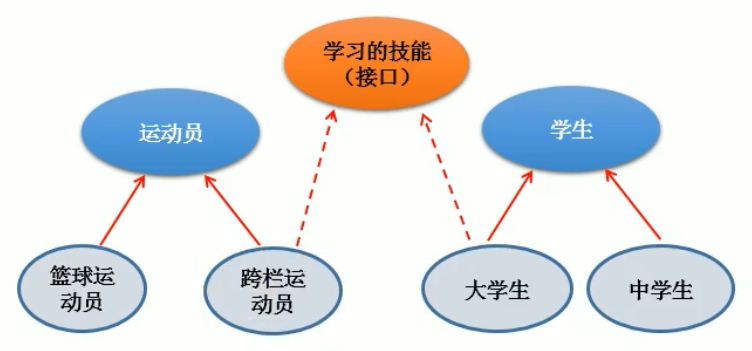
* + 1. static final

static final 修饰的称为全局常量。尤其是在接口中，属性都是static final的。

大写的属性一般都是常量。

## interface

接口。



接口使用interface来定义。

接口和类是并列的结构。

* + 1. 定义接口的结构

1. 在JDK7之前，只能定义全局常量和抽象方法。

全局常量：public static final 书写时可以不写这些修饰符

抽象方法：public abstract 书写时可以不写这些修饰符

B．在JDK8之后，还可以定义静态方法和默认方法。

C．接口中不能定义构造器，即接口不能实例化。

D．在Java开发中，接口都通过让类去实现[implement]的方式来使用。如果实现类覆盖了接口中的所有抽象方法，则这个类可以实例化；否则，这个类仍然是一个抽象类。

E．多实现：Java类可以实现多个接口，弥补了Java单继承性的局限。

F．格式：

class Plane extends Object implements Flayable {

@Override

public void fly() {

}

}

1. 接口和接口之间可以继承，且支持多继承。
   * 1. 接口的多态性

接口的具体使用体现了多态性。

USB的例子：USBTest.java

接口的实现类的集合，就是驱动。

* + 1. 接口的规范性

接口是一种规范。

* + 1. JDK8新特性

methodTest.java

可以在接口中定义静态方法和默认方法。

interface Method {

public static void static\_method() {

System.*out*.println("Declare static method is allowed in interface.");

}

public default void default\_method() {

System.*out*.println("Declare default method is allowed in interface.");

}

}

1. 接口中定义的静态方法，只能通过接口来调用。
2. 通过实现类的对象，可以调用接口中的默认方法。如果实现类重写了接口中的默认方法，调用时，调用的是重写以后的方法。
3. 如果子类[实现类]继承的父类和实现的接口中声明了同名同参数的方法，那么子类在没有重写此方法的前提下，调用的是父类中的方法。[类优先原则，只针对方法，对属性无效]。
4. 如果实现类实现了多个接口，而这些接口中定义了同名同参数的默认方法，则在实现类没有重写此方法的情况下，会报错。[接口冲突]。必须在实现类中重写此方法。
5. 在子类[实现类]中调用父类[接口]中被重写的方法，需要加super。

class ImMethod implements Method {

@Override

public void default\_method() {

System.*out*.println("Override default\_method");

}

public void method\_test() {

default\_method();

Method.super.default\_method();

}

}

# 专题

## Java数据越界

* + 1. int型越界

Math.abs(-2147483648) = -2147483648

<https://blog.csdn.net/j572864517/article/details/106938237/?utm_medium=distribute.pc_relevant.none-task-blog-title-2&spm=1001.2101.3001.4242>

## 子类对象实例化的全过程

* + 1. 从结果上来看

子类继承父类以后，就获取了分类中声明的属性和方法

创建子类的对象，在堆空间中，就会加载“所有父类”中声明的属性和方法

* + 1. 从过程上来看

当我们通过子类的构造器创建子类对象时，我们一定会直接或间接地调用其父类的构造器，进而调用其父类的父类的构造器，直到调用java.lang.Object类中的空参构造器。正因为加载了所有的父类的结构，所以在内存中才会有父类中的结构，子类对象才可以调用。

* + 1. 注意

虽然创建子类对象时，调用了父类构造器，但是自始至终只创建了一个对象，即子类的对象。

## Object类的使用

Java.lang.Object类是所有Java类的根父类

如果没有显式地指明一个类的父类，那么其父类就是Java.lang.Object类

Object类中的功能[方法]具有通用性

Object没有声明属性，只声明了一个空参构造器

* + 1. equals方法

== 和 equals() 的区别

== 可以使用在基本数据类型变量和引用数据类型变量中。使用在基本数据类型中，比较的是变量保存的数据是否相等，不一定要类型相同[有数据类型提升]；使用在引用数据变量中，比较的是两个变量的地址值是否相同，即是否指向同一个对象实体。

equals()方法的使用

equal()是一个方法，而不是一个运算符

equal()只适用于引用数据类型

Object中定义的 equals() 和 == 的作用是相同的，比较两个变量的地址值是否相同。

像String，Data，File，包装类，都重写了equals()方法，重写以后比的不再是地址，而比较的是实体内容。

通常情况下，自定义的类使用equals()，通常想比较的是对象的实体内容，就需要对Object的equals()进行重写。

* + 1. toString方法

String，Data等都 重写了toString方法，使得在调用toString()时，返回的是实体内容。

自定义类也可以重写toString方法，使得返回实体内容。

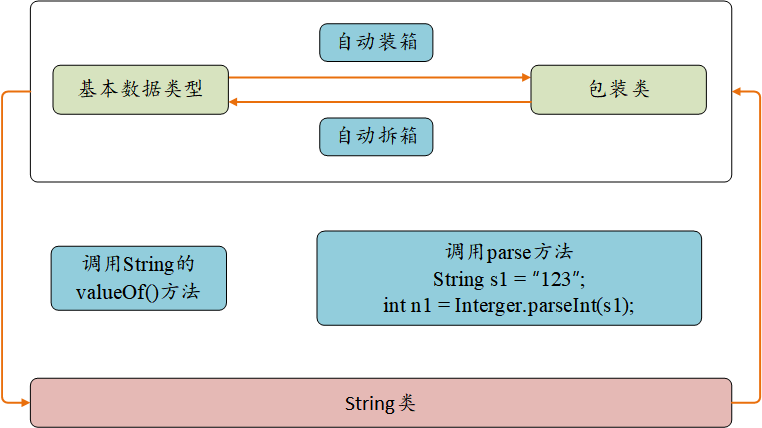
## 包装类的使用

针对Java中基本数据类型定义相应的引用类型——包装类[封装类]

Java提供了8中基本数据类型的包装类，使得基本数据类型具有类的特征。

|  |  |
| --- | --- |
| 基本数据类型 | 包装类 |
| byte | Byte |
| short | Short |
| **int** | **Integer** |
| long | Long |
| float | Float |
| double | Double |
| boolean | Boolean |
| **char** | **Character** |

前六个包装类都是数值型的，继承于Number



## 设计模式

* + 1. 单例设计模式

类中只存在一个对象。

类的构造器的访问权限为private。

“饿汉式”：SingleTonTest1.java。好处是线程安全，坏处是加载时间长。

“懒汉式”：SingleTonTest2.java。坏处是线程不安全。

* + 1. 代理模式

NetWorkTest.java

* + 1. 工厂模式

## main()方法

main()是程序的入口

main()也是一个普通的静态方法

main()可以作为和控制台交互的方式