

内部类与常用类

Java Platform Standard Edition

郑春光

课程目标

CONTENTS

ITEMS **1** 内部类

ITEMS **2** Object类

ITEMS **3** Object类常用方法

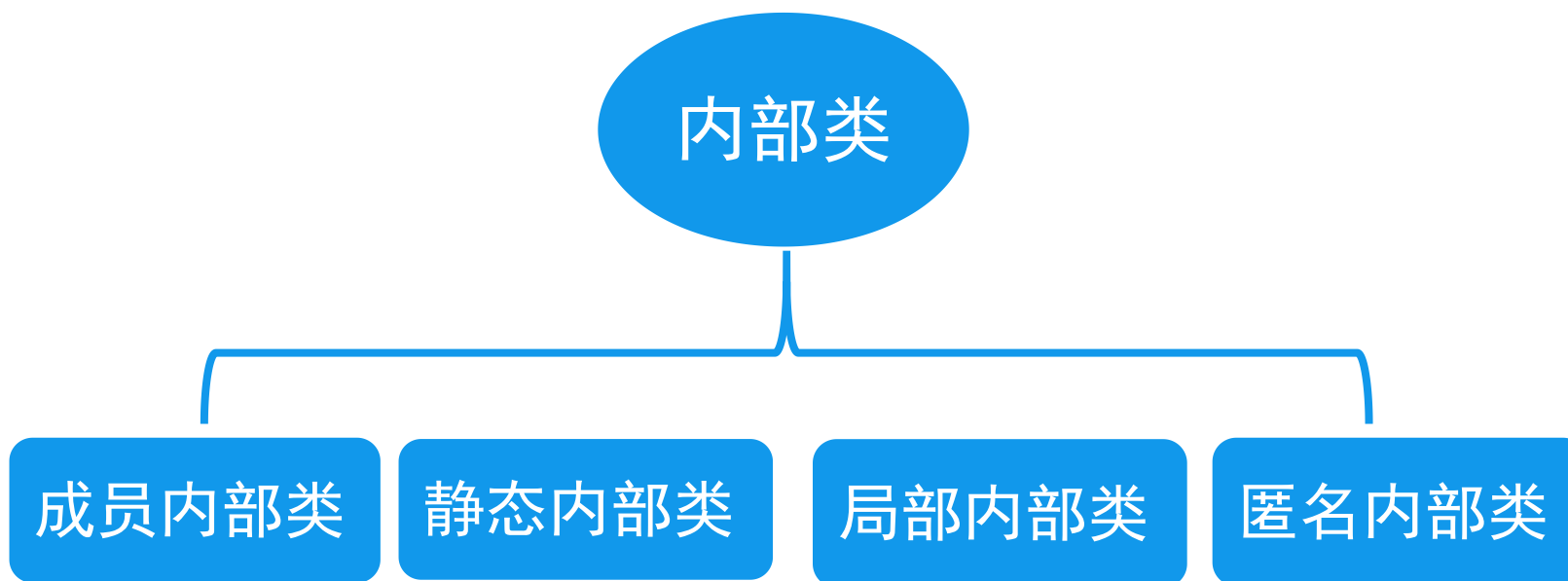
ITEMS **4** 包装类

ITEMS **5** String类

ITEMS **6** BigDecimal类

内部类

Java Platform Standard Edition





什么是内部类

- 概念：在一个类的内部再定义一个完整的类。
- 特点：
 - 编译之后可生成独立的字节码文件。
 - 内部类可直接访问外部类的私有成员，而不破坏封装。
 - 可为外部类提供必要的内部功能组件。

```
class Outer{  
    class Inner{  
  
    }  
}
```

编译

 Outer\$Inner.class
 Outer.class

- 在类的内部定义，与实例变量、实例方法同级别的类。
- 外部类的一个实例部分，创建内部类对象时，必须依赖外部类对象。
 - `Outer out = new Outer();`
 - `Inner in = out.new Inner();`
- 当外部类、内部类存在重名属性时，会优先访问内部类属性。
- 成员内部类不能定义静态成员。

- 不依赖外部类对象，可直接创建或通过类名访问，可声明静态成员。
- 只能直接访问外部类的静态成员（实例成员需实例化外部类对象）。
 - `Outer.Inner inner = new Outer.Inner();`
 - `Outer.Inner.show();`

- 定义在外部类方法中，作用范围和创建对象范围仅限于当前方法。
- 局部内部类访问外部类当前方法中的局部变量时，因无法保障变量的生命周期与自身相同，变量必须修饰为`final`。
- 限制类的使用范围。

- 没有类名的局部内部类（一切特征都与局部内部类相同）。
- 必须继承一个父类或者实现一个接口。
- 定义类、实现类、创建对象的语法合并，只能创建一个该类的对象。
- 优点：减少代码量。
- 缺点：可读性较差。

Object类

Java Platform Standard Edition

- 超类、基类，所有类的直接或间接父类，位于继承树的最顶层。
- 任何类，如没有书写extends显示继承某个类，都默认直接继承Object类，否则为间接继承。
- Object类中所定义的方法，是所有对象都具备的方法。
- Object类型可以存储任何对象。
 - 作为参数，可接受任何对象。
 - 作为返回值，可返回任何对象。

getClass()方法

- `public final Class<?> getClass() {}`
- 返回引用中存储的实际对象类型。
- 应用：通常用于判断两个引用中实际存储对象类型是否一致。

hashCode()方法

- `public int hashCode() {}`
- 返回该对象的十六进制的哈希码值。
- 哈希算法根据对象的地址或字符串或数字计算出来的int类型的数值。
- 哈希码并不唯一，可保证相同对象返回相同哈希码，尽量保证不同对象返回不同哈希码。

toString()方法

- `public String toString() {}`
- 返回该对象的字符串表示（表现形式）。
- 可以根据程序需求覆盖该方法，如：展示对象各个属性值。

equals()方法

- `public boolean equals(Object obj) {}`
- 默认实现为 `(this == obj)`，比较两个对象地址是否相同。
- 可进行覆盖，比较两个对象的内容是否相同。

equals()方法覆盖步骤

- 比较两个引用是否指向同一个对象。
- 判断obj是否为null。
- 判断两个引用指向的实际对象类型是否一致。
- 强制类型转换。
- 依次比较各个属性值是否相同。

finalize()方法

- 当对象被判定为垃圾对象时，由JVM自动调用此方法，用以标记垃圾对象，进入回收队列。
- 垃圾对象：没有有效引用指向此对象时，为垃圾对象。
- 垃圾回收：由GC销毁垃圾对象，释放数据存储空间。
- 自动回收机制：JVM的内存耗尽，一次性回收所有垃圾对象。
- 手动回收机制：使用System.gc()；通知JVM执行垃圾回收。

包装类

Java Platform Standard Edition

什么是包装类？

- 基本数据类型所对应的引用数据类型。
- Object可统一所有数据，包装类的默认值是null。

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

类型转换与装箱、拆箱

- 8种包装类提供不同类型间的转换方式：
 - Number父类中提供的6个共性方法。
 - `parseXXX()` 静态方法。
 - `valueOf()` 静态方法。
- 注意：需保证类型兼容，否则抛出`NumberFormatException`异常。
- JDK 5.0之后，自动装箱、拆箱。基本数据类型和包装类自动转换。

- Java预先创建了256个常用的整数包装类型对象。
- 在实际应用当中，对已创建的对象进行复用。

String类

Java Platform Standard Edition

String

- 字符串是常量，创建之后不可改变。
- 字符串字面值存储在字符串池中，可以共享。
- `String s = "Hello";` 产生一个对象，字符串池中存储。
- `String s = new String("Hello");` //产生两个对象，堆、池各存储一个。

- `public char charAt(int index)` : 根据下标获取字符。
- `public boolean contains(String str)` : 判断当前字符串中是否包含str。
- `public char[] toCharArray()` : 将字符串转换成数组。
- `public int indexOf(String str)` : 查找str首次出现的下标, 存在, 则返回该下标; 不存在, 则返回-1。
- `public int lastIndexOf(String str)` : 查找字符串在当前字符串中最后一次出现的下标索引。
- `public int length()` : 返回字符串的长度。
- `public String trim()` : 去掉字符串前后的空格。
- `public String toUpperCase()` : 将小写转成大写。
- `public boolean endsWith(String str)` : 判断字符串是否以str结尾。
- `public String replace(char oldChar, char newChar)` : 将旧字符串替换成新字符串
- `public String[] split(String str)` : 根据str做拆分。

- StringBuffer: 可变长字符串, JDK1.0提供, 运行效率慢、线程安全。
- StringBuilder: 可变长字符串, JDK5.0提供, 运行效率高、线程不安全。

BigDecimal

- 思考：以下程序输出结果是多少？

```
public class TestBigDecimal {  
    public static void main(String[] args) {  
        double d1=1.0;  
        double d2=0.9;  
        System.out.println(d1-d2);  
    }  
}
```

输出结果：

0.09999999999999998



很多实际应用中需要精确运算，而double是近似值存储，不在符合要求，需要借助BigDecimal。

BigDecimal

- 位置：java.math包中。
- 作用：精确计算浮点数。
- 创建方式：`BigDecimal bd=new BigDecimal(“1.0”);`
- 方法：
 - `BigDecimal add(BigDecimal bd)` 加
 - `BigDecimal subtract(BigDecimal bd)` 减
 - `BigDecimal multiply(BigDecimal bd)` 乘
 - `BigDecimal divide(BigDecimal bd)` 除

- 利用BigDecimal可以进行数值计算：

```
public class TestBigDecimal {  
    public static void main(String[] args) {  
  
        BigDecimal bd1 = new BigDecimal("1.0");  
        BigDecimal bd2 = new BigDecimal("0.9");  
  
        BigDecimal result1 = bd1.add(bd2);  
        System.out.println("bd1+bd2="+result1);  
  
        BigDecimal result2 = bd1.subtract(bd2);  
        System.out.println("bd1-bd2="+result2);  
  
        BigDecimal result3 = bd1.multiply(bd2);  
        System.out.println("bd1*bd2="+result3);  
  
        BigDecimal result4 = bd1.divide(bd2);  
        System.out.println("bd1/bd2="+result4);  
    }  
}
```

输出结果：

bd1+bd2=1.9

bd1-bd2=0.1

bd1*bd2=0.90

执行除法运算时，抛出错误

进行除法运算时，如果不能准确的计算出结果时需要指定保留的位数和取舍方式。

- 除法： `BigDecimal (BigDecimal bd, int scal, RoundingMode mode)`
- 参数 `scal` ： 指定精确到小数点后几位。
- 参数 `mode` ：
 - 指定小数部分的取舍模式，通常采用四舍五入的模式，
 - 取值为 `BigDecimal.ROUND_HALF_UP`。

- 内部类：
 - 在一个类的内部再定义一个完整的类。
 - 成员内部类、静态内部类、局部内部类、匿名内部类。
- Object类：
 - 所有类的直接或间接父类，可存储任何对象。
- 包装类：
 - 基本数据类型所对应的引用数据类型，可以使Object统一所有数据。
- String类：
 - 字符串是常量，创建之后不可改变，字面值保存在字符串池中，可以共享。
- BigDecimal：
 - 可精确计算浮点数。