JAVA笔记DAY6

1. 对象判断相等

```
        BirthDay b1 = new BirthDay(2000,1,1);
        BirthDay b2 = new BirthDay(2000,1,1);
        System.out.println(b1==b2);//false    哈希值相等    @15db9742
```

哈希值相等的对象不一定是同一个对象。

2. 匿名对象

```
new BirthDay(2020,1,1);//匿名对象
```

拥有普通引用对象的全部功能，一般无法再调用

一个对象可以被多个引用指向，也可以没有引用指向（匿名对象）

一个引用[变量],可以指向0个或1个对象

== 基本数据类型比较值　　引用数据类型比较地址

3. THIS关键字

1.解决构造方法中重名的问题

2. `this( )` 调用自身构造 `super` 调用父类 `super( )` 父类构造

3.可以调用类中其他动态方法

4.方法标识

属性名：数据类型-

空心：参数

实心：方法 圆绿-public 菱黄-protected 三蓝-default 方红-private

带s 静态

5. 封装

类的封装 缩小可见性 防止该类代码和数据被外部类随意访问（隐藏成员变量名）

属性私有化，方法公开

case 1：设计一个类并封装

类中使用this关键字 1.构造方法中调用构造方法 2.动态方法中调用其他方法

6. PACKAGE、IMPORT关键字

package 包 每个类前 用package指明所在包 写在首行

import 导入：在类前导入需要的包/类

同一个项目中，尽量不要创建相同的类

7. STATIC关键字

① 可以修饰变量

成员变量/全局变量 被叫做静态变量/类变量

② 可以修饰方法

③ 可以修饰代码块

代码块/动态代码块/初始化类中{}包裹的代码片段　类加载一次（调用该类的构造方法）运行一次

被static修饰的静态代码块　类首次加载时运行一次并只运行一次

先静后动　最后构造方法

累的加载顺序：调用类的构造方法

静态代码块/类变量：取决于书写顺序 --代码块/成员变量

8. 继承

extends

类声明处：　a extends b　a继承b

b被称为父类（fatherClass）/超类（superClass）

a 被称为子类（childClass）/继承类（derivedClass）

定义：新建一个类 从已有的类中获取属性和行为

子类对象　可以获取父类中全部属性和行为（可见的）

java的类单继承

一个父类可以拥有任意个子类

一个子类只有一个父类

9. 重写

- 重载：overload

方法同名不同参数

- 重写：override

父类方法不满足需求时，子类可以重写父类方法

1.重写方法 同名同参数同返回类型

2.重写时 可以添加@Override注解 加强代码可读性

3.只要父类方法被重写子类对象只能调用 重写后的该方法

10. SUPPER关键字

super() 可以调用父类构造方法

子类方法中会调用父类构造--super（） --缺省

如果要写super（），必须写在方法首行

super. 用来调用父类的方法或属性

11. FINAL关键字

1.修饰变量

变量->常量

2.修饰方法

方法不可被重写

3.修饰类

类不可被继承

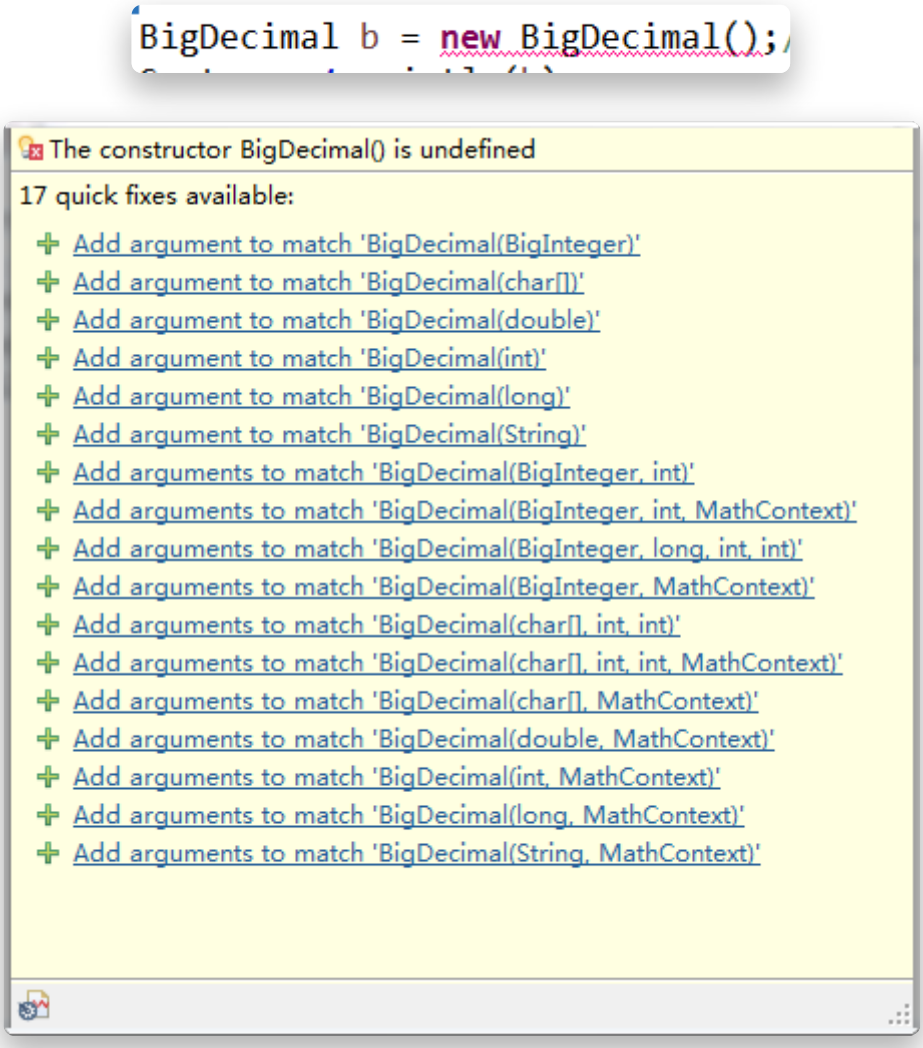
final通常和static一起使用

确保单例

12. BIGDECIMAL

```
1 BigDecimal b = new BigDecimal()
```

！！必须赋值



double精度不准确，String是确定的

常用方法

`add ( )` `subtract ( )` `multiply()` `divide ( )`

一个重要的参数 **scale** 精度

H4 加法

```
/**
     * 提供精确的加法运算
     *
     * @param v1 被加数
     * @param v2 加数
     * @return 两个参数的和
     */

    public static double add(double v1, double v2) {
        BigDecimal b1 = new BigDecimal(Double.toString(v1));
        BigDecimal b2 = new BigDecimal(Double.toString(v2));
        return b1.add(b2).doubleValue();
    }

    /**
     * 提供精确的加法运算
     *
     * @param v1 被加数
     * @param v2 加数
     * @return 两个参数的和
     */
    public static BigDecimal add(String v1, String v2) {
        BigDecimal b1 = new BigDecimal(v1);
        BigDecimal b2 = new BigDecimal(v2);
        return b1.add(b2);
    }

    /**
     * 提供精确的加法运算
     *
     * @param v1    被加数
     * @param v2    加数
     * @param scale 保留scale 位小数
```

```
     * @return 两个参数的和
     */
    public static String add(String v1, String v2, int scale) {
        if (scale < 0) {
            throw new IllegalArgumentException(
                    "The scale must be a positive integer or zero");
        }
        BigDecimal b1 = new BigDecimal(v1);
        BigDecimal b2 = new BigDecimal(v2);
        return b1.add(b2).setScale(scale,
BigDecimal.ROUND_HALF_UP).toString();
    }
```

H4 减法

```
/**
     * 提供精确的减法运算
     *
     * @param v1 被减数
     * @param v2 减数
     * @return 两个参数的差
     */
    public static double sub(double v1, double v2) {
        BigDecimal b1 = new BigDecimal(Double.toString(v1));
        BigDecimal b2 = new BigDecimal(Double.toString(v2));
        return b1.subtract(b2).doubleValue();
    }

    /**
     * 提供精确的减法运算。
     *
     * @param v1 被减数
     * @param v2 减数
     * @return 两个参数的差
     */
    public static BigDecimal sub(String v1, String v2) {
        BigDecimal b1 = new BigDecimal(v1);
```

```
        BigDecimal b2 = new BigDecimal(v2);
        return b1.subtract(b2);
    }

    /**
     * 提供精确的减法运算
     *
     * @param v1    被减数
     * @param v2    减数
     * @param scale 保留scale 位小数
     * @return 两个参数的差
     */
    public static String sub(String v1, String v2, int scale) {
        if (scale < 0) {
            throw new IllegalArgumentException(
                    "The scale must be a positive integer or zero");
        }
        BigDecimal b1 = new BigDecimal(v1);
        BigDecimal b2 = new BigDecimal(v2);
        return b1.subtract(b2).setScale(scale,
BigDecimal.ROUND_HALF_UP).toString();
    }
```

H4 乘法

```
/**
     * 提供精确的乘法运算
     *
     * @param v1 被乘数
     * @param v2 乘数
     * @return 两个参数的积
     */
    public static double mul(double v1, double v2) {
        BigDecimal b1 = new BigDecimal(Double.toString(v1));
        BigDecimal b2 = new BigDecimal(Double.toString(v2));
        return b1.multiply(b2).doubleValue();
    }
```

```java

    /**
     * 提供精确的乘法运算
     *
     * @param v1 被乘数
     * @param v2 乘数
     * @return 两个参数的积
     */
    public static BigDecimal mul(String v1, String v2) {
        BigDecimal b1 = new BigDecimal(v1);
        BigDecimal b2 = new BigDecimal(v2);
        return b1.multiply(b2);
    }

    /**
     * 提供精确的乘法运算
     *
     * @param v1    被乘数
     * @param v2    乘数
     * @param scale 保留scale 位小数
     * @return 两个参数的积
     */
    public static double mul(double v1, double v2, int scale) {
        BigDecimal b1 = new BigDecimal(Double.toString(v1));
        BigDecimal b2 = new BigDecimal(Double.toString(v2));
        return round(b1.multiply(b2).doubleValue(), scale);
    }

    /**
     * 提供精确的乘法运算
     *
     * @param v1    被乘数
     * @param v2    乘数
     * @param scale 保留scale 位小数
     * @return 两个参数的积
     */
    public static String mul(String v1, String v2, int scale) {
        if (scale < 0) {
            throw new IllegalArgumentException(
                    "The scale must be a positive integer or zero");
        }
        BigDecimal b1 = new BigDecimal(v1);
        BigDecimal b2 = new BigDecimal(v2);
```

```
56          return b1.multiply(b2).setScale(scale,
    BigDecimal.ROUND_HALF_UP).toString();
57      }
```

H4 除法

```
1  //精度
2  private static final int DEF_DIV_SCALE = 10;
3
4
5  /**
6       * 提供（相对）精确的除法运算，当发生除不尽的情况时，精确到
7       * 小数点以后10位，以后的数字四舍五入
8       *
9       * @param v1 被除数
10      * @param v2 除数
11      * @return 两个参数的商
12      */
13
14     public static double div(double v1, double v2) {
15         return div(v1, v2, DEF_DIV_SCALE);
16     }
17
18     /**
19      * 提供（相对）精确的除法运算。当发生除不尽的情况时，由scale参数指
20      * 定精度，以后的数字四舍五入
21      *
22      * @param v1    被除数
23      * @param v2    除数
24      * @param scale 表示表示需要精确到小数点以后几位。
25      * @return 两个参数的商
26      */
27     public static double div(double v1, double v2, int scale) {
28         if (scale < 0) {
```

```
            throw new IllegalArgumentException("The scale must be a positive integer or zero");
        }
        BigDecimal b1 = new BigDecimal(Double.toString(v1));
        BigDecimal b2 = new BigDecimal(Double.toString(v2));
        return b1.divide(b2, scale, BigDecimal.ROUND_HALF_UP).doubleValue();
    }

    /**
     * 提供（相对）精确的除法运算。当发生除不尽的情况时，由scale参数指
     * 定精度，以后的数字四舍五入
     *
     * @param v1    被除数
     * @param v2    除数
     * @param scale 表示需要精确到小数点以后几位
     * @return 两个参数的商
     */
    public static String div(String v1, String v2, int scale) {
        if (scale < 0) {
            throw new IllegalArgumentException("The scale must be a positive integer or zero");
        }
        BigDecimal b1 = new BigDecimal(v1);
        BigDecimal b2 = new BigDecimal(v1);
        return b1.divide(b2, scale, BigDecimal.ROUND_HALF_UP).toString();
    }

    /**
     * 提供精确的小数位四舍五入处理
     *
     * @param v     需要四舍五入的数字
     * @param scale 小数点后保留几位
     * @return 四舍五入后的结果
     */
    public static double round(double v, int scale) {
        if (scale < 0) {
            throw new IllegalArgumentException("The scale must be a positive integer or zero");
        }
        BigDecimal b = new BigDecimal(Double.toString(v));
        return b.setScale(scale, BigDecimal.ROUND_HALF_UP).doubleValue();
    }
```

```
68
69      /**
70       * 提供精确的小数位四舍五入处理
71       *
72       * @param v     需要四舍五入的数字
73       * @param scale 小数点后保留几位
74       * @return 四舍五入后的结果
75       */
76      public static String round(String v, int scale) {
77          if (scale < 0) {
78              throw new IllegalArgumentException(
79                      "The scale must be a positive integer or zero");
80          }
81          BigDecimal b = new BigDecimal(v);
82          return b.setScale(scale, BigDecimal.ROUND_HALF_UP).toString();
83      }
84
85      /**
86       * 取余数
87       *
88       * @param v1    被除数
89       * @param v2    除数
90       * @param scale 小数点后保留几位
91       * @return 余数
92       */
93      public static String remainder(String v1, String v2, int scale) {
94          if (scale < 0) {
95              throw new IllegalArgumentException(
96                      "The scale must be a positive integer or zero");
97          }
98          BigDecimal b1 = new BigDecimal(v1);
99          BigDecimal b2 = new BigDecimal(v2);
100         return b1.remainder(b2).setScale(scale,
    BigDecimal.ROUND_HALF_UP).toString();
101     }
102
103     /**
104      * 取余数  BigDecimal
105      *
106      * @param v1    被除数
107      * @param v2    除数
108      * @param scale 小数点后保留几位
109      * @return 余数
```

```
110      */
111     public static BigDecimal remainder(BigDecimal v1, BigDecimal v2, int
    scale) {
112         if (scale < 0) {
113             throw new IllegalArgumentException(
114                     "The scale must be a positive integer or zero");
115         }
116         return v1.remainder(v2).setScale(scale, BigDecimal.ROUND_HALF_UP);
117     }
118
119     /**
120      * 比较大小
121      *
122      * @param v1 被比较数
123      * @param v2 比较数
124      * @return 如果v1 大于v2 则 返回true 否则false
125      */
126     public static boolean compare(String v1, String v2) {
127         BigDecimal b1 = new BigDecimal(v1);
128         BigDecimal b2 = new BigDecimal(v2);
129         int bj = b1.compareTo(b2);
130         boolean res;
131         if (bj > 0)
132             res = true;
133         else
134             res = false;
135         return res;
136     }
```