1.1 基础常识

➢软件，即一系列按照特定顺序组织的计算机数据和指令的集合。有系统软件和应用软件之分。

➢GUI：图形化界面

➢常用的DOS命令：

```
dir :     列出当前目录下的文件以及文件夹
md :    创建目录
rd :      删除目录
cd :     进入指定目录
cd.. :  退回到上一级目录
cd\:     退回到根目录
del :     删除文件
exit :    退出 dos 命令行
```

1.2 Java语言概述

➢Java 属于第三代语言，第一代是打孔机，第二代是汇编

➢Java SE ： 桌面开发 Java EE ： web开发 Java ME:： 移动端开发

➢JDK与JRE与JVM 的区别和联系 JDK>JREJVM

➢JDK目前企业主要使用版本是8

JDK 1.0	Oak(橡树)	1996-01-23
JDK 1.1		1997-02-19
JDK 1.1.4	Sparkler（宝石）	1997-09-12
JDK 1.1.5	Pumpkin（南瓜）	1997-12-13
JDK 1.1.6	Abigail（阿比盖尔–女子名）	1998-04-24
JDK 1.1.7	Brutus（布鲁图–古罗马政治家和将军）	1998-09-28
JDK 1.1.8	Chelsea（切尔西–城市名）	1999-04-08
J2SE 1.2	Playground（运动场）	1998-12-04
J2SE 1.2.1	none（无）	1999-03-30
J2SE 1.2.2	Cricket（蟋蟀）	1999-07-08
J2SE 1.3	Kestrel（美洲红隼）	2000-05-08
J2SE 1.3.1	Ladybird（瓢虫）	2001-05-17
J2SE 1.4.0	Merlin（灰背隼）	2002-02-13
J2SE 1.4.1	grasshopper（蚱蜢）	2002-09-16
J2SE 1.4.2	Mantis（螳螂）	2003-06-26
Java SE 5.0 (1.5.0)	Tiger（老虎）	2004-09-30
Java SE 6.0 (1.6.0)	Mustang（野马）	2006-04
Java SE 7.0 (1.7.0)	Dolphin（海豚）	2011-07-28
Java SE 8.0 (1.8.0)	Spider（蜘蛛）	2014-03-18

1.3 Java 语言运行机制

●面向对象

➢两个基本概念：类、对象

➢三大特性：封装、继承、多态

●特点二：健壮性

➢吸收了C/C++语言的优点，但去掉了其影响程序健壮性的部分（如指针、内存的申请与释放等），提供了一个相对安全的内存管理和访问机制

●特点三：跨平台性

➢跨平台性：通过Java语言编写的应用程序在不同的系统平台上都可以运行。“Write once , Run Anywhere”

➢原理：只要在需要运行 java 应用程序的操作系统上，先安装一个Java虚拟机 (JVM Java Virtual Machine) 即可。由JVM来负责Java程序在该系统中的运行。

➢Java 编译后的文件是.class 文件

1.4 Java环境搭建

➢下载jdk,配置JDK

1、首先要打开系统环境变量配置的页面。具体操作是：桌面上找到“此电脑”，然后右键 “属性”。

2、然后打开高级系统配置，在弹出的界面打开环境变量

3、在弹出的页面，“系统变量区域”点新建按钮

4、新建->变量名：JAVA_HOME 变量值：C:\Program Files\Java\jdk1.8.0_181 (即JDK的安装路径)

5、在“系统变量”栏里找到Path,选中后单击编辑，新增%JAVA_HOME%\bin;

6、java -version 验证是否成功，出现JDK版本信息就是成功了

➢注意：JDK5以后不需要再配置classpath

```
public class HelloWorld {
    public static void main(String []args) {
       System.out.println("Hello World");
    }
}
cmd运行以下命令：
javac HelloWorld.java
java HelloWorld
//出现表示成功
Hello World
```

➢分析源码System.out.println 涉及知识点 后面讲到的final 和方法的重载

```
 * @author  unascribed
 * @since   JDK1.0
 */
public final class System {

    /* register the natives via the static initializer.
     *
     * VM will invoke the initializeSystemClass method to complete
     * the initialization for this class separated from clinit.
     * Note that to use properties set by the VM, see the constraints
     * described in the initializeSystemClass method.
```

```
     *
     * @param x  The <code>String</code> to be printed.
     */
    public void println( @Nullable String x) {
        synchronized (this) {
            print(x);
            newLine();
        }
    }

    /**
     * Prints an Object and then terminate the line.  This method calls
     * at first String.valueOf(x) to get the printed object's string value,
     * then behaves as
     * though it invokes <code>{@link #print(String)}</code> and then
     * <code>{@link #println()}</code>.
     *
     * @param x  The <code>Object</code> to be printed.
     */
    public void println( @Nullable Object x) {
        String s = String.valueOf(x);
        synchronized (this) {
            print(s);
            newLine();
        }
    }
```

➢一个源文件中最多只能有一个public类。其它类的个数不限，如果源文件包含一个public类，则文件名必须按该类名命名

➢Java方法由一条条语句构成，每个语句以“;”结束。

➢Java语言严格区分大小写。

➢大括号都是成对出现的，缺一不可

2.1 关键字

➢被Java语言赋予了特殊含义，用做专门用途的字符串（单词）

用于定义数据类型的关键字				
class	interface	enum	byte	short
int	long	float	double	char
boolean	void			
用于定义数据类型值的关键字				
true	false	null		
用于定义流程控制的关键字				
if	else	switch	case	default
while	do	for	break	continue
return				

用于定义访问权限修饰符的关键字				
private	protected	public		
用于定义类，函数，变量修饰符的关键字				
abstract	final	static	synchronized	
用于定义类与类之间关系的关键字				
extends	implements			
用于定义建立实例及引用实例，判断实例的关键字				
new	this	super	instanceof	
用于异常处理的关键字				
try	catch	finally	throw	throws
用于包的关键字				
package	import			
其他修饰符关键字				
native	strictfp	transient	volatile	assert

➢保留字：现有Java版本尚未使用，但以后版本可能会作为关键字使用。自己命名标记符时要避免使用这些保留字

byValue、cast、future、 generic、 inner、 operator、 outer、 rest、 var 、goto 、const

2.2标识符

➢凡是自己可以起名字的地方都叫标识符

由26个英文字母大小写，0-9 ， _或 $ 组成

数字不可以开头。

不可以使用关键字和保留字，但能包含关键字和保留字。

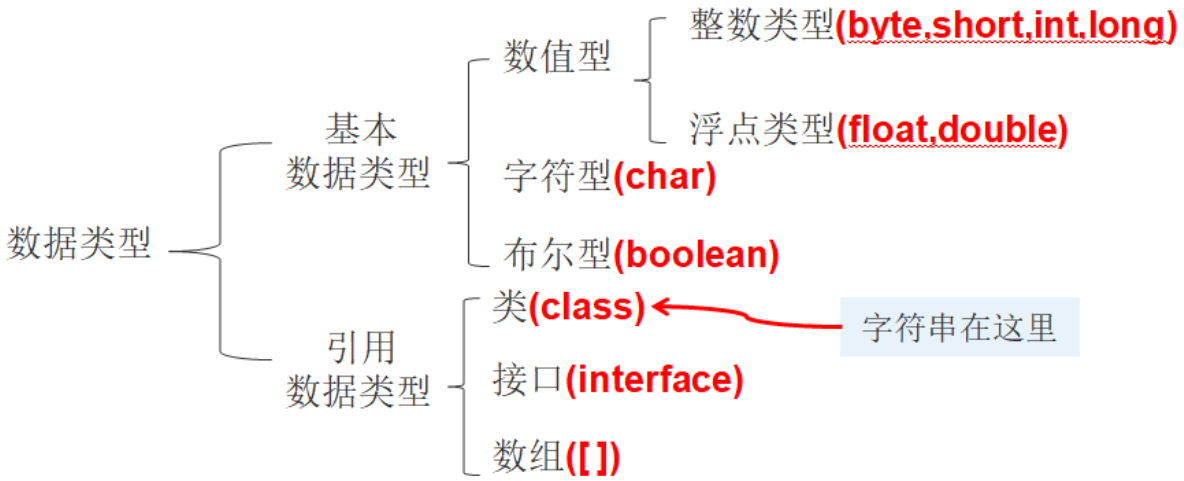
Java中严格区分大小写，长度无限制。

标识符不能包含空格。

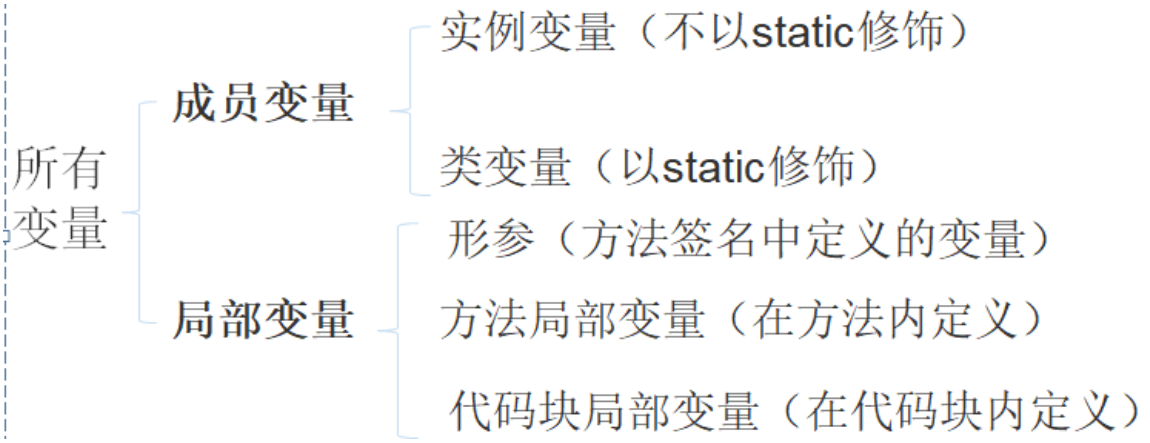
2.3 变 量

➢Java中每个变量必须先声明，后使用

- 按着数据类型分类



●按着声明位置分类



示例代码如下【包含知识点　自动类型转换和强制转换】

```
package com.neuedu.demo.rain;

/**
 *
 变量与类型转化
*/
public class DemoTest {
    //类变量
    private static final String All_PATH = "***";
    private static boolean flag;
    //实例变量
    private String cValue;
    //get set 方法
    public String getcValue() {
        return cValue;
    }
    public void setcValue(String cValue) {
        this.cValue = cValue;
```

```
    }

    public static void main(String[] args) {
        //局部变量
        float a = 1.0f;
        System.out.println(flag);
        System.out.println(a);
        System.out.println("a");
        System.out.println("b");
        System.out.println("++++++++++++++++++++++++++++++++++++++++b");
        System.out.print("a");
        System.out.print("b");
        System.out.println("++++++++++++++++++++++++++++++++++++++++b");
        //自动转换 byte,short,char—> int —> long—> float —> double
        int t13=12;
        long l2=t13;
        //强制转换 --大容量转小的
        int i = 128;
        byte b = (byte) i;
        System.out.println(b);
        int c = (int) 23.99;
        System.out.println(c);

        //包装类型转换
        Double f = 23.99;
        int g = f.intValue();
        System.out.println(g);
        int z = 12;
        Double d = Double.valueOf(z);
        System.out.println(d);
        /*1如何将字串 String 转换成整数 int?
        A. 有两个方法:
        1). int i = Integer.parseInt([String]); 或
                i = Integer.parseInt([String],[int radix]);
        2). int i = Integer.valueOf(my_str).intValue();
        注: 字串转成 Double, Float, Long 的方法大同小异.
        2 如何将整数 int 转换成字串 String ?
        A. 有叁种方法:
        1.) String s = String.valueOf(i);
        2.) String s = Integer.toString(i);
        3.) String s = "" + i;
        注: Double, Float, Long 转成字串的方法大同小异.*/
        //String 和int
```

```
62          String s1="13";
63          int t1=Integer.parseInt(s1);
64          int t2=Integer.valueOf(s1).intValue();
65          //int 转String
66          int tt1=13;
67          String ss1=String.valueOf(tt1);
68          String ss2=Integer.toString(tt1);
69          String ss3=""+tt1;
70
71      }
72  }
```

2.4 运算符

➢算数运算符

运算符	运算	范例	结果
+	正号	+3	3
-	负号	b=4; -b	-4
+	加	5+5	10
-	减	6-4	2
*	乘	3*4	12
/	除	5/5	1
%	取余	7%5	2
++ ++	自增（前）：先运算后取值 自增（后）：先取值后运算	a=2;b=++a; a=2;b=a++;	a=3;b=3 a=3;b=2
- - - -	自减（前）：先运算后取值 自减（后）：先取值后运算	a=2;b=- -a a=2;b=a- -	a=1;b=1 a=1;b=2
+	字符串相加	“He”+”llo”	“Hello”

➢赋值运算符　符号：= 注意和比较运算符==区分

➢比较运算符

运算符	运算	范例	结果
==	相等于	4==3	false
!=	不等于	4!=3	true
<	小于	4<3	false
>	大于	4>3	true
<=	小于等于	4<=3	false
>=	大于等于	4>=3	true
instanceof	检查是否是类的对象	“Hello” instanceof String	true

等号和equals代码示例

```
1  package com.neuedu.demo.object;
2
3  public class StackAndHeap {
4      public void testEquals(){
5          String s="abc";
6          String s1="abc";
7          String s2=new String("abc");
8          boolean flag=(s1==s);
9          boolean flag2=(s==s2);
10         boolean flag3=(s.equals(s2));
11         System.out.println(flag);
12         System.out.println(flag2);
13         System.out.println(flag3);
14     }
15
16     public static void main(String[] args) {
17         StackAndHeap stackAndHeap=new StackAndHeap();
18         stackAndHeap.testEquals();
19     }
20 }
21
```

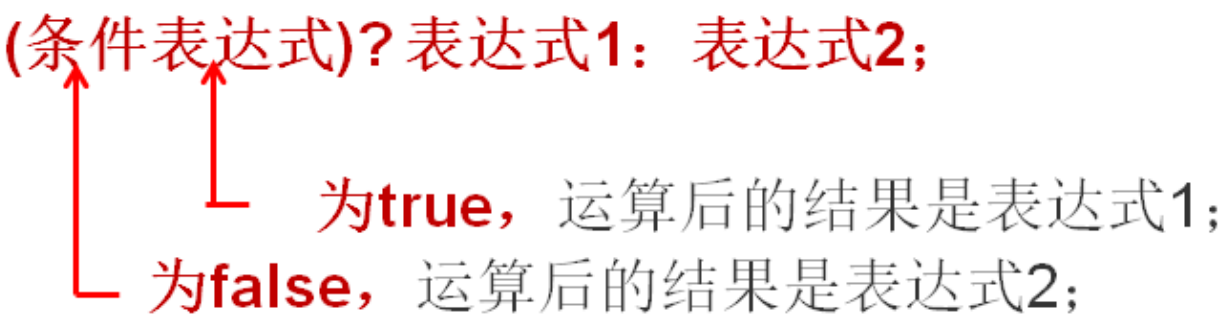
➢逻辑运算符

单&时，左边无论真假，右边都进行运算；

双&时，如果左边为真，右边参与运算，如果左边为假，那么右边不参与运算。

“|”和“||”的区别同理，||表示：当左边为真，右边不参与运算。

➢三元运算符



代码示例：

```
package com.neuedu.demo.rain;
/**
 *
 运算符
 */
public class Operation {
    /*
     *赋值运算符,算数运算符
     */
    public static void add(){
        int a=-1;
        a=a+1;
        System.out.println(a);
    }
   public static void add1(){
       int a=-1;
       a+=1;
       System.out.println(a);
   }
   public static void add2(){
       int a=-1;
       a+=a;
       System.out.println(a);
   }
   public static void add3(){
       int a=-1;
```

```
        a+=a+(-1);
        System.out.println(a);
    }
    /*
    * 逻辑运算符
    * */
    public static void test(){
         boolean flag=(1==1&&1==2);
         boolean flag2=(1==2&&1==1);
         boolean flag3=(1==1&&2==2);
         boolean flag4=(1==1||1==2);

        System.out.println(flag);
        System.out.println(flag2);
        System.out.println(flag3);
        System.out.println(flag4);
    }
    /*三元运算符*/
     public static  void test2(){
         int a=1>2?1:2;
         System.out.println(a);
     }
     public static void test3(){
         int a=10;
         int b=15;
         System.out.println(a>b?a:b);
     }
     public static void main(String[] args) {
        add();
        add1();
        add2();
        add3();
        test();
        test2();
        test3();
     }
}

```

2.5 流程控制

➢if…else和switch两种分支语句

练习题答案代码

```java
public class ifElse {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.println("输入数字");
        int num=sc.nextInt();
        System.out.println("得到输入结果"+num);
    }
}
```

➢程序中不应该出现过多的if…else 了解策略模式

➢for循环

break语句用于终止某个语句块的执行

```
{  ……
   break;
   ……
}
```

continue语句用于跳过某个循环语句块的一次执行

continue语句出现在多层嵌套的循环语句体中时，可以通过标签指明要跳过的是哪一层循环

代码示例

```java
package com.neuedu.demo.rain;

public class TestBreak {
    public static void main(String args[]){
        for(int i = 0; i<10; i++){
            if(i==3){
                break;
            }
            System.out.println(" i =" + i);
        }
        System.out.println("Game Over!");
    }

}
```

```java
package com.neuedu.demo.rain;

public class TestContinue {
    public static void main(String[] args) {
        for(int i = 0; i<10; i++){
            if(i==3){
                continue;
            }
            System.out.println(" i =" + i);
        }
        System.out.println("Game Over!");
    }
}

```

2.6 数组

➢数组是多个相同类型数据的组合，实现对这些数据的统一管理

➢数组中的元素可以是任何数据类型，包括基本数据类型和引用数据类型

➢数组属引用类型，数组型数据是对象(object)，数组中的每个元素相当于该对象的成员变量

➢一维数组 注意 [] 的位置 type var[] 或 type[] var

➢二维数组 二维数组是存储一维数组的一维数组。在java中是可以使用不等长的二维数组 的，这区别与c、c++

➢数组是引用类型，它的元素相当于类的成员变量，因此数组一经分配空间，其中的每个元素也被按照成员变量同样的方式被隐式初始化

```java
package com.neuedu.demo.rain;


import java.sql.Date;

/**
 * 数组（一维数组、二维数组）
 * */
public class ArrTest {
    /*一维数组*/
    public void arrTest() {
```

```
12
13          com.neuedu.demo.constructor.ConstructorJava p=new com.neuedu.demo.cor
14          //动态初始化
15          Boolean[] a = new Boolean[3];
16          a[0] = true;
17          a[2] = true;
18          a[1] = true;
19          //一维数组的遍历
20          for (int i = 0; i < a.length; i++) {
21              System.out.println(a[i]);
22          }
23          System.out.println(a.length);
24          //静态初始化
25          int[] a1 = new int[]{1, 2, 3};
26      }
27
28      /*二维数组*/
29      public void arrTest2(){
30          //二维数组动态初始化
31          int[][] a1=new int[3][2];
32          a1[0][0]=1;
33          a1[0][1]=3;
34          a1[1][0]=5;
35          a1[1][1]=6;
36          a1[2][0]=1;
37          a1[2][1]=8;
38          //二维数组的遍历
39
40          //二维数组静态初始化+练习2
41          int[][] arr=new int[][]{{3,8,2},{2,7},{9,0,1,6}};
42          System.out.println("打印二维数组开始");
43          System.out.println("二维数组的长度为"+arr.length);
44          int sum=0;
45          for(int i=0;i< arr.length;i++){
46              System.out.println("二维数组内的第"+i+"个一维数组的长度为"+arr[i].le
47              for(int j=0;j<arr[i].length;j++){
48                  sum+=arr[i][j];
49                  System.out.println(arr[i][j]);
50              }
51          }
52          System.out.println("打印二维数组结束,最后所有元素求和结果为"+sum);
53
54          System.out.println(a1);
```

```
    }
    public static void main(String[] args) {
        ArrTest arrTest = new ArrTest();
        arrTest.arrTest();
        arrTest.arrTest2();
    }
}

```

数组练习题

```
package com.neuedu.demo.object;

import java.util.Arrays;
/**
 * 学生类
 * */
public class Student {
   /* 学生姓名*/
    private String name;
    /*学生年龄*/
    private int age;
    /*参加的课程*/
    private String[] studyCours;
    /*兴趣*/
    private String[] likeSomethings;
    /*get set 方法*/
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
```

```
32
33      public String[] getStudyCours() {
34          return studyCours;
35      }
36
37      public void setStudyCours(String[] studyCours) {
38          this.studyCours = studyCours;
39      }
40
41      public String[] getLikeSomethings() {
42          return likeSomethings;
43      }
44
45      public void setLikeSomethings(String[] likeSomethings) {
46          this.likeSomethings = likeSomethings;
47      }
48      /*方法：显示学生信息*/
49      public void show(){
50          //for循环遍历所选课程
51          String cours=new String();
52          for(int i=0;i<studyCours.length;i++){
53              cours+=studyCours[i]+"、";
54          }
55          System.out.println(
56                  "学生姓名:"+name+"\n"+
57                  "学生年龄:"+age+"\n"+
58                  "参加课程:"+cours+"\n"+
59                  "兴趣:"+ Arrays.toString(likeSomethings)  //调用工具类直接打印
60          );
61      }
62      /*主函数的调用*/
63      public static void main(String[] args) {
64          //创建对象
65          Student student=new Student();
66          //存放属性值
67          student.setName("张三");
68          student.setAge(22);
69          String[] studyCours={"数学","语文","体育"};
70          student.setLikeSomethings(studyCours);
71          String[] likeSomethings={"看书","篮球","去酒吧"};
72          student.setStudyCours(likeSomethings);
73          //调用显示学生信息方法
74          student.show();
```

```
75
76      }
77  }
```

3.1 面向对象

➢面向对象的三大特征

封装（Encapsulation）

继承（Inheritance）

多态（Polymorphism）

```
1  package com.neuedu.demo.object;
2
3  public class FaceToObject {
4
5      //人
6      人{
7          //男人、女人
8          //年龄
9          操作机器(操作对象,操作类型){
10             //实际实现方法
11         }
12         操作动物(动物品种,装入的机器){
13             //装进去
14         }
15
16     }
17
18     冰箱{
19         开门(){};
20         关门(){};
21     }
22     抽象类动物{
23         进入(){}
24     }
25     大象{
26         进入(){
27             //大卸八块
28         };
```

```
    }
    老虎{
        进入（）{
            //分成五块
        };
    }
}
```

3.2 参数

➢形参、实参、可变参数

形参：方法声明时的参数

实参：方法调用时实际传给形参的参数值

可变参数：...可传可不传 与数组类型参数不能同时存在

```
package com.neuedu.demo.overload;
/**
 * 方法的重载
 * */
public class Overload {
    public static void main(String[] args) {
        Overload o=new Overload();
        o.sayHello();
        o.sayHello("dtrd","drttrd");
//        o.sayHello("dtrd","drttrd","dtrd","drttrd");
        String[] str=new String[]{"dtrd","drttrd","dtrd","drttrd"};
        o.sayHello(str);
    }
    /*无参数的重载*/
    public void sayHello(){
        System.out.println("我是无参数方法");
    }
    /*单参数int 类型的重载*/
    public void sayHello(int a){
        System.out.println("我是参数类型为int类型的方法，本次传入参数为："+a);
    }
    /*单参数String 类型的重载*/
    public  void sayHello(String str){
        System.out.println("我是参数类型为String类型的方法，本次传入参数为："+str
    }
    /*双参数重载方法*/
```

```
    public void sayHello(String str1,String str2){
        System.out.println("我是两个String 参数的方法，参数分别为："+str1+"和"+s

    }
//    /*可变参数的重载方法*/
//    public  void sayHello(String... str){
//        System.out.println("现在进入的是我");
//    }
    public  void sayHello(String[] str){
        System.out.println("我是数组类型参数的方法");
    }
}
```

➢参数传值原理：　　Java里方法的参数传递方式只有一种：值传递。 即将实际参数值的副本（复制品）传入方法内，而参数本身不受影响。

```
package com.neuedu.demo.overload;

public class SwapValue {
    public static void main(String[] args) {
        SwapValue s=new SwapValue();
        s.intValue();
        System.out.println("++++++++++++++++++++++++++");
        s.intValue2();
    }
    public void intValue(){
        int i=2;
        int j=8;
        System.out.println("交换前i为"+i+",j为"+j);

        this.exchageValue(i,j);
        System.out.println("交换后i为"+i+",j为"+j);

    }
    public void exchageValue(int i,int j){
        int temp=i;
        i=j;
        j=temp;
        System.out.println("交换方法里i为"+i+",j为"+j);
    }
    public void intValue2(){
        SwapInt swapInt=new SwapInt();
```

```
27          System.out.println("交换前i为"+swapInt.i+",j为"+swapInt.j);
28          this.swapValue(swapInt);
29          System.out.println("交换后i为"+swapInt.i+",j为"+swapInt.j);
30      }
31      class SwapInt{
32          int i=2;
33          int j=8;
34      }
35      public void swapValue(SwapInt swapInt){
36          int temp=swapInt.i;
37          swapInt.i=swapInt.j;
38          swapInt.j=temp;
39          System.out.println("交换方法里i为"+swapInt.i+",j为"+swapInt.j);
40      }
41  }
```

3.2 修饰符

➢四种访问权限修饰符

修饰符	类内部	同一个包	子类	任何地方
private	Yes			
(缺省)	Yes	Yes		
protected	Yes	Yes	Yes	
public	Yes	Yes	Yes	Yes

3.3 构造器

➢特点

它具有与类相同的名称

它不声明返回值类型。（与声明为void不同）

不能被static、final、synchronized、abstract、native修饰，不能有return语句返回

➢分类

隐式无参构造器（系统默认提供）

显式定义一个或多个构造器（无参、有参）

➢注意点

Java语言中，每个类都至少有一个构造器

默认构造器的修饰符与所属类的修饰符一致

一旦显式定义了构造器，则系统不再提供默认构造器

一个类可以创建多个重载的构造器

父类的构造器不可被子类继承

3.4 JavaBean

与数据库字段相对应的实体，注意get set方法和序列化问题

```
public class BaseEntity implements Serializable {

    /**
     * 序列化id
     */
    private static final long serialVersionUID = 1L;

    /**
     * 搜索值
     */
    @ApiModelProperty(value = "搜索值")
    private String searchValue;

    /**
     * 创建者
     */
    @ApiModelProperty(value = "创建者")
    private String createBy;

    /**
     * 创建时间
```

3.5 import导包问题

➢两种常用方式

```
import java.util.Date;
import java.util.List;
public class ListDemoThe {
    public static void main(String[] args) {

        Date  datedate=new Date();
        java.sql.Date  datedate2=new java.sql.Date(0);

        ListDemoThe list=new ListDemoThe();
    //    list.listDeamo2();
        list.arrToList();
```

4.1 继承与多态

➢注意：多态建立在继承与重写基础上的

➢在Java 中，继承的关键字用的是“extends”，即子类不是父类的子集，而是对父类的“扩展”。

➢Java中类只是支持单继承，不允许多重继承，接口可以多继承

示例代码

```
package com.neuedu.demo.extendsjava;

public class ManKind {
    private int sex;
    private int salary;

    public int getSex() {
        return sex;
    }

    public void setSex(int sex) {
        this.sex = sex;
    }

    public int getSalary() {
        return salary;
    }

    public void setSalary(int salary) {
        this.salary = salary;
    }

    public void manOrWomen() {
        String sexinfo = null;
        if (1==sex) {
            sexinfo = "man";
        } else if (0==sex) {
            sexinfo = "women";
        }
        System.out.println("这个人的性别是" + sexinfo);
    }
    public  void employeed(){
        String isHaveJob=salary==0?"no job":"job";
```

```
        System.out.println("这个人的工作状态是"+isHaveJob);
    }
}
```

```
package com.neuedu.demo.extendsjava;

import java.util.ArrayList;
import java.util.List;

public  class Kids extends ManKind {
    private int yearsOld;
    public void printAge(){
        System.out.println("年龄是"+yearsOld);
    }
    public static void ss(){

    }
  public void ww(){

  }
    public static void main(String[] args) {
        ManKind someKid=new Kids();
        List arrays=new ArrayList<String>();
        someKid.setSex(1);
        someKid.setSalary(0);
        someKid.employeed();
        someKid.manOrWomen();
    }


    @Override
    public  void manOrWomen() {
        System.out.println("这个人的性别是未知的");
    }
}
class A{

}
```

4.2 Object 类[万物皆对象]

➢Object类是所有Java类的根父类

➢如果在类的声明中未使用extends关键字指明其父类，则默认父类为Object类

4.3 final关键字

final标记的类不能被继承。提高安全性，提高程序的可读性。

String类、System类、StringBuffer类

final标记的方法不能被子类重写。

Object类中的getClass()。

final标记的变量(成员变量或局部变量)即称为常量。名称大写，且只能被赋值一次。

final标记的成员变量必须在声明的同时或在每个构造方法中或代码块中显式赋值，然后才能使用

4.4 抽象类和接口

➢随着继承层次中一个个新子类的定义，类变得越来越具体，而父类则更一般，更通用。类的设计应该保证父类和子类能够共享特征。有时将一个父类设计得非常抽象，以至于它没有具体的实例，这样的类叫做抽象类

➢接口(interface)是抽象方法和常量值的定义的集合。从本质上讲，接口是一种特殊的抽象类，这种抽象类中只包含常量和方法的定义，而没有变量和方法的实现

No.	区别点	抽象类	接口
1	定义	包含一个抽象方法的类	抽象方法和全局常量的集合
2	组成	构造方法、抽象方法、普通方法、常量、变量	常量、抽象方法
3	使用	子类继承抽象类(extends)	子类实现接口(implements)
4	关系	抽象类可以实现多个接口	接口不能继承抽象类，但允许继承多个接口
5	常见设计模式	模板设计	工厂设计、代理设计
6	对象	都通过对象的多态性产生实例化对象	
7	局限	抽象类有单继承的局限	接口没有此局限
8	实际	作为一个模板	是作为一个标准或是表示一种能力
9	选择	如果抽象类和接口都可以使用的话，优先使用接口，因为避免单继承的局限	
10	特殊	一个抽象类中可以包含多个接口，一个接口中可以包含多个抽象类	

示例代码

```
1 package com.neuedu.demo.abstractjava;
2 /*抽象类*/
```

```java
public abstract class Animal {
    //属性
    public int age;
    public String color;
    //无参构造器
    public Animal(){
    }

    //有参数构造器
    public Animal(int age,String color){
        this.age=age;
        this.color=color;
    }
    //抽象方法
    public abstract void whoAmI();
}
```

```java
package com.neuedu.demo.abstractjava;

public class Cat extends Animal implements IAnimal,IAnimal2{
     public Cat(){

     }
     public  Cat(int age,String name){
         super(age,name);
     }
    @Override
    public void speak(){
        System.out.println("喵喵喵");
    }

    @Override
    public void eat() {
        System.out.println("吃鱼");
    }

    @Override
    public void whoAmI() {
        System.out.println("我是小猫，年龄是"+age+"颜色是"+color);
```

```
    }
    class A{public  void s(){}}
}

```

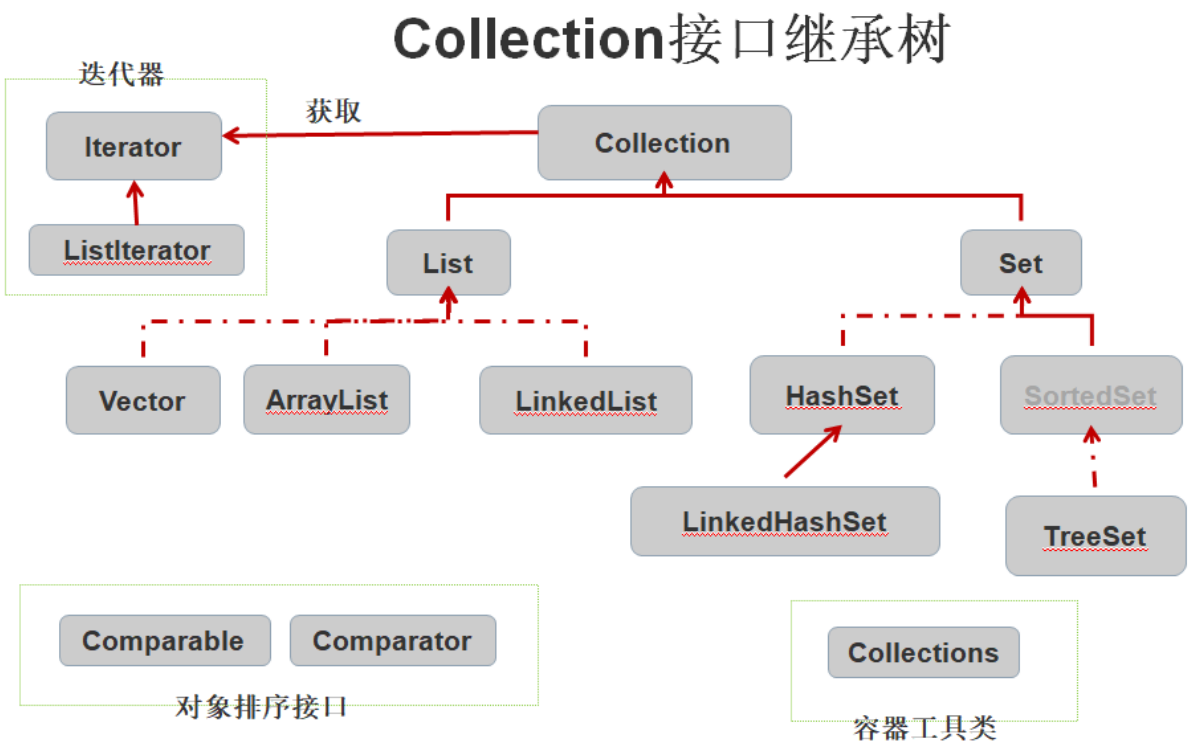
```
package com.neuedu.demo.abstractjava;

public class Dog  extends Animal implements IAnimal{
    @Override
    public void speak(){
        System.out.println("汪汪汪");
    }

    @Override
    public void eat() {
        System.out.println("吃骨头");
    }

    @Override
    public void whoAmI() {
        System.out.println("我是小狗年龄是"+age+"颜色是"+color);
    }
}
```

```
package com.neuedu.demo.abstractjava;

public class AnimalTest {
    public static void main(String[] args) {
        Animal cat=new Cat(8,"白色");
        cat.whoAmI();
        }

}
```

```
package com.neuedu.demo.abstractjava;
/*接口*/
public interface IAnimal {
```

```
    public static final String c="1231";
    void speak();

    public abstract void eat();
}
```

5.Java 集合概述



5.1 使用频率最高的实现类之ArrayList

➢ArrayList 是 List 接口的典型实现类

➢本质上，ArrayList是对象引用的一个变长数组

➢ArrayList 是线程不安全的，而 Vector 是线程安全的，即使为保证 List 集合线程安全，也不推荐使用Vector

5.2 使用频率最高的实现类之LinkedList

➢对于频繁的插入或删除元素的操作，建议使用LinkedList类，效率较高

示例代码

```
package com.neuedu.demo.listjava;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
/*List 基本操作 添加 清除 和遍历*/
```

```java
public class ListDemo {
    public static void main(String[] args) {
        ListDemo listDemo=new ListDemo();
        listDemo.testList();
    }

    public void testList(){
        List list=new ArrayList();
        //将指定的元素追加到此列表的末尾
        list.add("张三");
        //从列表中删除所有元素。
        list.clear();
        list.add(111);
        list.add(false);
        List listNew=new ArrayList();
        listNew.add("李四");
        //将指定集合中的所有元素追加到此列表的末尾。
        list.addAll(listNew);
        //遍历
        Iterator i = list.iterator();
        System.out.println("下面是迭代器遍历");
        while(i.hasNext()){
            System.out.println(i.next());
        }
        //for 循环遍历
        System.out.println("下面是for 循环遍历");
        for(int j=0;j<list.size();j++){
            System.out.println(list.get(j));
        }
        //增强for 循环遍历
        System.out.println("下面增强for 循环遍历");
        for(Object o:list){

            System.out.println(o);
        }
        //下面是forEach遍历
        System.out.println("下面forEach遍历");
        list.forEach(Object->{
            System.out.println(Object);
        });

    }
}
```

```
50
```

➢泛型：规范存贮类型，降低程序报错风险 注意：泛型擦除问题

```
1  package com.neuedu.demo.listjava;
2
3  import java.util.ArrayList;
4  import java.util.Arrays;
5  import java.util.Date;
6  import java.util.List;
7  /*
8  * 与普通的 Object 代替一切类型这样简单粗暴而言，泛型使得数据的类别可以像参数一样由外
9  当具体的类型确定后，泛型又提供了一种类型检测的机制，只有相匹配的数据才能正常的赋值，
10 泛型提高了程序代码的可读性，不必要等到运行的时候才去强制转换，在定义或者实例化阶段，
11 *
12 泛型是 Java 1.5 版本才引进的概念，在这之前是没有泛型的概念的，但显然，泛型代码能够很
13 这是因为，泛型信息只存在于代码编译阶段，在进入 JVM 之前，与泛型相关的信息会被擦除掉，
14 * */
15 public class ListDemoThe {
16     public static void main(String[] args) {
17         ListDemoThe list=new ListDemoThe();
18      //   list.listDeamo2();
19         list.arrToList();
20
21         List<String> l1 = new ArrayList<String>();
22         List<Integer> l2 = new ArrayList<Integer>();
23
24        // System.out.println(l1.getClass() == l2.getClass());
25     }
26     public void listDemo(){
27         List list=new ArrayList<>();
28         list.add(1);
29         list.add(2);
30         list.add(false);
31         int a = 0;
32         for(int i=0;i<list.size();i++){
33             a=a+ (int) list.get(i);
34         }
35     }
36     public void listDeamo2(){
37         List<Integer> list = new ArrayList<Integer>();
38         list.add(1);
```

```
        System.out.println(list);
    }

    public void arrToList(){
        int[] a=new int[]{1,2,3};
        List list= Arrays.asList(a);
        System.out.println(list);
    }

}

```

5.3 HashMap

➢Map接口的常用实现类：HashMap、TreeMap和Properties。

➢HashMap是 Map 接口使用频率最高的实现类。

➢允许使用null键和null值，与HashSet一样，不保证映射的顺序。

➢HashMap 判断两个 key 相等的标准是：两个 key 通过 equals() 方法返回 true，hashCode 值也相等。

➢HashMap 判断两个 value相等的标准是：两个 value 通过 equals() 方法返回 true。

```
package com.neuedu.demo.mapJava;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class HashMapDemo {
    public static void main(String[] args) {
        HashMapDemo hashMapDemo=new HashMapDemo();
        hashMapDemo.test01();
    }
    public void test01(){
        Map map=new HashMap();
        map.put("name","张三");
        map.put("age",12);
        map.put("isStudent",false);

        //迭代器方式遍历
```

```
        Iterator iterator=map.entrySet().iterator();
        System.out.println("我是迭代器遍历");
        while(iterator.hasNext()) {
            Map.Entry  e= (Map.Entry) iterator.next();
            System.out.println(e.getKey());
            System.out.println(e.getValue());
        }
        //增强for循环遍历
        System.out.println("我是增强for循环遍历");
        for(Object o:map.keySet()){
            System.out.println(o);
            System.out.println(map.get(o));
        }
        //Java8中新lambda方法
        System.out.println("我是Lambda遍历");
        map.forEach((key,value)->{
            System.out.println(key);
            System.out.println(value);
        });
        //判断map 是不是空
        Map map1=new HashMap();
        if(map.size()>0){
            System.out.println("map不是空");
        }
       if(map1.size()<=0){
           System.out.println("map1是空");
       }

        System.out.println("map判断是不是空"+map.isEmpty());
        System.out.println("map1判断是不是空"+map1.isEmpty());
    }
}

```

➢对比HashSet 、LinkedHashSet、ArrayList

```
package com.neuedu.demo.setJava;

import java.util.*;
/*Set集合示例*/
public class HashSetDemo {
    public static void main(String[] args) {
```

```
        HashSetDemo hashSetDemo=new HashSetDemo();
        hashSetDemo.test1();
    }
    //HashSet 特点 1 无序 2 元素不可重复 3 可以存null
     public  void test1(){
         Set set=new HashSet();
         set.add(1);
         set.add(1);
         set.add(0);
         set.add(null);
         System.out.println(set);
     // LinkedHashSet 特点 1 有序 2 元素不可重复 3 可以存null
         Set linkHashSet=new LinkedHashSet();
         linkHashSet.add(1);
         linkHashSet.add(1);
         linkHashSet.add(0);
         linkHashSet.add(null);
         System.out.println(linkHashSet);
     // ArrayList 特点 1 有序 2 元素可以重复
         List list=new ArrayList();
         list.add(1);
         list.add(1);
         list.add(0);

         System.out.println(list);
     }
}

```

5.4 枚举

➢在数学和计算机科学理论中，一个集的枚举是列出某些有穷序列集的所有成员的程序，或者是一种特定类型对象的计数。这两种类型经常（但不总是）重叠。 [1] 是一个被命名的整型常数的集合，枚举在日常生活中很常见，例如表示星期的SUNDAY、MONDAY、TUESDAY、WEDNESDAY、THURSDAY、FRIDAY、SATURDAY就是一个枚举。

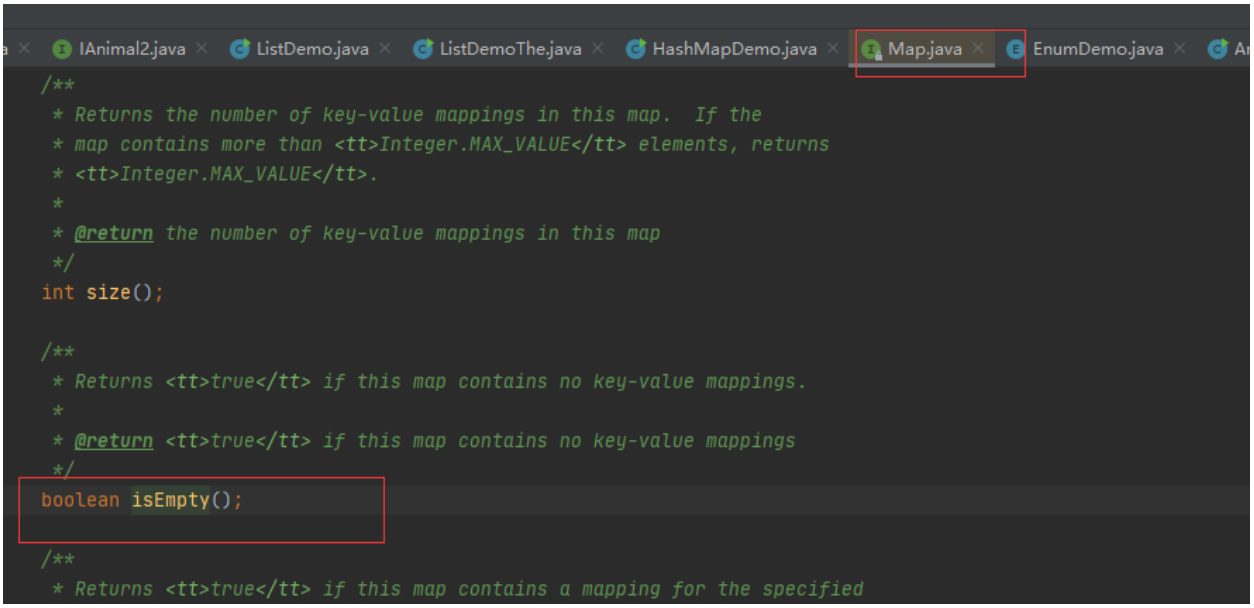
```
package com.neuedu.demo.enumJava;

import com.neuedu.demo.extendsjava.ManKind;

/*枚举*/
```

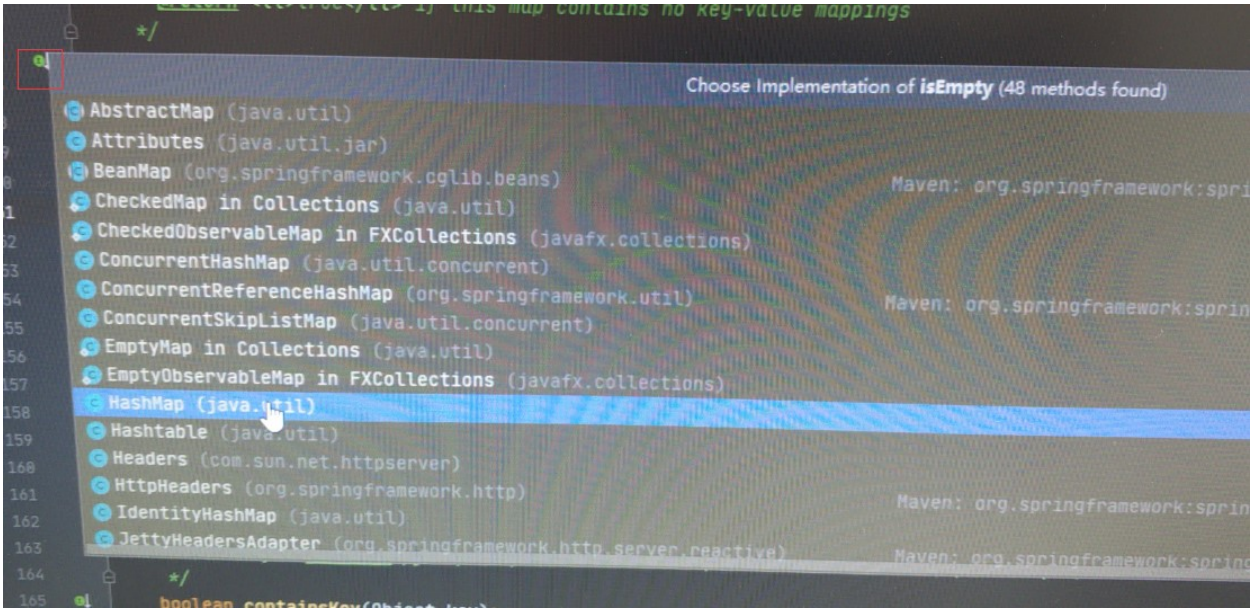
```java
public enum EnumDemo {
    //注意逗号和分号的位置
    COLOR_TYPE_RED(1, "红色", false),
    COLOR_TYPE_BLACK(2, "黑色", true),
    COLOR_TYPE_BLUE(3, "蓝色", true);
    //属性
    int code;
    boolean like;
    String message;

    //get 方法 枚举定义的是常量 所以不需要set 方法
    public int getCode() {
        return code;
    }

    public boolean isLike() {
        return like;
    }

    public String getMessage() {
        return message;
    }

    //构造器 用于上面定义枚举常量
    EnumDemo(int code, String message, boolean like) {
        this.code = code;
        this.message = message;
        this.like = like;
    }

    //遍历 根据code 值 置换 message
    public static String getMessage(int code) {
        String message = null;
        //利用.values()方法 获取枚举数组 方便遍历
        for (EnumDemo enumDemo : EnumDemo.values()) {
            //如果外部形参值和枚举里某个属性的code相等 直接返回message
            if (code == enumDemo.getCode()) {
                message = enumDemo.getMessage();
                return message;
            }
        }
        return message;
    }
```

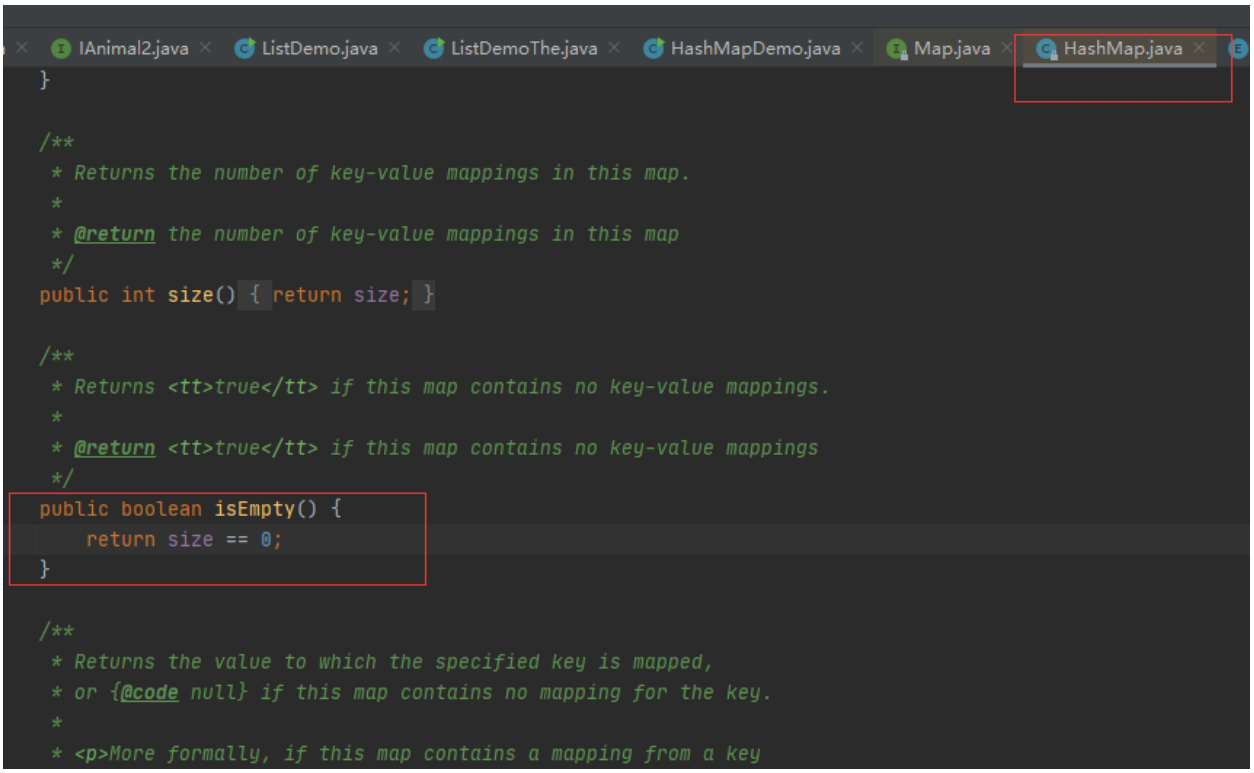
```
49
50 }
51
52 class Test {
53     public static void main(String[] args) {
54         int code = EnumDemo.COLOR_TYPE_BLACK.code;
55        String a= EnumDemo.COLOR_TYPE_RED.message;
56         String message = EnumDemo.getMessage(code);
57         System.out.println(message);
58
59     }
60 }
61
```

5.5 源码分析isEmpty()

通过map接口找到实现类







结论： JDK底层也是通过大小来判断是不是空的 引申自定义判断方法如下

```
//判断map 是不是空
        Map map=new HashMap();
        map.put("name","张三");
        Map map1=new HashMap();
        if(map.size()>0){
            System.out.println("map不是空");
        }
       if(map1.size()<=0){
           System.out.println("map1是空");
       }
```