Java 基础知识点复习资料

1、 classpath 的设置及作用：设置为 set classpath= “绝对路径”。作用是指定 Java 类的执行路径。是用来 寻找 class 文件的。

2、 path 的设置及作用：是内部命令。在编译 Java 的时候需要用到 javac 命令在执行 java 的时候要用到 java 命令，这两个命令不时 windows 自带的命令，所以我们使用的时候要设置好环境变量，这样就可以在任何目录下使用了。

3、 JAVA 数据类型：基本数据类型（数值型（整型（ byte.short.int.long ）、浮点型（ float.double ））、字符型 (char) 、布尔型 (boolean) ）、引用数据类型 (class. 数组、接口) 。

4、 JAVA 基本运算符号：赋值运算符、算术运算符、关系运算符、逻辑运算符、条件运算符、括号运算符。

5、 JAVA 程序基本控制语句：

1 、 if (判断条件)

{

语句 1 ;

语句 2 ;

}

2 、 if （判断条件）

{

语句 1 ;

}

else

{

语句 2 ;

}

3 、 switch （表达式）

{

case 选择值 1；语句主体 1；

break；

case 选择值 2；语句主体 2；

break；

case 选择值 3；语句主体 3；

break；

default：语句主体；

}

4、 for（赋初始值；判断条件；赋值增减量）

{

语句；

}

5、 while（判断条件）

{

语句 1；

语句 2；

}

6、 do

{

语句 1；

语句 2；

}

while（判断条件）；

break 跳出循环。 continue 跳出本次循环进入到下次循环。

6、 数组的概念及使用：数组是由一组相同类型的变量所组成的数据类型，它们以一个共同的名称来表示。

格式一：数据类型 数组名 []

数组名 =new 数据类型 [个数]

例： int i[]

i=new int[8]

格式二：数据类型 数组名 []=new 数据类型 [个数]

例： int i=new int[8]

7、 方法的声明及使用：完成某一特定功能的代码块。

返回值类型 方法名称（类型 参数 1，类型 参数 2 ）

{

程序语句；

return 表达式；

}

8、 方法的重载： 在同一个类中允许同时存在一个以上的同名方法，只要他们的数据类型参数个数不同即可。

例如： public static int Test(int i,intj)

{

return i+j;

p ublic static int Test(int x)

{

return x++;

}

pubic static double Tset(double m,double n)

{

return m+n;

}

public static int Test(int a,int b,int c)

{

return a-b+c;

}

}

在程序中 Test 方法被重载了 4 次，但每个重载了的方法所能接受参数的个数和类型不同。

9 、类的定义：

将具有相同属性及相同行为的一组对象称为类。广义的讲，具有共同性质的事物的集合就称为类。

class 类名称

{

数据类型 属性； // 声明成员变量（属性）

返回值的数据类型 方法名称（参数 1 ，参数 2 ）

{

程序语句；

return 表达式； // 定义方法的内容

}

}

例： Person.java

C lass Person

{

String name;

int age;

void talk()

{

System.out.println(" 我是： " +name+ " , 今年： " +age+ " 岁 ");

}

}

10 、对象的产生与使用：对象产生的基本形式 类名 对象名 =new 类名（） 对象可以调用类中的方法访问类中的成员变量，形势为 对象 . 属性 对象 . 方法名 ()

11 、匿名对象：没有明确的声明对象，也可以理解为只使用一次的对象，即没有任何一个具体的对象名称引用它。

12. 实现类的封装性

封装了相关的成员和方法，通过访问权限来使用内部的东西。

13. 构造方法的定义与使用

构造方法的定义方式：

```
class 类名称
{
 访问权限 类名称（类型 1 参数 1 ，类型 2 参数 2 ）
 {
   语句;
 }
}
```

注意： 1. 构造方法的名称必须和类名一致。

2. 构造方法无返回值。

例： class pig

```
{
  public pig()
  {
    System.out.println( “ It ’ s a pig. ” );
  }
}
```

14. 对象的比较（“ == ”、“ equals() ”

“ == ”比较的是对象所指的引用

“ equals() ”比较的是对象 。

15. 在类内部调用本类方法

范例： TestJavaThis1.java

```
class Person
{
```

```
String name;
int age;
public Person()
{
 System.out.println( " 1.public Person() " );
}
public Person(String name,int age)
{
 // 调用本类中无参构造方法
 this();
```

16.this 关键字的使用

1. 用来区分成员变量和局部变量 .

2. 用来表示本类内部的构造方法 .(this 必须放在构造方法中的第一行)

范例 :TestJavaThis.java

```
class Person
{
 private String name;
 private int age;
 public Person(String name, int age)
 {
 this.name = name;
 this.age = age;

 }
 public String talk()
 {
 return " 我是 : " +name+ " , 今年 : " +age+ " 岁 " ;
```

```
 }
}
public class TestJavaThis
{
 p ublic  static void main(String[] args)
 {
 Person p = new Person( “ 张三 ” ,25);
 System.out.println(p.talk());
 }
}
```

输出结果：

我是：张三，今年： 25 岁

17. 构造方法的重载

1. 方法名字相同
2. 参数的类型、排列方式、个数不同

例如： public static int Test(int i,intj)

```
{
 return i+j;
 p ublic  static int Test(int x)
{
 return x++;
}
pubic static double Tset(double m,double n)
{
 return m+n;
}
```

```
public static int Test(int a,int b,int c)
{
 return a-b+c;
}
}
```

在程序中 Test 方法被重载了 4 次，但每个重载了的方法所能接受参数的个数和类型不同。

18. 对象的引用传递

例：

S 1= ” abc ”

S 2= ” bcd ”

S 1=S2

把 S2 所指的引用 ” bcd ” 赋给了 S1 ， 这时 S1 的引用就是 ” bcd ” 。

19.static 的使用（方法、属性）

1、 static ：静态方法： 1 直接用类名来调用 2 一个静态方法可以直接调用另一静态方法调用非静态方法时用对象调用 3 静态方法中不能用 this 、 super4 静态方法不能调用非静态方法中的属性。 S tatic 声明的成员变量为全局变量局部变量不能声明成 static 。静态代码块是在 static 后加个大括号作用是给静态成员赋值。

20. 对象数组的使用（静态初始化、动态初始化）

1 数组是多个相同类型数据的集合，实现对这些数据的统一管理。

2 数组是引用数据类型，数组型数据是对象 (object) ，数组 的每个元素相当于该对象的成员变量。

3 数组中的元素可以是任何数据类型，包括基本类型和引用类型。

一维数组声明

一维数组的声明方式：

T ype var[] 或 type[] var;

例如：

int a[]; int[] a; double b[];

注！ Java 语言中声明数组时不能指定其长度（数组中元素的个数），例如：

int a[]5; // 错误的

正确的写法： int[] a; (数据类型 数组名 [] ； // 声明一维数组

a = new int[5] ; 数组名 = new 数据类型 [个数] ） // 分配内存给数组

创建基本一维数组

```
public class Test
{
 p ublic  static void main(String[] args)
{
 int[] s;
 s = new int[10];
for(int i=0;i<10;i++)
{
  s[ i ] =2*i+1;
 System.out.println(s[i]);
}
}
}
```

编译结果：

```
E:\>java Test
1
3
5
7
9
11
13
15
17
```

19

21. 类的继承

通过继承可以简化类的定义，扩展类的功能。

实现继承的方法： class 子类 extends 父类。

例：

```
 C lass A{
 String  name;
 i nt age;
}
 C lass B extends A{
String school ；
}
 public class Test
{
 public static void main(String args[]){
 B s=new B( );
 s.name= ” 周洋 ” ;
 s.age= ” 23 ” ;
 s.school= ” 绥化学院 ” ；
 System.out.println( “ 姓名： ” +s.name +”, 年龄： ” +s.age+ ” , 学校： ” +s.school);
}
}
```

22. 子类对象的实例化过程

子类对象在实例化时会默认先去调用父类中的无参构造方法，之后再调用子类中的相应构造方法。依次执行

范例： TestPersonStudentDemol.java

```
class Person
{
```

```
String name;
 I nt age;
// 父类的构造方法
public Person()
{
System.out.println( " 1.public Person(){} " );

}
}
class Student extends Person
{
String school;
// 子类的构造方法
 P ublic Student()
{

System.out.println( " 2.public Student(){} " );

}
}
public class TestPersonStudentDemol
{
public static void main(String[] args)
{

Student s = new Student();

}
```

}

输出结果：

1.public Person(){}

2.public Student(){}

23．方法的覆写

当一个子类继承一个父类，而子类中的方法与父类中的方法的名称，参数个数、类型完全一致时，就称子类中的这个方法覆写了父类中的方法。

特点：1、两个类存在继承关系；2、子类里写了一个方法，方法的声明与父类一致。

例如：

```
class Person{
 public void test(){
  System.out.println(“Person”);
}
}
class Student extends Person{
 public void test(){
  System.out.println(“Student”);
}
}
```

24． super 的使用

super 关键字出现在子类中，用 super 去调用父类中的有参的构造方法，所以 super 主要功能是完成子类调用父类的内容，也就是调用父类中的构造方法。

注意：用 super 调用父类中的构造方法，只能放在第一行。

格式：

super. 父类中的属性；

super. 父类中的方法；

例如：

（1） class Person

```
{
 String name ;
 int age;
 public Person()
 {
 }
 public String talk()
 {
  return" 我是： "+this.name", 今年； "+this.age+" 岁 ";
 }
}
class Student extends Person
{
 String school;
 public Student(String name,int age,String school)
 {
  super.name = name;
  super.age = age;
  System.out.println(super.talk());
  this.school = school;
 }
}
public class Test
{
 public static void main(String args[])
 {
  Student s = new Student(" 张三 "， 23， " 北京 ");
```

```
  System.out.println(" ， 学校 :"+s.school);
 }
}
```

（ 2 ） public class Kiss1

```
{
 String name;
 int age;
 Kiss1()
 {
  System.out.println(“ aaaaaaaaaaaaa ”);
 }
 Kiss1(String name,int age)
 {

  this.name = name;
  this.age = age;
  System.out.println(name+age);
 }

}
class Kiss2 extends Kiss1
{
 public Kiss2()
 {
 super("aaaaaaa",23);
 }
 public static void main(String[] args)
```

```
{
  Kiss2 p = new Kiss2();
}


}
```

25． this 与 super 的区别

this 关键字的作用： 1. 用来区分成员变量和局部变量 .

2. 用来表示本类内部的构造方法 .(this 必须放在构造方法中的第一行)

super 关键字出现在子类中，用 super 去调用父类中的有参的构造方法，所以 super 主要功能是完成子类调用父类的内容，也就是调用父类中的构造方法。用 super 调用父类中的构造方法，只能放在方法体中的第一行。

26． final 关键字

final 声明的变量就变成了常量，今后不可以修改。只能声明时初始化或者在构造方法中初始化。

final 声明的方法不能被子类复写。

final 声明的类不能被继承。

27．抽象类的使用

抽象类： java 可以创建一种类专门用来当作父类，这种类称为“抽象类”

抽象类的定义规则：

1、 抽象类和抽象方法都必须用 abstract 关键字类修饰。

2、 抽象类不能被直接实例化，也就是不能直接用 new 关键字去产生对象。

3、 抽象方法只需声明，而不需实现。

4、 含有抽象方法的类必须被声明为抽象类，抽象类的子类必须覆写所有的抽象方法后才能被实例化，否则这个子类还是个抽象类。

抽象类的定义方法：

```
Abstract class 类名称
{
// 定义抽象类
```

声明数据成员：

访问权限　返回值的数据类型　方法名称（参数）

{

？

}

Abstract 返回值的数据类型　方法名称（参数）；

// 定义抽象方法，在抽象方法里，没有定义方法体

}

28．接口的使用

接口（ interface ）是 java 所提供的另一种重要技术，它的结构和抽象类非常相似，也具有数据成员与抽象方法，但它与抽象类又有以下两点不同：

1．　接口里的数据成员必须初始化，且数据成员均为常量。

2．　接口里的方法必须全部声明为 abstract ，也就是说，接口不能象抽象类一样拥有一般的方法，必须全部都是“抽象方法”。

接口定义的语法如下：

interface　接口名称　// 定义抽象类

{

final 数据类型　成员名称 = 常量；　// 数据成员必须赋初值

abstract 返回值的数据类型　方法名称（参数 ... ）；

// 抽象方法，注意在抽象方法里，没有定义方法主体

}

在 java 中接口是用于实现多继承的一种机制，也是 java 设计中最重要的一个环节，每一个由接口实现的类必须在类内部覆写接口中的抽象方法，且可自由地使用接口中的常量。

既然接口里只有抽象方法，它只要声明而不用定义处理方式，于是自然可以联想到接口也没有办法象一般类一样，再用它来创建对象。利用接口打造新的类的过程，称之为接口的实现（ implementation ）。

接口的实现：

格式：

class 类名称 implements 接口 A，接口 B

{

……

}

29．对象的多态性、向上转型及向下转型的概念

Java 语言中有（方法重载）和（成员覆写）两种形式的多态。

overload	override
在一个类中	在具有继承关系的两个类中
函数名相同	函数的定义完全相同
参数列表不同	

多态其实就是表现在具有相同的代码，但是表现出来的内容却不同

向上转型：把子类类型对象的引用转化成父类类型对象的引用。

向下转型：把父类类型对象的引用转化成子类类型对象的引用。

30．抽象类及接口的应用

接口是为了实现 Java 的多继承。

31．异常的概念、分类及使用

异常：异常实际上是程序中错误导致中断了正常的指令流的一种事件。

异常的分类：异常是一个对象，它继承自 Throwable 类，所有的 Throwable 类的子孙类所产生的对象都是例外。

Error: 由 Java 虚拟机生成并抛出， Java 程序不做处理。

Runtime Exception ：由系统检测，用户的 Java 程序可不做处理，系统将它们交给缺省的异常处理程序。

Exception ： Java 编译器要求 Java 程序必须捕获或声明所有的非运行时异常。

throw ：用户自己产生异常。

32． Java 程序的编写规范

命名规范，常量用大写，类的声明顶格写，一个文件尽量只包含一个类。

33． try ... catch 语句的使用

用 try 来捕捉程序中是否会抛出异常，用 catch 来处理异常。

34 .throws 和 throw:

throws 是在方法后抛出一个可能有异常的声明，而 throw 是在方法内部故意抛出的一个异常。

35 .finally 关键字 :

放在 try(){ ..... }catch(){ ..... } 后面，用法是无论是否会抛出异常，都会执行 finally 主体中的语句。

39 .List,ArrayList 的使用

list 是用存放引用数据类型的数据的接口， list 存放的数据是有序的，而且可以重复。

ArrayList 是实现 list 接口的类。所以他也同样具有 list 中的特性。

ArrayList 中添加元素用的方法是 add(), 获取元素的方法是 get(), 并且是用循环输出 .

例： Example1.java

```
import java.util.*;
import java.text.*;
public class Example1
{
 public static void main(String[] arg)
 {

ArrayList p=new ArrayList();
p.add(“a”);
p.add(“b”);
p.add(“c”);
p.add(“b”);
p.add(“d”);
```

```
    for(int i=0;i<p.size();i++)
    System.out.println(p.get(i));
}
}
```

输出结果： a

b

c

b

d

此例证明了 ArrayList 的有序和重复性。

40 .Map.ListMap 的使用 .

M ap 是双值的，一个位置存两个东西。

Map 包括键和值两部分 .

键 : 只能存引用数据类型，无序的，不能重复。

值：只能存引用数据类型，允许重复。

put() 方法用来放此映射中关联指定键也指定值。

get() 方法用来取键在此映射中关联的值。

```
import java.util.*;
public Test
{
 public static void main(String[] arg)
 {
 HashMap hm=new HashMap();
 hm.put( " a " , " b " );
 hm.put( " c " , " d " );
 System.out.println( hm.get( " a " ));
}
```

}

输出结果为： b

41 .Set.HashSet 的使用

Set 也是用来存放引用数据类型的接口，存放的数据无序的，不能重复。

HashSet 是实现 Set 接口的一个类，它也同样具有 Set 的性质。

add() 方法是用来添加数据元素到相应的 HashSet 表里的方法，

输出的时候用迭代器，再用 hasNext() 方法判断是否有下一个元素，

再用 next() 方法输出下一个元素。

例题：

```
import java.util.*;
public class A
{
 public static void main(String[] arg)
{
 HashSet hs=new HashSet();
 hs.add( " a " );
 hs.add( " b " );
 hs.add( " c " );
 Iterator it=hs. iterator();
 while(it.hasNext())
 {
 System.out.println(it.next());
}
}
}
```

输出结果为 b

c

a

42 .Iterator 的使用：

返回在此 set 中的元素上进行迭代的迭代器。返回的元素没有特定的顺序， 返回值类型为 Iterator 。

用 set 类的实例化对象调 iterator() ，再赋给一个 Iterator 类型的变量，再用 while 循环判断是否有下一个迭代，如果有则输出下一个迭代。

43 .String 和 StringBuffer 的使用：

String 类用于比较两个字符串、查找和抽取串中的字符或子串、字符串与其它类型之间的相互转换等。 String 类对象的内容一旦被初始化就不能再改变。

StringBuffer 类用于内容可以改变的字符串，可以将其它各种类型的数据增加、插入到字符串中，也可以转置字符串中原来的内容。一旦通过 StringBuffer 生成了最终想要的字符串，就应该使用 StringBuffer.toString() 方法将其转换成 String 类，随后，就可以使用 String 类的各种方法操纵这个字符串了。

String x = “ a ” +4+ ” c ” ;

编译时等效于

String x = new StringBuffer().append(“ a ”).append(4).append(“ c ”).toString();

在实际的开发中，如果需要频繁改变字符串的内容就需要考虑用 StringBuffer 类实现 , 因为其

内容可以改变 , 所以执行性能会比 String 类更高 .

44 .Date 和 Calendar 的使用 :

类 Date 表示 日期和时间 特定的瞬间，精确到毫秒 .

```
import java.util.*;
public class A
{
 public static void maimnn(String[] arg)
 {
   Date  d= new Date ();
  System.out.println(d.toString());
 }
}
```

输出结果为 :Thu Aug 30 14:22:41 CST 2007

Calendar 是一个抽象类，主要完成字段之间相互操作的功能。

```
import java.util.*;
public class A
{
 public static void maimnn(String[] arg)
 {
   Calendar d=Calendar.getInstance();
   System.out.println(d.toString());
 }
}
```

输出结果为：

java.util.GregorianCalendar[time=11,areFieldsSet=true,areAllFieldsSet

=true,lenient=true,zone=sun.util.calendar.ZoneInfo[id="Asia/Shanghai",offset=288

00000,dstSavings=0,useDaylight=false,transitions=19,lastRule=null],firstDayOfWee

k=1,minimalDaysInFirstWeek=1,ERA=1,YEAR=2007,MONTH=7,WEEK_OF_YEAR=35,
WEEK_OF_MON

TH=5,DAY_OF_MONTH=31,DAY_OF_YEAR=243,DAY_OF_WEEK=6,DAY_OF_WEEK_IN
_MONTH=5,AM_PM=

0,HOUR=6,HOUR_OF_DAY=6,MINUTE=33,SECOND=11,MILLISECOND=921,ZONE_OF
FSET=28800000,

DST_OFFSET=0]