1、Java 是一种简单的、面向对象的、分布式的、强壮的、安全的、体系结构中立的、可移植的、高性能的、多线程的和动态的语言。

2、 编译器——javac

javac 的作用是将源程序（.java 文件）编译成字节码（.class 文件）。Java 源程序的后缀名必须是 java。javac 一次可以编译一个或多个源程序，对于源程序中定义的每个类，都会生成一个单独的类文件。例如，如果在 A.java 中定义了 A、B、C 三个类，则经过 javac 编译后要生成 A.class，B.class，C.class 三个类文件。

- javac 的调用格式为:

  javac［选项］ 源文件名表

- 其中源文件名表是多个带.java 后缀的源文件名

3、Java 的语言解释器——java

- java 命令解释执行 Java 字节码。其格式为：

  java［选项］类名〈参数表〉

这里的类名代表由编译器生成的带.class 后缀的类文件名，但在上述命令不需要带后缀。这个类必须是一个独立程序（不能是 Applet），程序中必须带有一个按如下格式声明的 main 方法。

```
public static void main(String [ ] args ) {…}
```

- 并且包含 main 方法的类的类名必须与类文件名相同，即与现在命令行中的“类名”相同。
- 在执行 java 命令时，若类名后带有参数表，则参数表中的参数依次直接传递给该类中的 main 方法的 args 数组，这样在 main 方法中就可以使用这些数组元素。

解释运行的三个阶段：

(1)载入：Java 解释器中的类载入器将字节码文件加载到内存(网上运行程序则通过网络下载到本地内存)；

(2)代码校验：Java 解释器中的代码检验器检查这些字节码的合法性；

(3)解释执行：合法的字节码程序由 Java 解释器逐句地解释运行。

4、Java 程序分两种：

- Java Application（Java 应用程序）：是一个完整的应用程序，可以独立地运行
  - 运行在 Java 虚拟机(JVM)上
  - 中间代码
  - 必须有 main 函数
- Java Applet（小应用程序）：不是一个完整的应用程序，而是框架程序中的一个模块，所以只能在 WWW 浏览器环境下运行
  - 可以没有 main 函数
  - 必须由某个支持 java 的浏览器来运行

5、利用编辑器编写 Java 源程序

  - 源文件名：主类名.java
- 利用编译器将源程序编译成字节码
  - 字节码文件名：源文件名.class
- 利用虚拟机（解释器）运行
  - 运行过程：载入、代码校验、解释执行

Application 程序

(1)Java 语言标识符的字母区分大小写；

(2)一个程序可以由一个或多个类组成，其中必须有也只能有一个主类。

(3)源文件名必须与程序的主类名一致，并且以 .java 为其后缀。

6、变量是用标识符命名的数据项，是程序运行过程中其值可以改变的量。

- Java 是强类型语言，这就意味着每一个变量都必须有一个数据类型。为了描述一个变量的类型和名字，必须用如下方式编写变量声明：

  类型　变量名;
- 使用变量之前必须先声明变量。

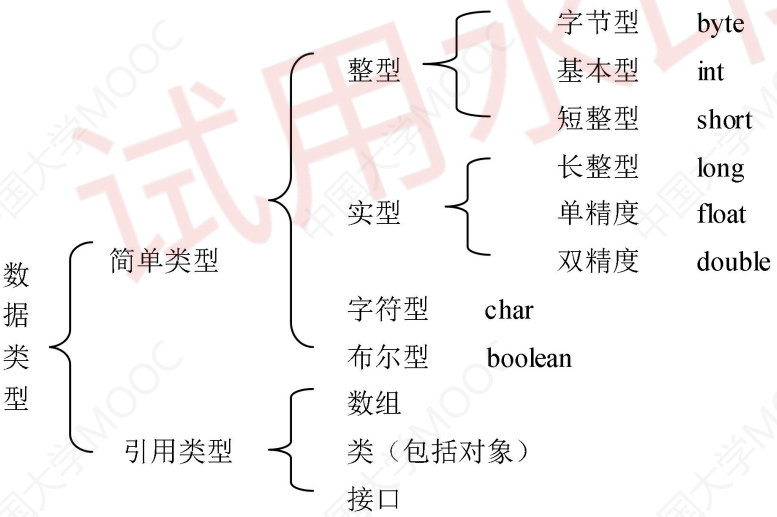
- ❖ 声明变量包括两项内容：变量名和变量的类型。通过变量名可使用变量包含的数据。变量的类型决定了它可以容纳什么类型的数值以及可以对它进行什么样的操作。
- ❖ 变量声明的位置，决定了该变量的作用域。

变量名应满足下面的要求：

- ➢ 必须是一个合法的标识符。
- ➢ 不能是一个关键字或者保留字（如 true、false 或者 null）。
- ➢ 在同一个作用域中必须是唯一的。

- ❖ Java 语言规定标识符由字母、下划线（_）、美元符（$）和数字组成，且第一个字符不能是数字。其中，字母包括：大、小写字母、汉字等。
- ❖ 一般约定：变量名是以小写字母开头。如果变量名包含了多个单词，则在每个单词的第一个字母大写，如：isVisible。下划线“_”可以用在常数中用它分离单词，因为常数名都是用大写字母，用下划线可以更清晰。

7、



Java 语言规范提供了两种数据类型：简单类型和引用类型。引用类型可使用一

个引用变量得到它的值或者得到由它所表示的值的集合，一个简单变量名是取该变量的真实值。

8、只有一个运算对象的运算符称为**一元运算符**。一元运算符支持前缀和后缀运算符。前缀运算符是指运算符出现在它的运算对象之前，例如：

operator op　　//前缀运算符

❖ 后缀运算符是指运算对象出现在运算符之前，例如：

op operator　　//后缀运算符

需要两个运算对象的运算符号称为二元运算符。比如赋值号（=）就是一个二元运算符。所有的二元运算符使用中缀运算符，即运算符出现在两个运算对象的中间：

op1 operator op2　　//中缀运算符

三元运算符需要三个运算对象。Java 语言有一个三元运算符“？：”，它是一个简要的 if-else 语句。三元运算符也是使用中缀运算符，例如： op1 ? op2 : op3　　//中缀运算符

9、

运算符	描述
? :	作用相当于 if-else 语句
[]	用于声明数组，创建数组以及访问数组元素
.	访问对象的成员变量和方法
(params)	以逗号分开的参数系列
(type)	将某一个值转换为 type 类型
new	创建一个新的对象或者新数组
instanceof	决定第一个运算对象是否为第二个运算对象的一个实例

运算符	用法	返回 true 的情况
&&	op1 && op2	op1 和 op2 都是 true
||	op1 || op2	op1 或者 op2 是 true
!	! op	op 为 false
^	op1 ^ op2	op1 和 op2 逻辑值不相同

10、**表达式**是由运算符、操作数和方法调用，按照语言的语法构造而成的符号序列。

❖ Java 语言的语句可分为以下几类：

- 表达式语句
- 复合语句
- 控制语句
- 包语句和引入语句

❖ 其中，表达式语句是用分号“;”终止表达式的语句，包括：

- 赋值表达式语句
- ++、--语句
- 方法调用语句
- 对象创建语句
- 变量的声明语句

11、**控制语句**用于改变程序执行的顺序。程序利用控制语句有条件地执行语句、循环地执行语句或者跳转到程序中的其他部分执行语句。

❖ Java 的控制语句有：

- if-else 语句
- switch 语句
- while 和 do-while 语句
- for 语句
- 跳转语句
- 异常处理语句

```
switch (expression) {
    case value1 :  {
         statements1;
         break;
    }
    ….
    case valueN :  {
        statementsN;
         break;
     }
    [default  :  {
        defaultStatements;
   }]
}
```

- 表达式 expression 的返回值类型必须是这几种类型之一：int、byte、char、short。
- case 子句中的值 valueI 必须是常量，而且所有 case 子句中的值应是不同的。
- default 子句是任选的。
- break 语句用来在执行完一个 case 分支后，使程序跳出 switch 语句，即终止 switch 语句的执行。如果某个 case 分支后没有 break 语句，程序将不再做比较而执行下一个分支。
- switch 语句的功能可以用 if-else 语句来实现，但某些情况下，使用 switch 语句更简炼。
- for 循环语句

```
for (initialization; termination; iteration)
{
```

```
    body;       //循环体
}
```

其中：

- initialization:初始化条件;
- termination :循环条件
- iteration ： 迭代，变更循环条件

初始化、终止以及迭代部分都可以为空语句(但分号不能省)，三者均为空的时候，相当于一个无限循环。

在初始化部分和迭代部分可以使用逗号语句，来进行多个操作。逗号语句是用逗号分隔的语句序列。

```
if (boolean-expression1)  {
    statements1;
 }
 else if(boolean-expression2)
 {
        statements2;
  } else  {
       …
          statementsN;
   }
```

- 布尔表达式 boolean-expression 是任意一个返回布尔数据类型的表达式，而且必须是.

- 每个单一语句后面都要有分号。为了增强程序的可读性，应将 if 或 else 后的语句用{ }括起来。
- else 子句是任选的，不能单独作为语句使用，它必须和 if 语句配对使用，并且总是与离它最近的 if 配对。

12、Java 语言有 3 种跳转语句：

- break 语句
- continue 语句
- return 语句

break 语句

其功能是从该语句所在的 switch 分支或循环中跳转出来，执行其后继语句。

break 语句的第二种使用情况就是跳出它所指定的块，并从紧跟该块后的第一条语句处执行。

continue 语句

- continue 语句用来结束本次循环，跳过循环体中下面尚未执行的语句，接着进行终止条件的判断，以决定是否继续循环。对于 for 语句，在进行终止条件的判断前，还要先执行迭代语句。它的格式为：

```
continue;
```

- 也可以用 continue 跳转到括号指明的外层循环中，这时的格式为

```
continue outerLable;
```

return 语句

- return 语句从当前方法中退出，返回到调用该方法的语句处，并从紧跟该语句的下一条语句继续程序的执行。返回语句有两种格式：

```
return expression;     //当方法需要返回某种类型数据时
return;                //当方法的返回类型为 void 时
```

- 单独一条 return 语句放在方法中间时，会产生“不可到达”编译错误,因为其后的语句将不会执行到。若真需要退出方法，可以通过将 return 语句嵌入某些语句（如 if-else）来使程序在未执行完方法中所有语句时退出。

13、数组的长度在数组创建的时候就已经确定。一旦创建以后，数组就有了固定

长度。如图所示，数组的长度为 10，第一个下标为 0，下标为 8 的元素为第 9 个元素。

数组元素就是数组中的一个成员，可以通过数组中的位置来访问它。

声明一个数组

- 声明数组时无需指明数组元素的个数，也不为数组元素分配内存空间
- 不能直接使用，必须经过初始化分配内存后才能使用

```
int[] anArray;
```

创建一个数组

- 用关键字 new 构成数组的创建表达式，可以指定数组的类型和数组元素的个数。元素个数可以是常量也可以是变量
- 基本类型数组的每个元素都是一个基本类型的变量；引用类型数组的每个元素都是对象的的引用

可使用 Java 的 new 运算符来创建一个数组

```
anArray = new int[10];
```

数组初始化程序

- 声明数组名时，给出了数组的初始值，程序便会利用数组初始值创建数组并对它的各个元素进行初始化

```
int    a[]={22, 33, 44, 55};
```

- 创建数组的时，如果没有指定初始值，数组便被赋予默认值初始值。
  - 基本类型数值数据，默认的初始值为 0；
  - boolean 类型数据，默认值为 false；
  - 引用类型元素的默认值为 null。
- 程序也可以在数组被构造之后改变数组元素值

```
boolean[] answers = { true, false, true, true, false };
```

访问数组元素

```
anArray[i] = i;
```

确定数组的大小

arrayname.length

由同类型的对象为数组元素组成的数组称为对象数组。数组可用于保存引用类型的多个对象。

Java 的二维数组实质上是一维数组的数组，如图所示。这个二维数组可用 arrayOfInts.length 代表其长度，该长度为一维数组的个数。arrayOfInts[i].length 表示第 i 行子数组的长度。

```
class Gauss
{   public static void main(String[] args)
     {   int[ ] ia = new int[101];
         for (int i = 0; i < ia.length; i++)
             ia[i] = i;
         int sum = 0;
         for (int i = 0; i < ia.length; i++)
             sum += ia[i];
         System.out.println(sum);
     }
}
public class Arrays
{   public static void main(String[] args)
    {  int[] a1 = { 1, 2, 3, 4, 5 };
       int[] a2;
       a2 = a1;
       for(int i = 0; i < a2.length; i++)   a2[i]++;
       for(int i = 0; i < a1.length; i++)
           System.out.println( "a1[" + i + "] = " + a1[i]);
    }
```

```
}
```

运行结果：

```
a1[0] = 2
a1[1] = 3
a1[2] = 4
a1[3] = 5
a1[4] = 6
```

14、二维数组的声明和构造

- int[ ][ ] myArray ;
  - myArray 可以存储一个指向 2 维整数数组的引用。其初始值为 null。
- int[ ][ ] myArray = new int[3][5] ;
  - 建立一个数组对象，把引用存储到 myArray。这个数组所有元素的初始值为零。
- int[ ][ ] myArray = { {8,1,2,2,9}, {1,9,4,0,3}, {0,3,0,0,7} };
  - 建立一个数组并为每一个元素赋值。

- 二维数组的长度

```
class UnevenExample2
{ public static void main( String[ ] arg )
   { int[ ][ ] uneven =
        { { 1, 9, 4 },
          { 0, 2},
          { 0, 1, 2, 3, 4 } };
     System.out.println("Length is: " +  uneven.length );
    }
}
```

运行结果：

Length is: 3

15、对象(Object) 有两个层次的概念:

- 现实生活中对象指的是客观世界的实体，它由状态(State)和行为(Behavior)构成 ；
- 程序中对象是现实世界对象的模型，是一组变量和相关方法的集合。变量（Variables）表示现实对象的状态，方法(Methods)表现现实世界对象的行为，这些变量和方法叫做这个对象的成员（Member）。

❖ 类（Class）

类是描述对象的“基本原型”，它定义一类对象所能拥有的数据和能完成的操作。在面向对象的程序设计中，类是程序的基本单元。

相似的对象可以归并到同一个类中去，就像传统语言中的变量与数据类型关系一样。

将变量和方法封装在一个类中，可以对成员变量进行隐藏，外部对类成员的访问都通过方法进行，能够保护类成员不被非法修改。

❖ 封装(Encapsulation)

封装把对象的所有组成部分组合在一起。封装定义程序如何引用对象的数据，实际上是用方法将类的数据隐藏起来，控制用户对类的修改和访问数据的程度。封装利于模块化和信息隐藏。

❖ 子类(Subclass)

子类是作为另一个类的扩充或修正而定义的一个类。

❖ 继承(Inheritance)

一个类从另一个类派生出来的过程叫继承。这个类叫子类(派生类)，而被继承的类叫该类的超类(父类)。继承的子类可利用父类中定义的方法和变量，就像它们属于子类本身一样。也可以改变继承来的方法和变量。

```
class Car
{   int color_number;
```

```
    int door_number;
    int speed;

    public void push_break() {
            …
     }
    public void add_oil() { …   }
}
class Trash_Car  extends  Car
{
    double amount;

    public void fill_trash()
    {
        …
    }
}
```

❖ 方法的覆盖(override)

在子类中重新定义父类中已有的方法。

```
class Car
{   int color_number;
    int door_number;
    int speed;

    public void push_break() {
            speed = 0;
```

```
    }
    public void add_oil() { … }
}
class Trash_Car  extends  Car
{
    double amount;

    public void fill_trash() { … }
    public void push_break()
    {
        speed = speed - 10;
    }
}
```

❖ 方法的重载(Overload)

在同一个类中至少有两个方法用同一个名字，但有不同的参数列表。

使用重载方法时，Java 编译器根据传递给这个参数的数目和类型确定正确的方法。

重载实现了对象的多态特性。

多态性的特点大大提高了程序的抽象程度和简洁性。

16、[类修饰符] class 类名称 [extends 父类名称][implements 接口名称列表]

```
 {

    变量定义及初始化;

    方法定义及方法体;
}
```

类修饰符是下列之一：

[public | abstract | final]

public　该关键字声明的类可以在其他的任何类中使用。默认时，该类只能被同一个程序包中其他的类使用。

abstract—抽象类，没有具体对象的概念类，没有具体实现功能，只用于扩展子类。例如：“鸟”,它可以派生出“鸽子”、“燕子”等具体类。

final—最终类，表示该类已经非常具体，没有子类可扩展。

17、类变量访问控制符

public:　任何其它类、对象只要可以看到这个类的话，那么它就可以存取变量的数据，或使用方法。

protected:同一类，同一包可以使用。不同包的类要使用，必须是该类的子类。

private:　不　允　许　任　何　其　他　类　存　取　和　调　用　。

default:　(前边没有修饰字的情况)在同一包中出现的类才可以直接使用它的数据和方法.

18、类变量

属于类的变量和方法——static

static 在变量或方法之前，表明它们是属于类的，称为类方法（静态方法）或类变量（静态变量）。若无 static 修饰，则是实例方法和实例变量。

类变量的生存期不依赖于对象的实例,其它类可以不通过对象实例访问它们。甚至可以在它的类的任何对象创建之前访问。

```
public class StaticVar
{  public static int number = 5;
}
public class OtherClass
{  public void method()
   {     int x = StaticVar.number;
   }
```

}

19、方法的声明与实现

方法是类的动态属性。对象的行为是由它的方法来实现的。一个对象可通过调用另一个对象的方法来访问该对象。

与类一样，方法也有两个主要部分：方法首部声明和方法体。方法声明的基本形式为：

返回类型 方法名( ) {

…… //方法体

}

方法声明的完整形式：

[方法修饰符] 返回类型 方法名称(参数 1,参数 2,…) [throws exceptionList]

{ …statements; //方法体：方法的内容

}

方法修饰符

[public |protected |private][static][final |abstract][native][synchronized]

返回类型可以是任意的 Java 数据类型，当一个方法不需要返回值时，返回类型为 void。

参数的类型可以是简单数据类型，也可以是引用数据类型（数组、类或接口），参数传递方式是值传递。

方法体是对方法的实现。它包括局部变量的声明以及所有合法的 Java 指令。局部变量的作用域只在该方法内部。

方法名可以是任何合法的 Java 标识符 。

（1）方法可以重载

Java 支持方法名重载，即多个方法可以共享一个名字。

（2）重载的方法不一定返回相同的数据类型，但参数必须有所区别：

- 参数的类型不同。例如，doubleIt(int x)和 doubleIt(String x)方法的两个版本的参数的类型不一样。
- 参数的顺序不同。这里是指一个方法有多个不同类型参数的情况，改变参数的顺序也算是一种区分方法。
- 参数的个数不同。

20、类方法

类方法独立于该类的任何对象，其他类不用实例化即可调用它们。

类方法可以调用其它的类方法

类方法只能访问 static 变量

类方法不能以任何形式引用 this 和 super

```
public class GeneralFunction  {
    public static int addUp(int x,int y)
      { return x+y;  }
}
public class UseGeneral  {
    public void method()  {
        int a = 9;
        int b =10;
        int c = GeneralFunction.addUp(a,b);
              //通过类名 GeneralFunction 引用 addUp 方法
    }
}
```

21、 对象声明

像声明基本类型的变量一样，对象声明的一般形式为：

类名 对象名;

声明一个引用变量时并没有对象生成(变量除了存储基本数据类型的数据，还能

存储对象的引用，用来存储对象引用的变量称为引用变量)

为对象分配内存及初始化

分配内存及初始化形式如下：

对象名= new 构造方法名([参数表]);

创建对象首先需说明新建对象所属的类，由与类同名的构造方法给出；然后要说明新建对象的名字，即赋值号左边的对象名；赋值号右边的 new 是为新建对象开辟内存空间的运算符，用 new 运算符开辟新建对象的内存之后，系统自动调用构造方法初始化该对象。若类中没有定义构造方法，系统会调用默认的构造方法。

对象的使用

对象的使用是通过一个引用类型的变量来实现，包括引用对象的成员变量和方法，通过运算符“ • ”可以实现对变量的访问和方法的调用。使用对象的基本形式如下：

<对象>.<域变量名>

<对象>.<方法名>

例如：

```
BirthDate date;
int day;
day = date.day;  //引用 date 的成员变量 day
date.tomorrow(); //调用 date 的方法 tomorrow()
```

22、定义和使用构造方法

❖ 构造方法是类的一种特殊方法，它的特殊性主要体现在如下几个方面：

- 构造方法的方法名与类名相同。
- 构造方法没有返回类型。
- 构造方法的主要作用是完成对象的初始化工作。
- 构造方法不能像一般方法那样用“对象.”显式地直接调用，应该用 new 关键字调用构造方法为新对象初始化。

定义了构造方法之后，就可以用如下的语句创建并初始化 Student 类的对象：

- Student card1＝new Student(张三, 男, 2004034567);
- Student card2＝new Student(李四, 女, 2003034666);
- 自定义无参的构造方法
  - 无参的构造方法对其子类的声明很重要。如果在一个类中不存在无参的构造方法，则要求其子类声明时必须声明构造方法，否则在子类对象的初始化时会出错
  - 在声明构造方法时，好的声明习惯是
    - 不声明构造方法
    - 如果声明，至少声明一个无参构造方法

23、方法修饰符

1. 抽象方法　abstract 修饰的抽象方法是一种仅含有方法声明部分，而没有方法体和具体的操作实现部分的方法。

2. 静态方法（类方法）　用 static 修饰符修饰的方法，是属于整个类的类方法，简称为类方法。

3. 最终方法　由 final 修饰符所修饰的类方法是最终方法。最终方法是不能被当前类的子类重新定义的方法。

4. 本地方法　native 修饰符一般用来声明用其他语言书写方法体的特殊方法，所有的 native 方法都没有方法体。

5. 同步方法　synchronized 修饰符主要用于多线程共存的程序中的协调和同步，保证这个 synchronized 方法不会被两个线程同时执行。

24、final 有三种使用方法

- final 在类之前，表示该类不能被继承。
- final 在方法之前，防止该方法被覆盖。
- final 在变量之前，定义一个常量。

25、抽象类与抽象方法

用 abstract 关键字来修饰一个类时，该类叫做抽象类；

用 abstract 来修饰一个方法时，该方法叫做抽象方法。

- 抽象类不能被直接实例化。因此它一般作为其它类的超类，与 final 类正好相反。
- 抽象类定义被所有子类共用的抽象方法，而实现细节由子类完成。
- 抽象类必须被继承，抽象方法必须被重写以实现具体意义。
- 抽象方法只需声明，而不需实现。定义了抽象方法的类必须是抽象类。

26、程序包（package）

由于 Java 编译器为每个类生成一个字节码文件，且文件名与类名相同，因此同名的类有可能发生冲突。为了解决这一问题，Java 提供包来管理类名空间。

如同目录是文件的松散的集合一样，包是类和接口的一种松散集合。一般并不要求处于同一个包中的类或者接口之间有明确的联系，如包含、继承等关系，但是由于同一包中的类在默认情况下可以互相访问，所以为了方便编程和管理，通常需要把相关的或在一起协同工作的类和接口放在一个包里。

27、打包

Java 中用 package 语句来将一个 Java 源文件中的类打成一个包。package 语句作为 Java 源文件的第一条语句，指明该文件中定义的类所在的包。(若缺省该语句，则指定为无名包)。

- 它的格式为：

```
package pkg1[.pkg2[.pkg3…]];
```

- Java 编译器把包对应于文件系统的目录管理，package 语句中，用 . 来指明目录的层次

28、使用包中的类可以通过以下 3 种方法之一进行：

1. 装载整个包

利用 import 语句载入整个包。例如：

```
import graphicPackage.*;
```

该语句必须位于源程序中的任何类和接口定义之前。

2. 装载一个类或接口

有时只需要某个包中的一个类或接口，无需装载整个包。如：

```
import graphicPackage.Circle;
```

3. 直接使用包名作类名的前缀

如果没有使用 import 语句装载某个包，可以直接在所需要的类名前加上包名作为前缀。例如： graplicPackage.Rectangle rectG;

29、在类的定义过程中，继承是一种由已有的类创建新类的机制。继承而得到的类为子类，被继承的类为**父类**，父类包括所有直接或间接被继承的类。

❖ 在类的声明中加入 extends 子句就可以创建一个类的子类：

```
class SubClass extends SuperClass {……}
```

30、一个父类可以同时拥有多个子类，而每一个子类则只能有唯一的父类。子类是对公共域和方法在功能、内涵方面的扩展和延伸。

父类、子类间的关系具有：

- 共享性　即子类可以共享父类的公共域和方法。
- 差异性　即子类和父类一定会存在某些差异，否则就应该是同一个类。
- 层次性　即由 Java 规定的单继承性，每个类都处于继承关系中的某一个层面。

31、定义一个子类，即在定义一个类的时候加上 extends 关键字，并在之后带上其父类名，其一般格式为：

```
[类的修饰符] class <子类名> extends <父类名>{
    <域定义>;
    <方法定义>;
}
```

32、域的继承与隐藏；方法的继承与覆盖

方法的覆盖

方法的覆盖（Override）是指子类重定义从父类继承来的一个同名方法，此时父类和子类中都存在一个同名方法，父类这个方法在子类中不复存在。这是子类通

过重新定义与父类同名的方法，实现自身的行为。

方法覆盖时应遵循的原则：

(1)覆盖后的方法不能比被覆盖的方法有更严格的访问权限。

(2)覆盖后的方法不能比被覆盖的方法产生更多的异常。

33、多态性是指同名的不同方法在程序中共存。即为同一个方法名定义几个版本的实现，运行时根据不同情况执行不同的版本。调用者只需使用同一个方法名，系统会根据不同情况，调用相应的不同方法，从而实现不同的功能。

34、多态性的实现有两种方式：

（1）覆盖实现多态性

通过子类对继承父类方法的重定义来实现。使用时注意：在子类重定义父类方法时，要求与父类中方法的原型（参数个数、类型、顺序）完全相同。

（2）重载实现多态性

在一个类中的定义多个同名方法的不同实现。定义方法时方法名相同，但方法的参数不同（参数的个数、类型、顺序不同）。这些方法同名的原因是具有类似的功能且目的相同，但在实现该功能的具体方式和细节方面有所不同，因此需要定义多种不同的方法体

35、在覆盖实现多态性的方式中，子类重定义父类方法，此时方法的名字、参数个数、类型、顺序完全相同，那么如何**区别这些同名**的不同方法呢？

此时这些方法是存在不同的类层次结构中，在调用方法时只需要指明是调用哪个类（或对象）的方法，就很容易把它们区分开来，其调用形式为：

对象名.方法名　　或　　类名.方法名

例如，IP电话的计费，若建立IP_Phone类的对象my，其调用为：my.charge_Mode();

假如charge_Mode()是一个类方法，则可使用类名，其调用为：IP_Phone.charge_Mode();

36、

- ❖ 若通过重载来实现多态性，则是在同一个类中定义多个同名方法。

- 由于重载发生在同一个类中，不能再用类名或对象名来区分不同的方法了，所以在重载中采用的区分方法是使用不同的形式参数表，包括形式参数的个数不同、类型不同或顺序的不同。

37、构造方法的重载是指同一个类中定义不同参数的多个构造方法，以完成不同情况下对象的初始化。例如，point 类可定义不同的构造方法创建不同的点对象。

```
point();          //未初始化坐标
point(x);         //初始化一个坐标
point(x, y);      //初始化两个坐标
```

一个类的若干个构造方法之间可以相互调用。当类中一个构造方法需要调用另一个构造方法时，可以使用关键字 this，并且这个调用语句应该是该构造方法的第一个可执行语句。

38、子类可以继承父类的构造方法，继承的方式遵循以下原则：

（1）子类无条件地继承父类的无参数的构造方法。

（2）如果子类没有定义构造方法，则它将继承父类的无参数构造方法作为自己的构造方法；如果子类定义了构造方法，则在创建新对象时，将先执行来自继承父类的无参数构造方法，然后再执行自己的构造方法。

（3）对于父类的带参数构造方法，子类可以通过在自己的构造方法中使用 super 关键字来调用它，但这个调用语句必须是子类构造方法的第一个可执行语句。

39、接口（interface）也有人翻译为界面，是用来实现类间多重继承功能的一种结构。

- 接口是在语法上与类有些相似。它定义了若干个抽象方法和常量，形成一个属性集合，该属性集合通常对应了某一组功能。
- 凡是需要实现这种特定功能的类，都可以继承并使用它。一个类只能直接继承一个父类，但可以同时实现若干个接口。实现（或继承）接口实际上就获得了多个特殊父类的属性，即实现了多重继承。
- 所谓多重继承，是指一个子类可以有一个以上的直接父类，该子类可以继承它所有父类的属性。

40、接口定义的仅仅是实现某一特定功能的一组对外的协议和规范，而并没有真正地实现这个功能。这些功能的真正实现是在继承这个接口的各个类中完成的。因为接口包含的是未实现的一些抽象的方法，它与抽象类有些相象。它们之间存在以下的**区别**：

- 接口不能有任何实现了的方法，而抽象类可以。
- 类可以继承（实现）许多接口，但只能继承一个父类。
- 类有严格的层次结构，而接口没有层次结构，没有联系的类可以实现相同的接口。

❖ 接口是由常量和抽象方法组成的特殊类。接口的定义包括两个部分：接口声明和接口体。声明接口一般格式如下：

❖ [public] interface 接口名 [extends 父接口名表] {

域类型　域名＝常量值;　//常量域声明

返回值　方法名(参数表); //抽象方法声明

}

❖ 接口声明中有两个部分是必需的：interface 关键字和接口的名字。用 public 修饰的接口是公共接口，可以被所有的类和接口使用；没有 public 修饰符的接口则只能被同一个包中的其他类和接口利用。

41、流是在输入输出之间流动的数据序列。

❖ 流一般分为输入流（Input Stream）和输出流（Output Stream）两类。流和物理文件是有区别的，流是一个动态的概念。比如一个文件，当向其中写数据时，它就是一个输出流；当从其中读取数据时，它就是一个输入流。当然，键盘只是一个输入流，而屏幕则只是一个输出流。

42、多任务

多任务是计算机操作系统同时运行几个程序或任务的能力。现代操作系统都支持多任务，多任务有两种形式：

➢ 基于进程的多任务

➢ 基于线程的多任务

- ❖ 程序、进程和线程
  - ➢ 程序是一段静态的代码，它是应用程序执行的蓝本。
  - ➢ 进程是程序的一次动态执行过程，它对应了从代码加载、执行到执行完毕的一个完整过程。程序可以被多次加载到系统的不同内存区域分别执行，形成不同的进程。
  - ➢ 线程是进程内部的一个顺序执行控制流。一个进程在执行过程中，可以产生多个线程同时执行。每个线程也有自己产生、存在和消亡的过程。

43、线程和进程的区别：

从逻辑的观点来看，多线程意味着一个程序的多行语句同时执行，但是多线程并不等于多次启动一个程序，操作系统也不会把每个线程当作独立的进程来对待。

- ➢ 两者的层次不同，进程是由操作系统来管理的，而线程则是在一个程序(进程)内部存在的。
- ➢ 不同进程的代码、内部数据和状态都是完全独立的，进程之间进行切换和通信的开销很大。
- ➢ 线程本身的数据通常只有寄存器数据以及程序执行时使用的堆栈，一个程序内的多个线程是共享同一内存空间和系统资源，线程的切换开销小，线程之间的通信很容易。

43、一个线程在它的生命周期中通常要经历五种状态。

- ➢ 新建(Newborn)
- ➢ 就绪(Runnable)
- ➢ 运行(Running)
  - ➢ 阻塞(Blocked)
- ➢ 死亡(Dead)

44、Java 通过面向对象的方法来处理程序错误。在 Java 中，程序的任何错误或不正常的执行都被称为**异常**。

- ➢ 在一个方法的运行过程中，如果发生了异常，则这个方法通过 Java 虚拟

机生成一个代表该异常的对象(包含了该异常的详细信息)，并把它交给运行时系统，运行时系统寻找相应的代码来处理这一异常。我们把生成异常对象并把它提交给运行时系统的过程称为抛出(throw)一个异常。

- 异常产生后，运行时系统从生成异常的方法开始进行回朔，直到找到包含相应异常处理的方法为止，这一个过程称为捕获(catch)一个异常。

45、除了 Java 类库所定义的异常类之外，用户也可以通过继承已有的异常类来定义自己的异常类，并在程序中使用(利用 throw 抛出异常，用 catch 捕捉异常)，这样异常类称为**自定义异常**。

46、异常的捕获与处理

1. try/catch/ finally 语句

- Java 的异常处理是通过 3 个关键词来实现的：try-catch-finally。用 try 来执行一段程序，如果出现异常，系统抛出(throws)一个异常，可以通过异常的类型来捕捉(catch)并处理它，或由最后的(finally) 最终处理器来处理。

```
try  {
    //接受监视的程序块,在此区域内发生的异常;
}  catch(要处理的异常类)  {
    //处理异常;
}  finally{
    //最终处理;
}
```

- try 子句

捕获异常的第一步就是用 try {…}语句指定了一段代码，该段代码就是一次捕获并处理异常的范围。在执行过程中，该段代码可能会产生并抛弃一个或多个异常，因此，它后面要用 catch 子句进行捕获时也要做相应的处理。

- catch 子句

每个 try 语句必须伴随一个或多个 catch 语句，用于捕获 try 代码块所产生

的异常并做相应的处理。 catch 语句有一个形式参数，用于指明其所能捕获得异常类型，运行时系统通过参数值把被抛弃的异常对象传递给 catch 语句。

❖ catch 子句（续）

程序设计中要根据具体的情况来选择 catch 语句的异常处理类型，一般应该按照 try 代码块中异常可能产生的顺序及其异常的类型进行捕获和处理，一般应尽量选择最具体的类异常型作为 catch 语句中指定要捕获的类型。

当然也可以用一个 catch 语句处理多个异常类型，这时它的异常类型应该是这多个异常类型的父类，但这种方式使得在程序中不能确切判断异常的具体类型。

❖ finally 子句

捕获异常的最后一步是通过 finally 语句为异常处理提供一个统一的出口，使得在控制流程转到程序的其他部分以前，能够对程序的状态作统一的管理。

finally 语句是一种强制的、无条件执行的语句，即无论在程序中是否出现异常，无论出现哪一种异常，无论 catch 语句的异常类型是否与所抛弃的异常的类型一致，也不管 try 代码块中是否包含有 break、continue、return 或者 throw 语句，都必须执行 finally 块中所包含的语句。

http://localhost:7000/questioncontent/1