1. ：java语言的概述
2. 计算机语言分为：编译型和解释型
3. 图形化界面（GUI）和命令行方式。
   1. DOS命令 dir,md,rd,cd,cd..,cd /,del,exit,ipconfig,ping
4. 语言的进化：
   1. 第一代：机器语言
   2. 第二代：汇编语言
   3. 第三代：高级语言
5. Java语言的特征：
6. JDK、PATH环境变量、classpath
   1. JDK:
   2. PATH:WINDOWS操作系统在执行命令时所要搜索的路径。
   3. CLASSPATH:JVM虚拟机，查找类的路径。
7. public class HelloWorld{

public static void main(String[] args){

System.out.println("HelloWorld");

}

}

6.1.源文件扩展名”java”,编译以后生成文件的扩展名class

6.2一个源文件里面只能有一个public类，其他类个数不限。如果源文件中包含public类，文件名必须同public类名保持一致。

6.3java程序的入口是main()方法。不去main（）方法的写法是“较为”固定。

6.4java语音是严格区分大小写。

6.5java语音的语句是以“;”(半角状态的分号)

6.6语句使用｛｝括起，成对出现。

1. Javac 和java
2. 注释：
   1. //:单行注释
   2. /\* \*/ :多行注释（多行注释不能够嵌套）
   3. /\*\* \*/:文档注释 javadoc -d 文件目录名 源文件名.java
3. 如何查看API.
4. ：基本语法：
5. 关键字：被java语言赋予特殊含义，用做专门用途的单词
6. 保留字：goto
7. 标识符：凡是自己可以命名起名字的地方都是标识符

规则：

3.1 26个大小写英文字母，0-9，或\_和$组成

3.2 数字不能开头

3.3 不可以使用关键字和保留字，但可以包含关键字和保留字

3.4 java区分大小写

3.5 标识符不能包含空格。

规范：

3.6 包名：所有字母小写

3.7 类名和接口名：多个单词组成，每个单词首字母大写：class HelloWorld

3.8变量名和方法名：多个单词组成，第一个单词首字母小写，第二个单词开始首字母大写： String firstNumber；

3.9常量名：所有字母大写，多个单词间用下划线连接：final String SCHOOL\_NAME;

1. 变量：
   1. 基本数据类型和引用类型
      1. 基本数据类型：
         1. 整数类型：byte，short，int，long，
         2. 浮点数：float，double
         3. 字符型：char
         4. 布尔类型：boolean
   2. 引用类型
      1. 数组
      2. 类（String）
      3. 接口
   3. 变量的运算
      1. 自动类型转换：
      2. 强制类型转换：语法： 数据类型 变量名 = （数据类型）数值

备注：引用类型的强制类型转换，需要存在继承关系。

* + 1. Byte、short、char之间运算时，结果为int型

1. 进制转换:
   1. 正整数的补码同源码相同
   2. 负整数的补码运算过程：源码》反码》加1
   3. 2进制》8进制》16进制：先算出2进制，转8进制，每3位为一个8进制，转16进制，每4位为一个16进制

11111011：二进制

11，111，011：八进制：（11）=1\*2^0+1\*2^1=3

(111)=1\*2^2+1\*2^1+1\*2^0=7

(011)=0\*2^2+1\*2^1+1\*2^0=3

1111,1011:十六进制：（1111）=1\*2^3+1\*2^2+1\*2^1+1\*2^0=15(F)

(1011) =1\*2^3+0\*2^2+1\*2^1+1\*2^0=11 (B)

1. 运算符：
   1. / ：int i = 12 ； i/5=?(是一个整数型)
   2. % ：-38%-3 计算的结果由第一个数决定.
   3. ++,-- :如果运算符在前面：先加1，在运算 ，运算符在后面：先运算，在加1
   4. 赋值运算符： x+=5; x=x+5;
   5. 短路&&或者｜｜　同　非短路＆　或　｜　的区别
   6. 三元运算符　　（表达式０）？表达式１：表达式２；如果表达式０为真，取表达式１的值，否则取表达式２的值。
2. 分支结构：
   1. Ｉｆ
      1. 一但满足某个条件表达式，则进入其执行语句块执行，执行完毕，跳出当前的条件判断结构。后面的分支语句不会在执行。
   2. Ｓｗｉｔｃｈ
      1. Byte,short,int,char,枚举，String
      2. Case 后面的是一个常量
      3. Default语句可以省略，如果没有满足的，就默认执行default语句。
      4. Break：一旦满足了case语句，进入执行其操作，直到遇到break，跳出switch结构。
      5. 建议：若要判断的语句的条件满足switch变量的数据类型，并且值不多的情况，可以使用switch，否则使用if
3. 循环结构
   1. 格式：

a初始化条件

b 循环条件

c迭代部分

d循环体

* 1. For（a;b;c）{

D;

}

A

While(b){

D;

C;

}

A

do{

D;

C;

}while(b);

* 1. While 和do while ：do while程序至少执行一次。 Do while的循环条件括号后面有一个“；”分号
  2. 嵌套循环：如果外层的循环次数是M,内层的循环次数是N，那么这个程序循环的次数是M\*N;

技巧：一般来说，外层循环控制行数，内层循环控制的是列数

* 1. Break和continue
     1. Break：是结束循环，跳出循环体
     2. Continue：是结束本次循环。
  2. 数组：
     1. 一维数组：
        1. 数组的初始化
           1. 静态初始化：Int[] array = new int[]{1,2,3,4,5};
           2. 动态初始化：int[] array = new int[3];

Array[0]=4;

通过下标访问，下标从0开始。数组有一个属性length，用于获 取数组的长度

* + 1. 多维数组：
       1. 数组的初始化
          1. 静态初始化：int[][] array = new int[][]{{1,2},{4,5,6},{7,8,9,0}};
          2. 动态初始化:int[][] array = new int[4][6]; array[1][2]=5;
          3. 注意二维数组的长度：

Int[][] array = new int[2][6];

二维数组行的长度：array.length;

二维数组列的长度：array[0].length ://第一行对应数组长度

* + 1. 数组常见异常：java.lang.ArrayIndexOutOfBoundsExecption :数组下标越界异常（下标从0开始，因此使用下标访问时最长长度为x[长度-1]）
    2. 数组工具类：Arrays类，可以使用类中的sort方法为数组排序。

1. 面向对象
2. 面向对象同面向过程 ：人开门 同 把大象放进冰箱
3. 面向对象的编程关注于类的设计
   1. 一个项目或者工程，不管多么庞大，一定是一个一个类构成的。
   2. 类是抽象的，好比是制造图纸（制造汽车的图纸），而具体的一辆辆的车，是根据图纸制造，而成的实实在在的一个实例（川O 00001）
4. 完成一个项目（功能）的思路
   1. 所有完成的功能对应的类的对象是否存在
   2. 若存在，则通过对象直接调用对应类中的属性或方法。
   3. 若不存在，需要穿件类的对象，如果类都不存在，就需要设计类。
5. 面向对象的三条主线：
   1. 类及类的构成成分：属性、方法、构造器、代码块、内部类
   2. 面向对象编程的特征：封装、继承、多态（多态之间一定有继承关系）
   3. 关键字：this、super、package、import、static、final......
6. 类的组成：
   1. 属性：（成员变量，Field）
      1. 成员变量同局部变量
         1. 相同点：
            1. 遵循变量声明的格式： 数据类型 变量名 = 初始值;
            2. 都有作用域s
         2. 不同点：
            1. 声明的位置不同：成员变量：声明在类的里面，方法的外面。局部变量：声明在方法的内，方法的形参，代码块。
            2. 成员变量的修饰符有4个：public、private、protected、缺省。局部变量没有修饰符
            3. 初始化值：成员变量：如果在声明的时候，不显式的赋值，那么不同数据类型会有不同的默认初始值。（一定会有值）。局部变量：一定要显式的赋值。（局部变量是没有默认初始化值的）。
            4. 两者内存中的位置不同:成员变量在堆控空间中，局部变量在栈空间中。
   2. 方法：（成员方法，函数，Method）
      1. 提供某种功能的实现。
      2. 格式： 修饰符 返回值类型（void/具体的返回值） 方法名（形参）{}
      3. 关于返回值：void：表明此方法不需要返回值。如果有返回值：在方法的最后一定有return 返回值类型所对应的变量/表达式。因此void和return不能同时存在。
      4. 方法内可以调用本类的其他方法或属性，但不能在方法内再定义方法
7. 面向对象编程思想的规则
   1. 设计并创建类的成分。
   2. 实例化类的对象。
   3. 通过“对象.属性”或“对象.方法”的形式完成某种功能。
8. 语法：

声明类语法格式：

[< 修饰符>] class < 类名>

{

[<属性声明>]

[<构造器声明>]

[<方法声明>]

}

说明：修饰符public：类可以被任意访问

类的正文要用{ }括起来

====================================================

声明属性的语法：

语法格式：

[<修饰符>] 类型 < 属性名> [=初值] ;

说明：修饰符 private: 该属性只能由该类的方法访问。

修饰符 public: 该属性可以被该类以外的方法访问。

类型：任何基本类型，如int、boolean或任何类。

=====================================================

声明方法的语法：

语法格式：

<修饰符> <返回类型> <方法名>([< 参数表>]) {

[< 语句>]

}

说明： 修饰符：public,private,protected 等。

返回类型：return语句传递返回值。没有返回值：void。

1. 封装性
   1. 为什么要使用封装？：当创建了类的对象后，如果直接通过“对象.属性”的方式对相应对象属性进行赋值的话，可能会出现不满足实际需求的情况。我们考虑不让对象来直接操作属性，而是通过“对象.方法”的形式来控制对象属性的方法
   2. 封装的思想：
      1. 将类的属性私有化。
      2. 提供公共的方法来实现赋值或者调用
   3. 封装有4个关键字：public,protected,缺省（不显示声明权限修饰符），private
2. 构造器
   1. 构造器类的成员。（成员变量、成员方法、构造器）
   2. 构造器的作用：
      1. 创建类对象实例
      2. 给创建的对象属性进行初始化
   3. 注意：
      1. 设计类时，如果不显式声明类的构造器，程序会默认提供一个空参的构造器。
      2. 一旦显式定义了构造器，那么系统将不再提供默认的构造器。
      3. 声明构造器的语法：权限修饰符 类名（参数）｛｝
      4. 类的多个构造器之间的重载。
3. 方法的重载
   1. 必须要在同一个类中
   2. 方法名必须相同
   3. 方法的参数列表不同
      1. 参数的个数不同
      2. 参数的类型不同
      3. 参数的排练顺序不同
4. This关键字
   1. This关键字表示当前对象或者是当前正在创建的对象。
   2. This表示当前对象的一个引用，是调用当前方法的那个对象的引用。
   3. 在类的构造器中调用重载的构造方法，this必须放在构造方法的第一行。（构造器重复代码的利用）
5. Java的值传递机制
   1. 方法的参数传递
      1. 形参：方法声明时，方法小括号内的参数
      2. 实参：调用方法时，实际传入的参数的值
   2. 规则：java中的参数传递是：值传递
      1. 形参如果是基本数据类型：将实参的值传递给形参的基本数据类型变量.(形参和实参的内存空间是不同的)
      2. 形参如果是引用数据类型：将实参的引用类型变量的值（对应的堆空间对象实例的首地址）传递给形参引用变量。（形参和实参都指向同一个内存空间）
6. Package/import
   1. Package
      1. 写在程序的第一行
      2. 每“.”一次，表示的一层文件目录
      3. 包名都要小写
   2. Import
      1. 显式导入指定包下的类或接口。
      2. 写在包的声明和类声明之间。
      3. 如果要引入多个类或者接口，就并列写出所有的包。
      4. 如果需要导入某个包下的所有类，那么可以写成：import gxjy.test.\*;
      5. Import static 表示导入指定类的static属性和方法。
7. 类的设计
8. 继承
   1. 为什么要继承
   2. 通过“class A extedns B”
   3. 子类继承父类后，父类中声明的属性、方法，在子类中可以获取到，并且子类还可以定义自己特有的部分。
   4. 父类中private修饰的成员，不能够被继承。
   5. Java的继承是单继承。一个类只能继承一个父类，一个父类可以有多个子类。
9. 权限修饰符

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 修饰符 | 同意一个类 | 同一个包 | 子类 | 整体 |
| Private | Y |  |  |  |
| Default | Y | Y |  |  |
| Protected | Y | Y | Y |  |
| Public | Y | Y | Y | Y |

1. 方法的重写和覆盖
   1. 为什么要重写？
   2. 前提：要做继承的基础上。
   3. 重写的方法名、参数列表、返回值类型必须相同。
   4. 权限修饰符，不能严格于被重写的方法。
2. Super
   1. Super是在子类方法中对父类对象的引用，使用super能在子类方法中引用父类的成员。
   2. 在子类的构造器中调用父类的构造器。
      1. 在子类的构造方法中可使用super(参数列表)语句调用父类的构造方法
      2. 如果子类的构造方法中没有显示地调用父类构造方法，也没有使用this关键字调用重载的其它构造方法，则系统默认调用父类无参数的构造方法
      3. 如果子类构造方法中既未显式调用父类构造方法，而父类中又没有无参的构造方法，则编译出错
      4. 注意：掌握创建对象时，关于构造器的调用过程和逻辑。
3. 多态
   1. 什么叫多态：
      1. Java引用变量有两种类型：编译时类型和运行时类型。编译时类型由声明该变量时使用的类型决定。运行时类型由实际赋给该变量的对象类型决定。（Person p = new Student();）如果编译时类型和运行时类型不一致，就是多态
      2. 同一个类型的多个变量，在执行同一个方法时，呈现出多种行为的特征。
      3. 子类能够替代父类使用。
   2. 多态的表现：
      1. 方法的重载和重写
      2. 子类对象的多态性
   3. 多态的前提条件：
      1. 继承关系
      2. 要有方法的重写
   4. 在多态的情况下（当父类类型的变量指向一个子类的对象）：不能调用子类中新添加的方法或者属性。（如果要解决这个问题，必须通过强制类型转换）
4. instanceof
   1. X instanceof A :X对象是否为A类的子类或者其本身对象。返回true或者是false。
5. 引用类型的转换
   1. 如果要转换成功，必须有继承关系。
   2. 最好使用instanceof进行检查后，在转换，避免出现类型转换异常。
   3. 从父类的对象转换为子类的类型。
6. Object类：
   1. Object是所有java类的根父类
   2. haseCode()
   3. Equals();
   4. toString();
7. 包装类
   1. 从基本数据类型转换为String类：调用的是String类重载的valueOf(XXX XX);方法
   2. 从String类转换为基本数据类型：调用的是相应包装类的parseXxx（String str）方法
8. 高级类的特性
9. Static关键字
   1. 静态的，可以用来修饰属性、方法、代码块、内部类
   2. Static修饰的属性
      1. 由类创建的所有的对象，都共用这一个属性。
      2. 当其中一个对象对此属性修改后，会导致其他对象对此属性调用是都是一样。（非static的属性，每个对象都各自拥有一份副本）
      3. Static属性是随着类的加载而加载的，并且只有一份。
      4. Static属性通过“类.类属性”的形式类访问调用。
      5. 类属性的加载早于对象，所有当有对象以后，可以使用“对象.类属性”来方法static属性。（“类.实例属性”是不能操作的）
      6. Static属性存在于静态区域中。
   3. Static修饰的方法
      1. 随着类的加载而加载，在内存中也只有一份
      2. 访问方式“类名.类方法”的方法调用
      3. Static 的类方法可以调用static的属性，但不能调用非static的属性。反之，非static的方法是可以调用static属性或者方法的。
   4. 静态代码块
      1. 它相当于一个无参的构造器
      2. 里面是可以输出语句。
      3. 随着类的加载而加载，static的代码块只加载一次
      4. 多个static代码块之间是按照顺序结构执行的。
      5. 主要的作用，初始化类的static属性
   5. 单例模式
      1. 私有化构造器

Private Bank（）

* + 1. 创建类的对象，同时设置为private的，通过公共的方法来调用
    2. 要求此对象是static的

Private static Bank instance = new Bank();

* + 1. 公共方法也是static的

Public static Bank getInstance(){

Return instance;

}

1. Main()方法
   1. Main（）是一个方法，是主方法，它是程序的入口。
   2. 参数就是一个字符串数组，从命令行输入的。
2. Final关键字
   1. Final：表示最终的，可以用来实现类、属性、方法
   2. Final修饰的类不能被继承
   3. Final修饰的方法不能被重写
   4. Final实现的属性，一旦初始化后，就不能在被赋值。（1.定义的时候显式的赋值2.代码块3.构造器）
3. 抽象类（abstract）
   1. Abstract可以用来修饰类、方法。
   2. 修饰类
      1. 不能被实例化。
      2. 抽象类有构造器
      3. 抽象方法所在的类，一定是抽象类
      4. 抽象类中不一定需要抽象方法。
   3. 修饰方法
      1. 没有方法体：public abstract void test（）;
      2. 抽象方法只保留了功能，而具体的执行方式，交给继承抽象类的子类，由子类来重写此方法。
      3. 如果子类继承了抽象类，并且重写了所有的抽象方法。这个了可以实例化。反之，如果子类没有重写所有的抽象方法，代表这个类中还有抽象方法的存在，那么这个类仍然是抽象类。
4. 接口
   1. 接口于类是一个并行的概念
   2. 接口可以看做是一个特殊的抽象类。里面只有：常量和抽象方法。
   3. 接口没有构造器
   4. 接口定义的就是一个功能，这个功能被所有实现类来实现（implements）
   5. 实现接口的类的，必须要重写其中所有的抽象方法。如果没有重写所有的重写方法，这个类是一个抽象类。
   6. 类可以实现多个接口。多个接口之间用逗号隔开。
   7. 接口与接口之间也可以继承，也就存在多态的关系。
   8. 如果一个类既要继承又要实现接口，那么extends在implements前面
5. 内部类
   1. 可以在类的内部再定义类。
   2. 内部类可以访问外部类的方法和属性，外部类不能方法内部类的方法和属性
6. 异常
   1. 结构体系：
      1. java.lang.Object
         1. java.lang.Throwable
            1. java.lang.Error:错误，java程序对此无能为力，不能显式的处理
            2. Java.lang.Excepiton:异常。需要处理的

RuntimeExcepiton:运行时异常。（java）

ArrayIndexOutBoundsException

NullPointerException

ArithmeticException

ClassCastException

非RuntimeExcepiton异常：编译时异常（javac）

IOException

* 1. 运行时异常同编译时异常的特点
     1. 运行时异常，是可以不显式处理的。
     2. 编译时异常，必须显式的处理。编译时异常，不是说有异常才处理，而是存在异常的隐患，必须在编译前，提示程序，万一出现异常，如何处理。
  2. 如何处理异常
     1. Java的异常是“抓抛模型”
        1. 抛：（自动抛出/主动抛出）
           1. 当我们执行代码时，一旦出现异常，就会在异常代码处生成一个对应的异常类型对象，并且将此对象抛出。

异常抛出后，程序就终止了（try语句块后面的代码不在执行）。

异常对象抛给了方法的调用者。

* + - 1. 抓：抓住上一步抛出的异常类对象。抓的意思：如何处理异常。
         1. Java有两种方式处理异常

Try...（可能出现异常的代码）...catch...（处理方式）...finally..（一定要执行的代码）...

注意：

Finally是可选的

Catch语句内有两个方法：getMessage()，printStackTrace();

可以有多个catch语句，try中抛出的异常类对象，从上往下去匹配catch中的异常类对象，一旦匹配成功，就执行catch中的代码，然后跳出，其后面的catch不在执行。

如果异常已经处理成功，异常后面的代码继续执行。

Catch可以同时捕获多个异常：catch(UserNotExistExecpiton|ArithmeticException e)

如果多个catch是“包含关系”，必须将子类放在父类的前面，否则将出现错误。

Try---catch可以嵌套

在声明放的时候，显式的使用throws+异常类型

可以同时抛出多个异常，用逗号隔开。

语法：public void test（）throws Excepiton1，Exception2

如何手工的抛出一个异常：

在方法的内部，使用throw+异常对象，来手工的抛出一个异常。

谁调用这个方法，谁就去处理这个异常。

自定义异常

一定要继承一个异常类（RuntimeExcepiton）

如何自定义一个异常：

Public class gxjyException extends RuntimeException{

Stataic final long serialVersionUID =-90234782309L；

Public gxjyException (){

Super();

}

Public gxjyException (String msg){

Super(msg);

}

}

自定义异常一定通过throw抛出。