1. JDK 的安装
   1. 环境变量的设置
      1. JAVA\_HOME
      2. Path
2. JDK, JRE, JVM
3. Java运行过程
   1. 编译，javac, .java🡪.class
   2. 执行, java
   3. 即使编译型语言，也是解释型语言
4. 标识符 Identifier
   1. 就是变量、函数、类、常量的命名规则
   2. 编码，Unicode UTF-8
   3. 字母/下划线/$ 开头，字母、数字、下划线、$符号组成
   4. 变量、函数、包，用小写字母开头，CamelCase
   5. 类，用大写字母开头，CamelCase
   6. 常量，全大写
   7. 不能为关键字
5. 变量
   1. 基本类型，存放值
   2. 引用类型，存放地址（32位地址）
   3. 局部变量 local variable。
      1. 属于函数/语句块，使用前必须初始化
   4. 成员变量 instance variable，field，attribute。
      1. 属于对象，若未初始化，则有默认值(0, 0.0, false, \u0000, null)
   5. 静态变量。
      1. 隶属于类本身的变量，类变量，用 static 修饰。
      2. 若未初始化，则有默认值(0, 0.0, false, \u0000, null)。
6. 基本数据类型
   1. 数字
      1. 整数
         1. byte, short, int, long
         2. 占用Byte数量：1，2，4，8
         3. Int，一般表示+-21亿的数量
         4. Long数字，带L
      2. 小数
         1. float, double
         2. 占用Byte数量： 4，8
         3. 一般用double，因为float很容易出现进度问题
         4. float数字，带 F
   2. boolean
   3. char
      1. 2字节，Unicode
      2. 保存时使用 UTF-8，（有些字母还是占据1个Byte）
   4. 数字扩展之非基本数据类型
      1. BigInteger，保存更大的整数
      2. BigDecimal，不会丢失进度
7. 引用类型
   1. 占用 4 Byte，即32位的地址，指向堆Heap内的地址
   2. 常见的类型
      1. String
      2. 数组、Collection
      3. 对象
      4. 接口
8. 类型转换
   1. 转换进度
      1. 无精度丢失
         1. Int 转 double
         2. char 转 int
         3. byte/short/int/long之向上转
         4. float/double，向上转
      2. 有精度丢失
         1. 向下转
         2. long转double
   2. 自动提升
      1. char+byte，int
      2. int+long, long
      3. float+double，double
      4. String + 数字/boolean/char，String
      5. 如果要利用自动提升，则先把二元运算符的第一个数字做类型提升
         1. (double)60\*60\*24\*365\*10000
   3. 强制转
      1. (int)doubleValue
      2. instanceof
         1. (Man)person
9. 运算符
   1. 算数运算符
      1. +, -, \*, /, %
      2. ++（先加、后加）
      3. --（先减、后减）
   2. 关系运算符
      1. >, <, >=, <=, !=, ==
   3. 逻辑运算符
      1. && || !
      2. 有短路效应
      3. 两边的运算值，必须时boolean值（不是其他数字等类型的）
   4. 位运算符
      1. &, |, ~, ^
      2. <<, >>, >>>
         1. >>，考虑到了数字符号
         2. >>>，不考虑正负数字符号，头部用0填充
         3. 快速的运行与 2 相关的乘除运算
   5. 扩展运算符
      1. +=, -=, \*=, /=. %=
   6. 三目运算符
      1. Type variable = boolean ? value1 : value2 ;
   7. 赋值运算符
      1. =
   8. 其他技巧
      1. If (booleanValue) ……
      2. If ( 0 == intValue) ……
10. 控制语句
    1. 分类：顺序结构、选择结构、循环结构
    2. If
    3. If--else
    4. If--else if—else
    5. switch
       1. case, break, default
       2. 没有 break 时的击穿现象
       3. 接收的可对比的数值类型
          1. 最大是int的整数类型: byte, short, int, char (即能自动转型为int的类型)
          2. String
          3. 枚举 enum
          4. 封装类 Byte Short Integer Character
          5. 不支持 long float double
    6. While
    7. Do-while
    8. For
    9. For AdvanceVersion (for int i : intArray)
    10. continue, break
        1. 注意，可以用标签，标签必须与循环搭配着使用，并且在使用前要提前定义好，定义在循环之前
        2. 如果没有标签，那么 continue\break 影响的只有当前循环，而不是父循环（在循环嵌套的情况下）
11. 面向对象基础
    1. 前戏1：对象的进化史
       1. 基本数据类型阶段
          1. 数据少，无数据管理时代
       2. 数组
          1. 同类型数据的聚集，弱管理
       3. 结构体
          1. 数据多了、复杂了
          2. 将不同类型的数据放在一起管理，强管理
       4. 对象
          1. 数据复杂、行为复杂
          2. 不同类型数据、不同的行为（函数）的聚集体
          3. 超强管理时代
    2. 前戏2：面向过程、面向对象、函数式编程
    3. 对象 & 类
       1. 类：抽象的、抽取对象的共性的、模板，可创建对象
       2. 对象：具体的，类的实例
    4. 类的定义

[public] class 类名 {

//属性

private Type attributeName;

private static Type 类属性名;

//可选的

public Type get/set方法{

}

//方法

Public Type 方法{

}

//构造方法

public 类名 {

首行有一个隐含的super()，永远有的！调用父类无参构造器

方法名必须与类名一致

无返回值类型

通过new来调用

}

}

* 1. 无参构造器
     1. 如果未定义任何构造器，系统会自动添加一个无参构造器（第一行是super() ）
     2. 如果定义了带参构造器，则不会有系统默认添加的无参构造器
     3. 强烈推荐总是创建无参构造器，这样自己的子类不会出现构造器问题
  2. 继承的类、 创建实例、构造器、内存分析
     1. 创建实例时，一直上溯到Object类的构造函数（通过super）
     2. 从Object逐步向下的时候，根据继承关系逐步增加新的属性与方法
     3. this指向的是对象自身
  3. this 的含义

普通方法中：代表调用本方法的对象

构造方法中：代表正要初始化的对象

还可用于调用本类的其他构造器

* 1. super的含义

指向直接父类

* 1. static

静态的方法、变量，可以直接通过类名来调用，从属于类

内存中，存放于堆Heap中的方法区

* 1. package

必须放在第一行（非注释性的第一行）

包名：域名倒着写 com.khanintel.somePackage

* 1. import

java.lang的类不用import，同package的类不用import

用于引用其他包的类

* 1. import static XXX

导入类的静态的变量、方法

* 1. final
     1. 修饰变量，变量为常量，禁止改
        1. 常量名都是全大写，多个单词用下划线隔开
     2. 修饰方法，继承后禁止重写
     3. 修饰类，禁止继承
  2. 三大基石：封装、继承、多态
  3. 封装：private, default, protected, public。
     1. 权限：自己、同package内、子类、所有类
        1. Private, 仅自己能访问
        2. Default, 自己、同Package可访问
        3. Protected，自己、同package、子类
        4. Public，自己、同package，子类、所有类
        5. 所有类，可以访问public
        6. 子类，可以访问public, protected
        7. 同一个包，可以访问public, protected, default
        8. 同一个类，可以访问 public, protected, default, private
     2. 小技巧：区分 protected/default，口诀是“远亲不如近邻by default”，就是default修饰符控制下，同package可访问，子类不可访问
     3. 封装的目标：高内聚、低耦合
        1. 封装掉复杂性
        2. 与外部的接口尽量简单
  4. 继承
     1. extends
     2. Java 单继承
     3. 两个好处:
        1. 代码重用
        2. 通过继承实现对现实世界更加准确的建模（动物世界）
  5. 实现接口

见下面的描述吧

* 1. 多态Polymorphism
     1. 条件：
        1. 父类引用执行子类对象
        2. 子类对父类函数进行了重写
        3. 继承
     2. 多态，仅仅是“方法”的多态，不是属性的多态！
  2. 动态绑定、静态绑定

还是看我的Onenote《继承--动态绑定 vs 静态绑定》《继承--是否可以重写静态方法》

* 1. 类、抽象类

抽象函数，只有函数定义，没有函数体，必须被子类重写

凡是包含抽象函数的类，都必须是抽象类

抽象类不一定要有抽象函数

抽象类、接口，都是一种规范

抽象类，可以拥有变量、常量、函数（普通方法、抽象方法）、构造器，就是不能有实例，不能被new

抽象类的引用（变量）可以指向子类对象

* 1. 接口
     1. 定义：interface （注意不是class，就是头部定义的 public interface someInterface {}）
     2. 文件是 .java文件，编译后是 .class文件
     3. 实现接口： implements。一个类可以实现多个接口，逗号隔开就好
     4. 接口之间，可以多继承
     5. Best Practice，尽量“不要”在接口里面定义常量，否则子类、孙类很难找到、操控、使用这个静态常量
     6. 定义的是一组规范，实现现实世界中这样的逻辑：如果你是...，则你必定能...。
  2. OuterClass, InnerClass, 函数内Class
  3. 重载 Overload

一个函数，多种输入的组合

函数名相同、输入不同

返回值不同不足以做重载

总结：两同三不同。

同一个类，同一个函数名

三不同：参数列表不同（参数的类型、个数、顺序）

* 1. 重写 Override (Overwrite)

英文：Override用的更加准确

==：

方法名保持一致

>=：

子类的权限修饰符可以大于等于父类的(private,default,protected,public之递增)

<=：

子类的返回值类型小于等于父类的类型

子类的异常类型小于等于父类的异常的类型（RuntimeException这种父类/子类都有的）

1. 内存机制
   1. Stack栈，Heap堆（堆里面含方法区）
   2. Stack
      1. 存放“局部变量”
      2. 基础数据类型存放的是变量的值（就是值本身），引用类型变量存放的是内存地址
      3. 方法运行的时候，也会开辟一块栈帧StackFrame用于存放函数处理时的变量信息
      4. “不可以被多个线程共享”
      5. 空间连续、空间小、速度快
   3. Heap
      1. 存放对象实例 与 方法区
      2. 可以被多线程共享，这才导致了我们的锁机制
      3. 空间不连续、空间大、速度慢、灵活
   4. 方法区：
      1. 存放类的信息（程序代码、静态变量、常量（含字符串））
      2. 可以被多线程共享，也有资源冲突的问题
2. 垃圾回收 Garbage Collection, GC
   1. 系统调用，不是程序员用程序调用。
   2. 程序员只能提醒系统，但具体GC时机，由系统自信确认。
      1. 通过System.gc() 建议GC回收
   3. 没有引用的对象会被回收（栈里面没有变量指向该对象了）
   4. 可以人工写代码，调用对象的finalize()方法来让GC回收掉
      1. finalize()来自于Object的方法，每个类都有。
      2. 但一般还是不要自己调用finalize()
3. 异常 Exception
   1. [Justin] 看自己写在代码里面的长长的注释吧，写的挺好的
4. Object类
   1. 所有类的根基类
   2. 常用方法
      1. toString()
         1. 默认实现return getClass().getName() + "@" + Integer.toHexString(hashCode());
      2. equals(), hashCode()
         1. BastPractice两者一起重写。
         2. 只要equals，则hashCode一定一样。
         3. hashCode一样，不一定equals。
      3. getClass()
5. 递归算法
   1. 递归头（什么时候调用自己、什么时候不掉用自己，总要有个时刻“终结”递归的）
   2. 递归体
6. 算法
   1. 冒泡
   2. 二分法BinarySearch