Java笔记

一些小点（记录在《Java语言程序设计（基础篇）》上的页数）

- 编译java和一些基础知识：15
- 标识符规则（名称）：34（需注意：可以包含'$'）
- 数据类型和范围：38
- 局部变量和方法参数在jvm中的储存方法是相同的，都是在stack上开辟空间来储存的
- jvm，每个线程有一个stack（动态变量），但是共享heap（常量池）
- java参数传递：基本类型传值，类传地址，不是传引用
  - 一个类只占据4字节的stack空间，仅仅记录地址——指向内存中的实例
  - 方法传递引用变量的参数的时候，只是把地址拷贝传进去，赋值给stack中的一个**新的引用**
  - 如果在方法中，只是对引用进行修改（如等号赋值），而没有对指向的对象地址空间进行修改（如StringBuffer的append），那么不会有影响
- 可变长参数列表：224

```
public static void test(double... numbers) {
    for(double a : numbers) {
        System.out.println(a);
    }
    for(int i = 0; i < numbers.length; i++) {
        System.out.println(numbers[i]);
    }
}
```

- Arrays：230（sort，binarySearch，equals，fill等）
  - `binarySearch(list, element)`前需要sort，且按照升序
  - `sort(list)` / `sort(list, from, to)`，to不参与排序，可以没有from或者to
  - `fill(list, element)` / `fill(list, from, to, element)`，to不参与赋值
- final基本数据类型不可变；对列表/类指的是不可改变引用对象（地址），因为这些的变量的值就是地址
- java的内存分配在堆上，c++的内存分配在栈上，所以不能`int a[100];`，需要赋值new给a。
- java没有指针，只有引用。
- 所有引用对象的初始化都需要new，除了string有特殊的初始化方法，和基本类型primitive一样是等号初始化
- java不会处理int类型的溢出，比如求`INT_MIN`的相反数，它就当作一个无符号数取反，确实溢出了但是结果正常。
  - 如果需要处理这个情况，就用long变量接住结果，然后和预期的结果进行比较判断是否溢出，有必要的话再throw一个exception。（https://stackoverflow.com/questions/3001836/how-does-java-handle-integer-underflows-and-overflows-and-how-would-you-check-

fo)

- 一个Java源文件中最多只能有一个public类，当有一个public类时，源文件名必须与之一致，否则无法编译，如果源文件中没有一个public类，则文件名与类中没有一致性要求。
- main()不是必须要放在public类中才能运行，而且public static void main（String[] args）可以有多个。
- java内存机制
  - stack：java的基本类型，int, short, long, byte, float, double, boolean, char和对象句柄(引用)。
    - 线程的私有财产
  - heap：java的类数据
    - 所有线程公用
  - 常量区：一些字面常量如String等等，经常被引用的量，编译时确定（写在.class文件中）
    - jdk1.7之后放在了heap中，所有线程公用
- java里面用runnable比Thread继承更好，因为它可以避免单类继承的限制，更为灵活
  - 通常用例：`new Thread t(new Xxx()).start();`传回调给Thread的start之后执行
- 重载和重写
  - 方法名字一样是重载override，所以有多个方法，根据参数列表决定
    - 这个情况下，父类不知道子类其他方法的存在所以不能调用；
    - 子类根据输入参数决定调用；首先匹配参数个数，然后匹配单个参数，如果不匹配再向上转型，找到它的包装类，再到父类，最后接口等，如果没找到就查找可变参数列表；还找不到就报错。
  - 方法名字和参数列表都一样是重写overload，所以写完只有一个方法
  - 实例：子类对象用父类类型
    - override的函数无法调用，再使用父类方法；
    - overload的函数可以调用，使用子类方法；除非子类调用super的方法，否则父类方法永无出头之日
    - overload的函数的返回值类型**可以不同**，但是必须是是向下cast的（子类的返回值类型是继承父类返回值类型的）；否则必须父子方法返回值类型相同
- 访问限制修饰符
  - private
  - protected
  - package（默认）
- 类型修饰符
  - static
    - static方法不能访问this和super等
- java数据类型
  - 基本数据类型primitive
    - 数值型：整数型和浮点型，int（4字节），byte（1），short（2），long（8），float（4），double（8）
    - 字符型：char（2字节），因为使用utf-8编码
    - 布尔型：boolean（1字节）
  - 引用数据类型reference

        - 类class——一切类的super：Object类
        - 接口interface
        - 数组array

    在基本数据类型中，除了boolean类型所占长度与平台有关外，其他数据类型长度都是与平台无关的。比如，int永远占4个字节（1 Byte = 8 bit）

    `==` compares object references, it checks to see if the two operands point to the same object (not *equivalent* objects, the **same** object). 所有的objects，除非引用的是同一个东西，不然是不会相等的

- 类型转换
    - 基本数据类型
        - double→float→long→int→short(char)→byte需要强制转换
        - 反过来精度变高，隐式转换
    - 引用数据类型，强制转换
        - 子类可以直接转换成父类对象（向上转型）
        - 父类对象转成子类（向下），只能在其内存本质是子类对象但是原引用是父类的条件下
        - instanceof可以判断类型，也可以getclass方法
- 名词
    - jdk包括jre和javac编译器
        - jre包括jvm和api
            - jvm是执行环境
    - jit是编译技术，just-in-time
    - javac将程序源代码编译，转换成java字节码，JVM通过解释字节码将其翻译成对应的机器指令，逐条读入，逐条解释翻译。
- 类的初始化
    - 先默认初始化（null，0）
    - 指定初始化（int a=999）
    - 构造函数初始化
- 垃圾回收
    - garbage is automatically collected by jvm
    - 如果不需要对象，将指向他的引用设置为null，jvm检查没有其他引用就会回收它
    - 如果内存由类自行管理，就需要警惕内存泄漏：
        - 如**对象数组**的活动区域和非活动区域，需要程序员决定是否回收，jvm是不知道的
        - 软引用（softReference，内存够用就不删除），强引用（一般引用，没有引用就删除），
- 序列化Serialization
    - 需要实现`java.io.serializable`
    - 序列化不保存静态变量，transient变量会被设置为初始值
    - `ObjectInputStream`，`ObjectOutputStream`是输入和输出
    - 序列化一个对象，需要它内部的所有成员都可以序列化
- 拷贝clone

    - 需要实现`Cloneable`
    - 如果是引用数据类型，只会浅拷贝
    - 在内部自定义的类也需要实现Cloneable接口并实现Clone函数`return (super).clone();`
- 迭代器
    - 迭代器不能用于map，见下图：
    -
- `List<String>`不能直接赋值给`List<Object>`，但是String对象可以赋值给Object引用
    - 因为String和Object有继承关系但是`List<String>`和`List<Object>`没有继承关系
- String：**用literal值初始化的和用构造方法初始化的两个的地址不同**
    - 用intern（）可以找到常量池中的原值
    - 用String（）构造出来的是真正的String类，两两不同；用literal得到的两两相同
    - 以上String对象只要值相同，intern就是想同的
    - 这两个是相同的：String.valueOf(a) == a.intern()；**但是Double类的valueOf就不是相同的，因为它是泛型类装箱，如Double a = Double.valueOf(b)返回的a==b也是false**
- 关于内部类inner
    - 生成新的：`Inner inner = this.new Inner();`，不是`Inner inner = new this.Inner();`（报错）
    - 定义在method中的class只能访问父类的final变量
- Integer在和int比较的时候会自动拆箱unbox到int再比较
- static{}语句块中的内容只在JVM加载类的时候被执行一次（无论实例化多少次，都只执行一次）
- 比较引用对象
    - ==：比较引用的地址，准确来说在heap中的地址
    - equals：来自Object，如果不重写的话就是比较引用的地址；String，Integer等对象重写了equals，比较内容是否相同
    - compareTo：实例化comparable接口需要实现
- 求长度比较
    - List：size()
    - int[]：length
    - String：length（）
- Exception
    - 如果定义一个异常类A，和一个继承了A的类B
    - 如果catch先抓父类A，再抓B，会报错
    - 如果先抓B再抓A就不会报错，但是没有意义……
    - 理由：抓住了父类，一定抓住了子类；抓住了子类不一定抓得住父类
    - 关于Try后面的return：
        - `try{ return; }catch(){} finally{} return;` 顺序：程序执行try块中return之前（包括return语句中的表达式运算）代码； 再执行finally块，最后执行try中return;

          finally块后面的return语句不再执行。
        - `try{ } catch(){return;} finally{} return;` 顺序：程序先执行try，如果遇到异常执行catch块，
            - 有异常：执行catch中return之前（包括return语句中的表达式运算）代码，再执

行finally语句中全部代码，最后执行catch块中return. finally块后面的return语句不再执行。
        - 无异常：执行完try再finally再执行最后的return语句。
    - `try{ return; }catch(){} finally{return;}` 顺序：程序执行try块中return之前（包括return语句中的表达式运算）代码； 再执行finally块，因为finally块中有return所以提前退出。
    - `try{} catch(){return;}finally{return;}` 顺序：程序执行catch块中return之前（包括return语句中的表达式运算）代码； 再执行finally块，因为finally块中有return所以提前退出。
    - `try{ return;}catch(){return;} finally{return;}`

      顺序：和上面规律一样。
- goto和const是未使用的保留字；类名不是保留字，可以用作变量名：
    - 比如`int Double = 1;`是合法的变量名