

Computer Science A

计算机科学A



「2.0 版」
刘淼
连龙



编者按——AP 手册 2.0 前言

『为什么要重编 AP 手册？』

还记得去年这个时候，我们推出了 AP 手册 1.0，随即在出国留学圈掀起了一阵小波澜，那或许是少见的有专业教研团队静下心来推动 AP 备考资源的整合和升级。第一版的 AP 手册推出之后便收到大量好评，作为一个有幸参与其中的编者，我对 TD 的教研成果能够帮助到考生备考 AP 感到十分欣喜。

然而我们并不满足于此。第一版手册确实已经足够优秀，清晰地覆盖到了 AP 考试要求掌握的绝大部分知识点，但是由于首次编写，我们显然还有不少能够做到更好地方，比如：对考题分析的关注不够全面，在备考经验的总结还可以继续丰富甚至于有的手册还有一些小 bug 需要处理。这些都让我们萌生了对 AP 手册进行全面更新的想法。

得益于 TD 的大量社群活动，我们发现在 TD 社群的各个角落里有大量学霸的潜伏，我们聚集并筛选了二十位极为优秀的同学组成了「AP 手册研发更新小组」，在 AP 手册 1.0 基础上，将十个常见 AP 科目的知识点整理更新工作分配出去。参与编撰的研发成员最基本门槛自然是对应科目 AP 5 分，同时更为难能可贵的则是大家都有着良好的学习总结能力和笔记习惯，同时深知 AP 备考的痛点和难点。

和去年一样，尽管参与者在相关科目的背景都足够靓丽，但我们的定位并非「教材」，而是一本「学长笔记」——即在确保知识框架严谨且完整的前提下（感谢主催刘森在审稿的时候熬了多个通宵在每一个科目的 Course Description 上一个一个 check 知识点），尽量丰富的展现出一个考生在备考过程中总结出来的各式经验。这不仅仅包括学长们在刷题过程中总结出来的做题技巧和在学习对应学科疑难知识点的“各种小窍门”，还有高质量的例题和真题练习甚至于部分学科还有真题解析。而这些，正是 AP 手册 2.0 区别于传统教辅的真正价值所在：这是一本『有温度的学长笔记』，学长们备考过程中所经历的不解、困顿，以及他们跳过的坑、走过的路，都可以在其中若隐若现的找到踪迹。

最后由衷的感谢所有参与者的付出，编写组成员一个多月的时间里投入了大量的时间和精力，绝大部分小组的成果都远超最初的预期，希望我们的工作能够为后面一届又一届的同学带来价值。

特别说明一下，因为工作量比较大，编撰的过程难免有疏漏。如果你在使用的过程中发现了有学术上的漏洞或者其他任何建议，欢迎发邮件到 testdaily@163.com，我们会有薄礼相赠。

愿我们的工作能让你的备考变得容易一些，也愿你的备考申请顺利。



前言

我们很兴奋，当我跟另外一位编辑完成这个手册的编写后，我们对自己写的东西爱不释手。这本册子凝聚了我们的心血，我希望各位读者能够好好利用手册这个资源去辅助 AP Computer Science A 的学习：认真阅读册子中给出的代码、举的例子以及每一章节之后的练习题。

我还记得自己刚开始学计算机的时候，是因为想做出一个能够属于自己的游戏，这样就能够开个号让自己拥有各种各样神装。于是我决定退出竞赛，抱着一本 C++ Primer Plus 开始了我的计算机之路。这条路很苦，但是真的很令人着迷。写出自己第一个 Hello World 程序后感觉自己确确实实能够干出点什么了。但是结果不出意料的是对着 IDE 发了一天的呆。

计算机真的很难 对于初学者来说，如果没有人来领着入门的话，很容易步入一个误区，从此不知所云。尤其是在半懂不懂的状态下从一门语言转换到另外一种语言的时候能让人癫狂。明明感觉自己是会的，但是就是理解不了，看不明白。我在当初学 C 语言后又转变到 Java 时就有这个感受。不过后来去一家软件公司实习后，一个贴心的程序猿带我介绍了面向对象的特性。从最基础讲起每种语言的细微变化，让我终于突破了那个瓶颈，从此一马平川。我希望我们的这个手册也一样能起到相同的效果，带你们飞到 5 分。

不过“师傅领进门，修行看个人”。能够取得一个 5 分并不仅仅是靠一本手册，或者各种各样教辅就能解决的。还是那句老生常谈的话，要有兴趣，要努力。如果仅仅只是为了拿一个 5 分为自己的申请增加优势的话，那我建议看完我们的第一章后，优先看第五章。第五章除能看懂的话，就接着往下学。否则，还是把时间留给我们其余的标化考试吧。AP 只是一个点缀，它对申请的作用不起到什么显著性影响。甚至在考完之后忘了寄 AP 成绩都是大有人在的。所以如果不是感兴趣的话，还是趁早干自己真正想干的事情吧。但如果真的想学、

喜欢的话，哪怕是想用来抵换大一计算机必修课学分的话，那就低头脚踏实地吧。

怎么去努力是大家比较关心的一个问题。最简单的方式就是砸时间，把它想象成玩阴阳师一样。就是肝体力，肝完体力休息休息接着肝。但是很多人做不到这一点，所以就需要高效的学习方法。对于CS的学习，我们有一个自带优势就是可以很方便的去写代码。代码的书写是提升对程序理解最快的，也同时是最好玩的。写完自己的程序后，调试bug，调戏程序都是学习CS乐趣的一部分。所以我建议我们边啃我们的手册（包括巴郎），边上手写程序。哪怕只是一段System.out.println("Hello Wolrd"); 都会让我们对于代码更加熟悉，做题速度更快，甚至有种忍不住拍下来发朋友圈的冲动。但是这并不代表我们不需要练习手写代码，因为如果我们使用过于强大的IDE的话对我们的编程习惯有不好的影响（比如eclipse），我们需要手写代码，但是也可以通过比较简单的IDE（强烈安利BlueJ）熟悉我们代码，加深我们的记忆。

总而言之，计算机科学真的是一门很有趣的学科，我很希望学习它的人最后都能喜欢上这门学科，并且学的出色。

最后，

```
if((you.getInterest()).contains("ComputerScience"))  
    System.out.println("欢迎入坑");
```

刘淼



作为一位学习计算机科学 9 年、在无人机公司的写代码的实习生，教大家写点代码对我不是难事，但要真正把一个手册写得让初学者看得明白、学会 Java，却不容易。在这次手册编写中，首先我自己熟悉了一遍 AP 对 Java 的考点，同时也深入了解了参考书的编写结构和流程，甚至把自己用的巴郎 AP Computer Science A 翻了个遍。无论是从题的选择，从例子的讲法，无不学到甚多。

事实上，学好计算机科学这个各项指数正在呈乘幂式发展的学科的确不容易，学习者首先要具有计算机思想和运行模型，因为无论是后期的算法实现还是 AP 所考查的程序分析和编写，对计算机思想与其运行方式都有极大要求。在掌握了这种思想之后，像 Java 这种面向对象型编程对你们都只是同类相推罢了。

在此推荐大家，若是想真真正正学会计算机科学之技能，以计算机的思维探索学科之奥妙，还需具体实践，躬身入代码之海洋，须知计算机之巧妙，不在代码，而在其运行至能效，得此者，不难学深入也。

最后在这里感谢 testdaily 的支持以及我教的同学们，从你们之中我逐步学到了如何向入门者阐述计算机知识、如何才能将课讲得生动有趣。同时也希望读者们在阅读本手册、做完题之后能进一步了解计算机科学，体会计算机科学的魅力。

连龙 Tony

9 March 2017

目录

第一章 考试介绍	6
Part I 考试简介.....	6
Part II 考试常见题型.....	6
Part III 考试内容及纲要.....	7
Part IV 训练方式.....	11
Part V 推荐参考书.....	12
Part VI 做题须知.....	12
第二章 Java 环境语言	13
Part I 变量、标识符和类型.....	13
Part II 操作、储存和表示.....	24
第三章 标准类(I)	41
Part I 类.....	41
Part III Math 类.....	47
第四章 Array 和 Array List	52
Part I 一维数组.....	52
Part II Array List.....	55
Part III 二维数组.....	57
第五章 类与对象	60
Part I 类与对象.....	60
Part II 方法.....	63
Part III this 关键字.....	70



Part IV 数据类型.....	71
第六章 继承和多态.....	77
Part I 方法概念扩充.....	77
Part II 继承 (inheritance)	79
Part III 多态(polymorphism).....	82
第七章 标准类(II).....	90
Part I Object.....	90
第八章 递归.....	98
Part I 认识递归.....	98
Part II 深入递归	103
第九章 搜索与排序.....	107
Part I 搜索(searching)	107
Part II 排序(sorting).....	108
第十章 程序设计与分析.....	114
Part I 瀑布模型(The waterfall Model).....	114
Part II 面向对象程序设计	116
Part III 程序分析.....	118

第一章 考试介绍

Part I 考试简介

AP 计算机科学 A 考试(AP Computer Science A,简称 AP CS A)是一场考察考生逻辑思维能力和计算机语言掌握能力的考试。考生需要以 JAVA 语言为基础，运用面向对象型编程和计算机编程语言及其逻辑的知识来解决问题。考试时长三个小时，分为以下两个部分：

Section I: Multiple Choice [1 hour and 30 minutes for 40 multiple-choice questions]

Section II: Free-Response [1 hour and 30 minutes for 4 problems involving extended reasoning]

Part II 考试常见题型

根据对 AP Course Description 的选择题样题分析可以得出（括号里是样题的题号）

Section I 有以下几种题型：

1. 对程序运行效果的分析（比如哪几个可以给出这个答案、这段程序会输出什么、运行了程序之后 variable 会变为什么，1、2、3、4、5、8、9、10、13、14、15、16、18、19、20、24）
2. 错误分析和改正（题目原意是如此，然后接着下面函数哪个能达到目的、为什么这段程序不能达到目标，怎么才可以最终达到目标，11、12、23、25）
3. 面向对象程序设计（主要是考如何把生活中的事情抽象成计算机的方式来描述，6、7、22）
4. 程序运行效率、代码的最佳位置等其他题（17、21）



Part III 考试内容及纲要

AP 编程所用的语言是 Java , 一门经典的面向对象编程语言 就是书上所说的 Object-Oriented Program Design , 主要要求我们掌握两大模块 , 一是如何才能用对象的思维抽象问题 , 如把数据抽象成各种变量及数组 ; 二是如何才能使用算法将抽象的内容 (题目可能会给出抽象好的内容 , 如变量定义) 。 以下的内容供大家参照初步了解 AP 计算机科学 A 的具体要求。

一、 Object-Oriented Program Design 面向对象设计

1. 我们需要明白如何用计算机的方式来看待问题;
2. 需要明白怎么把现实生活中的数据用计算机的方式描述;
3. 我们要知道类和接口的特点以及两种关系 (“is-a”就是 inheritance relationship 继承 , “has-a”就是 composition relationship 组成关系) ;
4. 明白 code reuse(override) 和 code overload 这两种易混淆的概念;
5. 需要明白数据呈现形式和算法实现形式;
6. 需要明白如何把操作分解 , 简单地说就是把一个操作变为很多子操作.

二、 *程序的实现方法和形式

1. Top-down 从上到下
2. Bottom-up 从下到上
3. Object-oriented 面向对象式
4. Encapsulation and information hiding 对一定对象封包和隐藏信息
5. Procedural abstraction 过程抽象

三、 程序结构

1. 知道 primitive types 和 reference types 的特点和区别

2. 知道常量、变量、方法及其参数、类、接口的定义和应用（这个及其重要，例如要学会如何使用 extend、implement 来创建子类和实现接口）

3. 知道如何使用 System.out.print、System.out.println 输出数据（或其他有特殊说明的输出方式）

4. 学会如何调用方法、如何顺序执行和条件执行、循环遍历、递归

5. 知道数字的表达式、字符串表达式和布尔表达式的计算

C. AP 要求的 Java 类（String 等里面的方法和参数需要记忆，考过很多次）

四、程序分析

1. 测试

a) *test case 的开发（包括 boundary case ）

b) 单元测试

c) 整体测试

2. 调试

a) 要知道不同分类的错误，compile-time error，run-time error 和 logic error

b) 错误的辨别和改正

c) 如何使用添加输出语句进行调试

3. 运行出的 exceptions



4. 程序更正

a) 前提条件和后来条件（就是执行前需要什么，执行后会是什么）

b) 断言

5. 算法分析

a) 要会分析出语句执行次数

b) 要会大体比较运行的时间（一般双方所用的时间和效率会差距很大）

6. 要学会用不同进制表示一个非负数，还要知道整数的最大最小限制integer.MAX_VALUE和integer.MIN_VALUE，知道计算机储存数的方式，如整数的最高位是用来判断正负的。TIPS：
不是最高位从0变为1就变成相反数了。

五、 标准数据结构（如何把题目的概念抽象成数据等）

1. 原始数据类型(primitive data types) e.g. int, boolean, double

2. 字符串

3. 类

4. List

5. 一维和二维数组

六、 标准操作和算法

1. 数据结构操作，如截断、添加、删除

2.顺序搜索和二叉树搜索

七、计算机在实际应用方面的问题（官网有说，但是很少涉及，大家就简单看看）

1.系统可靠性

2.隐私

3.法律条例和知识产权

4.计算机使用在社会和道德方面的后果

打*号的部分建议大家在阅读完本手册后阅读一下巴朗参考书，上面有很详细的解释。



Part IV 训练方式

在刚开始准备的时候推荐大家使用一个叫做 fenby 的网站上的 Java 课程附带的在线练习程序进行练习，比如程序运行后会出现什么结果这种题可以在网上运行后自己测试，自己写的程序也在上面可以进行检查看逻辑有没有错误。

到了考试前一个月，知识应该学得差不多的时候就该实打实地训练了。对于往年的选择题官网基本没有放出，Course Description 里面有一套参考题，推荐大家作为考前训练用，当然如果想测试自己真实水平也可以做做来练手，其他时候考前有时间的话每天一套参考书的选择题来防止知识的遗忘。

对于问答题，推荐大家不要用电脑来写，一是因为到时候不是机考，二是因为题目主要考察的是计算机思想，里面的 Class 很多是没有给实现的，所以如果不自己实现是没办法运行的，而做题时自己去实现完全没有必要（考纲说明里说了学生不需要写整个程序，只需要了解写过的部分并且写或修改程序以改变功能）。建议大家练习时拿 HB 铅笔（到时候会要求用铅笔写而不是水笔）和橡皮自己计时完成，一套题大概留出 15 分钟左右可以重新读一下题和自己的答案试着用参考数据模拟运行一下，看一看结果是不是和参考相同。对于问答题大家可以优先练 5~8 年的官方题，不够再练参考书的，选择题可以每次练一套参考书的，下面有说现在官方题里面的不需要掌握的知识点。

一开始写出的程序和参考答案不像不要紧，但是后面就要慢慢和参考答案靠近，因为参考答案上的算法一般效率都会比较高，而自己写的可能存在一些运行的问题或效率低。当然高手们练到后期会发现自己写的速度比参考速度还要高，那就另当别论，毕竟参考答案要给大多数学生看，建议考试和练习的时候不要炫技，分数要紧，稳扎稳打即可，不然老师要是看不懂或者自己写错某个步骤就不太好了。

Part V 推荐参考书

除了本手册，还推荐大家去看看巴朗 AP 计算机科学 A，它的优点在于把概念写得很完整，很多基础知识说明得很详细，而且，最主要的是它不是很难懂，不会的词查一下就能看明白句子的意思。上面的考纲的内容巴朗里都基本都有详细介绍。里面的题目也都很不错，易错点和不会考的知识点都有专门提醒，选择题和大题都很适合练习和模考，引进版的巴朗还配有中英目录翻译，对专有名词的理解还是有帮助的。虽然其中的内容有点超纲，但是看一看对于理解计算机科学的思想包括最终考试还是有好处的。

如果不想看英文材料，还可以以上面推荐的编程学习网站——fenby，上面的视频教程或许能让你学得更轻松，但是英文教材最好还是读一下，毕竟考试的内容就在上面。 fenby 网址：<http://www.fenby.com/>

Part VI 做题须知

有以下几个部分是现在不要的，如果在官方题中遇到可以不做了：

- 一、 GridWorld Case Study 从前年开始就删掉了
- 二、 instanceof 这个操作符以前在标准答案中出现过，现在 Course Description 中的描述是 not tested in the exam but potentially relevant/useful。在以后写多态的时候可以用这个去避免 ClassCastException，但并不强制要求掌握。
- 三、 三个 labs (The Magpie Lab, The Elevens Lab, The Picture Lab)，现在的描述是 optional and are not tested on the exam



第二章 Java 环境语言

Part I 变量、标识符和类型

一、变量(variable)和常量(constant)

在数学中学过方程和函数的同学都知道，我们常常设一个 x ，对其赋予一定的值，那么对于计算机而言，这个 x 就是一个数值量。在计算机中，如果这种量可以改变，那么它就是变量，如果不能，那么它就是常量。变量和常量都是用来储存数据使用的。

e.g.

圆周率 PI 从发现以来就没有改变过。如果我们认为它在程序运行时不会改变，那么我们就可以认为它是常量，取近似值储存。

e.g.

今天几点钟可以用一个数字表示，那么我们把这个数字赋给一个量 x ，现在是 7 点，那么 x 就是 7，然后到了 8 点，再把 8 赋给 x ， x 就是 8，这样能改变的 x 就是变量。

其实在不变化的情况下，常量和变量是基本上一致的，因此对于含常量的问题可以用普通变量的方法进行分析。

二、标识符(Identifiers)

这是大家在 Java 学习中遇到的第一个概念，**标识符**其实是我们所称的名字，它包含了变量名、参数名、常量名、方法名、类名。对于刚才介绍的变量 x 来说，它的标识符就是 x ，这样是不是很容易理解？

三、类型(Types)

1. 原始类型(primitive types,也称 built-in types)

AP 常用的原始类型有以下几种：

类型名	英	中
int	integer	整数
boolean	boolean	布尔代数
double	double precision float	双精度浮点

所谓的“整数”，就是包括正负在内的一定范围内的整数，具体储存范围后面会提到，在计算机内精确储存。

e.g.

往 `x` 储存 1，`x` 就变成了 1，就是 1，不会变成 1.0000000000…0001，是精确的。

“布尔代数”就是真或假，`true` 或 `false`，不可以存储其他数据，也是精确储存。

e.g.

太阳东升西落，这个判断的结果就是真的，因此用布尔代数来储存，就应该储存 `true`。

“双精度浮点”其实是我们生活中的小数，只不过它储存的不是精确数据，“双精度”是指储存数值的双精度，当然对应的有单精度浮点 `float`，只不过 AP 基本上不会考到，具体数值储存等会提到。

e.g.

往 `double` 类型的 `x` 赋 2.7，和 $0.1+0.1+\dots+0.1$ (27 项)，是不相等的，其原因就在于浮点类型会产生偏差，也就是 round-off error，之后就不能精确表示小数数值了，因此最终结果就不是 2.7，和我们赋的 2.7 就不相等了。

除此之外还有 `long` 等基本类型，AP 不要求掌握。

接着我们就来谈谈如何定义一个变量：

e.g.

```
int hour; //比如这个就可以作为时间的时长
```



```
boolean statement; //比如这个就可以储存太阳东升西落是否正确
```

```
double speed; //这个就可以储存河水的流速
```

通过上面的三个例子，大家应该知道了定义变量的方法。方式是：

```
E name;
```

其中 E 是类型，name 是变量的名字。

TIPS: java 中每一个语句结束都需要一个英文半角分号以结尾，不然会报错的。

2. 包装类型(wrapped types)

除了原始类型以外 Java 还有包围类型，这个之后学了类以后就会提到。

四、赋值

1. 赋值语句

前面其实已经提到了赋值的概念，不知道大家是否有注意。赋值就是把一个数字或对象给予一个变量。结合刚才的定义变量的方式我们来给变量赋值：

e.g.

```
int x;
x = 4;
```

好了，这样我们就把 4 给 x 了。但是……怎么看呢？我们还需要一句语句用于调试。

最后一句 System.out.println(x);**就是输出 x 的值，让我们能看见：**

```
int x;
x = 4;
System.out.println(x);
```

推荐大家到 fenby 的在线练习工具上面运行代码，这样可以不用下载软件。

在 <http://www.fenby.com/courses/sections/javade-gong-zuo-fang-shi/> 的在线练习

中把 System.out.println("Hello Fenby!");换成你想运行的代码就能运行了。

The screenshot shows a Java code editor with the file 'Hello.java' containing the following code:

```
1 public class Hello {  
2     public static void main(String[] args) {  
3         int x;  
4         x = 4;  
5         System.out.println(x);  
6     }  
7 }
```

Below the code are several buttons: '运行' (Run), '重置' (Reset), '变量输入' (Variable Input), '提示' (Tip), '历史记录' (History Record), and '全屏编辑(Ctrl+E)' (Full Screen Edit). The output window below shows the welcome message and a command prompt '\$ |'. A large watermark 'TestDaily' is visible across the interface.

图 1

是不是输出了我们想要的结果？

接着我们可以试试 boolean 和 double。

```
boolean a;  
double b;  
a = false;  
b = 120.1;  
System.out.println(a);  
System.out.println(b);
```

除了刚才运行的结果 4 现在下面是不是又出现了 false 和 120.1？（出现 4 是因为刚才显示出了这个，刷新网页就会消失，并不是程序里面输出 4）



```

1 public class Hello {
2     public static void main(String[] args) {
3         boolean a;
4         double b;
5         a = false;
6         b = 120.1;
7         System.out.println(a);
8         System.out.println(b);
9     }
10 }
```

Welcome to Fenby online IDE

4
false
120.1
\$ |

图 2

TIPS : 不止文字 , 代码也要排版 , 大家可以用空格或者 TAB 键对 fenby 上面自己的练习进行排版对齐 , 这样才会更好看 , 这同时也是一个程序员要形成的习惯。

2. 初始化时赋值

我们学会了初始化 (定义变量) 和赋值 , 那么这两条语句能不能直接进行 , 就是初始化的时候就赋值呢 ? 当然可以。

```
int a = 1;
```

这句代码就相当于定义 a 为 int 类型然后给其赋值 1。

等同于

```
int a;  
a = 1;
```

3. 初始化不赋值读取

那读者可能会想到 , 如果一个变量 , 不赋值就进行读取会怎样 ?

我们可以试一试。

e.g.

```
int a;//没有进行赋值  
System.out.println(a);
```

TIPS://… 是单行注释，后面是说明语句，计算机不会识别后面所写的内容，是给代码

的读者看的，/* … */是多行注释，其中 “…” 的部分属于注释的内容

The screenshot shows a Java code editor with a dark theme. A file named "Hello.java" is open. The code contains a single class definition:1 public class Hello {
2 public static void main(String[] args) {
3 int a;
4 System.out.println(a);
5 }
6 }

The line "System.out.println(a);" is highlighted in red, indicating a syntax error. Below the code editor, there are several buttons: "运行" (Run), "重置" (Reset), "变量输入" (Variable Input), "提示" (Hint), "历史记录" (History), and "全屏编辑(Ctrl+E)" (Full Screen Edit). The output window below shows the following error message:

```
4  
false  
120.1  
编译错误,请查看程序高亮行!  
/usercode/Hello.java:4: error: variable a might not have been initialized  
        System.out.println(a);  
               ^  
1 error  
Compilation Failed
```

图 3

答案是，出现错误，大家可以注意到这里是在编译(compile)过程中出现错误，这个可以先记下。这下大家可能会明白定义时赋值的好处，那就是变量至少会有默认值，不会出现错误。

4. 常量定义及赋值

我们可以通过 final 来定义常量。



e.g.

```
final int a = 10;
```

当然我们也能先定义，后初始化。

```
final int a;
```

```
//一些代码
```

```
a = 10;
```

这样定义并初始化的常量就不能修改了。

5.类型转换

我们有以下代码

e.g.

```
int a = 1;
```

```
int b;
```

如果我们想把 a 里面的值给 b，那很容易做到，只需要

```
int a = 1;
```

```
int b = a; //这里简写了，在定义的时候赋值
```

如果 b 是双精度浮点，这么做也可以。

但如果 a 是双精度浮点而 b 是整数呢？

e.g.

```
double a = 1;
```

```
int b = a;
```

```

1 public class Hello {
2     public static void main(String[] args) {
3         double a = 1;
4         int b = a; // Error line
5     }
6 }
7 
```

全屏编辑(Ctrl+E)

```

$ 
编译错误,请查看程序高亮行!
/usercode/Hello.java:4: error: possible loss of precision
int b = a;
           ^
required: int
found:   double
1 error
Compilation Failed
$ 

```

图 4

是的，程序不能运行。

原因是双精度浮点转换为整数会丢失小数点(其实计算机按照另一套方法储存双精度浮点，不是我们所用十进制方法) 后面的数字，所以错误中说 possible loss of precision，虽然我们的 double 是 1，小数点后面去除后和原先一样，但是同样会报错。

那如果我们确实需要把 double 赋值给 int，去除小数点后面的数呢？这就需要借助我们所说的类型转换这个概念了。

类型转换(cast)，就是把一个类型转成另外一个类型，其实上面把 int 赋值给 double 已经做过了一次类型转换，然而这次把 double 赋值给 int 需要做一次显式类型转换(explicit cast)，就是要让 java 编译器明白我们要把 double 的值赋值给 int，去除小数点后面的数。

e.g.

```

double a = 1;
int b = (int)a;
System.out.println(a);
System.out.println(b); 
```



Hello.java

```
1 public class Hello {  
2     public static void main(String[] args) {  
3         double a = 1;  
4         int b = (int)a;  
5         System.out.println(a);  
6         System.out.println(b);  
7     }  
8 }  
  
$  
$  
$  
$  
$  
$  
$  
$  
1.0  
1  
$ |
```

运行 重置 变量输入 提示 历史记录 全屏编辑(Ctrl+E)

图 5

这样就能看出来，上面是 double，下面是 int。

刚才我们没有使用小数，而是用了 1 这个整数进行尝试，这次我们换成小数 1.1。

e.g.

```
double a = 1.1;  
int b = (int)a;  
System.out.println(a);  
System.out.println(b);
```

Hello.java

```
1 public class Hello {  
2     public static void main(String[] args) {  
3         double a = 1.1;  
4         int b = (int)a;  
5         System.out.println(a);  
6         System.out.println(b);  
7     }  
8 }  
$  
$  
$  
$  
$  
$  
$  
$ 1.1  
1  
$ |
```

运行 重置 变量输入 提示 历史记录 全屏编辑(Ctrl+E)

图 6

大家需要注意的是，这个舍弃小数点后面的数的意思是截断，而不是四舍五入，也就是
1.1 截断后是 1，1.9 截断后仍然是 1。



Hello.java

```
1 public class Hello {  
2     public static void main(String[] args) {  
3         double a = 1.9;  
4         int b = (int)a;  
5         System.out.println(a);  
6         System.out.println(b);  
7     }  
8 }  
  
$  
$  
$  
$  
$  
$  
$  
$  
1.9  
1  
$
```

运行 重置 变量输入 提示 历史记录 全屏编辑(Ctrl+E)

图 7

如果我们想四舍五入呢？请参考后面的 Math 类。

Part II 操作、储存和表示

一、操作

1. 算数运算符

运算符	意义	示例
+	加	$1 + 1$ 返回 2
-	减	$1 - 1$ 返回 0
*	乘	$2 * 2$ 返回 4
/	除	$10 / 4$ 返回 2 而不是 2.5
%	除余/取模	$10 \% 4$ 返回 2

TIPS: 上面这些运算符大家应该都学过，然而大家可能会奇怪为什么 $10/4$ 返回 2，答案是 10 是整数，2 也是整数，那么计算机就会认为这个运算是整数运算，因此返回的结果也是整数结果。这个比较可能考，需要注意，凡是几个整数进行运算都会默认使用整数运算，如果要让程序进行小数运算，需要把整数转换为小数才可以。

对于常数运算，可以使用 10.0 让计算机认为我们输入的是浮点数，那么就会进行浮点运算。



The screenshot shows a Java code editor window titled "Hello.java". The code contains a class definition with a main method that prints the values of variables `a` and `b` to the console. Below the code editor is a toolbar with buttons for "运行" (Run), "重置" (Reset), "变量输入" (Variable Input), "提示" (Hint), "历史记录" (History Record), and a red button labeled "全屏编辑(Ctrl+E)". The output window below the toolbar displays the results of the execution: "Welcome to Fenby online IDE", "2.0", "2.5", and a prompt "\$".

```
1 public class Hello {  
2     public static void main(String[] args) {  
3         double a = 10/4;  
4         double b = 10.0/4;  
5         System.out.println(a);  
6         System.out.println(b);  
7     }  
8 }
```

运行 重置 变量输入 提示 历史记录 全屏编辑(Ctrl+E)

Welcome to Fenby online IDE
2.0
2.5
\$

图 8

或者我们也能显性地把 10 转换为双精度浮点，这样就可以进行浮点运算了。

The screenshot shows a Java code editor interface. At the top left, it says "Hello.java". Below is the code:

```

1 public class Hello {
2     public static void main(String[] args) {
3         double a = (double)10/4;
4         System.out.println(a);
5     }
6 }
```

Below the code are several buttons: "运行" (Run), "重置" (Reset), "变量输入" (Variable Input), "提示" (Hint), "历史记录" (History Record), and "全屏编辑(Ctrl+E)" (Full Screen Edit). The terminal window below shows the output of the program:

```

$ $ $ $ $ $ $ $ 2.5 $
```

图 9

2. 关系运算符

运算符	含义	示例
<code>==</code>	相等	<code>1 == 1</code> 返回 true
<code>!=</code>	不等	<code>1 != 1</code> 返回 false
<code>></code>	大于	<code>2>1</code> 返回 true
<code><</code>	小于	<code>1<2</code> 返回 true
<code>>=</code>	大于等于	<code>1>=1</code> 返回 true
<code><=</code>	小于等于	<code>1<=1</code> 返回 true

为什么要说这些运算符呢？这是因为这些返回 true 或 false 这些 boolean 值的操作符可以参与判断指导程序运行。



这里顺带介入 if 条件判断语句

if(条件){

 如果条件为真，则执行

}else{

 如果条件为假，则执行

}

比如某个网站需要根据使用者的信息进行操作。

e.g.

```
int age = 16;
if(age < 18) {
    System.out.println("不行，不能使用");
} else{
    System.out.println("可以使用");
}
```

Hello.java

```
1 public class Hello {
2     public static void main(String[] args) {
3         int age = 16;
4         if(age < 18){
5             System.out.println("不行，不能使用");
6         }else{
7             System.out.println("可以使用");
8         }
9     }
10 }
```

The screenshot shows a Java development environment with the following interface elements:

- File Menu:** 新建(N), 打开(O), 保存(S), 另存为(A), 退出(E), 分组(G), 全屏编辑(F11), 帮助(H), 关于(About)
- 工具栏:** 运行(Run), 重置(Reset), 变量输入(Variables Input), 提示(Alert), 历史记录(History Record), 全屏编辑(Ctrl+E)
- Code Editor:** Displays the Java code for Hello.java.
- Terminal:** Shows the command-line interface with multiple dollar signs (\$) and the printed output "不行，不能使用".

图 10

上方就是一个最简单的判断语句的使用方式，如果 age 变量小于 18，则执行上方的代码，显示“不行，不能使用”，不然就执行下方的代码，显示“可以使用”。

另外，这里使用了字符串的概念来显示文字，使用方法是：

```
System.out.println("文字");
```

TIPS:记得文字作为字符串要用英文半角引号引起起来，不然编译会出错的。

TIPS:不要使用==来测试浮点数，因为这样可能会有 round-off error 问题出现而得不到需要的结果。

3.逻辑运算符

运算符	含义	示例
!	非	if(!exist)
&&	且	if(x > 5 x < 10)
	或	if(age < 10 age > 60)

非就是把真的变成假的，假的变成真的。

e.g.

当文件找不到时 exist 变量的值会是 false，此时如果想执行代码可以使用

```
if (!exist) {
    //代码
}
```

当然这个代码等效于下面的代码：

```
if(exist) {
} else{
    //代码
}
```

且就是要两个都是真结果才是真。

```
if(a && b) {
    //代码
}
```



}

等效于：

```
if (a) {
    if (b) {
        //代码
    }
}
```

或是两个有一个是真结果就是真。

```
if (a || b) {
    //代码
}
```

等效于：

```
if (a) {
    //代码
}
if (b) {
    //代码
}
```

4. 赋值运算符

运算符	含义	示例
=	赋值	x = 2
+=	相加然后赋值给左端	x += 1
-=	相减然后赋值给左端	x -= 1
*=	相乘然后赋值给左端	x *= 2
/=	相除然后赋值给左端	x /= 2
%=	除余然后赋值给左端	n %= 2

e.g.

```
int a = 10;
```

```
a = a + 10;
```

```
// a 的值是 20
```

上方的代码相当于下方的

```
int a = 10;
```

```
a += 10;
```

```
// a 的值是 20
```

是不是变得更简洁了？

还有更简洁的一种，如果要 $a = a + 1$ ，除了 $a += 1$ 以外，还可以使用 $a++$ 和 $++a$ ，这两种知道意思即可。 $a++$ 和 $++a$ 的区别不在 ap cs java subset 当中涉及，我们只需要知道这两种都表示 a 自增 1 即可。

多重赋值行不行呢？可以

```
int a, b, c=1;
```

```
/*这里相当于
```

```
int a;
```

```
int b;
```

```
int c = 1;
```

```
*/
```

```
a = b = c;
```

```
//结果是 a、b、c 都是 1
```

二、数值的储存

1. 整数

整数是以一串比特 (bit) 的形式储存的，其最高位是符号位，其他位是数值位，最高位 0 代表正数，1 代表负数。不同的整数类型的比特数量不同，AP 中的 int 用 4 个字节 (32 bits, 一个字节内有 8 个 bit) 来储存整数。这样的整数类型属于有符号整数，最大值是 $2^{31}-1$ ，最小值是 -2^{n-1} 。对于正数，就可以直接按照高中数学教的方法进行二进制转换了。

以这样的方式储存可以保持原来的数值，不会丢失数值内容。



2. 浮点

double 使用 8 个字节。

在 Java Virtual Machine(JVM) 中浮点类型的存储方式是：

$$\text{sign} * \text{mantissa} * 2^{\text{exponent}}$$

这么储存数据，可能会有精度损失，因为我们平常所存储的很多小数不能用二进制完整表示。因此：

e.g.

$$0.1 * 2^6 = 0.1 + 0.1 + \dots + 0.1 \text{ (26 项)}$$

尽管在数学上我们知道左右两端的大小是一样的，但是仍然会出现不等的现象是由于 round-off error。并且 Java 中如果一个运算不能产生一个已定义的数值，那么就会得到一个 NaN(Not a Number)。

因此，我们需要掌握一种方法来比较两个 double 类型数据是否大小相同。我们利用数学上的一个性质：若两数之差为 0，则两数字相等。对于 double 类型来说，由于浮点的存储在两个数字有可能会在许多位置后才产生差值。因此在 Java 编程中，若想比较两个 double 是否相同，就采取两数之差小于一个极小值即可。即： $\text{double1} - \text{double2} < 0.0001$ ，且 double1 大于 double2 。若该条件返回值为 true，就可以认为 double1 和 double2 是相等的。

如果一个操作得到无限大或者无限小，那么它就会获得 Infinity 或者 $-\text{Infinity}$ 。

三、不同进制数值的表示

1. 十六进制

十六进制的表示方式：0xxx 或者 0XXX，不区分大小写，xx 代表的是 0~9、A~F 的数字，XX 中 A~F 对应 10~15，转换成十进制的方式是：

$$\text{最高位} * (16^{(\text{最高位数}-1)}) + \text{第二高位} * (16^{(\text{最高位数}-2)}) + \dots + \text{最低位} * (16^0)$$

e.g.

$$0xC2A = C * (16^2) + 2 * (16) + A = 12 * 16 * 16 + 2 * 16 + 10 = 3114$$

事实上，不光是 16 进制转换为 10 进制可以利用此方法。任何进制转换为 10 进制时都可以利用上述方法来转换。

2. 十进制

在需要时直接写即可

3. 十进制转其他进制

当我们需要将十进制的数字转换为其他进制的数字时，我们可以使用短除法。

短除法其实是一系列除法，将一个数用整数除法除以目标进制的数（二进制就除以 2），所取得的余数就是目标进制数中的一位。而除法运算的结果被作为下一次运算的被除数，直到结果为 0 则停止运算。最后读数字时要倒着读，就是最后得到的那位数字放在最前。

e.g.

将 13 转换为二进制。

首先在纸上写上 13，然后在左边写 2（转换为二进制），画上短除法的符号。在右边求出 $13/2$ 的余数，这里是 1，在横线下方写上 $13/2$ 的值，这里是 6（这里是整数类型除法）。以此重复知道最后除法结果为 0 为止（如这里 $1/2$ 结果为 0）。然后从下到上读，就是二进制的 13 了，这里是 1101。

	2		13	1
	2		6	0
	2		3	1
	2		1	1
			0		



四、优先级

1. 算数运算的优先级如下，a 代表最高优先级，c 代表最低优先级：

- a) 括号，从内层开始
- b) *, /, %
- c) +, -

2. 总体优先级，a 优先级最高，h 优先级最低：

- a) ()
- b) !, ++, -
- b) *, /, %
- c) +, -
- d) <, >, <=, >=
- e) ==, !=
- f) &&
- g) ||
- h) =, +=, -=, *=, /=, %=

五、控制结构

1. 决定控制结构

刚才

```
if (...) {  
} else {  
}
```

结构已经大体涉及过了，以后会经常用到，除此之外还有 switch 语句等，但是 AP 不会考。

如果只有一句代码，则不一定要用“{}”来包含代码

```
if (...)
```

 代码 1;

```
else
```

 代码 2;

2. 多个嵌套的 if 语句

```
if(布尔表达式 1) {
```

```
    if(布尔表达式 2)
```

语句；

```
}
```

其实也就相当于

```
if(布尔表达式 1&&布尔表达式 2)
```

语句；

TIPS:此时 else 后面连接的语句是和最近的 if 配对的，和代码的排版缩进无关。

3.拓展 if 语句

我们还能使用 else if 语句在 else 中再进行 if 判断执行指定的代码。

```
int a = 1;//change it
if(a == 1){
    System.out.println("One");
}else if(a == 2){
    System.out.println("Two");
}else if(a == 3){
    System.out.println("Three");
}else{
    System.out.println(a);
}
```



The screenshot shows a Java code editor with the file `Hello.java` open. The code contains a switch statement that prints "One" when `a` is 1. The output window shows the program's execution, starting with several dollar signs indicating the prompt, followed by the word "One".

```

1 public class Hello {
2     public static void main(String[] args) {
3         int a = 1; //change it
4         if(a == 1){
5             System.out.println("One");
6         }else if(a == 2){
7             System.out.println("Two");
8         }else if(a == 3){
9             System.out.println("Three");
10    }else{
11        System.out.println(a);
12    }

```

Execution buttons: 运行 (Run), 重置 (Reset), 变量输入 (Variable Input), 提示 (Prompt), 历史记录 (History Record), 全屏编辑 (Ctrl+E) (Fullscreen Edit).

图 11

六、循环结构

很多时候只执行代码一次是没有办法达到目的的，因此我们还需要循环结构。

1. for 循环

for 的用法是：

```
for(初始条件;终止条件;更新语句) {
    //循环内容
}
```

for 的运行方式是：

a)先运行初始条件，然后测试终止条件，如果终止条件不符合 (false) 则跳出循环继续执行。

b)如果终止条件符合 (true) 则继续运行，执行循环内容，然后运行更新语句。

c)重复执行 b)直到循环结束

e.g.

```
for( int i = 1; i < 5; i ++)  
    System.out.print(i + " ");
```

上面的代码会输出 1 2 3 4 。

TIPS: 初始条件内所指定的变量不应该在循环内容里面修改，并且在初始条件里面定义的变量是有范围的，只能在循环内部和条件下进行访问。

2. for each 循环

这种代码用于循环访问一个数组或者集合中的所有元素。用法如下。

```
for(SomeType element : collection){  
    //语句  
}
```

TIPS: for-each 循环不能被用来替换或删除元素，同时不显示 index

注：详细信息请看后文讲 array 的地方

3. while 循环

```
while(布尔测试条件){  
    //循环内容  
}
```

while 循环会先判断条件，如果符合 (true) 就会继续执行循环内容，不符合就会跳出循环执行下面的内容，执行完循环内容后会返回来判断条件，如果符合的话接着执行循环内容，一直重复知道条件不符合 (false) 为止。

七、函数（方法）

其实我们刚才调用的 System.out.println 就是一个函数，它的作用在于输出一定内容，然后换行。

函数（方法）在数学上的概念是数的一一对应，那在计算机科学上，我们可以把函数的



调用过程理解成传入一个或者多个参数 执行某些指令 再返回一个值(这个值可以是 Object) 的过程。

函数这么定义。

e.g.

```
public static int func() {
    //代码
}
```

public 指的是对于其他类也能调用，不只是本类的本个实例。static 等概念就涉及到类比较深了，大家学类的时候再看。

带参数的函数。

e.g.

```
public static int func(int a,int b) {
    //代码
}
```

调用上面定义的两个函数。

e.g.

```
func(); //调用的是没有参数的那个函数，没有使用返回值
System.out.println(func(1,2));
//调用的是有两个参数的函数，使用了返回值
```

八、错误(error)和异常(exception)

异常是一种 Java 程序在执行过程中抛出的错误情况。比如计算机运算出 1/0 就会抛出 ArithmeticException ，如果拿了负数作为 Array 的 index ，那么就会抛出 ArrayIndexOutOfBoundsException 异常。

异常可以被 catch 住，如果是没有 catch 住的异常就是 unchecked exception ，也就是说写代码的人没有设计如何处理这个异常，这样可能就会导致程序停止运行，如果这是你的程序，那么你可能需要改一下程序代码了。

还有一种就是被 catch 住的异常，也就是 checked exception，这类异常会得到处理，危害就没有上述的那么大，checked exception 不在 AP 范围内。

异常
ArithmeticException
ClassCastException
NullPointerException
ArrayIndexOutOfBoundsException
NoSuchElementException
IllegalStateException
IllegalArgumentExeception

以上就是 AP 计算机科学 A 需要掌握的异常了，我们不需要知道用 throw 方法抛出异常，但是需要知道这些异常产生的原因以及什么时候会抛出异常。



Chapter 2 : Java 环境语言 : Practice

1. What will be output by the following statement?

- ```
int age = 1;
if(age > 10){
 System.out.println(age);
}
```
- (A)1  
 (B)10  
 (C)no output  
 (D)unknown

2. What will be output by the following statement?

- ```
int age = 10;
age++;
if(age > 10){
    System.out.println(age);
}
```
- (A)12
 (B)11
 (C)no output
 (D)10

3. What will be output by the following statement?

- ```
System.out.println(5+3<6-1);
```
- (A)4  
 (B)5  
 (C)6  
 (D)false

4. What will be output by the following statement?

- ```
int i = 7;
if(i>0)
    if(i%2 == 0)
        System.out.println(i);
    else
        System.out.println(i+" is not positive");

```
- (A)7
 (B)1
 (C)no output
 (D)7 is not positive

5. What will be output by the following statement?

int age = 1;

```
int b = 2;  
b = age;  
b = 3;  
System.out.println(age);  
(A)1  
(B)3  
(C)no output  
(D)unknown
```

6. Which of the following pairs of declarations will cause an error?

I double x = 14.1;
int y = x;
II double x = 12.3;
int y = (int)x;
III int x = 14;
double y = x;
(A) None
(B) I only
(C) I and II
(D) I,II and III

7. Read following code

```
double a = 13/10;  
System.out.println(a);
```

The answer is 1 but the programmer intends the output to be 1.3.

Which of the following replacements for the first line of code will not fix the problem?

(A) double a = 13.0/10;
(B) double a = 13/(double)10
(C) double a = (double)13/10
(D) double a = (double) (13/10)

Follow us at WeChat: testdaily, and chase your dream together with more than 100,000 friends.

请关注微信公众号: testdaily，并回复关键字「答案」查看**本手册所有练习答案** ~





第三章 标准类(I)

Part I 类

类是什么后面会介绍，现在只需要知道是一个包含函数（方法）、变量的对象即可，只要不是原始数据类型的变量都是类的对象。

Part II String 类

一、字符串介绍

1.字符串

```
""  
"5920"  
"5\n2\n1"
```

这三个 String，显示出来分别是：

第一个是空字符串，什么都没有

第二个：

5920

第三个：

```
5  
2  
1
```

其中，\n 是换行符，读取到这个时计算机会直接换行并继续读取。

大家可能会问了，如果我要输出“5\n2\n1”怎么办呢？

那么就涉及到转义的概念了。

2.转义

如果想输出\n、“或\其实就需要用到转义了。

如果我们想输出”（双引号），我们需要在字符串内写上\"，这样就代表了双引号。

如果我们想输出\，我们就需要\\.因为计算机读取到时就会识别我们的转义字符。

二、String Objects

1. 初始化

我们可以用这样的方法初始化一个 String object

e.g.

```
String s = "aaa";
```

同时我们也可以使用这种方法初始化

e.g.

```
String s = new String("aaa");
```

2. 连接 String Objects

使用“+”运算符来连接 String Objects。

TIPS: String Object 是不可变的，使用“+”来连接意味着创建一个新的字符串，它的内容是原先两个字符串的内容连接起来的结果。

e.g.

```
String s = "aaa";
String t = "bbb";
System.out.println(s+t);
```



```
1 public class Hello {  
2     public static void main(String[] args) {  
3         String s = "aaa";  
4         String t = "bbb";  
5         System.out.println(s+t);  
6     }  
7 }
```

运行 重置 变量输入 提示 历史记录 全屏编辑(Ctrl+E)

```
$  
$  
$  
$  
$  
$  
$  
$  
$  
aaabbb  
$
```

图 1

我们也能将字符串和数字通过“+”运算符相连接：

e.g.

```
String s = "aaa";  
int t = 1;  
System.out.println(s+t);
```



图 2

int 被转换成了 String 再和前面的字符串相连接。

boolean、double 也类似，就不再举例。

3. 比较 String Objects 是否相同

前面说了，对于 `Object` 类，其 `equals` 方法是用来比较参数和方法所属变量是不是属于一个 `Object`。

然而 String 把它 override 了，因此对于 String，equals 方法判断的是它们内容是否相同，而不是是否是同一个 object。

因此如下代码输出 true :

```
System.out.println(new String("Hey").equals(new String("Hey"))));
```



```

1 public class Hello {
2     public static void main(String[] args) {
3         System.out.println(new String("Hey").equals(new
4             String("Hey")));
5     }

```

\$
\$
\$
\$
\$
\$
\$
\$
\$ true \$

图 3

4. 比较 String Objects 的字典序

String 的 compareTo 方法可以比较 String 对象的字典序，就是在字典中的先后。

s1.compareTo(s2) < 0 则 s1 在前

s1.compareTo(s2) > 0 则 s2 在后

s1.compareTo(s2) == 0 则 s1 和 s2 是相同字符串

TIPS:Java 是会识别大小写的。

Java 会把字符串的 ASCII 值拿来对比，所以字符与数字可以比较。

在 ASCII 中，数字<大写字母<小写字母。也就是数字会比其它的字符在字典序中早出现。同时，开头一样结束早的字典序也比较靠前。

e.

String s1 = "HOT";

```
String s2 = "HOTEL"  
String s3 = "dog";  
  
s1.compareTo(s2); //true, s1 先结束  
  
s1.compareTo(s3); //false, "H"比"d"出现得早
```

TIPS: 不要使用`==`来判断两个 String 内容是否相等，因为 Java 可能会把两个不同但是内容相同的 String 通过优化变为一个地址，而这样`==`就会变为 true，在其他时候可能是 false。正确的做法应该是使用 equals 方法。

5. 其他 String 类的方法

a)

```
str.length();
```

返回 String 的长度（字符多少）

b)

```
str.substring(aa);
```

返回 str 的子字符串，从 aa 这个位置开始，aa 是 int 类型的，从 0 开始。

c)

```
str.substring(aa, bb)
```

和上面的函数差不多，从 bb 这个 index 之后是你不想要的字符。

TIPS: 这个函数可能会抛出 StringIndexOutOfBoundsException

d)

```
str.indexOf(s)
```

返回第一次出现 s 的位置。

如果 s 没有在 str 中出现，那么就返回 -1。



Part III Math 类

一、介绍

这个类里面有各种常见的数学方法和常数（如圆周率 pi，自然对数的底数 e），具体见下文所示。

二、需要知道的函数

1. 绝对值

a)

```
int Math.abs(int x)
```

返回 x 的绝对值

TIPS: 这里函数原型的写法中，前面 int 代表返回值，括号里面的是参数，int 类型是指参数是 int 类型的，内部名称为 x，Math 是类名，abs 是方法名

b)

```
double Math.abs(double x)
```

返回 x 的绝对值

2. 幂

```
double Math.pow(double base, double exp)
```

计算 base^{exp}

参数要求： $\text{base} \geq 0$ 时 $\text{exp} > 0$ 或者 $\text{base} < 0$ 时 exp 是整数

TIPS: 这里的函数原型中有两个参数，调用的时候用英文半角逗号分隔调用的参数

3. 平方根

```
double Math.sqrt(double x)
```

求 x 的平方根， $x \geq 0$

4. 随机数

```
double Math.random()
```

返回一个随机数 r， $0.0 \leq r < 1.0$

三、常数

```
pi Math.PI  
e Math.E
```

四、特殊说明

Math 类没有实例变量，所以可以通过 Math.函数名的形式调用 Math 类的函数。

五、随机数

1.从 0 到 1 的随机数

```
double r = Math.random();
```

通过这样我们就得到了一个随机数 r , $0.0 \leq r < 1.0$

2.指定范围的随机数

接着进行运算，我们可以通过以下方法求出 $\text{lowValue} \leq x < \text{highValue}$ 的随机数

```
double r = (highValue - lowValue) * Math.random() + lowValue;
```

3.随机整数

如何获取一个 0 到 k-1 之间的随机整数？

使用如下方法：

```
int num = (int)(Math.random() * k);
```

如果要产生一定范围内的整数，也是一个道理

```
int num = (int)((highValue - lowValue) * Math.random() + lowValue);
```



Chapter 3 Practice

1.

Consider the declarations:

```
String s1 = "We have 25 students in our class.";
String s2 = "We have 25 students" + " in our class.";
int intNum = 25;
String s3 = "We have " + intNum + " students in our class.";
double doubleNum = 25;
String s4 = "We have " + doubleNum + " students in our class.;"
```

Which String doesn't represent "We have 25 students in our class."?

a)s1

b)s2

c)s3

d)s4

2.

What is the result of the following codes?

```
String s1 = "Hello";
String s2 = new String("!\n");
String s3 = "Across the wall we can reach every corner in the
world.";
System.out.print(s1+s2+s3);
```

a) Hello! Across the wall we can reach every corner in the world.

b) Hello Across the wall we can reach every corner in the world.

c) Hellofalse Across the wall we can reach every corner in the world.

d) Hello!

Across the wall we can reach every corner in the world.

3.

Consider the following declarations:

```
String s1 = "hello";
String s2 = new String("hello");
String s3 = "Hello";
String s4 = s1;
```

Which of the following expression evaluates to true?

- a)s1.equals(s3)
- b)s1.equals(s2)
- c)s3.equals(s4)
- d>All are false

4. Suppose that strA = "Potato", strB="potato", and strC="photo". Given that "A" comes before "a" in dictionary order, which is true?

- a)strA.compareTo(strB) < 0 && strB.compareTo(strC) <0
- b)strA.compareTo(strB) < 0 || strB.compareTo(strC) <0
- c)strC.compareTo(strA) < 0 && strA.compareTo(strB) <0
- d)strB.compareTo(strA) < 0 && !(strA.equals(strB))

5.

Here is a declaration of a double number



```
double x = -2.35;
```

After which of following codes it would transfer to the value of 4.

- a)int n = ((int) x) + 5;
- b)int n = (int) Math.abs(x);
- c)int n = (int)(x + 6);
- d)int n = (int) x + 6;

6.A program simulates to select one of 25 students in a class to sing a song. A number from 1-25 represent different students in this class. Which of the following statements store the result of the selected student?

- a)int num = (int) (Math.random() * 25) +1;
- b)int num = (int) (Math.random() * 25) ;
- c)int num = (int) (Math.random() * 26) ;
- d)int num = (int) (Math.random() * 26) +1;

Follow us at WeChat: testdaily, and chase your dream together with more than 100,000 friends.

请关注微信公众号: testdaily，并回复关键字「答案」查看**本手册所有练习答案** ~



第四章 Array 和 ArrayList

Part I 一维数组

一、数组定义

数组是一个对象，将相同类型的一列类型数据放在这一个对象之中。

二、数组的定义、初始化、读取、修改和获取长度

数组的定义方式是。

e.g.

```
double[] data;
```

或

```
double data[];
```

对于上面定义的数组，我们可以对其初始化，定义长度是 25。

e.g.

```
data = new double[25];
```

我们同样可以把定义和初始化合起来。

e.g.

```
double[] data = new double[25];
```

我们甚至可以在定义和初始化的时候往里面填数值。

e.g.

```
int[] data = {1, 2, 3};
```

这个时候长度自动识别了。就不需要再写，这种初始化方法叫 initializer list。

读取、写入数组可以通过以下方式进行。

```
arrayName[index]
```

index 是从 0~数组内元素的数量 -1 的整数，超出范围会出现异常，抛出

ArrayIndexOutOfBoundsException。

e.g.



如下代码输出

Hello 1

Hello 2

Hello 3

e.g.

```
int array[] = {1,2,3};
for(int i = 0;i < 3;i++)
    System.out.println("Hello "+array[i]);
```

数组的长度是固定的，想要获取数组长度，可以如下操作。

e.g.

arrayName.length

TIPS: **数组的长度不用()，字符串的长度要加()**

e.g.

```
//String
String str = "Tony";
System.out.println(str.length());
//Array
int[] arr = {1,2,3};
System.out.println(arr.length);
```

三、数组的作用

e.g.

要表示 10 个学生的身高，我们可以这么做：

```
double student1Height = xxx;
double student2Height = xxx;
...
double student10Height = xxx;
```

然而，如果有 100 个学生这个代码就要长到 100 行了。

使用数组就没那么麻烦。我们可以这么做：

```
double studentHeight[] = {xxx,xxx,...,xxx};
```

当然，如果你认为数组完全可以用多个变量代替，那么你就会发现下面的事情不用数组很难

做到：

```
double studentHeight[] = {xxx,xxx,...,xxx};
for(int i = 0; i < 100;i++){
    System.out.println(stndentHeight[i]);
}
```

四、遍历数组

1. for 循环遍历

```
for(int i = 0; i < array.length; i++) {  
    //对 array[i] 操作  
}
```

2. for-each 循环遍历

e.g.

```
int array[] = {1, 2, 3};  
for( int i : array){  
    System.out.println(i);  
}
```

这样就会把数组中的内容都输出出来。

TIPS: 在 foreach 循环中不能增加或者减少数组的内容，不然可能会导致循环出现问题。

五、把 Array 当做参数传递

Array 被当做一种 Object，他在被传递的时候传递的是 Reference 而不是值（具体后面会提到），所以 Java 不会把 Array 复制一份。传入后修改 array 的内容会导致原 array 变量读取出的是修改后的内容。

e.g.

```
// 我们这里有一个函数  
public static void changeArray(int[] b) {  
    for(int i = 0; i < b.length; i++)  
        b[i] += 3;  
}  
  
// 在主函数里  
  
int[] array = {1, 2, 3, 4};  
changeArray(array);  
System.out.print("The changed array is");  
for(int n: list) System.out.print(n + " ");  
  
// System.out.print 也是输出用的函数，只是不换行
```



结果会输出

The change array is 4 5 6 7

Part II Array List

一、 ArrayList 介绍

ArrayList 也是一种储存一组数据的方式，与 Array 不同，ArrayList 有以下优点：

- 1.ArrayList 长度可变，而 Array 长度固定。
- 2.在 ArrayList 中最后一个存储位置一定是 `list.size()-1`，在 Array 中数据可能是部分填满的，因此程序员必需持续跟踪最后一个使用的储存位置。
- 3.在 ArrayList 中，你可以通过一条语句插入或删除元素，元素会自动移动，然而在 Array 中插入或删除元素会需要你用代码移动剩下的元素。

二、 集合 API

ArrayList 类是一种集合 API(Application Programming Interface)，是一个 Java 提供的库，大部分 API 都在 `java.util` 里。

三、 List<E> 接口

`ArrayList<E>`实现了 `List<E>`接口（可以当成一种 `List <E>`），就是一列类型为 E 的元素，并允许重复。

`List<E>`可以让你：

1. 使用整数 `index` 来访问任何位置的元素
2. 在 `list` 中的任何地方插入元素
3. 遍历所有元素

四、 List<E>的方法

1. `boolean add (E obj)`

在 `list` 的末尾添加 `obj`，总是返回 `true`，如果元素不是 `E` 类型的，抛出 `ClassCastException`

2. int size()

返回 list 中的元素数量

3. E get(int index)

返回在 index 的元素

4. E set(int index, E element)

把在 index 位置的元素换成 element , 返回原先在 index 的元素 , 抛出 ClassCastException 如
果特定元素不是 E 类型的

5. void add(int index, E element)

TIPS: void 说明这个函数无返回值

在 index 处加入 element , 把 index 处及其之后的元素都会向后移动 1 个位置。 List 的大小
增加 1。

6. E remove(int index)

删除在 index 位置的元素 , 返回原来 index 位置的元素。 在 index 位置后的元素会向前移动
1 个位置。 List 的大小减一。

五、ArrayList<E> 类

1. 操作

在末尾对 ArrayList 进行操作很有效率 , 在中部操作 ArrayList 会造成元素移位。

2. ArrayList<E> 方法

```
ArrayList<E> name = new ArrayList<E>();
```

创建一个空 list。

TIPS: 如果 $index < 0 \text{ 或 } index > size()$, 那么此时操作会抛出 IndexOutOfBoundsException 。 add 操
作除外 对它来说在末尾添加元素是可以的 此时如果 $index < 0 \text{ 或 } index > size()$ 会抛出 exception。



Part III 二维数组

一、使用范围

二维数组经常用来代表矩阵等数据结构。

二、定义方式

如下代码可以。

```
E[][] name = new E[a][b];
```

E 是类型名，a、b 是数组的长度。

三、访问方法

name[r][c]，r 代表 row，c 代表 column。

四、遍历二维数组

遍历二维数组的方式和一维数组差不多，只需要两个循环相互嵌套即可

下面的例子会把 n 的所有数值改为 100。

e.g.

```
Integer[][] n = new Integer[10][10];
for(int i = 0; i < n.length; i++){
    for(int j = 0; j < n.length; j++){
        n[i][j] = 100;
    }
}
```

Chapter4 Practice

1. Which of the following correctly initializes an array `arr` to contain four element with value 1?

I `int[] arr = new int[4];`
II `int[] arr = {1,1,1,1};`
III `int arr = new int[4];`
`for(int i = 0; i < arr.length;i++)`

`arr[i] = 1;`

- a) I only
- b) III only
- c) II and III only
- d) I, II, and III

2. Refer to the following code segment. Array `arr` contains `int` values.

```
int sum = arr[0], i = 0;  
while(i < arr.length){  
    i++;  
    sum+= arr[i];  
}
```

Which will be the result of executing the segment?

- a) Sum of the array will be stored in sum;
- b) A run-time error will occur
- c) Sum of the array will be stored in the arr[0]
- d) An infinite loop will occur.

3. The following code segment will print how many SMALL:

```
for (int i = 2; i <= k; i++)  
    if( arr[i] < someValue)  
        System.out.print("SMALL");
```

- a) 0
- b) 1
- c) k - 1
- d) k - 2

4. Consider these declarations:

```
List<String> strList = new ArrayList<String>();
```

```
String ch = " ";
```

```
Integer intObj = new Integer(5);
```

Which will cause an error?

- a) `strList.add(ch);`
- b) `strList.add(ch+1);`
- c) `strList.add(intOb+1);`
- d) `strList.add(new String("Hey"));`



5. Consider the following code segment, applied to list, an ArrayList of Integer values.

```
int len = list.size();
for(int i = 0;i < len; i++)
{
    list.add(i + 1, new Integer(i));
    Object x = list.set(i, new Integer(i + 2));
}
```

If list is initially 6 1 8, what will it be following execution of the code segment?

- a) 2 3 4 2 1 8
- b) 2 3 4 6 2 2 0 1 8
- c) 2 3 4 0 1 2
- d) 2 3 4 6 1 8

Follow us at WeChat: testdaily, and chase your dream together with more than 100,000 friends.

请关注微信公众号: testdaily，并回复关键字「答案」查看**本手册所有练习答案** ~



第五章 类与对象

Part I 类与对象

一、 对象(Object)

a)对象与类的基本介绍

构成 Java 这个大厦的基石就是我们这一章：类与对象。它的**面向对象**这一个重要的特征，确定了 Java 的基本逻辑与思想，同时也是它最为显著的一个特性。Java 把这世界一切都能看成对象，即**万物皆是对象**。这些对象会根据其属性或者其他其特点进行分类。这种分类就的集合就被称为类，关于类的详细说明先按下暂且不表。

b)面向对象

面向对象式编程翻译过来就是“我们关注着一个对象”，在计算机里面，就是我们**关注对象所具有的的事物信息**。

e.g. 小 L 在 SAT 里面取得了 1601 分的成绩，觉得很高兴，想买个手机奖励自己。于是他去售货台跟柜员说了如下信息：

“我要一个 CPU 快的，屏幕大的，能够发短信打电话的。”

这个时候小 L 并没有见到一个实体的对象，但他把自己所希望买到的手机信息说了出来，他就关心上面的这些信息。

这个时候，售货员拿来了一个 iPhone。说这就是你想要的。

这个 iPhone 就是我们的**对象**，它是一个具体的实物，而不再是一些虚无缥缈的信息。

上面小 L 买手机的这个过程，就叫做面向对象。我们需要一个对象（iPhone），我们关注着这个对象所具有的信息（CPU 快，屏幕大，能打电话发短信）。



c)对象所具有的信息

i. 属性

属性可以理解成这个对象有什么。上面那个例子之中，手机的屏幕，CPU 就是手机的属性。

ii. 方法

方法是对象所执行的操作，也就是能干什么。在上面的例子中就是打电话、发短信。

二、类(Class)

a)类是什么

i.类是对象的特性。

ii.类是**相同属性和方法的一组对象**的集合。

b)类和对象的关系

类是抽象的概念，仅仅是一种模板。在客观世界中，它实际上是不存在的。但是根据这个类的特性，我们能够产生具体的对象。

e.g. “手机”类能够产生 iPhone7，三星 Note7 这些具体的对象。

c)定义类

i.定义类名

用“`public class+类名`”去定义一个类名

ii.编写类的属性(state)

“属性类型(String, int, double, etc)+属性名字”

iii.编写方法(behavior)

关于方法的详细定义与写法会在后文中讲到。这里现有一个概念即可。

e.g. 尝试阅读下列手机类的代码

```
//Define a class
public class CellPhone
{
    //Define states
    double screen; //first state
    double cpu;    //second state
    double memory; //third state

    //Define behaviors
    void call()    //first behavior
    {
        System.out.println("This phone can call someone!");
    }
    void sendMessage() //second behavior
    {
        System.out.println("This phone can text to anyone you like!");
    }
}
```

上面这个就是一个简单的手机类。它定义了手机有屏幕、cpu、内存组成。同时也
可以打电话、发短信。

c) 使用对象

i. 创建对象

类名 对象名 = new 类名(); //这个括号中可以填写东西，具体的填写内容将在方
法中的 constructor 详细解释。这是我们创建对象的一个最基础的版本。

e.g. CellPhone iPhone = new CellPhone();

这个也叫做类的实例化。我们 new 了一个类名，就得到 iPhone 这个对象，而这
个 iPhone 对象的类型就是 CellPhone 类型。

当我们 new 出来这个对象后，对象名所代表的就是这个对象所具有的信息。那么
我们对对象名去进行的操作就是我们对对象本身进行的操作。



i.i. 使用对象

引用对象的属性：对象名.属性

e.g. iPhone.screen = 5.0; //给 screen 属性赋值为 5.0。

引用对象的方法：对象名.方法名()

e.g. iPhone.sendMessage(); //调用 sendMessage() 方法

Part II 方法

一、方法 (method)

方法在编程中是不可或缺的，它是用来解决一类问题的代码有序组合，一种功能性模块。

一般我们的方法会是这样的

```
访问修饰符 返回值类型 方法名(参数列表) {
```

方法体：

```
}
```

对应的英文为

```
accessSpecifier returnType methodName(parameterList)
{
    functions;
}
```

e.g.

```
public void sendMessage() //没有参数没有返回值的方法
{
    /*implementation code*/
}
```

```
public String findPhone(location, ID) //有参数有返回值 String
{
    /*implementation code*/
}
```

a) 访问修饰符

访问修饰符可以修饰**属性**和**方法**的访问范围。其本身是可以省略变成默认值的，

但是在 AP 考试中，我们要求加上访问修饰符。在 AP Java subset 中我们仅仅考察 `private` 和 `public` 两种修饰方法，且深度较浅。对于**所有的方法来说都可以用 `public` 声明**，对于**所有的属性来说都可以用 `private` 来声明**。

i. `private` 这个修饰符所修饰的方法、属性**仅能在本类中使用**。如果在其他类（包括同包、子类）中访问 `private` 变量会在编译时报错误，出现 Compile Error.

为了避免外界程序能够所以调用我们的属性值，我们需要将我们的这些程序给包裹住，让外界不那么轻易地修改。如果要编写一个银行账户 `BankAccout` 的类，我们不用 `private` 去修饰我们的前等敏感信息的话，任何人都有权限去修改我们银行账户的钱，这是极其不安全的，我们要用 `private` 把这些信息（属性）给隐藏住。这就是我们 Java 中封装的概念。

ii. `public` 这个修饰符所修饰的方法、属性可以在定义的类外中使用。

b) 返回值类型

方法返回值的类型，如果方法不返回任何值，则返回值类型指定为 `void`；如果方法具有返回值，则需要指定返回值的类型，并且在方法体中使用 `return` 语句返回值。

e.g.

```
//有返回值类型  
/*some codes that necessary for the program*/  
public String getName() {  
    return name;
```



```

    }

//无返回值类型

public void print()
{
    System.out.println("This method contains no return
value");
}

```

b) 方法名

定义的方法的名字，必须使用合法的标识符。

d) 参数列表

传递给方法的参数列表，写在方法名后的括号中。参数可以有多个，**多个参数间以逗号隔开**，每个参数由参数类型和参数名组成，以空格隔开。根据方法是否带参、是否带返回值，可将方法分为四类：

- i. 无参无返回值方法
- ii. 无参带返回值方法
- iii. 带参无返回值方法
- iv. 带参带返回值方法

二、实例方法(instance method)

我们首先看一下一个简单的银行账户的类 BankAccount

```
public class BankAccount
{
    private String myPassword;
    private double myBalance;

    //constructors
    /*default constructor*/
    public BankAccount()
    {
        /*implementation code */
    }

    /*specific constructor*/
    public BankAccount(String password, double balance)
    {
        /*implementation code*/
    }

    //accessor
    public double getBalance()
    {
        /*implementation code*/
    }

    //mutators
    public void deposit(String password, double amount)
    {
        /*implementation code*/
    }

    public void withdraw(String password, double amount)
    {
        /*implementation code*/
    }
}
```

(出自 Barron , 经过删减)

a) 构造函数 (constructors)

我们知道对于要实例化一个对象的时候，我们要用 new 去实例化一个对象。对于

BankAccount 这个类中，我们用这个语句来实例化这个对象。



```
BankAccount david = new BankAccount();
```

在我们前面的学习中，我们知道 new 后面的 BankAccount() 是一种类名()，但实际上，这种说法是不严谨的。这个 new 后面的东西，应该是我们的**构造方法**。所以正确实例化一个对象对公式应该是

对象类型 对象名称 = new 构造方法(参数列表);

构造方法和我们普通方法的不同就是它可以初始化我们的对象，或者给对象的属性进行赋值。

构造方法的写法最特殊之处，就是其**没有返回值且与类名相同**。



我们把没有参数的构造方法叫做无参构造方法，把有参数的构造方法称之为有参的构造方法。

e.g. BankAccount 类的 //constructors 就是我们的构造方法。其第一段是我们的无参构造方法。事实上，无参构造方法可以在程序中可以不写出来，**系统会自动生成一个无参的构造方法**。

带参的构造方法就是为了让参数传递到对象的属性中。

对于我们 BankAccount 第二个构造方法，就是一种带参的构造方法。我们可以将参数利用这个方法传递到对象中。在我们 BankAccount 类中，我们可以写成

```
public BankAccount(String password, double balance)
{
    myPassword = password;
    myBalance = balance;
}
```

这个就是将我们参数 password, balance 分别传到 myPassword, myBalance

之中。当我们使用含参构造方法 new 一个对象后，这个构造方法就会被调用

```
BankAccount david = new BankAccount("hello", 2333.3)
```

上面这个调用方法就是我们实例化了一个 david 的银行账户，把 hello 这个密码赋给了 myPassword, 2333.3 赋值给 myBalance 中。

我们要注意的是，**当我们指定了一个构造方法（无论是带参还是不带参）后，系统都不会自动添加无参的构造方法了。**

b) 访问函数 (accessor)

访问函数顾名思义，就是访问我们对象的属性。因为我们经常将属性值设置为 private 类型，从客户端 (client) 中无法直接获取得到。所以我们要写一个访问函数，**能够得到对象的属性。**

一般的访问函数会写成 get 函数。想看到对象的属性值，就是 get+ 属性的名字这样的命名方法。而其是拥有返回值，但不修改属性值得一类函数。

在我们的 BankAccount 这个类，我们 getBalance 就是一种访问函数。

```
public double getBalance()
{
    return myBalance;
}
```

这个函数就得到了我们对象属性的 myBalance 值。因为这个函数是在 BankAccount 类里面，所以才可以直接访问我们用 private 修饰的属性值。

c) 修改器(mutator)



修改器其实是我们修改属性的一类方法，也会被叫做 set 方法。通常我们在想修改属性的时候，我们不能够让客户端随意直接修改我们的属性值。我们可以通过设立一些规则让客户端不轻易的修改我们的属性值。这里就用我们的修改器来构造一个渠道，在一定规则内允许客户端去修改我们的属性值。

修改器一般不需要返回值，但是它的根本是对我们的属性值进行修改。通常我们需要使用参数来设立我们的规则。所以常见的修改器是一个有参无返回值的函数。

在 BankAccount 类，我们的 deposit 就是一个 mutator，我们想让密码和账户密码一致的情况下，存储我们一定的钱。所以这个函数实际上会被写成

```
public void deposit(String password, double amount)
{
    if(password.equals(myPassword))
        myBalance += amount;
    else
        System.out.println("Wrong Password!");
}
```

三、静态方法 (static method)

静态方法实际上也被称为类方法(class method)，这个名字更适合我们的静态方法。因为事实上，这些方法并不随着我们的对象的创建才能使用，而是直接在类创建的时候就能够直接通过类的调用使用。

e.g. 在一个手机类中

```
public class CellPhone
{
    private double cpu;

    public double getCPU() {
        return cpu;
    }

    public void setCPU(double c) {
```

```

        cpu=c;
    }

    public static String nothing() {
        System.out.println("I am a static method and I have nothing
to do with object.");
    }
}

```

这个时候，如果我们在一个客户端 CellPhoneTest 类中直接调用我们的 nothing() 是可以的，但是如果调用我们的实例方法就不行了。

```

public class CellPhoneTest
{
    public static void main(String[] args)
    {
        CellPhone.nothing(); // OK, since nothing() is a static
method
        CellPhone.getCPU(); // compile error, getCPU() is an
instance method
    }
}

```

能理解上面的概念最好，AP Java subset 对静态的考试要求较小。我们在 AP 中可以将静态理解为“与众不同的”，或者“无关的”，也就是与对象无关的。

e.g. Which following variables should be declared as static?

- (a) wheels (b) passengers (c) size (d) gpa

这道题答案就选择 d。因为显然 d 和我们 a, b, c 是无关的。后三个和我们的交通工具有关，但是 d 他是一个跟学生成绩有关的变量。所以我们将 d 声明为 static。

Part III this 关键字

this 关键字的作用在 ap 中其实不大，在部分情况下是可以删除的。



当我们在使用一些 IDE 自带的生成方法，尤其是构造器的时候。构造器的参数名称会跟我们属性的名称一样，导致我们后期修改代码的时候很容易造成误解。我们通常只是通过名字来判断这个量是一个属性，还是某个参数，这个时候，如果名字相同很容易造成混淆。那么我们为了避免混淆，就使用 this 关键字。

e.g. 阅读下面这个函数构造器中，属性为 cpu。

```
public CellPhone(double cpu) {
    cpu=cpu
}
```

这个时候，我们很难去区分到底 cpu 是哪个，所以我们要加上修饰符 this 去区分，代表着，虽然名字是一样的，但是我们指得是“这个对象”的属性，而不是参数。事实上，上述代码虽然不会遇到 compile error，但是程序实际上运行的是我们的参数的 cpu 被我们属性的 cpu 赋值，属性本身的值没有改变，得到的是一个 0.0 的值。如果我们要想让程序执行我们想要的结果，我们需要在属性的 cpu 前加一个 this 来指明。

我们对代码进行一下修改就可以

```
public CellPhone(double cpu) {
    this.cpu=cpu;
}
```

这样就可以成功的将我们参数中的值赋给属性了。

Part IV 数据类型

一、原始数据(primitive data)

AP Java subset 中最主要的三种原始数据为 int, double, boolean。每一个都有自己独自的记忆空间。这就意味着其中一个的值改变了，其他的值不会改变。

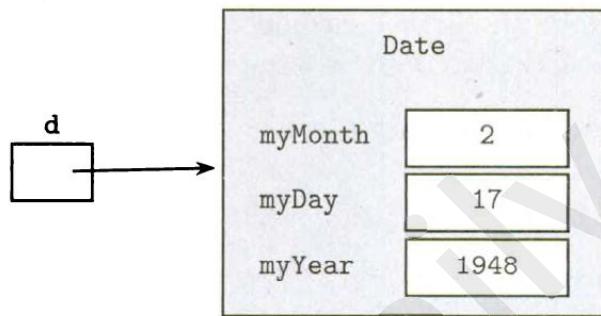
num1	num2
3	3

二、引用数据类型(reference data)

引用数据其实是一种地址。如

```
Date d = new Date(2,17,1946);
```

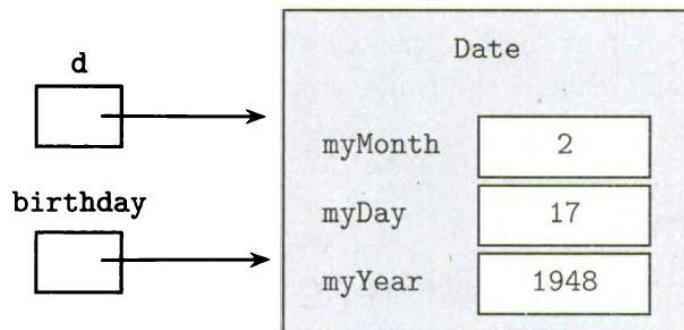
这个时候 d 就是一种引用数据类型，它的类型是 Date.因为我们实际上并没有把我们 Date 的数据存储到 d 里面，而是 d 是一个地址，指向我们的内存中。



如果我们这个时候在声明如下语句

```
Date birthday = d;
```

那么我们就创建了一个 Date 类型的引用数据类型 `birthday`，只不过他和 `d` 是同一个地址，指向了同一块内存。这就意味着如果我对 `d` 所指向的内存进行改动，那么我 `birthday` 所指向的内存也跟着改动。这种改动，通常是用方法进行改动。这会造成一些意料之外的错误。在计算机中，我们把两个指向相同的内存但是名称不同的地址叫做混淆(aliasing).





但是我们要注意的是，如果我们搭配上了 **new** 关键字之后，就意味着我们生成了一个新的内存，开辟了一个新的空间，尽管两个地址所指向的内存具有相同的数据，但是这两个内存是截然不同的，地址也是截然不同的。

如果一个引用数据类型是一个 null 的话，用它去调用函数会抛出一个 **NullPointerException，就是我们常说的空指针异常。**

三、方法参数

方法参数中分为形式参数(formal parameters)，和我们的实际参数(actual parameters)。

e.g.

```
public void withdraw(String password, double amount)
```

这个时候，password, amount 这两个就是两个形式参数，不具有真实的意义，只是一个形式而已。可以把它们想象成为实参保留位置的两个参数

但是当我们对一个对象去调用这个方法的时候，调用时带的有意义的参数就是我们的实际参数。

```
b.withdraw( "YES" ,2333);
```

这里的“ Yes” 和我们的 2333 就是我们的实际参数，因为他们是一个值而不是一个虚有的名字。

四、参数传递

a)原始类型作为参数传递

原始数据，正如其本身，传递的是我们的值。对于一个方法来说，如果原始类型数据作为方法参数的话，会为实参创造一个新的内存。也就是说，任何对**实际参数**的改变都**不会影响到我们程序中原本的值**。

e.g.

```
public class Test{
    public static void foo(int a,int b){
        a++;
        b++;
    }

    public static void main(String[] args){
        int a =6;
        int b =7;
        foo(a,b);
        System.out.println(a+ " " + b);
    }
}
```

这个的执行结果就是

6 7

为什么会和我们预期的不一样呢？就是因为我们在进入到 `foo(a,b)` 这个函数中给 `a,b` 创建了一个新的内存。这就意味着，我们怎么更改我们新内存中的值，也不会对本身我们刚开始声明的 `a,b` 有影响。当我们 `foo(a,b)` 这个函数退出后，我们这个新创建的内存会被删去。所以，我们输出的值就是 `a, b` 原本的值 6, 7.

b)引用类型参数传递

引用类型参数传递是一种地址的传递，所以并没有新内存生成，只是在方法内创建了一个地址指代相同位置的形式参数。我们知道，由于引用数据的特殊性，不管我们是对原始地址本身所指向的内存进行修改还是创建的形式参数的地址进行修改，我们最终都会对地址本身所指向的内存进行修改。

e.g.

```
public static void chargeFee(BankAccount b,String password,
double fee){
    final double MIN_BALANCE = 10.00;
    if (b.getBalance()<MIN_BALANCE)
        b.withdraw(password,fee);
```

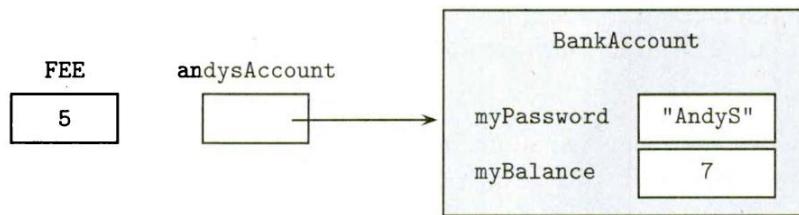


```

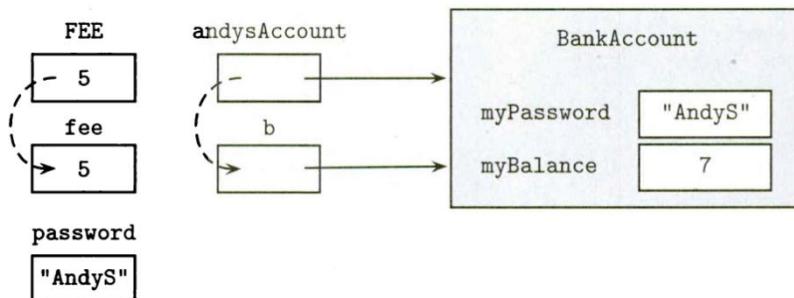
    }

public static void main(String[] args) {
    final double FEE = 5.00;
    BankAccount andysAccount = new BankAccount("AndyS", 7.00);
    chargeFee(andysAccount, "AndyS", FEE);
}
  
```

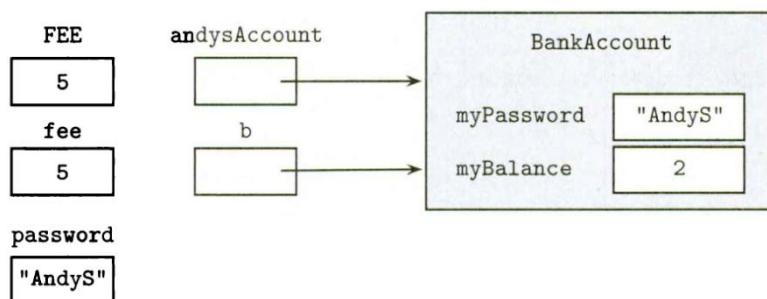
当我们 chargeFee 这个函数被使用前的时候



当我们调用这个函数的时候

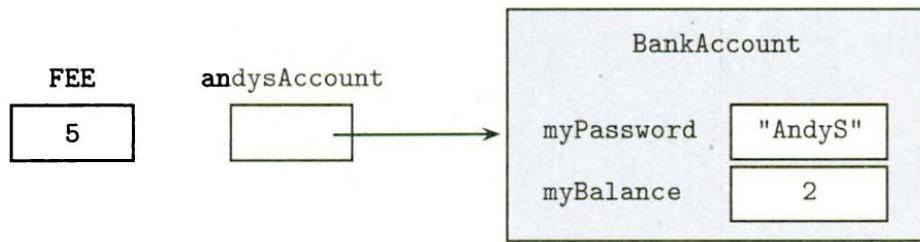


当我们退出这个函数之前的时候，我们对已经对内存改变了。因为我们通过方法中创建的引用类型参数，对内存储存的值进行改变了。



当我们退出这个函数，我们原始数据类型创建的新内存被抹去，新创建的地址

b 同样被抹去（形式参数会在方法结束后被删掉），但是我们的对象已经被更改



Chapter5 Practice

Create a BowlScores Class

Write a class called **BowlScores** that can be used to handle the scoring for each player, in the game of bowling.



Follow us at WeChat: testdaily, and chase your dream together with more than 100,000 friends.

请关注微信公众号: testdaily，并回复关键字「答案」查看**本手册所有练习答案** ~



第六章 继承和多态

Part I 方法概念扩充

一、方法重载 (overload)

方法重载的概念很简单。Java 允许我们使用相同的函数名去处理不同类型的参数。

e.g.

```
//valid  
public void receive(int i) {  
    System.out.println("Received one int data");
```

```

        System.out.println("i="+i);
    }

//valid
public void receive(float f) {
    System.out.println("Received one float data");
    System.out.println("f="+f);
}

//valid
public void receive(String s) {
    System.out.println("Received a String");
    System.out.println("s="+s);
}

//invalid
public void receive (int num){
    System.out.println("Invalid method");
}

```

我们需要注意的是我们第一、二、三个方法都是可以运行的，因为他们的参数类型是

不

同的。但是我们的第四个方法就会出现 compile error，是因为我们的参数类型和第一个参数类型冲突，都是 int 型，尽管变量名称不同。

二、 方法重写 (override)

对于一个已经存在的方法进行重写，就是方法重写。重写的意思就是对方法体的一个修改，让方法更完善或者更新其功能调用方法时会优先调用重写后的方法。方法重写时，我们需要保证返回值类型、方法名、参数类型以及个数都和重写前的方法相同才叫做方法的重写。

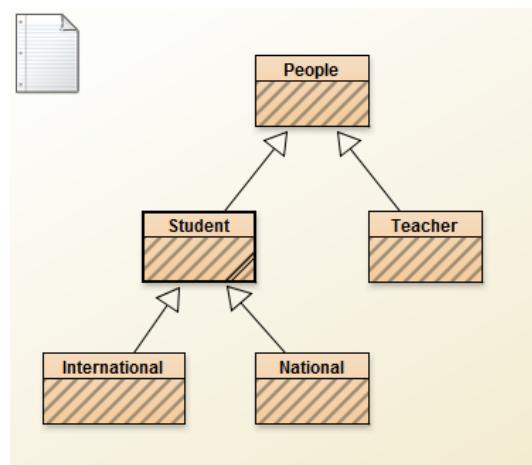
e.g. `toString` 方法实际上是一个 `String` 类的方法，但是我们在使用这个方法的时候，通常会对其原始的功能不够满意，所以我们对 `toString` 方法在我们写类的时候重写 `toString` 这个方法。这就时方法重写的一个最常见的例子。是不是要举例说明一下如 `Integer` 的 `toString` 和默认 `toString` 的区别（虽然我在我的第七章里面说过）



Part II 继承 (inheritance)

一、概念

继承实际上是一种程序的关系。如同字面描述的，他是一种继承关系。就像儿子继承了父亲一样。在程序中，我们将父类程序称为 super class，将子类称为 subclass。



在上面这个图中，People 是 Student , Teacher 的父类。Student 是 International 和 National 的父类。这个图我们在后续会学到，我们称之为 UML 图(UML diagram)。

如何判断是否是继承关系呢，就是利用“是”这种关系，对应的就是“is a/an”的关系。
上面这个图，Teacher “is a” People, International (student) “is a” Student. 就可以判断继承关系了。事实上 尽管我们可以说 Student 不仅仅是一个 People, 还是一个 Citizen，但是事实上，在 Java 中我们仅允许子类继承一个父类，而不能继承多个父类。

二、extends 关键字

如何在代码中继承一个类呢？就利用 extends 关键字去表示。

```

public class Superclass {
    // private instance variables

    //constructors
    //public methods
    //private methods
  
```

```

}

public class Subclass extends Superclass{
    //additional private instance variables

    //constructors
    //additional public methods
    //inherited public methods, which are overridden
    //additional private methods
}

```

在第二个类 Subclass 中，我们使用了 extends 关键字把 Subclass 和 Superclass 的
关

系建立起来。代表着 Subclass 继承了我们的 Superclass。

但是并不是继承就代表着子类继承了父类的全部，仅有一部分是可以被继承的。

**能够被继承的在 AP Java subset 中，仅仅是 public 的方法，其余的一切均不可以被
继承，尤其是父类的构造器是不会被继承的。**

三、super 关键字

super 在对象的内部使用，其意义**代表着父类对象**。其有两种使用方法，在构造器中使
用，以及调用父类的方法。

a) 构造器的使用

有时候，我们子类会和父类的一些参数相同，我们就可以利用 super 关键字将我
们的程序缩短，直接调用父类的无参构造方法 super() 或者 super(参数列表)。

e.g.

```

public class Student{
    private int score;
    private double gpa;

    public Student(){
        score = 0;
        gpa = 0.0;
    }
}

```



```

public Student(int s, double g) {
    score = s;
    gpa = g;
}

public class GradStudent extends Student {
    private double rent;

    public GradStudent() {
        super(); // note1
        rent = 0.0;
    }

    public GradStudent(int s, double g, double r) {
        super(s, g); // note2
        rent = r;
    }
}

```

这两个类就说明了我们 super 的构造方法是如何使用得了。我们的 note1 就是直接调用了父类 Student 中的无参构造方法。而我们的 note2 就是调用了父类有参的构造方法，参数列表的 s , g 对应的 Student 中有参构造方法的两个参数，利用父类的构造方法，传给我们子类的对象属性中。

但是在构造方法中，我们需要注意几个重点：

- i. 子类的构造的过程当中**必须**调用其父类的方法。
 - ii. 如果子类的构造方法中没有显示调用父类的构造方法，则系统默认调用父类**无参**的构造方法。
 - iii. 如果显示的调用构造方法，**必须在子类的构造方法的第一行**。
 - iv. 如果子类构造方法中既没有显示调用父类的构造方法，而父类又没有无参的构造方法，则出现 compile-error
- b) 调用父类方法

调用父类方法时，我们要注意我们**只能**调用父类方法用 public 声明的函数。

不能调用父类用 private 修饰的属性，以及 private 修饰的方法。

e.g.在父类 Student 中存在这样的方法

```
public double getGPA() {
    return gpa;
}
```

在子类中，我们想调用父类中 getGPA 这个方法，我们就可以用 super 关键字来表明我们的函数位置。

```
super.getGPA();
```

但是事实上，我们如果把 super 去掉的话，这个方法也是可以使用的。因为子类继承了父类的这个属性，这个函数可以无条件的被子类所使用。但是如果出现重写的情况下，我们的 super 就变成了一个很有必要的关键字了。

e.g.在父类 Student 中我们写 toString 函数

```
public String toString(){
    String str = "";
    str += "Score:      "+score+"\n";
    str += "GPA:        "+GPA+"\n";
    return str;
}
```

在子类 GradStudent 我们重写 toString 函数的话，就需要 super 来调用父类的 toString 了。

```
public String toString(){
    String str = super.toString();
    str += "rent:      "+rent+"\n";
    return str;
}
```

Part III 多态(polymorphism)

一、概念

多态就是对象的多种形态。他可以是引用的多态，以及方法的多态。

二、引用多态

引用的多态就是父类的引用即可以指向本类的对象，也可以指向子类的对象。



在 Student 和 GradStudent 的例子中就可以体现引用的多态。

我们既可以用父类引用指向本类对象，也是我们最常见的一种

```
Student php = new Student(); // valid
```

也可以用父类引用指向子类对象

```
Student ruby = new GradStudent(); // valid
```

但是需要注意，我们不能用子类引用指向父类对象。这是因为子类中包含父类不存在的

属性和方法，无法向上兼容到父类中。如下列代码就是不可行的：

```
GradStudent java = new Student(); // invalid, compile error
```

三、方法多态

如果我们创建的是父类本类的对象，如 php 对象，我们调用的方法就是本类方法。但如果我们创建的是一个用父类引用指向子类对象的话，调用的方法就是子类重写的方法或者继承的方法。

e.g. 在 Student 和 GradStudent 这两个类中，我们仍然可以完成方法的多态。

在 Student 类中，我们创建一个 status 的方法

```
public void status() {
    //method has not been overridden
    System.out.println("I am a Student!");
}
```

紧接着，我们在 GradStudent 中，重写这个方法

```
public void status() {
    //method has been overridden
    System.out.println("I am a GradStudent!");
}
```

之后，我们在客户端调用利用 php 和 ruby 分别调用 status 这个函数

```
php.status(); //print "I am a Student!"
ruby.status(); //print "I am a GradStudent!"
```

虽然我们调用的是同样的方法名，但是程序自动判断优先调用哪个函数。**父类引用的子类对象会优先调用重写后的方法，而不是父类本身的方法。** 继承也是一个道理，子类中无需重写，直接调用父类继承来的方法。

但是父类引用指向的子类对象是不允许调用父类不存在的方法，如果要强行使用，必须
要将我们的父类引用指向的子类对象进行**向下转型**才可以。

如果我在子类 GradStudent 中写一个 recomLetter 方法的话

```
public void recomLetter() {
    System.out.println("I got a recommendation!");
}
```

那么我在使用 ruby 去调用这个方法的话，就会出现 compile error

```
ruby.recomLetter(); //invalid. Throw compile error
```

四、兼容类型

事实上，我们通过转型的方法，可以允许父类引用的子类对象调用子类独有的方法，只
需要在父类的引用上做一个强制向下转型(downcasting)就可以。也就是

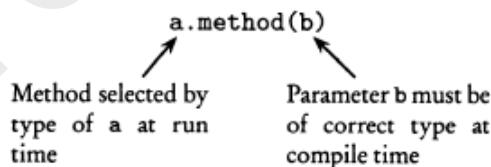
```
(subclass) superClassObj.subclassMethod;
```

e.g. 如果我们想让 ruby 调用 recomLetter 这个方法的话，我们就需要将 ruby 做一个向
下转型就可以了。

```
(GradStudent) ruby.recomLetter(); //valid
```

五、多态方法规则

对于如下方法



存在这条声明: Superclass a = new Subclass();

a)方法 method 必须在 Superclass 中存在，否则需要在调用方法前进行类型转换。

b)如果方法 method 存在多态，在程序运行时会优先调用子类被重写的方法。

c)参数 b 在编译时会检查是否符合父类 Superclass 中 method 中参数 b 的类型，执行
的时候会按照 b 规则去执行方法，有可能因为参数类型的不同而去寻找参数类型相同
的函数。



六、类型转换异常(ClassCastException)

出现这个是因为类型转换出现了错误。若 UnderGrad 继承 Student 类，但是 UnderGrad 和 GradStudent 没有任何联系的话

```
Student u = new UnderGrad();
System.out.println((String) u);    //ClassCastException
                                    //u is not an instance of String
int x = ((GradStudent) u).getID(); //ClassCastException
                                    //u is not an instance of GradStudent
```

Part IV 抽象类(abstract class)

一、语法定义

抽象类使用 **abstract** 关键字修饰，该类为抽象类。

二、应用场景

a)在某些情况下，父类只是知道其子类应该包含什么方法，但是无法准确的知道这些子类如何实现这些方法。换句话说，**抽象类是约束子类必须拥有那些方法，而且不关注子类如何去实现这种方法的。**

b)可以把多个具有相同特征的类中抽象出一个抽象类，以这个抽象类作为子类的模板，给子类一个框架，让子类方便编程的同时也约束了子类的随意性。

三、使用规则

a)**abstract** 定义抽象类。

b)**abstract** 定义抽象方法，而且其没有具体的实现细节，并且以分号结尾。

e.g. public **abstract** void testDaily();

c)包含抽象方法的类一定是抽象类。

d)抽象类中可以包含普通的方法，也可以没有抽象方法。

e)抽象类不能直接实例化一个对象，但是可以利用抽象类的引用去实例化一个子类。

Part VI 接口(interface)

一、概念

接口可以被理解为一种特殊的类，由全局常量和公共的抽象方法组成。

类是一个具体的实现体，接口是定义了某一类必须拥有的方法。接口不关心类的内部数据，也不关系类的实现细节，但是它规定了类中必须拥有那些方法，也就是 `abstract` 方法。

二、接口声明

和类的定义不同，定义接口不再需要 `class` 关键字，而是使用 `interface` 关键字取代。

```
abstract  
[修饰符] interface 接口名 [extends 父接口1,父接口2...]  
{  
    零个到多个常量定义...  
    零个到多个抽象方法的定义...  
}
```

因为其包含了抽象方法，所以我们的接口一定需要 `abstract`。不过好在，如果我们没有写 `abstract` 关键字，系统会默认我们写了一个 `abstract`。也就是说，我们虽然不需要写出来，但是系统中在识别这个接口的时候，一定会带着这个 `abstract`。

我们注意到，接口可以继承，而且可以继承多个父接口。这个和类是不同的，类只允许继承一个父类，但是接口却允许继承多个父接口。

三、接口定义

a)常量

接口中的属性是常量，所以我们要用 `public static final` 修饰，如果没有写，系统也会帮我们自动添加。

b)方法

接口中的所有方法都是抽象方法，所以用 `abstract` 修饰。即使我们没有添加 `abstract`



关键字，系统也会帮我们自动添加。

四、使用接口

一个类可以实现一个或多个接口，实现接口需要用关键字 `implements`。Java 中，一个类只能继承一个父类，不够灵活，通过多个接口实现可以进行补充。

继承父类实现接口的语法为：

```
[修饰符] class 类名 extends 父类 implements 接口1, 接口2...
{
    类体部分 //如果继承了抽象类，需要实现继承的抽象方法；要实现接口中的抽象方法。
}
```

如果要继承父类，继承父类必须在实现接口之前

Chapter6 Practice

```
public class BankAccount{
    private double myBalance;

    public BankAccount() {
        myBalance = 0.0;
    }
    public BankAccount(double balance) {
        myBalance = balance;
    }
    public void deposit(double amount) {
        myBalance += amount;
    }
    public void withdraw(double amount) {
        myBalance -= amount;
    }
}
```

```

        public double getBalance() {
            return myBalance;
        }
    }

    public class SavingsAccount extends BankAccount{
        private double myInterestRate;

        public SavingsAccount() {
            /* implementation now shown */
        }
        public SavingsAccount(double balance, double rate) {
            /* implementation now shown */
        }
        public void addInterest(){           //Add interest to balance
            /* implementation now shown */
        }
    }

    public class CheckingAccount extends BankAccount{
        private static final double FEE = 2.0;
        private static final double MIN_Balance = 50.0;

        public CheckingAccount(double balance) {
            /* implementation now shown */
        }
        /* FEE of $2 deducted if withdrawal leaves balance less than
         * MIN_BALANCE. Allows for negative balance. */
        public void withdraw(double amount) {
            /* implementation now shown */
        }
    }
}

```

Follow us at WeChat: testdaily, and chase your dream together with more than 100,000 friends.

请关注微信公众号: testdaily，并回复关键字「答案」查看**本手册所有练习答案** ~



1. Of the methods shown, how many different nonconstructor methods can be invoked by a SavingsAccount object?

- A.1
- B.2
- C.3
- D.4
- E.5

2.Which of the following correctly implements the default constructor of the SavingsAccount class?

- I. myInterestRate = 0;
super();
- II. super();
myInterestRate = 0;



III. `super();`

- A. II only
- B. I and II only
- C. II and III only
- D. III only
- E. I, II and III

3. Which is a correct implementation of the constructor with parameters in the SavingsAccount class?

- A. `myBalance = balance;`
`myInterestRate = rate;`
- B. `getBalance() = balance;`
`myInterestRate = rate;`
- C. `super();`
`myInterestRate = rate;`
- D. `super(balance);`
`myInterestRate = rate;`
- E. `super(balance, rate);`

4. Which is a correct implementation of the CheckingAccount constructor?

- I. `super(balance);`
- II `super();`
`deposit(balance);`
- III. `deposit(balance);`
- A. I only
- B. II only
- C. III only
- D. II and III only
- E. I, II, and III

5. Write implementation code for the withdraw method in the CheckingAccount class.

```
super.withdraw(amount);
if(getBalance()<MIN_BALANCE)
    super.withdraw(FEE);
```

6. Redefining the withdraw method in the CheckingAccount class is an example of

- A. method overloading
- B. method overriding
- C. downcasting
- D. dynamic bonding(late bonding)
- E. static bonding(early bonding)

Use the following for Question 7-9

A program to test the BankAccount, SavingsAccount, and CheckingAccount classes has these declaration:

```
BankAccount b = new BankAccount(1400);
BankAccount s = new SavingsAccount(1000, 0.04);
BankAccount c = new CheckingAccount(500);
```

7. Which method call will cause an error?

- A. b.deposit(200);
- B. s.withdraw(500);
- C. c.withdraw(500);
- D. s.deposit(10000);
- E. s.addInterest();

8. In order to test polymorphism, which method must be used in the program?

- A. Either a SavingsAccount constructor or a CheckingAccount constructor
- B. addInterest
- C. deposit
- D. withdraw
- E. getBalance

9. Which of the following will not cause a ClassCastException to be thrown?

- A. ((SavingsAccount) b).addInterest();
- B. ((CheckingAccount) b).withdraw(200);
- C. ((CheckingAccount) c).deposit(800);
- D. ((CheckingAccount) s).withdraw(150);
- E. ((SavingsAccount) c).addInterest();

第七章 标准类(II)

Part I Object

一、Object 介绍

所有类都是 Object 类的子类 ,即所有类都直接或间接地继承 Object。在类层级数之中 , Object 在顶部。这就是说 , Object 是所有类的直接或间接的超类 , 因此 Object 有的函数所有类都有 ,但是注意 ,因为子类可以 override 自己的超类所具有的函数 ,Object 所具有的函数的效果可能和子类不一样。

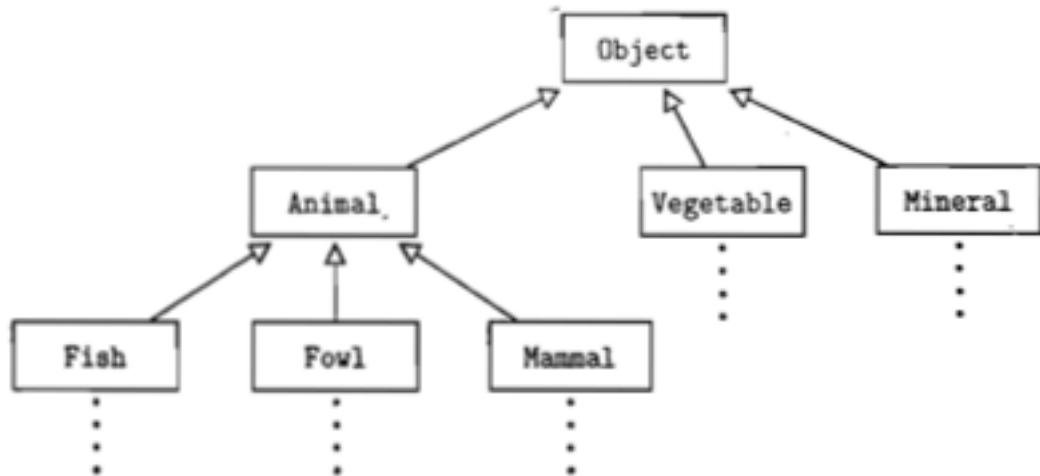


图 1 , 来自巴朗

二、Object 中的方法

1. `toString` 方法

用法 : `xxx.toString();`

意义 : 将一个 object 转换为字符串

e.g.

```

int a = 120;
String b;
b = a.toString();
//b 就会是"120"

```

把数字转换成字符串 , 这很好理解 , 那么其他 Object 呢 ?

我们可以试一试 , 新建一个 Class1.java , 填入如下内容 :

```

public class Class1{

}

public class HelloWorld{
    public static void main(String []args) {
        Class1 class1obj = new Class1();
        System.out.println(class1obj);
    }
}

```

接着我们看看会输出什么，答案是：

```
Class1@6d06d69c
```

为什么呢？原来，在 Object 类没有被 override 的情况下，一个类的 `toString` 生成的字符串就是“这个类的名字@一个十六进制数值”。对于有一些类，它们 `toString` 之后可不是这样的，这是因为它们或者它们的父类 override 了 Object 类的 `toString` 函数。

```
2.equals  
aaa.equals(bbb)
```

如果参数中的 Object (bbb) 和函数所在的 Object (aaa) 是一个 Object 的话，就返回 true。现在我们来理解一下什么叫“一个 Object”。

e.g.

```
Date d1 = new Date("January", 14, 2017);  
Date d2 = d1;  
Date d3 = new Date("January", 14, 2017);
```

运行之后我们可以发现，d1 和 d2 的 `equals` 返回 true，而 d2 和 d3 (或 d1 和 d3) 的 `equals` 却返回 false，这是为什么呢？

我们可以用下面的方法来理解。

在理解之前我们需要知道，与原始类型不同，Object 和之前提到的 String 的储存方式是类似的，Object 是通过内存地址储存在变量里的，因此当我们定义 `d1=new Date(...)` 的时候，计算机是通过 `new` 来创建了一个 Date 对象并且把地址赋值给了 `d1`。理解了这个之后我们就来看看上面返回值的原因。

首先，我们通过 `new` 来让 Java 给我们创建了一个新的 Date 类的对象，`d1`。

之后，我们把 `d1` 的地址给了 `d2`。

再然后，我们又通过 `new` 创建了一个 Date 类的对象，`d3`。

虽然这里 `d1` 和 `d3` 创建时的参数是一样的，`d1` 和 `d3` 的地址是不一样的，换句话说，`d1` 和 `d3` 是两个 Object，对 `d1` 进行操作并不会影响到 `d3`。



大家可能就会明白了，原来“同一个 Object”指的就是地址相同。

默认的 equals 其实和`==`是等价的，`d1.equals(d2)`和 `d1 == d2` 返回值是相同的，从这里我们也可以看出来，`==`对于 object 来说也是比较内存地址，而不是它们的属性。

如果你也写了一个类，并且你想要 equals 不止比较地址，你可以 override equals 函数来达到你所想要的其他效果，比如 String 类的 equals 函数是判断字符串是否相同而不是地址是否相等。

Part II 包装类

一、包装类介绍

所谓包装类，顾名思义，就是把一个东西包装起来的类，那把什么东西包装起来？为什么要包装？这我们要看一下。

a) 包装的对象和原因

包装类是把一个对象或者一个原始类型数值包装起来的对象，为什么没事需要把一个对象或者原始类型数值包装起来呢？其实是因为有以下两种情况：

i. 有的 Java 函数需要 object 作为参数，因此需要把原始类型数值包装成 object 再传入

a) 它可以被用于需要对象项目 Java 容器类中

b) 包装类的作用

包装类能做什么？它能做两件事：

i. 封包

ii. 拆包

所谓的封包，其实就是用一个原始类型数值来创建一个 object，让 object 带有这个

原始类型数值。而拆包，就是从这个 object 里面获得这个在创建时放入的原始类型数值。

Java 其实想得很周到，给每一个原始类型数值都有对应的包装类，而我们需要知道的，就只有 Integer 和 Double 这两个包装类。

二、Integer 类

看了名字我们就知道，Integer 类是个储存整数的类。实际上它是包装 Int 类型的数值的一个类。它有以下几个函数我们需要知道：

a) Integer(int value)

这也是它的构造函数，通过 new Integer(value) 来创建包装了 value 的 Integer 类。这也就是我们所叫的“封包”。

b) int compareTo(Object other)

如果参数也是 Integer 的话，那么就可以进行比较了。注意，如果返回 0 则说明相等，如果小于零那么这个包装类里的数值比 other 小，如果大于零那么这个包装类里的数值比 other 大。

这下大家就能体会到包装类的作用了，这个函数的参数是个 object，如果是原始类型可是没办法传入进行比较的。

大家可能会想问，为什么不把参数变为 Int 来分别比较 Integer 和其他 Object、Integer 和 Int 呢？

答案是，Integer 类实现了 Comparable 接口，对于这个接口参数的类型需要是 Object，修改为 Int 后这样的实现方式是不行的。

大家可能又会问，如果我就不传一个 Integer 类到这个函数里会如何？答案是会报错。对于 other 不是 Integer 类的情况，这个函数会抛出 ClassCastException。

c) int intValue()

返回包里面的数值（返回为 Int 类型），这也就是我们所说的“拆包”。



d) boolean equals(Object obj)

只在这个 Integer 类型和 obj 的数值相等时返回 true。注意到了吗，这个和前面所说 Object 的 equals 函数不同。就算比较的两个 object 不是 “一个 Object” ，它们也可以相等，只需要数值相等就可以了。这是因为 Integer 这个类重写了 Object 类的 equals 函数，使其不止判断地址而是判断了类中所包含的数值，在相等的时候返回 true。

e) String toString()

返回代表这个数值的 String 对象，什么是 “代表” ？就是说，对于 Integer 类型，toString()返回的是数值，而不是 Object 类那样的 “类名@十六进制数字” 。

举个例子：

```
Integer a = new Integer(10); //封包
Integer b = new Integer(10); //封包
System.out.println(a == b);
System.out.println(a.equals(b));
System.out.println(a.intValue());
System.out.println(a.compareTo(b));
```

结果：

```
false
true
10
0
```

为什么是这样呢？我们要先对代码进行分析。

这段代码，首先创建了一个 Integer 对象，并且把地址给了 a，然后又创建了一个 Integer 对象，把地址给了 b。这样我们可以知道，这两个 Integer 类里面的 int 数值是一样的，都是 10，但是这两个 Integer 对象地址不一样，也就是说不是 “一个 Object”，因此我们可以知道 a == b 返回 false。然而前面说过了 a == b 和 a.equals(b) 是等价的，那为什么这里会不同呢？

别忘了，我们在讲 Integer 对象的方法时说过，Integer 把 Object 里面的 equals 方法重写了，也就是只要这两个 Integer 对象所代表的数值相等，那么 equals 就会返回 true，也正是这样。

a.intValue()就是把这个对象拆包，把数值取出来了。而 compareTo 返回 0 代表这两个 Integer 类对象代表的数值相等。

三、Double 类

对于 Double 类我们就不需要讲太多，因为和 Integer 很像，我们就直接讲 Double 类里面的方法。

1. Double(double value)

用 double 数值创建一个 Double 类，也就是封包。

2. double doubleValue()

返回 Double 对象所代表的数值，就是拆包。

3. int compareTo(Object other)

比较两个 double 对象。如果 other 不是 Double，会抛出 ClassCastException。

4. boolean equals(Object obj)

5. String toString()

TIPS: Integer, Double 和 String 都实现了 Comparable 接口，都可以通过 compareTo 方法进行比较。

Chapter7 Practice

Question 1-6 refer to the following Date class declaration:

```
public class Date
{
    private int myDay;
    private int myMonth;
    private int myYear;
    public Date(){
        ...
    }
    public Date(int mo, int day, int yr){}
```



```

...
}

public int day(){
    ...
}

public int month(){
    ...
}

public int year(){
    ...
}

//Returns String representation of Date as "m/d/y", e.g. 4/18/1985
public String toString(){
    ...
}
}

```

1. Which of the following correctly constructs a Date object?

- a) Date d = new (2,13,1947);
- b) Date d = new Date(2,13,1947);
- c) Date d;
- d) new (2,13,1947);
- e) Date d;
- f) Date(2,13,1947);

2. Which of the following will cause an error message?

I Date d1 = new Date(8,2,1947);

Date d2 = d1;

II Date d1 = null;

Date d2 = d1;

III Date d = null;

int x = d.year();

- a) I only
- b) II only
- c) III only
- d) II and III only

3. A client program creates a Date object as follows:

Date d = new Date(1,13,2002);

Which of the following subsequent code segments will cause an error?

- a) String s = d.toString();
- b) int x = d.day();
- c) Date e = d;
- d) Date e = new Date(1,13,2002);

4. Consider a write() method that is added to the class:

```
public void write(){  
/* code */  
}
```

Which of the following could be used as /* code */

I System.out.println(myMonth + "/" + myDay + "/" + myYear);

II System.out.println(month() + "/" + day() + "/" + year());

III System.out.println(this);

- a) I only
- b) II only
- c) III only
- d) I, II, and III

5. What will running this program cause?

```
Date d1 = new Date(month, day, year);
```

```
Date d2 = d1;
```

- a) create an object of Date and pass its address to d1 and d2
- b) create an object of Date and give it to d1, d2
- c) create an object of Date and pass its address to d1 and copy it to d2
- d) create an object of Date and give it to d2 but pass address to d1

Follow us at WeChat: testdaily, and chase your dream together with more than 100,000 friends.

请关注微信公众号: testdaily，并回复关键字「答案」查看**本手册所有练习答案** ~



第八章 递归

Part I 认识递归

一、递归介绍

递归方法是什么？递归方法就是调用自己的方法。e.g. 如下方法就是一个递

归方法：

```
public static void recursion()
```



```
{
    recursion();
}
```

一旦有使用者调用这个方法，这个方法会在运行时调用自己，看看会有什么情况？首先使用者调用了 recursion()，然后在这个方法里 recursion()会被再调用，在调用中再次调用……然后一直继续下去，最后导致一个叫做“栈”的结构被占满，程序就报错了。造成的效果包括电脑卡机、崩溃和一个栈满的 Exception。

照上面这么说，递归有什么用？不就是一个玩弄人的恶作剧吗？其实上面的程序是递归的不正确的使用方法，对于其我们需要避免，而正确地使用递归，我们可以轻松解决很多计算机问题。

要想正确地使用递归，就需要先学会如何分析它。

二、分析递归

1.一个递归小例子

e.g.

```
public static void words() {
    String word = IO.readString(); // 这是一个读取字符串的函数
    if (word.equals(".")) {
        System.out.println(); // 这里没有传入参数，意思就是直接换行
    } else
        words()
    System.out.println(word);
}
```

比如有使用者调用了这个 words 函数，然后输入：

```
hold
my
hand
```

那么就会输出

```
hand  
my  
hold
```

我们看到这个程序读取几列文字，以“.”结尾，然后把这列文字倒过来输出。这要怎么理解呢？

我们要知道，在Java里，每一次调用一个函数都会先暂停当前函数的状态，然后启动一个新的状态来执行这个函数，这两个函数中定义的变量（也称局部变量），是不相同的，也就是说我们在当前words函数里改变的变量和我们再次调用的words函数里的变量是不一样的。这是由于计算机的“栈”的概念，你可以把“栈”想成是一个立在地上的竹竿，我们可以往上面套同样大小的圆环，先套进去的在下，后套进去的在上，只有等上面拿出来了才能拿出下面的。每一次调用函数（，都是往“栈”里面存储了一个新的状态，就像网竹竿里面套了圆环一样。

知道了“栈”是什么，我们接下来用栈这个概念来分析一下这个程序。

首先我们自己在其他地方调用了words函数，此时我们的栈（自己之前用来调用words的那个函数不算）中有一个words函数了，在这个函数中变量word是“hold”，然后它会要求我们输入一个字符串，我们输入“hold”，不是“.”，因此就再调用words函数，同样地，我们创建了一个新的状态，在word里面存储了“my”，接着再次调用，又创建了一个新的状态，在word里面存储了“hand”。最后，我们输入了一个“.”，这样程序发现word里面就是“.”，因此输出换行，然后输出word的内容，也就是“.”，最后离开这个words函数，这个words函数就被从“栈”里面取出了，此时的栈如图1所示，有三个函数，这三个函数准备输出的内容（word变量的内容）在图上有说明。

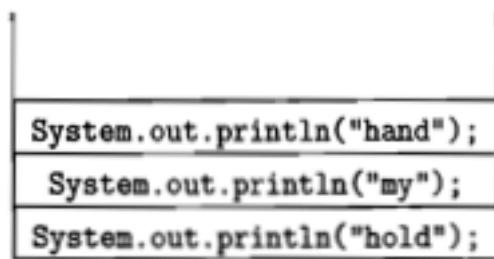


图 1 , 来自巴朗

现在我们已经有一个“.”和一个换行符在屏幕上。接着我们继续执行，取出栈最顶上的那个 words 函数，执行存储时执行的那句代码以下的内容，也就是 System.out.println(word); , word 在这个函数里是 hand , 就输出 hand 加上换行 , 接着道理一样 , 取出 word 函数 , 输出 my , 再取出 word 函数 , 输出 hold. 然后就组成了我们的输出 :

hand
my
hold

TIPS:每一次调用函数时，函数内定义的变量（局部变量）都会被重新创建，两个变量修改、读取都不会干扰到对方。

2. 递归通法

除了上面那个恶作剧递归函数，每个递归函数都有两个部分：

- a) 一个基本的条件来终止递归的继续调用
- b) 非基本条件使递归向基本条件运行

也就是说，递归的最终目的是遇到那个我们需要的“基本条件”，然后就不继续递归了。

e.g.

```

public void draw(int n) {
    if (n == 0)
        System.out.println("That's all.");
    }
}
    
```

```
else{
    for(int i = 1; i <= n;i++)
        System.out.print("*");
    System.out.println();
    draw (n - 1);
}
}
```

draw(5)运行的结果是：

```
*****
****
***
**
*
That's all.
```

在上面的例子中， $n == 0$ 就是我们所需要达到的目的，我们在这个方法的结尾调用了 draw，后面就没有内容了，这样的递归就是结尾递归（tail recursion）。



Part II 深入递归

AP 考试大纲中说了，AP 考生不需要会写递归程序，因此我们只要学会分析递归就可以了。

一、解读递归

下面是一道自编的题，虽然题型和 AP 不同，但是大家就领会一下精神。

Read the following code and calculate the value of fib(5).

```
public static int fib(int n){  
    if(n == 1||n == 2)  
        return 1;  
    else  
        return fib(n - 1) + fib(n - 2);  
}
```

于是我们就要开始展开分析工作了，首先我们可以发现，在 $n == 1$ 或 $n == 2$ 的时候这个方法会返回 1，其他时候会返回 $\text{fib}(n - 1) + \text{fib}(n - 2)$ ，也就可以写成如下函数（ n 是正整数）：

$$\text{Fib}(n) = \begin{cases} 1, & n = 1, 2 \\ \text{Fib}(n - 1) + \text{Fib}(n - 2), & n \geq 3 \end{cases}$$

然后我们就要开始计算函数的返回值了，我们从 $\text{fib}(5)$ 开始，通过对数学函数的分析我们可以画出如下的分析图。

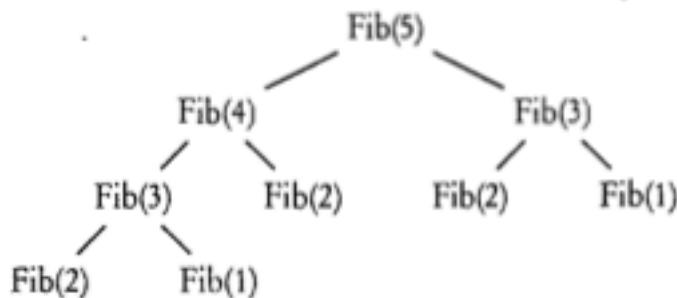


图 2 ,来自巴

朗

因为每一个函数都会调用两个函数，所以树状分析图看起来会比栈图简洁明了，从

下往上，把下面的两个小树枝（子节点）的数值计算出来，相加就得到上面的数值，最后就能计算出 $\text{fib}(5)$ 了，结果是 5。

二、递归的效率

从上面的图 2 可以看到，上面的函数每一次递归都需要调用两个子函数，而这两个子函数又会调用其它函数。刚才我们说过，函数调用的时候需要使用栈来存储当前函数和调用的函数的状态，因此在有大量函数调用的情况下，就要频繁访问栈来存取函数的状态，这样就会消耗掉大量的时间。因此如果有简单的循环能解决这个问题，就尽量不要使用递归来解决。另外递归还有如下需要注意的地方：

1. 在递归中尽量不要使用过多局部变量、局部数组，因为在函数调用时计算机要再次初始化局部变量、局部数组，而这样会消耗大量时间
2. 只在递归可以大量简化代码的时候使用，如果能用循环等简单解决就不要使用递归
3. 递归在以下两个方面很有用：分支树的处理和二进制等分支搜索
4. 可以使用帮助方法（helper method）来调用递归，这样可以使代码更简洁



Chapter8 Practice

1. Which of the following are true? (1)

- I Every recursive algorithm can be written iteratively.
- II Tail recursion is often used in algorithms that repetition can do.
- III In a recursive definition, an object is defined in terms of a simpler case of itself.
- a) I only
- b) III only
- c) I and II only
- d) I and III only

Question 2 and 3 refer to method t:

```
//Precondition: n >= 1
int t(int n){
    if(n == 1 || n == 2)
        return 2 * n;
    else
        return t(n - 1) - t(n - 2);
}
```

2. What will be returned by t(5)?

- a) 5
- b) 3
- c) 0
- d) -4

3. For the method call t(6), how many calls to t will be made, including the original call?

- a) 4
- b) 2
- c) 15
- d) 25

4. Consider method foo:

```
int foo(int x){
    if(x == 1 || x == 3)
        return x;
    else
        return x * foo(x - 1);
}
```

What will be the value z after execution of the following statement?

- ```
int z = foo(foo(3) + foo(4));
```
- a)  $(15!)/(2!)$
  - b)  $3! + 4!$
  - c)  $(7!)!$
  - d) 15

5. Consider the following method:

```
void doSomething(int n){
 if(n > 0){
 doSomething(n - 1);
 System.out.print(n);
 doSomething(n - 1);
 }
}
```

What would be output following the call `doSomething(3)`?

- a) 3211211
- b) 1121213
- c) 1213121
- d) 1211213

Follow us at WeChat: testdaily, and chase your dream together with more than 100,000 friends.

请关注微信公众号: testdaily，并回复关键字「答案」查看**本手册所有练习答案** ~





## 第九章 搜索与排序

### Part I 搜索(searching)

#### 一、顺序搜索(sequential searching)

顺序搜索就是把数组从头到尾搜索一遍，直到找到所要找的元素或者搜索完所有元素为止。

e.g.

```
/*
 * @param elements an array containing the items to be searched
 * @param target the item to be found in element.
 * @return an index of target in elements if found; -1 otherwise;
 */
public static int sequentialSearch(int[] elements, int target) {
 for (int j = 0; j < elements.length; j++) {
 if (elements[j] == target) //String uses equals()
 return j;
 }
 return -1;
}
```

顺序搜索最好的情况是我们只需要进行一次搜索就可以。最坏的情况是我们要搜索的内容在数组最后或者根本不存在，这个时候我们要遍历整个数组。

#### 二、二进制搜索(binary searching)

二进制搜索是一种高效率的检索方法。将搜索范围逐渐减小到只剩下一个。

其思想就是我们查看一个数组中间的内容，如果比我们想要的大，那么我就在左边搜索，如果比我们想要的小，我们就在右边搜索。然后查看判断后数组中间的数，这样依次进行下去。

但是二进制搜索有一个缺点，就是**利用二进制搜索的数组必须是排列好的，否则无法使用二进制搜索。**

```
/*
 * @param elements an array containing the items to be searched.
```

```

Precondition: items in elements are sorted in ascending
order.

@param target. the item to be found in elements.
@return an index of target in elements if target found; -1
otherwise.

*/
public static int binarySearch(int[] elements, int target) {
 int left = 0;
 int right = elements.length - 1;
 while (left <= right) {
 int middle = (left + right) / 2;
 if (target < elements[middle])
 right = middle - 1;
 else if (target > elements[middle])
 left = middle + 1;
 else
 return middle
 }
 return -1;
}

```

这就是一个如何进行二进制搜索的代码。理解起来稍微绕一点脑子，可以对着代码在脑海中执行一遍。

e.g. 假设我们在一个数组中要搜索 5 这个数字

|      |      |      |      |      |      |      |      |      |
|------|------|------|------|------|------|------|------|------|
| a[0] | a[1] | a[2] | a[3] | a[4] | a[5] | a[6] | a[7] | a[8] |
| 1    | 4    | 5    | 7    | 9    | 12   | 15   | 20   | 21   |

First pass: mid = (8+0)/2 = 4. Check a[4].  
Second pass: mid = (0+3)/2 = 1. Check a[1].  
Third pass: mid = (2+3)/2 = 2. Check a[2]. Yes! Key is found.

## Part II 排序(sorting)

### 一、选择排序(selection sort)

选择排序就是首先在数组中找到最小的一个值放在第一位，然后再找次小的放到第二位，以此类推一直到 n-1 次排序后，就能够得到一个递增的数列了。



```

/*
 *param elements an array containing the items to be sorted.

Postcondition: elements contains its original items and items
in elements are sorted in ascending order.

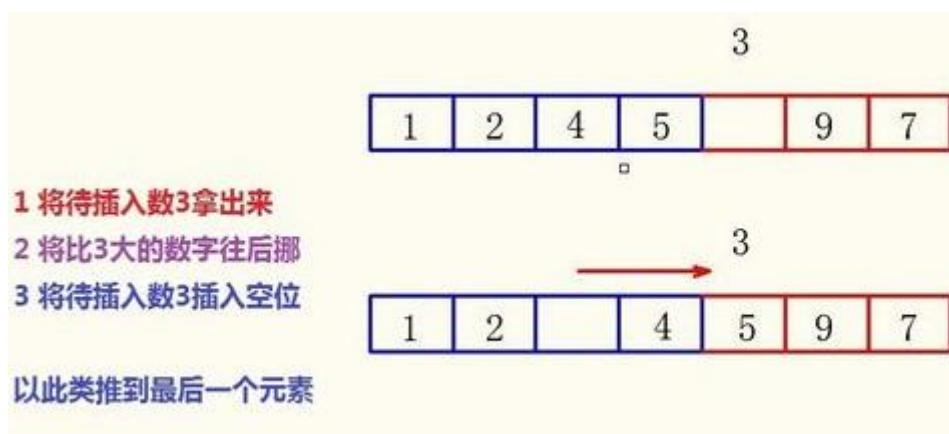
*/
public static void selectionSort(int[] elements) {
 for (int j = 0; j < elements.length) {
 int minIndex = j;
 for (int k = j+1; k < elements.length; k++) {
 if (elements[k] < elements[minIndex])
 minIndex = k;
 }
 int temp = elements[j];
 elements[j] = elements[minIndex];
 elements[minIndex] = temp;
 }
}

```

## 二、插入排序(insertion sort)

插入排序的原本是一个排列好的数组插入一个数组。当我们把第一个数 `elements[0]` 看成一个数组时，我们就能够对整个数组进行插入排列。我们是将要插入的数组的第一个数拿出来，一个一个往前推，看哪个合适插在哪里。之后依次类推。

e.g.蓝色是我们已经排好的数组，红色为未排好的。



/\*

```

@param elements an array containing the items to be sorted.

Postcondition: elements contains its original items and items
in elements are sorted in ascending order.

*/
public static void insertionSort(int[] elements) {
 for(int j = 1; j < elements.length; j++) {
 int temp = elements[j];
 int possibleIndex = j;
 while(possibleIndex > 0 && temp < elements[possibleIndex - 1]) {
 elements[possibleIndex] = elements[possibleIndex - 1];
 possibleIndex--;
 }
 elements[possibleIndex] = temp;
 }
}

```

### 三、归并排序(merge sort)

归并排序是一种递归排序，它将数组拆成几个小的单元，之后对小的单元进行排序。然后再把小的单元组合在一起再进行排序。

它的缺点十分明显：它需要初始化或者占用一个跟原始数组一样长的数据。它还有个显著特点，就是排序时间不受原本数据排序的影响，不论是最好的情况还是最坏的情况。

这种排序方法不要求在 AP 考试中写出，理解其作用原理即可。

```

public static void mergeSort(int[] elements) {
 int n = elements.length;
 int[] temp = new int[n];
 mergeSortHelper(elements, 0, n-1, temp);
}

/*
 * @param elements, an array containing the items to be sorted.
 * @param from, the begining index of the items in elements to be
sorted.
 * @param to, the ending index of the items in elements to be sorted.
 * @param temp, a temporary array to use during the merge process.
 * Precondition:
 * (elements.length == 0 ||
 * 0 <= from <= to <= elements.length &&

```



```

* elements.length == temp.length)
* Postcondition:
* elements contains its original items and the items in
* elements[from]...<= elements[to] are sorted in ascending order
*/
private static void mergeSortHelper(int[] elements, int from, int to,
int[] temp){
 if (from < to){
 int middle = (from + to)/2;
 mergeSortHelper(elements, from , middle, temp);
 mergeSortHelper(elements, middle+1, to, temp);
 merge(elements, from, middle,to,temp);
 }
}

private static void merge(int[] elements, int from, int mid, int to,
int[] temp){
 int i = from;
 int j = mid +1;
 int k = from;

 while(i<=mid && j <=to){
 if(elements[i]<elements[j]){
 temp[k] = elements[i];
 i++;
 }
 else{
 temp[k] = elements[j];
 j++;
 }
 k++;
 }

 while (i <= mid){
 temp[k] = elements[i];
 i++;
 k++;
 }

 while(j<=to){
 temp[k] = elements[j];
 j++;
 k++;
 }
}

```

```
for (k = from; k<=to; k++)
 elements[k] = temp[k];
}
```

## Chapter9 Practice

1. The decision to choose a particular sorting algorithm should be made based on

- I. Run-time efficiency of the sort
- II. Size of the array
- III. Space efficiency of the algorithm
  - A. I only
  - B. II only
  - C. III only
  - D. I and II only
  - E. I, II and III

2. The following code fragment does a sequential search to determine whether a given integer,

value , is stored in an array a[0]...a[n-1].

```
int i = 0;
while(/* boolean expression */) {
 i++;
}
if(i == n)
 return -1 //value not found
else
 return; //value found at location i
```

Which of the following should replace /\* boolean expression \*/ so that the algorithm works as intended?(hint: careful about OutOfBound error)

- A. value != a[i]
- B. i<n && value == a[i]
- C. value !=a[i] && i<n
- D. i <n && value!=a[i]
- E. i<n || value !=a[i]

3. A feature of data that is used for a binary search but not necessarily used for a sequential search

is

- A. length of list.
- B type of data.



- C. order of data.
- D. smallest value in the list.
- E. median value of the data.

4. Assume that  $a[0] \dots a[N-1]$  is an array of  $N$  positive integers and that the following assertion is true:

$a[0] > a[k]$  for all  $k$  such that  $0 < k < N$

Which of the following must be true?

- A. The array is sorted in ascending order.
- B. The array is sorted in descending order.
- C. All values in the array are different.
- D.  $a[0]$  holds the smallest value in the array.
- E.  $a[0]$  holds the largest value in the array.

5. The code shown sorts array  $a[0]..a[a.length-1]$  in descending order.

```
public static void sort(Comparable[] a) {
 for(int i = 0; i < a.length-1; i++)
 for (int j = 0; j <a.length - i - 1; j++)
 if(a[j].compareTo(a[j+1])<0)
 swap(a , j , j + 1); //swap a[j] and [j+1]
}
```

This is an example of

- A. selection sort
- B. insertion sort
- C. mergesort
- D. quicksort.
- E. None of the above

Follow us at WeChat: testdaily, and chase your dream together with more than 100,000 friends.

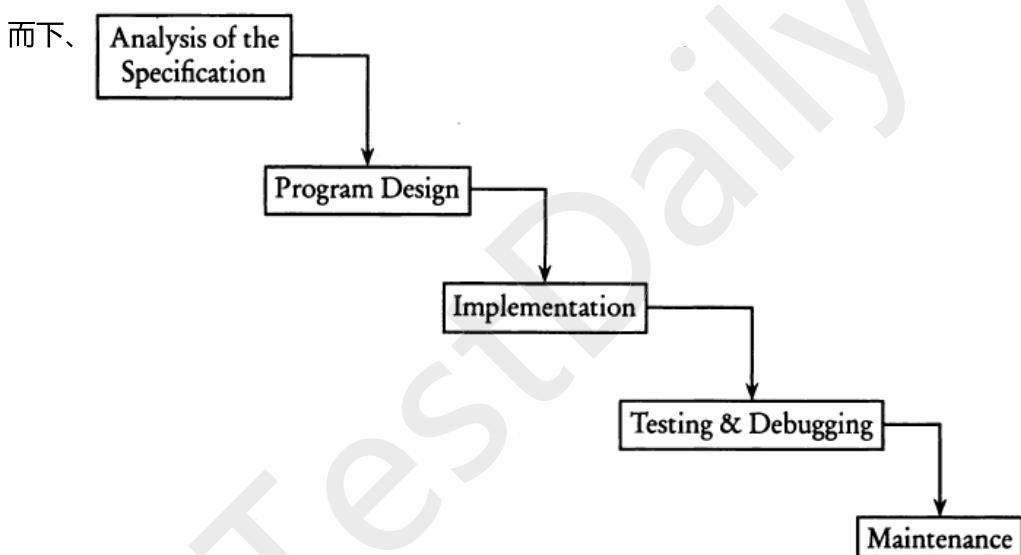
请关注微信公众号: testdaily，并回复关键字「答案」查看**本手册所有练习答案** ~

## 第十章 程序设计与分析

### Part I 瀑布模型(The waterfall Model)

设计一个成功的程序需要大量的步骤，而瀑布模型就是一种很有逻辑性的程序设计周期。

瀑布模型核心思想是按流程将问题化简，将功能的实现与设计分开，便于分工协作，即采用结构化的分析与设计方法将逻辑实现与物理实现分开。将软件生命周期划分为制定计划和需求分析、软件设计、程序编写、软件测试和运行维护等五个基本活动，并且规定了它们自上而下、



#### a) 程序需求分析

程序需求分析是设计一个程序的第一步，在设计一个程序前，我们要知道我们想让程序干什么。一般根据客户的需求进行设计我们的程序，对于程序的需求进行分析，确保顾客想要的能够实现，并且程序员将顾客所没提及到的一些事情进行阐述说明。确保最后做出来的是顾客想要的。



## b) 程序设计

对于小型程序我们可以进行直接编写。但是作为一个优秀的程序员，需要有一个对于自己程序的一个具体计划：这个程序有什么，能干什么，需要几个模块，分为几个部分。诸如此类。

## c ) 程序编写

程序编写要根据语言进行区分，在我们设计程序的时候就应该确定了是面向对象还是面向过程。在 AP 中，我们选择的是 Java 语言。要根据 Java 语言的一些特性编写我们的程序。

## d) 调试和测试

### i. 数据测试

我们在调试数据的时候，我们要把数据类型和可能的值都要考虑进去，以防止出现出乎意料的 bugs。事实上，产品使用者能干出的事情是丧心病狂的，所以我们在推出产品前一定要对数据类型进行全面测试。同时，这也涉及到了程序稳健性(robustness)问题。我们不能够允许程序给出错误答案，不能够允许在数据类型错误的时候程序崩溃，不能允许对于错误类型数据程序照样执行。

e.g. 对于一个排列函数，参数为 int 类型。这个函数是为了将数据插入到现有数组中，并且排列顺序正确。

已知数组：2，5，233。测试值就需要包括：比 2 小的；2 到 5 之间的；5 到 233 之间的；233 以上的；2,5,233 这三个数；负数。

### ii.bug 类型

**编译错误**(compile-time error)是在编译时的错误。比如语言使用错误，调用错误，缺少符号等。

**运行错误(run-time error)**是指编译时未报错，但是运行时出现的错误。比如子类函数在调用父类函数时出现的问题，数据中分母为 0，无限循环.在 Java 中，这些错误都会以 Exception 的方式抛出(throws)。

**逻辑错误(intent or logic error)**是并没有明显提示，程序可以正常运行。但结果并不是我们想要的。

e)程序维护(maintenance)

维护数据，更新方法，增加新特性，等等。

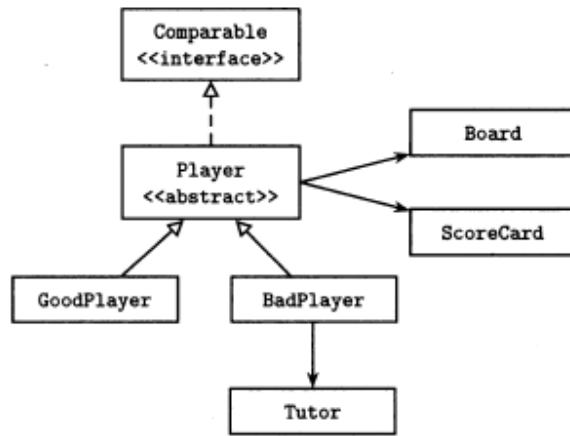
## Part II 面向对象程序设计

### 一、面向对象程序设计步骤

面向对象程序设计需要有几个步骤：

- 确定类
- 确定行为
- 确定类之间的关系
  - 继承关系(is-a 的关系)
  - 组合关系(has-a 的关系，AP 中并未涉及)

### 二、UML 图



绘制 UML 图是一个程序员的好习惯，尽管这并不是 AP Java subset 中的考点。所以这里并不深入的去讲解。有兴趣的同学可以进行展开了解。

### 三、程序开发

- a) 自下而上开发(bottom-up)：写多个零散的代码块，再逐渐组合连接关系。
- b) 自上而下开发(top-down)：写出主体块，再分解为多个零散的代码。

### 四、方法书写

方法书写的具体讲解在前面的章节有提到。对于属性的隐藏(private 来修饰属性)也有提到。在书写方法的时候，我们为了节约 cpu 的占用率以及迅速得出结果我们在书写部分方法时要采用一些**算法(algorithm)**来帮助我们。一些好算法可以**节约 cpu 运算时间以及节约我们的内存**，这些算法就是**高效算法**。但是对于算法的使用实际上还存在一定范围的争辩。采不采用算法完全取决于程序员本身。

### 五、确定类

关于类名的确认一定要简洁直接，但是要避免几种命名：

Program (永远不要用 program 来命名，完全不清楚是用来干什么的)

Teacher (男和女有时候有不同的名字，性别错了有些尴尬)

Data , Record (跟方法名字可能重合，避免混淆)

Class(和 class 关键字重名，不要使用)

## Part III 程序分析

这一部分并不在 AP CSA Course Description 里作为要求，但是仍然鼓励进行了解。我们仅对 AP Java subset 中出现的术语进行讲解。

我们在题目中会遇到 Precondition 和 Postcondition，这两个就是先决条件，和后置条件。先决条件这个条件在程序中一定是真的，后置条件就是运行后的结果。

Javadoc 在题目中也会出现，我们目前要理解到的程度就是 javadoc 就是给后面的属性名字进行解释。在 AP 中常见的@param 以及@return。

e.g. @param element

这个的意思就是我们有一个参数叫做 element。理解到这种程度即可。常见的 javadoc 有：

| 标签               | 说明                                                     | JDK 1.1 doclet | 标准 doclet  | 标签类型              |
|------------------|--------------------------------------------------------|----------------|------------|-------------------|
| @author 作者       | 作者标识                                                   | √              | √          | 包、类、接口            |
| @version 版本号     | 版本号                                                    | √              | √          | 包、类、接口            |
| @param 参数名 描述    | 方法的入参名及描述信息，如入参有特别要求，可在此注释。                            | √              | √          | 构造函数、方法           |
| @return 描述       | 对函数返回值的注释                                              | √              | √          | 方法                |
| @deprecated 过期文本 | 标识随着程序版本的提升，当前 API 已过期，仅为了保证兼容性依然存在，以此告之开发者不应再用这个 API。 | √              | √          | 包、类、接口、值域、构造函数、方法 |
| @throws 异常类名     | 构造函数或方法所会抛出的异常。                                        |                | √          | 构造函数、方法           |
| @exception 异常类名  | 同 @throws。                                             | √              | √          | 构造函数、方法           |
| @see 引用          | 查看相关内容，如类、方法、变量等。                                      | √              | √          | 包、类、接口、值域、构造函数、方法 |
| @since 描述文本      | API 在什么程序的什么版本后开发支持。                                   | √              | √          | 包、类、接口、值域、构造函数、方法 |
| {@link 包#成员 标签}  | 链接到某个特定的成员对应的文档中。                                      |                | √          | 包、类、接口、值域、构造函数、方法 |
| {@value}         | 当对常量进行注释时，如果想将其值包含在文档中，则通过该标签来引用常量的值。                  |                | √(JDK 1.4) | 静态值域              |

## **APPENDIX A**

### **AP Computer Science Java Subset**

The AP Java subset is intended to outline the features of Java that may appear on the AP Computer Science A Exam. The AP Java subset is NOT intended as an overall prescription for computer science courses — the subset itself will need to be supplemented in order to address all topics in a typical introductory curriculum. For example, input and output must be part of a course on computer programming. However, there are many ways to handle input and output in Java. Because of this variation, details of input and output (except for basic text output using `System.out.print` and `System.out.println`) are not tested on the AP Computer Science A Exam.

This appendix describes the Java subset that students will be expected to understand when they take the AP Computer Science A Exam. A number of features are also mentioned that are potentially relevant in an introductory computer science course but are not tested on the exam. Omission of a feature from the AP Java subset does not imply any judgment that the feature is inferior or not worthwhile.

The AP Java subset was selected to

1. enable the test designers to formulate meaningful questions.
2. help students with test preparation.

3. enable instructors to follow a variety of approaches in their courses.

To help students with test preparation, the AP Java subset was intentionally kept small. Language constructs and library features were omitted that did not add significant functionality and that can, for the formulation of exam questions, be expressed by other mechanisms in the subset.

The AP Java subset gives instructors flexibility in how they use Java in their course. For example, some courses teach how to perform input/output using streams or readers/writers, others teach graphical user interface construction, and yet others rely on a tool or library that handles input/output. For the purpose of the AP Computer Science A Exam, these choices are incidental and are not central for the problem solving process or for the mastery of computer science concepts.

The AP Java subset does not address handling of user input at all. That means that the subset is not complete. To create actual programs, instructors need to present additional mechanisms in their courses.

The following section contains the language features that may be tested on the AP Computer Science A Exam. The Java Quick Reference contains a list specifying which Standard Java classes, interfaces, constants, and methods may be used on the exam. This document is available to students when they take the exam, is available at AP Central, and is included in Appendix B.

| Tested in the AP CS A Exam                                                                                                                                                                                                                     | Notes            | Not tested in the AP CS A Exam, but potentially relevant/useful                                                                                                                                  |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Comments</b><br>/* */, //, and /** */<br>Javadoc @param and @return comment tags                                                                                                                                                            |                  | Javadoc tool                                                                                                                                                                                     |
| <b>Primitive Types</b><br>int,<br>double,<br>boolean                                                                                                                                                                                           |                  | char, byte, short, long,<br>float                                                                                                                                                                |
| <b>Operators</b><br>Arithmetic: +, -, *, /, %<br>Increment/Decrement: ++, --<br>Assignment: =, +=, -=, *=,<br>/=, %=<br>Relational: ==, !=, <, <=,<br>>, >=<br>Logical: !, &&,   <br>Numeric casts: (int), (double)<br>String concatenation: + | 1, 2, 3,<br>4, 5 | &,  , ^<br>(char), (float)<br>StringBuilder<br>Shift: <<, >>, >>><br>Bitwise: ~, &,  , ^<br>Conditional: ?:                                                                                      |
| <b>Object Comparison</b><br>object identity (==, !=) vs.<br>object equality (equals),<br>String compareTo                                                                                                                                      |                  | implementation of equals<br>Comparable                                                                                                                                                           |
| <b>Escape Sequences</b><br>\", \\, \n inside strings                                                                                                                                                                                           |                  | \', \t, \unnnn                                                                                                                                                                                   |
| <b>Input / Output</b><br>System.out.print,<br>System.out.println                                                                                                                                                                               | 6                | Scanner, System.in,<br>System.out, System.err,<br>Stream input/output,<br>GUI input/output,<br>parsing input: Integer.parseInt,<br>Double.parseDouble<br>formatting output:<br>System.out.printf |

| Tested in the AP CS A Exam                                                                                                                                                                 | Notes  | Not tested in the AP CS A Exam, but potentially relevant/useful                                                                             |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Exceptions</b><br>ArithmaticException,<br>NullPointerException,<br>IndexOutOfBoundsException,<br>ArrayIndexOutOfBoundsException,<br>IllegalArgumentException                            |        | try/catch/finally<br>throw, throws<br>assert                                                                                                |
| <b>Arrays</b><br>1-dimensional arrays,<br>2-dimensional rectangular arrays,<br>initializer list: { ... },<br>row-major order of<br>2-dimensional array elements                            | 7, 8   | new <i>type</i> [] { ... } ,<br>ragged arrays (non-rectangular),<br>arrays with 3 or more dimensions                                        |
| <b>Control Statements</b><br>if, if/else,<br>while, for,<br>enhanced for (for-each),<br>return                                                                                             |        | switch,<br>break, continue,<br>do-while                                                                                                     |
| <b>Variables</b><br>parameter variables,<br>local variables,<br>private instance variables:<br>visibility (private)<br>static (class) variables:<br>visibility (public, private),<br>final |        | final parameter variables,<br>final local variables,<br>final instance variables                                                            |
| <b>Methods</b><br>visibility (public, private),<br>static, non-static,<br>method signatures,<br>overloading, overriding,<br>parameter passing                                              | 9, 10  | visibility (protected),<br>public static void<br>main(String[] args),<br>command line arguments,<br>variable number of parameters,<br>final |
| <b>Constructors</b><br>super(), super(args)                                                                                                                                                | 11, 12 | default initialization of instance<br>variables, initialization blocks,<br>this(args)                                                       |



| Tested in the AP CS A Exam                                                                                                                                                                                                                                                                            | Notes  | Not tested in the AP CS A Exam, but potentially relevant/useful                                                           |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|---------------------------------------------------------------------------------------------------------------------------|
| <b>Classes</b><br><code>new,</code><br><code>visibility (public),</code><br>accessor methods,<br>modifier (mutator) methods<br>Design/create/modify class.<br>Create subclass of a superclass<br>( <code>abstract</code> , <code>non-abstract</code> ).<br>Create class that implements an interface. | 13, 14 | <code>final,</code><br><code>visibility (private, protected),</code><br>nested classes,<br>inner classes,<br>enumerations |
| <b>Interfaces</b><br>Design/create/modify an interface.                                                                                                                                                                                                                                               | 13, 14 |                                                                                                                           |
| <b>Inheritance</b><br>Understand inheritance hierarchies.<br>Design/create/modify subclasses.<br>Design/create/modify classes that implement interfaces.                                                                                                                                              |        |                                                                                                                           |
| <b>Packages</b><br><code>import packageName.className</code>                                                                                                                                                                                                                                          |        | <code>import packageName.* ,</code><br>static import,<br><code>package packageName ,</code><br>class path                 |
| <b>Miscellaneous OOP</b><br>“is-a” and “has-a” relationships,<br><code>null,</code><br><code>this,</code><br><code>super.method(args)</code>                                                                                                                                                          | 15, 16 | <code>instanceof</code><br>( <i>class</i> ) cast<br><code>this.var, this.method(args),</code>                             |
| <b>Standard Java Library</b><br><code>Object,</code><br><code>Integer, Double,</code><br><code>String,</code><br><code>Math,</code><br><code>List&lt;E&gt;, ArrayList&lt;E&gt;</code>                                                                                                                 | 17, 18 | <code>clone,</code><br>autoboxing,<br><br><code>Collection&lt;E&gt;,</code><br><code>Arrays, Collections</code>           |

## Notes

1. Students are expected to understand the operator precedence rules of the listed operators.
2. The increment/decrement operators `++` and `--` are part of the AP Java subset. These operators are used only for their side effect, not for their value. That is, the postfix form (for example, `x++`) is always used, and the operators are not used inside other expressions. For example, `arr[x++]` is not used.
3. Students need to understand the “short circuit” evaluation of the `&&` and `||` operators.
4. Students are expected to understand “truncation towards 0” behavior as well as the fact that positive floating-point numbers can be rounded to the nearest integer as `(int)(x + 0.5)`, negative numbers as `(int)(x - 0.5)`.
5. String concatenation `+` is part of the AP Java subset. Students are expected to know that concatenation converts numbers to strings and invokes `toString` on objects.
6. User input is not included in the AP Java subset. There are many possible ways for supplying user input: e.g., by reading from a Scanner, reading from a stream (such as a file or a URL), or from a dialog box. There are advantages and disadvantages to the various approaches. The exam does not prescribe any one approach. Instead, if reading input is necessary, it will be indicated in a way similar to the following:

```
double x = /* call to a method that reads a floating-point number*/
or
```

```
double x = ...; // read user input
```

7. Both arrays of primitive types (e.g., int[], int[][])) and arrays of objects (e.g., Student[], Student[][]) are in the subset.
8. Students need to understand that 2-dimensional arrays are stored as arrays of arrays. For the purposes of the AP CS A Exam, students should assume that 2-dimensional arrays are rectangular (not ragged) and the elements are indexed in row-major order. For example, given the declaration

```
int[][] m = {{1, 2, 3}, {4, 5, 6}};
```

m.length is 2 (the number of rows), m[0].length is 3 (the number of columns), m[r][c] represents the element at row r and column c, and m[r] represents row r (e.g., m[0] is of type int[] and references the array {1, 2, 3}).

Students are expected to be able to access a row of a 2-dimensional array, assign it to a 1-dimensional array reference, pass it as a parameter, and use loops (including for-each) to traverse the rows. However, students are not expected to analyze or implement code that replaces an entire row in a 2-dimensional array, such as

```
int[][] m = {{1, 2, 3}, {4, 5, 6}};
int[] a = {7, 8, 9};
m[0] = a; // Outside the Subset
```

9. The main method and command-line arguments are not included in the subset. In free-response questions, students are not expected to invoke programs. In the *AP Computer Science Labs*, program invocation with main may occur, but the main method will be kept very simple.
10. Students are required to understand when the use of static methods is appropriate. In the exam, static methods are always invoked through a class (explicitly or implicitly), never an object (i.e., *ClassName.staticMethod()* or *staticMethod()*, not *obj.staticMethod()*).
11. If a subclass constructor does not explicitly invoke a superclass constructor, the Java compiler automatically inserts a call to the no-argument constructor of the superclass.
12. Students are expected to implement constructors that initialize all instance variables. Class constants are initialized with an initializer:

```
public static final int MAX_SCORE = 5;
```

The rules for default initialization (with 0, false or null) are not included in the subset. Initializing instance variables with an initializer is not included in the subset. Initialization blocks are not included in the subset.

13. Students are expected to write interfaces or class declarations when given a general description of the interface or class.
14. Students are expected to extend classes and implement interfaces. Students are also expected to have knowledge of inheritance that includes understanding the concepts of method overriding and polymorphism. Students are expected to implement their own subclasses.

Students are expected to read the definition of an abstract class and understand that the abstract methods need to be implemented in a subclass. Students are similarly expected to read the definition of an interface and understand that the abstract methods need to be implemented in an implementing class.
15. Students are expected to understand that conversion from a subclass reference to a superclass reference is legal and does not require a cast. Class casts (generally from Object to another class) are not included in the AP Java subset. Array type compatibility and casts between array types are not included in the subset.
16. The use of this is restricted to passing the implicit parameter in its entirety to another method (e.g., *obj.method(this)*) and to descriptions such as "the implicit parameter this". Students are not required to know the idiom "*this.var = var*", where *var* is both the name of an instance variable and a parameter variable.
17. The use of generic collection classes and interfaces is in the AP Java subset, but students need not implement generic classes or methods.
18. Students are expected to know a subset of the constants and methods of the listed Standard Java Library classes and interfaces. Those constants and methods are enumerated in the Java Quick Reference (Appendix B).

## A P P E N D I X B

### Exam Appendix – Java Quick Reference

Accessible methods from the Java library that may be included on the exam

```
class java.lang.Object
• boolean equals(Object other)
• String toString()

class java.lang.Integer
• Integer(int value)
• int intValue()
• Integer.MIN_VALUE // minimum value represented by an int or Integer
• Integer.MAX_VALUE // maximum value represented by an int or Integer

class java.lang.Double
• Double(double value)
• double doubleValue()

class java.lang.String
• int length()
• String substring(int from, int to) // returns the substring beginning at from
 // and ending at to-1
• String substring(int from) // returns substring(from, length())
• int indexOf(String str) // returns the index of the first occurrence of str;
 // returns -1 if not found
• int compareTo(String other) // returns a value < 0 if this is less than other
 // returns a value = 0 if this is equal to other
 // returns a value > 0 if this is greater than other

class java.lang.Math
• static int abs(int x)
• static double abs(double x)
• static double pow(double base, double exponent)
• static double sqrt(double x)
• static double random() // returns a double in the range [0.0, 1.0)

interface java.util.List<E>
• int size()
• boolean add(E obj) // appends obj to end of list; returns true
• void add(int index, E obj) // inserts obj at position index (0 ≤ index ≤ size),
 // moving elements at position index and higher
 // to the right (adds 1 to their indices) and adjusts size

• E get(int index)
• E set(int index, E obj) // replaces the element at position index with obj
 // returns the element formerly at the specified position
• E remove(int index) // removes element from position index, moving elements
 // at position index + 1 and higher to the left
 // (subtracts 1 from their indices) and adjusts size
 // returns the element formerly at the specified position

class java.util.ArrayList<E> implements java.util.List<E>
```

ADVANCED  
PLACEMENT

---

# Computer Science A

---



主 编 | Jian Yu

主 催 | 刘 淼

感 谢 | Diamo

和 一 直 支 持 T e s t D a i l y 的 你