对Java编程思想的忠告Java认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/607/2021_2022__E5_AF_B9Java_E7_BC_96_c104_607496.htm 编写Java程序的注意事项，对Java编程思想的忠告。 (1) 类名首字母应该大写。字段、方法以及对象（句柄）的首字母应小写。对于所有标识符，其中包含的所有单词都应紧靠在一起，而且大写中间单词的首字母。例如： 1. ThisIsAClassName 2. thisIsMethodOrFieldName 若在定义中出现了常数初始化字符，则大写static final基本类型标识符中的所有字母。这样便可标志出它们属于编译期的常数。 Java包（Package）属于一种特殊情况：它们全都是小写字母，即便中间的单词亦是如此。对于域名扩展名称，如com，org，net或者edu等，全部都应小写（这也是Java 1.1和Java 1.2的区别之一）。 (2) 为了常规用途而创建一个类时，请采取"经典形式"，并包含对下述元素的定义： equals() hashCode() toString() clone()（implement Cloneable） implement Serializable (3) 对于自己创建的每一个类，都考虑置入一个main()，其中包含了用于测试那个类的代码。为使用一个项目中的类，我们没必要删除测试代码。若进行了任何形式的改动，可方便地返回测试。这些代码也可作为如何使用类的一个示例使用。 (4) 应将方法设计成简要的、功能性单元，用它描述和实现一个不连续的类接口部分。理想情况下，方法应简明扼要。若长度很大，可考虑通过某种方式将其分割成较短的几个方法。这样做也便于类内代码的重复使用（有些时候，方法必须非常大，但它们仍应只做同样的一件事情）。 (5) 设计一个类时，请设身处地为客户程序员考虑一下（

类的使用方法应该是非常明确的）。然后，再设身处地为管理代码的人考虑一下（预计有可能进行哪些形式的修改，想想用什么方法可把它们变得更简单）。(6) 使类尽可能短小精悍，而且只解决一个特定的问题。下面是对类设计的一些建议： 1.一个复杂的开关语句：考虑采用"多形"机制 2.数量众多的方法涉及到类型差别极大的操作：考虑用几个类来分别实现 3.许多成员变量在特征上有很大的差别：考虑使用几个类 （7）让一切东西都尽可能地"私有"private.可使库的某一部分"公共化"（一个方法、类或者一个字段等等），就永远不能把它拿出。若强行拿出，就可能破坏其他人现有的代码，使他们不得不重新编写和设计。若只公布自己必须公布的，就可放心大胆地改变其他任何东西。在多线程环境中，隐私是特别重要的一个因素只有private字段才能在非同步使用的情况下受到保护。 （8）谨惕"巨大对象综合症".对一些习惯于顺序编程思维、且初涉OOP领域的新手，往往喜欢先写一个顺序执行的程序，再把它嵌入一个或两个巨大的对象里。根据编程原理，对象表达的应该是应用程序的概念，而非应用程序本身。 （9） 若不得已进行一些不太雅观的编程，至少应该把那些代码置于一个类的内部。 （10） 任何时候只要发现类与类之间结合得非常紧密，就需要考虑是否采用内部类，从而改善编码及维护工作（参见第14章14.1.2小节的"用内部类改进代码"）。 （11） 尽可能细致地加上注释，并用javadoc注释文档语法生成自己的程序文档。 （12）避免使用"魔术数字"，这些数字很难与代码很好地配合。如以后需要修改它，无疑会成为一场噩梦，因为根本不知道"100"到底是指"数组大小"还是"其他全然不同的东西".所以，我们应创建一个常数，

并为其使用具有说服力的描述性名称，并在整个程序中都采用常数标识符。这样可使程序更易理解以及更易维护。（13）涉及构建器和异常的时候，通常希望重新丢弃在构建器中捕获的任何异常如果它造成了那个对象的创建失败。这样一来，调用者就不会以为那个对象已正确地创建，从而盲目地继续。（14）当客户程序员用完对象以后，若你的类要求进行任何清除工作，可考虑将清除代码置于一个良好定义的方法里，采用类似于cleanup（）这样的名字，明确表明自己的用途。除此以外，可在类内放置一个boolean（布尔）标记，指出对象是否已被清除。在类的finalize（）方法里，请确定对象已被清除，并已丢弃了从RuntimeException继承的一个类（如果还没有的话），从而指出一个编程错误。在采取象这样的方案之前，请确定 finalize（）能够在自己的系统中工作（可能需要调用System.runFinalizersOnExit（true），从而确保这一行为）。（15）在一个特定的作用域内，若一个对象必须清除（非由垃圾收集机制处理），请采用下述方法：初始化对象；若成功，则立即进入一个含有finally从句的try块，开始清除工作。（16）若在初始化过程中需要覆盖（取消）finalize（），请记住调用super.finalize（）（若Object属于我们的直接超类，则无此必要）。在对finalize（）进行覆盖的过程中，对super.finalize（）的调用应属于最后一个行动，而不应是第一个行动，这样可确保在需要基础类组件的时候它们依然有效。（17）创建大小固定的对象集合时，请将它们传输至一个数组（若准备从一个方法里返回这个集合，更应如此操作）。这样一来，我们就可享受到数组在编译期进行类型检查的好处。此外，为使用它们，数组的接收者也许并

不需要将对象"造型"到数组里。（18）尽量使用interfaces，不要使用abstract类。若已知某样东西准备成为一个基础类，那么第一个选择应是将其变成一个interface（接口）。只有在不得不使用方法定义或者成员变量的时候，才需要将其变成一个abstract（抽象）类。接口主要描述了客户希望做什么事情，而一个类则致力于（或允许）具体的实施细节。（19）在构建器内部，只进行那些将对象设为正确状态所需的工作。尽可能地避免调用其他方法，因为那些方法可能被其他人覆盖或取消，从而在构建过程中产生不可预知的结果（参见第7章的详细说明）。（20）对象不应只是简单地容纳一些数据；它们的行为也应得到良好的定义。（21）在现成类的基础上创建新类时，请首先选择"新建"或"创作".只有自己的设计要求必须继承时，才应考虑这方面的问题。若在本来允许新建的场合使用了继承，则整个设计会变得没有必要地复杂。（22）用继承及方法覆盖来表示行为间的差异，而用字段表示状态间的区别。一个非常极端的例子是通过对不同类的继承来表示颜色，这是绝对应该避免的：应直接使用一个"颜色"字段。（23）为避免编程时遇到麻烦，请保证在自己类路径指到的任何地方，每个名字都仅对应一个类。否则，编译器可能先找到同名的另一个类，并报告出错消息。若怀疑自己碰到了类路径问题，请试试在类路径的每一个起点，搜索一下同名的。class文件。（24）在Java 1.1 AWT中使用事件"适配器"时，特别容易碰到一个陷阱。若覆盖了某个适配器方法，同时拼写方法没有特别讲究，最后的结果就是新添加一个方法，而不是覆盖现成方法。然而，由于这样做是完全合法的，所以不会从编译器或运行期系统获得任何出错提

示只不过代码的工作就变得不正常了。（25）用合理的设计方案消除"伪功能".也就是说，假若只需要创建类的一个对象，就不要提前限制自己使用应用程序，并加上一条"只生成其中一个"注释。请考虑将其封装成一个"独生子"的形式。若在主程序里有大量散乱的代码，用于创建自己的对象，请考虑采纳一种创造性的方案，将些代码封装起来。（26）警惕"分析瘫痪".请记住，无论如何都要提前了解整个项目的状况，再去考察其中的细节。由于把握了全局，可快速认识自己未知的一些因素，防止在考察细节的时候陷入"死逻辑"中。（27）警惕"过早优化".首先让它运行起来，再考虑变得更快但只有在自己必须这样做、而且经证实在某部分代码中的确存在一个性能瓶颈的时候，才应进行优化。除非用专门的工具分析瓶颈，否则很有可能是在浪费自己的时间。性能提升的隐含代价是自己的代码变得难于理解，而且难于维护。（28）请记住，阅读代码的时间比写代码的时间多得多。思路清晰的设计可获得易于理解的程序，但注释、细致的解释以及一些示例往往具有不可估量的价值。无论对你自己，还是对后来的人，它们都是相当重要的。如对此仍有怀疑，那么请试想自己试图从联机Java文档里找出有用信息时碰到的挫折，这样或许能将你说服。（29）如认为自己已进行了良好的分析、设计或者实施，那么请稍微更换一下思维角度。试试邀请一些外来人士并不一定是专家，但可以是来自本公司其他部门的人。请他们用完全新鲜的眼光考察你的工作，看看是否能找出你一度熟视无睹的问题。采取这种方式，往往能在最适合修改的阶段找出一些关键性的问题，避免产品发行后再解决问题而造成的金钱及精力方面的损失。（30）良

好的设计能带来最大的回报。简言之，对于一个特定的问题，通常会花较长的时间才能找到一种最恰当的解决方案。但一旦找到了正确的方法，以后的工作就轻松多了，再也不用经历数小时、数天或者数月的痛苦挣扎。我们的努力工作会带来最大的回报（甚至无可估量）。而且由于自己倾注了大量心血，最终获得一个出色的设计方案，成功的快感也是令人心动的。坚持抵制草草完工的诱惑那样做往往得不偿失。

（31） 可在Web上找到大量的编程参考资源，甚至包括大量新闻组、讨论组、邮寄列表等。 以上就是作者通过亲身体会，提出的对Java编程思想的忠告。 