JAVA更多的类谜题75：头还是尾 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/138/2021_2022_JAVA_E6_9B_B4_E5_A4_9A_c97_138566.htm 这个程序的行为在1.4版和5.0版的Java平台上会有些变化。这个程序在这些版本上会分别做些什么呢？（如果你只能访问5.0版本的平台，那么你可以在编译的时候使用-source 1.4标记，以此来模拟1.4版的行为。） import java.util.Random. public class CoinSide { private static Random rnd = new Random(). public static CoinSide flip() { return rnd.nextBoolean() ? Heads.INSTANCE : Tails.INSTANCE. } public static void main(String[ ] args) { System.out.println(flip()). } } class Heads extends CoinSide { private Heads() { } public static final Heads INSTANCE = new Heads(). public String toString() { return "heads". } } class Tails extends CoinSide { private Tails() { } public static final Tails INSTANCE = new Tails(). public String toString() { return "tails". } } 该程序看起来根本没有使用5.0版的任何新特性，因此很难看出来为什么它们在行为上应该有差异。事实上，该程序在1.4或更早版本的平台上是不能编译的： CoinSide.java:7: incompatible types for ?: neither is a subtype of the other second operand: Heads third operand : Tails return rnd.nextBoolean() ? ^ 条件操作符（？:）的行为在5.0版本之前是非常受限的[JLS2 15.25]。当第二个和第三个操作数是引用类型时，条件操作符要求它们其中的一个必须是另一个的子类型。Heads和Tails彼此都不是对方的子类型，所以这里就产生了一个错误。为了让这段代码能够编译，你可以将其中一个操作数转型为二者的公共超类： return rnd.nextBooleam() ?

(CoinSide)Heads.INSTANCE : Tails.INSTANCE. 在5.0或更新的版本中，Java语言显得更加宽大了，条件操作符在第二个和第三个操作数是引用类型时总是合法的。其结果类型是这两种类型的最小公共超类。公共超类总是存在的，因为Object是每一个对象类型的超类型。在实际使用中，这种变化的主要结果就是条件操作符做正确的事情的情况更多了，而给出编译期错误的情况更少了。对于我们当中的语言菜鸟来说，作用于引用类型的条件操作符的结果所具备的编译期类型与在第二个和第三个操作数上调用下面的方法的结果相同： T choose(T a,T b) { } 本谜题所展示的问题在1.4和更早的版本中发生得相当频繁，迫使你必须插入只是为了遮掩你的代码的真实目的而进行的转型。这就是说，该谜题本身是人为制造的。在5.0版本之前，使用类型安全的枚举模式来编写CoinSide对程序员来说会显得更自然一些[EJ Item 21]： 下面的程序全部是由同步化（synchronized）的静态方法组成的。那么它会打印出什么呢？在你每次运行这段程序的时候，它都能保证会打印出相同的内容吗？ public class PingPong{ public static synchronized void main(String[] a){ Thread t = new Thread(){ public void run(){ pong(). } }. t.run(). System.out.print("Ping"). } static synchronized void pong(){ System.out.print("Pong"). } } 在多线程程序中，通常正确的观点是程序每次运行的结果都有可能发生变化，但是上面这段程序总是打印出相同的内容。在一个同步化的静态方法执行之前，它会获取与它的Class对象相关联的一个管程（monitor）锁[JLS 8.4.3.6]。所以在上面的程序中，主线程会在创建第二个线程之前获得与PingPong.class相关联的那个锁。只要主线程占有着这个锁

，第二个线程就不可能执行同步化的静态方法。具体地讲，在main方法打印了Ping并且执行结束之后，第二个线程才能执行pong方法。只有当主线程放弃那个锁的时候，第二个线程才被允许获得这个锁并且打印Pong 。根据以上的分析，我们似乎可以确信这个程序应该总是打印PingPong。但是这里有一个小问题：当你尝试着运行这个程序的时候，你会发现它总是会打印PongPing。到底发生了什么呢？