1. a【强制】代码中的命名均不能以下划线或美元符号开始，也不能以下划线或美元符号结束。  
   反例：\_name / \_\_name / O b j e c t / n a m e / n a m e Object / name\_ / name Object/name/​name / Object$
2. a【强制】代码中的命名严禁使用拼音与英文混合的方式，更不允许直接使用中文的方式。  
   说明：正确的英文拼写和语法可以让阅读者易于理解，避免歧义。注意，即使纯拼音命名方式  
   也要避免采用。  
   正例：alibaba / taobao / youku / hangzhou 等国际通用的名称，可视同英文。  
   反例：DaZhePromotion [打折] / getPingfenByName() [评分] / int 某变量 = 3
3. a【强制】类名使用 UpperCamelCase 风格，必须遵从驼峰形式，但以下情形例外：DO / BO /  
   DTO / VO / AO  
   正例：MarcoPolo / UserDO / XmlService / TcpUdpDeal / TaPromotion  
   反例：macroPolo / UserDo / XMLService / TCPUDPDeal / TAPromotion
4. a【强制】方法名、参数名、成员变量、局部变量都统一使用 lowerCamelCase 风格，必须遵从  
   驼峰形式。  
   正例： localValue / getHttpMessage() / inputUserId
5. a【强制】常量命名全部大写，单词间用下划线隔开，力求语义表达完整清楚，不要嫌名字长。  
   正例：MAX\_STOCK\_COUNT  
   反例：MAX\_COUNT
6. a【强制】抽象类命名使用 Abstract 或 Base 开头；异常类命名使用 Exception 结尾；测试类  
   命名以它要测试的类的名称开始，以 Test 结尾。
7. a【强制】中括号是数组类型的一部分，数组定义如下：String[] args;  
   反例：使用 String args[]的方式来定义。
8. a【强制】POJO 类中布尔类型的变量，都不要加 is，否则部分框架解析会引起序列化错误。  
   反例：定义为基本数据类型 Boolean isDeleted；的属性，它的方法也是 isDeleted()，RPC  
   框架在反向解析的时候，“以为”对应的属性名称是 deleted，导致属性获取不到，进而抛出异  
   常。
9. a【强制】包名统一使用小写，点分隔符之间有且仅有一个自然语义的英语单词。包名统一使用  
   单数形式，但是类名如果有复数含义，类名可以使用复数形式。  
   正例： 应用工具类包名为 com.alibaba.open.util、类名为 MessageUtils（此规则参考  
   spring 的框架结构）
10. a【强制】杜绝完全不规范的缩写，避免望文不知义。  
    反例：AbstractClass“缩写”命名成 AbsClass；condition“缩写”命名成 condi，此类随  
    意缩写严重降低了代码的可阅读性。
11. b【推荐】为了达到代码自解释的目标，任何自定义编程元素在命名时，使用尽量完整的单词  
    组合来表达其意。  
    正例：从远程仓库拉取代码的类命名为 PullCodeFromRemoteRepository。  
    反例：变量 int a; 的随意命名方式。
12. b【推荐】如果模块、接口、类、方法使用了设计模式，在命名时体现出具体模式。  
    说明：将设计模式体现在名字中，有利于阅读者快速理解架构设计理念。  
    正例：public class OrderFactory;  
    public class LoginProxy;  
    public class ResourceObserver;
13. b【推荐】接口类中的方法和属性不要加任何修饰符号（public 也不要加），保持代码的简洁  
    性，并加上有效的 Javadoc 注释。尽量不要在接口里定义变量，如果一定要定义变量，肯定是  
    与接口方法相关，并且是整个应用的基础常量。  
    正例：接口方法签名：void f();  
    接口基础常量表示：String COMPANY = “alibaba”;  
    反例：接口方法定义：public abstract void f();  
    说明：JDK8 中接口允许有默认实现，那么这个 default 方法，是对所有实现类都有价值的默  
    认实现。
14. b【推荐】使用 CountDownLatch 进行异步转同步操作，每个线程退出前必须调用 countDown  
    方法，线程执行代码注意 catch 异常，确保 countDown 方法被执行到，避免主线程无法执行  
    至 await 方法，直到超时才返回结果。  
    说明：注意，子线程抛出异常堆栈，不能在主线程 try-catch 到。
15. b【推荐】避免 Random 实例被多线程使用，虽然共享该实例是线程安全的，但会因竞争同一  
    seed 导致的性能下降。  
    说明：Random 实例包括 java.util.Random 的实例或者 Math.random()的方式。  
    正例：在 JDK7 之后，可以直接使用 API ThreadLocalRandom，而在 JDK7 之前，需要编码保  
    证每个线程持有一个实例。
16. b【推荐】在并发场景下，通过双重检查锁（double-checked locking）实现延迟初始化的优  
    化问题隐患(可参考 The “Double-Checked Locking is Broken” Declaration)，推荐解  
    决方案中较为简单一种（适用于 JDK5 及以上版本），将目标属性声明为 volatile 型。
17. b【推荐】循环体中的语句要考量性能，以下操作尽量移至循环体外处理，如定义对象、变量、  
    获取数据库连接，进行不必要的 try-catch 操作（这个 try-catch 是否可以移至循环体外）。
18. b【推荐】接口入参保护，这种场景常见的是用于做批量操作的接口。
19. b【推荐】与其“半吊子”英文来注释，不如用中文注释把问题说清楚。专有名词与关键字保持  
    英文原文即可。  
    反例：“TCP 连接超时”解释成“传输控制协议连接超时”，理解反而费脑筋。
20. b【推荐】代码修改的同时，注释也要进行相应的修改，尤其是参数、返回值、异常、核心逻辑  
    等的修改。  
    说明：代码与注释更新不同步，就像路网与导航软件更新不同步一样，如果导航软件严重滞后，  
    就失去了导航的意义。
21. c【参考】谨慎注释掉代码。在上方详细说明，而不是简单地注释掉。如果无用，则删除。  
    说明：代码被注释掉有两种可能性：1）后续会恢复此段代码逻辑。2）永久不用。前者如果没  
    有备注信息，难以知晓注释动机。后者建议直接删掉（代码仓库保存了历史代码）。
22. c【参考】对于注释的要求：第一、能够准确反应设计思想和代码逻辑；第二、能够描述业务含  
    义，使别的程序员能够迅速了解到代码背后的信息。完全没有注释的大段代码对于阅读者形同  
    天书，注释是给自己看的，即使隔很长时间，也能清晰理解当时的思路；注释也是给继任者看  
    的，使其能够快速接替自己的工作。
23. c【参考】好的命名、代码结构是自解释的，注释力求精简准确、表达到位。避免出现注释的  
    一个极端：过多过滥的注释，代码的逻辑一旦修改，修改注释是相当大的负担。  
    反例：  
    // put elephant into fridge  
    put(elephant, fridge);  
    方法名 put，加上两个有意义的变量名 elephant 和 fridge，已经说明了这是在干什么，语  
    义清晰的代码不需要额外的注释。
24. c【参考】特殊注释标记，请注明标记人与标记时间。注意及时处理这些标记，通过标记扫描，  
    经常清理此类标记。线上故障有时候就是来源于这些标记处的代码。  
    1） 待办事宜（TODO）:（ 标记人，标记时间，[预计处理时间]）  
    表示需要实现，但目前还未实现的功能。这实际上是一个 Javadoc 的标签，目前的 Javadoc  
    还没有实现，但已经被广泛使用。只能应用于类，接口和方法（因为它是一个 Javadoc 标签）。  
    2） 错误，不能工作（FIXME）:（标记人，标记时间，[预计处理时间]）  
    在注释中用 FIXME 标记某代码是错误的，而且不能工作，需要及时纠正的情况。
25. c【参考】volatile 解决多线程内存不可见问题。对于一写多读，是可以解决变量同步问题，  
    但是如果多写，同样无法解决线程安全问题。如果是 count++操作，使用如下类实现：  
    AtomicInteger count = new AtomicInteger(); count.addAndGet(1); 如果是 JDK8，推  
    荐使用 LongAdder 对象，比 AtomicLong 性能更好（减少乐观锁的重试次数）。
26. c【参考】 HashMap 在容量不够进行 resize 时由于高并发可能出现死链，导致 CPU 飙升，在  
    开发过程中可以使用其它数据结构或加锁来规避此风险。
27. c【参考】ThreadLocal 无法解决共享对象的更新问题，ThreadLocal 对象建议使用 static  
    修饰。这个变量是针对一个线程内所有操作共享的，所以设置为静态变量，所有此类实例共享  
    此静态变量 ，也就是说在类第一次被使用时装载，只分配一块存储空间，所有此类的对象(只  
    要是这个线程内定义的)都可以操控这个变量。
28. c【参考】合理利用好集合的有序性(sort)和稳定性(order)，避免集合的无序性(unsort)和不稳定性(unorder)带来的负面影响。说明：有序性是指遍历的结果是按某种比较规则依次排列的。稳定性指集合每次遍历的元素次序是一定的。如：ArrayList 是 order/unsort；HashMap 是 unorder/unsort；TreeSet order/sort。
29. c【参考】利用 Set 元素唯一的特性，可以快速对一个集合进行去重操作，避免使用 List 的contains 方法进行遍历、对比、去重操作。
30. c【参考】枚举类名建议带上 Enum 后缀，枚举成员名称需要全大写，单词间用下划线隔开。  
    说明：枚举其实就是特殊的常量类，且构造方法被默认强制是私有。  
    正例：枚举名字为 ProcessStatusEnum 的成员名称：SUCCESS / UNKOWN\_REASON。
31. a【强制】线上应用不要依赖 SNAPSHOT 版本（安全包除外）。  
    说明：不依赖 SNAPSHOT 版本是保证应用发布的幂等性。另外，也可以加快编译时的打包构建。
32. a【强制】二方库的新增或升级，保持除功能点之外的其它 jar 包仲裁结果不变。如果有改变，  
    必须明确评估和验证，建议进行 dependency:resolve 前后信息比对，如果仲裁结果完全不一  
    致，那么通过 dependency:tree 命令，找出差异点，进行排除 jar 包。
33. a【强制】二方库里可以定义枚举类型，参数可以使用枚举类型，但是接口返回值不允许使用枚  
    举类型或者包含枚举类型的 POJO 对象。
34. a【强制】依赖于一个二方库群时，必须定义一个统一的版本变量，避免版本号不一致。  
    说明：依赖 springframework-core,-context,-beans，它们都是同一个版本，可以定义一  
    个变量来保存版本：${spring.version}，定义依赖的时候，引用该版本。
35. b【推荐】给 JVM 设置-XX:+HeapDumpOnOutOfMemoryError 参数，让 JVM 碰到 OOM 场景时输出  
    dump 信息。  
    说明：OOM 的发生是有概率的，甚至有规律地相隔数月才出现一例，出现时的现场信息对查错  
    非常有价值。
36. b【推荐】在线上生产环境，JVM 的 Xms 和 Xmx 设置一样大小的内存容量，避免在 GC 后调整堆  
    大小带来的压力。
37. c【参考】服务器内部重定向使用 forward；外部重定向地址使用 URL 拼装工具类来生成，否则  
    会带来 URL 维护不一致的问题和潜在的安全风险
38. a【强制】所有的类都必须添加创建者和创建日期。
39. a【强制】方法内部单行注释，在被注释语句上方另起一行，使用//注释。方法内部多行注释  
    使用/\* \*/注释，注意与代码对齐。
40. a【强制】所有的枚举类型字段必须要有注释，说明每个数据项的用途。
41. a【强制】在一个 switch 块内，每个 case 要么通过 break/return 等来终止，要么注释说明程  
    序将继续执行到哪一个 case 为止；在一个 switch 块内，都必须包含一个 default 语句并且  
    放在最后，即使它什么代码也没有。
42. a【强制】在 if/else/for/while/do 语句中必须使用大括号。即使只有一行代码，避免采用单行的编码方式：if (condition) statements;