KISS 原则（Keep It Simple, Stupid)

思想为源

1．单元测试必须测试的三种场景：

A．正常数据场景：用来测试代码的主逻辑

B．边界数据场景：用来测试代码（或数据）在边界的情况下逻辑是否正确

C．异常数据场景：用来测试出现异常非故障时能否按照预期运行

2．依赖抽象（依赖倒置原则）：尽量抽象；表面类型必须是抽象的；任何类都不应该从具体类派生；尽量不要覆写基类的方法；抽象不关注细节。

Java 多线程

1．继承自 Thread 类的多线程不必也不能覆写 start 方法

2．不适用 stop 方法停止线程：stop 方法会导致代码逻辑不完整，一旦执行 stop 方法，即终止正在运行的线程，不管线程逻辑是否完整；stop 方法会破坏原子逻辑（同步代码块）

3．Interrupted() 方法不能终止一个线程，只会改变中断标志位，可以是用 isInterrupted() 方法判断

4．线程优先级只使用三个等级 -- 优先级常量：Thread.MAX_PRIORITY; Thread.NORM_PRIORITY; Thread.MIN_PRIORITY.

5．使用线程异常处理器提升系统可靠性。 setUncaughtExceptionHandler() 实现线程异常的捕捉和处理。需要注意：共享资源锁定问题；脏数据引起系统逻辑（原子操作被中断）；内存溢出

6．volatile 关键字：确保每个线程对本地变量的访问和修改都是直接与主内存交互的，而不是与本线程的工作内存交互，保证每个线程都能获得最新的变量值。

7．Volatile 不能保证数据是同步的，只能保证线程能够获得最新值。

8．异步运算考虑使用 Callable 接口，该接口实现多线程可以有返回值。使用 Executors 静态工具类调用线程池的实现涉及到：工作线程 (Worker)；任务接口 (Task)；任务队列 (Work Queue)，使用线程池减少的事线程的创建和销毁时间。

9. Executors 中创建线程池的便捷方法：

A. 单线程池：newSingleThreadExecutor，单线程池

B. 缓冲功能的线程池：newCachedThreadPool，线程数量没有限制（不能超过 Integer 的最大值

C. 固定线程数量的线程池：newFixedThreadPool，初始化时已经决定了线程的最大数量，若任务添加的能力超出了线程处理的能力，则建立阻塞队列容纳多余的任务。

10. 显式锁（Lock 类）和 内部锁（sychronized 关键字）

A. 显式锁的锁定和释放必须在一个 try ... finally 块中。

B. 对同步资源来说，显式锁是对象级别的锁，而内部锁是类级别的锁，也就是说 Lock 锁是跟随对象的，synchronized 是跟随类的。

C. Lock 和 synchronized 不同：

- Lock 支持更细粒度的锁控制。 支持读写锁 (ReentrantReadWriteLock().readLock())
- Lock 可实现公平锁，synchronized 只能实现非公平锁。公平锁：选择等待时间最长的线程获得锁
- Lock 是代码级的，synchronized 是 JVM 级的

11. 阻塞队列的长度是固定的

12. CountDownLatch 协调子线程。CountDownLatch 类是一个倒数的同步计数器。(见 java_thread.Runner)

13. CyclicBarrier 让多线程齐步走，调用其 await() 方法

性能和效率

提升 Java 性能的基本方法：

A. 不要在循环条件中计算 while(i<count*2);

B. 尽可能把变量、方法声明为 final static 类型；

C. 缩小变量的作用范围；

D. 频繁字符串操作使用 StringBuilder （线程不安全）或者 StringBuffer（线程安全）；

E. 覆写 Exception 的 fillInStackTrace，如果我们在开发时不需要关注栈信息，覆盖之，该方法比较耗时；不建立冗余对象

枚举和注解

1. 使用枚举定义常量　在编译期间限定类型，不允许发生越界的情况；枚举具有内置方法；枚举可以自定义方法；

2. 使用构造函数协助描述枚举项

3. switch 带来空值异常

4. switch 的 default 代码块中增加 AssertionError 错误

5. 枚举项的数量限制在 64 个以内

泛型和反射

1. Java 的泛型是类型擦除的，Java 的泛型在编译期有效，在运行期被删除

2. 不能初始化泛型参数和数组

3. 不同的场景使用不同的泛型通配符：

   A. 泛型结构只参与“读”操作则界定上限（extends 关键字）

   B. 泛型结构只参与“写”操作则界定下限（super 关键字）

4. 泛型不能协变和逆变。协变和逆变是指宽类型和窄类型在某种情况下替换或交换的特性。协变是用一种窄类型替换宽类型，而逆变则是用宽类型覆盖窄类型。Java 为了保证运行期的安全性，必须保证泛型参数类型是固定的。

5. 严格限定泛型类型采用多重界限 <T extends Staff & Passenger>

6. 获得一个 Class 对象有三种途径：

   A. 类属性方式：String.class；B. 对象的 getClass 方式：new String().getClass()；

   C. forName 方法加载：Class.forName(“java.lang.String”).

7. getDeclaredMethod 方法获得的是自身类的所有的方法，包括公有方法，私有方法，不受限于访问权限；getMethod 方法获得的是所有的 public 访问级别的方法，包括从父类继承的方法。

8. 反射访问属性或方法时将 Accessible 设置为 true：动态修改一个类或方法或执行方法时都会受 Java 安全体系的制约，而安全处理非常消耗资源，因此对运行期要执行的方法或要修改的属性就提供了 Accessible 可选项：由开发者决定是否要逃避安全体系的检查。

9. 动态加载不适合数组

10. 动态代理可以使代理模式更加灵活

11. 使用反射增加装饰模式的普适性 example ： java_generic.Decorate

12. 反射模式用于模板方法 example： java_generic.TemplateMethodPattern

异常

1. 提倡异常封装，通过封装可以一次抛出多个异常

2. 不要在 finally 块中处理返回值

3. 不要在构造函数中抛出异常

一些准则

1. 不要让常量蜕变成变量

2. 三元操作符的类型务必一致

3. 避免带有变长参数的方法重载：变长参数必须是方法中的最后一个参数；一个方法不能定义多个变长参数

4. 覆写方法必须满足的条件：重写方法不能缩小访问权限；参数列表必须与被重写方法相同；返回类型必须与被重写方法的相同或者是其子类；重写方法不能抛出新的异常，或者超出父类范围的异常，但是可以抛出更少、更有限的异常或者不抛异常。

5. 覆写的方法参数与父类相同，不仅仅是类型、数量，还包括显示形式。(int ... p int [] p)

6. 少用静态导入。(import static ...) 导入的是具有明确、清晰表象意义的工具类

7. Serializable 接口（序列化标志接口）目的是为了可持久化。JVM 根据 SerialVersionUID 来判断一个类版本。

8. 显示声明 serialVersionUID 可以避免对象不一致
9. 异变任务使用脚本语言编写 example：java_rule.InvokeScript

基本类型

1. 用偶判断，不用奇判断。Java 中的取余运算是 ： dividend - dividend / divisor * divisor，即：-1%2 = -1，判断 a%2==-1 有问题
2. 用整数类型处理货币，或者使用 BigDecimal 处理货币，浮点数不精确。
3. 基本类型转换时使用主动声明的方式减少不必要的 bug.
4. 单元测试中，有一项测试叫边界测试。如果一个方法接受的是 int 类型的参数，三个值必须测：0、正最大、负最大。
5. 包装类型参与运算时，要做 null 值校验。
6. 谨慎包装类型的大小比较

类、对象及方法

1. 接口中不要存在实现代码
2. 静态变量一定要先声明后赋值
3. 不要覆写父类的静态方法。
4. 构造函数简化，再简化，不应该出现复杂的构造函数，也避免在构造函数中初始化其他类。
5. 覆写 equals 方法时，要注意 null 值情景，还有自反性
6. 覆写 equals 方法必须覆写 hashCode 方法。Map 类会根据 hash 值来确定位置。
7. 使用 package-info 类为包服务

字符串

1. 推荐使用 String 直接量赋值。直接赋值可以使用字符串池，new 会直接创建对象。
2. String 的 replaceAll(“”, “”) 第一个参数是正则表达式。要注意方法中传递参数的要求
3. String、StringBuffer、StringBuilder

A. String 类是不可变的量，创建后就不可修改。在字符串不经常变换的场景使用。

B. StringBuffer 是一个可变字符序列，并且它是线程安全的。频繁进行字符串的运算，并且在多线程环境中使用。

C. StringBuilder 同 StringBuffer 一样，只是它不是线程安全的。频繁进行字符串运算（拼接、替换、删除等），且运行在单线程环境中。

测试 StringBuilder 和 StringBuffer 线程安全 example：java_thread.StringBufferTest

4. 在复杂字符串操作中使用正则表达式

数组和集合

1. 性能考虑，数组是首选。

2. 警惕数组的浅拷贝 （ Arrays.copyOf(ori, new) ）

3. 非常有必要在集合初始化时声明容量。当知道大概容量时，初始化为其 1.5 倍

4. 最值计算时使用集合最简单，使用数组性能最优

5. 原始类型数组不能作为 Arrays.asList 的输入参数，否则会引起程序逻辑混乱。

6. asList 方法产生的 List 对象不可更改。add 和 remove 方法不可用

7. 列表遍历不是那么简单，适时选择最优的遍历方式。实现 RandomAccess 接口的不要用 Iterator 遍历。

8. subList 产生的列表只是一个视图，所有的修改动作直接作用于原表。

9. 使用 subList 处理局部列表，subList 生成子类表后，保持原列表为只读状态 (Collections.unmodifiableList())

10. Comparable 接口可以作为实现类的默认排序法，Comparator 接口则是一个人的扩展排序工具。

11. 集合中的元素必须做到 compareTo 和 equals 同步，实现了 compareTo 方法，就应该覆写 equals 方法

12. 更优雅的集合运算方式：

    A. 并集： list1.addAll(lsit2)

    B. 交集： list1.retainAll(list2)

    C. 差集： list1.removeAll(list2)

D. 无重复的并集： lis2.removeAll(list1); list1.addAll(list2);

13. 多线程使用 Vector 或 HashTable。Vector 是 ArrayList 的多线程版本，HashTable 是 HashMap 的多线程版本

14. Collections 中有好多处理集合的方法

15. SortedSet 接口只定义了在给集合加入元素时将其排序，不能保证元素修改后的排序结果。因此，TreeSet（继承了 SortedSet 接口）适用于不变量的集合数据排序。重新排序可使用： new TreeSet<Person>(new ArrayList<Person>(set));

16. 集合大家族：

A. List，实现 List 接口的集合主要有：ArrayList、LinkedList、Vector、Stack，其中 ArrayList 是一个动态数组，支持随机存储；LinkedList 是一个双向链表；Vector 是一个线程安全的动态数组；Stack 是一个对象栈，遵循先进后出原则。

B. Set，Set 是不包含重复元素的集合，其主要实现类有：EnumSet、HashSet、TreeSet，其中 EnumSet 是枚举类型专用 Set；HashSet 是以哈希码决定其元素位置的 Set，其原理与 HashMap 相似，提供快速的插入和查找方法；TreeSet 是一个自动排序的 Set。

C. Map，可以分为排序 Map 和非排序 Map,排序 Map 主要是 TreeMap 类，它根据 Key 值进行自动排序；非排序 Map 主要包括：HashMap、HashTable、Properties、EnumMap 等，其中 Properties 是 HashTable 子类，EnumMap 要求其 Key 必须是某一个枚举类型。

D. Queue，队列分为两类：阻塞式队列，队列满了以后再插入元素会抛出异常，主要包括 ArrayBlockingQueue、PriorityBlockingQueue、LinkedBlockingQueue，其中 ArrayBlockingQueue 是一个以数组方式实现的有界阻塞队列，PriorityBlockingQueue 是依照优先级组建的队列，LinkedBlockingQueue 是通过链表实现的阻塞队列；另一类是非阻塞队列，无边界的，只要内存允许，都可以持续追加元素，最常用的是 PriorityQueue 类。

E. Deque，双端队列，支持在头、尾插入和移除元素，主要实现：ArrayDeque、LinkedBlockingQueue、LinkedList.

F. 数组与集合最大的区别就是数组能容纳基本类型，而集合不行，更重要的一点是所有集合底层存储的都是数组。

G. 数组的工具类是 java.util.Arrays 和 java.lang.reflect.Array，集合的工具类是 java.util.Collections