# 技术管理专项作业训练：

# 要求：

选择你常用的语言进行梳理，写出一份关于这个语言的技术管理文档；

可参考google等大厂的规范；

按照：a、强制；b、推荐；c、允许 三个级别来写您的规范

内容不需要太多，至少包含40条规则；（大四同学至少25条）

提交时间：助教老师联系

方式：助教老师发布github链接

Java语言技术管理文档

强制规则

1. 遵循字段命名规范：非公共非静态字段以`m`开头，静态字段以`s`开头，静态`final`字段为`ALL\_CAPS\_WITH\_UNDERSCORES`。

2. 限制代码行长度：代码中每一行文本的长度不得超过100个字符，除非是注释行或导入语句行。

3. 处理异常：不得忽略任何异常，必须有原则地处理每个异常。例如：

java

void setServerPort(String value) {

try {

serverPort = Integer.parseInt(value);

} catch (NumberFormatException e) {

throw new IllegalArgumentException("Invalid port number: " + value);

}

}

4. 不捕获通用异常：禁止捕获`Exception`或`Throwable`，除非在测试代码和顶级代码中，并且必须添加注释说明原因。

5. 明确导入：必须使用`import foo.Bar;`明确导入类，不得使用`import foo.\*;`。

6. 使用标准大括号样式：左大括号不单独占一行，条件语句必须添加大括号。

7. 编写Javadoc注释：每个类和重要的公共方法都必须包含Javadoc注释，至少用一句话说明用途。

8. 谨慎使用日志记录：必须根据日志级别合理使用日志记录，避免在发布build中记录过多信息。

9. 测试方法命名规范：测试方法必须使用下划线将被测试的内容与被测试的具体案例区分开来。

10. 限制变量作用域：必须尽可能缩小局部变量的作用域。

11. 避免使用终结器：不得使用终结器，应使用良好的异常处理代替。

12. 使用标准Java注释：注释必须位于同一语言元素的其他修饰符之前，`@Deprecated`注解必须添加`@deprecated` Javadoc标记。

13. 保持代码风格一致：修改代码时，必须与周围代码的风格保持一致，包括空格、注释等。

推荐规则

1. 避免使用`System.out.println()`：推荐避免使用`System.out.println()`或`printf()`进行日志记录。

2. 使用try-with-resources：推荐使用`try-with-resources`语句来自动关闭资源，避免资源泄漏。

3. 编写清晰的Javadoc注释：推荐编写清晰、详细的Javadoc注释，遵循相关规范。

4. 避免日志记录隐私信息：推荐避免在日志中泄露安全方面的信息和隐私信息。

5. 使用枚举代替常量类：推荐使用枚举类型代替常量类，以提高代码的可读性和可维护性。

6. 避免捕获通用异常：推荐捕获具体的异常类型，而不是通用的`Exception`或`Throwable`。例如：

java

try {

someComplicatedIOFunction(); // may throw IOException

someComplicatedParsingFunction(); // may throw ParsingException

someComplicatedSecurityFunction(); // may throw SecurityException

} catch (IOException e) {

handleError(e);

} catch (ParsingException e) {

handleError(e);

} catch (SecurityException e) {

handleError(e);

}

7. 避免过长的参数列表：推荐尽量避免方法或构造函数的参数列表过长，以提高代码的可读性。

8. 使用多捕获块：如果多个异常需要相同的处理逻辑，推荐使用多捕获块。例如：

java

try {

...

} catch (ClassNotFoundException | NoSuchMethodException e) {

handleError(e);

}

9. 避免过度使用嵌套：推荐尽量减少嵌套的深度，以提高代码的可读性。

10. 合理使用日志记录：推荐在捕获异常时记录详细的日志信息，以便后续排查问题。例如：

java

try {

...

} catch (Exception e) {

log.error("An unexpected error occurred", e);

}

11. 编写简短方法：推荐编写短小精炼的方法，尽量不超过40行代码。

12. 在标准位置定义字段：推荐在文件顶部或紧接在使用字段的方法之前定义字段。

13. 为import语句排序：推荐按照Android包、第三方包、`java`和`javax`的顺序排序import语句。

14. 使用空格缩进：推荐使用4个空格缩进块，不使用制表符。

15. 遵循字段命名规范：推荐将首字母缩写词视为字词，使名称更具可读性。

16. 使用TODO注释：推荐使用TODO注释作为短期的临时解决方案，并包含具体日期或事件。

17. 限制日志记录频率：推荐实施频率限制机制，避免出现大量重复日志副本。

18. 重构长方法：推荐将长方法拆解为多个短方法，以提高代码的可读性和可维护性。

19. 避免重复捕获异常：推荐避免在多个层次重复捕获相同的异常，以减少冗余代码。例如：

java

try {

someMethod();

} catch (Exception e) {

handleError(e);

}

而不是：

java

try {

try {

someMethod();

} catch (Exception e) {

handleError(e);

}

} catch (Exception e) {

handleError(e);

}

允许规则

1. 捕获异常并抛出`RuntimeException`：在某些情况下，允许捕获异常并抛出一个新的`RuntimeException`实例，但必须确保这是最合适的处理方式。例如：

java

void setServerPort(String value) {

try {

serverPort = Integer.parseInt(value);

} catch (NumberFormatException e) {

throw new RuntimeException("Invalid port number: " + value, e);

}

}

2. 使用多行注释：允许使用多行注释，但必须确保其不会导致代码可读性下降。

3. 捕获多个异常：允许将多个异常作为多个`catch`块的一部分分别进行捕获。

4. 重构代码以捕获异常：允许通过多个`try`块重构代码，获得更精细的错误处理过程。

5. 在`try-catch`块中声明变量：如果通过一个会抛出受检异常的方法的返回值来初始化变量，允许在`try`块中进行初始化。

6. 使用Java 8特性：允许使用Java 8特性，如Lambda表达式、Stream API等，但必须确保团队成员熟悉这些特性。

7. 使用匿名内部类：允许使用匿名内部类，但必须确保其不会导致代码可读性下降。

8. 使用嵌套类：允许使用嵌套类，但必须确保其不会导致代码结构过于复杂。

9. 使用单行注释：允许使用单行注释，但必须确保其清晰、简洁。

10. 使用`if (LOCAL\_LOG)`块：允许使用`if (LOCAL\_LOG)`块来控制日志记录，但必须确保其中不包含有效逻辑。

11. 使用`if (LOCAL\_LOGV)`块：允许使用`if (LOCAL\_LOGV)`块来控制`VERBOSE`级别的日志记录。

12. 使用`String` +运算符：允许使用`String`对象的`+`运算符进行字符串拼接，但必须注意性能影响。

13. 捕获异常并记录日志：允许捕获异常并记录日志，但必须确保日志记录不会对性能产生过大影响。例如：

java

void setServerPort(String value) {

try {

serverPort = Integer.parseInt(value);

} catch (NumberFormatException e) {

log.warn("Invalid port number: " + value, e);

}

}

14. 忽略异常：如果确信忽略异常是合适的处理方式，允许忽略异常，但必须添加注释充分说明理由。例如：

java

void setServerPort(String value) {

try {

serverPort = Integer.parseInt(value);

} catch (NumberFormatException e) {

// Method is documented to ignore invalid user输入.

// serverPort will remain unchanged.

}

}

15. 使用第三方库：允许使用第三方库，但必须确保其与项目兼容，并且不会引入安全问题。