Java代码规范

一、命名风格

命名的统一规约

1. **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。**
2. **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。**

   正确的英文拼写和语法可以让阅读者易于理解，避免歧义。
3. **杜绝完全不规范的缩写，避免望文不知义**

   比如 Abstract 不可缩写为 Abs，要用全称

   此类随意缩写严重降低了代码的可阅读性
4. **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。**

具体元素的命名

1. **类名** 统一使用【帕斯卡命名法】，即每一个单词的第一个字母都要大写
2. **方法名、参数名、成员变量、局部变量**都统一使用【小驼峰命名法】，即第一个单词首字母小写，后面其他单词的首字母都要大写
3. **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
4. **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。
5. **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。

   indi： 个体项目，指个人发起，但非自己独自完成的项目，可公开或私有项目，版权主要属于发起者。

   包名为indi.发起者名.项目名.模块名..*　　比如 indi.wyj.calculate.judgenumber

   pers： 个人项目，指个人发起，独自完成，可分享的项目，版权主要属于个人。

   包名为pers.个人名.项目名.模块名..*　　比如 pers.wyj.calculate.judgenumber

   priv： 私有项目，指个人发起，独自完成，非公开的私人使用的项目，版权属于个人。

   包名为priv.个人名.项目名.模块名..*　　比如 priv.wyj.calculate.judgenumber

   team： 团队项目指由团队发起，并由该团队开发的项目，版权属于该团队所有。

   包名为team.团队名.项目名.模块名..*

   com： 公司项目：由项目发起的公司所有。

   包名为com.公司名.项目名.模块名..*
6. 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。

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。

7. 中括号[]是数组类型的一部分，数组定义如下：String[] args；而不能使用String args[];
8. **具体的 方法 命名规约**

   1） 获取单个对象的方法用 get 做前缀。

   2） 获取多个对象的方法用 list 做前缀。

   3） 获取统计值的方法用 count 做前缀。

   4） 插入的方法用 save/insert 做前缀。

   5） 删除的方法用 remove/delete 做前缀。

   6） 修改的方法用 update 做前缀。

二、代码格式

1. 大括号的使用约定。如果是大括号内为空，则简洁地写成{}即可，不需要换行；如果是非空代码块则：
   1） 左大括号前不换行。
   2） 左大括号后换行。
   3） 右大括号前换行。
   4） 右大括号后还有 else 等代码则不换行；表示终止的右大括号后必须换行。
2. 左小括号和字符之间不出现空格；同样，右小括号和字符之间也不出现空格。
3. **if / for / while / switch / do 等保留字**与括号之间都必须加空格。
4. 任何二目、三目运算符的左右两边都需要加一个空格。

   运算符包括 赋值运算符=、逻辑运算符&&、加减乘除符号等。
5. 采用 4 个空格缩进，禁止使用 tab 字符。

   除非你的tab键设置为一个tab为4个空格
6. 注释的双斜线与注释内容之间有且仅有一个空格

1—6示例

```
public static void main(String[] args) {
    // 缩进 4 个空格
    String say = "hello";
    // 运算符的左右必须有一个空格
    int flag = 0;
    // 关键词 if 与括号之间必须有一个空格，括号内的 f 与左括号，0 与右括号不需要空格
    if (flag == 0) {
        // 注释的开始位置与所注释语句对齐，这一条见下面【注释规约】
        System.out.println(say);
    }

    // 左大括号前加空格且不换行；左大括号后换行
    if (flag == 1) {
        System.out.println("world");
    // 右大括号前换行，右大括号后有 else，不用换行
    } else {
        System.out.println("ok");
```

```
    // 在右大括号后直接结束，则必须换行
    }
}
```

7. **单行字符数限制不超过 120 个，超出需要换行，换行时遵循如下原则：**
    1） 第二行相对第一行缩进 4 个空格，从第三行开始，不再继续缩进，参考示例。
    2） 运算符与下文一起换行。
    3） 方法调用的点符号与下文一起换行。
    4） 方法调用时，多个参数，需要换行时，在逗号后进行。
    5） 在括号前不要换行，见反例

8. **方法参数在定义和传入时，多个参数逗号后边必须加空格。**

正例：
```
    StringBuffer strbuf = new StringBuffer();
    // 超过 120 个字符的情况下，换行缩进 4 个空格，点号和方法名称一起换行
    strbuf.append("hello").append("编程语言")...
    .append("java")...
    .append("python")...
    .append("C++");
    method(args1, args2, args3, ...,
    argsX);
```
反例：
```
    StringBuffer strbuf = new StringBuffer();
    // 超过 120 个字符的情况下，不要在括号前换行
    sb.append("hello").append("编程语言")...append
    ("java");
    // 参数很多的方法调用可能超过 120 个字符，不要在逗号前换行
    method(args1, args2, args3, ...
    , argsX);
```

9. **方法体内的执行语句组、变量的定义语句组、不同的业务逻辑之间或者不同的语义之间插入一个空行。相同业务逻辑和语义之间不需要插入空行。**
    即没必要加入多个空行来分隔不同的逻辑代码

三、OO规约(Object Oriented 规约)

1. 避免通过一个类的对象引用访问此类的静态变量或静态方法，无谓增加编译器解析成本，统一直接使用类名来访问静态属性和方法
2. **所有的覆写方法，必须加@Override 注解。**
    它可以强制一个子类必须重写父类方法或者实现接口的方法，
    这样可以防止由于拼写等低级错误而导致的方法没有被重写，进而出现的bug
3. **当一个类有多个构造方法，或者多个同名方法，这些方法应该按顺序放置在一起，便于阅读**
4. 类内方法的排列顺序依次是：公有方法或保护方法 > 私有方法 > getter/setter方法
    第4条规则优于该条规则

公有方法和保护方法是类的调用者最关心的方法，也是最需要维护的

私有方法属于黑盒实现的，承载的信息较低，放在最后

5. setter 方法中，参数名称与类成员变量名称一致，this.成员名 = 参数名。

   getter/setter 方法中，不要增加其他逻辑，增加排查问题的难度(即此类方法仅用于检索和更新)

6. 任何类、方法、参数、变量，严控访问范围。过于宽泛的访问范围，不利于模块解耦。

四、语句元素

控制结构

1. 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。

2. 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，比如 if (condition) statements;

五、注释规约

1. **类、类属性、类方法的注释必须使用 Javadoc 规范，使用 /**内容 */格式，不得使用 // xxx 注释**
2. **所有的抽象方法（包括接口中的方法）必须要用 Javadoc 注释、**

   除了返回值、参数、异常说明外，还必须指出该方法做什么事情，实现什么功能。

   对子类的实现要求，或者调用注意事项，需要一并说明

3. 所有的类都必须添加创建者和创建日期
4. 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**

5. **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。**

   如果注释与实际代码更新不同步，落后太多，就失去了注释的意义

6. **谨慎注释掉代码。在上方详细说明，而不是简单的注释掉。如果无用，则删除。**

   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

7. 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，使其能够快速理解自己写的代码

8. 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，应当尽力避免无用的注释