Java代码规范   
1. 标识符命名规范

* 1. 概述

标识符的命名力求做到统一、达意和简洁。

* + 1. 统一

统一是指，对于同一个概念，在程序中用同一种表示方法，比如对于供应商，既可以用supplier，也可以用provider，但是我们只能选定一个使用，至少在一个Java项目中保持统一。统一是最为重要的，如果对同一概念有不同的表示方法，会使代码混乱难以理解。即使不能取得好的名称，但是只要统一，阅读起来也不会太困难，因为阅读者只要理解一次。

1.1.2 达意

达意是指，标识符能准确的表达出它所代表的意义，比如： newSupplier, OrderPaymentGatewayService等；而 supplier1, service2，idtts等则不是好的命名方式。准确有两层含义，一是正确，二是丰富。如果给一个代表供应商的变量起名是 order，显然没有正确表达。同样的，supplier1 远没有targetSupplier意义丰富。   
1.1.3 简洁

简洁是指，在统一和达意的前提下，用尽量少的标识符。如果不能达意，宁愿不要简洁。比如：theOrderNameOfTheTargetSupplierWhichIsTransfered 太长， transferedTargetSupplierOrderName则较好，但是transTgtSplOrdNm就不好了。省略元音的缩写方式不要使用，我们的英语往往还没有好到看得懂奇怪的缩写（一层次大神除外）。   
1.1.4 骆驼法则

Java中，除了静态常量等特殊情况，大部分情况下标识符使用骆驼法则，即单词之间不使用特殊符号分割，而是通过首字母大写（第一个单词首字母小写）来分割。比如: supplierName, addNewContract，而不是 supplier\_name, add\_new\_contract。   
1.1.5 英文 vs 拼音

尽量使用通俗易懂的英文单词，如果不会可以向队友求助，实在不行则使用汉语拼音，避免拼音与英文混用。比如表示归档，用archive比较好, 用pigeonhole则不好，用guiDang尚可接受。   
1.2 包名

使用小写字母如 com.xxx.settlment，不要 com.xxx.Settlement 单词间不要用字符隔开，比如 com.xxx.settlment.jsfutil，而不要com.xxx.settlement.jsf\_util   
 建议使用edu.nju.xxx作为包名  
1.3 类名   
1.3.1 首字母大写

类名要求首字母大写，比如 SupplierService, PaymentOrderAction；不要 supplierService, paymentOrderAction.   
1.3.2 后缀

类名往往用不同的后缀表达额外的意思，如下表：

|  |  |  |
| --- | --- | --- |
| 后缀名 | 意义 | 举例 |
| Service | 这个类是服务类，里面包含了给其他类提同业务服务的方法 | PaymentOrderService |
| Impl | 这个类是一个实现类，而不是接口 | PaymentOrderServiceImpl |
| Inter | 这个类是一个接口 | LifeCycleInter |
| Dao | 这个类封装了数据访问方法 | PaymentOrderDao |
| Listener | 响应某种事件的类 | PaymentSuccessListener |
| Event | 这个类代表了某种事件 | PaymentSuccessEvent |
| Factory | 生成某种对象工厂的类 | PaymentOrderFactory |

1.4 方法名

首字母小写，如 addOrder() 不要 AddOrder()   
动词在前，如 addOrder()，不要orderAdd()   
动词前缀往往表达特定的含义，如下表：

|  |  |  |
| --- | --- | --- |
| 前缀名 | 意义 | 举例 |
| create | 创建 | createOrder() |
| delete | 删除 | deleteOrder() |
| add | 创建，暗示新创建的对象属于某个集合 | addPaidOrder() |
| remove | 删除 | removeOrder() |
| init | 初始化，暗示会做些诸如获取资源等特殊动作 | initObjectPool |
| destroy | 销毁，暗示会做些诸如释放资源的特殊动作 | destroyObjectPool |
| open | 打开 | openConnection() |
| close | 关闭 | closeConnection() |
| read | 读取 | readUserName() |
| write | 写入 | writeUserName() |
| get | 获得 | getName() |
| set | 设置 | setName() |
| prepare | 准备 | prepareOrderList() |
| copy | 复制 | copyCustomerList() |
| modity | 修改 | modifyTotalAmount() |
| calculate | 数值计算 | calculateCommission() |
| dispatch | 判断程序流程转向 | dispatchUserRequest() |
| start | 开始 | startOrderProcessing() |
| stop | 结束 | stopOrderProcessing() |
| send | 发送某个消息或事件 | sendOrderPaidMessage() |
| receive | 接受消息或时间 | receiveOrderMessgae() |
| respond | 响应用户动作 | responseOrderClicked() |
| find | 查找对象 | findNewSupplier() |
| update | 更新对象 | updateCommission() |

1.5 域（field）名   
1.5.1 静态常量

全大写用下划线分割，如 ：

public static final String ORDER\_PAID\_EVENT = "ORDER\_PAID\_EVENT";   
1.5.2 枚举

全大写，用下划线分割，每个枚举常量占一行，如 ：

|  |
| --- |
| public enum Events {  ORDER\_PAID,  ORDER\_CREATED  } |

1.5.3 其他

首字母小写，骆驼法则，如：

public String orderName;   
1.6 局部变量名

参数和局部变量名首字母小写，骆驼法则。

2. 代码格式

2.1 源文件编码

源文件使用UTF-8编码

2.2 缩进

缩进使用Tab缩进

2.3 包的导入

删除不用的导入，尽量不要使用整个包的导入

在eclipse下可使用快捷键 ctrl+shift+O 修正导入

2.4 类格式 、方法格式 、代码块格式 、缩进风格

大括号的开始在代码块开始的行尾，闭合在和代码块同一缩进的行首，例如：

|  |
| --- |
| package com.test;  public class TestStyle extends SomeClass implements AppleInter, BananaInter {  public static final String THIS\_IS\_CONST = "CONST VALUE";  private static void main(String[] args) {  int localVariable = 0;  }  public void compute(String arg) {  if (arg.length() > 0) {  System.out.println(arg);  }  for (int i = 0; i < 10; i++) {  System.out.println(arg);  }  while (condition) {  }  do {  otherMethod();  } while (condition);  switch (i) {  case 0:  callFunction();  break;  case 1:  callFunctionb();  break;  default:  break;  }  }  } |

2.6.2 空格的使用   
2.6.2.1 表示分割时用一个空格

变量与小括号之间不用空格，不能这样：

|  |
| --- |
| if ( a > b ) {  //do something here  }; |

2.6.2.2 二元三元运算符两边用一个空格隔开

如下：

|  |
| --- |
| a + b = c;  b - d = e;  return a == b ? 1 : 0; |

不能如下：

|  |
| --- |
| a+b=c;  b-d=e;  return a==b?1:0; |

2.6.2.3 逗号语句后如不换行，紧跟一个空格   
如下：   
call(a, b, c);   
不能如下：   
call(a,b,c);   
2.6.3 空行的使用   
空行可以表达代码在语义上的分割，注释的作用范围，等等。将类似操作，或一组操作放在一起不用空行隔开，而用空行隔开不同组的代码， 如图：

|  |
| --- |
| order = orderDao.findOrderById(id);  //update properties  order.setUserName(userName);  order.setPrice(456);  order.setStatus(PAID);  orderService.updateTotalAmount(order);  session.saveOrUpdate(order); |

上例中的空行，使注释的作用域很明显.   
    连续两行的空行代表更大的语义分割。   
    方法之间用空行分割   
    域之间用空行分割   
    超过十行的代码如果还不用空行分割，就会增加阅读困难   
3. 注释规范   
3.1 注释 vs 代码

注释宜少而精，不宜多而滥，更不能误导 。命名达意，结构清晰，类和方法等责任明确，往往不需要，或者只需要很少注释，就可以让人读懂；相反，代码混乱，再多的注释都不能弥补。所以，应当先在代码本身下功夫。 不能正确表达代码意义的注释，只会损害代码的可读性。过于详细的注释，对显而易见的代码添加的注释，罗嗦的注释，还不如不写 。

注释要和代码同步，过多的注释会成为开发的负担   
注释不是用来管理代码版本的，如果有代码不要了，直接删除，svn会有记录的，不要注释掉，否则以后没人知道那段注释掉的代码该不该删除。   
3.2 Java Doc   
表明类、域和方法等的意义和用法等的注释，要以javadoc的方式来写。Java Doc是给类的使用者来看的，主要介绍“是什么，怎么用”等信息。凡是类的使用者需要知道的，都要用Java Doc 来写。非Java Doc的注释，往往是给代码的维护者看的，着重告述读者为什么这样写，如何修改，注意什么问题等。 如下：

|  |
| --- |
| /\*\*  \* This is a class comment  \*/  public class TestClass {  /\*\*  \* This is a field comment  \*/  public String name;  /\*\*  \* This is a method comment  \*/  public void call() {  }  } |

3.3 块级别注释   
块级别注释，单行时用 //, 多行时用 /\* .. \*/。   
较短的代码块用空行表示注释作用域   
较长的代码块要用   
/\*------ start: ------\*/   
和   
/\*-------- end: -------\*/ 包围

如：

|  |
| --- |
| /\*----------start: 订单处理 ------- \*/  //取得dao  OrderDao dao = Factory.getDao("OrderDao");  /\* 查询订单 \*/  Order order = dao.findById(456);  //更新订单  order.setUserName("uu");  order.setPassword("pass");  order.setPrice("ddd");  orderDao.save(order);  /\*----------end: 订单处理 ------- \*/ |

3.4 行内注释   
行内注释用 // 写在行尾   
4 最佳实践和禁忌   
4.1 每次保存的时候，都让你的代码是最美的

程序员都是懒惰的，不要想着等我完成了功能，再来优化代码的格式和结构，等真的把功能完成，很少有人会再愿意回头调整代码。   
4.2 每个if while for等语句，都不要省略大括号{}   
看下面的代码：

|  |
| --- |
| if (a > b)  a++; |

如果在以后维护的时候，需要在a > b 时，把b++，一不小心就会写成：

|  |
| --- |
| if (a > b)  a++;  b++; |

这样就错了，因为无论a和b是什么关系，b++都会执行。 如果一开始就这样写：

|  |
| --- |
| if (a > b) {  a++;  } |

相信没有哪个笨蛋会把b++添加错的。而且，这个大括号使作用范围更明显，尤其是后面那行很长要折行时。   
4.4 善用TODO:   
在代码中加入 //TODO: ，大部分的IDE都会帮你提示，让你知道你还有什么事没有做。比如：

|  |
| --- |
| if (order.isPaid()) {  //TODO: 更新订单  } |

4.5 在需要留空的地方放一个空语句或注释，告述读者，你是故意的   
比如：

|  |
| --- |
| if (!exists(order)) {  ;  } |

或：

|  |
| --- |
| if (!exists(order)) {  //nothing to do  } |

4.6 不要再对boolean值做true false判断   
比如：

|  |
| --- |
| if (order.isPaid() == true) {  // Do something here  } |

不如写成：

|  |
| --- |
| if (order.isPaid()) {  //Do something here  } |

后者读起来就很是 if order is paid, .... 要比 if order's isPaid method returns true, … 更容易理解   
4.7 减少代码嵌套层次   
代码嵌套层次达3层以上时，一般人理解起来都会困难。下面的代码是一个简单的例子：

|  |
| --- |
| public void demo(int a, int b, int c) {  if (a > b) {  if (b > c) {  doJobA();  } else if (b < c) {  doJobB()  }  } else {  if (b > c) {  if (a < c) {  doJobC();  }  }  }  } |

减少嵌套的方法有很多：   
    合并条件   
    利用 return 以省略后面的else   
    利用子方法   
比如上例，合并条件后成为：

|  |
| --- |
| public void demo(int a, int b, int c) {  if (a > b && b > c) {  doJobA();  }  if (a > b && c > b) {  doJobB();  }  if (a <= b && c < b && a < c) {  doJobC();  }  } |

如果利用return 则成为：

|  |
| --- |
| public void demo(int a, int b, int c) {  if (a > b) {  if (b > c) {  doJobA();  return;  }  doJobB()  return;  }  if (b > c) {  if (a < c) {  doJobC();  }  }  } |

利用子方法，就是将嵌套的程序提取出来放到另外的方法里。   
4.8 程序职责单一

关注点分离是软件开发的真理。人类自所以能够完成复杂的工作，就是因为人类能够将工作分解到较小级别的任务上，在做每个任务时关注更少的东西。让程序单元的职责单一，可以使你在编写这段程序时关注更少的东西，从而降低难度，减少出错。

4.9 变量的声明，初始化和被使用尽量放到一起   
比方说如下代码：

|  |
| --- |
| int orderNum= getOrderNum();  //do something withou orderNum here  call(orderNum); |

上例中的注释处代表了一段和orderNum不相关的代码。orderNum的声明和初始化离被使用的地方相隔了很多行的代码，这样做不好，不如这样：

|  |
| --- |
| //do something withou orderNum here  int orderNum= getOrderNum();  call(orderNum); |

4.10 缩小变量的作用域   
能用局部变量的，不要使用实例变量，能用实例变量的，不要使用类变量。变量的生存期越短，以为着它被误用的机会越小，同一时刻程序员要关注的变量的状态越少。实例变量和类变量默认都不是线程安全的，局部变量是线程安全的。比如如下代码：

|  |
| --- |
| public class OrderPayAction{  private Order order;  public void doAction() {  order = orderDao.findOrder();  doJob1();  doJob2();  }  private void doJob1() {  doSomething(order);  }  private void doJob2() {  doOtherThing(order);  }  } |

上例中order只不过担当了在方法间传递参数之用，用下面的方法更好：

|  |
| --- |
| public class OrderPayAction{  public void doAction() {  order = orderDao.findOrder();  doJob1(order);  doJob2(order);  }  private void doJob1(Order order) {  doSomething(order);  }  private void doJob2(Order order) {  doOtherThing(order);  }  } |

4.11 尽量不要用参数来带回方法运算结果   
比如：

|  |
| --- |
| public void calculate(Order order) {  int result = 0;  //do lots of computing and store it in the result  order.setResult(result);  }  public void action() {  order = orderDao.findOrder();  calculate(order);  // do lots of things about order  } |

例子中calculate方法通过传入的order对象来存储结果， 不如如下写：

|  |
| --- |
| public int calculate(Order order) {  int result = 0;  //do lots of computing and store it in the result  return result;  }  public void action() {  order = orderDao.findOrder();  order.setResult(calculate(order));  // do lots of things about order  } |