**Java Programming Style Guidelines**

By 11545296 Jaehoon Lee

**Reviewer:** Jaehoon Lee

**To Whom:** Dylan Qiao

1. **General recommendation**

This document lists java coding recommendations common in the java development community and from my opinion to your codes that you have been fixed.

This recommendation documents are mainly based on my (Jaehoon’s Opinion) ideas, standards collected from number of sources, local requirements and needs, as well as suggestions given in different examples.

This document is basically showing several reasons for introducing a new guideline rather than what you have done in your baseline code to give idea of mix style issues with language technical issues, the present document does not contain java technical recommendations, it focuses mainly on programming style and techniques of what you could use in different areas that you have been working on.

Development environment in this case (NetBeans) can improve the readability of code by access visibility, color coding, automatic formatting and so on, the programmer should never rely on such features. Source code should always be considered larger than the IDE it is developed within and should be written in a way that maximizes its readability independent of any IDE.

There are number of things examples in this guideline document for example:

1. Layout of the recommendation
2. Recommendation importance
3. Automatic style checking
4. Naming Conventions
5. General naming conventions
6. Specific naming conventions

And etc...

1. **Layout of the Recommendation**

The recommendations are grouped by topic and each recommendation is numbered to make it easier to refer to during reviews.

Layout for the recommendation is as follows:

|  |
| --- |
| Guideline short description |
| Example if applicable |
| Motivation, background and additional information. |

* 1. **recommendation importance**

in the guideline section the terms must, should and can have special meaning. A must requirement must be followed, a should is a strong recommendation, and a can is a general guideline.

1. **general recommendations**

|  |
| --- |
| 1. Any violation to the guide is allowed if it enhances readability. |
| The main goal of the recommendation is to improve readability and thereby the understanding and the maintainability and general quality of the code. |

1. **Naming Convention**

|  |
| --- |
| Names are too complicated. |
| carparkId, numberOfCarsParked, adhocTicketDAO.createTicket |
| These names are too complicated, and common practice in the java development community and also the naming convention used by Sun for the Java core packages.  In general the use of such constants should be minimized.  @Override  public boolean isFull() {  return (numberOfCarsParked >= capacity);  } |

**3.1 name representing**

|  |
| --- |
| Names representing methods must be verbs and written in mixed case starting with lower case. |
| getName() , isFull() ,IAdhocTicket issueAdhocTicket() |
| Common practice in the Java development community and also the naming convention used by Sun for the Java core packages. This is identical to variable names, but methods in Java are already distinguishable from variables by their specific form. |

* 1. **Abbreviation and acronyms should not be uppercase when used as name.**

|  |
| --- |
| Names representing methods must be verbs and written in mixed case starting with lower case. |
| IAdhocTicket(); // not IADOCTicket();  issueAdhocTicket() // not ISSUEAdocTicket(); |
| Using all uppercase for the base name will give conflicts with the naming conventions given above. A variable of this type whould have to be named dVD, hTML etc. which obviously is not very readable. Another problem is illustrated in the examples above; When the name is connected to another, the readability is seriously reduced; The word following the acronym does not stand out as it should. |

* 1. **Private class variables should have a letter “f” prefix. “f” comes from the world field; object field**

|  |
| --- |
| Class Name{  Private String fName;  ….  } |
| Apart from its name and its type, the *scope* of a variable is its most important feature. Indicating class scope by using "f" prefix makes it easy to distinguish class variables from local scratch variables. This is important because class variables are considered to have higher significance than method variables, and should be treated with special care by the programmer.  A side effect of the "f" prefix naming convention is that it nicely resolves the problem of finding reasonable variable names for setter methods:  void setName(String name){  fName = name;  }  It should be noted that scope identification in variables have been a controversial issue for quite some time. It seems, though, that this practice now is gaining acceptance and that it is becoming more and more common as a convention in the professional development community. |

* 1. **interator variables should be called I , j ,k etc.**

|  |
| --- |
| @Override  public void recordAdhocTicketExit(IAdhocTicket ticket) {  numberOfCarsParked--;  adhocTicketDAO.removeFromCurrentList(ticket);  for (int i = 0; i < observers.size(); i++){  observers.get(i).notifyCarparkEvent();  }  } |
| The notation is taken from mathematics where it is an established convention for indicating iterators. Variables named j, k etc. should be used for nested loops only. |

* 1. **Complement names must be used for complement entities.**

|  |
| --- |
| Get/ set , add/remove , create/destroy, / start/stop, insert/delete, min/max |
| Reduce complexity by symmetry. |

**3.6 negated Boolean variable names must be avoided.**

|  |
| --- |
| @Override  public boolean isSeasonTicketValid(String ticketId) {  ISeasonTicket seasonTicket = seasonTicketDAO.findTicketById(ticketId);  return (seasonTicket != null) && (System.currentTimeMillis() >= seasonTicket.getEndValidPeriod());  } |
| The problem arise when the logical not operator is used and double negative arises. It is not immediately apparent what !isNotError means. |

**3.7 abbreviations and acronyms should not be uppercase when used as name.**

|  |
| --- |
| @Override  public float calculateAdHocTicketCharge(long entryDateTime) {  long stayTime = System.currentTimeMillis() - entryDateTime;  float fifteenMinuteLotsStayed = (stayTime / FIFTEEN\_MINUTES) + 1;  return fifteenMinuteLotsStayed \* FIFTEEN\_MINUTE\_PRICE;  } |
| Using all uppercase for the base name will give conflicts with the naming conventions given above. A variable of this type whould have to be named dVD, hTML etc. which obviously is not very readable. Another problem is illustrated in the examples above; When the name is connected to another, the readability is seriously reduced; The word following the acronym does not stand out as it should. |

**3.8 singleton classes should return their sole instance through method getInstance**

|  |
| --- |
| public boolean isSeasonTicketInUse(String ticketId) {  ISeasonTicket seasonTicket = seasonTicketDAO.findTicketById(ticketId);  return seasonTicket.getCurrentUsageRecord() != null;  } |
| Common practice in the Java community though not consistently followed by Sun in the JDK. The above layout is the preferred pattern. |

**4.0 - SPACING**

**ALL method names should be immediately followed by a left parenthesis.**

|  |
| --- |
| args [0]; // no!  args[0]; // Yes! |

**4.1 binary operators should have a space on either side.**

|  |
| --- |
| A=b+c // no!  A = b+c; // no!  A=b + c // no!  A = b + c ; // yes! |

**4.2 commas and semicolons are always followed by whitespace.**

|  |
| --- |
| For (int i = 0; I < 10 ; i++) // no!  For (int I = 0; I < 10; I ++) // yes! |

**4.3 The keywords if , while , for , switch , and catch must be followed by a space.**

|  |
| --- |
| if(hungry) // NO!f  if (hungry) // YES!    while(pancakes < 7) // NO!  while (pancakes < 7) // YES!    for(int i = 0; i < 10; i++) // NO!  for (int i = 0; i < 10; i++) // YES!    catch(TooManyPancakesException e) // NO!  catch (TooManyPancakesException e) // YES! |

**5. Class member ordering**

|  |
| --- |
| Class Name  {  // fields  // constructors  // methods |

**6. All public classes and public and protected functions within public classes should be documented using the Java documentation (javadoc) conventions.**

This makes it easy to keep up-to-date online code documentation

|  |
| --- |
| This makes it easy to keep up-to-date online code documentation |

**Reference**

[1]java programming style guideline

*https://www.cs.bgu.ac.il/~majeek/presentations/JavaProgrammingStyle%20Guidelines.html – 2015*

[2] java ranch- java programming style guideline

*https://www.javaranch.com/styleLong.jsp*

[3] java code conventions

*http://java.sun.com/docs/codeconv/html/CodeConvTDC.doc.html*