Java Coding Standards for FIT9131

Short Version for Introductory Programming Courses[i]
Faculty of Information Technology, Monash University

You are expected to follow these Java coding standards when writing Java code in FIT9131 this semester. Every time a standard is not adhered to, the reason must be clearly documented.

1. General Code Layout

1.1. Use a space around operators (arithmetic, relational, logical, conditional, assignment) and keywords. For example:

```
int total = counter + 1;
```

1.2. Use a space between `if`, `switch`, `while` and the following parenthesis. For example:

```
if (index > 12)
    …
```

1.3. Do not use a space before the parenthesis in a method call. For example:

```
aPerson.setAge(30);
```

1.4. Place each statement on a separate line. An exception is in a `switch` statement where a `break` may be placed on the same line as the preceding statement.

1.5. Define and initialise each variable or field (attribute) on a separate line. For example:

```
int total = 0;
int index = 0;
```

1.6. Place a blank line between methods (and constructors).

2. Layout of Class Declaration

2.1. Place each class in a separate file.

2.2. The elements of a class should be declared in the following order:

import statements; *class comment*; *class header*; *field definitions*; *constructors*; *methods*.

2.3. Methods in a class declaration should be arranged in alphabetical order of the name of method.

3. Brackets and Indenting

3.1. Use a consistent level of indentation. Three or four spaces is suggested.

3.2. Indent your code one level per block (pair of braces), as in the example in 3.5.

3.3. Vertically align matching braces. Note that this convention differs from that used in the textbook *Objects First with Java*, Barnes & Kölling.

3.4. Use braces in `if ..else` and loop constructs only when necessary to define blocks of code with more than one statement. Note that this convention differs from that used in the textbook *Objects First with Java*, Barnes & Kölling.

3.5. Align second and additional conditions for `if` statements that run over multiple lines under the first condition after the parenthesis. Align arguments in method calls that run over multiple lines under the first argument. Take care that no code extends past the side of the screen or page.

```
public boolean setName(String newGivenNames,
                       String newFamilyName)
{
    if (newGivenNames.equals("") ||
        newFamilyName.equals(""))
        System.out.println("Name incomplete");
    else
    {
        givenNames = new String(newGivenNames);
        familyName = new String(newFamilyName);
    }
}
```

4. Declaring and Initialising Variables

4.1. Initialise all variables (except fields in a class declaration) when they are created.

4.2. Explicitly define all fields and methods as `private`, `protected` or `public`.

4.3. Fields should not be specified as `public` (except for `final` fields).

4.4. Constant fields should be declared as `static`.

5. Naming

5.1. Use descriptive and meaningful names for all identifiers (names of classes, methods and variables). Avoid ambiguous names. Avoid abbreviations.

5.2. Use the convention whereby names with multiple words are joined and have uppercase letters at the start of the second and successive words. This is sometimes called *camel* case. Do not use underscore, minus or punctuation characters to separate words.

5.3. Class names start with a capital letter and have an uppercase letter starting every word in their name. Java requires that a class be stored in a file with the same name as the class and with a `.java` extension (e.g: `CreditCardAccount.java`). Therefore the class:

```
public class CreditCardAccount
{ …
```

would be stored in a file called `CreditCardAccount.java`.

Class names such as `creditCardAccount`, `creditcardaccount` and `CREDITCARDACCOUNT` are not allowed.

5.4. Method names start with a lowercase letter and the first letter of each word after that is in uppercase. For example, the name of a method to print a bank statement could be:

```
public printBankStatement(int accountNumber)
```

Method names such as `printbankstatement`, `PrintBankStatement`, `PRINTBANKSTATEMENT` and `print_bank_statement` are not allowed.

5.5. Object and variable names start with a lowercase letter and the first letter of each word after that is in uppercase (the same as for methods). For example:

```
int positionCounter = 0;
String familyName = new String();
```

Variable names such as `POSITIONCOUNTER`, `position_counter` and `PositionCounter` are not allowed.

5.6. Constants (or final variables) are written in uppercase with underscores (_) to separate words. For example:

```
final int MAX_LINES_ON_SCREEN = 24;
```

6. Documentation

6.1. Place a comment at the top of each method and class definition to explain its purpose.

6.2. Include comments in code only when necessary to explain complex code. Note that, where possible, it is preferable to write clear comprehensible code.

6.3. Any comments written INSIDE a method to clarify the code should be written as in the example below:

```
int index = -1;  // -1 indicates that there are no
                 // entries in the array
```

6.4. Alternatively, if the comment will not fit on one line, as in the previous example, write the comment on a line of its own, as follows:

```
// -1 indicates that there are no entries in the array
int index = -1;
```

6.5. BlueJ will create a skeleton comment at the top of the class, which you are to fill out. The comment should include (at least) the purpose of the class, the author(s) and the date created. Every person who contributed to the class must be named as an author or otherwise appropriately acknowledged.

```
/**
 * Store the details of a credit card,
 * provide validation, balance, repayment and
 * purchase information
 * @author      J. Jones
 * @version     1.3 27 Jan 2008
 */
```

6.6. Class comments must be recognised by `Javadoc`. In other words, they should start with the comment symbol `/**`

6.7. For each extra line of comment you include, you must add * to the start of the line. The * is to be preceded by one space and followed by one space before the start of that line of the comment.

See the textbook *Objects First with Java*, Barnes & Kölling for further details of Javadoc and other options available.

6.8. Document each method in a similar manner to a class, following the template given by BlueJ.

[i] 1These coding standards were developed from the Java Programming Standards, published by Doug Lea http://gee.cs.oswego.edu/ and the Program Style Guide in Appendix J in Objects First with Java, Barnes & Kölling.