Code Conventions

# File Organization

## Beginning Comments:

All source ﬁles should begin with a c-style comment that lists the programmer(s), the date, a copyright notice, and also a brief description of the purpose of the program. For example:

/\*

\* Classname

\* Version info

\*

\* Copyright notice

\*/

## **Package and Import Statements:**

The ﬁrst non-comment line of most Java source ﬁles is a package statement. After that,

import statements can follow. For example:

package java.awt;

import java.awt.peer.CanvasPeer;

## **Indentation:**

4 spaces should be used as the unit of indentation.

## **Line Length:**

Avoid lines longer than 80 characters.

## **Wrapping Lines:**

When an expression will not ﬁt on a single line, break it according to these general principles:

* Break after a comma.
* Break before an operator.

Here are some examples of breaking method calls:

function(longExpression1, longExpression2, longExpression3,

longExpression4, longExpression5);

var = function1(longExpression1,

function2(longExpression2,

longExpression3));

# Comments

All kinds of comments should start with a capital letter, end without period and with a space between the comments character (// or /\*).

## **Block Comments:**

A block comment should be preceded by a blank line to set it apart from the rest of the code.

Block comments have an asterisk “\*” at the beginning of each line except the ﬁrst.

/\*

\* Here is a block comment

\* with two or more lines

\*/

## **Single-Line Comments:**

A single-line comment should be preceded by a blank line. Here’s an example

of a single-line comment in Java

if (condition) {

/\* Handle the condition. \*/

...

}

## **End of Line Comments:**

It shouldn’t be used on consecutive multiple lines for text comments; however, it can be used in consecutive multiple lines for commenting out sections of code. Examples of all three styles follow

if (something > 1) {

...

}

else

return false; // Explain why here.

//if (something < 1)

// ...

//}

//else

// return false;

## **Documentation Comments:**

Doc comments describe Java classes, interfaces, constructors, methods, and ﬁelds. Each doc

comment is set inside the comment delimiters /\*\*...\*/, with one comment per API.

/\*\*

\* The Example class provides ...

\*/

class Example { ...

# Declarations

## **Number per Line**

One declaration per line is reccomended since it allows any comments. Example:

int size; // Size of something

int age; // Age of someone

## **Placement**

Put declarations only at the beginning of blocks. Don’t wait to declare variables until their ﬁrst use; it can confuse the unwary programmer.

void MyMethod() {

int int1; // beginning of method block

if (condition) {

int int2; // beginning of "if" block

...

}

}

The one exception to the rule is indexes of *for* loops, which in Java can be declared in the *for*

statement:

for (int i = 0; i < maxLoops; i++) { ...

## **Initialization:**

Local variables should be initializaed where they are declared. The one reason not to initialize variables where they were declared is when this variable needs na computational ocurring.

## **Class and Interface Declarations**

* No space between a method name and the parenthesis “(“ starting its parameter list
* Open brace “{” appears at the end of the same line as the declaration statement
* Closing brace “}” starts a line by itself indented to match its corresponding opening

statement, except when it is a null statement the “}” should appear immediately after the “{“

class Sample extends Object {

int ivar1;

int ivar2;

Sample(int i, int j) {

ivar1 = i;

ivar2 = j;

}

int emptyMethod() {}

...

}

• Methods are separated by a blank line

# Statements

## **return Statements**

A *return* statement with a value should not use parentheses unless they make the return value

more obvious in some way. Example:

return;

return myDisk.size();

return (size ? size : defaultSize);

## **if, if-else Statements:**

The if-else class of statements should have the following form:

if (condition) {

statements;

}

if (condition) {

statements;

} else {

statements;

}

if (condition) {

statements;

} else if (condition) {

statements;

} else if (condition) {

statements;

}

Note: if statements always use braces {}. Avoid the following form:

if (condition) //AVOID! THIS OMITS THE BRACES {}!

statement;

## **while Statements**

A while statement should have the following form:

while (condition) {

statements;

}

## **do-while Statements:**

A do-while statement should have the following form:

do {

statements;

} while (condition);

## **switch Statements:**

A switch statement should have the following form:

switch (condition) {

case ABC:

statements;

break;

case DEF:

statements; break;

case XYZ:

statements; break;

default:

statements;

break;

}

## **try-catch Statements:**

A try-catch statement should have the following format:

try {

statements;

} catch (ExceptionClass e) {

statements;

}

# White Space

Blank spaces should be used in the following circumstances:

* A keyword followed by a parenthesis should be separated by a space. Example:

while (true) {

...

}

Note that a blank space should not be used between a method name and its opening parenthesis. This helps to distinguish keywords from method calls.

* A blank space should appear after commas in argument lists.
* All binary operators except . should be separated from their operands by spaces. Blank

spaces should never separate unary operators such as unary minus, increment (“++”), and

decrement (“--”) from their operands. Example:

a += c + d;

a = (a + b) / (c \* d);

while (d++ = s++) {

n++;

}

prints("size is " + foo + "\n");

* The expressions in a for statement should be separated by blank spaces. Example:

for (expr1; expr2; expr3)

* Casts should be followed by a blank. Examples:

myMethod((byte) aNum, (Object) x);

myFunc((int) (cp + 5), ((int) (i + 3)) + 1);