**Java Coding Standards**

**1. Documentation**

Documentation is critical for coding. You must document each method using javadoc tags. You also must document within functions to show what conditionals and difficult code is doing.

**1.1 Method Documentation**

Please use the following javadoc standards which are explained by the following example using the notes to far right.

|  |  |
| --- | --- |
| /\*\* | begin doc with /\*\* |
| \* Returns the sum of two result values by first converting them to  \* either integer or float values. The result is based on the data  \* type of the first argument.  \* <p> | This is a one or two sentence summary of the purpose of the method. It is ended by a <p>. |
| \* The returned value is a ResultValue. The value is populated with  \* a string representation of the sum. The type is set to either  \* Token.INTEGER or Token.FLOAT.  \* | Further explanation of what this method does. |
| \* @param parser provides the error method for raising errors  \* @param resOp1 first operand for the addition  \* @param resOp2 second operand for the addition | Each parameter must be documented. |
| \* @return the sum of the two operands as a ResultValue. | Summarize what is being returned. |
| \* @throws Exception if either operand is not numeric | If the method raises an exception, explain what it returns. |
| \*/ | End the documentation with an \*/ |

static public ResultValue add(Parser parser, ResultValue resOp1, ResultValue resOp2) throws Exception

{

ResultValue res = new ResultValue();

// Convert to numeric values

Numeric numOp1 = new Numeric(parser, resOp1, "+", "1st operand");

Numeric numOp2 = new Numeric(parser, resOp2, "+", "2nd operand");

// the data type of the result is based on the data type of the first operand

switch(numOp1.type)

{

case Token.INTEGER:

// operand1 is an integer, result will be integer

int intValue = 0;

switch(numOp2.type)

{

case Token.INTEGER:

intValue = numOp1.integerValue + numOp2.integerValue;

break;

case Token.FLOAT:

intValue = numOp1.integerValue + (int) numOp2.doubleValue;

break;

default:

parser.error("unknown data type for second operand of add: '%s'"

, resOp1.value);

}

res.value = Integer.toString(intValue);

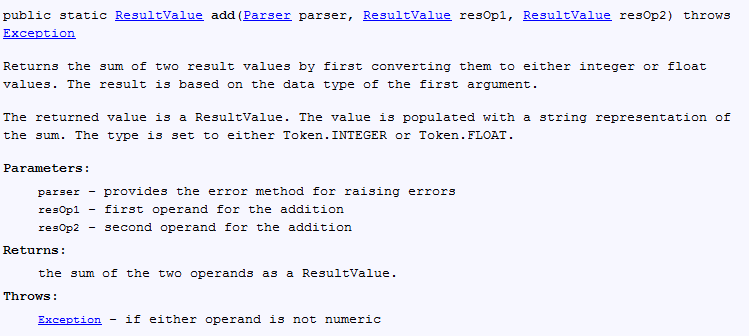
res.type = Token.INTEGER;

break;

case Token.FLOAT:

...

The javadoc utility would generate this documentation based on the above:



**1.2 Documentation within methods**

Document each conditional (if, switch, for, while). Explain what it is doing.

Document more difficult code to explain what it is doing.

# Indentation and White Space

Indenting code and including some white space makes code easier to follow.

## Flow Control Constructs

Flow control constructs (e.g., if, for, while, switch) need indentation. Indent the statements within these constructs one tab. Braces used to syntactically allow multiple statements should be at the same level of indentation as the control statement.

Line up beginning and ending curly braces.

For an **if** which is on the **else** branch of another **if**, it is unnecessary to indent the **else if**.

Note: within your source code editor, specify the replacement of tabs with spaces. Tabs should be set to four spaces.

Examples:

// Look for grades after the midterm date

for (element : gradeM)

{

// check for grades after midterm date

if (element.dueDate.compareTo(midTermDate)> 0)

{ // It is after the midterm date

// Check for a non-existing grade or 0

if (element.grade >= 0)

{ // Include it in our average

count += 1;

sum += element.grade;

}

// check for an empty grade

else if (element.grade == -1)

{ // Student didn't turn in this assignment

// Keep track of the last date of an empty

if (element.dueDate.compareTo(lastDate) > 0)

lastDate = element.dueDate;

// count the number of empty grades

emptyCount += 1;

}

}

}

## Subroutine Indentation

The declarations and code for a subroutine should be indented one tab.

## White Space

White space is used to make blocks of code and comments stand out. It is often useful to show that comments apply to a particular block of code using white space.

Example:

// Adjust the student record to show that the student is active

// with a declared major.

student.lastEnrollDate = currentDate;

student.statusCd = STATUS\_ACTIVE;

student.declaredMajor = major;

// Update the student record on the database

rc = updateStudent (student);

**3. Variable Naming**

The first letter in variable names must begin with a lower case letter. For arrays, it is easier to simply end the name with an "M" to indicate multiple occurrences. Use mixed camel case for variable names:

declaredMajor

symbolTable

studentM

i, j, rc (return code), and element are common names for control variables.

When possible, start variables with the name of the class or a reasonable abbreviation:

Customer customer;

Customer customerAssociatedParty;

ResultValue resOperand1;

Token tokenOperand1;

You may want to start int variables with i, double variables with d, and boolean variables with b.

**4. getter and setter methods**

Most businesses having coding standards that require the use of get*AttributeName* and set*AttributeName* methods for accessing and setting the values of an object. For software requiring high speed, the overhead does not justify the use of getter and setter methods. For CS4713, we do not want to use getter and setter methods.