## Coding Style Rules

<table>
<thead>
<tr>
<th>Rules</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Classes</strong></td>
<td></td>
</tr>
<tr>
<td>• Use nouns</td>
<td>QueueBlock, ReversalADT, LinkedList, CustomerAccount, not MaintainCustomerData</td>
</tr>
<tr>
<td>• Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).</td>
<td></td>
</tr>
<tr>
<td>✓ Begin with upper case letters</td>
<td></td>
</tr>
<tr>
<td>• First letter of each internal word capitalized</td>
<td></td>
</tr>
<tr>
<td><strong>Methods</strong></td>
<td></td>
</tr>
<tr>
<td>• Use verbs</td>
<td>examine, delete, isEmpty, toString</td>
</tr>
<tr>
<td>✓ Begin with lower case letters</td>
<td></td>
</tr>
<tr>
<td>• First letter of each internal word capitalized</td>
<td></td>
</tr>
<tr>
<td><strong>Instance Variables</strong></td>
<td></td>
</tr>
<tr>
<td>• Use nouns, plurals for collection classes</td>
<td>items, firstAction, list</td>
</tr>
<tr>
<td>✓ Begin with lower case letters</td>
<td></td>
</tr>
<tr>
<td>• First letter of each internal word capitalized</td>
<td></td>
</tr>
<tr>
<td>• Do not start with underscore _ or dollar sign $ characters, even though both are allowed.</td>
<td></td>
</tr>
<tr>
<td>• One-character variable names should be avoided except for common temporary variables.</td>
<td></td>
</tr>
<tr>
<td><strong>Constants</strong></td>
<td></td>
</tr>
<tr>
<td>✓ All characters in upper case</td>
<td>private static final int MAX_SEQUENCE</td>
</tr>
<tr>
<td>• Words are separated by underscores (&quot; &quot;)</td>
<td></td>
</tr>
<tr>
<td><strong>Files</strong></td>
<td></td>
</tr>
<tr>
<td>✓ File names should be the same as the (principal) class stored in the file.</td>
<td></td>
</tr>
<tr>
<td><strong>Layout</strong></td>
<td></td>
</tr>
<tr>
<td>✓ No space between a method name and the parenthesis &quot;(&quot; starting its parameter list</td>
<td>public class QueueBlock {</td>
</tr>
<tr>
<td>✓ Open brace &quot;){&quot; appears at the end of the same line as the declaration statement</td>
<td>.</td>
</tr>
<tr>
<td>✓ Closing brace &quot;}&quot; starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the &quot;}&quot; should appear immediately after the &quot;{&quot;</td>
<td>.</td>
</tr>
<tr>
<td>✓ The internal statements are indented by 2 spaces.</td>
<td>. public int isZero (){return 0;}</td>
</tr>
<tr>
<td>.</td>
<td>. }</td>
</tr>
<tr>
<td>Layout</td>
<td>Methods</td>
</tr>
<tr>
<td>--------</td>
<td>---------</td>
</tr>
</tbody>
</table>
| ✓ A *method* should either appear on a single line, as is shown for `isZero` in the example above, or surrounding its statements in the same way as a class.  
✓ Note that the opening brace ends the first line of the method's code, and the closing brace is on its own line flush with the left side of the first line (*not*, for example, at the end of the fifth line) This has two advantages:  
  o separates the text of the method "declaration" from its internal statements,  
  o allows the reader to determine where a method ends by looking down for a brace with the same level of indentation.  
✓ The internal statements are indented by 2 spaces. | ✓ A *method* should either appear on a single line, as is shown for `isZero` in the example above, or surrounding its statements in the same way as a class.  
✓ Note that the opening brace ends the first line of the method's code, and the closing brace is on its own line flush with the left side of the first line (*not*, for example, at the end of the fifth line) This has two advantages:  
  o separates the text of the method "declaration" from its internal statements,  
  o allows the reader to determine where a method ends by looking down for a brace with the same level of indentation.  
✓ The internal statements are indented by 2 spaces. |
| ✓ An *iteration* statement should be formatted in the same way as a method  
✓ If an iteration statement has only one clause but it is too long to appear on a single line, it should be indented on the following line. | ✓ An *iteration* statement should be formatted in the same way as a method  
✓ If an iteration statement has only one clause but it is too long to appear on a single line, it should be indented on the following line. |
| • public String toString () {  
  int i;  
  String s="";  
  for (i=first; i<=last; i++) {  
    s = s+items[i].toString()+"\n";  
  }  
  return s;  
} | • public String toString () {  
  int i;  
  String s="";  
  for (i=first; i<=last; i++) {  
    s = s+items[i].toString()+"\n";  
  }  
  return s;  
} |

```java
public String toString () {  
  int i;  
  String s="";  
  for (i=first; i<=last; i++) {  
    s = s+items[i].toString()+"\n";  
  }  
  return s;  
}
```
| Selection Statements |  • Rules for iteration statement apply to the *if* part and the *else* part (or to the *switch* part and to each case).  
• The example illustrates a case where one part appears on a single line and the other part is broken around a block of statements.  
✓ Braces and semicolons should always be included.  
✓ Write if statements in their simplest form.  
(Warning: Checkstyle’s error message for this is “Conditional logic can be removed”) |
| --- | --- |
| if (!isFull()) {  
    last++;  
    items[last] = a;  
} else throw new ExceptionFull("enqueueing to full queue"); |
| Not: if (foo) {  
    return true;  
} else {  
    return false;  
} Instead: return (foo); |

| Indentation | ✓ All code and documentation should not be longer than **80** characters per line.  
  o Some editors will wrap lines of text without inserting carriage-return characters.  
  o When the code is then viewed on a non-wrapping editor, paragraphs will appear as a single line.  
  o Also text entered in a wider window will format differently on an 80 character window.  
 ✓ When an expression will not fit on a single line, break it according to these general principles:  
  o Break after a comma.  
  o Break before an operator.  
  o Prefer higher-level breaks to lower-level breaks.  
  o Align the new line with the beginning of the expression at the same level on the previous line.  
  o If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>if (!isEmpty()) { list = list.succ; }</td>
</tr>
</tbody>
</table>

| Spacing | ✓ Two blank lines should always be used in the following circumstances:  
  o Between sections of a source file  
  o Between class and interface definitions |
| --- | --- |
| Layout | Spacing | ✓ One blank line should always be used in the following circumstances:  
|        |         |   o Between methods  
|        |         |   o Between the local variables in a method and its first statement  
|        |         |   o Before a block or single-line comment  
|        |         |   o Between logical sections inside a method to improve readability |   
|        | Javadoc for Classes | o All classes and their fields (both class and instance variables) and methods should be preceded by javadoc comments. |   
|        | Javadoc for Methods | ✓ Documentation for methods should include:  
|        |         |   ✓ A sentence description of the method’s purpose, terminated with a full stop  
|        |         |   ✓ a @return statement whenever the return type is not void  
|        |         |   ✓ a @param statement for each parameter passed to the method  
|        |         |   ✓ an @exception statement for any exception thrown |   
|        | Javadoc for Instance Variables | ✓ Documentation for instance variable should include the use of the instance variable |   
|        | Block Comments | ● Used to provide descriptions of files, methods, data structures and algorithms. |   

```java
/**
* A basic recursive (linked) list
* @author your name (and any authors of included or modified code)
* @version the date
*/

protected Object examine (){    
    if (!isEmpty()) return list.item;    
}

/**
* contains the user password
*/
private String password;

/**
* Here is a block comment.
*/
```
## Coding Style Rules

### Single-line Comments
- May be used at the beginning of each file, before each method and other places, such as within methods.
- Block comments inside a function or method should be indented to the same level as the code they describe.

```java
class MyClass {
    public void method1() {
        /* single-line comment */
        // code...
    }
}
```

### Trailing Comments
- Very short comments can appear on the same line as the code they describe.
- Should be shifted far enough to separate them from the statements.
- If more than one short comment appears in a chunk of code, they should all be indented to the same tab setting.

```java
if (condition) {
    /* trailing comment */
    ...
}
```

### End-of-line Comments
- The // comment delimiter can comment out a complete line or only a partial line.
- Do not use on consecutive multiple lines for text comments.
- Can be used in consecutive multiple lines for commenting out sections of code.

```java
while (b < 0) {
    //comment 1
    b++; //comment 2
    //code
    //code
    //code
}
```

- Indicates that the rule is checked by Checkstyle but needs to be corrected manually.
- Indicates that the rule is checked by Checkstyle and can be automatically corrected using Eclipse Formatter.

### Documentation

#### Single-line Comments
- Short comments can appear on a single line indented to the level of the code that follows.
- If a comment can't be written in a single line, it should follow the block comment format.
- A single-line comment should be preceded by a blank line.

```java
if (condition) {
    /* short comment here */
    ...
}
```

#### Trailing Comments
- Very short comments can appear on the same line as the code they describe.
- Should be shifted far enough to separate them from the statements.
- If more than one short comment appears in a chunk of code, they should all be indented to the same tab setting.

```java
if (b != 0) {
    return 0; /* trailing comment 1 */
} else {
    return 10; /* trailing comment 2 */
}
```

### Material taken and modified from:
- Code Conventions for the Java Programming Language (http://java.sun.com/docs/codeconv/)
- CITS1220 and CITS2200 Java Programming Conventions