Java Style Guidelines

Commenting: Each class must have a cursory header comment describing what behaviors the class provides, each method containing logic (i.e. not getters or setters) must have a cursory header comment describing what the function provides, and each major control structure must have a cursory comment describing the how control structure logic operates.

The right brace of each class and method should be followed by a // end comment such as

} // end main()

You can use either a C-Style comment, /\* … \*/, or Java style // comment at the top of the file for the class header comment, but all other comments should be Java style // comments unless they are JavaDoc comments.

Source file names must start with a capital letter, be letters and numbers only, and match the name of the public class with the main() method to be executed.

Line lengths should be kept within 80 characters (width of a normal terminal window) unless doing so is not possible or reduces code clarity. It is sometimes necessary to set automated formatters to 78 character line width in order to achieve an 80 character line width. Lines should not wrap around to the 1st column.

Indentation level must be 3 spaces each and there must not be any TABs in the source code

Braces alignment: Braces can either be vertically aligned or K&R aligned

**if (true) if (true) {**

**{ }**

**}**

Use Braces: Major logic statements (if, switch, for, while, do) MUST have an associated code block (i.e. set of braces) even if there is only one or no other statements associated with the logic statement. This is also required for case statements of more than 1 associated statement.

**e.g. if (x < y) {**

**x = y;**

**}**

Class and Variable names will follow the [camel naming convention](https://en.wikipedia.org/wiki/CamelCase) (except as noted).

Class names will start with a capital letter.

Class and method-local variable names will start with a lower-case letter.

Final constant names will be all capital letters with “\_” separating the logical words (ex. NAME\_INDEX).

Variables

Use Meaningful Variable Names: Variables and named constants should have meaningful names which are descriptive of their contents for example: accountBalance

Only declare one variable per statement.

To clearly distinguish between local and instance data, refer to all instance data with the prefix this.

**e.g. public void setAccountBalance(double accountBalance) {**

**this.accountBalance = accountBalance;**

**}**

Methods

Method names start with a lower-case letter and use the [camel naming convention](https://en.wikipedia.org/wiki/CamelCase)

Should only have one return statement (or no return in case of void method) when practical. Using additional returns to process error conditions if it improves readability of the code is acceptable.

Program method main() should be at the top of a class after any constructor(s).

Spacing

One space before and after operators

**e.g. i = 3;**

No space after left parens, braces, and brackets and no space before right parens, braces, and brackets

**e.g. (true) {12} [12]**

Use parens to improve understandability of logic.

Instead of while(i <= maxGuess - 1)

Code as while(i <= (maxGuess – 1))

Visibility within a class:   
Place all private below protected

Place all protected below public

Place all public at the top

Do not use Magic Numbers

e.g. Instead of

**case 2:**

**System.out.println("Print Queue: ");**

Use a final or enum to make the code more understandable

**case PRINT\_QUEUE:**

**System.out.println("Print Queue: ");**

All references to instance data should use this.

Helps to distinguish instance data from local data:

/\* Class Year holds the current year

\*/

**public class Year {**

**// return the Year**

**public int getYear() {**

**return this.year;**

**}**

**// set the Year**

**public void setYear(int year) {**

**this.year = year;**

**}**

**// Year defined as an int as all years are numeric integers**

**private int year;**

**} // end Year**

Java Design Guidelines

Visibility:

Use private to make fields accessible to the class only, all non-final instance variables should be private to support information hiding.

Use protected to make fields accessible to class, package, and subclasses

Use public to make fields accessible to everything

Do not use default visibility.

When should a method be static?

Ask yourself "does it make sense to call this method, even if no Object has been constructed yet?" If so, it should definitely be static, otherwise it should not be static.

When should a variable be static in a class?

Use static variables when the value of the variable is not unique for each object.

Conversely, if the value is unique for each object the variable must not be static.

Static variables make the design more complex as you need to make sure that the access to static variables in a multi-threaded environment is synchronized.

When should a local variable be static?

Variables declared as static inside a function are statically allocated while having the same scope as automatic local variables. Hence whatever values the function puts into its static local variables during one call will still be present when the function is called again.

When should a class be static

Use a static class to contain methods that are not associated with a particular object. For example, it is a common requirement to create a set of methods that do not act on instance data and are not associated to a specific object in your code. You could use a static class to hold those methods.