Conventions for Java sourcecode

Guidelines and conventions for the external form as well as suggestions for naming and documentation in Java programs are laid down in the document „Code Conventions for the Java™ Programming Language“.

http://www.oracle.com/technetwork/java/codeconvtoc-136057.html

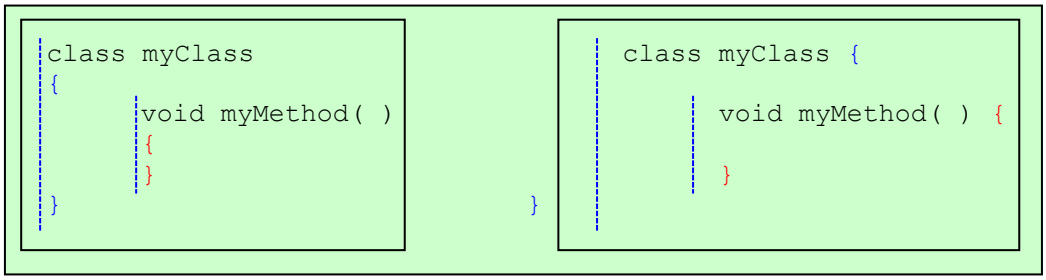
Format design (style)

The formatting of source code is not relevant to the Java compiler - within certain limits. He only knows a few immutable rules for correct spelling and syntax (expression and combination of instructions). The parser is not interested in linebreaks, which are important for the human reader. For the parser (code analyzer) a token is important to distinguish the different units:

- Spaces between units (type declarations and variable names)
- Semicolon at the end of a statement
- Correct provisions of pairs of brackets

In addition, of course, the compiler checks the spelling of units (instructions, key words). A misspelled statement can not be executed and produces an error message. User-defined data types (objects, variables, methods) are not allowed to use these reserved keywords (see table).

abstract	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	public	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	static	void
char	finally	long	strictfp	volatile
class	float	native	super	while
const	for	new	switch	
continue	goto	package	synchronized	



In general there is freedom of the external design of the source code. This shows up in the form of various programs in publications. There are different views about the particular arrangement of braces. Each programmer has to find out what kind appears to be better to her. Keep a style that shows the nesting of statement blocks (by indentation) clearly.

Recognized guidelines

In software companies detailed guidelines for formatting are common. They serve to unify sourcecode for larger projects involving multiple programmers at work. All participants must adhere to these rules. In addition, some formal guidelines have been coined that all programmers should note:

- Appropriate documentation
- Consistent adherence to the selected or specified formatting style

Designation

For naming variables and methods in the following lexical conventions have proven:

Class names should be *nouns* that describe the nature and importance of the class as short and to the point.The first letter is **capitalized.**You can also consist of several words that are strung together without spaces or underscore. The first letter of each word is capitalized.

```
My class First class
class college student
```

Method names should be *verbs* whose first letter is **lowercase.**Mated parts of a name without spaces / underscores concatenated with the first letter of each word is capitalized.

```
gibAus void ()
holMirDasSofortHer int ()
```

Variable names should be short and self-describing.The first letter is **lowercase.**Most letter names are used for count variables: i, j, k, m, for letters c, d, e

```
int i;
char c;
float own wide;
```

Constant names written with capitals, components are distinguished by underscore.

```
static final int min_width = 4;
static final int max_width = 999;
static final int GET_THE_CPU = 1;
```

Constants must not be instantly initialized :

```
static final int min_width;
. . .
min_width = 4;
```

Position of the type declarations

The arrangement of declarations in a source text is checked by the compiler. There are two different sets of rules for the positioning of type declarations. A distinction is made between the two levels, which are also relevant for the type of data types (see data types):

- Components at the class level (object variables and methods)
- Variables within a statement block (local variables)

a) Class components

Class-level attributes and methods can be declared at any point. Two types frequently encounter in example programs:

1) First attributes, then methods declarations

```
class myClass {

   int a;                // Attributes
   double b;

   void getter() {     // Methods
   }
   void setter() {
   }
}
```

2) First methods then attributes:

```
class myClass {

   void setter() {     // Methods
   }
   void getter() {
   }
   int a;                // Attributes
   double b;
}
```

The position of the attributedeclaration at the class level is indifferent to the compiler. The human analysis of a class is facilitated if the attributes (which are indeed referenced in the methods) are located at the **beginning** of the classcode. In the following chapters, the first variant (first attributes, then methods) is preferred.

Position of variable declarations in a method

In principle, variables can be declared within a method at any point, in any case, of course, before the instructions that operate on them. It is not absolutely necessary (although recommended) to declare variables at the very start of the instruction block (as in C).The same is true for constants.

```
void perimeter() {
  int a;                    // Variable
  final double PI = 3.14; // Konstante
  a = 3;
  System.out.println(„Perimeter " + (a*PI));
}
```

Here the preferred example of formatting :

```
class myCircle {

    int ganzZahl;                              // Attributes
    double kommaZahl;
    String name;

    void methode() {                           // Methods
         ganzZahl  = ganzZahl + 3;
         kommaZahl = kommaZahl + 2.14;
    }
}
```