Basics

Application Entry Point

```
public static void main(String[] args) {
    // TODO code application logic here
}
```

Naming Conventions

Class name

Should be CamelCase and start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.

Interface name

Should be CamelCase and start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.

Method name

Should be CamelCase and start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.

Variable name

Should be CamelCase and start with lowercase letter e.g. firstName, orderNumber etc.

Package name

Should be in lowercase letter e.g. java, lang, sql, util etc.

Constants name

Should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

Terminal

Output to terminal

```
System.out.println("Hello World");
```

Input from terminal

```
Scanner console = new Scanner(System.in);
console.next();        // String
console.nextInt();     // Integer
console.nextDouble();  // Double
```

Clean code

Don't make a mess of your code. Take care of structure, whitespace, indentation, ...

Code should read like a good story. It should be easy to understand while wanting you to keep reading.

Operators

Conditional operators

- **AND:** &&
- **OR:** ||
- **NOT:** !

Comparison operators

- **equal to:** ==
- **not equal to:** !=
- **greater than:** >
- **greater than or equal to:** >=
- **less than:** <
- **less than or equal to:** <=

Primitive Types

byte (8 bits) [+127 to -128]

```
byte frameStart = 65;
```

char (16 bits) [All Unicode characters]

```
char startOfAlfabet = 'a';
```

short (16 bits) [+32,767 to -32,768]

```
short numberOfPagesInBook = 132;
```

integer (32 bits) [+2,147,483,647 to -2,147,483,648]

```
int framesReceived = 83000;
```

long (64 bits) [+9,223,372,036,854,775,807 to -9,223,372,036,854,775,808]

```
long numberOfDatabaseRecords = 2870L;
```

float (32 bits) [3.402,823,5 E+38 to 1.4 E-45]

```
float pi = 3.1415f;
```

double (64 bits) [1.797,693,134,862,315,7 E+308 to 4.9 E-324]

```
double areaOfCircle = 12.015897;
```

boolean (1 bit) [false, true]

```
boolean success = true;
```

Naming things

Name your classes, objects, attributes, methods, variables, programs, ... as you would name your children, with great care.

Never invent synonyms for things that are actually the same. This is considered bad practice and will not lead to good software.

Control flow

Execute code based on conditions using if-else statements.

```
if (<condition>) {
  // Code if condition is true
} else if (<other_condition>) {
  // Code if other_condition is true
} else {
  // If none of conditions above are true
}
```

A while loop is used when the number of iterations is not known beforehand. Based on the condition the while might not be executed at all.

```
while (<condition>) {
  // Code block
}
```

A do-while loop is used when the number of iterations is not known beforehand. Contrary to the while loop, the do-while is executed at least once.

```
do {
  // Code block
} while (<condition>);
```

A for-loop is used when the number of iterations is known beforehand.

```
for (<initialization> ; <condition> ;
<increment>) {
  // Code block
}
```

Comments

While comments can sometimes be useful they can also be dangerous. As implementation changes, comments do not always follow these changes.

If you take care of your code you should almost never need comments to explain what the code is doing.

Comments can however be useful to explain certain design decisions or thoughts behind an idea.

Comments can also be used to add TODO's to your code. For example code that should be refactored but you currently don't have the time for.

Class Basics

Defining a class

```
<access_modifier> class <name_of_class> {
  // IMPLEMENTATION
}
```

Defining constructors

```
<access_modifier> <name_of_class>
(<actual_arguments>) {
  // Initialize your attributes
}
```

Defining a method

```
<access_modifier> <return_type> <name>
(<actual_arguments>) {
  // Do some processing
  return <value>;
}
```

Return Types

- **Primitive-types:** byte, char, short, int, long, float, double, boolean
- **No return value:** void
- **Reference to object:** any class name, ex.: String

Calling a method

```
<object_instance>.<method_name>
(<formal_arguments>);
```

Access modifiers

- **private:** no access except inside class itself
- **protected:** no access except inside class and subclass
- **default (package):** ....
- **public:** access from everywhere

Composition

Class contains instance variable (attribute) of other class

Has-a relationship

Inheritance

Subclass inherits from superclass

```
<access_modifier> class <name_of_class>
extends <superclass> {
  // IMPLEMENTATION
}
```

Is-a relationship