Basics

Application Entry Point

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

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;
```

Terminal

Output to terminal

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

Class Basics

Defining a Class

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

Defining a method

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

Actual Argument List

```
(<argument_name|literal>,
<argument_name|literal>, ...)
```

Return Types

- **integer:** byte, char, short, int, long
- **floating-point:** float, double
- **logical:** boolean
- **no return value:** void
- **an object:** any class name, ex.: String

Calling a method

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

Formal Argument List

```
(<type> <argument_name>, <type>
<argument_name>, ...)
```

Access modifiers

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

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.

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.

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.