# Clean object calisthenics

Introduction

Clean object calisthenics is a clean code discipline invented by Damir Majer that can be considered as a compass gathering principles that are intended to improve the overall code quality.  
Each of the following principle will be presented with a definition, the references to other principles and an example in ABAP.

1. 4 rules of simple design

Presented by Kent Beck while he was developing XP in the 90s, the rules are the following one, ordered by priority and complementing each other:

1. Pass the tests (It works)

Writing tests (comprehensive tests) actually could lead us to better designs

Once the tests are passed, we are empowered to keep our code clean. We do this by refactoring the code

**The fact that we have these tests eliminates the fear that cleaning up the code will break it!**

1. Reveal intention (Easy to read and understand)

we need to be expressive

*Choose a good name that represents the things*

*Keep your functions and classes small.*It’s easier to name, write, and understand it.

*Challenge and commit yourself to write code so that it reads as documentation*

[gradually improving names](https://blog.jbrains.ca/permalink/the-four-elements-of-simple-design#references).

1. No duplication (DRY)

DRY everything should be said *once and only once*

single responsibility

Kent expressed it as saying everything should be said "Once and only Once."

Every piece of knowledge should have one and only one representation

Once and Only Once

Bad example :

DATA iv\_i TYPE i VALUE 1.

DO 10 TIMES.

IF is\_strike( iv\_i ).

rv\_score += 10 + strike\_bonus( iv\_i ).

iv\_i += 1.

ELSEIF is\_spare( iv\_i ).

rv\_score += 10 + spare\_bonus( iv\_i ).

iv\_i += 2.

ELSE.

rv\_score += sum\_of\_balls\_in\_frame( iv\_i ).

iv\_i += 2.

ENDIF.

Good example :

DATA iv\_frame\_index TYPE i VALUE 1.

DO 10 TIMES.

IF is\_strike( iv\_frame\_index ).

rv\_score += 10 + strike\_bonus( iv\_frame\_index ).

iv\_frame\_index += 1.

ELSEIF is\_spare( iv\_frame\_index ).

rv\_score += 10 + spare\_bonus( iv\_frame\_index ).

iv\_frame\_index += 2.

ELSE.

rv\_score += sum\_of\_balls\_in\_frame( iv\_frame\_index ).

iv\_frame\_index += 2.

ENDIF.

METHOD credit.

DATA(lv\_balance) = go\_wallet->get\_balance( ).

ADD iv\_credit TO lv\_balance.

go\_wallet->set\_balance( lv\_balance ).

ENDMETHOD.

CLASS lcl\_transaction IMPLEMENTATION.

METHOD constructor.

go\_wallet = io\_wallet.

ENDMETHOD.

METHOD credit.

go\_wallet->credit( iv\_credit ).

ENDMETHOD.

METHOD debit.

go\_wallet->debit(

EXPORTING

iv\_debit = iv\_debit

EXCEPTIONS

not\_enough\_funds = 4

).

IF sy-subrc <> 0.

WRITE 'Transaction cancelled because not enough funds'.

ENDIF.

ENDMETHOD.

ENDCLASS.

The order of 2nd and 3rd rule might be slightly different in some places, in Robert C. Martin’s book: [Clean Code book](http://www.amazon.com/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882), it’s *No duplication* first and *Reveals Intention* the later, while in [Martin Fowler article](https://www.martinfowler.com/bliki/BeckDesignRules.html), it’s Reveals Intention first and No duplication the later.

**Minimizes duplication** seems to trump **Maximizes clarity**, but when reading code, the reverse seems to be true.

1. Fewest Eléments

These questions can be asked to yourself when you take some time to have a step back after writing some code:

* **Do I have any dead code?**Sometimes as we working through our system, we build things that aren’t final, and there are times that in the end, we don’t need it at all in the final product. If it’s happened, no question asked, just delete it.
* **Have I extracted too far?**Sometimes we can do over-extract while trying to better express our intent, for example when extracting methods for readability but we do it too far to the extent that every method has its own class.

**No.3 - Living Objects**

A primitive type is a basic type that a programming language provides.

static methods,