|  |
| --- |
| Names  **The Golden Rules** |
| Use descriptive names |
| max\_wait\_time\_in\_seconds, ~~iso3166tab~~. |
| Language |
| Prefer object orientation over imperative programming |
| I.e. classes over functions and reports |
| Prefer functional over procedural language constructs |
| E.g. index += 1 instead ADD 1 to index |
| Comments |
| Express yourself in code, not in comments |
| Delete code instead of commenting it |
| Formatting |
| Be consistent |
| Optimize for reading, not for writing |
| Constants |
| Use constants instead of magic numbers |
| E.g. typekind\_date instead 'D' |
| Tables |
| Use the right table type |
| HASHED: large, filled at once, never modified, read often  SORTED: large, always sorted, filled over time or modified, read often  STANDARD: small, array-like |
| Booleans |
| Use XSDBOOL to set Boolean variables |
| empty = xsdbool( itab IS INITIAL ) |
| Conditions |
| Try to make conditions positive |
| IF has\_entries = abap\_true. |
| Consider decomposing complex conditions |
| DATA(example\_provided) = xsdbool(…)  IF example\_provided = abap\_true AND  one\_example\_fits = abap\_true. |
| Ifs |
| Keep the nesting depth low |
| ~~ELSE.~~  ~~IF <other>.~~  ~~ELSE.~~  ~~IF <something>.~~ |
| Regular expressions |
| Consider assembling complex regular expressions |
| CONSTANTS classes …  CONSTANTS interfaces …  … = |{ classes }|{ interfaces }|. |
| Classes: Object orientation |
| Prefer objects to static classes |
| Prefer composition over inheritance |
| DATA delegate TYPE REF TO  ~~CLASS a DEFINITION INHERITING FROM~~ |
| Don’t mix stateful and stateless in the same class |
| Classes: Scope |
| Members PRIVATE by default, PROTECTED only if needed |
| Testing: Principles |
| Write testable code |
| *There are no tricks to writing tests, there are only tricks to writing testable code. (Google)* |
| Enable others to mock you |
| CLASS my\_super\_object DEFINITION.  INTERFACES you\_can\_mock\_this. |
| Readability rules |
| given\_some\_data( ).  do\_the\_good\_thing( ).  and\_assert\_that\_it\_worked( ). |
| Test classes |
| Call local test classes by their purpose |
| CLASS unit\_tests  ~~CLASS tests\_for\_the\_class\_under\_test~~ |
| Code under test |
| Test interfaces, not classes |
| DATA cut TYPE REF TO some\_interface  ~~DATA cut TYPE REF TO some\_class~~ |
| Injection |
| Use test seams as temporary workaround |
| They are *not* a permanent solution! |
| Don’t misuse LOCAL FRIENDS to invade the tested code |
| ~~CLASS unit\_tests LOCAL FRIENDS cut.~~  ~~cut->db\_reader = stub\_db\_reader~~ |
| Test Methods |
| Test methods names: reflect what’s given and expected |
| METHODS accepts\_emtpy\_user\_input  ~~METHODS test\_1~~ |
| Use given-when-then |
| given\_some\_data( ).  do\_the\_good\_thing( ).  assert\_that\_it\_worked( ). |
| “When” is exactly one call |
| given\_some\_data( ).  do\_the\_good\_thing( ).  ~~and\_another\_good\_thing( ).~~  assert\_that\_it\_worked( ). |
| Assertions |
| Few, focused assertions |
| ~~assert\_not\_initial( itab ).~~  assert\_equals( act = itab exp = exp ). |
| Use the right assert type |
| assert\_equals( act = itab exp = exp ).  ~~assert\_true( itab = exp ).~~ |
| Assert content, not quantity |
| assert\_contains\_message( key )  ~~assert\_equals( act = lines( messages )~~  ~~exp = 3 ).~~ |
| Assert quality, not content |
| assert\_all\_lines\_shorter\_than( … ) |

Clean ABAP

|  |
| --- |
| Methods: Object orientation |
| Prefer instance to static methods |
| METHODS a  ~~CLASS-METHODS a~~ |
| Public instance methods should be part of an interface |
| INTERFACES the\_interface.  ~~METHODS a~~ |
| Methods: Method body |
| Do one thing, do it well, do it only |
| Descend one level of abstraction |
| ~~do\_something\_high\_level ( ).~~  ~~DATA(low\_level\_op) = |a { b }|.~~ |
| Keep methods small |
| 3-5 statements, o~~ne page, 1000 lines~~ |
| Methods: Parameter number |
| Aim for few IMPORTING parameters, at best less than three |
| ~~METHODS a IMPORTING b c d e~~ |
| Split methods instead of adding OPTIONAL parameters |
| METHODS a IMPORTING b  METHODS c IMPORTING d  ~~METHODS x~~  ~~IMPORTING b~~  ~~D~~ |
| RETURN, EXPORT, or CHANGE exactly one parameter |
| ~~METHODS do\_it~~  ~~EXPORTING a~~  ~~CHANGING b~~ |
| Error handling: Return codes |
| Prefer exceptions to return codes |
| METHODS check RAISING EXCEPTION  ~~METHODS check RETURNING result~~ |
| Don’t let failures slip through |
| DATA(result) = check( input )  IF result = abap\_false. |
| Error handling: Exceptions |
| Exceptions are for errors, not for regular cases |
| Use class-based exceptions |
| METHODS do\_it RAISING EXCEPTION  ~~METHODS do\_it EXCEPTIONS~~ |
| Error handling: Throwing |
| Throw one type of exception |
| ~~METHODS a RAISING EXCEPTION b c d~~ |
| Throw CX\_STATIC\_CHECK for manageable situations |
| RAISE EXCEPTION no\_customizing |
| Throw CX\_NO\_CHECK for usually unrecoverable situations |
| RAISE EXCEPTION db\_unavailable |
| Error handling: Catching |
| Wrap foreign exceptions instead of letting them invade your code |
| CATCH foreign INTO DATA(error).  RAISE EXCEPTION NEW my( error ).  ~~RAISE EXCEPTION error.~~ |