Riscure coding standard

0. **Do not use *goto***
   No exceptions.
1. **Methods should be easily readable**
   Maximum 30 lines of code and a cyclomatic complexity of at most 10, exempted are GUI code and parsers.
2. **Use braces around all statements**
   Including single statements, improves readability and prevents mistakes caused by indenting.
3. **Use default statement for conditional operation**
   An *else* for every *if* and a *default* for every *switch*, unless you can prove the disjunction of the guards is true. Avoid switch-case fall throughs, but comment any fall throughs.
4. **Do not use nested conditional tertiary (?:) operators**
5. **Use a single *return* statement**
   Improves readability and forces more readable structure.
6. **Do not use catch all**
   Catch Exception, Error, or Throwable can hide new exceptions being added, handle each exception or its parent individually and use RuntimeException to catch unchecked exceptions and at the very least log an informative error message.
7. **Write unit tests**
   Achieve at least 80% line coverage and 70% branch coverage.
8. **Do not use magic numbers**
   Declare (static final) constants for all numbers except 0 and 1. Exempted are nameless constants defined in crypto algorithm specifications, this exemption does **not** include key lengths, block sizes, etc. and their derivatives.
9. **Write JavaDoc for all members**
   Unless parent class/interface already documents member, in that case only document implementation specifics. Note that JavaDoc is API documentation and not usage documentation and that therefore in and of itself is not complete.
10. **Use interfaces**
    Interfaces provide a clean interface and can be easily stubbed to facilitate testing.
11. **Use parentheses when using mixed operators**
    ((a == b) && (c == d)) is more readable then *a == b && c == d.*
12. **Use e*num* for enumerations**
    Enums provide compiler checks as opposed to using integers.

13. **Extract variables for booleans in conditional expressions**
    if((s != null) || (s.length() < 4)) is less readable than if(hasCorrectLength). The latter is also more descriptive.
14. **Hide implementation details**
    Do not let public classes implement interfaces they do not primarily represent, use nested classes instead.
15. **Throw appropriate exceptions with informative messages**
    Throw specific exceptions for specific problems. Provide descriptive exception messages. Do not throw Error.
16. **Solve all compiler warnings**
    If a compiler warning needs suppression, comment the suppression.
17. **Do not use declarations that hide other declarations**
    Variables with the same name in a nested scope hide the higher level variable, excepted are constructors, getters, and setters.
18. **Sanitise input using the project's precondition library**
    If the input is supposed to confirm to some rules that cannot be solved at the compilation level, sanitise the input using the project's precondition library in order to make preconditions explicit.
19. **Do not check in commented out code or TODOs**
    Exempted are snapshot releases. Version control can be used to retrieve old code, issue tracking for reporting open issues. Source code may contain references to issues.
20. **Postfix variable names with units when applicable**
    A variable representing time in microseconds would be named *time_us* for example.
21. **Use explicit and specific import statements**
    Import only that which is required and nothing more. This also means no global imports.
22. **Use the Google Guava framework for collection operations**
    Prevents writing repetitive and error-prone collections code.

Riscure formatting standard

```java
package foo.bar.baz;

import java.util.List;
import java.net.Socket;

/**
 * JavaDoc
 */
@Deprecated
class Example<S, T extends Element & List> extends Base {
    /*
     * Variables and constants should be declared on top.
     */
    final static int SOME_CONSTANT = 1;
    int[] myArray = {1, 2, 3, 4, 5, 6};
    int theInt = 1;
    String someString = "Hello";
    double aDouble = 3.0;

    /* Constructors are next. */
    public Example() {}

    /* Followed by all other methods. */
    private void foo(int a, int b, int c, int d, int e, int f) {
        switch (a) {
            case 0:
                Other.doFoo();
                break;
            default:
                Other.doBaz();
        }
    }

    @Override
    public boolean bar(@SuppressWarnings("unused") List v) {
        boolean result;
        for (int i = 0; i < 10; i++) {
            v.add(new Integer(i));
        }
        if (v.isEmpty()) {
            result = true;
        } else {
            result = false;
        }
        return result;
    }
}
```