# adbc - Design by Contract for AspectJ

User manual - version 0.1

Adbc is a small and lightweight library that adds support for Design by Contract to the AspectJ programming language. The library essentially consists of a number of aspects that monitor your contracts at runtime and will throw an exception whenever a contract is broken.

## Requirements

Java 6 (or later) and AspectJ (tested on 1.6.12) are required.

## Installation

Include `adbc.jar` on the build path of your AspectJ project and contract enforcement should be enabled automatically. More specifically, if you are using Eclipse+AJDT, right-click your AspectJ project, go to "Properties", "AspectJ Build", "InPath" and click the "Add (External) JARs..." button. (See the Caveats section if an exception is thrown.)

Because the aspects within adbc can advise any method call and advice execution, you probably want to hide its advice markers. You can do this in AJDT by right-clicking any advice marker, then go to "AspectJ Tools", "Configure advice markers...".

If you want to tinker with adbc on a small toy project before enabling it on your own projects, just have a look at the included example in the `adbc/source/src/be/ac/ua/ansymo/example_bank` folder.

## Usage

Contracts are specified as Java annotations: You can specify preconditions (`@requires`), postconditions (`@ensures`) and invariants (`@invariant`). Pre -and postcondition annotations are specified at the level of methods and advice. Invariants are specified at the level of classes and aspects.

The contracts themselves are written in the form of Javascript expressions, as a String parameter of the annotation. If a contract consists of multiple parts, you can also pass an array of Strings. The following example of a simple `Square` class demonstrates the syntax of contracts:

```
@invariant("$this.getWidth()==$this.getHeight()")
class Square {
    ...

    @requires("s > 0"})
    @ensures({"$this.getHeight()==s", "$this.getWidth()==s"})
    public void setSize(int s) {...}

    @ensures("$result=="$this.getWidth()*$this.getWidth()")
    public int getArea() {...}

    @ensures({"$this.getX()==$old($this.getX())+x",
      ""$this.getY()==$old($this.getY())+y"")
    public void move(int x, int y) {...}

    ...
}
```

This example features all the different constructs that can be accessed from a contract:

`$this` The this object (You currently can't use the this object implicitly yet..)

`parameters` You can simply access method/advice parameters via their name

`$result` The return value of a method/advice, available in postconditions

`$old(expr)` The old function evaluates an expression before the method/advice is executed, stores the result, such that it is available in postconditions. This is useful if, for example, you want to compare the old value of a field with the new value.

With contract enforcement enabled, contracts are checked at runtime, as well as whether methods/advice are adhering to the Liskov/advice substitution principle. (For more information on Liskov substitution, just have a look at Wikipedia. For more information on advice substitution, which is essentially Liskov substitution for aspects, have a look at section 3 in the paper "Design by Contract for Aspects, by Aspects".) Whenever a contract is broken, a `ContractEnforcementException` is thrown, indicating which part of the contract was broken, and who is to blame.

## Configuration

To tweak the impact that adbc has on program performance, the library can be configured by modifying the static fields in the `AdbcConfig` class. You can alter these fields at any time to disable contract enforcement entirely, disable postcondition checking or disable substitution principle checking.

## Troubleshooting

- In case Eclipse throws an exception if you try to include `adbc.jar` to the AspectJ build path, you can get around this problem by simply putting the adbc source code into your project instead. This

seems due to an AJDT bug similar to #244300. Note that you may be able to include `adbc.jar` on the Aspect Path instead of the Inpath, but then you will only get contract enforcement on classes, not aspects.

- If parameter names are not available in contracts, try passing the "-g:var" command-line parameter to the compiler. (This should be enabled by default when using AJDT.) Otherwise, if parameter names cannot be retrieved, you can use "arg0", "arg1", .. instead.

## Caveats

- Keep in mind that adbc is currently still a proof of concept. This means some basic features are still missing: attaching invariants directly to fields, or inheriting contracts from the super class. The performance of adbc could also be improved with some caching.
- The advice substitution principle cannot be enforced yet on higher-order advice (advice that advises advice..), unless this advice accesses the non-static part of the `thisjoinpoint` object. Our contract enforcement advice needs access to that object, but it is created lazily by the higher-order advice, so it may or may not be available..
- Checking Liskov substitution currently assumes that overriding methods use the same parameter names as the overridden method. (This could be solved using the Paranamer library..)
- There is basic support for the `@advisedBy` annotation/clause, but several things can be improved:
  - An advice mentioned in an `@advisedBy` clause has to be mentioned by its absolute/canonical name. It would be nicer if you could use its simple name (+ an import statement).
  - If multiple advice are mentioned in an `@advisedBy` clause:
    - We do not enforce the ordering of the listed advice, but assume this is done by a separate declare precedence statement.
    - When resolving the `$proc` variable, we assume that the advice mentioned in the clause use the same parameter names as the join point they advise. (Should be possible to figure out the mapping from the advice's names to those used by the join point.. Could be done by examining the advice's pointcut, but I'd rather not re-invent parts of the AspectJ compiler..)
    - If an advice is mentioned in an `@advisedBy` clause and its pointcut makes use of constructs that can only be determined at runtime, like cflow or if, you'll currently need to copy them into a `@pointcutRuntimeTest` annotation attached to the advice. This is needed to determine the effective specification of methods that mention such an advice in their `@advisedBy` clause. The `@pointcutRuntimetest` annotation is technically redundant information, but it's tricky to fix this since there's currenly no such thing as a reflection API for pointcuts.. Another option would be to compile the effective specifications as a preprocessing step, as it can be done statically given the source code..
  - Even though advice can have names (using an `@AdviceName` annotation), AspectJ currently does not support overriding advice, so it's of course not possible either to make use of this feature in an `@advisedBy` clause..

## Contact

If you have any questions, suggestions or other feedback, feel free to contact me at tim.molderez@ua.ac.be.