Design
by
Contract™
Design by Contract

A discipline of analysis, design, implementation, management
Applications

*Getting the software right*
Analysis
Design
Implementation
Debugging
Testing
Management
Maintenance
Documentation
Every software element is intended to satisfy a certain goal, for the benefit of other software elements (and ultimately of human users).

This goal is the element’s contract.

The contract of any software element should be
- Explicit.
- Part of the software element itself.
The imperative and the applicative

<table>
<thead>
<tr>
<th><em>do</em></th>
<th><em>ensure</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>( balance := balance - sum )</td>
<td>( balance = \text{old balance} - sum )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PRESCRIPTIVE</th>
<th>DESCRIPTIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>How?</td>
<td>What?</td>
</tr>
<tr>
<td>Operational</td>
<td>Denotational</td>
</tr>
<tr>
<td>Implementation</td>
<td>Specification</td>
</tr>
<tr>
<td>Command</td>
<td>Query</td>
</tr>
<tr>
<td>Instruction</td>
<td>Expression</td>
</tr>
<tr>
<td>Imperative</td>
<td>Applicative</td>
</tr>
</tbody>
</table>
How not to do it

\[ r(i: \text{INTEGER}): \text{BOOLEAN} \text{ is} \]
\begin{align*}
\text{require} & \quad i \geq 0 \text{ or } i < 0 \\
\text{ensure} & \quad \text{Result} = \text{true or Result} = \text{false}
\end{align*}

\[ r(x: \text{BANK\_ACCOUNT}): \text{BOOLEAN} \text{ is} \]
\begin{align*}
\text{require} & \quad x \neq \text{Void and } x.\text{balance} > 0
\end{align*}
Exercise 2

Inheritance

Diagram:

```
+-------+          +-------+
|       |          |       |
|   *   |          |   +   |
| PRINTER|          | POSTSCRIPT_PRINTER |
|       |          |       |
| +      |          | +      |
| TEXT_PRINTER |          | POSTSCRIPT_PRINTER |
```

Inheritance
Is the solution correct?

Inheritance
Contracts and inheritance

Issues: what happens, under inheritance, to

- Class invariants?
- Routine preconditions and postconditions?
Invariant Inheritance rule:

- The invariant of a class automatically includes the invariant clauses from all its parents, “and”-ed.

Accumulated result visible in flat and interface forms.
Assertion redeclaration rule

When redeclaring a routine, we may only:

- Keep or weaken the precondition
- Keep or strengthen the postcondition
A simple language rule does the trick!

Redefined version may have nothing (assertions kept by default), or

\begin{align*}
\text{require else } & \text{new_pre} \\
\text{ensure then } & \text{new_post}
\end{align*}

Resulting assertions are:

- \text{original_precondition or new_pre}

- \text{original_postcondition and new_post}
Correct Solution

```
Correct Solution

is_printable: BOOLEAN
   do
      Result := f.is_text_file
   end

Inheritance

print (f: FILE)*
   require is_printable
   is_printable: BOOLEAN *

is_printable: BOOLEAN
   do
      Result := f.is_postscript_file
   end
```