Introduction to Programming
Bertrand Meyer

Lecture 10: Abstraction

Last revised 5 December 2005

Routine: algorithm abstraction

To abstract is to capture the essence behind the details and the specifics

Implies giving a name to the result

In programming:

- Data abstraction: class
- Algorithm (operational) abstraction: routine

A routine is one of the two kinds of feature...

... the other is attribute

We have encountered lots of routines already, without the name.

A routine

\[ r(\text{arg}: \text{TYPE}; ...) \]

- Header comment
  - require
    - \text{Precondition} (boolean expression) ...
  - do
    - \text{Body} (instructions) ...
  - ensure
    - \text{Postcondition} (boolean expression)

end

Uses of routines

Bottom-up: capture existing algorithm, possibly for reuse

Top-down: placeholder routines — attractive alternative to pseudocode.

Two kinds of routine

Procedure: doesn’t return a result
- \text{Yields a command}
- \text{Calls are instructions}

Function: returns a result

\[ f(\text{arg}: \text{TYPE}; ...) \text{RESULT_TYPE} \]

- \text{... (The rest as before) ...}
- \text{Yields a query}
- \text{Calls are expressions}
Features: the full story

A class is characterized by its features
Each feature is an operation on the corresponding objects: query or command
Features are grouped into categories for readability
Class clauses:
- Indexing
- Inheritance
- Creation
- Feature (any number)
- Invariant

Anatomy of a class:

Uniform access principle

It doesn’t matter to the client whether you look up or compute

A call such as

```
ibm_company. yearly_average
```

could use an attribute or a function

An object has an interface

An object has an implementation

Information hiding
Uniform Access Principle

Expressed more technically:

Features should be accessible to clients the same way whether implemented by storage or by computation.

Uniform Access: an example

balance = list_of_deposits.total - list_of_withdrawals.total

(A1)

(A2)

What clients may do

class METRO_STATION

feature

x, y: REAL -- Coordinates of metro station
size: REAL -- Size of bounding square
upper_left: POSITION -- Upper-left position of bounding square
adjust_position

-- Set positions of bounding square

do

upper_left := (x + size/2, y + size/2)

end

end

Use procedures:

upper_left := (3, upper_left, y)

upper_left := set_x(3)

upper_left := move(3, h)
### Possible client privileges

If class A has an attribute \( \texttt{att}: \texttt{SOME\_TYPE} \), what may a client class C with
\[ a : A \]
do with \( a \texttt{\_att} \)?

The attribute may be:
- Secret
- Read-only
- Read, restricted write
- Full write

Example: modify \( x \) with \texttt{move in POINT} procedure

### Abstraction and client privileges

If class A has an attribute \( \texttt{att}: \texttt{SOME\_TYPE} \), what may a client class C with
\[ a : A \]
do with \( a \texttt{\_att} \)?

Read access if attribute is exported
- \( a \texttt{\_att} \) is an expression.
- An assignment \( \texttt{a\_att} := v \) would be syntactically illegal!

(It would assign to an expression, like \( x := y \).)

### Applying abstraction principles

Beyond read access: full or restricted write, through exported procedures.

Full write privileges: \texttt{set\_attribute} procedure, e.g.

```plaintext
\texttt{set\_temperature(u: REAL)} is
  -- Set temperature value to \( u \)
require
  \texttt{not\_under\_minimum: u > -273}
  \texttt{not\_above\_maximum: u < 2000}
do
  \texttt{temperature := u}
  \texttt{update\_database}
ensure
  \texttt{temperature\_set: temperature = u}
end
```

Client will use e.g. \( x.\texttt{set\_temperature}(21.5) \).

### Other uses of a setter procedure

\[ \texttt{set\_temperature(u: REAL)} \]
  -- Set temperature value to \( u \)
require
  \texttt{not\_under\_minimum: u > -273}
  \texttt{not\_above\_maximum: u < 2000}
do
  \texttt{temperature := u}
  \texttt{update\_database}
ensure
  \texttt{temperature\_set: temperature = u}
end

### Having it both ways

Make it possible to call a setter procedure

```plaintext
\texttt{temperature: REAL assign set\_temperature}
```

Then the syntax

\[ x.\texttt{temperature} := 21.5 \]

is accepted as a shorthand for \( x.\texttt{set\_temperature}(21.5) \).

Retains contracts etc.
Information hiding

```plaintext
class A
  feature f...
  g...
  feature (NONE)
    h...
  feature (A, A)
    i, k, l...
  feature (A, B, C)
  m, n...
end
```

Status of calls in a client with `af`: A:

- `a`, `f`, `a.g` valid in any client
- `a.h` invalid everywhere (including in `A`'s own text)
- `a.i` valid only in `B`, `C` and their descendants
- `a.m` valid in `B`, `C` and their descendants, as well as in `A` and its descendants

Exporting selectively

```plaintext
class LINKABLE[G]
  feature (LINKED_LIST)
    put_right(...) is do ... end
    right: G is do ... end
    ...
end
```

An example of selective export

`LINKABLE` exports its features to `LINKED_LIST`.
Does not export them to the rest of the world.
Clients of `LINKED_LIST` don't need to know about `LINKABLE` cells.

Information hiding

Information hiding only applies to use by clients, using dot notation or infix notation, as with `a.f` (Qualified calls).

Unqualified calls (within class) not subject to information hiding:

```plaintext
class A
  feature (NONE)
    h is ... do ... end
  feature
    f is do ... end
end
```

End of lecture 10