Introduction to Programming

Language abstractions

Exercise Session 11
B.Herlig, (Marco Piccioni, Ilinca Ciupa)

5. November 2007

Exercise Session Week 7

Abstraction

- Features: Commands, Queries and Routines
- Uniform Access Principle
- Revisiting Information Hiding
An abstraction

René Magritte, “La trahison des images”, 1929

This is not a car
Another abstraction

This is not a car either

What is a software system?

A software system is not a model of reality; it is at best a model of a model of some part of some reality

[B. Meyer, Object Oriented Software Construction 2nd ed.]
Example

A banking information system is not a model of a bank

It is a software transposition of a model devised by a certain bank to describe which some managers think are relevant aspects of the bank's business

What really matters...

...is not how close you are to the “real world”, but how good your abstractions are
Quiz 1: Find the right abstractions

... (from a meeting with a customer)...
“The elevator should automatically open his doors when it stops at a certain floor”

ELEVATOR is a reasonable abstraction, that we may represent with a class
door, floor... maybe, or maybe not

Discussion

If the semantic of a door is merely that of being opened and closed, a BOOLEAN attribute is_open in class ELEVATOR will suffice

Otherwise if there is some particular behaviour associated to the elevator door (for example some voice message announcing the floor number) you may want DOOR to be a custom class
Quiz 2: Implication of your abstraction

Can you point out which could be the problems in representing a floor as an INTEGER type?

You probably have to restrict the range of possible integers to the number of floors you have in your building.

A custom class FLOOR can help you with that, encapsulating the restrictions in a feature set_floor that may check that the argument passed is a valid floor.

Features: the full story

- **Client view** (specification)
  - Command
  - No result
  - Feature
  - Returns result
  - Query

- **Internal view** (implementation)
  - Procedure
  - Routine
  - Computation
  - Memory
  - Feature
  - Function
  - Attribute
A routine

\[ r(\text{argument1:TYPE1; argument2:TYPE2;...}):\text{TYPE\_RETURNED} \]

-- Header comment

\textbf{require}

--your precondition here (boolean expression)

\textbf{local}

--your local variables here

\textbf{do}

--your routine body here (compound)

\textbf{ensure}

--your postcondition here (boolean expression)

end

Two kinds of routine

Procedure: doesn't return a result

- Yields a \texttt{command}
- Calls are \texttt{instructions}

Function: returns a result

- Yields a \texttt{query}
- Calls are \texttt{expressions}
Quiz 3: using the right names

class CAR

feature

  wheels: ARRAY[WHEEL]  ------> attribute (query)
  engine: ENGINE        ------> attribute (query)

feature

  check_wheel(a_wheel: INTEGER) --> routine, procedure (command)
  do
    ...
  end

  fuel_left: REAL         ------> routine, function (query)
  do
    ...
  end

Quiz 4

A client of class PERSON wants to be able to query about the person's age:
What do you choose for implementation? An attribute or a function?

Actually you can choose both:
  • An attribute means updating it every time age changes,
  • a function means calculating age on demand, from an attribute of type DATE representing birthdate.
Uniform Access Principle

Should a client of a class really care about knowing “how” the provider does his job?

No, as far as he delivers what promised

As a consequence, a call like

   my_friend.age

should always be possible, independently of which implementation you chose for feature `age`

Uniform Access Principle

Expressed more technically:

Features should be accessible to clients the same way whether implemented by **storage** or by **computation**
Possible client privileges

If class $A$ has an attribute $att : SOME_TYPE$, what may a client class $C$ with $a : A$ do with $a.att$?

The attribute may be:

- Secret
- Read-only
- Read, restricted write
- Full write

- $a.att$ permitted in $C$ (for access)
- $a.att$ invalid

Information hiding

Status of calls in a client with $a1: A$:

- $a1.f, a1.g$: valid in any client
- $a1.h$: invalid everywhere (including in $A$'s own text!)
- $a1.j$: valid only in $B, C$ and their descendants (not valid in $A$!)
- $a1.m$: valid in $B, C$ and their descendants, as well as in $A$ and its descendants
Quiz 5

class CAR
create make
feature {NONE}
  make
  do
    --some code here
  end
  ...
end

class RACE
feature
  organize
  do
    create a_car.make
    ...
  end
  a_car: CAR
end

Danger?

That looks like a violation of information hiding!

But in effect it is not, because it holds only for creation procedures, that are part of the interface of the class

As you can use a procedure listed as a creation procedure like a normal procedure, you still cannot do things like

  a_car.make

if make is not available
Tomorrow

In tomorrow’s exercise session:

• Personal classroom exercise feedback
• Parallel: work on assignment 6
  -> please bring your laptops!

End Exercise Session Week 7