

# 1 Introduction

## 1.1 Basic definitions

- Feature
  - Command: No return value, but does modify objects. On the syntactical level, it is an instruction
  - Query: Returns a value, but does not modify any objects. The syntax equivalent is the expression.
    - \* Functions get their results through computation
    - \* Attributes are values directly stored in memory
  - For queries, there is the uniform access principle, which states that it doesn't matter to the client whether a query is implemented as a function or attribute. Features should be accessible to clients the same way whether implemented by storage or by computation.
  - Creation Procedure: Commands to initiate objects, can be several. There is also a *default\_create*, which is inherited by all classes, and does nothing by default.
- Feature Calls
  - Unqualified calls: Feature calls which apply to the current object
  - Qualified calls: Feature calls which apply to a certain object, causing this object to become the current object.
- Class clauses
  - Indexing
  - Inheritance
  - Creation
  - Feature
  - Invariant
- Specimen: A syntactic element, such as a class name or an instruction, but no delimiters. The type of a specimen is its construct. See Describing syntax
- Abstract syntax tree: Shows the syntax structure with all its specimens, but obviously without any delimiters, A tree has nodes, each one of the following kind:
  - Root: Node with no incoming branch.
  - Leaf: Node without outgoing branches
  - Internal node: Neither of the former
- Basic elements of a program text:
  - Terminals
    - \* Identifiers: Names chosen by the programmer

Add reference

- \* Keywords
  - \* Special symbols, such as a period
- Describing a program
  - Semantic rules: Define the effect of programming, satisfying the syntax rules
  - Syntax rules: Define how to make up specimens out of tokens satisfying the lexical rules
  - Lexical rules: Define how to make up tokens out of characters
- Syntax: The way you write a program; characters grouped into words, grouped into bigger structures.
- Semantics: The effect you expect from this program at runtime
- Identifier: Name chosen by the programmer to represent certain program elements, such as classes, features or runtime values. If it denotes a runtime value, it is called an identity or variable if it can change its value. During execution, an entity may become attached to an object.
- Executing a system consists of creating a root object, which in an instance of a designated class from the system, the root class, using a designated creation procedure, called its root procedure.

## 1.2 Variables

- Types
  - Reference types: Entities with a reference value
  - Expanded types: Entities with an object as a value
  - A type is one of:
    - \* A non-generic class
    - \* A generic derivation, i.e. the name of a class followed by a list of types, the actual generic parameters, in brackets
- Setters: It is possible to make assignments such as  $x.att := val$ , which is shorthand for  $x.set\_att(val)$
- Effect of an assignment
  - Reference types: Reference assignment
  - Expanded types: Value copy
- Variable copy
  - Shallow object duplication (creates a new object):  $b := a.twin$
  - Deep object duplication (creates a new object):  $b := a.deep\_twin$
  - Shallow field-by-field copy (does not create an object):  $b.copy(a)$

Maybe add the objet creation??

### 1.3 Interface

- A client of a software mechanism is a system of any kind - such as a software element or a human - that uses it. For its client, the mechanism is a supplier
- Interface: The description of techniques enabling clients to use these mechanisms. For example: GUIs (Graphical User Interface), command line interfaces (shell, bash,...), or APIs
- An object can be an instance of a class, if the class is the generating class of the object

### 1.4 Information Hiding

- For its clients, an attribute may be:
  - Secret
  - Read-only
  - Read, but partially write restricted (only certain things are allowed to be written)
  - Writing one or more classes in curly brackets after the keyword *feature* exports these features only to these classes and its descendants. If no class is listed, the features are exported to *ANY*.

Information hiding only applies to use by clients using dot or infix notation. Unqualified calls are not subject to information hiding.

### 1.5 Control structures

Requires a lot of work

- Sequence or compound
- Loop
  - Loop invariant
    - \* Satisfied **after** initialization, after the *from* clause
    - \* Preserved by every loop iteration executed with the exit condition not satisfied. So in the end, the loop invariant and the exit condition hold!
  - Loop variant
    - \* Non-negative (i.e.  $\geq 0$ ) integer expression, right after initialization
    - \* Decreases while remaining non-negative for every iteration of the body with exit condition not satisfied.
- Conditional

## 1.6 Contracts

- Contracts are made of assertions, each containing an assertion tag and a condition (a Boolean expression)
- Precondition
  - Property that a feature imposes on every client
  - If there is no *require* clause, is treated as one, with one only being true.
- Postcondition
  - Property that a feature guarantees every client
  - Can make use of keyword *old*
- Class invariant
  - The invariant expresses consistency requirements between queries of a class

Check for an explanation of this keyword

## 1.7 Miscellaneous

- Semistrict operators
  - Let us define the order of expression evaluation
  - **and then** is the semistrict version of **and**. Use it if a condition only makes sense when another is true.
  - **or else** is the semistrict version of **or**. Use it if a condition only makes sense when another is false
  - **implies** is always semstrict!

## 2 Describing syntax