Deel 2: Nominaal Programmeren

Het principe van *nominaal programmeren* met je eigen woorden kunnen uitleggen.

Nominal methods only guarantee their effect under stated conditions. For that purpose, specifications of nominal methods include so-called *preconditions* imposing restrictions under which clients can invoke them.

Het concept class invariant en de semantiek ervan kunnen uitleggen.

A class *invariant* is a condition that must hold at all steady times.

> Example
> The specification of the class of bank accounts will include a class invariant stating that the balance of each bank account may not go beyond the credit limit.

Class invariants that apply to class properties have semantics that is slightly different from the semantics of class invariants that apply to object properties. Broadly speaking, invariants that apply to class properties must be satisfied at all times. Invariants that apply to object properties must only be satisfied at steady times.

Class invariants that apply to *class properties* must be satisfied at all times. More in particular, such invariants must be satisfied from the very moment their class becomes involved in the application. Each time we inspect a class property, we may thus assume that its value satisfies all restrictions imposed on it.

Class invariants that apply object properties do not need to be satisfied by all objects of their class at all times. We allow temporary violations of invariants imposed on object properties for two reasons.

First, the initialization of a new object with several properties involves successive steps. Upon entry to a constructor, the newly created object typically does not satisfy all of the class invariants imposed on its properties. Only when we move towards the end of the constructor, and we have initialized most of the properties of the newly created object, that object will satisfy all its invariants.

Secondly, changing the state of some objects can lead to intermediate stages in which some of them temporaritly violate their class invariants. This can be the case if a single method must change the state of several objects at the same time.

Het concept class preconditions en de semantiek ervan kunnen uitleggen.

Formal argument types impose restrictions on the clients of a class. Types are often not powerful enough to express all the restrictions that apply to the arguments involved in a method. That's why preconditions become handy; they impose additional restrictions on arguments involved in a method. It offers an instrument to inform clients of class to not invoke methods under exceptional circumstances. So we deal with illegal values with the help from preconditions.

Preconditions must hold upon entry to their method. Preconditions imply duties for the clients of class. We can also substitute invocations of mutators and constructors in effect clauses by their preconditions, so we can obtain preconditions imposed on the specified method.

De specificatie van post-condities, effect-clauses en return kunnen uitleggen.

Triviaal.