Template for definitions of classes in Object-Z:

ClassName[*generic parameters*]

visibility list
inherited classes
type definitions
constant definitions
state schema
initial state schema
operation schemas

---

history invariant

The state schema does not have any name, and it is implicitly (automatically) included in the initial state schema and in every operation associated with the class. In the case of operations, also the primed state schema (variables with apostrophe) is included automatically.

Examples of two class definitions follow:

Vehicle

$\upharpoonright(curspeed, Accelerate)$

$manufacturer : Company$

$maxspeed : INT$

$maxspeed > 0 \land maxspeed < 500$

$curspeed : \mathbb{R}$

$curspeed < maxspeed$

Init

$curspeed = 0$

Accelerate

$\Delta(curspeed)$
$dspeed? : \mathbb{R}$

$curspeed' = curspeed + dspeed?$

OpenDoor

$curspeed = 0$

The class *Vehicle* has two constants (*manufacturer* and *maxspeed*), one state variable (*curspeed*), an initial state, and the operation *Accelerate*. Its visibility list (at the beginning) says that just the variable *curspeed* and the operation *Accelerate* represent the public interface of *Vehicle*. The Δ-list must enumerate all state variables that the operation updates. We defined just the precondition ($curspeed = 0$) for the operation *OpenDoor* because it does not change any state variable.

Car

Vehicle

$fuel : \mathbb{R}$
$engine : Engine$
$lights : Lights$

Init

$curspeed = 0$
$fuel = 50$

...

To declare inheritance, we just have to write the name of a superclass (*Vehicle*) at the beginning of the schema for our subclass (*Car*). Note also that the schema for *Car* introduces two new components: *engine* and *lights*.

ElectricCar

$Car[batterylevel/fuel]$
...

We can also rename some of the state variables. For example, here we use *batterylevel* instead of *fuel*.

Standard dot-notation is used (1) to access fields of state variables inside schemas, and (2) to execute operations on the objects pointed to by state variables, like in the mainstream object-oriented programming languages (Java, C#, C++).

Object-Z supports three ways of defining operations. The first option is to create a normal schema, the second option is delegation to an operation called on a particular state variable (component), and the last is through composition of multiple schemas. Examples of all the options follow.

Car

...

Init

$curspeed = 0$
$lights.mode = dim$

Accelerate

$\Delta(curspeed)$
$dspeed? : \mathbb{R}$

$curspeed' = curspeed + dspeed?$
$engine.IncreaseRevs$

$TurnOff \mathrel{\widehat=} engine.Stop$

$TurnOn \mathrel{\widehat=} engine.Start \land lights.TurnOn$

Specification of the body (effects) of some operation associated with a class may involve calls of operations upon components (variables) of its state. These calls should be typically defined in the constraint part of the enclosing schema. For example, we can add the call of *engine.IncreaseRevs* to the operation *Accelerate*. We can also redefine (override) the specifications for operations in subclasses, like we did for *Accelerate* here.