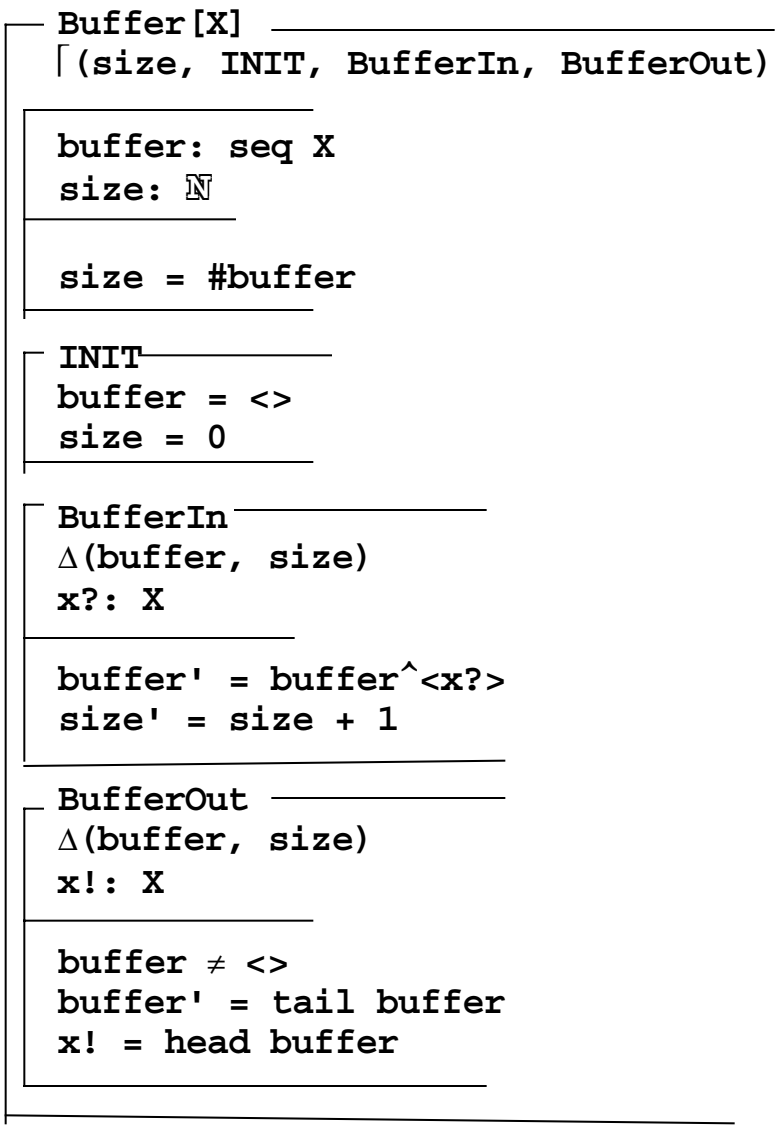
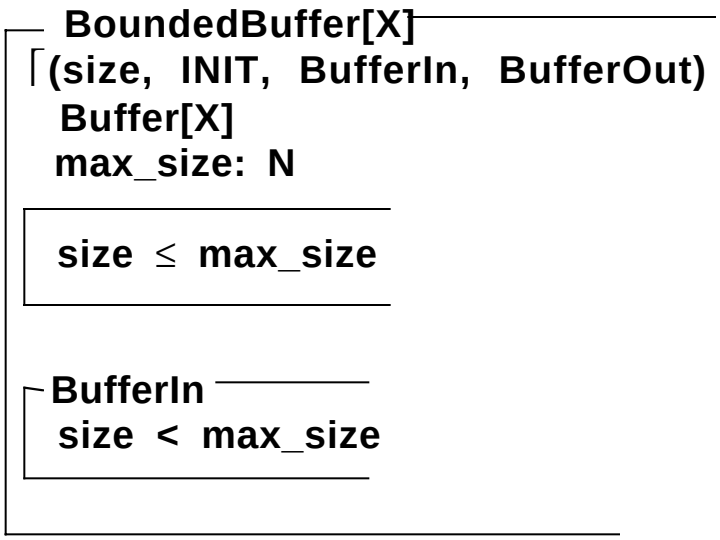
The Buffer class in Object-Z resembles the Z specification.



Notice this this version of the buffer is not bounded — items can be added without limit. As the object-oriented approach suggests, we start with the most general form of structure and specialize to obtain classes of immediate interest. Now we can specialize to the bounded buffer.

One class inherits another by importing it. The variables, constants, state schema, initial schema, and operations are inherited, but the visibility list is not. We now describe the bounded buffer by inheriting the buffer and making incremental changes.



Object History & History Invariants

Since an operation may change the state of an object, re-applying the same operation to the same object may not produce the same result. However, if the entire history of prior operation applications (and arguments) is considered, then the outcome is uniquely determined.

If an object's initial state is `init` **and its history of operation applications is (... ((**`init`**.op$_1$).op$_2$) ...).op$_n$, then its state and consequent behavior is known at every point.**

Some history-sensitive properties can be more readily expressed from this global perspective, without making explicit the internal state at every point. The is the role of (optional) *history invariant*.

The construction of useful history invariants is facilitated by a more expressive logic — *temporal logic* that expresses time dependencies. Object-Z uses the following temporal logic operators:

- **¤ P — *always* P, P holds at every stage in the history**
- **◊ P — *eventually* P, eventually there is a stage at which P holds**
- **¡ P — *next* P, P holds at the next stage**

For example, in our recent Buffer example, it might be reasonable for the specification to require that any item placed in a buffer should eventually be removed. This could be handily expressed by

¤(◊(#(op=BufferOut) = #(op=BufferIn)))