viewed/modified inside worlds), and `WWorld`, which represents worlds. The methods of `WObject` and its subclasses are automatically instrumented in order to indirect all instance variable accesses (reads and writes) through the world in which they are being evaluated.

A `WWorld` w contains a hash table that, similar to a *transaction log*, associates `WObject`s with

- a *reads* object, which holds the "old" values of each slot of the `WObject` when it was first read in w, or the special `Don't Know` value for each slot that was never read in w, and
- a *writes* object, which holds the most recent values of each slot of the `WObject`, or the special `Don't Know` value for slots that were never written to in w.

The keys of this hash table are referenced weakly to ensure that the *reads* and *writes* objects associated with a `WObject` that is no longer referenced by the program will be garbage collected. Also, *reads* and *writes* objects are instantiated lazily, so (for example) an object that has been read but not written to in a world will have a *reads* object, but not a *writes* object, in that world.

5.2 The Slot Update Operation: $(x_i \leftarrow v)_w$

To store the value v in x's i^{th} slot in world w,

1. If w does not already have a *writes* object for x, create one.
2. Write v into the i^{th} slot of the *writes* object.

5.3 The Slot Lookup Operation: $(x_i)_w$

To retrieve the value stored in x's i^{th} slot in world w,

1. Let $w_{current} = w$ and $ans = undefined$.
2. If $w_{current}$ has a *writes* object for x and the value stored in the i^{th} slot of the *writes* object is not `Don't Know`, set *ans* to that value and go to step 5.
3. If $w_{current}$ has a *reads* object for x and the value stored in the i^{th} slot of the *reads* object is not `Don't Know`, set *ans* to that value and go to step 5. (This step ensures the "no surprises" property, i.e., that a slot value does not appear to change spontaneously in w when it is updated in one of w's ancestors.)
4. Otherwise, set $w_{current}$ to $w_{current}$'s parent, and go to step 2.
5. If $w_{current} = w$, skip to step 8.
6. If w does not already have a *reads* object for x, create one.
7. If the value stored in the i^{th} slot of the *reads* object is `Don't Know`, write *ans* into that slot.
8. Return *ans*.

Note that the slots of a new `WObject` are always initialized with `nil`s in the top-level world. This mirrors the semantics of object instantiation in Smalltalk and ensures that lookup always terminates.

(We initially implemented the slot lookup operation in Smalltalk, but later re-implemented it as a primitive, which resulted in a significant performance improvement. See Section 6.3 for details.)