Object Destruction

Now that clients can create `PhoneEntry` objects, let us consider how to allow clients to destroy them again. One obvious design is to add a `destroy` operation to the factory — after all, seeing that a factory knows how to create objects, it stands to reason that it also knows how to destroy them again:

Slice

```
exception PhoneEntryNotExists {
    string name;
    string phNum;
};

interface PhoneEntryFactory {
    PhoneEntry* create(string name, string phNum)
        throws PhoneEntryExists;
    void destroy(PhoneEntry* pe)      // Bad idea!
        throws PhoneEntryNotExists;
};
```

While this works (and certainly can be implemented without problems), it is generally a bad idea. For one, an immediate problem we need to deal with is what should happen if a client passes a proxy to an already-destroyed object to `destroy`. We could raise an `ObjectNotExistException` to indicate this, but that is not a good idea because it makes it ambiguous as to which object does not exist: the factory, or the entry. (By convention, if a client receives an `ObjectNotExistException` for an invocation, what does not exist is the object the operation was targeted at, not some other object that in turn might be contacted by the operation.) This forces us to add a separate `PhoneEntryNotExists` exception to deal with the error condition, which makes the interface a little more complex.

A second and more serious problem with this design is that, in order to destroy an entry, the client must not only know which entry to destroy, but must also know *which factory created the entry*. For our example, with only a single factory, this is not a real concern. However, for more complex systems with dozens of factories (possibly in multiple server processes), it rapidly becomes a problem: for each object, the application code somehow has to keep track of which factory created what object; if any part of the code ever loses track of where an object originally came from, it can no longer destroy that object.

Of course, we could mitigate the problem by adding an operation to the `PhoneEntry` interface that returns a proxy to its factory. That way, clients could ask each object to provide the factory that created the object. However, that needlessly complicates the Slice definitions and really is just a band-aid on a fundamentally flawed design. A much better choice is to add the `destroy` operation to the `PhoneEntry` interface instead:

Slice

```
interface PhoneEntry {
    idempotent string name();
    idempotent string getNumber();
    idempotent void setNumber(string phNum);
    void destroy();
};
```

With this approach, there is no need for clients to somehow keep track of which factory created what object. Instead, given a proxy to a `PhoneEntry` object, a client simply invokes the `destroy` operation on the object and the `PhoneEntry` obligingly commits suicide. Note that we also no longer need a separate exception to indicate the "object does not exist" condition because we can raise `ObjectNotExistException` instead — the exception exists precisely to indicate this condition and, because `destroy` is now an operation on the phone entry itself, there is no ambiguity about which object it is that does not exist.

Topics

See Also

- Object Creation