Idempotency and Life Cycle Operations

You may be tempted to write the life cycle operations as follows:

Slice

```
interface PhoneEntry {
    // ...
    idempotent void destroy(); // Wrong!

};

interface PhoneEntryFactory {
    idempotent PhoneEntry* create(string name, string phNum)
        throws PhoneEntryExists;
};
```

The idea is that `create` and `destroy` can be idempotent operations because it is safe to let the Ice run time retry the operation in the event of a temporary network failure. However, this assumption is not true. To see why, consider the following scenario:

1. A client invokes `destroy` on a phone entry.
2. The Ice run time sends the request to the server on the wire.
3. The connection goes down just after the request was sent, but before the reply for the request arrives in the client. It so happens that the request was received by the server and acted upon, and the reply from the server back to the client is lost because the connection is still down.
4. The Ice run time tries to read the reply for the request and realizes that the connection has gone down. Because the operation is marked idempotent, the run time attempts an automatic retry by re-establishing the connection and sending the request a second time, which happens to work.
5. The server receives the request to destroy the entry but, because the entry is destroyed already, the server returns an `ObjectNotExistException` to the client, which the Ice run time passes to the application code.
6. The application receives an `ObjectNotExistException` and falsely concludes that it tried to destroy a non-existent object when, in fact, the object did exist and was destroyed as intended.

A similar scenario can be constructed for `create`: in that case, the application will receive a `PhoneEntryExists` exception when, in fact, the entry did not exist and was created successfully.

These scenarios illustrate that `create` and `destroy` are *never* idempotent: sending one `create` or `destroy` invocation for a particular object is not the same as sending two invocations: the outcome depends on whether the first invocation succeeded or not, so `create` and `destroy` are not idempotent.

See Also

- Idempotent Operations
- Automatic Retries