Hotel mixin sample -- queue mixin mixin

In our toy-world of the *Mixin Hotel*, it happens that all rooms are not only occupied, but also often reserved in advance; interested parties and their reservations must be sent away. However, often reservations are canceled, so it is a good idea to put people who can't reserve a room on a waiting list and call them back when a matching reservation is canceled.

This slice is a supplement to the reservation slice (see above).

The class `QueueMixinMixin` provides the logic for the queue slice.

- It intercepts the `Reserve` method of the `ReservationMixin`. If the target class' `Reserve` throws the `NoRoomsException`, the exception is caught, and the reservation is entered into the `Repository<QueuedReservationInfo>`.
- It intercepts the `Cancel` method of the `ReservationMixin`. After execution of the original `Cancel`, the interceptor looks for a matching reservation in the queue (repository). If it finds one, the operator is notified; if the reserving party from the queue confirms the confirmation, the reservation is moved to the reservation repository.

In code:

```
    [OverrideTarget]
    public virtual IReservationInfo Reserve (string name, int week)
    {
      try
      {
        var reservationInfo = Base.Reserve (name, week);
        return reservationInfo;
      }
      catch (NoRoomException)
      {
        _queueInfoRepository.Add (new QueueInfo (name, week));
        Console.WriteLine ("No more rooms. Name {0} added to queue.",
name);
        return null;
      }
    }

    [OverrideTarget]
    public virtual IReservationInfo Cancel (string name, int week)
    {
      var reservationInfo = Base.Cancel (name, week);
      var queueInfo = _queueInfoRepository.Find
().FirstOrDefault<IQueueInfo> (resInfo => resInfo.Week == week);
      if (queueInfo != null)
      {
        var reply = _yesNoHandler.YesNo (string.Format ("please notify {0}
for a room for week {1}. Does he/she accept?", queueInfo.Name,
queueInfo.Week));
        if (reply)
        {
          _queueInfoRepository.Remove (queueInfo);
          Base.Reserve (queueInfo.Name, queueInfo.Week);
        }
      }
      return reservationInfo;
    }
```

Two extra methods exist:

- the trade-mark `CountQueuedReservations` for testing
- `SetYesNoHandler` for dependency injection (also testing)

A *yes/no-handler* returns `true` ("yes") or `false` ("no"). It is intended to be a user-interface element for returning whether the queued party on the phone replied "yes" or "no" to the concierge's question whether the party wants to be unqueued and enforce its reservation.

For automatic testing we want to plug in one of two fakes:

- an affirmative yes/no-handler (always returns `true`)
- a declining yes/no-handler (always returns `false`)

These yes/no-handlers are implemented in the `Hotel.Tests` project.

(For completeness, here is the implementation for such a yes/no-handler, for a TTY user interface.)