```
 1  package Class::Observable;
 2
 3  # $Id: Observable-1.pm,v 1.1 2006/04/11 20:30:29 mjd Exp mjd $
 4
 5  use strict;
 6  use Class::ISA;
 7  use Scalar::Util qw( weaken );
 8
 9  $Class::Observable::VERSION = '1.04';
10
11  my %O = ();
12  my %P = ();
13
14
15  # Add one or more observers (class name, object or subroutine) to an
16  # observable thingy (class or object). Return new number of observers.
17
18  sub add_observer {
19      my ( $item, @observers ) = @_;
20      $O{ $item } ||= [];
21      foreach my $observer ( @observers ) {
22          $item->observer_log( "Adding observer '$observer' to ",
23                               "'", _describe_item( $item ), "'" );
24          my $num_items = scalar @{ $O{ $item } };
25          $O{ $item }->[ $num_items ] = $observer;
26          if ( ref( $observer ) ) {
27              weaken( $O{ $item }->[ $num_items ] );
28          }
29      }
30      return scalar @{ $O{ $item } };
31  }
32
33
34  # Remove one or more observers from an observable thingy. Return new
35  # number of observers.
36  # TODO: Will this work with subroutines?
37
38  sub delete_observer {
39      my ( $item, @observers_to_remove ) = @_;
40      unless ( ref $O{ $item } eq 'ARRAY' ) {
41          return 0;
42      }
43      my %ok_observers = map { $_ => 1 } @{ $O{ $item } };
44      foreach my $observer_to_remove ( @observers_to_remove ) {
45          $item->observer_log( "Removing observer '$observer_to_remove' f
rom ",
46                               "'", _describe_item( $item ), "'" );
47          my $removed = delete $ok_observers{ $observer_to_remove };
48          if ( $removed ) {
49              $item->observer_log( "Found observer '$observer_to_remove';
 ",
50                                   "removing..." );
51          }
52      }
53      $O{ $item } = [ keys %ok_observers ];
54      return scalar keys %ok_observers;
55  }
56
57
58  # Remove all observers from an observable thingy. Return number of
59  # observers removed.
60
61  sub delete_all_observers {
62      my ( $item ) = @_;
63      $item->observer_log( "Removing all observers from ",
64                           "'", _describe_item( $item ), "'" );
```

```
 65      my $num_removed = 0;
 66      return $num_removed unless ( ref $O{ $item } eq 'ARRAY' );
 67      $num_removed = scalar @{ $O{ $item } };
 68      $O{ $item } = [];
 69      return $num_removed;
 70  }
 71
 72
 73  # Backward compatibility
 74
 75  sub delete_observers {
 76      goto \&delete_all_observers;
 77  }
 78
 79
 80  # Tell all observers that a state-change has occurred. No return
 81  # value.
 82
 83  sub notify_observers {
 84      my ( $item, $action, @params ) = @_;
 85      $action ||= '';
 86      $item->observer_log( "Notification from '", _describe_item( $item )
, "'",
 87                           "with '$action'" );
 88      my @observers = $item->get_observers;
 89      foreach my $o ( @observers ) {
 90          $item->observer_log( "Notifying observer '$o'" );
 91          eval {
 92              if ( ref $o eq 'CODE' ) {
 93                  $o->( $item, $action, @params );
 94              }
 95              else {
 96                  $o->update( $item, $action, @params );
 97              }
 98          };
 99          if ( $@ ) {
100              $item->observer_error(
101                  "Failed to send observation from '$item' to '$o': $@" )
;
102          }
103      }
104  }
105
106
107  # Retrieve *all* observers for a particular thingy. (See docs for what
108  # *all* means.) Returns a list of observers
109
110  sub get_observers {
111      my ( $item ) = @_;
112      $item->observer_log( "Retrieving observers using ",
113                           "'", _describe_item( $item ), "'" );
114      my @observers = ();
115      my $class = ref $item;
116      if ( $class ) {
117          $item->observer_log( "Retrieving object-specific observers from
 ",
118                               "'", _describe_item( $item ), "'" );
119          push @observers, $item->_obs_get_observers_scoped;
120      }
121      else {
122          $class = $item;
123      }
124      $item->observer_log( "Retrieving class-specific observers from '$cl
ass' ",
125                           "and its parents" );
126      push @observers, $class->_obs_get_observers_scoped,
```

```
       127                         $class->_obs_get_parent_observers;
       128      $item->observer_log( "Found observers '", join( "', '", @observers
), "'" );
       129      return @observers;
       130  }
       131
       132
       133  # Copy all observers from one item to another. This also copies
       134  # observers from parents.
       135
       136  sub copy_observers {
       137      my ( $item_from, $item_to ) = @_;
       138      my @from_observers = $item_from->get_observers;
       139      foreach my $observer ( @from_observers ) {
       140          $item_to->add_observer( $observer );
       141      }
       142      return scalar @from_observers;
       143  }
       144
       145
       146  sub count_observers {
       147      my ( $item ) = @_;
       148      $item->observer_log( "Counting observers using ",
       149                           "'", _describe_item( $item ), "'" );
       150      my @observers = $item->get_observers;
       151      return scalar @observers;
       152  }
       153
       154
       155  # Find observers from parents
       156
       157  sub _obs_get_parent_observers {
       158      my ( $item ) = @_;
       159      my $class = ref $item || $item;
       160
       161      # We only find the parents the first time, so if you muck with
       162      # @ISA you'll get unexpected behavior...
       163
       164      unless ( ref $P{ $class } eq 'ARRAY' ) {
       165          my @parent_path = Class::ISA::super_path( $class );
       166          $item->observer_log( "Finding observers from parent classes ",
       167                               "'", join( "', '", @parent_path ), "'" );
       168          my @observable_parents = ();
       169          foreach my $parent ( @parent_path ) {
       170              next if ( $parent eq 'Class::Observable' );
       171              if ( $parent->isa( 'Class::Observable' ) ) {
       172                  push @observable_parents, $parent;
       173              }
       174          }
       175          $P{ $class } = \@observable_parents;
       176          $item->observer_log( "Found observable parents for '$class': ",
       177                               "'", join( "', '", @observable_parents ),
"'" );
       178      }
       179
       180      my @parent_observers = ();
       181      foreach my $parent ( @{ $P{ $class } } ) {
       182          push @parent_observers, $parent->_obs_get_observers_scoped;
       183      }
       184      return @parent_observers;
       185  }
       186
       187
       188  # Return observers ONLY for the specified item
       189
       190  sub _obs_get_observers_scoped {
```

```
191      my ( $item ) = @_;
192      return () unless ( ref $O{ $item } eq 'ARRAY' );
193      return @{ $O{ $item } };
194  }
195
196
197  # Used in debugging
198
199  sub _describe_item {
200      my ( $item ) = @_;
201      return "Class $item" unless ( ref $item );
202      my $item_class = ref $item;
203      if ( $item->can( 'id' ) ) {
204          return "Object of class $item_class with ID ", $item->id();
205      }
206      return "Instance of class $item_class";
207  }
208
209
210  my ( $DEBUG );
211  sub DEBUG     { return $DEBUG; }
212  sub SET_DEBUG { $DEBUG = $_[0] }
213
214
215  sub observer_log {
216      shift; $DEBUG && warn @_, "\n";
217  }
218
219  sub observer_error {
220      shift; die @_, "\n";
221  }
222
223  1;
224
225  __END__
```