Contract a pattern
Solution

1 Class *MY_LIST*

```
note
  description: "Lists without commitment to any particular
      representation."
  author: ""
  date: "$Date$"
  revision: "$Revision$"

deferred class
  MY_LIST [G]

inherit
  ANY
    redefine
      default_create
    end

feature -- Initialization
   default_create  is
      -- Create an empty list
    do
    ensure then
      is_empty: is_empty
    end

feature -- Status report
  is_empty: BOOLEAN is
      -- Is the list empty?
    deferred
    end

   has_active_iterators : BOOLEAN is
      -- Are there any active iterators attached to Current?
    do
      Result := active_iterator_count > 0
    end
```

```
feature -- Element change
    extend (x: G) is
        -- Add 'x' at the front
      require
        no_iterators: not has_active_iterators
      deferred
      ensure
        not_empty: not is_empty
      end

   remove is
        -- Remove an element from the front
      require
        not_empty: not is_empty
        no_iterators: not has_active_iterators
      deferred
      end

feature -- Iteration
   new_iterator: MY_ITERATOR [G] is
        -- New iterator attached to the list
      deferred
      end

feature {MY_ITERATOR} -- Iteration
    active_iterator_count : INTEGER
        -- Number of active iterators

    activate_iterator is
        -- Record a new active iterator
      do
         active_iterator_count := active_iterator_count + 1
      ensure
        one_more_iterator: active_iterator_count = old active_iterator_count
             + 1
      end

    deactivate_iterator is
        -- Record that an iterator became inactive
      require
         iterators_exist : active_iterator_count > 0
      do
         active_iterator_count := active_iterator_count - 1
      ensure
         one_less_iterator : active_iterator_count = old active_iterator_count
             - 1
      end

invariant
   interator_count_non_negative: active_iterator_count >= 0
```

```
end
```

2 Class *MY_ITERATOR*

```
note
  description: "Iterators through lists."
  author: ""
  date: "$Date$"
  revision: "$Revision$"

deferred class
  MY_ITERATOR [G]

inherit
  ANY
    redefine
      copy
    end

feature {NONE} -- Initialization
  make (l: MY_LIST [G]) is
      -- Create an iterator attached to 'l'
    require
      l_exists: l /= Void
    do
      list := l
    ensure
      is_off: off
    end

feature -- Access
  item: G is
      -- Current item
    require
      not_off: not off
    deferred
    end

  list: MY_LIST [G]
      -- The list to which iterator is attached

feature -- Status report
  off: BOOLEAN is
      -- Is the iterator not pointing to any valid location in the list?
    deferred
    end

feature -- Basic operations
  frozen start is
      -- Start iteration
```

```
    require
      is_off: off
    do
      if not list.is_empty then
        list.activate_iterator
        internal_start
      end
    ensure
      not_off_in_nonempty: not list.is_empty implies not off
      one_more_iterator_in_nonempty: not list.is_empty implies list.
          active_iterator_count = old list.active_iterator_count + 1
    end

  frozen forth is
      -- Go one step forward
    require
      not_off: not off
    do
      internal_forth
      if off then
        list.deactivate_iterator
      end
    ensure
      one_less_iterator_if_off: off implies list.active_iterator_count =
          old list.active_iterator_count - 1
    end

  frozen go_off is
      -- Deactivate iteration
    require
      not_off: not off
    do
      internal_go_off
      list.deactivate_iterator
    ensure
      is_off: off
      one_less_iterator: list.active_iterator_count = old list.
          active_iterator_count - 1
    end

feature -- Copy
  copy (other: like Current) is
      -- Copy 'other'
    do
      if other /= Current then
        standard_copy (other)
        if not off then
          list.activate_iterator
        end
      end
```

```
    ensure then
      same_list: list = other.list
      same_off: off = other.off
      same_item_if_not_off: not off implies item = other.item
      one_more_iterator_if_not_off : not off implies list.
          active_iterator_count = old list. active_iterator_count + 1
    end

feature {NONE} -- Implementation
  internal_start is
      -- Move cursor to the first element of ' list '
    require
      not_empty: not list.is_empty
    deferred
    ensure
      not_off: not off
    end

  internal_forth is
      -- Move cursor one step forward
    require
      not_off: not off
    deferred
    end

  internal_go_off is
      -- Make cursor point to no valid element of ' list '
    deferred
    ensure
      is_off: off
    end
end
```

3 Classes *MY_LINKABLE* and *MY_LINKED_LIST*

```
note
  description: "Linkable cells."
  author: ""
  date: "$Date$"
  revision: "$Revision$"

class
  MY_LINKABLE [G]

create
  put

feature -- Access
  item: G
      -- Content of cell
```

```
    right: like Current
        -- Right neighbor

feature -- Element change
    put (v: like item) is
        -- Make 'v' the cell's 'item'
      do
        item := v
      ensure
        item_inserted: item = v
      end

    put_right (other: like Current) is
        -- Put 'other' to the right of current cell
      do
        right := other
      ensure
        chained: right = other
      end
end

note
    description: "Lists implemented as singly-linked lists."
    author: ""
    date: "$Date$"
    revision: "$Revision$"

class
    MY_LINKED_LIST [G]

inherit
    MY_LIST [G]

feature -- Status report
    is_empty: BOOLEAN is
        -- Is the list empty?
      do
        Result := (first = Void)
      end

feature -- Element change
    extend (x: G) is
        -- Add 'x' at the front
      local
        new: MY_LINKABLE [G]
      do
        create new.put (x)
        new.put_right ( first )
        first := new
```

```
    end

  remove is
      -- Remove an element from the front
    do
      first := first.right
    end

feature -- Iteration
  new_iterator: MY_LINKED_ITERATOR [G] is
      -- New iterator attached to current
    do
      create Result.make (Current)
    end

feature {MY_LINKED_ITERATOR} -- Implementation
  first: MY_LINKABLE [G]
      -- First cell of the list
end
```

4 Class *MY_LINKED_ITERATOR*

```
note
  description: "Iterators through linked lists."
  author: ""
  date: "$Date$"
  revision: "$Revision$"

class
  MY_LINKED_ITERATOR [G]

inherit
  MY_ITERATOR [G]
    redefine
      list
    end

create {MY_LINKED_LIST}
  make

feature -- Access
  item: G is
      -- Current item
    do
      Result := active.item
    end

  list: MY_LINKED_LIST [G]
      -- The list to which iterator is attached
```

```
feature -- Status report
  off : BOOLEAN is
      -- Is the iterator not pointing to any valid location in the list ?
    do
      Result := (active = Void)
    end

feature {NONE} -- Implementation
  active: MY_LINKABLE [G]
      -- Current list cell

  internal_start is
      -- Move cursor to the first element of ' list '
    do
      active := list . first
    end

  internal_forth is
      -- Move cursor one step forward
    do
      active := active. right
    end

  internal_go_off is
      -- Make cursor point to no valid element of ' list '
    do
      active := Void
    end
end
```

5 Class *MY_ARRAYED_LIST*

```
note
  description: "Lists that store their elements in arrays."
  author: ""
  date: "$Date$"
  revision: "$Revision$"

class
  MY_ARRAYED_LIST [G]

inherit
  MY_LIST [G]
    redefine
      default_create
    end

feature -- Initialization
  default_create is
      -- Create an empty list
```

```
    do
      create array.make (1, default_capacity)
    end

feature -- Status report
  is_empty: BOOLEAN is
      -- Is the list empty?
    do
      Result := (first_index = 0)
    end

feature -- Element change
  extend (x: G) is
      -- Add 'x' at the front
    do
      if first_index = array.count then
        array. conservative_resize (1, array.count + default_capacity)
      end
      first_index := first_index + 1;
      array.put (x, first_index);
    end

  remove is
      -- Remove an element from the front
    do
      first_index := first_index - 1;
    end

feature -- Iteration
  new_iterator: MY_ARRAYED_ITERATOR [G] is
      -- New iterator attached to current
    do
      create Result.make (Current)
    end

feature {MY_ARRAYED_ITERATOR} -- Implementation
  array: ARRAY [G]
      -- Array to store list elements

  first_index : INTEGER
      -- Index of the first list element in the array

  default_capacity : INTEGER is 10
end
```

6 Class *MY_ARRAYED_ITERATOR*

```
note
  description: "Iterators through arrayed lists."
  author: ""
```

```
  date: "$Date$"
  revision: "$Revision$"

class
  MY_ARRAYED_ITERATOR [G]

inherit
  MY_ITERATOR [G]
    redefine
      list
    end

create {MY_ARRAYED_LIST}
  make

feature -- Access
  item: G is
      -- Current item
    do
      Result := list.array [index]
    end

  list : MY_ARRAYED_LIST [G]
      -- The list to which iterator is attached

feature -- Status report
  off : BOOLEAN is
      -- Is the iterator not pointing to any valid location in the list ?
    do
      Result := (index = 0)
    end

feature {NONE} -- Implementation
  index: INTEGER
      -- Index of current element

  internal_start is
      -- Move cursor to the first element of ' list '
    do
      index := list . first_index
    end

  internal_forth is
      -- Move cursor one step forward
    do
      index := index - 1
    end

  internal_go_off is
      -- Make cursor point to no valid element of ' list '
```

```
    do
      index := 0
    end

invariant
  index_not_too_small: index >= 0
  index_not_too_large : index <= list. first_index
end
```