Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

EiffelRSS

ADT Developer Guide

Michael Käser <kaeserm@student.ethz.ch>
Martin Luder <luderm@student.ethz.ch>
Thomas Weibel <weibelt@student.ethz.ch>

Informatik
Computer Science

Abstract

ADT contains the deferred classes SORTABLE and ORDER_RELATION which can be used to implement sortable structures.

Contents

List of Figures

1 Overview

ADT contains the deferred classes SORTABLE and ORDER_RELATION which can be used to implement sortable structures.

SORTABLE_TWO_WAY_LIST inherits from SORTABLE and TWO_WAY_LINKED_LIST to implement a sortable doubly-linked list.

See figure 1 for an overview of the cluster.

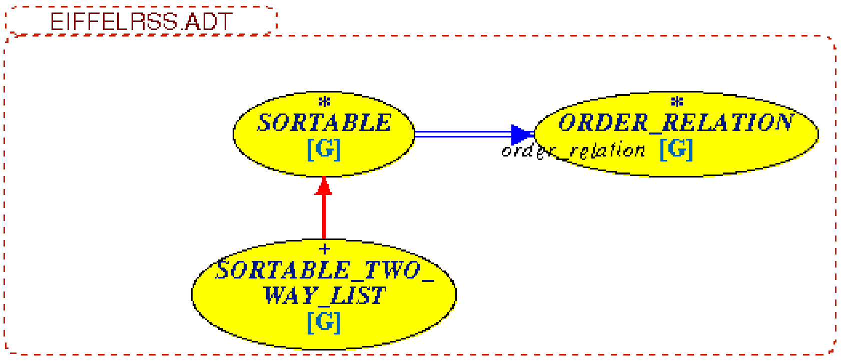

Figure 1: BON diagram of cluster ADT

Figure 2 shows the class SORTABLE_TWO_WAY_LIST.

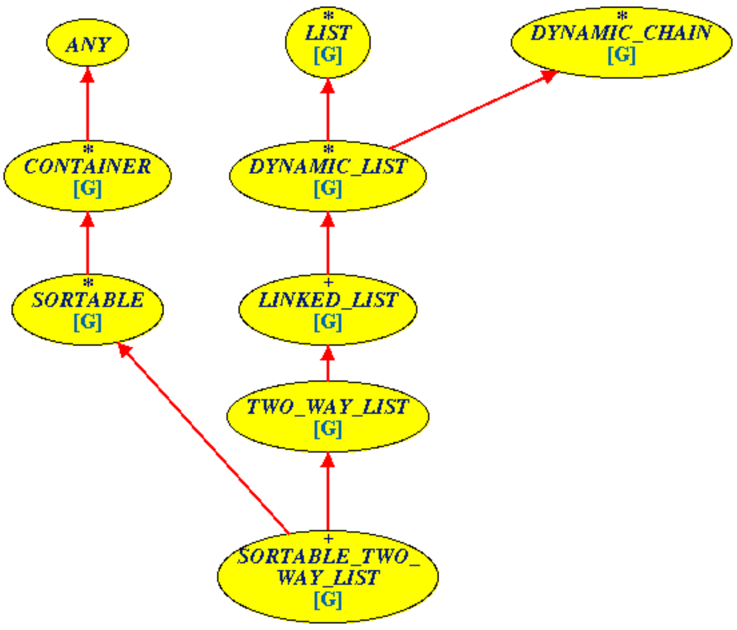

Figure 2: BON diagram of class SORTABLE_TWO_WAY_LIST

2 Usage

The following example shows a simple use-case for `SORTABLE_TWO_WAY_LIST`.

2.1 SORT_BY_NAME - a sorter for an address class

```
class
  SORT_BY_NAME[G -> ADDRESS]

inherit
  ORDER_RELATION[G]

feature -- Criterion

  ordered (first, second: G): BOOLEAN is
      -- Are 'first' and 'second' ordered (true if '
      first' < 'second')
    require else
      first_non_void: first /= Void
      second_non_void: second /= Void
    do
      Result := first.name < second.name
    end

end -- class SORT_BY_NAME
```

2.2 ADDRESS - a simple address class

```
class
  ADDRESS

inherit
  ANY
    redefine
      out
    end

create
  make

feature -- Initialization

  make (a_name: STRING; a_planet: STRING; a_phone_number
  : INTEGER) is
      -- Create a new address with name, street, city
      and phone number
    require
      non_empty_name: a_name /= Void and then not a_name↘
      →.is_empty
      non_empty_planet: a_planet /= Void and then not ↘
      →a_planet.is_empty
    do
      name := a_name
      planet := a_planet
      phone_number := a_phone_number
    ensure
      name_set: name = a_name
      planet_set: planet = a_planet
      phone_number_set: phone_number = a_phone_number
    end

feature -- Arguments

  name, planet: STRING

  phone_number: INTEGER

feature -- Output

  out: STRING is
      -- Returns a string representation of an address ↘
      →object
    do
      Result := "- Name: " + name + "%N- Planet: " + ↘
      →planet + "%N- Phone number: " + phone_number.out ↘
      →+ "%N"
    end

end -- class ADDRESS
```

2.3 Using ADDRESS and SORT_BY_NAME

```
class
  ADDRESS_BOOK

create
  make

feature -- Initialization

  make is
      -- Creation procedure.
    do
      create address_list.make
```

```
         create address.make ("Zaphod", "Betelgeuse", 12)
         address_list.extend (address)

         create address.make ("Marvin", "Sirius", 96)
         address_list.extend (address)

         create address.make ("Ford", "Betelgeuse", 25)
         address_list.extend (address)

         create address.make ("Trillian", "Earth", 23)
         address_list.extend (address)

         create address.make ("Arthur", "Earth", 42)
         address_list.extend (address)

         create address.make ("Slartibartfast", "Magrathea"↘
         →, 43)
         address_list.extend (address)

         io.put_string ("No particular sorting:%N")
         io.put_string ("=====================%N")
         address_list.do_all (agent print_address)

         io.put_string ("By name:%N")
         io.put_string ("========%N")
         address_list.set_order (create {SORT_BY_NAME[↘
         →ADDRESS]})
         address_list.sort
         address_list.do_all (agent print_address)
      end

feature -- Arguments

   address_list: SORTABLE_TWO_WAY_LIST[ADDRESS]
   address: ADDRESS

feature -- Output

   print_address (an_address: ADDRESS) is
         -- Prints address
      require
         address_non_void: address /= Void
      do
         io.put_string (an_address.out + "%N")
      end

end -- class ADDRESS_BOOK
```

3 Features of SORTABLE_TWO_WAY_LIST

Because `SORTABLE_TWO_WAY_LIST` inherits from `TWO_WAY_LIST`, all features of this class can also be applied to a `SORTABLE_TWO_WAY_LIST` object, e.g. `extend`, `prune` etc.

3.1 Initialization

3.1.1 make

```
make is
  -- Create an empty two way list, with no order ↘
  →relation
```

3.1.2 make_with_order_relation

```
make_with_order_relation (an_order_relation: ↘
→ORDER_RELATION[G]) is
  -- Create an empty two way list, with '↘
  →an_order_relation' as order relation
```

3.2 Access

3.2.1 has

```
has (v: G): BOOLEAN is
  -- Does structure include 'v'?
  -- (Reference or object equality,
  -- based on 'object_comparison'.)
```

3.3 Order relation

3.3.1 has_order

```
has_order: BOOLEAN is
  -- Is an order relation defined?
```

3.3.2 set_order

```
set_order (an_order_relation: ORDER_RELATION[G]) is
  -- Set the order relation.
```

3.4 Sorting

3.4.1 sort

```
sort is
  -- Sort all items.
```

3.4.2 sorted

```
sorted: BOOLEAN is
  -- Is the structure sorted?
```