

CSK

VDMTools[®]

VDM++ Sorting Algorithms

CSK

How to contact CSK:

http://www.csk.co.jp/index_e.html Web
① VDM.SP@cii.csk.co.jp General information

VDM++ Sorting Algorithms — Revised for V6.8.3

© COPYRIGHT 2005 by CSK CORPORATION

The software described in this document is furnished under a license agreement.
The software may be used or copied only under the terms of the license agreement.

This document is subject to change without notice

1 Introduction

This document contains a sorting example. The class diagram can be seen in Figure 1. The structure of the example is known as the *strategy* pattern. This pattern defines a family of algorithms, encapsulates each one and make them interchangeable. The *strategy* pattern lets the algorithm vary independently from clients that use it. The `SortMachine` class is the client that uses the different sorting algorithms. The `Sorter` class is an abstract class that defines a common interface to all supported algorithms.

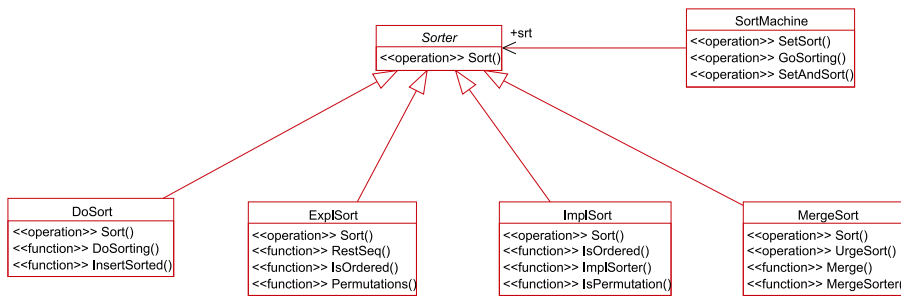


Figure 1: Class diagram for the sort example

2 Sort Machine

class *SortMachine*

instance variables

$srt : \text{Sorter} := \text{new MergeSort} ();$

The instance variable “srt” is an object reference to the sorting algorithm currently in use. The initial sorting algorithm is MergeSort.

Setting/changing which sorting algorithm to use.

operations

public

$\text{SetSort} : \text{Sorter} \xrightarrow{o} ()$

$\text{SetSort}(s) \triangleq$

$srt := s;$

Sorting with the sorting algorithm currently in use.

public

$\text{GoSorting} : \mathbb{Z}^* \xrightarrow{o} \mathbb{Z}^*$

$\text{GoSorting}(arr) \triangleq$

$\text{return } srt.\text{Sort}(arr);$

Set/change first the sorting algorithm and sort afterwards.

public

$\text{SetAndSort} : \text{Sorter} \times \mathbb{Z}^* \xrightarrow{o} \mathbb{Z}^*$

$\text{SetAndSort}(s, arr) \triangleq$

($srt := s;$

$\text{return } srt.\text{Sort}(arr)$

)

end *SortMachine*

3 Sorter class

```
class Sorter
operations
public
     $Sort : \mathbb{Z}^* \xrightarrow{o} \mathbb{Z}^*$ 
     $Sort(arg) \triangleq$ 
        is subclass responsibility
end Sorter
```

4 MergeSort

class *MergeSort* is subclass of *Sorter*

operations

public

$Sort : \mathbb{Z}^* \xrightarrow{o} \mathbb{Z}^*$

$Sort(l) \triangleq$

return *MergeSorter*(*l*)

functions

$MergeSorter : \mathbb{R}^* \rightarrow \mathbb{R}^*$

$MergeSorter(l) \triangleq$

cases *l* :

$[] \rightarrow l,$

$[e] \rightarrow l,$

others \rightarrow let $l1 \curvearrowright l2 \in \{l\}$ be st $\text{abs}(\text{len } l1 - \text{len } l2) < 2$ in

let $l-l = MergeSorter(l1),$

$l-r = MergeSorter(l2)$ in

$Merge(l-l, l-r)$

end;

$Merge : \mathbb{Z}^* \times \mathbb{Z}^* \rightarrow \mathbb{Z}^*$

$Merge(l1, l2) \triangleq$

cases $mk-(l1, l2)$:

$mk-([], l), mk-(l, []) \rightarrow l,$

others \rightarrow if $\text{hd } l1 \leq \text{hd } l2$

then $[\text{hd } l1] \curvearrowright Merge(\text{tl } l1, l2)$

else $[\text{hd } l2] \curvearrowright Merge(l1, \text{tl } l2)$

end

pre $\forall i \in \text{inds } l1 \cdot l1(i) \geq 0 \wedge$

$\forall i \in \text{inds } l2 \cdot l2(i) \geq 0$

end *MergeSort*

5 DoSort

class *DoSort* is subclass of *Sorter*

operations

public

$Sort : \mathbb{Z}^* \xrightarrow{o} \mathbb{Z}^*$

$Sort(l) \triangleq$

return *DoSorting*(*l*)

functions

$DoSorting : \mathbb{Z}^* \rightarrow \mathbb{Z}^*$

$DoSorting(l) \triangleq$

if $l = []$

then $[]$

else let *sorted* = *DoSorting*(tl *l*) in

InsertSorted(hd *l*, *sorted*);

$InsertSorted : \mathbb{Z} \times \mathbb{Z}^* \rightarrow \mathbb{Z}^*$

$InsertSorted(i, l) \triangleq$

cases true :

$(l = []) \rightarrow [i],$

$(i \leq \text{hd } l) \rightarrow [i] \frown l,$

others $\rightarrow [\text{hd } l] \frown InsertSorted(i, \text{tl } l)$

end

end *DoSort*

An overview of the test coverage information for the *DoSort* class is listed in the table below. The test coverage information is generated using the argument file *sort.arg*.

Test Suite : vdm.tc

Class : DoSort

Name	#Calls	Coverage
DoSort'DoSorting	undefined	undefined
DoSort'InsertSorted	undefined	undefined
DoSort'Sort	undefined	undefined
Total Coverage		0%

6 ImplSort

The class *ImplSort* is an example of an sorting algorithm defined by implicit functions.

class *ImplSort* is subclass of *Sorter*

operations

public

$Sort : \mathbb{Z}^* \xrightarrow{o} \mathbb{Z}^*$

$Sort(l) \triangleq$

 return *ImplSorter*(*l*)

functions

public

$ImplSorter(l : \mathbb{Z}^*) r : \mathbb{Z}^*$

post

$IsPermutation(r, l) \wedge IsOrdered(r);$

$IsPermutation : \mathbb{Z}^* \times \mathbb{Z}^* \rightarrow \mathbb{B}$

$IsPermutation(l1, l2) \triangleq$

$\forall e \in (\text{elems } l1 \cup \text{elems } l2) \cdot$

$\text{card } \{i \mid i \in \text{inds } l1 \cdot l1(i) = e\} =$

$\text{card } \{i \mid i \in \text{inds } l2 \cdot l2(i) = e\};$

$IsOrdered : \mathbb{Z}^* \rightarrow \mathbb{B}$

$IsOrdered(l) \triangleq$

$\forall i, j \in \text{inds } l \cdot i > j \Rightarrow l(i) \geq l(j)$

end *ImplSort*

7 ExplSort

The class *ExplSort* is a refinement of the algorithm described in *ImplSort*.

class *ExplSort* is subclass of *Sorter*

operations

public

$Sort : \mathbb{Z}^* \xrightarrow{o} \mathbb{Z}^*$

$Sort(l) \triangleq$

let $r \in Permutations(l)$ be st $IsOrdered(r)$ in

return r

functions

$Permutations : \mathbb{Z}^* \rightarrow \mathbb{Z}^*\text{-set}$

$Permutations(l) \triangleq$

cases l :

$[], [-] \rightarrow \{l\},$

others $\rightarrow \bigcup \{ \{ [l(i)] \curvearrowright j \} \mid$

$j \in Permutations(RestSeq(l, i)) \} \mid$

$i \in \text{inds } l \}$

end;

$RestSeq : \mathbb{Z}^* \times \mathbb{N} \rightarrow \mathbb{Z}^*$

$RestSeq(l, i) \triangleq$

$[l(j) \mid j \in (\text{inds } l \setminus \{i\})]$

pre $i \in \text{inds } l$ post $\text{elems } RESULT \subseteq \text{elems } l \wedge$

$\text{len } RESULT = \text{len } l - 1;$

$IsOrdered : \mathbb{Z}^* \rightarrow \mathbb{B}$

$IsOrdered(l) \triangleq$

$\forall i, j \in \text{inds } l \cdot i > j \Rightarrow l(i) \geq l(j)$

end *ExplSort*