

# VDM++ specification of calculating railway fare

Shin Sahara

Hosei University

## Abstract

This is an example of industrial requirement specification.  
This example calculates fare by distance using sequence of record.

## Contents

<b>1</b>	<b>Class Diagram</b>	<b>2</b>
<b>2</b>	<b>Calculate_fare</b>	<b>3</b>
<b>3</b>	<b>FareTableDictionary</b>	<b>4</b>
3.1	FareTable type . . . . .	4
<b>4</b>	<b>RailNet</b>	<b>6</b>
<b>5</b>	<b>RouteSearch</b>	<b>7</b>
<b>6</b>	<b>RouteSearchByDijkstras</b>	<b>8</b>
<b>7</b>	<b>Shortest_route</b>	<b>8</b>
<b>8</b>	<b>DijkstrasAlgorithm</b>	<b>9</b>
<b>9</b>	<b>Test</b>	<b>11</b>
<b>10</b>	<b>TestApp Class</b>	<b>14</b>
10.1	Responsibility . . . . .	14
10.2	Operation : run . . . . .	14
<b>11</b>	<b>TestCaseT0001,T0002,T0003,T0004 classes</b>	<b>15</b>
11.1	Responsibility . . . . .	15
<b>12</b>	<b>UseFare</b>	<b>18</b>
<b>13</b>	<b>References, Index</b>	<b>20</b>

# 1 Class Diagram

This is the class diagram of the VDM++ specification. The layer of the class is on upper right of the class box.

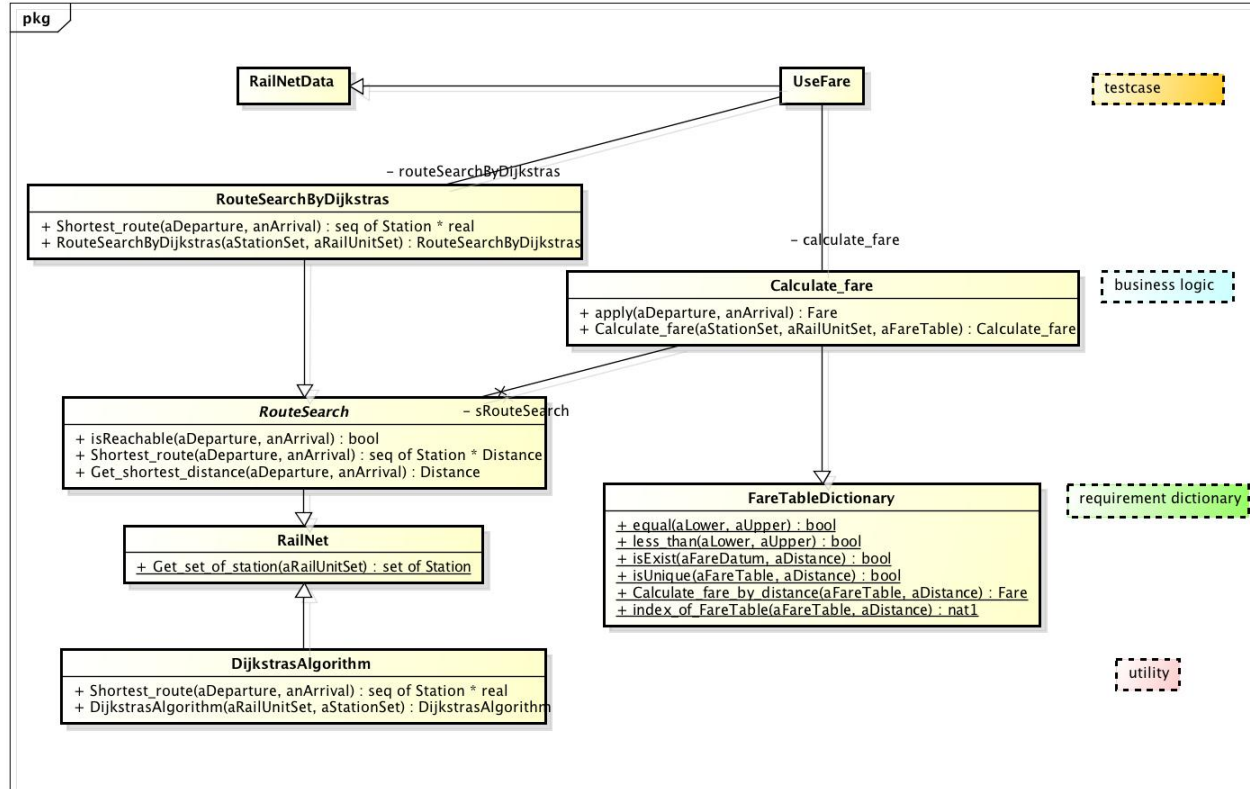


Figure 1: Class Diagram

## 2 Calculate\_fare

I am a requirement specification of calculating fare.

```

.....
class
Calculate_fare is subclass of FareTableDictionary
instance variables
    sFareTable : FareTable;
    sStationSet : RailNet'StationSet;
    sRailUnitSet : RouteSearch'RailUnitSet;
    sRouteSearch : RouteSearch;

operations
public
    Calculate_fare : RailNet'StationSet * RouteSearch'RailUnitSet * FareTable ==> Calculate_fa
    Calculate_fare (aStationSet, aRailUnitSet, aFareTable) ==
        (  sFareTable := aFareTable;
          sStationSet := aStationSet;
          sRailUnitSet := aRailUnitSet;
          sRouteSearch := new RouteSearchByDijkstras (aStationSet, aRailUnitSet);
          return self
        );
public
    apply : RailNet'Station * RailNet'Station ==> Fare
    apply (aDeparture, anArrival) ==
        let distance = sRouteSearch.Get_shortest_distance (aDeparture, anArrival) in
        return Calculate_fare_by_distance (sFareTable, distance)
    pre sRouteSearch.isReachable (aDeparture, anArrival)
end
Calculate_fare
.....
Test Suite :      vdm.tc
Class :          Calculate_fare

```

Name	#Calls	Coverage
Calculate_fare'apply	18	✓
Calculate_fare'Calculate_fare	18	✓
<b>Total Coverage</b>		<b>100%</b>

### 3 FareTableDictionary

I am a requirement dictionary that defines the term of Fare Table. I define the noun by the class, the type, the value, and the instance variable. Moreover, I define the predicate by the function and the operation.

```

.....
class
FareTableDictionary
types
  public Fare = nat;
public
  FareDatum::fLower : RailNet'Distance
              fUpper : RailNet'Distance
              fFare :- Fare;
.....

```

#### 3.1 FareTable type

Each element of FareTable doesn't overlap.

```

.....
public FareTable = seq of FareDatum
inv wFareTable ==
  forall i,j in set inds wFareTable &
    less_than (wFareTable (i).fLower, wFareTable (i).fUpper) and
    j = i + 1 =>
    equal (wFareTable (i).fUpper, wFareTable (j).fLower)
functions
public static
  Calculate_fare_by_distance : FareTable * RailNet'Distance -> Fare
  Calculate_fare_by_distance (aFareTable, aDistance) ==
    let n = index_of_FareTable (aFareTable, aDistance) in
    aFareTable (n).fFare
pre isUnique (aFareTable, aDistance)
post let n = index_of_FareTable (aFareTable, aDistance) in
  RESULT = aFareTable (n).fFare ;
public static
  isExist : FareDatum * RailNet'Distance +> bool
  isExist (aFareDatum, aDistance) ==
    aFareDatum.fLower <= aDistance and aDistance < aFareDatum.fUpper;
public static
  isUnique : FareTable * RailNet'Distance -> bool
  isUnique (aFareTable, aDistance) ==
    exists1 i in set inds aFareTable & isExist (aFareTable (i), aDistance);
public static
  index_of_FareTable : FareTable * RailNet'Distance -> nat1
  index_of_FareTable (aFareTable, aDistance) ==
    let i in set inds aFareTable be st isExist (aFareTable (i), aDistance) in
    i;
public static

```

```

less_than : Fare * Fare -> bool
less_than (aLower, aUpper) ==
  aLower < aUpper;
public static
  equal : Fare * Fare -> bool
  equal (aLower, aUpper) ==
    aLower = aUpper
end
FareTableDictionary

```

```

.....
Test Suite :    vdm.tc
Class :       FareTableDictionary

```

Name	#Calls	Coverage
FareTableDictionary'equal	288	✓
FareTableDictionary'isExist	204	✓
FareTableDictionary'isUnique	18	✓
FareTableDictionary'less_than	1800	✓
FareTableDictionary'index_of_FareTable	36	✓
FareTableDictionary'Calculate_fare_by_distance	18	✓
<b>Total Coverage</b>		<b>100%</b>

## 4 RailNet

I am a domain model of the Rail Net. I am independent of calculating fare model.

```

.....
class
RailNet
types
  public Station = token;
  public Distance = real;
  public StationSet = set of Station
  inv wStationSet == card wStationSet >= 2;
public
  RailUnit::fDeparture : Station
           fArrival : Station
           fDistance : Distance
  inv wRailUnit ==
    wf_RailUnit (wRailUnit.fDeparture, wRailUnit.fArrival, wRailUnit.fDistance);
  public RailUnitSet = set of RailUnit
  inv wRailUnitSet == wRailUnitSet <> {}
functions

  wf_RailUnit : Station * Station * Distance -> bool
  wf_RailUnit (aDeparture, anArrival, aDistance) ==
    isDifferent_station (aDeparture, anArrival) and isDistance_not_zero (aDistance);

  isDifferent_station : Station * Station -> bool
  isDifferent_station (aStation1, aStation2) ==
    aStation1 <> aStation2;

  isDistance_not_zero : Distance -> bool
  isDistance_not_zero (aDistance) ==
    aDistance > 0;
public static
  Get_set_of_station : RailUnitSet -> set of Station
  Get_set_of_station (aRailUnitSet) ==
    dunion {{wRailUnit.fDeparture, wRailUnit.fArrival} | wRailUnit in set aRailUnitSet}
end
RailNet
.....
Test Suite :    vdm.tc
Class :        RailNet

```

Name	#Calls	Coverage
RailNet'wf_RailUnit	3930	✓
RailNet'Get_set_of_station	0	0%
RailNet'isDifferent_station	3930	✓
RailNet'isDistance_not_zero	3930	✓
<b>Total Coverage</b>		<b>63%</b>

## 5 RouteSearch

I am an abstract class of route search.

```

.....
class
RouteSearch is subclass of RailNet
instance variables
  protected sStationSet : StationSet;
  protected sRailUnitSet : RailUnitSet;
  inv let wStationSet = Get_set_of_station (sRailUnitSet) in
    wStationSet subset sStationSet

operations
public
  Shortest_route : Station * Station ==> seq of Station * Distance
  Shortest_route (aDeparture, anArrival) ==
    is subclass responsibility;
public
  Get_shortest_distance : Station * Station ==> Distance
  Get_shortest_distance (aDeparture, anArrival) ==
    ( def mk_(-, distance) = Shortest_route (aDeparture, anArrival)
in
    return distance
  )
pre isReachable (aDeparture, anArrival) ;
public
  isReachable : Station * Station ==> bool
  isReachable (aDeparture, anArrival) ==
    let mk_ (r, -) = Shortest_route (aDeparture, anArrival) in
    return len r > 0
end
RouteSearch
.....
Test Suite :    vdm.tc
Class :        RouteSearch

```

Name	#Calls	Coverage
RouteSearch'isReachable	54	✓
RouteSearch'Shortest_route	0	0%
RouteSearch'Get_shortest_distance	33	✓
<b>Total Coverage</b>		<b>94%</b>

## 6 RouteSearchByDijkstras

I am a concrete class of route search using Dijkstra's Algorithm.

```

.....
class
RouteSearchByDijkstras is subclass of RouteSearch
operations
public
  RouteSearchByDijkstras : StationSet * RailUnitSet ==> RouteSearchByDijkstras
  RouteSearchByDijkstras (aStationSet, aRailUnitSet) == atomic
    ( sStationSet := aStationSet;
      sRailUnitSet := aRailUnitSet
    );
.....

```

## 7 Shortest\_route

I get a shortest route by using Dijkstra's Algorithm.

```

.....
public
  Shortest_route : Station * Station ==> seq of Station * real
  Shortest_route (aDeparture, anArrival) ==
    ( dcl wAlgorithm : DijkstrasAlgorithm := new DijkstrasAlgorithm (sRailUnitSet, sStationS
      return wAlgorithm.Shortest_route (aDeparture, anArrival)
    )
end
RouteSearchByDijkstras
.....
Test Suite :      vdm.tc
Class :          RouteSearchByDijkstras

```

Name	#Calls	Coverage
RouteSearchByDijkstras'Shortest_route	96	✓
RouteSearchByDijkstras'RouteSearchByDijkstras	36	✓
<b>Total Coverage</b>		<b>100%</b>



## 8 DijkstrasAlgorithm

I'm the Dijkstra's Algorithm. I get the shortest route from a vertex to another vertex.

```

.....
class
DijkstrasAlgorithm is subclass of RailNet
types
  public Decided = <NotDecided> | <Decided>;
  public XDecided = map Station to Decided;
  public VShortest = map Station to real;
  public PPrevStation = map Station to Station
instance variables
  public sRailUnitSet : RailUnitSet;
  public sStationSet : StationSet;
  public x : XDecided := { |-> };
  public v : VShortest := { |-> };
  public p : PPrevStation := { |-> };

values

  vStation = 1000000000000000000
operations
public
  DijkstrasAlgorithm : set of RailUnit * set of Station ==> DijkstrasAlgorithm
  DijkstrasAlgorithm(aRailUnitSet,aStationSet) ==
    ( sRailUnitSet := aRailUnitSet;
      sStationSet := aStationSet
    );
public
  Shortest_route : Station * Station ==> seq of Station * real
  Shortest_route(aDeparture,anArrival) ==
    ( dcl i : Station := aDeparture;
      for all wStation in set sStationSet
      do if wStation = aDeparture
        then ( x(aDeparture) := <Decided>;
              v(aDeparture) := 0
            )
        else ( x(wStation) := <NotDecided>;
              v(wStation) := vStation
            ) ;
      for all - in set sStationSet
      do ( def Ni = {u.fArrival | u in set sRailUnitSet & u.fDeparture = i};
          Nu = {u | u in set sRailUnitSet & u.fDeparture = i}
        in
          ( for all j in set Ni

```

```

do ( if x(j) = <NotDecided>
    then def w = v(i) + d(Nu,i,j) in
        if w < v(j)
        then ( v(j) := w;
                p(j) := i
              )
      ) ;
def NiNotDecided = {e | e in set Ni & x(e) = <NotDecided>}
in
    if NiNotDecided <> {}
    then let s in set NiNotDecided be st forall s1 in set NiNotDecided & v(s1) < v(s)
        ( i := s;
          x(i) := <Decided>
        )
    ) ;
def wShortestPath = makeRoot(p,aDeparture,anArrival);
wShortest = v(anArrival) in
return mk_(wShortestPath,wShortest)
)
functions
d : RailUnitSet * Station * Station -> real
d(aRailUnitSet,aStation1,aStation2) ==
let di in set aRailUnitSet be st di.fDeparture = aStation1 and di.fArrival = aStation2
di.fDistance
operations
makeRoot : PPrevStation * Station * Station ==> seq of Station
makeRoot(aPPrevStation,aDeparture,anArrival) ==
( dcl wShortestPath : seq of Station := [],
  wStation : Station := anArrival;
  while wStation <> aDeparture
  do ( wShortestPath := [wStation]^wShortestPath;
        wStation := aPPrevStation(wStation)
      ) ;
  return [aDeparture]^wShortestPath
)
pre anArrival in set dom aPPrevStation
end
DijkstrasAlgorithm
.....
Test Suite : vdm.tc
Class : DijkstrasAlgorithm

```

Name	#Calls	Coverage
DijkstrasAlgorithm'd	576	✓
DijkstrasAlgorithm'makeRoot	96	✓
DijkstrasAlgorithm'Shortest_route	96	✓
DijkstrasAlgorithm'DijkstrasAlgorithm	96	✓
<b>Total Coverage</b>		<b>100%</b>

## 9 Test

I'm a simple regression test.

```

.....
class
TestSimple is subclass of RailNetData
values
public
    vRouteSearch = new RouteSearchByDijkstras (RailNetData'vStationSet,RailNetData'vRailUnitSe
public
    vMaxValue = 100000000;
public
    vCalculate_fare = new Calculate_fare (

                                RailNetData'vStationSet,
                                RailNetData'vRailUnitSet,
                                [
                                    mk_FareTableDictionary'FareDatum (0,3,150),
                                    mk_FareTableDictionary'FareDatum (3,8,160),
                                    mk_FareTableDictionary'FareDatum (8,10,190),
                                    mk_FareTableDictionary'FareDatum (10,20,220),
                                    mk_FareTableDictionary'FareDatum (20,vMaxValue,250)])

operations
public
    run : () ==> seq of char * bool * map nat to bool
    run () ==
        let testcases = [
            t1 (),t2 (),t3 (),t4 (),t5 (),
            t6 (),t7 (),t8 ()],
            testResults = makeOrderMap (testcases) in
        return mk_ ("The result of regression test = ", forall i in set inds testcases & testcase
functions
public static
    makeOrderMap : seq of bool +> map nat to bool
    makeOrderMap (s) ==
        {i |-> s (i) | i in set inds s}
operations
public
    print : seq of char ==> ()
    print (s) ==
        let - = new IO ().echo (s) in
        skip;
public
    t1 : () ==> bool
    t1 () ==
        let mk_ (-,d) = vRouteSearch.Shortest_route (vYotsuya,vShinagawa) in
        return d = 9.5 and
            vRouteSearch.Get_shortest_distance (vYotsuya,vShinagawa) = 9.5;
public

```

```

t2 : () ==> bool
t2 () ==
  trap <RuntimeError> with ( print("\t t2 meet the deliberate pre-
condition error.\n");
    return true
  ) in
  return vRouteSearch.isReachable (vTokyo,vCopenhagen) = false;
public
t3 : () ==> bool
t3 () ==
  let mk_ (r,d) = vRouteSearch.Shortest_route (vTokyo,vShinjuku) in
  return r = [vTokyo,vYotsuya,vShinjuku] and
    d = 7.7;
public
t4 : () ==> bool
t4 () ==
  let mk_ (r,d) = vRouteSearch.Shortest_route (vIkebukuro,vYotsuya) in
  return r = [vIkebukuro,vShinjuku,vYotsuya] and
    d = 8.6;
public
t5 : () ==> bool
t5 () ==
  let mk_ (r,d) = vRouteSearch.Shortest_route (vShinagawa,vYotsuya) in
  return r = [vShinagawa,vTokyo,vYotsuya] and
    d = 9.3;
public
t6 : () ==> bool
t6 () ==
  return vCalculate_fare.apply (vIkebukuro,vTokyo) = 220;
public
t7 : () ==> bool
t7 () ==
  return vCalculate_fare.apply (vIkebukuro,vShinjuku) = 160;
public
t8 : () ==> bool
t8 () ==
  return vCalculate_fare.apply (vYotsuya,vShinagawa) = 190
end
TestSimple
.....
Test Suite :      vdm.tc
Class :          TestSimple

```

Name	#Calls	Coverage
TestSimple't1	3	✓
TestSimple't2	3	86%
TestSimple't3	3	✓
TestSimple't4	3	✓
TestSimple't5	3	✓

Name	#Calls	Coverage
TestSimple't6	3	✓
TestSimple't7	3	✓
TestSimple't8	3	✓
TestSimple'run	3	✓
TestSimple'print	3	✓
TestSimple'makeOrderMap	3	✓
<b>Total Coverage</b>		<b>98%</b>

## 10 TestApp Class

### 10.1 Responsibility

I'm a industrial level regression test of calculating railway fare.

```
.....
class
TestApp
.....
```

### 10.2 Operation : run

Appending regression test cases to Testsuite, executing, and decides succeeded.

```
.....
operations
public
  run : () ==> ()
  run () ==
    ( dcl ts : TestSuite := new TestSuite ("The regression test of calculating railway fare
      tr : TestResult := new TestResult ();
      tr.addListener(new PrintTestListener());
      ts.addTest(new TestCaseT0001 ("TestCaseT0001 Calculate fare from Tokyo to Shinjuku.\n
      ts.addTest(new TestCaseT0002 ("TestCaseT0002 Calculate fare from vIkebukuro to vYots
      ts.addTest(new TestCaseT0003 ("TestCaseT0003 Calculate fare from Yotsuya to Shinagaw
      ts.addTest(new TestCaseT0004 ("TestCaseT0004 Calculate fare from Yotsuya to Copenhag
      ts.run(tr);
      if tr.wasSuccessful () = true
      then def - = new IO ().echo ("*** All regression test succeeded. *
** \n") in
          skip
        else def - = new IO ().echo ("*** There are errors in the regression test cases. *
** \n") in
          skip
        )
    end
TestApp
.....
Test Suite :      vdm.tc
Class :          TestApp
```

Name	#Calls	Coverage
TestApp'run	3	85%
<b>Total Coverage</b>		<b>85%</b>

## 11 TestCaseT0001,T0002,T0003,T0004 classes

### 11.1 Responsibility

I'm testcases of the Get\_shortest\_distance and the Calculate\_fare.

```

.....
class
TestCaseT is subclass of TestCase,RailNetData
values
public
    vRouteSearch = new RouteSearchByDijkstras (RailNetData'vStationSet,RailNetData'vRailUnitSe
public
    vMaxValue = 100000000;
public
    vCalculate_fare = new Calculate_fare (

                                RailNetData'vStationSet,
                                RailNetData'vRailUnitSet,
                                [
                                    mk_FareTableDictionary'FareDatum (0,3,150),
                                    mk_FareTableDictionary'FareDatum (3,8,160),
                                    mk_FareTableDictionary'FareDatum (8,10,190),
                                    mk_FareTableDictionary'FareDatum (10,20,220),
                                    mk_FareTableDictionary'FareDatum (20,vMaxValue,250)])

operations
public
    print : seq of char ==> ()
    print (s) ==
        let - = new IO ().echo (s) in
        skip
    end
TestCaseT
class
TestCaseT0001 is subclass of TestCaseT
operations
public
    TestCaseT0001 : seq of char ==> TestCaseT0001
    TestCaseT0001 (name) ==
        setName(name) ;
public
    test01 : () ==> ()
    test01 () ==
        (   def wDistance =
                vRouteSearch.Get_shortest_distance (vTokyo,vShinjuku)
in
        assertTrue("\t test01 Fault in calculating.\n",
            wDistance = 7.7 and
            vCalculate_fare.apply (vTokyo,vShinjuku) = 160)
        )
    end

```

```

TestCaseT0001
class
TestCaseT0002 is subclass of TestCaseT
operations
public
    TestCaseT0002 : seq of char ==> TestCaseT0002
    TestCaseT0002 (name) ==
        setName(name) ;
public
    test01 : () ==> ()
    test01 () ==
        (   def wDistance =
                vRouteSearch.Get_shortest_distance (vIkebukuro, vYotsuya)
in
            assertTrue("\t test01 Fault in calculating.\n",
                wDistance = 8.6 and
                vCalculate_fare.apply (vIkebukuro, vYotsuya) = 190)
        )
end
TestCaseT0002
class
TestCaseT0003 is subclass of TestCaseT
operations
public
    TestCaseT0003 : seq of char ==> TestCaseT0003
    TestCaseT0003 (name) ==
        setName(name) ;
public
    test01 : () ==> ()
    test01 () ==
        (   def wDistance =
                vRouteSearch.Get_shortest_distance (vYotsuya, vShinagawa)
in
            assertTrue("\t test01 Fault in calculating.\n",
                wDistance = 9.5 and
                vCalculate_fare.apply (vYotsuya, vShinagawa) = 190)
        )
end
TestCaseT0003
class
TestCaseT0004 is subclass of TestCaseT
operations
public
    TestCaseT0004 : seq of char ==> TestCaseT0004
    TestCaseT0004 (name) ==
        setName(name) ;
public

```



```
test01 : () ==> ()
test01 () ==
  ( trap <RuntimeError> with ( print("\t TestCaseT0004'test01 meet the planned pre-
condition error.\n")
    ) in
    def wDistance =
      vRouteSearch.Get_shortest_distance (vYotsuya,vCopenhagen)
in
  assertTrue("\t test01 Fault in calculating.\n",
    wDistance = 9.5 and
    vCalculate_fare.apply (vYotsuya,vShinagawa) = 190)
end
TestCaseT0004
.....
```

## 12 UseFare

I'm a combinatoial test of calculating fare.

```

.....
class
UseFare is subclass of RailNetData
values
public
    vRouteSearch = new RouteSearchByDijkstras (RailNetData'vStationSet,RailNetData'vRailUnitSe
public
    vMaxValue = 100000000;
public
    vCalculate_fare = new Calculate_fare (

                                                RailNetData'vStationSet,
                                                RailNetData'vRailUnitSet,
                                                [
                                                    mk_FareTableDictionary'FareDatum (0,1,150),
                                                    mk_FareTableDictionary'FareDatum (1,2,160),
                                                    mk_FareTableDictionary'FareDatum (2,3,190),
                                                    mk_FareTableDictionary'FareDatum (3,4,220),
                                                    mk_FareTableDictionary'FareDatum (4,vMaxValue,250)])

instance variables
    public t0 : TestSimple := new TestSimple ();
    public t1 : TestApp := new TestApp ();

traces
T0 :
    t0.run ()
    ;
; T1 :
    t1.run ()
    ;
; T2 :
    let s1 in set {vTokyo,vShinjuku,vShinagawa,vYotsuya,vIkebukuro,vCopenhagen} in
    let s2 in set {vShinjuku,vShinagawa,vYotsuya,vIkebukuro,vTokyo,vCopenhagen} in
    vRouteSearch.Get_shortest_distance (s1,s2)
    ;
; T3 :
    let s1 in set {vTokyo,vShinjuku,vShinagawa,vYotsuya,vIkebukuro,vCopenhagen} in
    let s2 in set {vShinjuku,vShinagawa,vYotsuya,vIkebukuro,vTokyo,vCopenhagen} in
    vCalculate_fare.apply (s1,s2)
    ;
end
UseFare
.....
Test Suite :    vdm.tc
Class :        UseFare

```

Name	#Calls	Coverage
------	--------	----------

Name	#Calls	Coverage
Total Coverage		undefined

## 13 References, Index

VDM++[\[1\]](#) is a formal specification description language that extended VDM-SL[\[2\]](#) developed by IBM Vienna Research Center in the mid-1970 and further object oriented extension.

### References

- [1] Kyushu University. *VDMTools VDM++ Language Manual*. Kyushu University, 2.0 edition, 2016.
- [2] Kyushu University. *VDMTools VDM-SL Language Manual*. Kyushu University, 2.0 edition, 2016.