Train Fare System Specification by VDM++

Shin Sahara

Hosei University

Abstract

This is an example of requirement specification for calculation of train fare. This example uses operations, and a map type. This example includes type invariants, instance variable invariants, post-conditions, pre-conditions, and a simple regression test too. There is another test mechanism called "combinatorial test".

Contents

1 Fare

I'm a train fare in requirement specification layer.

```
class
Fare
types
 public Money = nat;
 public Section = set of Station
 inv s == card s = 2;
 public Station = seq of char
 inv s == s <> "";
 public FareTable = map Section to Money
instance variables
 sFareTable : FareTable := { |-> };

operations
public
 Fare : FareTable ==> Fare
 Fare (aFareTable) ==
   (   sFareTable := aFareTable;
       return self
   ) ;
```

1.1 Calculate_fare

Calculate fare of section using a map which maps section to money.
This operation is a sample of exception processing.

```
public
 Calculate_fare : Section ==> Money
 Calculate_fare (aSection) ==
   if aSection not in set dom sFareTable
   then exit <SectionIsNotInFareTable>
   else return sFareTable (aSection)
 post RESULT = sFareTable (aSection) ;
```

1.2 AppendFareTable

Append a maplet to sFareTable.
This operation is a sample of exception processing.

```
public
 AppendFareTable : FareTable ==> ()
 AppendFareTable (aFareTable) ==
   if dom aFareTable subset dom sFareTable
   then exit <DuplicatedFareData>
   else sFareTable := sFareTable munion aFareTable
```

```
  post sFareTable = sFareTable~ munion aFareTable ;
public
  GetFareTable : () ==> FareTable
  GetFareTable () ==
    return sFareTable
end
Fare
```

Test Suite : vdm.tc
Class : Fare

Name	#Calls	Coverage
Fare‘Fare	3	√
Fare‘GetFareTable	0	0%
Fare‘Calculate_fare	2	√
Fare‘AppendFareTable	0	0%
Total Coverage		**53%**

2 TestApp Class

2.1 Responsibility

Do the regression test of calculating railway fare.

```
class
TestApp
```

2.2 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 succeeded calculating.\n"));
        ts.addTest(new TestCaseT0002 ("TestCaseT0002 not succeeded calculating.\n"));
        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	1	82%
Total Coverage		**82%**

3 TestCaseT

3.1 Responsibility

Test the Get_shortest_distance and the Calculate_fare.

```
.....................................................................
class
TestCaseT is subclass of TestCase
instance variables
  public sFare : Fare := new Fare({{"Tokyo","Shinagawa"} |-> 220,{"Tokyo","Shinjuku"} |-> 180

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() ==
    (   assertTrue("\n test01 Fault in calculating.\n",
              sFare.Calculate_fare({"Tokyo","Shinagawa"}) = 220)
    )
end
TestCaseT0001
class
TestCaseT0002 is subclass of TestCaseT
operations
public
  TestCaseT0002 : seq of char ==> TestCaseT0002
  TestCaseT0002(name) ==
    setName(name);
public
  test01 : () ==> ()
  test01() ==
    trap <SectionIsNotInFareTable> with print("\t test01 meet the deliberate <SectionIsNotI
    (   assertTrue("\t test01 didn't meet the deliberate <SectionIsNotInFareTable> exceptio
              sFare.Calculate_fare({"Osaka","Kyoto"}) = 190)
    )
end
TestCaseT0002
```

. . . . . .

4 UseFare1 – Combinatorial test

I am a combinatoial test of calculating fare.

```
.....................................................................
class
UseFare2 is subclass of Fare
values

  vFare = new Fare({{"Tokyo","Shinagawa"} |-> 220,{"Tokyo","Shinjuku"} |-> 180})
traces
T1 :
  let s1 in set {"Tokyo","Shinagawa","Shinjuku","Osaka"} in
  let s2 in set {"Tokyo","Shinagawa","Shinjuku","Kyoto"} in
  vFare.Calculate_fare({s1,s2})
  ;
end
UseFare2
.....................................................................
```

5 Reference、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.