VDM Tools Introduction

Aims

- Syntax checking, type checking and editing a model
- Revise previous lab questions about operations (Explicit/Implicit)
- Introduce Functions (Explicit/Implicit), Mapping, Composite Type
- Using the interpreter to test your model

1. Case Study: Student Course Registration

Part 1: Write a formal specification using VDM++ syntax to create a course registration system specification.

(Aims: To use Mapping and Composite type)

1. Create a class called "Course registration"
2. Declare data type below:-
    a. "CourseStudent" - map type as "CourseID" to a set of "Student"
    b. "CourseID" - string – Example: "TMX1023"
    c. "Student" - record type – Example: ID, Name ("001","Jason")
3. Declare an instance variable below: -
    a. "courseStudentList" from "CourseStudent" type and initialize with empty value
4. Write a following operations:-
    a. create a operation called "getStudent(id,name)" and return a Student type value
        - to create and return a student record based on parameter "id" and "name"
    b. create a operation called "registerCourse(courseID,studentList)"
        - add courseID and studentList into "courseStudentList" variable

          Hint : use one of map operator to join the data
    c. create a operation called "getRegisteredStudentList(courseID)" and return set of set of Student
        - return set of student taking the course

          Hint: can use operator "rng" and "<:"

          Notes: "return set of set of Student" uses the operator “rng” because “rng” return a result of set type and result is in set type. Therefore return set of set of Student

```
1: class CourseRegistration
2:
3:     types
4:         public CourseID = seq of char;
5:         public  Student:: id: seq of char
6:                   name:seq of char;
7:         public CourseStudent=map CourseID to  set of Student ;
8:
9:     instance variables
10:        courseStudentList : CourseStudent:={|->};
11:
12:    operations
13:
14:        public getStudent:seq of char * seq of char==>Student
15:        getStudent(id,name)==(
16:            return mk_Student(id,name);
17:        );
18:
19:        public registerCourse:CourseID * set of Student==> ()
20:        registerCourse(courseID,studentList)==(
21:            courseStudentList :=  courseStudentList munion {courseID  |-> studentList};
22:        );
23:
24:        public getRegisteredStudentList:CourseID==> set of set of Student
25:        getRegisteredStudentList(courseID)==(
26:            return rng ({courseID} <: courseStudentList);
27:        );
28:
29:
30: end CourseRegistration
```

Part 2 Test Model:

By using the specification created in part one, test your model to come out with the following output.

1. Create an object for the class:

   >> create objectName:= new className()

2. Pass parameter of student's named Jason with student id "001":

   Hint: calling operation from Question 4(a).

   >> print.objectName.operationName(parameter1, parameter2);

3. Create list of student attached to course TMK1023. Set of student and id:

   {("001","Jason"),("003","Karen"),("005","Peter")}

   Create another list of student attached to course TMK2223. Set of student and id:

   {("011","Mark"),("074","Jack") }

   Pass parameter of course ID and list of student with student's id.

   Hint: calling operations from Question 4(b) and Question 4(a).

   >> print objectName.registerCourse("courseName",

   {objectName.operation(parameter1,parameter2),

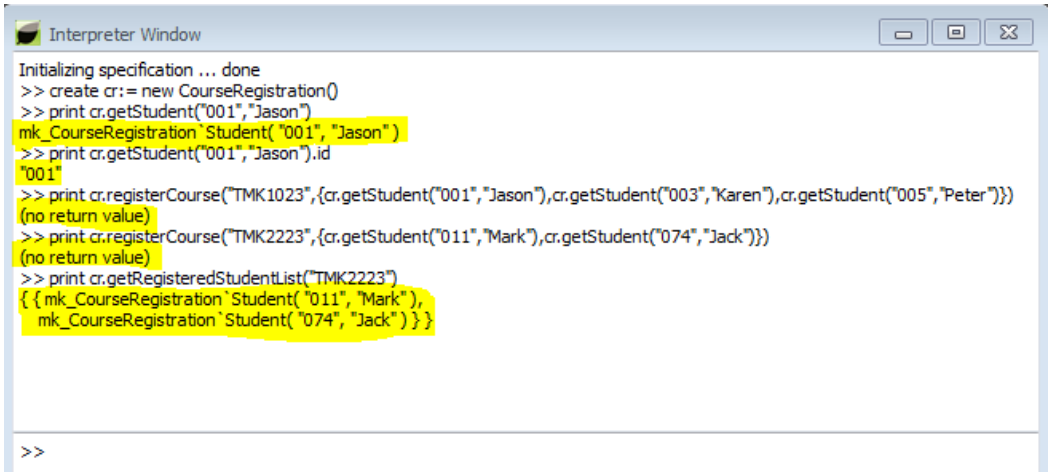
   objectName.operation(parameter1,parameter2)}

4. List out the student's which attached to the course "TMK2223".

   Hint: Pass parameter "TMK2223" into the operation created in Question 4(c)

   >> print objectName.operation2("TMK2223")

   The expected outcome of the above model testing is as below. Refer to the yellow highlighted lines that will be the result you get.



2. Case Study: Cashier machine

Part 1: Write a specification to model the functions of a cashier machine:

(Aims: Learn to write explicit and implicit functions)

```
explicit function definition =
function name, ':', discretionary type, ' -> ',
discretionary type,
function name, '(', parameters, ')', '==',
function body,
['pre', expression],
['post', expression]
```

```
function-name : parameter-types -> result-type
      function-name(parameters) ==
        expression
      pre predicate
      post predicate
```

1. Create a class cashRegister
2. Write explicit functions which enable the cash machine to:
    a. Calculate the total price.
    This function calculates the total price of the items bought by the customers.
    The conditions for this function are: the unit price and quantity should be more than 0.
    Hints: 2 parameters - sum(unitPrice, qty)

    b. Calculate tax applied
    This function calculates the amount tax applied out of the total price.
    Hints: 2 parameters - calculateTax(total,taxPercent)

    c. Calculate discounted item
    This function calculates the discounted price of the item bought.
    Hints: 2 parameters: discountedPrice(price, discountPercent).

    d. Calculate discounted item for members
    This function calculates the collection of member point. (Members are eligible to collect a certain percent of point upon their purchases)
    Hints: calculateMemberPoint(total, percent)

```
1: class CashRegister
2:
3: functions
4:
5:  public sum:real*real*real->real
6:  sum(total,unitPrice,qty)== total+(unitPrice*qty)
7:  pre total>=0 and unitPrice>0 and qty >0;
8:
9: public calculateTax:real*real->real
10:  calculateTax(total,taxPercent)== total*(taxPercent/100);
11:
12:  public discountedPrice:real*real-> real
13:  discountedPrice(price , discountPercent) == price * ((100-discountPercent )/ 100);
14:
15:  public calculateMemberPoint:real*real->real
16:  calculateMemberPoint(total,percent)==total * (percent/100);
17:
18: end CashRegister
19:
```

Part 2: Write implicit functions according to part 1 specifications.

```
implicit function definition =
function name, '(', [parameter types], ')',
[result type],
['pre', expression],
'post', expression,
```

```
function-name(parameters) result-name result-type
      pre predicate
      post predicate
```

```
1: class CashRegisterImplicit
2:
3: functions
4:
5:  public sum(total:real,unitPrice:real,qty:real) result:real
6:  pre total>=0 and unitPrice>0 and qty >0
7: post result=total+(unitPrice*qty);
8:
9: public calculateTax(total:real,taxPercent:real) result:real
10: post result=total*(taxPercent/100);
11:
12: public discountedPrice(price:real , discountPercent:real) result:real
13: post result=price * ((100-discountPercent )/ 100);
14:
15: public calculateMemberPoint(total:real,percent:real) result:real
16: post result=total * (percent/100);
17:
18: end CashRegisterImplicit
```

End of Lab 3