Department of Computer Science, CUI Lahore Campus

Formal Methods

By

Farooq Ahmad

Sets in VDM-SL

Declaring set variables in VDM

Declaring sets in VDM-SL

- A type in the formal specification is a set. The **types** clause is the appropriate place to define new types.
- **types**
    - Student
- To indicate a value to be of the set type in VDM-SL, the type constructor **–set** is appended to the type associated with the *elements* of the set.
    - *aNumber*: $\mathbb{N}$
    - *someNumbers*: $\mathbb{N}$-**set**
    - *someOtherNumbers*: $\mathbb{Z}$-**set**

Declaring sets in VDM-SL

```
types
    Day = <MON> | <TUE> | <WED> | <THU> | <FRI> | <SAT> | <SUN>
```

- We might declare an item of data, *importantDays* say, to hold a collection of days as follows:
  - *importantDays*: *Day*-**set**

Defining sets in VDM-SL

- *someNumbers* = {*2, 4, 28, 19, 10*}
- *importantDays* = {FRI, SAT, SUN}

Ordering is not important in sets so,

Above sets could equally be defined as:

- *someNumbers* = {*28, 2, 10, 4, 19*}
- *importantDays* = {SUN, FRI, SAT}

Sub-ranges

A second way of defining a set in VDM-SL is to use **subranges**. This method can be used when a set of continuous integers is required. For example:

- *someRange* = {5,…,15}
- *someRange = {5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}*
- {7,…,6} = {}

FINITE AND INFINITE SETS IN VDM-SL

- When using a type the ‘is of type’ symbol (:) is to be used rather than the ‘is an element of’ symbol used on sets (∈).
- *Finite SET*
  - *smallNumbers* = $\{x \mid x: \mathbb{N} \bullet 1 \leq x \leq 10\}$
- *Infinite SET*
  - *infiniteSet* = $\{x \mid x: \mathbb{Z} \bullet x < 0\}$

SET UNION

- The **union** of two sets, *j* and *k* returns a set that contains all the elements of the set *j and* all the elements of the set *k*. It is denoted by:
  - $j \cup k$

Union

- If j = {<MON>, <TUE>, <WED>, <SUN>}
- And k = {<MON>, <FRI>, <TUE>}
- then $j \cup k$ = {<MON>, <TUE>, <WED>, <SUN>, <FRI>}

SET INTERSECTION

- The intersection of two sets *j* and *k* returns a set that contains all the elements that are common to both *j* and *k*. It is denoted by:
- $j \cap k$

Intersection

- if j = {<MON>, <TUE>, <WED>, <SUN>}
- and k = {<MON>, <FRI>, <TUE>}
- then $j \cap k$ = {<MON>, <TUE>}

SET DIFFERENCE

- The **difference** of *j* and *k* is the set that contains all the elements that belong to *j* but do not belong to *k*. It is denoted by:
  - j \ k

Difference

- if j = {<MON>, <TUE>, <WED>, <SUN>}
- and k = {<MON>, <FRI>, <TUE>}
- then $j \setminus k$ = {<WED>, <SUN>}

Difference on sets

- Symbol for difference is "\"
- Not Allowed
  - {<MON>, <TUE>, <WED>} \ <TUE>
- Allowed
  - {<MON>, <TUE>, <WED>} \ {<TUE>}

Subsets

- Membership (∈) and set non-membership (∉) operators check whether or not a particular *element* is present in a particular set.
- Another set operator that returns a Boolean result is the **subset** operator (⊆).
- Unlike the set membership operators, this operator takes *two* sets. It returns TRUE if *all* the elements in the first set are also elements of the second set and FALSE otherwise.

Subsets

- $\{a,d,e\} \subseteq \{a,b,c,d,e,f\}$
- $\{a,b,c,d,e,f\} \subseteq \{a,d,e\}$
- $\{a,d,e\} \subseteq \{d,a,e\}$
- $\{a,d,e\} \subset \{a,b,c,d,e,f\}$
- $\{a,d,e\} \subset \{d,a,e\}$
- $\{a,d,e\} \not\subset \{a,x,y,k\}$

Subsets

- $\{a,d,e\} \subseteq \{a,b,c,d,e,f\}$ True
- $\{a,b,c,d,e,f\} \subseteq \{a,d,e\}$ False
- $\{a,d,e\} \subseteq \{d,a,e\}$ True
- $\{a,d,e\} \subset \{a,b,c,d,e,f\}$ True
- $\{a,d,e\} \subset \{d,a,e\}$ False
- $\{a,d,e\} \not\subset \{a,x,y,k\}$ True

Cardinality

Cardinality returns the number of items in a set.

e.g.

- **card** {7, 2, 12} = 3
- **card** {4,…,10} = 7
- **card** { } = 0

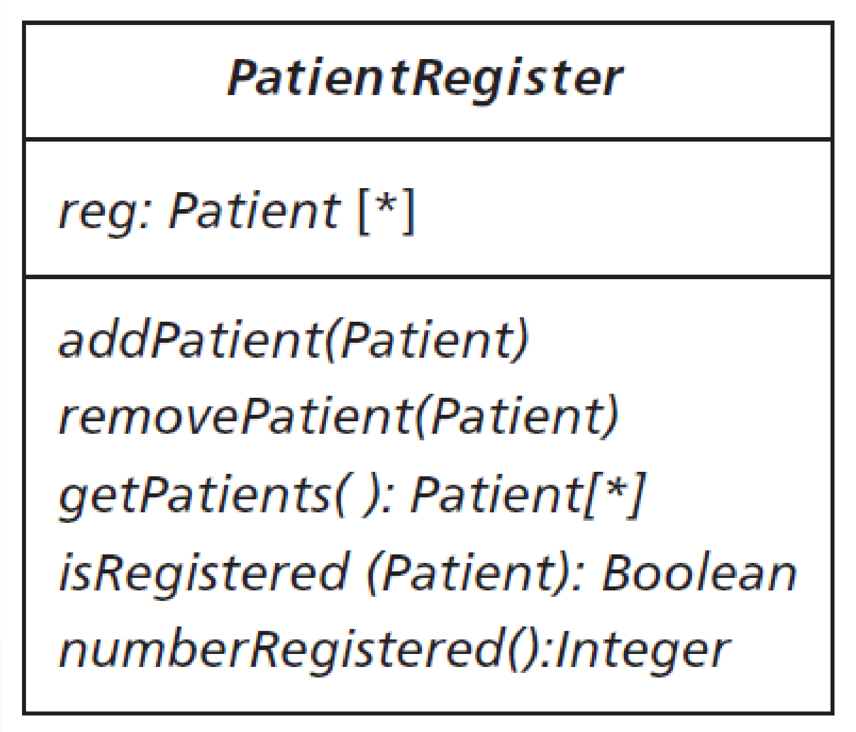
Duplicate items are excluded as:

- **card** {7, 2, 12, 2, 2} = **card** {7, 2, 12} = 3

The Patient Register

- To illustrate the use of sets in a formal specification we will consider a system that registers patients at a doctor's surgery. We will assume that the surgery can deal with a maximum of 200 patients on its register.
- It will be necessary to add and remove patients from the register. As well as this, the register must be able to be interrogated so that the list of patients and the number of patients registered can be returned.
- Also, a check can be made to see if a given patient is registered.

UML Model for Patient Register



Collection elements

- Here we have used the UML collection syntax ([*]), to indicate a collection of values. For example, the type of the *reg* attribute is not a single patient but a collection of zero or more patients.
- ***reg: Patient [*]***
- Similarly, the *getPatients* operation does not return a single patient but many (zero or more) patients:
- ***getPatients( ): Patient [*]***

Modelling the PatientRegister Class in VDM-SL: Types and values

- Types whose internal details are not relevant to the specification can be declared to be TOKEN types in VDM as follows:

types

$$Patient = \text{TOKEN}$$

values

$$LIMIT \colon \mathbb{N} = 200$$

State of the systems

state *PatientRegister* **of**

reg: *Patient*-**set**

inv mk-*PatientRegister* $(r) \underline{\Delta}$ **card** $r \leq LIMIT$

init mk-*PatientRegister* $(r) \underline{\Delta}\ r = \{\ \}$

end

Modelling the *PatientRegister* Class in VDM-SL

types

$\quad Patient = \text{TOKEN}$

values

$\quad LIMIT: \mathbb{N} = 200$

state *PatientRegister* **of**

$\quad reg: Patient\text{-}\mathbf{set}$

inv mk-*PatientRegister* $(r) \triangleq \mathbf{card}\ r \leq LIMIT$

init mk-*PatientRegister* $(r) \triangleq r = \{\ \}$

end

Operation: Add patient

addPatient (*patientIn*: *Patient*)

ext wr *reg*: *Patient*-**set**

pre $patientIn \notin reg \land \textbf{card}\ reg < LIMIT$

post $reg = \overline{reg} \cup \{patientIn\}$

Operation: Remove patient

removePatient (*patientIn*: *Patient*)

ext wr *reg*: *Patient*-**set**

pre $patientIn \in reg$

post $reg = \overline{reg} \setminus \{patientIn\}$

Operation: Get patient

$getPatients\ (\,)\ output: Patient\text{-}\mathbf{set}$

$\mathbf{ext\ rd}\ reg: Patient\text{-}\mathbf{set}$

$\mathbf{pre} \quad \mathrm{TRUE}$

$\mathbf{post} \quad output = reg$

Operation: Query about registration

isRegistered (*patientIn*: *Patient*) *query*: $\mathbb{B}$

ext rd *reg*: *Patient*-**set**

pre TRUE

post *query* $\Leftrightarrow$ *patientIn* $\in$ *reg*

Operation: Query about the number of registered patients

$numberRegistered\;(\,)\;total\colon \mathbb{N}$

$\mathbf{ext\ rd}\; reg\colon Patient\text{-}\mathbf{set}$

$\mathbf{pre}\qquad \mathrm{TRUE}$

$\mathbf{post}\qquad total = \mathbf{card}\; reg$

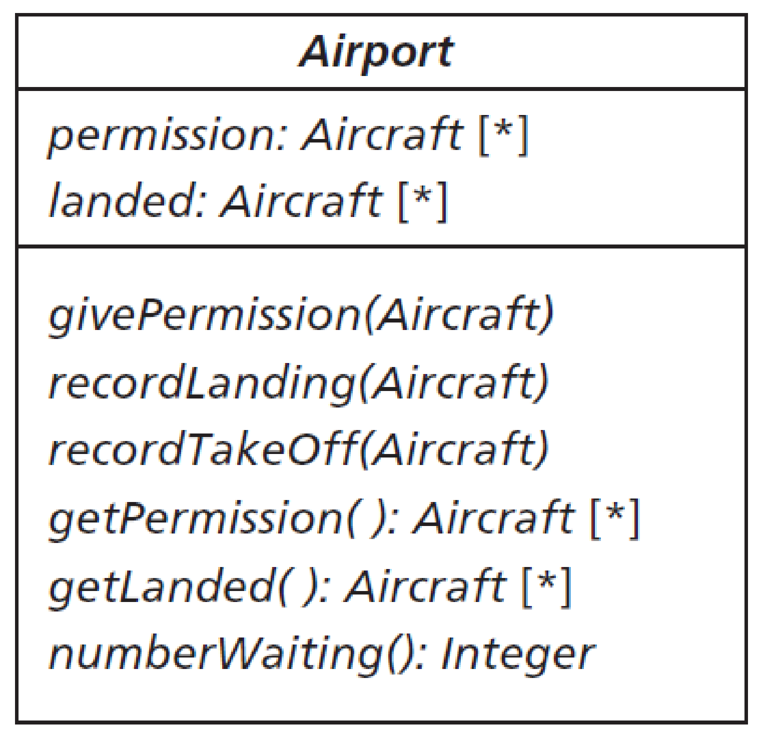
The Airport Class

- A system that keeps track of aircraft that are allowed to land at a particular airport. Aircraft must apply for permission to land at the airport prior to landing. When an aircraft arrives to land at the airport it should only have done so if it had previously been given permission. When an aircraft leaves the airport its permission to land is also removed.

Operations on System

- **givePermission**: records the fact that an aircraft has been granted permission to land at the airport.
- **recordLanding**: records an aircraft as having landed at the airport.
- **recordTakeOff**: records an aircraft as having taken off from the airport.
- **getPermission**: returns the aircrafts currently recorded as having permission to land.
- **getLanded**: returns the aircrafts currently recorded as having landed.
- **numberWaiting**: returns the number of aircrafts granted permission to land but not yet landed.

UML class diagram for Airport class



Exercise: Write the formal specification of the UML class diagram into VDM-SL

Reference and reading material

- Chapter # 5: Sets, of the book “Formal Software Development, from VDM to Java” by Quentin Charatan and Aaron Kans