Modelling Exercise using Mappings
An Electronic Purse

Steve Riddle, John Fitzgerald and Peter Gorm Larsen

March 2010

1 Scenario

You work for a small company that has been awarded a contract to develop the software to manage a new form of "electronic purse" for a bank. The bank will issue all of its customers with smartcards which record a unique identifier and the current balance on the card. Money can be transferred between cards, and some other logging functionality will also be required.

The bank wishes to be sure that **no money can be created** in the system. That is, once the system has been initialised with a collection of cards with a given sum of money of them, it is not possible for more money to enter the system.

You have been given an initial model in VDM++ for the cards (represented by the class `Purse`) and the overall bank system (represented by the class `System`). Your task is to develop the model further by addressing the questions given below.

The initial model is as follows. A purse has a unique identifier (of type `CardId`), and a current balance. The balance and card number are not to be modified directly: accessor methods have been defined to read (`GetBalance, GetCardNo`) and modify (`IncreaseBal, DecreaseBal`) the instance variables. Finally a constructor operation `Purse` initialises a new instance of `Purse`.

```
class Purse
types
public CardId = ???

instance variables
private balance: nat;
private cardNo: CardId;

```

```
operations
public IncreaseBal: nat ==> ()
IncreaseBal(sum)==
balance := balance + sum;

public DecreaseBal: nat ==> ()

pure public GetBalance:() ==> nat

pure public GetCardNo: () ==> CardId

public Purse: CardId * nat ==> Purse
Purse(newId, startbal) ==
( cardNo := newId;
  balance := startbal ) ;

functions
-- no functions currently defined

end Purse
```

The system keeps track of all purses by use of a mapping relating card numbers to purses. Operations are defined to transfer cash between purses and a constructor operation to initialise the system.

```
class System

instance variables

private Purses: map Purse'CardId to Purse;

types
-- no types currently defined

operations

public Transfer: Purse'CardId * Purse'CardId * nat ==> ()
Transfer(fromId, toId, sum) == ???
;

public System: set of Purse ==> System
System(PurseSet) == ???
;
```

```
functions

-- no functions currently defined
end System
```

2 Questions

The first questions require you to complete the initial model. Further questions will deal with extensions to the model. *For all questions, ensure that you consider any preconditions that may be required in the definition of operations and functions. Bear in mind that uniqueness of card identifiers must be maintained, and any use of partial operators must be protected*

Use the file name `PursesSimple.vdmpp` for your model, with the classes as defined above, and complete the model by answering the following questions.

1. The type `CardId` is not currently defined. Complete the type definition by choosing an appropriate abstraction.
2. The operation definition for `IncreaseBal` has been given. Complete the definition for `DecreaseBal` (decrease the balance by the given sum), and accessors `GetBalance` and `GetCardNo`.
3. The `Purses` mapping relates CardIds to Purses, so that a Purse with CardId `c` is tracked in the mapping as `Purses(c)`. Add any invariant that may be necessary to the `Purses` mapping, and complete the constructor defined for `System`: this should build a mapping of CardId to Purse for all members of the given set of purses. *Hint:* Use a map comprehension expression.
4. The effect of the `Transfer` operation is to move a given sum from the purse identified by `fromId` to the purse identified by `toId`. This is to be achieved by using the `IncreaseBal` and `DecreaseBal` operations defined in `Purse`. Complete the operation definition.

At this point you may wish to define a test class, with some test values and execute your model using Overture.

Save the current version of your model in the file `PursesSimple.vdmpp`. Further work on the model should be saved in the file `PursesWithLogging.vdmpp`.

Logging. To keep track of activity in the system a transaction log is required. Each transaction will record the details of a transfer: the cardId debited, the cardId credited, and the amount. The transactions should be logged in the order of occurrence.

5. Add a datatype `Transaction` to the System class to record the details required.

6. Add the transaction log to the internal variables of the System and make any necessary adjustments to the constructor operation. The log should be initially empty for a new system.

7. Add a reporting operation `TotalTransferred` with the following signature:

   ```
   public TotalTransferred:() ==> nat
   ```

   The operation should return the total of all transfers recorded in the log. Complete this operation definition: you may wish to use a recursive auxiliary function to calculate the sum.

8. Now consider the stated requirement of the bank, that no money can be created in the system. How would you model this as a constraint in the system? Extend the System class so that this constraint is recorded, and include a short comment in the model explaining the approach you have used.

   Note: There is no single correct answer to this.