

# 入退出管理システム VDM++ 仕様

佐原 伸

タオベアーズ

## 目次

|      |                        |    |
|------|------------------------|----|
| 1    | 黒板クラス                  | 5  |
| 1.1  | 型                      | 5  |
| 1.2  | インスタンス変数               | 5  |
| 1.3  | 構成子                    | 6  |
| 1.4  | initialize             | 6  |
| 1.5  | 属性の設定                  | 6  |
| 1.6  | 属性の取得                  | 8  |
| 1.7  | 実際と仮想の世界を比較表示する        | 8  |
| 1.8  | 利用者居場所 DB から削除する       | 9  |
| 1.9  | 利用者居場所 DB を比較する        | 10 |
| 1.10 | 実際と仮想の利用者居場所が等しいかを判定する | 11 |
| 1.11 | ドア DB から削除する           | 12 |
| 1.12 | DB から削除する              | 13 |
| 1.13 | 利用者 ID の重複がない          | 13 |
| 1.14 | ID の重複がない              | 13 |
| 2    | 共通定義クラス                | 15 |
| 3    | ドアクラス                  | 16 |
| 4    | 入口ドアクラス                | 18 |
| 5    | 出口ドアクラス                | 20 |
| 6    | 仮想ドアクラス                | 22 |
| 7    | ガードコマンドクラス             | 24 |

|      |                                 |    |
|------|---------------------------------|----|
| 8    | 案内ガードコマンドクラス                    | 27 |
| 8.1  | ガードコマンド_案内の check               | 27 |
| 8.2  | ガードコマンド_案内の update              | 28 |
| 9    | ガードコマンド_ID 端末                   | 30 |
| 10   | ガードコマンド_ID 端末_OK                | 31 |
| 10.1 | ガードコマンド_ID 端末_OK の check        | 31 |
| 10.2 | ガードコマンド_ID 端末_OK の update       | 32 |
| 11   | ガードコマンド_ID 端末_OK_仮想世界           | 34 |
| 11.1 | ガードコマンド_ID 端末_OK_仮想世界の check    | 34 |
| 11.2 | ガードコマンド_ID 端末_OK_仮想世界の update   | 35 |
| 12   | ガードコマンド_ID 端末_NG                | 36 |
| 12.1 | ガードコマンド_ID 端末_NG の check        | 36 |
| 12.2 | ガードコマンド_ID 端末_NG の update       | 37 |
| 13   | ガードコマンド_ID 端末_γ_OK              | 38 |
| 13.1 | ガードコマンド_ID 端末_γ_OK の check      | 38 |
| 13.2 | ガードコマンド_ID 端末_γ_OK の update     | 39 |
| 14   | ガードコマンド_ID 端末_γ_OK_仮想世界         | 41 |
| 14.1 | ガードコマンド_ID 端末_γ_OK_仮想世界の check  | 41 |
| 14.2 | ガードコマンド_ID 端末_γ_OK_仮想世界の update | 42 |
| 15   | ガードコマンド_ID 端末_γ_NG              | 43 |
| 15.1 | ガードコマンド_ID 端末_γ_NG の check      | 43 |
| 15.2 | ガードコマンド_ID 端末_γ_NG の update     | 44 |
| 16   | ガードコマンド_センサークラス                 | 45 |
| 16.1 | ガードコマンド_センサーの check             | 45 |
| 16.2 | ガードコマンド_センサーの update            | 46 |
| 17   | ガードコマンド_センサークラス_仮想世界            | 48 |
| 17.1 | ガードコマンド_センサー_仮想世界の check        | 48 |
| 17.2 | ガードコマンド_センサー_仮想世界の update       | 49 |
| 18   | ガードコマンド_利用者クラス                  | 50 |
| 18.1 | ガードコマンド_利用者の update             | 50 |
| 18.2 | 案内すべき利用者である                     | 50 |
| 19   | ガードコマンド_利用者_認証                  | 52 |
| 19.1 | ガードコマンド_利用者_認証の check           | 52 |

|      |                             |    |
|------|-----------------------------|----|
| 19.2 | ガードコマンド_利用者_認証の update      | 53 |
| 20   | ガードコマンド_利用者_認証_仮想世界         | 55 |
| 20.1 | ガードコマンド_利用者_認証_仮想世界の check  | 55 |
| 20.2 | ガードコマンド_利用者_認証_仮想世界の update | 56 |
| 21   | ガードコマンド_利用者_ドアを開ける          | 58 |
| 21.1 | ガードコマンド_利用者_ドアを開ける、の check  | 58 |
| 21.2 | ガードコマンド_利用者_ドアを開ける、の update | 59 |
| 22   | ガードコマンド_利用者_ドアを閉める          | 61 |
| 22.1 | ガードコマンド_利用者_ドアを閉める、の check  | 61 |
| 22.2 | ガードコマンド_利用者_ドアを閉める、の update | 62 |
| 23   | ガードコマンド_利用者案内有移動クラス         | 64 |
| 23.1 | ガードコマンド_利用者案内有移動の check     | 64 |
| 24   | ガードコマンド_利用者案内無し移動クラス        | 66 |
| 24.1 | ガードコマンド_利用者案内無し移動の check    | 66 |
| 25   | 案内表示クラス                     | 68 |
| 26   | ID カードクラス                   | 69 |
| 27   | ID 端末クラス                    | 70 |
| 28   | 場所グラフクラス                    | 73 |
| 28.1 | インスタンス変数 s 場所 DB            | 73 |
| 28.2 | インスタンス変数 s 移動可能場所集合         | 74 |
| 28.3 | 構成子                         | 74 |
| 28.4 | 場所に関する操作群                   | 75 |
| 29   | オブジェクトクラス                   | 79 |
| 30   | スケジューラクラス                   | 80 |
| 31   | センサークラス                     | 82 |
| 32   | TestApp クラス                 | 84 |
| 33   | TestCaseS クラス               | 85 |
| 34   | TestCaseST0001 クラス          | 88 |
| 35   | TestCaseST0002 クラス          | 89 |

|      |                           |     |
|------|---------------------------|-----|
| 36   | TestCaseST0003 クラス        | 90  |
| 37   | TestCaseST0004 クラス        | 93  |
| 38   | TestCaseST0005 クラス        | 94  |
| 39   | TestCaseST0006 クラス        | 95  |
| 40   | TestCaseST0007 クラス        | 96  |
| 41   | TestCaseST0008 クラス        | 98  |
| 42   | TestCaseST0009 クラス        | 100 |
| 43   | TestCaseST0010 クラス        | 102 |
| 44   | タッチパネルクラス                 | 104 |
| 45   | 利用者クラス                    | 106 |
| 45.1 | インスタンス変数 . . . . .        | 106 |
| 45.2 | 操作 . . . . .              | 106 |
| 45.3 | 移動する気になった時間である . . . . .  | 108 |
| 45.4 | 到着時間が一番早い利用者を選ぶ . . . . . | 109 |
| 46   | 参考文献等                     | 111 |

## 1 黒板クラス

ガードコマンドから参照および更新される属性を持つ。

.....

class

黒板 is subclass of オブジェクト

.....

### 1.1 型

基本的に、属性を写像で持ち、各属性の写像を DB と称する。黒板属性 DB は、属性と DB の対応を表す。

DB には、案内表示 DB、ID 端末 DB、ID カード DB、センサー DB、タッチパネル DB という場所に関連した DB と、実際の利用者居場所 DB、仮想世界利用者居場所 DB、ドア DB がある。

仮想世界利用者居場所 DB は、認証情報とセンサー情報から推定する利用者の居場所を持つ DB である。

.....

types

```
public 黒板属性 DB = map オブジェクト型 to DB;
public 場所関連 DB = map 場所グラフ '場所 ID to 案内表示 |
                        map 場所グラフ '場所 ID to ID 端末 |
                        map ID to ID カード |
                        map 場所グラフ '場所 ID to センサー |
                        map 場所グラフ '場所 ID to タッチパネル;
public DB = map 利用者 to 場所グラフ '場所 ID |
              map ドア 'ドア位置 to ドア 'ドア型 | 場所関連 DB
```

.....

### 1.2 インスタンス変数

s 場所グラフは、利用者が移動する場所を表す、場所グラフを持つ。

s 黒板属性 DB は、すべてのガードコマンドの処理が終わるまで更新されない黒板属性 DB である。各ガードコマンドが、重複して同じ黒板属性 DB を更新した場合は、後の方で実行されたガードコマンドの結果が残る。

s 更新黒板属性 DB は、各ガードコマンドが更新する黒板属性 DB である。

.....

instance variables

```
public s 場所グラフ : 場所グラフ := new 場所グラフ ();
s 黒板属性 DB : 黒板属性 DB;
s 更新黒板属性 DB : 黒板属性 DB := { l-> };
```

.....

### 1.3 構成子

```

.....
operations
public
  黑板 : 黑板属性 DB ==> 黑板
  黑板 (a 黑板属性 DB) ==
    (
      s 黑板属性 DB := a 黑板属性 DB;
      def w オブジェクト型集合 = dom s 黑板属性 DB in
      for all w オブジェクト型 in set w オブジェクト型集合
      do (
        s 更新黑板属性 DB := s 更新黑板属性 DB ++ {w オブジェクト型 |-> { |-> }}
      )
    )
  );
.....

```

### 1.4 initialize

```

.....
public
  initialize : 黑板属性 DB ==> ()
  initialize (a 黑板属性 DB) ==
    (
      s 黑板属性 DB := s 黑板属性 DB ++ a 黑板属性 DB;
      def w オブジェクト型集合 = dom s 黑板属性 DB in
      for all w オブジェクト型 in set w オブジェクト型集合
      do (
        s 更新黑板属性 DB := s 更新黑板属性 DB ++ {w オブジェクト型 |-> { |-> }}
      )
    )
  );
.....

```

### 1.5 属性の設定

```

.....
public
  全属性を設定する : 黑板属性 DB * 黑板属性 DB ==> ()
  全属性を設定する (a 黑板属性 DB, a 更新黑板属性 DB) ==
    (
      s 黑板属性 DB := a 黑板属性 DB;
      s 更新黑板属性 DB := a 更新黑板属性 DB;
      skip
    )
  );
public

```

```

設定する : オブジェクト型 * DB ==> ()
設定する (a オブジェクト型, a 追加 DB) ==
(
  cases a オブジェクト型:
    ( mk_token("実際の利用者居場所") ), ( mk_token("仮想世界利用者居場所") ) +>
      s 黒板属性 DB(a オブジェクト型) := 利用者居場所 DB から削除する (s 黒板属性 DB (a オブジェクト型), a 追加 DB),
    ( mk_token("ドア") ) +>
      s 黒板属性 DB(a オブジェクト型) := ドア DB から削除する (s 黒板属性 DB (a オブジェクト型), a 追加 DB),
    ( mk_token("案内表示") ), ( mk_token("ID 端末") ), ( mk_token("ID カード") ), ( mk_token("タッチパネル") ), ( mk_token("センサー") ) +>
      s 黒板属性 DB(a オブジェクト型) := DB から削除する (s 黒板属性 DB (a オブジェクト型), a 追加 DB),
    others +> s 黒板属性 DB(a オブジェクト型) := (dom a 追加 DB) <-: s 黒板属性 DB (a オブジェクト型)
  end ;
  s 黒板属性 DB(a オブジェクト型) := s 黒板属性 DB (a オブジェクト型) ++ a 追加 DB;
  skip
)
pre let wDB = s 黒板属性 DB (a オブジェクト型) in
  a オブジェクト型 in set dom s 黒板属性 DB and
  cases a オブジェクト型 :
    (mk_token("実際の利用者居場所")), (mk_token("仮想世界利用者居場所")) +>
      利用者 ID の重複がない (wDB),
    others +> ID の重複がない (wDB)
  end
post let wDB = s 黒板属性 DB (a オブジェクト型) in
  cases a オブジェクト型 :
    (mk_token("実際の利用者居場所")), (mk_token("仮想世界利用者居場所")) +>
      利用者 ID の重複がない (wDB),
    others +> ID の重複がない (wDB)
  end ;
public
更新黒板属性を設定する : オブジェクト型 * DB ==> ()
更新黒板属性を設定する (a オブジェクト型, aDB) ==
  s 更新黒板属性 DB(a オブジェクト型) := s 更新黒板属性 DB (a オブジェクト型) ++ aDB
pre a オブジェクト型 in set dom s 更新黒板属性 DB ;
.....

```

## 1.6 属性の取得

```

.....
public
  得る : () ==> 黒板属性 DB * 黒板属性 DB
  得る () ==
    return mk_(s 黒板属性 DB, s 更新黒板属性 DB);
public
  黒板属性を得る : () ==> 黒板属性 DB
  黒板属性を得る () ==
    return s 黒板属性 DB;
public
  更新黒板属性を得る : () ==> 黒板属性 DB
  更新黒板属性を得る () ==
    return s 更新黒板属性 DB;
public
  場所グラフを得る : () ==> 場所グラフ
  場所グラフを得る () ==
    return s 場所グラフ;
.....

```

## 1.7 実際と仮想の世界を比較表示する

実際と仮想の世界を比較表示する。

```

.....
public
  実際と仮想の利用者居場所を比較表示する : (map 利用者 to 場所グラフ '場所 ID) * (map 利用者 to 場所グラフ '場所 ID) ==> ()
  実際と仮想の利用者居場所を比較表示する (a 利用者のいる場所 DB, a 仮想利用者のいる場所 DB) ==
    def mk_(w 実際の利用者居場所にはしか居ない利用者集合, w 仮想世界利用者居場所にはしか居ない利用者集合, w 両方にいる利用者集合) =
      利用者居場所 DB を比較する (a 利用者のいる場所 DB, a 仮想利用者のいる場所 DB);
      - = new IO().echo
        (
          "\n 実際の利用者居場所にはしか居ない利用者集合 = " ^
          VDMUtil.val2seq_of_char[set of 利用者] (w 実際の利用者居場所にはしか居ない利用者集合) ^
          "\n");

```



```

- = new IO ().echo
(
  "\n 仮想世界利用者居場所にしか居ない利用者集合 = " ^
  VDMUtil'val2seq_of_char[set of 利用者] (w 仮想世界利用者居場所にしか居ない利
  用者集合) ^
  "\n");
- = new IO ().echo
(
  "\n 両方にいる利用者集合 = " ^
  VDMUtil'val2seq_of_char[set of 利用者] (w 両方にいる利用者集合) ^
  "\n");
- = new IO ().echo
(
  "\n 実際の利用者のいる場所 DB" ^
  VDMUtil'val2seq_of_char[map 利用者 to 場所グラフ '場所 ID] (a 利用者のいる場
  所 DB) ^
  "\n");
- = new IO ().echo
(
  "\n 仮想利用者のいる場所 DB" ^
  VDMUtil'val2seq_of_char[map 利用者 to 場所グラフ '場所 ID] (a 仮想利用者のい
  る場所 DB) ^
  "\n") in
  skip

```

.....

## 1.8 利用者居場所 DB から削除する

a 削除 DB 中の利用者と同一利用者 ID を持つ利用者を、a 利用者のいる場所 DB から削除する。

```

functions
public

```

利用者居場所 DB から削除する : (map 利用者 to 場所グラフ ‘場所 ID’) \* (map 利用者 to 場所グラフ ‘場所 ID’) -> map 利用者 to 場所グラフ ‘場所 ID’

利用者居場所 DB から削除する (a 利用者のいる場所 DB, a 削除 DB) ==

```
let w 利用者集合 = dom a 利用者のいる場所 DB,
    w 利用者 ID 集合 = {w 利用者.ID を得る () | w 利用者 in set w 利用者集合 },
    w 削除利用者集合 = dom a 削除 DB,
    w 削除利用者 ID 集合 = {w 利用者.ID を得る () | w 利用者 in set w 削除利用者集合 },
    w 削除後利用者 ID 集合 = w 利用者 ID 集合 \ w 削除利用者 ID 集合,
    w 削除後利用者集合 = {w 利用者 | w 利用者 in set w 利用者集合 & w 利用者.ID を得る () in set w 削除後利用者 ID 集合 } in
    w 削除後利用者集合 <: a 利用者のいる場所 DB;
```

.....

## 1.9 利用者居場所 DB を比較する

実際と仮想の利用者居場所 DB を比較し、実際の利用者居場所にしか居ない利用者集合、仮想世界利用者居場所にしか居ない利用者集合、両方にいる利用者集合の 3 つ組みを返す。

.....

```
public
```

利用者居場所 DB を比較する : (map 利用者 to 場所グラフ ‘場所 ID’) \* (map 利用者 to 場所グラフ ‘場所 ID’) ->

set of 利用者 \* set of 利用者 \* set of 利用者

利用者居場所 DB を比較する (a 実際の利用者居場所 DB, a 仮想世界利用者居場所 DB) ==

```
let w 実際の利用者集合 = dom a 実際の利用者居場所 DB,
    w 実際の利用者 ID 集合 = {w 利用者.ID を得る () | w 利用者 in set w 実際の利用者集合 },
    w 仮想世界利用者集合 = dom a 仮想世界利用者居場所 DB,
    w 仮想世界利用者 ID 集合 = {w 利用者.ID を得る () | w 利用者 in set w 仮想世界利用者集合 },
    w 実際の利用者居場所にしか居ない利用者 ID 集合 = w 実際の利用者 ID 集合 \ w 仮想世界利用者 ID 集合,
    w 実際の利用者居場所にしか居ない利用者集合 =
        {w 利用者 | w 利用者 in set w 実際の利用者集合 & w 利用者.ID を得る () in set w 実際
        の利用者居場所にしか居ない利用者 ID 集合 },
    w 仮想世界利用者居場所にしか居ない利用者 ID 集合 = w 仮想世界利用者 ID 集合 \ w 実際の利用
    者 ID 集合,
    w 仮想世界利用者居場所にしか居ない利用者集合 =
        {w 利用者 | w 利用者 in set w 仮想世界利用者集合 & w 利用者.ID を得る () in set w 仮
        想世界利用者居場所にしか居ない利用者 ID 集合 },
    w 両方にいる利用者 ID 集合 = w 仮想世界利用者 ID 集合 inter w 実際の利用者 ID 集合,
    w 両方にいる利用者集合 =
        {w 利用者 | w 利用者 in set w 実際の利用者集合 & w 利用者.ID を得る () in set w 両方
        にいる利用者 ID 集合 } in
    mk_ (w 実際の利用者居場所にしか居ない利用者集合, w 仮想世界利用者居場所にしか居ない利用者集合
    , w 両方にいる利用者集合);
```

## 1.10 実際と仮想の利用者居場所が等しいかを判定する

実際と仮想の利用者居場所が等しいかを判定する。

public

実際と仮想の利用者居場所が等しい : (map 利用者 to 場所グラフ ‘場所 ID) \* (map 利用者 to 場所グラフ ‘場所 ID) -> bool

実際と仮想の利用者居場所が等しい (a 実際の利用者居場所 DB, a 仮想世界利用者居場所 DB) ==

```
let w 実際の利用者集合 = dom a 実際の利用者居場所 DB,
    w 実際の利用者 ID 集合 = {w 利用者.ID を得る () | w 利用者 in set w 実際の利用者集合 },
    w 仮想世界利用者集合 = dom a 仮想世界利用者居場所 DB,
    w 仮想世界利用者 ID 集合 = {w 利用者.ID を得る () | w 利用者 in set w 仮想世界利用者集合 },
    w 実際の利用者居場所には居ない利用者 ID 集合 = w 実際の利用者 ID 集合 \ w 仮想世界利用者 ID 集合,
    w 仮想世界利用者居場所には居ない利用者 ID 集合 = w 仮想世界利用者 ID 集合 \ w 実際の利用者 ID 集合 in
    if w 実際の利用者居場所には居ない利用者 ID 集合 = {} and w 仮想世界利用者居場所には居ない利用者 ID 集合 = {}
    then ( forall w 実際の利用者 in set w 実際の利用者集合 &
        let w 実際の利用者 ID = w 実際の利用者.ID を得る (),
            {w 同じ ID を持つ仮想世界利用者 } =
                {w 仮想世界利用者 | w 仮想世界利用者 in set w 仮想世界利用者集合 & w 実際
                の利用者 ID = w 仮想世界利用者.ID を得る ()} in
            w 実際の利用者. 属性を得る () = w 同じ ID を持つ仮想世界利用者. 属性を得る () )
    else false;
```

.....

## 1.11 ドア DB から削除する

ドア DB から、a 削除 DB に含まれるドアの ID を持つドアを削除する。

public

ドア DB から削除する : (map ドア ‘ドア位置 to ドア ‘ドア型) \* (map ドア ‘ドア位置 to ドア ‘ドア型) -> map ドア ‘ドア位置 to ドア ‘ドア型

ドア DB から削除する (a ドア DB, a 削除 DB) ==

```
if a ドア DB <> { |-> }
then let w ドア集合 = rng a ドア DB,
    w ドア ID 集合 = {w ドア.ID を得る () | w ドア in set w ドア集合 },
    w 削除ドア集合 = rng a 削除 DB,
    w 削除ドア ID 集合 = {w ドア.ID を得る () | w ドア in set w 削除ドア集合 },
    w 削除後ドア ID 集合 = w ドア ID 集合 \ w 削除ドア ID 集合,
    w 削除後ドア集合 = {w ドア | w ドア in set w ドア集合 & w ドア.ID を得る () in set w 削除後ドア ID 集合 } in
    a ドア DB :> w 削除後ドア集合
else a ドア DB;
```

## 1.12 DB から削除する

DB から、a 削除 DB に含まれる ID を持つオブジェクトを削除する。

.....

public

DB から削除する : 場所関連 DB \* 場所関連 DB -> 場所関連 DB

DB から削除する (aDB, a 削除 DB) ==

```
if aDB <> { |-> }
then let w 集合 = rng aDB,
      wID 集合 = {w.ID を得る () | w in set w 集合},
      w 削除集合 = rng a 削除 DB,
      w 削除 ID 集合 = {w.ID を得る () | w in set w 削除集合},
      w 削除後 ID 集合 = wID 集合 \ w 削除 ID 集合,
      w 削除後集合 = {w | w in set w 集合 & w.ID を得る () in set w 削除後 ID 集合} in
      aDB :> w 削除後集合
else aDB;
```

.....

## 1.13 利用者 ID の重複がない

.....

public

利用者 ID の重複がない : DB -> bool

利用者 ID の重複がない (aDB) ==

```
(let w 集合 = dom aDB,
  wID 集合 = {e.ID を得る () | e in set w 集合} in
  card w 集合 = card wID 集合);
```

.....

## 1.14 ID の重複がない

.....

public

ID の重複がない : DB -> bool

ID の重複がない (aDB) ==

```
(let w 集合 = rng aDB,
  wID 集合 = {e.ID を得る () | e in set w 集合} in
  card w 集合 = card wID 集合)
```

.....

.....

end

黒板

.....

Test Suite : vdm.tc

Class : 黒板

| Name                    | #Calls | Coverage |
|-------------------------|--------|----------|
| 黒板 ‘得る                  | 0      | 0%       |
| 黒板 ‘黒板                  | 0      | 0%       |
| 黒板 ‘設定する                | 0      | 0%       |
| 黒板 ‘黒板属性を得る             | 0      | 0%       |
| 黒板 ‘DB から削除する           | 0      | 0%       |
| 黒板 ‘ID の重複がない           | 0      | 0%       |
| 黒板 ‘全属性を設定する            | 0      | 0%       |
| 黒板 ‘場所グラフを得る            | 0      | 0%       |
| 黒板 ‘更新黒板属性を得る           | 0      | 0%       |
| 黒板 ‘initialize          | 0      | 0%       |
| 黒板 ‘ドア DB から削除する        | 0      | 0%       |
| 黒板 ‘利用者 ID の重複がない       | 0      | 0%       |
| 黒板 ‘更新黒板属性を設定する         | 0      | 0%       |
| 黒板 ‘利用者居場所 DB を比較する     | 0      | 0%       |
| 黒板 ‘利用者居場所 DB から削除する    | 0      | 0%       |
| 黒板 ‘実際と仮想の利用者居場所が等しい    | 0      | 0%       |
| 黒板 ‘実際と仮想の利用者居場所を比較表示する | 0      | 0%       |
| Total Coverage          |        | 0%       |

## 2 共通定義クラス

本モデルで共通に使用する型である、ID と時間、及び、デバッグ用のログを出すか否かを示す静的インスタンス変数 debug を定義する。debug の値が大きい方が、より詳細なログを出力する。

```
.....  
class  
共通定義  
types  
    public ID = nat;  
    public 時間 = int  
instance variables  
    public static debug : nat := 5;  
  
end  
共通定義  
.....
```

### 3 ドアクラス

実際のドアと仮想のドア（実際にはドアはないが、移動する場所の間にあるとみなし、処理を統一化するために存在する）両方を表す。内側から通る時と外側から通る時では、異なるドアとしている。

```

.....
class
  ドア is subclass of オブジェクト
values
public
  タイムアウト時間 = 5
types
  public ドア型 = 仮想ドア | 入口ドア | 出口ドア;
  public ドア開閉状態 = <開> | <閉>;
  public ドア錠状態 = <ロック> | <アンロック>;
  public ドア位置 = 場所グラフ '場所 ID * 場所グラフ '場所 ID
instance variables
  public s ドア位置 : 場所グラフ '場所 ID * 場所グラフ '場所 ID;
  public s ドア開閉状態 : ドア開閉状態 := <開>;
  public s ドア錠状態 : ドア錠状態 := <アンロック>;
  public s ドアが開いた時間 : [時間];

operations
public
  タイムアウトである : () ==> bool
  タイムアウトである () ==
    is subclass responsibility;
public
  通過可能である : () ==> bool
  通過可能である () ==
    is subclass responsibility;
public
  施錠する : () ==> ()
  施錠する () ==
    s ドア錠状態 := <ロック>;
public
  施錠されている : () ==> bool
  施錠されている () ==
    return s ドア錠状態 = <ロック>;
public
```



```

解錠する : () ==> ()
解錠する () ==
    s ドア錠状態 := <アンロック>;
public
解錠されている : () ==> bool
解錠されている () ==
    return s ドア錠状態 = <アンロック>;
public
開ける : () ==> ()
開ける () ==
    s ドア開閉状態 := <開>;
public
閉める : () ==> ()
閉める () ==
    is subclass responsibility;
public
開いている : () ==> bool
開いている () ==
    return s ドア開閉状態 = <開>;
public
閉まっている : () ==> bool
閉まっている () ==
    return s ドア開閉状態 = <閉>;
public
位置を得る : () ==> 場所グラフ '場所 ID * 場所グラフ '場所 ID
位置を得る () ==
    return s ドア位置;
public
ID を得る : () ==> 場所グラフ '場所 ID * 場所グラフ '場所 ID
ID を得る () ==
    return 位置を得る ()
end
ドア
.....

```

## 4 入口ドアクラス

実際のドアの、外側から入ることに関する部分を表す。

```

.....
class
入口ドア is subclass of ドア
operations
public
  入口ドア : ( 場所グラフ ‘場所 ID * 場所グラフ ‘場所 ID) * ドア錠状態 * ドア開閉状態 ==> 入口
  ドア
  入口ドア (a ドア位置, a ドア錠状態, a ドア開閉状態) ==
    (
      s ドア位置 := a ドア位置;
      s ドア錠状態 := a ドア錠状態;
      s ドア開閉状態 := a ドア開閉状態;
      if s ドア開閉状態 = <開>
      then s ドアが開いた時間 := s 現在時間
      else s ドアが開いた時間 := nil
    );
public
  入口ドア : ( 場所グラフ ‘場所 ID * 場所グラフ ‘場所 ID) * ドア錠状態 * ドア開閉状態 * [時間
] ==> 入口ドア
  入口ドア (a ドア位置, a ドア錠状態, a ドア開閉状態, a ドアが開いた時間) ==
    (
      s ドア位置 := a ドア位置;
      s ドア錠状態 := a ドア錠状態;
      s ドア開閉状態 := a ドア開閉状態;
      s ドアが開いた時間 := a ドアが開いた時間
    );
public
  コピーする : ( 場所グラフ ‘場所 ID * 場所グラフ ‘場所 ID) ==> 入口ドア
  コピーする (a ドア位置) ==
    (
      return new 入口ドア (a ドア位置, s ドア錠状態, s ドア開閉状態, s ドアが開いた時間)
    );
public
  タイムアウトである : () ==> bool
  タイムアウトである () ==
    return s 現在時間 > s ドアが開いた時間 + タイムアウト時間;
public

```

```
通過可能である : () ==> bool
通過可能である () ==
    return s ドア開閉状態 = <開>;
public
    施錠する : () ==> ()
    施錠する () ==
        s ドア錠状態 := <ロック>;
public
    閉める : () ==> ()
    閉める () ==
        s ドア開閉状態 := <閉>
end
入口ドア
.....
```

## 5 出口ドアクラス

実際のドアの内側から出ることに関係する部分を表す。

```

.....
class
  出口ドア is subclass of ドア
  operations
  public
    出口ドア : ( 場所グラフ ‘場所 ID * 場所グラフ ‘場所 ID) * ドア錠状態 * ドア開閉状態 ==> 出口
    ドア
    出口ドア (a ドア位置, a ドア錠状態, a ドア開閉状態) ==
      (
        s ドア位置 := a ドア位置;
        s ドア錠状態 := a ドア錠状態;
        s ドア開閉状態 := a ドア開閉状態;
        if s ドア開閉状態 = <開>
        then s ドアが開いた時間 := s 現在時間
        else s ドアが開いた時間 := nil
      );
  public
    出口ドア : ( 場所グラフ ‘場所 ID * 場所グラフ ‘場所 ID) * ドア錠状態 * ドア開閉状態 * [時間
] ==> 出口ドア
    出口ドア (a ドア位置, a ドア錠状態, a ドア開閉状態, a ドアが開いた時間) ==
      (
        s ドア位置 := a ドア位置;
        s ドア錠状態 := a ドア錠状態;
        s ドア開閉状態 := a ドア開閉状態;
        s ドアが開いた時間 := a ドアが開いた時間
      );
  public
    コピーする : ( 場所グラフ ‘場所 ID * 場所グラフ ‘場所 ID) ==> 出口ドア
    コピーする (a ドア位置) ==
      (
        return new 出口ドア (a ドア位置, s ドア錠状態, s ドア開閉状態, s ドアが開いた時間)
      );
  public
    タイムアウトである : () ==> bool
    タイムアウトである () ==
      return s 現在時間 > s ドアが開いた時間 + タイムアウト時間;
  public

```

```
通過可能である : () ==> bool
通過可能である () ==
    return s ドア開閉状態 = <開>;
public
    施錠する : () ==> ()
    施錠する () ==
        skip;
public
    閉める : () ==> ()
    閉める () ==
        s ドア開閉状態 := <閉>
end
出口ドア
.....
```

## 6 仮想ドアクラス

仮想のドア（実際にはドアはないが、移動する場所の間にあるとみなし、処理を統一化するために存在する）を表す。

内側から通る時と外側から通る時では、異なるドアとしている。

```

.....
class
仮想ドア is subclass of ドア
operations
public
  仮想ドア : ( 場所グラフ '場所 ID * 場所グラフ '場所 ID) ==> 仮想ドア
  仮想ドア (a ドア位置) ==
    (
      s ドア位置 := a ドア位置;
      s ドア錠状態 := <アンロック>;
      s ドア開閉状態 := <開>;
      s ドアが開いた時間 := nil
    );
public
  仮想ドア : ( 場所グラフ '場所 ID * 場所グラフ '場所 ID) * ドア錠状態 * ドア開閉状態 * [時間] ==> 仮想ドア
  仮想ドア (a ドア位置, a ドア錠状態, a ドア開閉状態, a ドアが開いた時間) ==
    (
      s ドア位置 := a ドア位置;
      s ドア錠状態 := a ドア錠状態;
      s ドア開閉状態 := a ドア開閉状態;
      s ドアが開いた時間 := a ドアが開いた時間
    );
public
  コピーする : ( 場所グラフ '場所 ID * 場所グラフ '場所 ID) ==> 仮想ドア
  コピーする (a ドア位置) ==
    (
      return new 仮想ドア (a ドア位置, s ドア錠状態, s ドア開閉状態, s ドアが開いた時間)
    );
public
  タイムアウトである : () ==> bool
  タイムアウトである () ==
    return false;
public
  通過可能である : () ==> bool
  通過可能である () ==
    return true;

```

```
public
  施錠する : () ==> ()
  施錠する () ==
    skip;
```

```
public
  閉める : () ==> ()
  閉める () ==
    skip
```

```
end
```

仮想ドア

.....

## 7 ガードコマンドクラス

ガード条件が満たされた場合に、指定された処理を行うオブジェクトを定義した、ガードコマンドクラスの抽象クラス。

```

.....
class
ガードコマンド is subclass of オブジェクト
instance variables
    protected s 局所黑板 : 黑板 := new 黑板 ();

operations
public
    check : 黑板 ==> ()
    check (-) ==
        is subclass responsibility;
public
    update : 黑板 ==> ()
    update (-) ==
        is subclass responsibility;
.....

```

### 7.0.1 ある場所の利用者を次の場所へ移動させる

利用者を移動させ、移動後、利用者の移動意志を無くす。

```

.....
public
ある場所の利用者を次の場所へ移動させる : 場所グラフ ‘利用者のいる場所 DB * 利用者 * 場所グラフ ‘場所 ID * 場所グラフ ‘場所 ID * seq of char * seq of char * seq of char ==>
    ()

ある場所の利用者を次の場所へ移動させる (a 利用者のいる場所 DB, a 利用者, a 居場所 ID, a 次の場所 ID, a オブジェクト型, a クラス名, a メッセージ) ==
    (    dcl w 更新利用者 : 利用者 := a 利用者. コピーする (a クラス名);
.....

```



```

def a 場所グラフ = s 局所黑板. 場所グラフを得る () in
if a 場所グラフ. 利用者が移動可能である (a 利用者のいる場所 DB, a 次の場所 ID)
then ( w 更新利用者.前にいた場所を設定する (a 居場所 ID);
      s 局所黑板.更新黑板属性を設定する (mk_token (a オブジェクト型), {w 更新利用者. 到着時間を更新する (
の場所 ID});

      def - = debug >= 5 => new IO ().echo
        (
          "\n"^a メッセージ^
          VDMUtil.val2seq_of_char[利用者 * 場所グラフ '場所 ID * 場所グ
ラフ '場所 ID]

          (
            mk_ (w 更新利用者, a 居場所 ID, a 次の場所 ID))^
            "\n") in
          skip
        )
      else def - = debug >= 5 => new IO ().echo
        (
          "\n 利用者が移動できません!!!"^
          VDMUtil.val2seq_of_char[利用者 * 場所グラフ '場所 ID * 場所グラ
フ '場所 ID]

          (
            mk_ (w 更新利用者, a 居場所 ID, a 次の場所 ID))^
            "\n") in
          skip
        )
    )
functions
public
  利用者集合に含まれている : set of 利用者 * 利用者 -> bool
  利用者集合に含まれている (a 利用者集合, a 利用者) ==
    let w 利用者 ID 集合 = {w 利用者.ID を得る () | w 利用者 in set a 利用者集合 },
        w 利用者 ID = a 利用者.ID を得る () in
        w 利用者 ID in set w 利用者 ID 集合
end
ガードコマンド
.....
Test Suite :      vdm.tc
Class :          ガードコマンド

```

| Name                         | #Calls | Coverage  |
|------------------------------|--------|-----------|
| ガードコマンド 'check               | 0      | 0%        |
| ガードコマンド 'update              | 0      | 0%        |
| ガードコマンド '利用者集合に含まれている        | 0      | 0%        |
| ガードコマンド 'ある場所の利用者を次の場所へ移動させる | 0      | 0%        |
| <b>Total Coverage</b>        |        | <b>0%</b> |

## 8 案内ガードコマンドクラス

利用者を案内するアルゴリズムを持つ。

本来は、ID 認証やセンサーから得られる情報だけで構築した、仮想世界の利用者居場所の情報から案内すべきであるが、正しい位置情報を得ることが非常に困難であるため、適切に案内ができないので、実際の利用者居場所の情報から案内を作成している。

```
.....
class
```

```
ガードコマンド_案内 is subclass of ガードコマンド
```

```
instance variables
```

```
public s 場所 ID : 場所グラフ ‘場所 ID;
```

```
operations
```

```
public
```

```
ガードコマンド_案内 : 場所グラフ ‘場所 ID ==> ガードコマンド_案内
```

```
ガードコマンド_案内 (a 場所 ID) ==
```

```
( s 場所 ID := a 場所 ID
```

```
);
.....
```

### 8.1 ガードコマンド\_案内の check

場所に一番早く来た利用者を案内し、移動させる。

移動結果は、局所黒板の更新黒板属性に設定する。

```
.....
public
```

```
check : 黒板 ==> ()
```

```
check (a 大域黒板) ==
```

```
( def mk_ (w 黒板属性 DB, w 更新黒板属性 DB) = a 大域黒板. 得る () in
```

```
s 局所黒板.全属性を設定する (w 黒板属性 DB, w 更新黒板属性 DB);
```

```
def mk_ (w 黒板属性 DB, -) = s 局所黒板. 得る ();
```

```
w 場所グラフ = s 局所黒板.s 場所グラフ;
```

```
w 利用者のいる場所 DB = w 黒板属性 DB (mk_token ("実際の利用者居場所"));
```

```
w 利用者の居場所 ID = s 場所 ID;
```

```
w 利用者集合 = dom (w 利用者のいる場所 DB :> {w 利用者の居場所 ID}) in
```

```
( if w 利用者集合 <> {}
```

```
then def w 選択した一人の利用者 = 利用者 ‘到着時間が一番早い利用者を選ぶ (w 利用者集
```

```
合);
.....
```

```

w 移動場所 ID = w 場所グラフ. 案内可能な次の場所 ID を得る (w 利用者のいる場
所 DB) in
    ( if w 移動場所 ID = nil
      then return
      else ( dcl w 更新案内表示 : 案内表示 := new 案内表示 (w 利用者の居場
所 ID, w 選択した一人の利用者, w 移動場所 ID);
            s 局所黑板.更新黑板属性を設定する (mk.token("案内表示"), {w 利
用者の居場所 ID |-> w 更新案内表示 });
            def - = debug >= 5 => new IO().echo
              (
                "\n 利用者が、案内表示を作成しました。" ^
                VDMUtil.val2seq_of_char[利用者 * 場所グラフ '場
所 ID * 案内表示]
              )
            mk_(w 選択した一人の利用者, w 利用者の居場
所 ID, w 更新案内表示))^
            "\n") in
          skip
        )
      )
    );

```

## 8.2 ガードコマンド\_案内の update

局所黑板の案内表示 DB を大域黑板に設定する。

```

public
update : 黑板 ==> ()
update (a 大域黑板) ==
  def w 更新黑板属性 DB = s 局所黑板. 更新黑板属性を得る ();
  w 更新案内表示 DB = w 更新黑板属性 DB (mk.token("案内表示")) in
  a 大域黑板.設定する (mk.token("案内表示"), w 更新案内表示 DB)
end

```

ガードコマンド\_案内

```

Test Suite :    vdm.tc
Class :        ガードコマンド_案内

```

| Name                   | #Calls | Coverage  |
|------------------------|--------|-----------|
| ガードコマンド_案内 'check      | 0      | 0%        |
| ガードコマンド_案内 'update     | 0      | 0%        |
| ガードコマンド_案内 'ガードコマンド_案内 | 0      | 0%        |
| <b>Total Coverage</b>  |        | <b>0%</b> |

## 9 ガードコマンド\_ID 端末

ID 端末ガードコマンドの抽象クラス。

.....  
`class`

ガードコマンド\_ID 端末 `is subclass of` ガードコマンド

`instance variables`

`public s` 利用者 : 利用者;

`public s`ID 端末 : ID 端末;

`end`

ガードコマンド\_ID 端末

.....  
**Test Suite :** vdm.tc

**Class :** ガードコマンド\_ID 端末

| Name           | #Calls | Coverage  |
|----------------|--------|-----------|
| Total Coverage |        | undefined |

## 10 ガードコマンド\_ID 端末\_OK

ID 端末アルファまたはベータで、利用者の認証が可だった場合のガードコマンド。

```

class
ガードコマンド_ID 端末_OK is subclass of ガードコマンド_ID 端末
operations
public
  ガードコマンド_ID 端末_OK : ID 端末 ==> ガードコマンド_ID 端末_OK
  ガードコマンド_ID 端末_OK (anID 端末) ==
    (  sID 端末 := anID 端末
      );

```

### 10.1 ガードコマンド\_ID 端末\_OK の check

ID 端末が利用不可能で、かつ ID カードが正規の物であるならば、ドアを解錠し、ID 端末利用可能にする。

```

public
check : 黑板 ==> ()
check (a 大域黑板) ==
  (  def mk_ (w 黑板属性 DB, w 更新黑板属性 DB) = a 大域黑板. 得る () in
    s 局所黑板. 全属性を設定する (w 黑板属性 DB, w 更新黑板属性 DB);
    def mk_ (w 黑板属性 DB, -) = s 局所黑板. 得る ();
    w 位置 = sID 端末. 位置を得る ();
    wID 端末 DB = w 黑板属性 DB (mk_token ("ID 端末"));
    wID 端末 = wID 端末 DB (w 位置);
    w 仮想移動する利用者 = wID 端末.s 利用者;
    w ドア DB : map ドア 'ドア位置 to ドア 'ドア型 = w 黑板属性 DB (mk_token ("ドア"));
    {mk_ (f, t)} = {mk_ (f, t) | mk_ (f, t) in set dom w ドア DB & f = w 位置 };
    w 開くべきドア = w ドア DB (mk_ (f, t)) in
  (  if wID 端末. 利用不可能である () and wID 端末. 正当な利用者から認証依頼された () and w 開くべきドア. 施錠さ
    then (  dcl w 更新 ID 端末 : ID 端末 := wID 端末. コピーする (w 位置),
          w 更新開くべきドア : ドア := w 開くべきドア. コピーする (mk_ (f, t));

```

```

w 更新開くべきドア.解錠する ();
w 更新 ID 端末.利用可能にする ();
s 局所黑板.更新黑板属性を設定する (mk_token ("ドア"), {mk_ (f, t) |-> w 更
新開くべきドア });
s 局所黑板.更新黑板属性を設定する (mk_token ("ID 端末"), {w 更新 ID 端末. 位置を得る () |-> w 更
新 ID 端末 });

def - = debug >= 5 => new IO ().echo
(
  "\n 利用者が、解錠しました。"^
  VDMUtil'val2seq_of_char[利用者 * 場所グラフ '場所 ID *
場所グラフ '場所 ID]
  (
    mk_ (w 仮想移動する利用者, f, t))^
  "\n") in
  skip
)
);

```

## 10.2 ガードコマンド\_ID 端末\_OK の update

局所黑板のドア DB と ID 端末 DB とを大域黑板に設定する。

```

public
update : 黑板 ==> ()
update (a 大域黑板) ==
  def w 更新黑板属性 DB = s 局所黑板. 更新黑板属性を得る ();
  w ドア DB = w 更新黑板属性 DB (mk_token ("ドア"));
  wID 端末 DB = w 更新黑板属性 DB (mk_token ("ID 端末")) in
  (
    a 大域黑板.設定する (mk_token ("ドア"), w ドア DB);
    a 大域黑板.設定する (mk_token ("ID 端末"), wID 端末 DB)
  )
end

```

ガードコマンド\_ID 端末\_OK

```

Test Suite :    vdm.tc
Class :        ガードコマンド_ID 端末_OK

```



| Name                              | #Calls | Coverage  |
|-----------------------------------|--------|-----------|
| ガードコマンド_ID 端末_OK'check            | 0      | 0%        |
| ガードコマンド_ID 端末_OK'update           | 0      | 0%        |
| ガードコマンド_ID 端末_OK'ガードコマンド_ID 端末_OK | 0      | 0%        |
| <b>Total Coverage</b>             |        | <b>0%</b> |

## 11 ガードコマンド\_ID 端末\_OK\_仮想世界

ID 端末アルファまたはベータで利用者の認証が可だった場合の仮想世界用ガードコマンド。

```
.....
class
```

```
ガードコマンド_ID 端末_OK_仮想世界 is subclass of ガードコマンド_ID 端末
```

```
operations
```

```
public
```

```
ガードコマンド_ID 端末_OK_仮想世界 : ID 端末 ==> ガードコマンド_ID 端末_OK_仮想世界
```

```
ガードコマンド_ID 端末_OK_仮想世界 (anID 端末) ==
```

```
( sID 端末 := anID 端末
```

```
);
```

### 11.1 ガードコマンド\_ID 端末\_OK\_仮想世界の check

ID 端末が利用不可能で、かつ ID カードが正規の物であるならば、仮想世界で利用者がドアを通過し終わり、次の場所へ移動したと仮定する。

```
.....
public
```

```
check : 黑板 ==> ()
```

```
check (a 大域黑板) ==
```

```
( def mk_ (w 黑板属性 DB, w 更新黑板属性 DB) = a 大域黑板. 得る () in
```

```
s 局所黑板. 全属性を設定する (w 黑板属性 DB, w 更新黑板属性 DB);
```

```
def mk_ (w 黑板属性 DB, -) = s 局所黑板. 得る ();
```

```
w 仮想利用者のいる場所 DB = w 黑板属性 DB (mk_token ("仮想世界利用者居場所"));
```

```
w 位置 = sID 端末. 位置を得る ();
```

```
wID 端末 DB = w 黑板属性 DB (mk_token ("ID 端末"));
```

```
wID 端末 = wID 端末 DB (w 位置);
```

```
w 仮想移動する利用者 = wID 端末.s 利用者;
```

```
w ドア DB : map ドア 'ドア位置 to ドア 'ドア型 = w 黑板属性 DB (mk_token ("ドア"));
```

```
{mk_ (f, t)} = {mk_ (f, t) | mk_ (f, t) in set dom w ドア DB & f = w 位置};
```

```
w 仮想利用者の移動元 ID = f;
```

```
w 仮想利用者の移動先 ID = t;
```

```
w 仮想移動済利用者集合 = dom (w 仮想利用者のいる場所 DB :> {w 仮想利用者の移動  
先 ID});
```

```

w 開くべきドア = w ドア DB (mk_ (w 仮想利用者の移動元 ID, t)) in
( if wID 端末. 利用可能である () and wID 端末. 正当な利用者から認証依頼された () and
  not 利用者集合に含まれている (w 仮想移動済利用者集合, w 仮想移動する利用者) and
  w 開くべきドア. 解錠されている ()
then ( ある場所の利用者を次の場所へ移動させる
      (w 仮想利用者のいる場所 DB, w 仮想移動する利用者, w 仮想利用者の移動
        元 ID, w 仮想利用者の移動先 ID,
        "仮想世界利用者居場所", "ガードコマンド_ID 端末_OK_仮想世界",
        "利用者が認証され、ドアが解錠され利用可能なので、利用者が移動したと
        仮定します。")
      )
    )
);

```

## 11.2 ガードコマンド\_ID 端末\_OK\_仮想世界の update

局所黒板の仮想世界利用者居場所 DB を大域黒板に設定する。

```

public
update : 黒板 ==> ()
update (a 大域黒板) ==
  def w 更新黒板属性 DB = s 局所黒板. 更新黒板属性を得る ();
  w 仮想世界利用者居場所 DB = w 更新黒板属性 DB (mk_token ("仮想世界利用者居場所")) in
  ( a 大域黒板.設定する (mk_token ("仮想世界利用者居場所"), w 仮想世界利用者居場所 DB)
  )
end

```

ガードコマンド\_ID 端末\_OK\_仮想世界

```

Test Suite :      vdm.tc
Class :          ガードコマンド_ID 端末_OK_仮想世界

```

| Name   | #Calls | Coverage |
|--|--------|----------|
| ガードコマンド_ID 端末_OK_仮想世界 'check                 | 0      | 0%       |
| ガードコマンド_ID 端末_OK_仮想世界 'update                | 0      | 0%       |
| ガードコマンド_ID 端末_OK_仮想世界 'ガードコマンド_ID 端末_OK_仮想世界 | 0      | 0%       |
| Total Coverage                               |        | 0%       |

## 12 ガードコマンド\_ID 端末\_NG

ID 端末で利用者の認証が不可だった場合のガードコマンド。

```

class
ガードコマンド_ID 端末_NG is subclass of ガードコマンド_ID 端末
operations
public
  ガードコマンド_ID 端末_NG : ID 端末 ==> ガードコマンド_ID 端末_NG
  ガードコマンド_ID 端末_NG (anID 端末) ==
    (  sID 端末 := anID 端末
      );

```

### 12.1 ガードコマンド\_ID 端末\_NG の check

ID 端末が利用不可能で、かつ ID カードが正規の物でないならば、ID 端末を利用可能にする。それ以外、何もしない。

```

public
check : 黒板 ==> ()
check (a 大域黒板) ==
  (  def mk_ (w 黒板属性 DB, w 更新黒板属性 DB) = a 大域黒板. 得る () in
    s 局所黒板. 全属性を設定する (w 黒板属性 DB, w 更新黒板属性 DB);
    def mk_ (w 黒板属性 DB, -) = s 局所黒板. 得る ();
    w 位置 = sID 端末. 位置を得る ();
    wID 端末 DB = w 黒板属性 DB (mk_token ("ID 端末"));
    wID 端末 = wID 端末 DB (w 位置) in
    if wID 端末. 利用不可能である () and not wID 端末. 正当な利用者から認証依頼された ()
    then (  dcl w 更新 ID 端末 : ID 端末 := wID 端末. コピーする (w 位置);

```

```

w 更新 ID 端末.利用可能にする ();
s 局所黑板.更新黑板属性を設定する (mk_token ("ID 端末"), {w 更新 ID 端末. 位置を得る () |-> w 更
新 ID 端末 });

def - = debug >= 5 => new IO ().echo
(
  "\n 利用者が、認証できませんでした。"~
  VDMUtil'val2seq_of_char[利用者 * ID 端末]
  (
    mk_ (wID 端末.s 利用者,wID 端末))^
  "\n") in
skip
)
);

```

## 12.2 ガードコマンド\_ID 端末\_NG の update

局所黑板の ID 端末 DB を大域黑板に設定する。

```

public
update : 黑板 ==> ()
update (a 大域黑板) ==
  def w 更新黑板属性 DB = s 局所黑板. 更新黑板属性を得る ();
  wID 端末 DB = w 更新黑板属性 DB (mk_token ("ID 端末")) in
  a 大域黑板.設定する (mk_token ("ID 端末"),wID 端末 DB)
end

```

ガードコマンド\_ID 端末\_NG

Test Suite : vdm.tc

Class : ガードコマンド\_ID 端末\_NG

| Name                              | #Calls | Coverage |
|-----------------------------------|--------|----------|
| ガードコマンド_ID 端末_NG'check            | 0      | 0%       |
| ガードコマンド_ID 端末_NG'update           | 0      | 0%       |
| ガードコマンド_ID 端末_NG'ガードコマンド_ID 端末_NG | 0      | 0%       |
| Total Coverage                    |        | 0%       |

## 13 ガードコマンド\_ID 端末 γ\_OK

ID 端末 γ で利用者の認証が可だった場合のガードコマンド。

```

class
ガードコマンド_ID 端末 γ_OK is subclass of ガードコマンド_ID 端末
operations
public
  ガードコマンド_ID 端末 γ_OK : ID 端末 ==> ガードコマンド_ID 端末 γ_OK
  ガードコマンド_ID 端末 γ_OK (anID 端末) ==
    (  sID 端末 := anID 端末
      );

```

### 13.1 ガードコマンド\_ID 端末 γ\_OK の check

ID 端末が利用不可能で、かつ ID カードが正規の物であるならば、タッチパネルを入力可能にし、ID 端末 γ を利用可能にする。

```

public
check : 黒板 ==> ()
check (a 大域黒板) ==
  (  def mk_ (w 黒板属性 DB, w 更新黒板属性 DB) = a 大域黒板. 得る () in
    s 局所黒板. 全属性を設定する (w 黒板属性 DB, w 更新黒板属性 DB);
    def mk_ (w 黒板属性 DB, -) = s 局所黒板. 得る ();
    w 位置 = sID 端末. 位置を得る ();
    wID 端末 DB = w 黒板属性 DB (mk_token ("ID 端末"));
    wID 端末 = wID 端末 DB (w 位置);
    w タッチパネル DB = w 黒板属性 DB (mk_token ("タッチパネル"));
    w タッチパネル : タッチパネル = w タッチパネル DB (wID 端末. 位置を得る ()) in
    if wID 端末. 利用不可能である () and wID 端末. 正当な利用者から認証依頼された ()
    then (  dcl w 更新タッチパネル : タッチパネル := w タッチパネル. コピーする (w 位置),
           w 更新 ID 端末 : ID 端末 := wID 端末. コピーする (w 位置);

```

```

w 更新タッチパネル.入力可能にする ();
w 更新 ID 端末.利用可能にする ();
s 局所黒板.更新黒板属性を設定する (mk_token ("タッチパネル"), {w 更新タッチパネル. 位置を得る () |
新タッチパネル });
s 局所黒板.更新黒板属性を設定する (mk_token ("ID 端末"), {w 更新 ID 端末. 位置を得る () |-> w 更
新 ID 端末 });

def - = debug >= 5 => new IO ().echo
(
  "\n 利用者が、タッチパネルへ入力可能になりました。" ^
  VDMUtil `val2seq_of_char [利用者 * タッチパネル]
  (
    mk_ (wID 端末.s 利用者, w タッチパネル)) ^
  "\n") in
skip
)
);

```

## 13.2 ガードコマンド\_ID 端末 γ\_OK の update

局所黒板のタッチパネル DB と ID 端末 DB とを大域黒板に設定する。

```

public
update : 黒板 ==> ()
update (a 大域黒板) ==
  def w 更新黒板属性 DB = s 局所黒板. 更新黒板属性を得る ();
  w タッチパネル DB = w 更新黒板属性 DB (mk_token ("タッチパネル"));
  wID 端末 DB = w 更新黒板属性 DB (mk_token ("ID 端末")) in
  (
    a 大域黒板.設定する (mk_token ("タッチパネル"), w タッチパネル DB);
    a 大域黒板.設定する (mk_token ("ID 端末"), wID 端末 DB)
  )
end

```

ガードコマンド\_ID 端末 γ\_OK

```

Test Suite :      vdm.tc
Class :          ガードコマンド_ID 端末 γ_OK

```

| Name                      | #Calls | Coverage |
|---------------------------|--------|----------|
| ガードコマンド_ID 端末 γ_OK`check  | 0      | 0%       |
| ガードコマンド_ID 端末 γ_OK`update | 0      | 0%       |

| Name  | #Calls | Coverage |
|---|--------|----------|
| ガードコマンド_ID 端末 $\gamma$ _OK'ガードコマンド_ID 端末 $\gamma$ _OK | 0      | 0%       |
| Total Coverage  |        | 0%       |



## 14 ガードコマンド\_ID 端末 γ\_OK\_仮想世界

ID 端末 γ で利用者の認証が可だった場合の仮想世界用ガードコマンド。

```
.....
class
```

```
ガードコマンド_ID 端末 γ_OK_仮想世界 is subclass of ガードコマンド_ID 端末
```

```
operations
```

```
public
```

```
ガードコマンド_ID 端末 γ_OK_仮想世界 : ID 端末 ==> ガードコマンド_ID 端末 γ_OK_仮想世界
```

```
ガードコマンド_ID 端末 γ_OK_仮想世界 (anID 端末) ==
```

```
(  sID 端末 := anID 端末
  );
.....
```

### 14.1 ガードコマンド\_ID 端末 γ\_OK\_仮想世界の check

ID 端末が利用不可能で、かつ ID カードが正規の物であるならば、仮想世界で利用者がタッチパネルを使用し終わり、出口へ移動したと仮定する。

```
.....
public
```

```
check : 黑板 ==> ()
```

```
check (a 大域黑板) ==
```

```
(  def mk_ (w 黑板属性 DB, w 更新黑板属性 DB) = a 大域黑板. 得る () in
```

```
  s 局所黑板.全属性を設定する (w 黑板属性 DB, w 更新黑板属性 DB);
```

```
  def mk_ (w 黑板属性 DB, -) = s 局所黑板. 得る ();
```

```
    w 仮想利用者のいる場所 DB = w 黑板属性 DB (mk_token ("仮想世界利用者居場所"));
```

```
    w 位置 = sID 端末. 位置を得る ();
```

```
    wID 端末 DB = w 黑板属性 DB (mk_token ("ID 端末"));
```

```
    wID 端末 = wID 端末 DB (w 位置);
```

```
    w 仮想移動する利用者 = wID 端末.s 利用者;
```

```
    w ドア DB : map ドア 'ドア位置 to ドア 'ドア型 = w 黑板属性 DB (mk_token ("ドア"));
```

```
    {mk_ (f, t)} = {mk_ (f, t) | mk_ (f, t) in set dom w ドア DB & f = w 位置};
```

```
    w 仮想利用者の移動元 ID = f;
```

```
    w 仮想利用者の移動先 ID = t;
```

```
    w 仮想移動済利用者集合 = dom (w 仮想利用者のいる場所 DB :> {w 仮想利用者の移動
先 ID});
```

```

    w 開くべきドア = w ドア DB (mk_ (w 仮想利用者の移動元 ID, t)) in
  if wID 端末. 利用可能である () and wID 端末. 正当な利用者から認証依頼された () and
    not 利用者集合に含まれている (w 仮想移動済利用者集合, w 仮想移動する利用者) and
    w 開くべきドア. 解錠されている ()
  then (   ある場所の利用者を次の場所へ移動させる
          (w 仮想利用者のいる場所 DB, w 仮想移動する利用者, w 仮想利用者の移動元 ID, w 仮
            想利用者の移動先 ID,
            "仮想世界利用者居場所", "ガードコマンド_ID 端末 γ_OK_仮想世界",
            "利用者が認証され、タッチパネルが利用可能になったので、利用者がタッチパ
            ネルを使用し終わり、出口へ移動したと仮定します。")
          )
    );

```

.....

## 14.2 ガードコマンド\_ID 端末 γ\_OK\_仮想世界の update

局所黒板の仮想世界利用者居場所 DB を大域黒板に設定する。

```

public
update : 黒板 ==> ()
update (a 大域黒板) ==
  def w 更新黒板属性 DB = s 局所黒板. 更新黒板属性を得る ();
    w 仮想世界利用者居場所 DB = w 更新黒板属性 DB (mk_token ("仮想世界利用者居場所")) in
    (   a 大域黒板.設定する (mk_token ("仮想世界利用者居場所"), w 仮想世界利用者居場所 DB)
    )
end

```

ガードコマンド\_ID 端末 γ\_OK\_仮想世界

```

Test Suite :      vdm.tc
Class :          ガードコマンド_ID 端末 γ_OK_仮想世界

```

| Name   | #Calls | Coverage  |
|--|--------|-----------|
| ガードコマンド_ID 端末 γ_OK_仮想世界 'check                   | 0      | 0%        |
| ガードコマンド_ID 端末 γ_OK_仮想世界 'update                  | 0      | 0%        |
| ガードコマンド_ID 端末 γ_OK_仮想世界 'ガードコマンド_ID 端末 γ_OK_仮想世界 | 0      | 0%        |
| <b>Total Coverage</b>                            |        | <b>0%</b> |

## 15 ガードコマンド\_ID 端末 γ \_NG

ID 端末 γ で利用者の認証が不可だった場合のガードコマンド。

```

class
ガードコマンド_ID 端末 γ _NG is subclass of ガードコマンド_ID 端末
operations
public
  ガードコマンド_ID 端末 γ _NG : ID 端末 ==> ガードコマンド_ID 端末 γ _NG
  ガードコマンド_ID 端末 γ _NG (anID 端末) ==
    (  sID 端末 := anID 端末
      );

```

### 15.1 ガードコマンド\_ID 端末 γ \_NG の check

ID 端末が利用不可能で、かつ ID カードが正規の物でないならば、ID 端末を利用可能にする。それ以外、何もしない。

```

public
check : 黒板 ==> ()
check (a 大域黒板) ==
  (  def mk_ (w 黒板属性 DB, w 更新黒板属性 DB) = a 大域黒板. 得る () in
    s 局所黒板. 全属性を設定する (w 黒板属性 DB, w 更新黒板属性 DB);
    def mk_ (w 黒板属性 DB, -) = s 局所黒板. 得る ();
    w 位置 = sID 端末. 位置を得る ();
    wID 端末 DB = w 黒板属性 DB (mk_token ("ID 端末"));
    wID 端末 = wID 端末 DB (w 位置) in
    if wID 端末. 利用不可能である () and not wID 端末. 正当な利用者から認証依頼された ()
    then (  dcl w 更新 ID 端末 : ID 端末 := wID 端末. コピーする (w 位置);

```

```

w 更新 ID 端末.利用可能にする ();
s 局所黑板.更新黑板属性を設定する (mk_token ("ID 端末"), {w 更新 ID 端末. 位置を得る () |-> w 更
新 ID 端末 });

def - = debug >= 5 => new IO ().echo
(
  "\n 利用者が、認証できませんでした。"~
  VDMUtil.val2seq_of_char[利用者 * ID 端末]
  (
    mk_ (wID 端末.s 利用者,wID 端末))~
  "\n") in
  skip
)
);

```

## 15.2 ガードコマンド\_ID 端末 γ \_NG の update

局所黑板の ID 端末 DB を大域黑板に設定する。

```

public
update : 黑板 ==> ()
update (a 大域黑板) ==
  def w 更新黑板属性 DB = s 局所黑板. 更新黑板属性を得る ();
  wID 端末 DB = w 更新黑板属性 DB (mk_token ("ID 端末")) in
  ( a 大域黑板.設定する (mk_token ("ID 端末"),wID 端末 DB)
  )
end

```

ガードコマンド\_ID 端末 γ \_NG

Test Suite : vdm.tc

Class : ガードコマンド\_ID 端末 γ \_NG

| Name                                    | #Calls | Coverage |
|---|--------|----------|
| ガードコマンド_ID 端末 γ _NG'check               | 0      | 0%       |
| ガードコマンド_ID 端末 γ _NG'update              | 0      | 0%       |
| ガードコマンド_ID 端末 γ _NG'ガードコマンド_ID 端末 γ _NG | 0      | 0%       |
| Total Coverage                          |        | 0%       |

## 16 ガードコマンド\_センサークラス

センサーのガードコマンド。

.....

class

ガードコマンド\_センサー is subclass of ガードコマンド

instance variables

public s 場所 ID : 場所グラフ '場所 ID';

operations

public

ガードコマンド\_センサー : 場所グラフ '場所 ID ==> ガードコマンド\_センサー

ガードコマンド\_センサー (a 場所 ID) ==

( s 場所 ID := a 場所 ID

);

.....

### 16.1 ガードコマンド\_センサーの check

センサーに反応がない場所に居るはずの利用者を、仮想世界利用者居場所 DB から削除する。

.....

public

check : 黑板 ==> ()

check (a 大域黑板) ==

( def mk\_ (w 黑板属性 DB, w 更新黑板属性 DB) = a 大域黑板. 得る () in

s 局所黑板. 全属性を設定する (w 黑板属性 DB, w 更新黑板属性 DB);

def mk\_ (w 黑板属性 DB, -) = s 局所黑板. 得る ();

w 利用者のいる場所 DB = w 黑板属性 DB (mk\_token ("実際の利用者居場所"));

w 利用者の居場所 ID = s 場所 ID;

w センサー DB = w 黑板属性 DB (mk\_token ("センサー"));

w センサー = w センサー DB (w 利用者の居場所 ID);

w 利用者の居場所 ID 集合 = w センサー. 人が居る位置を得る ();

w 利用者集合 = dom (w 利用者のいる場所 DB :-> w 利用者の居場所 ID 集合) in

( decl w 更新センサー : センサー := w センサー. コピーする (w 利用者の居場所 ID);

```

if w 利用者集合 = {}
then ( w 更新センサー.センサー検知状態を設定する (<人は居ない>);
      def - = debug >= 9 => new IO().echo
      (
        "\n センサーに反応がありませんでした。" ^
        VDMUtil'val2seq_of_char[map 利用者 to 場所グラフ '場所 ID]
        (
          w 利用者のいる場所 DB) ^
        "\n") in
      skip
    )
else ( w 更新センサー.センサー検知状態を設定する (<人が居る>);
      def - = debug >= 9 => new IO().echo
      (
        "\n センサーで、利用者を関知しました。" ^
        VDMUtil'val2seq_of_char[map 利用者 to 場所グラフ '場所 ID]
        (
          w 利用者のいる場所 DB) ^
        "\n") in
      skip
    ) ;
s 局所黒板.更新黒板属性を設定する (mk.token ("センサー"), {w 更新センサー.ID を得る () |-> w 更新センサー });
skip
);

```

.....

## 16.2 ガードコマンド\_センサーの update

局所黒板のセンサー DB と仮想世界利用者居場所 DB を大域黒板に設定する。

.....

```

public
update : 黒板 ==> ()
update (a 大域黒板) ==
  def w 更新黒板属性 DB = s 局所黒板.更新黒板属性を得る ();
  w センサー DB = w 更新黒板属性 DB (mk.token ("センサー"));

```

```

w 仮想世界利用者居場所 DB = w 更新黑板属性 DB (mk_token ("仮想世界利用者居場所")) in
(
  a 大域黑板.設定する (mk_token ("センサー"), w センサー DB);
  a 大域黑板.設定する (mk_token ("センサー"), w 仮想世界利用者居場所 DB);
  skip
)
end

```

ガードコマンド\_センサー

.....

**Test Suite :** vdm.tc

**Class :** ガードコマンド\_センサー

| Name                       | #Calls | Coverage  |
|----------------------------|--------|-----------|
| ガードコマンド_センサー 'check        | 0      | 0%        |
| ガードコマンド_センサー 'update       | 0      | 0%        |
| ガードコマンド_センサー 'ガードコマンド_センサー | 0      | 0%        |
| <b>Total Coverage</b>      |        | <b>0%</b> |

## 17 ガードコマンド\_センサークラス\_仮想世界

センサーの仮想世界用ガードコマンド。

.....

```
class
```

ガードコマンド\_センサー\_仮想世界 is subclass of ガードコマンド

instance variables

```
public s 場所 ID : 場所グラフ '場所 ID;
```

operations

```
public
```

ガードコマンド\_センサー\_仮想世界 : 場所グラフ '場所 ID ==> ガードコマンド\_センサー\_仮想世界

ガードコマンド\_センサー\_仮想世界 (a 場所 ID) ==

```
( s 場所 ID := a 場所 ID
);
```

.....

### 17.1 ガードコマンド\_センサー\_仮想世界の check

センサーに反応がない場所に居るはずの利用者を、仮想世界利用者居場所 DB から削除する。

.....

```
public
```

```
check : 黑板 ==> ()
```

```
check (a 大域黑板) ==
```

```
( def mk_ (w 黑板属性 DB, w 更新黑板属性 DB) = a 大域黑板. 得る () in
  s 局所黑板. 全属性を設定する (w 黑板属性 DB, w 更新黑板属性 DB);
  def mk_ (w 黑板属性 DB, -) = s 局所黑板. 得る ();
  w 仮想利用者のいる場所 DB = w 黑板属性 DB (mk_token ("仮想世界利用者居場所"));
  w 利用者の居場所 ID = s 場所 ID;
  w センサー DB = w 黑板属性 DB (mk_token ("センサー"));
  w センサー = w センサー DB (w 利用者の居場所 ID);
  w 利用者の居場所 ID 集合 = w センサー. 人が居る位置を得る ();
  w 仮想利用者集合 = dom (w 仮想利用者のいる場所 DB :=> w 利用者の居場所 ID 集合) in
  if w センサー. 人は居ない () and w 仮想利用者集合 <> {}
  then ( skip
        )
      );
```

.....



## 17.2 ガードコマンド\_センサー\_仮想世界の update

局所黒板のセンサー DB と仮想世界利用者居場所 DB を大域黒板に設定する。

```
.....
public
update : 黒板 ==> ()
update (a 大域黒板) ==
  def w 更新黒板属性 DB = s 局所黒板. 更新黒板属性を得る ();
    w 仮想世界利用者居場所 DB = w 更新黒板属性 DB (mk_token ("仮想世界利用者居場所")) in
  ( a 大域黒板.設定する (mk_token ("仮想世界利用者居場所"), w 仮想世界利用者居場所 DB);
    skip
  )
end
```

ガードコマンド\_センサー\_仮想世界

```
.....
Test Suite :      vdm.tc
Class :          ガードコマンド_センサー_仮想世界
```

| Name                                 | #Calls | Coverage |
|--------------------------------------|--------|----------|
| ガードコマンド_センサー_仮想世界 'check             | 0      | 0%       |
| ガードコマンド_センサー_仮想世界 'update            | 0      | 0%       |
| ガードコマンド_センサー_仮想世界 'ガードコマンド_センサー_仮想世界 | 0      | 0%       |
| Total Coverage                       |        | 0%       |

## 18 ガードコマンド\_利用者クラス

利用者のガードコマンド。

```
.....
class
ガードコマンド_利用者 is subclass of ガードコマンド
instance variables
    public s 場所 ID : 場所グラフ '場所 ID;

operations
public
    ガードコマンド_利用者 : 場所グラフ '場所 ID ==> ガードコマンド_利用者
    ガードコマンド_利用者 (a 場所 ID) ==
        ( s 場所 ID := a 場所 ID
        );
.....
```

### 18.1 ガードコマンド\_利用者の update

局所黑板の実際の利用者居場所 DB と仮想世界利用者居場所 DB を大域黑板に設定する。

```
.....
public
update : 黑板 ==> ()
update (a 大域黑板) ==
    def w 更新黑板属性 DB = s 局所黑板. 更新黑板属性を得る ();
    w 更新利用者のいる場所 DB = w 更新黑板属性 DB (mk_token ("実際の利用者居場所"));
    w 更新仮想利用者のいる場所 DB = w 更新黑板属性 DB (mk_token ("仮想世界利用者居場所")) in
    ( a 大域黑板.設定する (mk_token ("実際の利用者居場所"), w 更新利用者のいる場所 DB);
      a 大域黑板.設定する (mk_token ("仮想世界利用者居場所"), w 更新仮想利用者のいる場所 DB);
      skip
    );
.....
```

### 18.2 案内すべき利用者である

移動する気になった、案内すべき利用者であるか判定する。

```
.....
protected
```

案内すべき利用者である : 利用者 \* set of 利用者 ==> bool

案内すべき利用者である (a 案内する利用者, a 利用者集合) ==

```
(  if 利用者集合に含まれている (a 利用者集合, a 案内する利用者) and a 案内する利用者. 移動する気になった時間である
    then return true
    else return false
)
```

end

ガードコマンド\_利用者

.....

Test Suite : vdm.tc

Class : ガードコマンド\_利用者

| Name                     | #Calls | Coverage |
|--------------------------|--------|----------|
| ガードコマンド_利用者 'update      | 0      | 0%       |
| ガードコマンド_利用者 'ガードコマンド_利用者 | 0      | 0%       |
| ガードコマンド_利用者 '案内すべき利用者である | 0      | 0%       |
| Total Coverage           |        | 0%       |

## 19 ガードコマンド\_利用者\_認証

利用者が認証された場合を記述したガードコマンド。

```

.....
class
ガードコマンド_利用者_認証 is subclass of ガードコマンド_利用者
operations
public
  ガードコマンド_利用者_認証 : 場所グラフ ‘場所 ID ==> ガードコマンド_利用者_認証
  ガードコマンド_利用者_認証 (a 場所 ID) ==
    ( s 場所 ID := a 場所 ID
    );
.....

```

### 19.1 ガードコマンド\_利用者\_認証の check

利用者の認証を ID 端末に依頼する。

認証結果は、局所黒板の更新黒板属性に設定する。

```

.....
public
  check : 黒板 ==> ()
  check (a 大域黒板) ==
    (
      def mk_ (w 黒板属性 DB, w 更新黒板属性 DB) = a 大域黒板. 得る () in
      s 局所黒板.全属性を設定する (w 黒板属性 DB, w 更新黒板属性 DB);
      def mk_ (w 黒板属性 DB, -) = s 局所黒板. 得る ();
      w 場所グラフ = s 局所黒板.s 場所グラフ;
      w 利用者のいる場所 DB = w 黒板属性 DB (mk_token ("実際の利用者居場所"));
      wID 端末 DB : map ID to ID 端末 = w 黒板属性 DB (mk_token ("ID 端末"));
      w 利用者の居場所 ID = s 場所 ID;
      wID 端末 = wID 端末 DB (w 利用者の居場所 ID);
      w 利用者集合 = dom (w 利用者のいる場所 DB := {w 利用者の居場所 ID}) in
      (
        if w 利用者集合 = {}
        then return ;
        def {w 案内する利用者} = w 利用者集合 in
        (
          if w 場所グラフ.ID 端末がある (w 利用者の居場所 ID) and not wID 端末. 既に認証依頼している (w 案内する利用者)
          then (
            decl w 更新 ID 端末 : ID 端末 := wID 端末. コピーする (w 利用者の居場所 ID);

```

```

w 更新 ID 端末. 認証依頼する (w 案内する利用者);
s 局所黒板. 更新黒板属性を設定する (mk_token ("ID 端末"), {w 更新 ID 端末. 位置を得る () | -
新 ID 端末 });

def - = debug >= 5 => new IO ().echo
(
  "\n 利用者が、認証を依頼しました。" ^
  VDMUtil 'val2seq_of_char [利用者 * ID 端末]
  (
    mk_ (w 案内する利用者, wID 端末)) ^
    "\n") in
  skip
)
)
)
)
pre let mk_ (w 黒板属性 DB, -) = a 大域黒板. 得る (),
  w 利用者のいる場所 DB = w 黒板属性 DB (mk_token ("実際の利用者居場所")),
  w 利用者の居場所 ID = s 場所 ID,
  w 利用者集合 = dom (w 利用者のいる場所 DB := {w 利用者の居場所 ID}) in
  card w 利用者集合 in set {0,1};

```

## 19.2 ガードコマンド\_利用者\_認証の update

局所黒板の ID 端末 DB を大域黒板に設定する。

```

public
update : 黒板 ==> ()
update (a 大域黒板) ==
  def w 更新黒板属性 DB = s 局所黒板. 更新黒板属性を得る ();
    w 更新 ID 端末 DB = w 更新黒板属性 DB (mk_token ("ID 端末")) in
  ( a 大域黒板. 設定する (mk_token ("ID 端末"), w 更新 ID 端末 DB)
  )
end

```

ガードコマンド\_利用者\_認証

```

Test Suite :    vdm.tc
Class :        ガードコマンド_利用者_認証

```

| Name                           | #Calls | Coverage  |
|--------------------------------|--------|-----------|
| ガードコマンド_利用者_認証 'check          | 0      | 0%        |
| ガードコマンド_利用者_認証 'update         | 0      | 0%        |
| ガードコマンド_利用者_認証 'ガードコマンド_利用者_認証 | 0      | 0%        |
| <b>Total Coverage</b>          |        | <b>0%</b> |

## 20 ガードコマンド\_利用者\_認証\_仮想世界

利用者が認証された場合を記述した仮想世界用ガードコマンド。

```

.....
class
ガードコマンド_利用者_認証_仮想世界 is subclass of ガードコマンド_利用者
operations
public
  ガードコマンド_利用者_認証_仮想世界 : 場所グラフ '場所 ID ==> ガードコマンド_利用者_認証_仮想
世界
  ガードコマンド_利用者_認証_仮想世界 (a 場所 ID) ==
    ( s 場所 ID := a 場所 ID
    );
.....

```

### 20.1 ガードコマンド\_利用者\_認証\_仮想世界の check

利用者の認証を ID 端末に依頼する。

認証結果は、更新黑板属性の仮想世界利用者居場所 DB に設定する。

```

.....
public
check : 黑板 ==> ()
check (a 大域黑板) ==
  ( def mk_ (w 黑板属性 DB, w 更新黑板属性 DB) = a 大域黑板. 得る () in
    s 局所黑板. 全属性を設定する (w 黑板属性 DB, w 更新黑板属性 DB);
    def mk_ (w 黑板属性 DB, -) = s 局所黑板. 得る ();
    w 場所グラフ = s 局所黑板.s 場所グラフ;
    w 仮想利用者のいる場所 DB = w 黑板属性 DB (mk_token ("仮想世界利用者居場所"));
    w 利用者のいる場所 DB = w 黑板属性 DB (mk_token ("実際の利用者居場所"));
    w 利用者の居場所 ID = s 場所 ID;
    w 利用者集合 = dom (w 利用者のいる場所 DB :> {w 利用者の居場所 ID});
  );
.....

```

```

w 仮想利用者集合 =
    dom (w 仮想利用者のいる場所 DB :> {w 利用者の居場所 ID}) in
(
    if w 利用者集合 = {}
    then return ;
    def {w 利用者} = w 利用者集合 in
    (
        if w 場所グラフ.ID 端末がある (w 利用者の居場所 ID) and not 利用者集合に含まれている (w 仮想利用者集合, w 利用者)
        then (
            s 局所黑板.更新黑板属性を設定する (mk_token ("仮想世界利用者居場所"), {w 利用者 |-> w 利用者の居場所 ID});
            def - = debug >= 5 => new IO ().echo
            (
                "\n 実際の利用者が認証を依頼したので、仮想世界利用者居
                場所に登録します。" ^
                VDMUtil.val2seq_of_char [利用者 * 場所グラフ '場所 ID]
                (
                    mk_ (w 利用者, w 利用者の居場所 ID)) ^
                    "\n") in
                skip
            )
        )
    )
)
pre let mk_ (w 黑板属性 DB, -) = a 大域黑板. 得る (),
    w 利用者のいる場所 DB = w 黑板属性 DB (mk_token ("実際の利用者居場所")),
    w 利用者の居場所 ID = s 場所 ID,
    w 利用者集合 = dom (w 利用者のいる場所 DB :> {w 利用者の居場所 ID}) in
card w 利用者集合 in set {0, 1};

```

## 20.2 ガードコマンド\_利用者\_認証\_仮想世界の update

局所黑板の ID 端末 DB と仮想世界利用者居場所 DB を大域黑板に設定する。

```

public
update : 黑板 ==> ()
update (a 大域黑板) ==
    def w 更新黑板属性 DB = s 局所黑板. 更新黑板属性を得る ();

```



```

    w 更新仮想世界利用者居場所 DB = w 更新黒板属性 DB (mk_token ("仮想世界利用者居場所")) in
    (
      a 大域黒板.設定する (mk_token ("仮想世界利用者居場所"), w 更新仮想世界利用者居場所 DB)
    )
end

```

ガードコマンド\_利用者\_認証\_仮想世界

.....  
**Test Suite :**        vdm.tc  
**Class :**            ガードコマンド\_利用者\_認証\_仮想世界

| Name                                     | #Calls | Coverage |
|--|--------|----------|
| ガードコマンド_利用者_認証_仮想世界 'check               | 0      | 0%       |
| ガードコマンド_利用者_認証_仮想世界 'update              | 0      | 0%       |
| ガードコマンド_利用者_認証_仮想世界 'ガードコマンド_利用者_認証_仮想世界 | 0      | 0%       |
| Total Coverage                           |        | 0%       |

## 21 ガードコマンド\_利用者\_ドアを開ける

利用者がドアを開けるガードコマンド。

```

class
ガードコマンド_利用者_ドアを開ける is subclass of ガードコマンド_利用者
operations
public
  ガードコマンド_利用者_ドアを開ける : 場所グラフ ‘場所 ID ==> ガードコマンド_利用者_ドアを開ける
  ガードコマンド_利用者_ドアを開ける (a 場所 ID) ==
    ( s 場所 ID := a 場所 ID
    );

```

### 21.1 ガードコマンド\_利用者\_ドアを開ける、の check

移動先へのドアが解錠されているならば、ドアを開ける。

ドアを開けた結果は、局所黑板の更新黑板属性に設定する。

```

public
check : 黑板 ==> ()
check (a 大域黑板) ==
  ( def mk_ (w 黑板属性 DB, w 更新黑板属性 DB) = a 大域黑板. 得る () in
    s 局所黑板. 全属性を設定する (w 黑板属性 DB, w 更新黑板属性 DB);
    def mk_ (w 黑板属性 DB, -) = s 局所黑板. 得る ();
    w 場所グラフ = s 局所黑板.s 場所グラフ;
    w 利用者のいる場所 DB = w 黑板属性 DB (mk_token ("実際の利用者居場所"));
    w 利用者の居場所 ID = s 場所 ID;
    w 利用者集合 = dom (w 利用者のいる場所 DB :> {w 利用者の居場所 ID});
    移動可能場所集合 =
      w 場所グラフ. 移動可能場所を得る (s 場所 ID);
    {w 利用者の移動先場所 ID} = 移動可能場所集合;
    w ドア位置 = mk_ (w 利用者の居場所 ID, w 利用者の移動先場所 ID);
    w ドア DB : map ドア ‘ドア位置 to ドア ‘ドア型 = w 黑板属性 DB (mk_token ("ドア"));

```

```

    w 開くべきドア = w ドア DB (w ドア位置) in
  ( if w 利用者集合 = {}
    then return ;
    let {w 利用者} = w 利用者集合 in
    if w 開くべきドア. 解錠されている () and w 開くべきドア. 閉まっている ()
    then ( dcl w 更新開くべきドア : ドア := w 開くべきドア. コピーする (w ドア位置);
           w 更新開くべきドア. 開ける ();
           s 局所黑板. 更新黑板属性を設定する (mk_token ("ドア"), {mk_ (w 利用者の居場
所 ID, w 利用者の移動先場所 ID) |-> w 更新開くべきドア });
           def - = debug >= 5 => new IO ().echo
              (
                "\n 利用者が、ドアを開けました。" ^
                VDMUtil.val2seq_of_char [利用者 * 場所グラフ '場所 ID * 場
所グラフ '場所 ID]
              )
              (
                mk_ (w 利用者, w 利用者の居場所 ID, w 利用者の移動先場
所 ID)) ^
              "\n") in
           skip
        )
    )
  )
pre let mk_ (w 黑板属性 DB, -) = a 大域黑板. 得る (),
    w 場所グラフ = s 局所黑板. s 場所グラフ,
    移動可能場所集合 = w 場所グラフ. 移動可能場所を得る (s 場所 ID),
    w 利用者のいる場所 DB = w 黑板属性 DB (mk_token ("実際の利用者居場所")),
    w 利用者の居場所 ID = s 場所 ID,
    w 利用者集合 = dom (w 利用者のいる場所 DB :- {w 利用者の居場所 ID}) in
card 移動可能場所集合 = 1 and
card w 利用者集合 in set {0, 1};

```

## 21.2 ガードコマンド\_利用者\_ドアを開ける、の update

局所黑板の利用者の居る場所 DB を大域黑板に設定する。

```

public
update : 黑板 ==> ()
update (a 大域黑板) ==
  def w 更新黑板属性 DB = s 局所黑板. 更新黑板属性を得る ();

```

```

    w ドア DB = w 更新黒板属性 DB (mk_token ("ドア")) in
    (
      a 大域黒板.設定する (mk_token ("ドア"), w ドア DB)
    )
end

```

ガードコマンド\_利用者\_ドアを開ける

.....  
**Test Suite :**        vdm.tc  
**Class :**            ガードコマンド\_利用者\_ドアを開ける

| Name                                   | #Calls | Coverage  |
|--|--------|-----------|
| ガードコマンド_利用者_ドアを開ける 'check              | 0      | 0%        |
| ガードコマンド_利用者_ドアを開ける 'update             | 0      | 0%        |
| ガードコマンド_利用者_ドアを開ける 'ガードコマンド_利用者_ドアを開ける | 0      | 0%        |
| <b>Total Coverage</b>                  |        | <b>0%</b> |

## 22 ガードコマンド\_利用者\_ドアを閉める

利用者がドアを閉めるガードコマンド。

```

class
ガードコマンド_利用者_ドアを閉める is subclass of ガードコマンド_利用者
operations
public
  ガードコマンド_利用者_ドアを閉める : 場所グラフ ‘場所 ID ==> ガードコマンド_利用者_ドアを閉める
  ガードコマンド_利用者_ドアを閉める (a 場所 ID) ==
    ( s 場所 ID := a 場所 ID
    );

```

### 22.1 ガードコマンド\_利用者\_ドアを閉める、の check

移動先へのドアが解錠されているならば、ドアを閉める。

ドアを開けた結果と、利用者が前にいた場所を設定した結果は、局所黑板の更新黑板属性に設定する。

```

public
check : 黑板 ==> ()
check (a 大域黑板) ==
  ( def mk_ (w 黑板属性 DB, w 更新黑板属性 DB) = a 大域黑板. 得る () in
    s 局所黑板. 全属性を設定する (w 黑板属性 DB, w 更新黑板属性 DB);
    def mk_ (w 黑板属性 DB, -) = s 局所黑板. 得る ();
    w 利用者のいる場所 DB = w 黑板属性 DB (mk_token ("実際の利用者居場所"));
    w 利用者の居場所 ID = s 場所 ID;
    w 利用者集合 = dom (w 利用者のいる場所 DB :> {w 利用者の居場所 ID}) in
    ( if w 利用者集合 = {}
      then return ;
      def w 利用者 = 利用者 ‘到着時間が一番早い利用者を選ぶ (w 利用者集合) in
      ( if w 利用者. 移動していない ()
        then return ;
        def w 利用者の異動元場所 = w 利用者. 前にいた場所を得る ();

```

```

        w ドア DB : map ドア 'ドア位置 to ドア 'ドア型 = w 黒板属性 DB (mk_token ("ド
ア")) in
        (
            if w 利用者の異動元場所 = w 利用者の居場所 ID
            then return ;
            def w 閉めるべきドア = w ドア DB (mk_ (w 利用者の異動元場所, w 利用者の居場
所 ID)) in
            if w 閉めるべきドア. 開いている ()
            then (
                dcl w 更新閉めるべきドア : ドア := w 閉めるべきドア. コピーする (mk_ (w 利
用者の異動元場所, w 利用者の居場所 ID)),
                w 更新利用者 : 利用者 := w 利用者. コピーする ("ガードコマン
ド_利用者_ドアを閉める");
                w 更新閉めるべきドア. 閉める ();
                w 更新閉めるべきドア. 施錠する ();
                w 更新利用者. 前にいた場所を設定する (nil);
                s 局所黒板. 更新黒板属性を設定する (mk_token ("ドア"), {mk_ (w 利用
者の異動元場所, w 利用者の居場所 ID) |-> w 更新閉めるべきドア });
                s 局所黒板. 更新黒板属性を設定する (mk_token ("実際の利用者居場所
"), {w 更新利用者 |-> w 利用者の居場所 ID});
                def - = debug >= 5 => new IO ().echo
                (
                    "\n 利用者が、ドアを閉めました。" ^
                    VDMUtil'val2seq_of_char [利用者 * 場所グラフ '場
所 ID * 場所グラフ '場所 ID]
                    (
                        mk_ (w 利用者, w 利用者の異動元場所, w 利用者の居
場所 ID)) ^
                        "\n") in
                    skip
                )
            )
        )
    );

```

## 22.2 ガードコマンド\_利用者\_ドアを閉める、の update

局所黒板の利用者の居る場所 DB とドア DB を大域黒板に設定する。

```

public

```

```

update : 黒板 ==> ()
update (a 大域黒板) ==
  def w 更新黒板属性 DB = s 局所黒板. 更新黒板属性を得る ();
    w 更新ドア DB = w 更新黒板属性 DB (mk_token ("ドア"));
    w 更新利用者のいる場所 DB = w 更新黒板属性 DB (mk_token ("実際の利用者居場所")) in
  (
    a 大域黒板.設定する (mk_token ("ドア"),w 更新ドア DB);
    a 大域黒板.設定する (mk_token ("実際の利用者居場所"),w 更新利用者のいる場所 DB);
    skip
  )
end

```

ガードコマンド\_利用者\_ドアを閉める

```

.....
Test Suite :      vdm.tc
Class :          ガードコマンド_利用者_ドアを閉める

```

| Name                                   | #Calls | Coverage  |
|--|--------|-----------|
| ガードコマンド_利用者_ドアを閉める 'check              | 0      | 0%        |
| ガードコマンド_利用者_ドアを閉める 'update             | 0      | 0%        |
| ガードコマンド_利用者_ドアを閉める 'ガードコマンド_利用者_ドアを閉める | 0      | 0%        |
| <b>Total Coverage</b>                  |        | <b>0%</b> |

## 23 ガードコマンド\_利用者案内有移動クラス

利用者が案内されて移動するガードコマンド。

```

.....
class
ガードコマンド_利用者案内有移動 is subclass of ガードコマンド_利用者
operations
public
  ガードコマンド_利用者案内有移動 : 場所グラフ '場所 ID ==> ガードコマンド_利用者案内有移動
  ガードコマンド_利用者案内有移動 (a 場所 ID) ==
    ( s 場所 ID := a 場所 ID
    );
.....

```

### 23.1 ガードコマンド\_利用者案内有移動の check

場所に案内表示装置があれば、案内表示に従って移動する。ある時間、案内表示に従わないこともある。また、すでに移動した利用者の案内表示が残っているので、案内する利用者が適切かどうかチェックしている。移動結果は、局所黒板の更新黒板属性に設定する。

```

.....
public
  check : 黒板 ==> ()
  check (a 大域黒板) ==
    ( def mk_ (w 黒板属性 DB, w 更新黒板属性 DB) = a 大域黒板. 得る () in
      s 局所黒板. 全属性を設定する (w 黒板属性 DB, w 更新黒板属性 DB);
      def mk_ (w 黒板属性 DB, -) = s 局所黒板. 得る ();
      w 場所グラフ = s 局所黒板.s 場所グラフ;
      w 利用者のいる場所 DB = w 黒板属性 DB (mk_token ("実際の利用者居場所"));
      w 利用者の居場所 ID = s 場所 ID;
      w 利用者集合 = dom (w 利用者のいる場所 DB :> {w 利用者の居場所 ID}) in
      ( if w 利用者集合 = {}
        then return ;
        if w 場所グラフ. 案内表示装置がある (w 利用者の居場所 ID)
        then ( def w 案内表示 DB = w 黒板属性 DB (mk_token ("案内表示")) in
              if w 利用者の居場所 ID in set dom w 案内表示 DB
              then def w 案内表示 = w 案内表示 DB (w 利用者の居場所 ID);
                  w 利用者の移動先場所 ID = w 案内表示.s ガイドする場所 ID;
                  w 移動する利用者 = w 案内表示.s 案内する利用者;
                  w ドア位置 = mk_ (w 利用者の居場所 ID, w 利用者の移動先場所 ID);

```



```

        w ドア DB : map ドア ‘ドア位置 to ドア ‘ドア型 = w 黒板属性 DB (mk_token ("
ドア"));

        w 開くべきドア = w ドア DB (w ドア位置) in
        (   if w 開くべきドア. 通過可能である () and
            w 場所グラフ. 利用者が移動可能である (w 利 用 者 の い る 場
所 DB, w 利用者の移動先場所 ID) and
            案内すべき利用者である (w 移動する利用者, w 利用者集合) and
            w 移動する利用者. 移動する気になった時間である ()
        then (   ある場所の利用者を次の場所へ移動させる
                (w 利用者のいる場所 DB, w 移動する利用者, w 利用者
の居場所 ID, w 利用者の移動先場所 ID,
                "実際の利用者居場所", "ガードコマンド_利用者案内
有移動", "利用者が、案内表示に従って移動しました。")
                )
            )
        )
    )
end
ガードコマンド_利用者案内有移動
.....
Test Suite :      vdm.tc
Class :          ガードコマンド_利用者案内有移動

```

| Name                               | #Calls | Coverage |
|------------------------------------|--------|----------|
| ガードコマンド_利用者案内有移動 ‘check            | 0      | 0%       |
| ガードコマンド_利用者案内有移動 ‘ガードコマンド_利用者案内有移動 | 0      | 0%       |
| Total Coverage                     |        | 0%       |

## 24 ガードコマンド\_利用者案内無し移動クラス

利用者が案内されずに移動するガードコマンド。

```

.....
class
ガードコマンド_利用者案内無し移動 is subclass of ガードコマンド_利用者
operations
public
  ガードコマンド_利用者案内無し移動 : 場所グラフ '場所 ID ==> ガードコマンド_利用者案内無し移動
  ガードコマンド_利用者案内無し移動 (a 場所 ID) ==
    ( s 場所 ID := a 場所 ID
    );
.....

```

### 24.1 ガードコマンド\_利用者案内無し移動の check

場所に案内表示装置が無ければ、移動可能な場所の一つに移動する。この場合の、ある時間、移動しないことがあり得る。

移動結果は、局所黑板の更新黑板属性に設定する。

```

.....
public
  check : 黑板 ==> ()
  check (a 大域黑板) ==
    ( def mk_ (w 黑板属性 DB, w 更新黑板属性 DB) = a 大域黑板. 得る () in
      s 局所黑板. 全属性を設定する (w 黑板属性 DB, w 更新黑板属性 DB);
      def mk_ (w 黑板属性 DB, -) = s 局所黑板. 得る ();
      w 場所グラフ = s 局所黑板.s 場所グラフ;
      移動可能場所集合 = w 場所グラフ. 移動可能場所を得る (s 場所 ID);
      w 利用者のいる場所 DB = w 黑板属性 DB (mk_token ("実際の利用者居場所"));
      w 利用者の居場所 ID = s 場所 ID;
      w 利用者集合 = dom (w 利用者のいる場所 DB :> {w 利用者の居場所 ID}) in
      ( if w 利用者集合 = {}
        then return ;
        let w 利用者の移動先場所 ID = w 場所グラフ. 気まぐれに次の移動場所を得る (移動可能場所集合) in
          def w 移動する利用者 = 利用者 '到着時間が一番早い利用者を選ぶ (w 利用者集合);
          w ドア位置 = mk_ (w 利用者の居場所 ID, w 利用者の移動先場所 ID);
          w ドア DB = w 黑板属性 DB (mk_token ("ドア"));

```

```

    w 開くべきドア = w ドア DB (w ドア位置) in
  (
    if not w 場所グラフ.案内表示装置がある (w 利用者の居場所 ID) and
      w 開くべきドア.通過可能である () and
      w 場所グラフ.利用者が移動可能である (w 利用者のいる場所 DB, w 利用者の移動先場
所 ID) and
      w 移動する利用者.移動する気になった時間である ()
    then (
      ある場所の利用者を次の場所へ移動させる
      (w 利用者のいる場所 DB, w 移動する利用者, w 利用者の居場所 ID, w 利用
者の移動先場所 ID,
      "実際の利用者居場所", "ガードコマンド_利用者案内無し移動", "利用
者が、移動しました。")
    )
  )
end

```

ガードコマンド\_利用者案内無し移動

.....  
**Test Suite :** vdm.tc  
**Class :** ガードコマンド\_利用者案内無し移動

| Name                                 | #Calls | Coverage |
|--------------------------------------|--------|----------|
| ガードコマンド_利用者案内無し移動 'check             | 0      | 0%       |
| ガードコマンド_利用者案内無し移動 'ガードコマンド_利用者案内無し移動 | 0      | 0%       |
| Total Coverage                       |        | 0%       |

## 25 案内表示クラス

```

.....
class
案内表示 is subclass of オブジェクト
instance variables
    public s 案内表示 ID : 場所グラフ '場所 ID;
    public s 案内する利用者 : 利用者;
    public s ガイドする場所 ID : 場所グラフ '場所 ID;

operations
public
案内表示 : 場所グラフ '場所 ID * 利用者 * 場所グラフ '場所 ID ==> 案内表示
案内表示 (a 案内表示 ID, a 利用者, a 場所 ID) ==
    (
        s 案内表示 ID := a 案内表示 ID;
        s 案内する利用者 := a 利用者;
        s ガイドする場所 ID := a 場所 ID
    );
public
ID を得る : () ==> 利用者 '利用者 ID
ID を得る () ==
    return s 案内表示 ID
end
案内表示
.....

```

## 26 ID カードクラス

```

.....
class
ID カード is subclass of オブジェクト
types
    public ID カード ID = ID;
    public ID カード状態 = <有効> | <無効>
instance variables
    public sID カード ID : ID カード ID;
    public sID カード状態 : ID カード状態 := <無効>;

operations
public
    ID カード : ID カード ID * ID カード状態 ==> ID カード
    ID カード (aID カード ID, aID カード状態) == atomic
        ( sID カード ID := aID カード ID;
          sID カード状態 := aID カード状態
        );
public
    ID を得る : () ==> ID カード ID
    ID を得る () ==
        return sID カード ID;
public
    有効にする : () ==> ()
    有効にする () ==
        sID カード状態 := <有効>;
public
    有効である : () ==> bool
    有効である () ==
        return sID カード状態 = <有効>;
public
    無効にする : () ==> ()
    無効にする () ==
        sID カード状態 := <無効>
end
ID カード
.....

```

## 27 ID 端末クラス

.....

class

ID 端末 is subclass of オブジェクト

types

public ID 端末位置 = 場所グラフ ‘場所 ID’;  
 public ID 端末状態 = <利用可能> | <利用不可能>;  
 public 端末型 = < $\alpha$ > | < $\beta$ > | < $\gamma$ >

instance variables

public sID 端末位置 : ID 端末位置;  
 public s 端末型 : 端末型;  
 public sID 端末状態 : ID 端末状態 := <利用可能>;  
 public s 利用者 : [利用者] := nil;

operations

public

ID 端末 : ID 端末位置 \* 端末型 ==> ID 端末  
 ID 端末 (aID 端末位置, a 端末型) == atomic  
 ( sID 端末位置 := aID 端末位置;  
   s 端末型 := a 端末型  
 );

public

ID 端末 : ID 端末位置 \* 端末型 \* ID 端末状態 \* [利用者] ==> ID 端末  
 ID 端末 (aID 端末位置, a 端末型, aID 端末状態, a 利用者) == atomic  
 ( sID 端末位置 := aID 端末位置;  
   s 端末型 := a 端末型;  
   sID 端末状態 := aID 端末状態;  
   s 利用者 := a 利用者  
 );

public

コピーする : ID 端末位置 ==> ID 端末  
 コピーする (aID 端末位置) ==  
 ( return new ID 端末 (aID 端末位置, s 端末型, sID 端末状態, s 利用者)  
 );

public

```

認証依頼する : 利用者 ==> ()
認証依頼する (a 利用者) ==
    (  利用不可能にする () ;
      s 利用者 := a 利用者
    ) ;
public
既に認証依頼している : 利用者 ==> bool
既に認証依頼している (a 利用者) ==
    return s 利用者 <> nil and a 利用者.ID を得る () = s 利用者.ID を得る ();
public
認証依頼された : () ==> bool
認証依頼された () ==
    return s 利用者 <> nil;
public
正当な利用者から認証依頼された : () ==> bool
正当な利用者から認証依頼された () ==
    return 認証依頼された () and s 利用者. 有効な ID を持っている ();
public
利用不可能にする : () ==> ()
利用不可能にする () ==
    sID 端末状態 := <利用不可能>;
public
利用可能にする : () ==> ()
利用可能にする () ==
    sID 端末状態 := <利用可能>;
public
利用可能である : () ==> bool
利用可能である () ==
    return sID 端末状態 = <利用可能>;
public
利用不可能である : () ==> bool
利用不可能である () ==
    return sID 端末状態 = <利用不可能>;
public
位置を得る : () ==> ID 端末位置
位置を得る () ==
    return sID 端末位置;
public

```

ID を得る : ( ) ==> ID 端末位置

ID を得る ( ) ==

**return** 位置を得る ( )

**end**

ID 端末

.....



## 28 場所グラフクラス

利用者が居る場所のグラフである。

```

class
場所グラフ is subclass of オブジェクト
values
public
    外部の場所の最大許容人数 = 3;
public
    案内表示を見る場所の最大許容人数 = 2;
public
    ひとりだけ = 1;
public
    案内可能な場所集合 = {4, 7}
types
    public 場所 ID = ID
    inv w 場所 ID == w 場所 ID in set {0,...,9};
    public 場所 = <外部> | <前室> | <閲覧室 A> | <閲覧室 B>;
public
    場所情報 :: 場所 : 場所
                許容人数 : nat;
    public 場所 DB = map 場所 ID to 場所情報;
    public 利用者のいる場所 DB = map 利用者 to 場所 ID

```

### 28.1 インスタンス変数 s 場所 DB

s 場所 DB は、ある場所と対応する場所の情報を持つ。

```

instance variables

```

```
s 場所 DB : 場所 DB := {
    0 |-> mk_場所情報 (<外部>, 外部の場所の最大許容人数),
    1 |-> mk_場所情報 (<外部>, ひとりだけ),
    2 |-> mk_場所情報 (<前室>, 案内表示を見る場所の最大許容人数),
    3 |-> mk_場所情報 (<前室>, ひとりだけ),
    4 |-> mk_場所情報 (<前室>, ひとりだけ),
    5 |-> mk_場所情報 (<閲覧室 A>, ひとりだけ),
    6 |-> mk_場所情報 (<閲覧室 A>, ひとりだけ),
    7 |-> mk_場所情報 (<前室>, ひとりだけ),
    8 |-> mk_場所情報 (<閲覧室 B>, ひとりだけ),
    9 |-> mk_場所情報 (<閲覧室 B>, ひとりだけ)};
```

## 28.2 インスタンス変数 s 移動可能場所集合

s 移動可能場所集合は、ある場所から移動できる先の場所の集合を持つ。

```
public
```

```
s 移動可能場所集合 : map 場所 ID to set of 場所 ID := {
    0 |-> {1},
    1 |-> {2},
    2 |-> {3, 4, 7},
    3 |-> {0},
    4 |-> {5},
    5 |-> {6},
    6 |-> {2},
    7 |-> {8},
    8 |-> {9},
    9 |-> {2}};
```

## 28.3 構成子

```
operations
```

```
public
```

場所グラフ : 場所 DB \* map 場所 ID to set of 場所 ID ==> 場所グラフ

場所グラフ (a 場所 DB, a 移動可能場所集合) == atomic

```
( s 場所 DB := a 場所 DB;
  s 移動可能場所集合 := a 移動可能場所集合
);
```

.....

## 28.4 場所に関する操作群

.....

public

案内可能な次の場所 ID を得る : 利用者のいる場所 DB ==> [場所 ID]

案内可能な次の場所 ID を得る (a 利用者のいる場所 DB) ==

```
( def w 案内可能な場所集合 = 場所グラフ '案内可能な場所集合 in
  return 案内可能な次の場所 ID を得る補助 (a 利用者のいる場所 DB, w 案内可能な場所集合)
);
```

public

案内可能な次の場所 ID を得る補助 : 利用者のいる場所 DB \* set of 場所 ID ==> [場所 ID]

案内可能な次の場所 ID を得る補助 (a 利用者のいる場所 DB, a 場所 ID 集合) ==

```
( if a 場所 ID 集合 = {}
  then return nil ;
  def w 場所 = 気まぐれに次の移動場所を得る (a 場所 ID 集合) in
  if 利用者が移動可能である (a 利用者のいる場所 DB, w 場所)
  then return w 場所
  else return 案内可能な次の場所 ID を得る補助 (a 利用者のいる場所 DB, a 場所 ID 集合
\ {w 場所} )
);
```

public

移動可能な次の場所 ID を得る : 利用者のいる場所 DB \* 場所 ID ==> [場所 ID]

移動可能な次の場所 ID を得る (a 利用者のいる場所 DB, a 居場所 ID) ==

```
( for all w 次の場所 ID 候補 in set s 移動可能場所集合 (a 居場所 ID)
  do if 利用者が移動可能である (a 利用者のいる場所 DB, w 次の場所 ID 候補)
    then return w 次の場所 ID 候補
    else skip;
  return nil
);
```

pre a 居場所 ID in set dom s 移動可能場所集合 ;

public

次の場所 ID を得る : 場所 ID ==> 場所 ID

次の場所 ID を得る (a 場所 ID) ==

```
(  def w 次の場所 ID 候補集合 = s 移動可能場所集合 (a 場所 ID) in
    let s 場所 ID in set w 次の場所 ID 候補集合 in
    return s 場所 ID
)
pre a 場所 ID in set dom s 移動可能場所集合 and
let w 次の場所 ID 候補集合 = s 移動可能場所集合 (a 場所 ID) in
w 次の場所 ID 候補集合 <> {} ;
```

public

利用者のいる場所 ID を得る : 利用者のいる場所 DB \* 利用者 ==> 場所 ID

利用者のいる場所 ID を得る (a 利用者のいる場所 DB, a 利用者) ==

```
(  def w 居場所集合 = rng ({a 利用者 } <: a 利用者のいる場所 DB) in
    let {e} = w 居場所集合 in
    return e
)
pre let w 居場所集合 = rng ({a 利用者 } <: a 利用者のいる場所 DB) in
exists e in set w 居場所集合 & {e} = w 居場所集合 ;
```

public

ある場所の許容人数を得る : 場所 ID ==> nat

ある場所の許容人数を得る (a 場所 ID) ==

```
(  let mk_場所情報 (-, w 許容人数) = s 場所 DB (a 場所 ID) in
    return w 許容人数
) ;
```

public

ある場所の人数を得る : 利用者のいる場所 DB \* 場所 ID ==> nat

ある場所の人数を得る (a 利用者のいる場所 DB, a 場所 ID) ==

```
(  return card ある場所の利用者集合を得る (a 利用者のいる場所 DB, a 場所 ID)
) ;
```

public

ある場所の利用者集合を得る : 利用者のいる場所 DB \* 場所 ID ==> set of 利用者

ある場所の利用者集合を得る (a 利用者のいる場所 DB, a 場所 ID) ==

```
(  return dom (a 利用者のいる場所 DB :> {a 場所 ID})
) ;
```

public

```

移動可能場所を得る : 場所 ID ==> set of 場所 ID
移動可能場所を得る (a 場所 ID) ==
    ( return s 移動可能場所集合 (a 場所 ID)
    )
pre a 場所 ID in set dom s 移動可能場所集合 ;
public
案内表示装置がある : 場所 ID ==> bool
案内表示装置がある (a 場所 ID) ==
    ( return a 場所 ID = 2
    );
public
ID 端末がある : 場所 ID ==> bool
ID 端末がある (a 場所 ID) ==
    ( return a 場所 ID in set {1,4,5,7,8}
    );
public
利用者が移動可能である : 利用者のいる場所 DB * 場所 ID ==> bool
利用者が移動可能である (a 利用者のいる場所 DB, a 次の場所 ID) ==
    def w 許容人数 = ある場所の許容人数を得る (a 次の場所 ID);
    w 人数 = ある場所の人数を得る (a 利用者のいる場所 DB, a 次の場所 ID) in
    if w 人数 < w 許容人数
    then return true
    else return false ;
public static
気まぐれに次の移動場所を得る : set of 場所 ID ==> 場所 ID
気まぐれに次の移動場所を得る (a 場所 ID 集合) ==
    ( def 移動場所候補列 = VDMUtil.set2seq[場所 ID] (a 場所 ID 集合);
      w 乱数 = MATH.rand(card a 場所 ID 集合);
      wVDMJ を考慮した乱数 = if w 乱数 = -1
                              then 0
                              else w 乱数;
      w 添字 = wVDMJ を考慮した乱数 + 1;
    )

```

```

w 移動先 =
    移動場所候補列 (w 添字) in
(
    def -=
        debug >= 9 => new IO().echo
        (
            "\n「気まぐれに次の移動場所を得る」で生成された乱数と rand の引数 = "^
            VDMUtil'val2seq_of_char[int * int] (mk_ (w 乱数, card a 場所 ID 集合
        ))^
            "\n") in
        return w 移動先
    )
)
pre a 場所 ID 集合 <> {}
end
場所グラフ

```

.....

**Test Suite :** vdm.tc

**Class :** 場所グラフ

| Name                      | #Calls | Coverage  |
|---------------------------|--------|-----------|
| 場所グラフ '場所グラフ              | 0      | 0%        |
| 場所グラフ 'ID 端末がある           | 0      | 0%        |
| 場所グラフ '案内表示装置がある          | 0      | 0%        |
| 場所グラフ '次の場所 ID を得る        | 0      | 0%        |
| 場所グラフ '移動可能場所を得る          | 0      | 0%        |
| 場所グラフ 'ある場所の人数を得る         | 0      | 0%        |
| 場所グラフ '利用者が移動可能である        | 0      | 0%        |
| 場所グラフ 'ある場所の許容人数を得る       | 0      | 0%        |
| 場所グラフ 'ある場所の利用者集合を得る      | 0      | 0%        |
| 場所グラフ '利用者のいる場所 ID を得る    | 0      | 0%        |
| 場所グラフ '案内可能な次の場所 ID を得る   | 0      | 0%        |
| 場所グラフ '気まぐれに次の移動場所を得る     | 0      | 0%        |
| 場所グラフ '移動可能な次の場所 ID を得る   | 0      | 0%        |
| 場所グラフ '案内可能な次の場所 ID を得る補助 | 0      | 0%        |
| <b>Total Coverage</b>     |        | <b>0%</b> |

## 29 オブジェクトクラス

.....  
`class`

オブジェクト `is subclass of` 共通定義

`types`

`public` オブジェクト型 = `token`

`instance variables`

`public static` s 現在時間 : 時間 := 0;

`end`

オブジェクト  
.....

## 30 スケジューラクラス

```

.....
class
スケジューラ is subclass of オブジェクト
operations
public
doOneCycle : 黑板 * set of ガードコマンド ==> ()
doOneCycle (a 大域黑板, a ガードコマンド集合) ==
( for all a ガードコマンド in set a ガードコマンド集合
  do ( def -=
        debug >= 8 => new IO().echo
        (
          "\n ガードコマンド check。" ^
          VDMUtil'val2seq_of_char[時間 * ガードコマンド] (mk_(s 現在時間
,a ガードコマンド)) ^
          "\n") in
          a ガードコマンド.check(a 大域黑板)
        ) ;
    for all a ガードコマンド in set a ガードコマンド集合
    do ( def -=
        debug >= 8 => new IO().echo
        (
          "\n ガードコマンド update。" ^
          VDMUtil'val2seq_of_char[時間 * ガードコマンド] (mk_(s 現在時間
,a ガードコマンド)) ^
          "\n") in
          a ガードコマンド.update(a 大域黑板)
        ) ;
    def -=
        debug >= 5 => new IO().echo
        (
          "\n 現在時間 = " ^
          VDMUtil'val2seq_of_char[時間] (s 現在時間) ^
          "のすべてのガードコマンド update を完了しました。" ^
          "\n") in
        skip
    )
end
スケジューラ

```



.....

## 31 センサークラス

```

.....
class
センサー is subclass of オブジェクト
types
    public センサー検知状態 = <人が居る> | <人は居ない>;
    public 人が居る位置 = set of 場所グラフ '場所 ID
instance variables
    public s センサー ID : ID;
    public s 人が居る位置 : 人が居る位置;
    public s センサー検知状態 : センサー検知状態 := <人は居ない>;

operations
public
    センサー : ID * 人が居る位置 * センサー検知状態 ==> センサー
    センサー (anID, a 人が居る位置, a センサー検知状態) == atomic
        ( s センサー ID := anID;
          s 人が居る位置 := a 人が居る位置;
          s センサー検知状態 := a センサー検知状態
        );
public
    コピーする : ID ==> センサー
    コピーする (anID) ==
        return new センサー (anID, s 人が居る位置, s センサー検知状態);
public
    人が居る位置を得る : () ==> 人が居る位置
    人が居る位置を得る () ==
        return s 人が居る位置;
public
    センサー検知状態を設定する : センサー検知状態 ==> ()
    センサー検知状態を設定する (a センサー検知状態) ==
        s センサー検知状態 := a センサー検知状態;
public
    人は居ない : () ==> bool
    人は居ない () ==
        return s センサー検知状態 = <人は居ない>;
public

```

```
ID を得る : () ==> ID
ID を得る () ==
    return s センサー ID
end
センサー
```

.....

## 32 TestApp クラス

```

.....
class
TestApp
operations
public static
    Execute : () ==> ()
    Execute () ==
        (   dcl ts : TestSuite := new TestSuite ();
            ts.AddTest(new TestCaseST0009 ("TestCaseST0009 : 利用者の一人が場所 4 で認証されてい
            ない。"));
            ts.AddTest(new TestCaseST0007 ("TestCaseST0007 : 利用者 3 人が場所 0 から、順次、気ま
            ぐれに移動する"));
            ts.AddTest(new TestCaseST0006 ("TestCaseST0006 : 利用者 1 人が場所 7 から、気まぐれに
            移動する"));
            ts.AddTest(new TestCaseST0005 ("TestCaseST0005 : 利用者 3 人が場所 1, 4, 7 から、それ
            ぞれ認証されて気まぐれに移動する"));
            ts.AddTest(new TestCaseST0004 ("TestCaseST0004 : 案内表示に従って利用者 2 人が、場
            所 2 から 4 または 7 へ移動する"));
            ts.AddTest(new TestCaseST0003 ("TestCaseST0003 : 利用者 3 人が場所 1, 4, 7 から、それ
            ぞれ 2, 5, 8 へ認証されて移動し、ドアを閉める"));
            ts.AddTest(new TestCaseST0002 ("TestCaseST0002 : 利用者が場所 1 から 2 へ認証されて移
            動し、ドアを閉める"));
            ts.AddTest(new TestCaseST0001 ("TestCaseST0001 : 案内無しに利用者が場所 0 から 1 へ移
            動する"));
            ts.Run()
        )
    end
TestApp
.....

```

### 33 TestCaseS クラス

.....  
class

TestCaseS is subclass of TestCase, オブジェクト

instance variables

protected s スケジューラ : スケジューラ := new スケジューラ ();

```

protected s 大域黑板 : 黑板 := new 黑板 ({
    mk_token ("実際の利用者居場所") |-> { |-> },
    mk_token ("仮想世界利用者居場所") |-> { |-> },
    mk_token ("案内表示") |-> { |-> },
    mk_token ("ドア") |->
    {mk_ (0,1) |-> new 仮想ドア (mk_ (0,1)),
     mk_ (1,2) |-> new 入口ドア (mk_ (1,2), <ロック>, <閉>),
     mk_ (2,3) |-> new 仮想ドア (mk_ (2,3)),
     mk_ (2,4) |-> new 仮想ドア (mk_ (2,4)),
     mk_ (2,7) |-> new 仮想ドア (mk_ (2,7)),
     mk_ (3,0) |-> new 出口ドア (mk_ (3,0), <アンロック>, <閉>),
     mk_ (4,5) |-> new 入口ドア (mk_ (4,5), <ロック>, <閉>),
     mk_ (5,6) |-> new 仮想ドア (mk_ (5,6)),
     mk_ (6,2) |-> new 出口ドア (mk_ (6,2), <アンロック>, <閉>),
     mk_ (7,8) |-> new 入口ドア (mk_ (7,8), <ロック>, <閉>),
     mk_ (8,9) |-> new 仮想ドア (mk_ (8,9)),
     mk_ (9,2) |-> new 出口ドア (mk_ (9,2), <アンロック>, <閉>)},
    mk_token ("ID 端末") |->
    {1 |-> new ID 端末 (1, <α>),
     4 |-> new ID 端末 (4, <β>),
     7 |-> new ID 端末 (7, <β>),
     5 |-> new ID 端末 (5, <γ>),
     8 |-> new ID 端末 (8, <γ>)},
    mk_token ("タッチパネル") |->
    {5 |-> new タッチパネル (5, <入力不可能>),
     8 |-> new タッチパネル (8, <入力不可能>)},
    mk_token ("ID カード") |->
    {1 |-> new ID カード (1, <有効>), 2 |-> new ID カード (2, <有効>),
     3 |-> new ID カード (3, <有効>), 4 |-> new ID カード (4, <有効>),
     5 |-> new ID カード (5, <無効>)},
    mk_token ("センサー") |->
    let センサー 2 = new センサー (2, {2,3,4,7}, <人は居ない>),
        センサー 5 = new センサー (5, {5,6}, <人は居ない>),
        センサー 8 = new センサー (8, {8,9}, <人は居ない>) in
    {2 |-> センサー 2, 3 |-> センサー 2, 4 |-> センサー 2, 7 |-> センサ
ー 2,
     5 |-> センサー 5, 6 |-> センサー 5,
     8 |-> センサー 8, 9 |-> センサー 8}});

```

```

protected ガードコマンド集合 : set of ガードコマンド := def mk_(w 黒板属性 DB, -) = s 大域黒板. 得る () in
{
  new ガードコマンド_センサー_仮想世界 (2), new ガードコマンド_センサー_仮想世界 (3),
  new ガードコマンド_センサー_仮想世界 (4), new ガードコマンド_センサー_仮想世界 (5),
  new ガードコマンド_センサー_仮想世界 (6), new ガードコマンド_センサー_仮想世界 (7),
  new ガードコマンド_センサー_仮想世界 (8), new ガードコマンド_センサー_仮想世界 (9),
  new ガードコマンド_センサー (2), new ガードコマンド_センサー (3), new ガードコマンド_センサー (4),
  new ガードコマンド_センサー (5), new ガードコマンド_センサー (6), new ガードコマンド_センサー (7),
  new ガードコマンド_センサー (8), new ガードコマンド_センサー (9),
  new ガードコマンド_利用者案内無し移動 (0), new ガードコマンド_利用者案内無し移動 (1),
  new ガードコマンド_利用者案内無し移動 (3), new ガードコマンド_利用者案内無し移動 (4),
  new ガードコマンド_利用者案内無し移動 (5), new ガードコマンド_利用者案内無し移動 (6),
  new ガードコマンド_利用者案内無し移動 (7), new ガードコマンド_利用者案内無し移動 (8),
  new ガードコマンド_利用者案内無し移動 (9), new ガードコマンド_利用者案内有移動 (2),
  new ガードコマンド_ID_端末_OK (wID_端末 DB (1)), new ガードコマンド_ID_端末_NG (wID_端末 DB (2)),
  new ガードコマンド_ID_端末_OK (wID_端末 DB (4)), new ガードコマンド_ID_端末_NG (wID_端末 DB (5)),
  new ガードコマンド_ID_端末_OK (wID_端末 DB (7)), new ガードコマンド_ID_端末_NG (wID_端末 DB (8)),
  new ガードコマンド_ID_端末_OK_仮想世界 (wID_端末 DB (1)), new ガードコマンド_ID_端末_OK_仮想世界 (wID_端末 DB (7)),
  new ガードコマンド_ID_端末_γ_OK (wID_端末 DB (5)), new ガードコマンド_ID_端末_γ_NG (wID_端末 DB (6)),
  new ガードコマンド_ID_端末_γ_OK (wID_端末 DB (8)), new ガードコマンド_ID_端末_γ_NG (wID_端末 DB (9)),
  new ガードコマンド_ID_端末_γ_OK_仮想世界 (wID_端末 DB (5)), new ガードコマンド_ID_端末_γ_OK_仮想世界 (wID_端末 DB (9)),
  new ガードコマンド_利用者_ドアを開ける (1), new ガードコマンド_利用者_ドアを開ける (3),
  new ガードコマンド_利用者_ドアを開ける (4), new ガードコマンド_利用者_ドアを開ける (6),
  new ガードコマンド_利用者_ドアを開ける (7), new ガードコマンド_利用者_ドアを開ける (9),
  new ガードコマンド_利用者_ドアを閉める (0), new ガードコマンド_利用者_ドアを閉める (2),
  new ガードコマンド_利用者_ドアを閉める (5), new ガードコマンド_利用者_ドアを閉める (8),
  new ガードコマンド_利用者_認証_仮想世界 (1), new ガードコマンド_利用者_認証_仮想世界 (4),
  new ガードコマンド_利用者_認証_仮想世界 (5), new ガードコマンド_利用者_認証_仮想世界 (7),
  new ガードコマンド_利用者_認証_仮想世界 (8),
  new ガードコマンド_利用者_認証 (1), new ガードコマンド_利用者_認証 (4),
  new ガードコマンド_利用者_認証 (5), new ガードコマンド_利用者_認証 (7),
  new ガードコマンド_利用者_認証 (8),
  new ガードコマンド_案内 (2)};

```

operations

protected

```

SetUp : () ==> ()
SetUp () ==
    (  MATH'srand(10);
        skip
    );
protected
TearDown : () ==> ()
TearDown () ==
    skip
end
TestCaseS
.....

```

## 34 TestCaseST0001 クラス

```

.....
class
TestCaseST0001 is subclass of TestCaseS
instance variables
    protected ガードコマンド集合 : set of ガードコマンド := {
        new ガードコマンド_利用者案内無し移動 (0)};

operations
public
    TestCaseST0001 : seq of char ==> TestCaseST0001
    TestCaseST0001 (nm) ==
        TestCase(nm);
protected
    RunTest : () ==> ()
    RunTest () ==
        (  s 現在時間 := 100;
            def wID カード = s 大域黑板. 黑板属性を得る () (mk.token ("ID カード"));
            利用者 1 = new 利用者 (1, wID カード (1), s 現在時間, 10) in
            (  s 大域黑板.initialize({
                mk.token ("実際の利用者居場所") |->
                    { 利用者 1 |-> 0}});
                for i = 125 to 210 by 25

```



```

do ( s スケジューラ.doOneCycle(s 大域黑板, ガードコマンド集合);
    s 現在時間 := i
  );
let mk_ (w 大域黑板属性 DB, -) = s 大域黑板. 得る (),
    w 利用者のいる場所 DB = w 大域黑板属性 DB (mk_token ("実際の利用者居場所")) in
AssertTrue
  (let {w 利用者} = dom w 利用者のいる場所 DB,
    {w 場所} = rng w 利用者のいる場所 DB in
    w 利用者.ID を得る () = 1 and w 場所 = 1 and w 利用者. 前にいた場所を得る () = 0)
)
end
TestCaseST0001
.....

```

## 35 TestCaseST0002 クラス

```

.....
class
TestCaseST0002 is subclass of TestCaseS
instance variables
  protected ガードコマンド集合 : set of ガードコマンド := def mk_ (w 黑板属性 DB, -) = s 大域黑板. 得る () in
    {
      new ガードコマンド_センサー (2),
      new ガードコマンド_利用者_認証 (1),
      new ガードコマンド_利用者_認証_仮想世界 (1),
      new ガードコマンド_利用者_ドアを開ける (1),
      new ガードコマンド_利用者_ドアを閉める (2),
      new ガードコマンド_ID_端末_OK (wID 端末 DB (1)),
      new ガードコマンド_ID_端末_OK_仮想世界 (wID 端末 DB (1)),
      new ガードコマンド_ID_端末_NG (wID 端末 DB (1)),
      new ガードコマンド_利用者案内無し移動 (1)};

operations
public
  TestCaseST0002 : seq of char ==> TestCaseST0002
  TestCaseST0002 (nm) ==
    TestCase(nm);
protected

```

```

RunTest : () ==> ()
RunTest () ==
(
  s 現在時間 := 100;
  def wID カード = s 大域黑板. 黑板属性を得る () (mk.token ("ID カード"));
  利用者 1 = new 利用者 (1, wID カード (1), s 現在時間, 10) in
  (
    s 大域黑板.initialize({
      mk.token ("実際の利用者居場所") |->
        { 利用者 1 |-> 1 });
    for i = 125 to 300 by 25
    do (
      s スケジューラ.doOneCycle(s 大域黑板, ガードコマンド集合);
      s 現在時間 := i
    );
    let mk_ (w 大域黑板属性 DB, -) = s 大域黑板. 得る (),
      w 利用者のいる場所 DB = w 大域黑板属性 DB (mk.token ("実際の利用者居場所")),
      w 仮想利用者のいる場所 DB = w 大域黑板属性 DB (mk.token ("仮想世界利用者居場所
")) in
    (
      AssertTrue
        (let {w 利用者} = dom w 利用者のいる場所 DB,
          {w 場所} = rng w 利用者のいる場所 DB,
          {w 仮想利用者} = dom w 仮想利用者のいる場所 DB,
          {w 仮想場所} = rng w 仮想利用者のいる場所 DB in
          s 大域黑板. 実際と仮想の利用者居場所が等しい (w 利用者のいる場所 DB, w 仮想利
用者のいる場所 DB) and
          w 利用者.ID を得る () = 1 and
          w 場所 = 2 and w 利用者. 前にいた場所を得る () = nil and
          w 仮想利用者.ID を得る () = 1 and
          w 仮想場所 = 2 and w 利用者. 前にいた場所を得る () = nil)
        )
      )
    )
  end
TestCaseST0002
.....

```

## 36 TestCaseST0003 クラス

```

.....
class
TestCaseST0003 is subclass of TestCaseS

```

## instance variables

```

protected ガードコマンド集合 : set of ガードコマンド := def mk_ (w 黒板属性 DB, -) = s 大域黒板. 得る () in
{
  new ガードコマンド_センサー_仮想世界 (2), new ガードコマンド_センサー_仮想世界 (3),
  new ガードコマンド_センサー_仮想世界 (4), new ガードコマンド_センサー_仮想世界 (5),
  new ガードコマンド_センサー_仮想世界 (6), new ガードコマンド_センサー_仮想世界 (7),
  new ガードコマンド_センサー_仮想世界 (8), new ガードコマンド_センサー_仮想世界 (9),
  new ガードコマンド_センサー (2), new ガードコマンド_センサー (3),
  new ガードコマンド_センサー (4), new ガードコマンド_センサー (5),
  new ガードコマンド_センサー (6), new ガードコマンド_センサー (7),
  new ガードコマンド_センサー (8), new ガードコマンド_センサー (9),
  new ガードコマンド_利用者_認証 (1), new ガードコマンド_利用者_認証 (4),
  new ガードコマンド_利用者_認証 (7),
  new ガードコマンド_利用者_認証_仮想世界 (1), new ガードコマンド_利用者_認証_仮想世界 (4),
  new ガードコマンド_利用者_認証_仮想世界 (7),
  new ガードコマンド_利用者_ドアを開ける (1), new ガードコマンド_利用者_ドアを開ける (4),
  new ガードコマンド_利用者_ドアを開ける (7), new ガードコマンド_利用者_ドアを閉める (2),
  new ガードコマンド_利用者_ドアを閉める (5), new ガードコマンド_利用者_ドアを閉める (8),
  new ガードコマンド_ID_端末_OK (wID 端末 DB (1)), new ガードコマンド_ID_端末_NG (wID 端末
  new ガードコマンド_ID_端末_OK (wID 端末 DB (4)), new ガードコマンド_ID_端末_NG (wID 端末
  new ガードコマンド_ID_端末_OK (wID 端末 DB (7)), new ガードコマンド_ID_端末_NG (wID 端末
  new ガードコマンド_ID_端末_OK_仮想世界 (wID 端末 DB (1)), new ガードコマンド_ID_端末_OK_
  new ガードコマンド_ID_端末_OK_仮想世界 (wID 端末 DB (7)),
  new ガードコマンド_利用者案内無し移動 (1), new ガードコマンド_利用者案内無し移動 (4),
  new ガードコマンド_利用者案内無し移動 (7)};

```

## operations

## public

```

TestCaseST0003 : seq of char ==> TestCaseST0003

```

```

TestCaseST0003 (nm) ==

```

```

  TestCase(nm) ;

```

## protected

```

RunTest : () ==> ()

```

```

RunTest () ==

```

```

  ( s 現在時間 := 100;

```

```

    def wID カード = s 大域黒板. 黒板属性を得る () (mk.token ("ID カード"));

```

```

    利用者 1 = new 利用者 (1, wID カード (1), s 現在時間, 10);

```

```

    利用者 2 = new 利用者 (2, wID カード (2), s 現在時間, 10);

```

```

    利用者 3 = new 利用者 (3, wID カード (3), s 現在時間, 10) in
  ( s 大域黑板.initialize({
      mk.token ("実際の利用者居場所") |->
        { 利用者 1 |-> 1, 利用者 2 |-> 4, 利用者 3 |-> 7 } });
    for i = 125 to 200 by 25
    do ( s スケジューラ.doOneCycle(s 大域黑板, ガードコマンド集合);
        s 現在時間 := i
      );
    let mk_ (w 大域黑板属性 DB, -) = s 大域黑板. 得る (),
        w 利用者のいる場所 DB = w 大域黑板属性 DB (mk.token ("実際の利用者居場所")),
        w 仮想利用者のいる場所 DB = w 大域黑板属性 DB (mk.token ("仮想世界利用者居場所
  ")); in

  AssertTrue
    (let {w 利用者 1} = dom (w 利用者のいる場所 DB :> {2}),
        {w 利用者 2} = dom (w 利用者のいる場所 DB :> {5}),
        {w 利用者 3} = dom (w 利用者のいる場所 DB :> {8}),
        利用者 ID1 = w 利用者 1.ID を得る (),
        利用者 ID2 = w 利用者 2.ID を得る (),
        利用者 ID3 = w 利用者 3.ID を得る (),
        {w 仮想利用者 1} = dom (w 仮想利用者のいる場所 DB :> {1}),
        {w 仮想利用者 2} = dom (w 仮想利用者のいる場所 DB :> {4}),
        {w 仮想利用者 3} = dom (w 仮想利用者のいる場所 DB :> {7}),
        仮想利用者 ID1 = w 仮想利用者 1.ID を得る (),
        仮想利用者 ID2 = w 仮想利用者 2.ID を得る (),
        仮想利用者 ID3 = w 仮想利用者 3.ID を得る () in
      利用者 ID1 = 1 and 利用者 ID2 = 2 and 利用者 ID3 = 3 and
      w 利用者 1. 前にいた場所を得る () = 1 and
      w 利用者 2. 前にいた場所を得る () = 4 and w 利用者 3. 前にいた場所を得る () = 7 and
      仮想利用者 ID1 = 1 and
      仮想利用者 ID2 = 2 and 仮想利用者 ID3 = 3 and
      w 仮想利用者 1. 前にいた場所を得る () = nil and
      w 仮想利用者 2. 前にいた場所を得る () = nil and w 仮想利用者 3. 前にいた場所を得る () = nil)
    )
  )
end
TestCaseST0003
.....

```

## 37 TestCaseST0004 クラス

```

.....
class
TestCaseST0004 is subclass of TestCaseS
instance variables
    protected ガードコマンド集合 : set of ガードコマンド := {
        new ガードコマンド_利用者案内有移動 (2),
        new ガードコマンド_案内 (2)};

operations
public
    TestCaseST0004 : seq of char ==> TestCaseST0004
    TestCaseST0004 (nm) ==
        TestCase(nm);
protected
    RunTest : () ==> ()
    RunTest () ==
        (
            s 現在時間 := 100;
            def wID カード = s 大域黑板. 黑板属性を得る () (mk.token ("ID カード"));
            利用者 1 = new 利用者 (1, wID カード (1), s 現在時間, 10);
            利用者 2 = new 利用者 (2, wID カード (2), s 現在時間, 10) in
            (
                s 大域黑板.initialize({
                    mk.token ("実際の利用者居場所") |->
                        { 利用者 1 |-> 2, 利用者 2 |-> 2}});
                for i = 125 to 300 by 25
                do (
                    s スケジューラ.doOneCycle(s 大域黑板, ガードコマンド集合);
                    s 現在時間 := i
                );
                let mk_ (w 大域黑板属性 DB, -) = s 大域黑板. 得る (),

```

```

        w 利用者のいる場所 DB = w 大域黒板属性 DB (mk.token ("実際の利用者居場所")) in
    AssertTrue
        (let {w 利用者 a} = dom (w 利用者のいる場所 DB :> {4}),
          {w 利用者 b} = dom (w 利用者のいる場所 DB :> {7}),
          利用者 IDa = w 利用者 a.ID を得る (),
          利用者 IDb = w 利用者 b.ID を得る () in
          利用者 IDa in set {1,2} and 利用者 IDb in set {1,2} and 利用者 IDa <> 利用
者 IDb and

          w 利用者 a. 前にいた場所を得る () = 2 and
          w 利用者 b. 前にいた場所を得る () = 2)
    )
end
TestCaseST0004

```

## 38 TestCaseST0005 クラス

このテストケースは、vdmj の乱数生成に依存している。

```

class
TestCaseST0005 is subclass of TestCaseS
operations
public
    TestCaseST0005 : seq of char ==> TestCaseST0005
    TestCaseST0005 (nm) ==
        TestCase(nm);
protected
    RunTest : () ==> ()
    RunTest () ==
        ( s 現在時間 := 100;
          def wID カード = s 大域黒板. 黒板属性を得る () (mk.token ("ID カード"));
          利用者 1 = new 利用者 (1, wID カード (1), s 現在時間 + 100, 25);
          利用者 2 = new 利用者 (2, wID カード (2), s 現在時間 + 125, 50);
          利用者 3 = new 利用者 (3, wID カード (3), s 現在時間 + 150, 75) in
          ( s 大域黒板.initialize({
              mk.token ("実際の利用者居場所") |->
              { 利用者 1 |-> 1, 利用者 2 |-> 4, 利用者 3 |-> 7 }));
            for i = 125 to 300 by 25

```

```

do ( s スケジューラ.doOneCycle(s 大域黑板, ガードコマンド集合);
    s 現在時間 := i
  );
let mk_ (w 大域黑板属性 DB, -) = s 大域黑板. 得る (),
    w 利用者のいる場所 DB = w 大域黑板属性 DB (mk_token ("実際の利用者居場所")),
    w 仮想利用者のいる場所 DB = w 大域黑板属性 DB (mk_token ("仮想世界利用者居場所
")) in
  AssertTrue
    (let {w 利用者 1} = dom (w 利用者のいる場所 DB :> {2}),
        {w 利用者 2} = dom (w 利用者のいる場所 DB :> {5}),
        {w 利用者 3} = dom (w 利用者のいる場所 DB :> {8}),
        利用者 ID1 = w 利用者 1.ID を得る (),
        利用者 ID2 = w 利用者 2.ID を得る (),
        利用者 ID3 = w 利用者 3.ID を得る (),
        {w 仮想利用者 1} = dom (w 仮想利用者のいる場所 DB :> {2}),
        {w 仮想利用者 2} = dom (w 仮想利用者のいる場所 DB :> {4}),
        {w 仮想利用者 3} = dom (w 仮想利用者のいる場所 DB :> {7}),
        仮想利用者 ID1 = w 仮想利用者 1.ID を得る (),
        仮想利用者 ID2 = w 仮想利用者 2.ID を得る (),
        仮想利用者 ID3 = w 仮想利用者 3.ID を得る () in
      利用者 ID1 = 1 and 利用者 ID2 = 2 and 利用者 ID3 = 3 and
      w 利用者 1. 前にいた場所を得る () = nil and
      w 利用者 2. 前にいた場所を得る () = 4 and w 利用者 3. 前にいた場所を得る () = 7 and
      仮想利用者 ID1 = 1 and
      仮想利用者 ID2 = 2 and 仮想利用者 ID3 = 3 and
      w 仮想利用者 1. 前にいた場所を得る () = 1 and
      w 仮想利用者 2. 前にいた場所を得る () = nil and w 仮想利用者 3. 前にいた場所を得る () = nil)
    )
  )
end
TestCaseST0005
.....

```

## 39 TestCaseST0006 クラス

```

.....
class
TestCaseST0006 is subclass of TestCaseS
operations

```

```

public
  TestCaseST0006 : seq of char ==> TestCaseST0006
  TestCaseST0006 (nm) ==
    TestCase(nm);
protected
  RunTest : () ==> ()
  RunTest () ==
    ( s 現在時間 := 100;
      def wID カード = s 大域黑板. 黑板属性を得る () (mk.token ("ID カード"));
      利用者 1 = new 利用者 (1, wID カード (1), s 現在時間 + 100, 25) in
      ( s 大域黑板.initialize({
          mk.token ("実際の利用者居場所") |->
            { 利用者 1 |-> 7 }));
        for i = 125 to 300 by 25
        do ( s スケジューラ.doOneCycle(s 大域黑板, ガードコマンド集合);
            s 現在時間 := i
          );
        let mk_ (w 大域黑板属性 DB, -) = s 大域黑板. 得る (),
            w 利用者のいる場所 DB = w 大域黑板属性 DB (mk.token ("実際の利用者居場所")),
            w 仮想利用者のいる場所 DB = w 大域黑板属性 DB (mk.token ("仮想世界利用者居場所
        ")); in
          AssertTrue
            (let {w 利用者 1} = dom (w 利用者のいる場所 DB :> {9}),
              利用者 ID1 = w 利用者 1.ID を得る (),
              {w 仮想利用者 1} = dom (w 仮想利用者のいる場所 DB :> {8}),
              仮想利用者 ID1 = w 仮想利用者 1.ID を得る () in
              利用者 ID1 = 1 and w 利用者 1. 前にいた場所を得る () = 8 and
              仮想利用者 ID1 = 1 and
              w 仮想利用者 1. 前にいた場所を得る () = 7)
            )
        )
    end
  TestCaseST0006
  .....

```

## 40 TestCaseST0007 クラス

```

.....
class

```



```

TestCaseST0007 is subclass of TestCaseS
operations
public
    TestCaseST0007 : seq of char ==> TestCaseST0007
    TestCaseST0007 (nm) ==
        TestCase(nm);
protected
    RunTest : () ==> ()
    RunTest () ==
        ( s 現在時間 := 100;
          def wID カード = s 大域黑板. 黑板属性を得る () (mk.token ("ID カード"));
            利用者 1 = new 利用者 (1, wID カード (1), s 現在時間 + 100, 25);
            利用者 2 = new 利用者 (2, wID カード (2), s 現在時間 + 125, 50);
            利用者 3 = new 利用者 (3, wID カード (3), s 現在時間 + 150, 75) in
          ( s 大域黑板.initialize({
              mk.token ("実際の利用者居場所") |->
                { 利用者 1 |-> 0, 利用者 2 |-> 0, 利用者 3 |-> 0 }));
            for i = 125 to 750 by 25
            do ( s スケジューラ.doOneCycle(s 大域黑板, ガードコマンド集合);
                s 現在時間 := i
              ) ;
            let mk_ (w 大域黑板属性 DB, -) = s 大域黑板. 得る (),
                w 利用者のいる場所 DB = w 大域黑板属性 DB (mk.token ("実際の利用者居場所")),

```

```

w 仮想利用者のいる場所 DB = w 大域黑板属性 DB (mk.token ("仮想世界利用者居場所
")) in
  AssertTrue
    (let {w 利用者 1} = dom (w 利用者のいる場所 DB :> {4}),
      {w 利用者 2} = dom (w 利用者のいる場所 DB :> {2}),
      {w 利用者 3} = dom (w 利用者のいる場所 DB :> {7}),
      利用者 ID1 = w 利用者 1.ID を得る (),
      利用者 ID2 = w 利用者 2.ID を得る (),
      利用者 ID3 = w 利用者 3.ID を得る (),
      {w 仮想利用者 1} = dom (w 仮想利用者のいる場所 DB :> {4}),
      {w 仮想利用者 2} = dom (w 仮想利用者のいる場所 DB :> {9}),
      {w 仮想利用者 3} = dom (w 仮想利用者のいる場所 DB :> {8}),
      仮想利用者 ID1 = w 仮想利用者 1.ID を得る (),
      仮想利用者 ID2 = w 仮想利用者 2.ID を得る (),
      仮想利用者 ID3 = w 仮想利用者 3.ID を得る () in
      利用者 ID1 = 1 and 利用者 ID2 = 2 and 利用者 ID3 = 3 and
      w 利用者 1.前にいた場所を得る () = 2 and
      w 利用者 2.前にいた場所を得る () = nil and w 利用者 3.前にいた場所を得る () = 2 and
      仮想利用者 ID1 = 1 and
      仮想利用者 ID2 = 2 and 仮想利用者 ID3 = 3 and
      w 仮想利用者 1.前にいた場所を得る () = 2 and
      w 仮想利用者 2.前にいた場所を得る () = 8 and w 仮想利用者 3.前にいた場所を得る () = 7)
    )
  )
end
TestCaseST0007
.....

```

## 41 TestCaseST0008 クラス

```

.....
class
TestCaseST0008 is subclass of TestCaseS
operations
public
  TestCaseST0008 : seq of char ==> TestCaseST0008
  TestCaseST0008 (nm) ==
    TestCase(nm);
protected

```

```

RunTest : () ==> ()
RunTest () ==
(
  s 現在時間 := 100;
  def wID カード = s 大域黑板. 黑板属性を得る () (mk.token ("ID カード"));
  利用者 1 = new 利用者 (1, wID カード (1), s 現在時間 + 100, 25);
  利用者 2 = new 利用者 (2, wID カード (2), s 現在時間 + 125, 50);
  利用者 3 = new 利用者 (3, wID カード (3), s 現在時間 + 150, 75) in
(
  s 大域黑板.initialize({
    mk.token ("実際の利用者居場所") |->
    { 利用者 1 |-> 9, 利用者 2 |-> 4, 利用者 3 |-> 2}});
  for i = 125 to 325 by 25
  do (
    s スケジューラ.doOneCycle(s 大域黑板, ガードコマンド集合);
    s 現在時間 := i
  );
  let mk_ (w 大域黑板属性 DB, -) = s 大域黑板. 得る (),
    w 利用者のいる場所 DB = w 大域黑板属性 DB (mk.token ("実際の利用者居場所")),

```

```

w 仮想利用者のいる場所 DB = w 大域黒板属性 DB (mk.token ("仮想世界利用者居場所
")) in
  ( AssertTrue
    (let {w 利用者 1} = dom (w 利用者のいる場所 DB :> {7}),
        {w 利用者 2} = dom (w 利用者のいる場所 DB :> {6}),
        {w 利用者 3} = dom (w 利用者のいる場所 DB :> {2}),
        利用者 ID1 = w 利用者 1.ID を得る (),
        利用者 ID2 = w 利用者 2.ID を得る (),
        利用者 ID3 = w 利用者 3.ID を得る (),
        {w 仮想利用者 1} = dom (w 仮想利用者のいる場所 DB :> {7}),
        {w 仮想利用者 2} = dom (w 仮想利用者のいる場所 DB :> {5}),
        仮想利用者 ID1 = w 仮想利用者 1.ID を得る (),
        仮想利用者 ID2 = w 仮想利用者 2.ID を得る () in
        利用者 ID1 = 1 and 利用者 ID2 = 2 and 利用者 ID3 = 3 and
        w 利用者 1.前にいた場所を得る () = 2 and
        w 利用者 2.前にいた場所を得る () = 5 and w 利用者 3.前にいた場所を得る () = nil and
        仮想利用者 ID1 = 1 and
        仮想利用者 ID2 = 2 and
        w 仮想利用者 1.前にいた場所を得る () = 2 and
        w 仮想利用者 2.前にいた場所を得る () = 4);
    if not s 大域黒板.実際と仮想の利用者居場所が等しい (w 利用者のいる場所 DB, w 仮想
    利用者のいる場所 DB)
    then s 大域黒板.実際と仮想の利用者居場所を比較表示する (w 利用者のいる場
    所 DB, w 仮想利用者のいる場所 DB)
    )
  )
)
end
TestCaseST0008
.....

```

## 42 TestCaseST0009 クラス

```

.....
class
TestCaseST0009 is subclass of TestCaseS
operations
public

```

```

TestCaseST0009 : seq of char ==> TestCaseST0009
TestCaseST0009 (nm) ==
    TestCase(nm) ;
protected
RunTest : () ==> ()
RunTest () ==
    ( s 現在時間 := 100;
      def wID カード = s 大域黑板. 黑板属性を得る () (mk.token ("ID カード"));
        利用者 1 = new 利用者 (1, wID カード (1), s 現在時間 + 100, 25);
        利用者 2 = new 利用者 (2, wID カード (5), s 現在時間 + 125, 50);
        利用者 3 = new 利用者 (3, wID カード (3), s 現在時間 + 150, 75) in
      ( s 大域黑板.initialize({
          mk.token ("実際の利用者居場所") |->
            { 利用者 1 |-> 9, 利用者 2 |-> 4, 利用者 3 |-> 2 } });
        for i = 125 to 325 by 25
        do ( s スケジューラ.doOneCycle(s 大域黑板, ガードコマンド集合);
            s 現在時間 := i
          ) ;
        let mk_ (w 大域黑板属性 DB, -) = s 大域黑板. 得る (),
            w 利用者のいる場所 DB = w 大域黑板属性 DB (mk.token ("実際の利用者居場所")),

```

```

w 仮想利用者のいる場所 DB = w 大域黑板属性 DB (mk.token ("仮想世界利用者居場所
")) in
  AssertTrue
    (let {w 利用者 1} = dom (w 利用者のいる場所 DB :> {7}),
      {w 利用者 2} = dom (w 利用者のいる場所 DB :> {4}),
      {w 利用者 3} = dom (w 利用者のいる場所 DB :> {2}),
      利用者 ID1 = w 利用者 1.ID を得る (),
      利用者 ID2 = w 利用者 2.ID を得る (),
      利用者 ID3 = w 利用者 3.ID を得る (),
      {w 仮想利用者 1} = dom (w 仮想利用者のいる場所 DB :> {7}),
      {w 仮想利用者 2} = dom (w 仮想利用者のいる場所 DB :> {4}),
      仮想利用者 ID1 = w 仮想利用者 1.ID を得る (),
      仮想利用者 ID2 = w 仮想利用者 2.ID を得る () in
      利用者 ID1 = 1 and 利用者 ID2 = 2 and 利用者 ID3 = 3 and
      w 利用者 1.前にいた場所を得る () = 2 and
      w 利用者 2.前にいた場所を得る () = nil and w 利用者 3.前にいた場所を得る () = nil and
      仮想利用者 ID1 = 1 and
      仮想利用者 ID2 = 2 and
      w 仮想利用者 1.前にいた場所を得る () = 2 and
      w 仮想利用者 2.前にいた場所を得る () = nil)
    )
  )
end
TestCaseST0009
.....

```

## 43 TestCaseST0010 クラス

```

.....
class
TestCaseST0010 is subclass of TestCaseS
operations
public
  TestCaseST0010 : seq of char ==> TestCaseST0010
  TestCaseST0010 (nm) ==
    TestCase(nm);
protected

```

```

RunTest : () ==> ()
RunTest () ==
(
  s 現在時間 := 100;
  def wID カード = s 大域黑板. 黑板属性を得る () (mk.token ("ID カード"));
  利用者 1 = new 利用者 (1, wID カード (1), s 現在時間 + 100, 25);
  利用者 2 = new 利用者 (2, wID カード (5), s 現在時間 + 125, 50);
  利用者 3 = new 利用者 (3, wID カード (3), s 現在時間 + 150, 75) in
  (
    s 大域黑板.initialize({
      mk.token ("実際の利用者居場所") |->
        { 利用者 1 |-> 9, 利用者 2 |-> 5, 利用者 3 |-> 2 });
    for i = 125 to 325 by 25
    do (
      s スケジューラ.doOneCycle(s 大域黑板, ガードコマンド集合);
      s 現在時間 := i
    );
    let mk_ (w 大域黑板属性 DB, -) = s 大域黑板. 得る (),
        w 利用者のいる場所 DB = w 大域黑板属性 DB (mk.token ("実際の利用者居場所")),
        w 仮想利用者のいる場所 DB = w 大域黑板属性 DB (mk.token ("仮想世界利用者居場所
  ")); in
    AssertTrue
      (let {w 利用者 1} = dom (w 利用者のいる場所 DB :> {7}),
        {w 利用者 2, w 利用者 3} = dom (w 利用者のいる場所 DB :> {2}),
        利用者 ID1 = w 利用者 1.ID を得る (),
        利用者 ID2 = w 利用者 2.ID を得る (),
        利用者 ID3 = w 利用者 3.ID を得る (),
        {w 仮想利用者 1} = dom (w 仮想利用者のいる場所 DB :> {7}),
        {w 仮想利用者 2} = dom (w 仮想利用者のいる場所 DB :> {5}),
        仮想利用者 ID1 = w 仮想利用者 1.ID を得る (),
        仮想利用者 ID2 = w 仮想利用者 2.ID を得る () in
        { 利用者 ID1, 利用者 ID2 } = {1, 2} and 利用者 ID3 = 3 and
        {w 利用者 1. 前にいた場所を得る (), w 利用者 2. 前にいた場所を得る ()} = {2, 6} and
        w 利用者 3. 前にいた場所を得る () = nil and
        仮想利用者 ID1 = 1 and
        仮想利用者 ID2 = 2 and
        w 仮想利用者 1. 前にいた場所を得る () = 2 and
        w 仮想利用者 2. 前にいた場所を得る () = nil)
      )
    )
  end
)
TestCaseST0010
.....

```

## 44 タッチパネルクラス

.....

class

タッチパネル is subclass of オブジェクト

types

public タッチパネル状態 = <入力可能> | <入力不可能>;

public タッチパネル位置 = 場所グラフ ‘場所 ID

instance variables

public s タッチパネル状態 : タッチパネル状態 := <入力不可能>;

public s タッチパネル位置 : タッチパネル位置;

operations

public

タッチパネル : タッチパネル位置 \* タッチパネル状態 ==> タッチパネル

タッチパネル (a タッチパネル位置, a タッチパネル状態) == atomic

( s タッチパネル状態 := a タッチパネル状態;

s タッチパネル位置 := a タッチパネル位置

);

public

コピーする : タッチパネル位置 ==> タッチパネル

コピーする (a タッチパネル位置) ==

( return new タッチパネル (a タッチパネル位置, s タッチパネル状態)

);

public

位置を得る : () ==> タッチパネル位置

位置を得る () ==

return s タッチパネル位置;

public

ID を得る : () ==> タッチパネル位置

ID を得る () ==

return 位置を得る ();

public

入力可能にする : () ==> ()

入力可能にする () ==

s タッチパネル状態 := <入力可能>;

public



```
入力不可能にする : () ==> ()
入力不可能にする () ==
    s タッチパネル状態 := <入力不可能>;
public
    入力可能である : () ==> bool
    入力可能である () ==
        return s タッチパネル状態 = <入力可能>;
public
    入力不可能である : () ==> bool
    入力不可能である () ==
        return s タッチパネル状態 = <入力不可能>
end
タッチパネル
.....
```

## 45 ユーザークラス

.....

class

ユーザー is subclass of オブジェクト

types

public ユーザー ID = ID

.....

### 45.1 インスタンス変数

s 到着時間は、ユーザーがある場所に到着した時間を示す。

s 躊躇する最大時間は、移動を躊躇する最大時間である。

.....

instance variables

public s ユーザー ID : ユーザー ID;

public sID カード : ID カード;

public s 到着時間 : 時間;

public s 前にいた場所 : [場所グラフ '場所 ID'] := nil;

public s 躊躇する最大時間 : 時間;

public s 作成者名 : seq of char := "";

.....

### 45.2 操作

.....

operations

public

ユーザー : ユーザー ID \* ID カード \* 時間 \* 時間 ==> ユーザー

ユーザー (a ユーザー ID, aID カード, a 到着時間, a 気まぐれ時間) == atomic

( s ユーザー ID := a ユーザー ID;

  sID カード := aID カード;

  s 到着時間 := a 到着時間;

  s 躊躇する最大時間 := a 気まぐれ時間

);

public

利用者 : 利用者 ID \* ID カード \* 時間 \* [場所グラフ ‘場所 ID’] \* 時間 \* `seq of char` ==> 利用者

利用者 (a 利用者 ID, aID カード, a 到着時間, a 前にいた場所, a 気まぐれ時間, a 作成者名) == `atomic`

```
( s 利用者 ID := a 利用者 ID;
  sID カード := aID カード;
  s 到着時間 := a 到着時間;
  s 前にいた場所 := a 前にいた場所;
  s 躊躇する最大時間 := a 気まぐれ時間;
  s 作成者名 := a 作成者名
);
```

#### 45.2.1 属性を得る

利用者の属性が同じかを判定するための属性を得る。

`public`

属性を得る : () ==> 利用者 ID \* ID カード \* 時間

属性を得る () ==

```
return mk_(s 利用者 ID, sID カード, s 躊躇する最大時間);
```

`public`

コピーする : `seq of char` ==> 利用者

コピーする (a 作成者名) ==

```
( return new 利用者 (s 利用者 ID, sID カード, s 到着時間, s 前にいた場所, s 躊躇する最大時間,
a 作成者名)
);
```

`public`

ID カードを得る : () ==> ID カード

ID カードを得る () ==

```
return sID カード;
```

`public`

ID を得る : () ==> 利用者 ID

ID を得る () ==

```
return s 利用者 ID;
```

`public`

有効な ID を持っている : () ==> `bool`

有効な ID を持っている () == `sID カード`.

```
有効である ();
```

`public`

```

到着時間を更新する : () ==> 利用者
到着時間を更新する () ==
    ( s 到着時間 := s 現在時間;
      return self
    );
public
前にいた場所を設定する : [場所グラフ '場所 ID] ==> ()
前にいた場所を設定する (a 場所 ID) ==
    s 前にいた場所 := a 場所 ID;
public
前にいた場所を得る : () ==> [場所グラフ '場所 ID]
前にいた場所を得る () ==
    return s 前にいた場所;
public
移動していない : () ==> bool
移動していない () ==
    return s 前にいた場所 = nil;
.....

```

### 45.3 移動する気になった時間である

利用者が移動する気になった時間を、ランダムに決定する。

```

.....
public
移動する気になった時間である : () ==> bool
移動する気になった時間である () ==
    def w 乱数 = MATH.rand(s 躊躇する最大時間);
    w 時間の乱れ =
        if s 躊躇する最大時間 = 0
        then 0
        else w 乱数 / s 躊躇する最大時間;

```

```

w 移動する気になった時間 = s 躊躇する最大時間 * w 時間の乱れ in
(
  def -=
    debug >= 9 => new IO().echo
    (
      "\n「移動する気になった時間である」で生成された乱数 = " ^
      VDMUtil.val2seq_of_char[int](w 乱数) ^
      "\n") in
    skip;
    return s 現在時間 >= w 移動する気になった時間 + s 到着時間
  );

```

#### 45.4 到着時間が一番早い利用者を選ぶ

到着時間が一番早い利用者の集合から一つを選ぶ。到着時間が一番早い利用者が複数存在する場合、現在の選び方は処理系依存である。

```

public static
到着時間が一番早い利用者を選ぶ : set of 利用者 ==> 利用者
到着時間が一番早い利用者を選ぶ (a 利用者集合) ==
  let w 次の利用者 in set a 利用者集合 be st
    forall w 利用者 in set a 利用者集合 & w 次の利用者.s 到着時間 <= w 利用者.s 到着時間 in
      return w 次の利用者
  pre a 利用者集合 <> {}
  post forall w 利用者 in set a 利用者集合 & RESULT.s 到着時間 <= w 利用者.s 到着時間
end
利用者

```

Test Suite : vdm.tc

Class : 利用者

| Name           | #Calls | Coverage |
|----------------|--------|----------|
| 利用者 'ID を得る    | 0      | 0%       |
| 利用者 'コピーする     | 0      | 0%       |
| 利用者 '属性を得る     | 0      | 0%       |
| 利用者 '移動していない   | 0      | 0%       |
| 利用者 'ID カードを得る | 0      | 0%       |
| 利用者 '到着時間を更新する | 0      | 0%       |
| 利用者 '前にいた場所を得る | 0      | 0%       |

| Name                  | #Calls | Coverage  |
|-----------------------|--------|-----------|
| 利用者 '前にいた場所を設定する'     | 0      | 0%        |
| 利用者 '有効な ID を持っている'   | 0      | 0%        |
| 利用者 '移動する気になった時間である'  | 0      | 0%        |
| 利用者 '到着時間が一番早い利用者を選ぶ' | 0      | 0%        |
| 利用者 '利用者'             | 0      | 0%        |
| 利用者 '利用者'             | 0      | 0%        |
| <b>Total Coverage</b> |        | <b>0%</b> |

## 46 参考文献等

VDM++<sup>[2]</sup> は、1970 年代中頃に IBM ウィーン研究所で開発された VDM-SL<sup>[1]</sup> を拡張し、さらにオブジェクト指向拡張した形式仕様記述言語で、フリーソフトウェア<sup>\*1</sup>である。

### 参考文献

- [1] Kyushu University. VDM-SL 言語マニュアル. Kyushu University, 第 2.0 版, 2016. Revised for VDMTools V9.0.2.
- [2] Kyushu University. VDMTools VDM++ 言語マニュアル. Kyushu University, 第 2.0 版, 2016. Revised for VDMTools V9.0.2.

---

<sup>\*1</sup> GNU General Public License v3.0