

実務家のための形式手法

厳密な仕様記述を志すための形式手法入門 第二版

# 実践法:モデル化の課題例

－ 図書館システム －

## 実践法:モデル化の課題例(このモジュールについて)



VDM を用いたモデル化の課題を通して、実際の問題に対して**自身で VDM を適用**するための準備状況を確認するモジュール

### ■ 事前知識・経験

- ソフトウェア開発と形式手法導入に関する基礎知識
- VDM++ と VDMTools の基礎知識

### ■ 学習目標

- VDM を用いたモデル化の手順をたどることができる

### ■ 主な学習項目

- 図書館の例題を通した手順のポイントの確認

SEC-FM1-07-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center 2

講師の方へ:

このモジュールでは、図書館機能の一部を例題にモデル化の演習を行います。

1.具体的な記述をどのように行なうか

2.詳細化の手順はどのようなものか

といった点を意識させるようにお話しください。

講習の前に、ご自身でもツールを使い、記述および検証を試みることをお勧めします。

### 【VDM言語仕様参考資料】

・SCSK Corporation. VDM-SL 言語マニュアル. SCSK Corporation, 第 1.2 版, 2012. Revised for VDMTools V9.0.2.

・SCSK Corporation. VDM++ 言語マニュアル. SCSK Corporation, 第 1.2 版, 2012. Revised for VDMTools V9.0.2.

・佐原伸. 形式手法の技術講座—ソフトウェアトラブルを予防する. ソフトリサーチセンター, 2008.

## 1. 要求概要



### ■ 要求聞き出し(elicitation)の結果

- 本を図書館の蔵書として追加、削除する。
- 題名と著者と分野のいずれかで本を検索する。
- 利用者が、蔵書を借り、返す。
- 蔵書の追加・削除や、利用者への貸出・返却は職員が行う。
- 利用者や職員の権限などは考慮なくてよい。
- 最大蔵書数は10000、利用者一人への最大貸出冊数は3冊とする。

今回の図書館機能における「日本語仕様」に相当する。

実際は、他の機能による制約もみたとようにシステムを構築することになる。

## 2. 要求仕様 ver.0 (非オブジェクトモデル)



- VDM++はオブジェクト指向仕様記述言語であるが、関数型言語の機能も持っているので、オブジェクト指向にこだわる必要はない
  - 一般に、**関数型指向でモデル化した方が抽象度が高く、記述行数も少なく済む**
    - オブジェクト指向は、設計仕様以降に適用した方がよい
- 最初は、クラスをモジュールと考えて、事前条件・事後条件・不変条件だけを考えたモデルを作成してみる
  - このような仕様を、実行可能な**陽仕様**に対して、**陰仕様**と呼ぶ

SEC-FM1-07-2

Copyright © 2012,2013 IPA

IPA Software Engineering Center 4

なぜ形式手法か 14 枚目のスライド(なぜ形式手法か? 理論的根拠)にあるように、副作用のある操作は再利用が難しい。

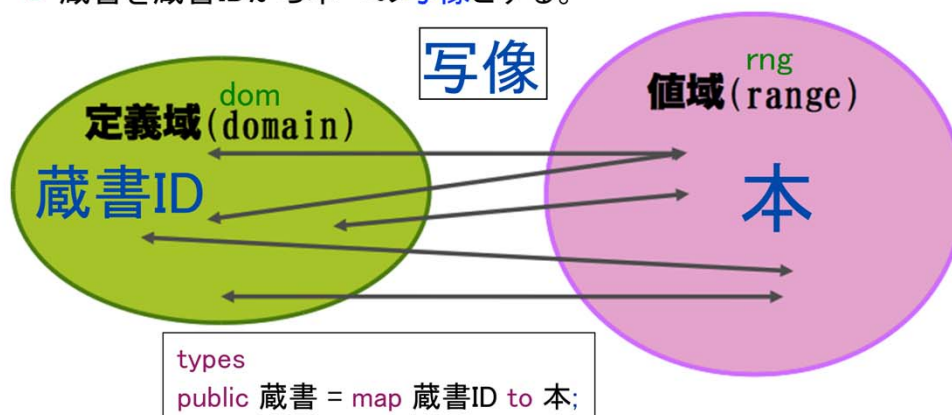
関数型言語による記述は抽象度の高い記述を可能とし、副作用がなく、オブジェクトよりも細粒度の操作単位での再利用を可能とする。

また、クラス設計も、最初は単なるモジュールとみなして、データ構造の詳細にこだわらず、トップダウンに型定義を進めるのがよい。

### 3. 要求仕様 ver.0 (Library0)



- 日本語仕様では、**抽象的な意味の本と、図書館の蔵書として管理すべき個々の本が明確に区別されていない。**
- 図書館では、同じ本を何冊か蔵書として持つことがあるので、以下のモデルでは、著者や題名という属性を持つ本に対し、
  - 蔵書を蔵書IDから本への**写像**とする。



ここでは、この図書館における本の抽象化を行なっている。

対象となるシステムあるいは問題領域により、どの性質に注目すべきか異なる。

## 4. 図書館システムの仕様 – 型、値、操作



### ■ 仕様作成に必要な型と値の定義

- クラスは図書館0とし、そこで、すべての必要な型を定義する。
  - 本を図書館の蔵書として追加、削除する。
  - 題名と著者と分野のいずれかで本を検索する。
  - 利用者が、蔵書を借り、返す。
  - 蔵書の追加・削除や、利用者への貸出・返却は職員が行う。
  - 利用者や職員の権限などは考慮しなくてよい。
  - 最大蔵書数は10000、利用者一人への最大貸出冊数は3冊とする。

## 型と値の定義



```
class 図書館0
types
public 蔵書 = map 蔵書ID to 本;
public 本 ::
  f題名 :
  f著者 :
  f分野集合 : set of 分野;
public 題名 = seq of char;
public 著者 = seq of char;
public 分野 = seq of char;
public 蔵書ID = token;
public 職員 = token;
public 利用者 = token;
```

1. 今後のモデル化で、項目が追加される可能性が強いので、本はレコード型として定義した。
2. 蔵書を一意に識別するため、蔵書IDを定義する。
3. 蔵書ID、職員、利用者は、他と識別するためにtokenとした。
4. 著者などは、モデルを詳細化するに伴いオブジェクトとしてモデル化する可能性があるが、ここでは単に名前だけ管理することにした。

```
values
public v最大蔵書数 = 10000;
public v最大貸出数 = 3;

end 図書館0
```

## 蔵書として追加、削除する の陰仕様



```
class 図書館0
types
public 蔵書 = map 蔵書ID to 本;

operations
public 蔵書を追加する: 蔵書 ==> ()
蔵書を追加する(a追加本) == is not yet specified
pre ???
post ???;

public 蔵書を削除する: 蔵書 ==> ()
蔵書を削除する(a削除本) == is not yet specified
pre ???
post ???;

end 図書館0
```

1. ??? に当てはめるべき日本語の文章を考えよ。
2. その日本語名を持つ関数のインタフェースを定義せよ。
3. 必要なインスタンス変数があれば定義せよ。
4. 関数の陽仕様を定義せよ。



## 蔵書として追加、削除する の陰仕様



```
operations
public
  蔵書を追加する: 蔵書 ==> ()
  蔵書を追加する(a追加本) == is not yet specified
pre
  蔵書に存在しない(a追加本, i蔵書)
post
  蔵書が追加されている(a追加本, i蔵書, i蔵書~);

public
  蔵書を削除する: 蔵書 ==> ()
  蔵書を削除する(a削除本) == is not yet specified
pre
  蔵書に存在する(a削除本, i蔵書)
post
  蔵書が削除されている(a削除本, i蔵書, i蔵書~);
```

## インスタンス変数と、蔵書に存在しない のインタフェース



```
instance variables
private i蔵書 : 蔵書 := {}->};
inv
card dom i蔵書 <= v最大蔵書数;
```

1. インスタンス変数は蔵書IDから本への写像である蔵書型のi蔵書
  - 1.1. 濃度(集合の要素数)がv最大蔵書数以下であるという**不変条件**を持つ。
  - 1.2. **初期値**は空の写像である。

```
functions
蔵書に存在しない : 蔵書 * 蔵書 +> bool
蔵書に存在しない(a追加本, a図書館蔵書) == is not yet specified;
```

## 蔵書に存在しない、蔵書に存在する の陽仕様



### functions

蔵書に存在する : 蔵書 \* 蔵書 +> bool

蔵書に存在する(a追加本, a図書館蔵書) ==

dom a追加本 subset dom a図書館蔵書;

-- a追加本 = (dom a追加本 < a図書館蔵書)

-- forall id in set dom a追加本 &

-- id in set dom a図書館蔵書

蔵書に存在しない : 蔵書 \* 蔵書 +> bool

蔵書に存在しない(a追加本, a図書館蔵書) ==

not 蔵書に存在する(a追加本, a図書館蔵書);

## 蔵書が追加されている、蔵書が削除されている の陽仕様



### functions

蔵書が追加されている : 蔵書 \* 蔵書 \* 蔵書 +> bool

蔵書が追加されている(a追加本, a図書館蔵書, a図書館蔵書旧値) ==

a図書館蔵書 = a図書館蔵書旧値 **munion** a追加本;

蔵書が削除されている : 蔵書 \* 蔵書 \* 蔵書 +> bool

蔵書が削除されている(a削除本, a図書館蔵書, a図書館蔵書旧値) ==

a図書館蔵書 = **dom** a削除本 <-: a図書館蔵書旧値;

-- a図書館蔵書旧値 = a図書館蔵書 **munion** a削除本;

## 本を検索する の陰仕様



public

本を検索する: (題名 | 著者 | 分野) ==> 蔵書

本を検索する(a検索キー) == is not yet specified

pre

???

post

???

## 本を検索する の陰仕様



Public

本を検索する: (題名 | 著者 | 分野) ==> 蔵書

本を検索する(a検索キー) == is not yet specified

pre

検索キーが空でない(a検索キー)

post forall book in set rng RESULT &

(book.f題名 = a検索キー or

book.f著者 = a検索キー or

a検索キー in set book.f分野集合);

forallは全称限量子、RESULTは返値を表す

## 本を貸す の陰仕様



```
types
public 貸出 = inmap 利用者 to 蔵書;

instance variables
i貸出 : 貸出 := {}->;

operations
public
  本を貸す: 蔵書 * 利用者 * 職員 ==> ()
  本を貸す(a貸出本, a利用者, -) == is not yet specified
pre
  ???
post
  ???;
```

## 本を貸す の陰仕様



operations

public

本を貸す: 蔵書 \* 利用者 \* 職員 ==> ()

本を貸す(a貸出本, a利用者, -) == is not yet specified

pre

貸出可能である(a利用者, a貸出本, i貸出, i蔵書, v最大貸出数)

post

貸出に追加されている(a貸出本, a利用者, i貸出, i貸出~);



## 本を貸す の事前条件の関数



public

貸出可能である : 利用者 \* 蔵書 \* 貸出 \* 蔵書 \* nat1 +> bool

貸出可能である(a利用者, a貸出本, a貸出, a図書館蔵書, a最大貸出数) ==  
蔵書に存在する(a貸出本, a図書館蔵書) and  
貸出に存在しない(a貸出本, a貸出) and  
最大貸出数を超えていない(a利用者, a貸出本, a貸出, a最大貸出数);

貸出に存在しない : 蔵書 \* 貸出 +> bool

貸出に存在しない(a貸出本, a貸出) ==  
not 貸出に存在する(a貸出本, a貸出);

貸出に存在する : 蔵書 \* 貸出 +> bool

貸出に存在する(a貸出本, a貸出) ==  
let w蔵書 = merge rng a貸出 in  
dom a貸出本 subset dom w蔵書;

## 最大貸出数を超えていない の陽仕様



public

最大貸出数を超えていない : 利用者 \* 蔵書 \* 貸出 \* nat1+> bool

最大貸出数を超えていない(a利用者, a貸出本, a貸出, a最大貸出数) ==

if a利用者 not in set dom a貸出

then card dom a貸出本 <= a最大貸出数

else card dom a貸出(a利用者) + card dom a貸出本 <= a最大貸出数;

## 本を貸す の事後条件の関数



貸出に追加されている : 蔵書 \* 利用者 \* 貸出 \* 貸出 +> bool

貸出に追加されている(a貸出本, a利用者, a貸出, a貸出旧値) ==

if a利用者 in set dom a貸出

then a貸出 = a貸出旧値 ++ {a利用者 |-> (a貸出(a利用者) munion a貸出本)}

else a貸出 = a貸出旧値 munion {a利用者 |-> a貸出本};

## 本を返す の陰仕様



operations

public

本を返す: 蔵書 \* 利用者 \* 職員 ==> ()

本を返す(a返却本, a利用者, -) == is not yet specified

pre

???

post

???

## 本を返す の陰仕様



operations

public

本を返す: 蔵書 \* 利用者 \* 職員 ==> ()

本を返す(a返却本, a利用者, -) == is not yet specified

pre

貸出に存在する(a返却本, i貸出)

post

貸出から削除されている(a返却本, i貸出, i貸出~);

## 本を返す の事後条件



```
貸出から削除されている : 蔵書 * 貸出 * 貸出 +> bool
貸出から削除されている(a削除本, a貸出, a貸出旧値) ==
    貸出に存在する(a削除本, a貸出旧値) and
    貸出に存在しない(a削除本, a貸出);
```

## 要求仕様 ver.0 (Library0) 陰仕様の作成



- 以上で作成した陰仕様は、要求仕様として、必要十分である。
  - しかし、要求仕様の正当性検証は静的チェックしか行えない
  - また、妥当性検証は事後条件をレビューするしかない
- VDMは本来段階的洗練(refinement)を使って、事後条件から手続き的仕様に証明しながら変換して、正当性検証を行う
  - しかし、現場で証明を行うのは、技術的、工数的に難しいため、VDMToolsやOverturetoolといったVDMの処理系では、仕様を実行して 検証することを行っている。
  - そこで、陰仕様モデルの次に、陽仕様モデルを作成し、動的チェックを行うことで、正当性検証と妥当性検証を行う。

## 5. 要求仕様 ver.1 (Library1) 陰仕様から陽仕様の作成

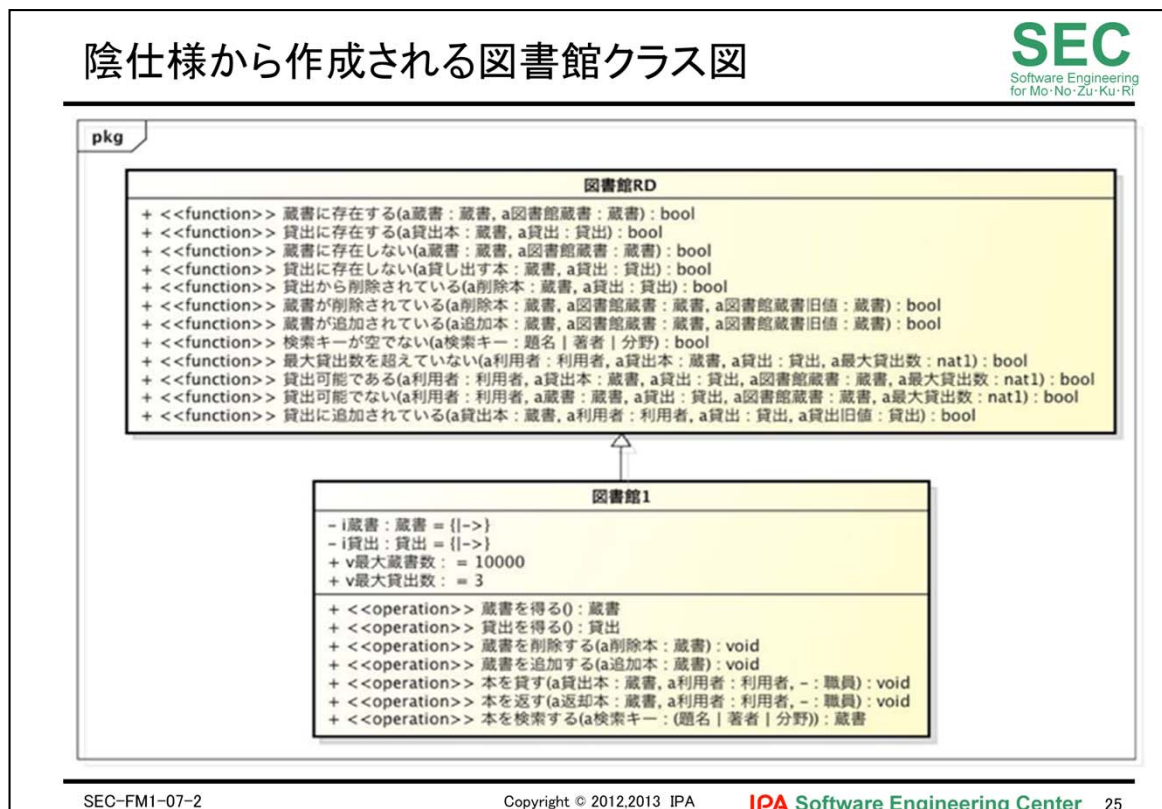


■ 図書館0クラス (Library0.vpp) の陰仕様 (下請け関数は陽仕様) を元に、陽仕様を作成せよ。本を返す、は下記の通り。

- 本を検索する、の操作本体には写像内包式を使うと良い。
- 利用者が既に本を借りている場合と、そうでない場合に場合分け する必要がある。

```
public
本を返す: 蔵書 * 利用者 * 職員 ==> ()
本を返す(a返却本, a利用者, -) ==
  let w利用者への貸出本 = i貸出(a利用者),
      w利用者への貸出本new = dom a返却本 <-: w利用者への貸出本 in
  if w利用者への貸出本new = {}->
  then i貸出 := {a利用者} <-: i貸出
  else i貸出 := i貸出 ++ {a利用者 |-> w利用者への貸出本new}
pre 貸出に存在する(a返却本, i貸出)
post 貸出から削除されている(a返却本, i貸出, i貸出~);
```





ver.1 は、ver.0 を基に詳細化した記述である。

陽仕様モデルはそれぞれの手続きの中身を記述している。

## 蔵書を追加する の陽仕様



public

蔵書を追加する : 蔵書 ==> ()

蔵書を追加する(a追加本) ==

i蔵書 := i蔵書 **munion** a追加本

pre

蔵書に存在しない(a追加本, i蔵書)

post

蔵書が追加されている(a追加本, i蔵書, i蔵書~);

## 蔵書を削除する の陽仕様



public

蔵書を削除する: 蔵書 ==> ()

蔵書を削除する(a削除本) ==

i蔵書 := dom a削除本 <-: i蔵書

pre

蔵書に存在する(a削除本, i蔵書) and

貸出に存在しない(a削除本, i貸出)

post

蔵書が削除されている(a削除本, i蔵書, i蔵書~);

## 本を検索する の陽仕様



```
public
本を検索する: (題名 | 著者 | 分野) ==> 蔵書
本を検索する(a検索キー) ==
    return {id |-> i蔵書(id) | id in set dom i蔵書 &
              (i蔵書(id).f題名 = a検索キー or
               i蔵書(id).f著者 = a検索キー or
               a検索キー in set i蔵書(id).f分野集合)}
pre 検索キーが空でない(a検索キー)
post forall book in set rng RESULT &
    (book.f題名 = a検索キー or
     book.f著者 = a検索キー or
     a検索キー in set book.f分野集合);
```

## 本を貸す の陽仕様



public

本を貸す: 蔵書 \* 利用者 \* 職員 ==> ()

本を貸す(a貸出本, a利用者, -) ==

if a利用者 in set dom i貸出

then i貸出 := i貸出 ++ {a利用者 |-> (i貸出(a利用者) munion a貸出本)}

else i貸出 := i貸出 munion {a利用者 |-> a貸出本}

pre

貸出可能である(a利用者, a貸出本, i貸出, i蔵書, v最大貸出数)

post

貸出に追加されている(a貸出本, a利用者, i貸出, i貸出~);

## 6. 要求仕様 ver.1 (Library1) のテスト



### 回帰テスト用操作

public

蔵書を得る : () ==> 蔵書

蔵書を得る() == return i蔵書;

public

貸出を得る : () ==> 貸出

貸出を得る() == return i貸出;

## 要求仕様 ver.1 (Library1) 仕様の実行テスト



- 検証は仕様の内部矛盾をチェックするだけであり、ユーザーの要求に合致するかという妥当性のチェックはできない。
- そこで、仕様実行を使用した実行可能仕様の検証と妥当性チェックが必要になる。

モデルを詳細化しても、基本的に、事前条件および事後条件は変わらない。

詳細化に伴う書き換えに関して回帰テストを実施することにより、詳細化の前後で、意図せずに仕様が変更されていないことを確認できる。

## 要求仕様 ver.1 (Library1) 回帰テスト



### ■ 回帰テスト・ライブラリー

- VDMUnit.vpp

### ■ 回帰テストソース参照

- MyTest.vpp
- MyTestCase.vpp

### ■ 回帰テスト実行

- `debug new TestApp().run()`



## MyTest.vpp



```
class TestApp
operations
public run : () ==> ()
run() == (
  dcl ts : TestSuite := new TestSuite("図書館貸し出しモデルの回帰テスト\n"),
    tr : TestResult := new TestResult();
  tr.addListener(new PrintTestListener());
  ts.addTest(new TestCaseUT0001("TestCaseUT0001:✂ノーマルケースの単体テスト"));
  ts.addTest(new TestCaseUT0002("TestCaseUT0001:✂エラーケースの単体テスト"));
  ts.run(tr);
  if tr.wasSuccessful() = true then
    def - = new IO().echo("*** 全回帰テストケース成功。***") in skip
  else
    def - = new IO().echo("*** 失敗したテストケースあり!! ***") in skip
);
end TestApp
```

## MyTestCase.vpp その1



```
class TestCaseComm is subclass of TestCase

operations
public print : seq of char ==> ()
print(a文字列) ==
    let - = new IO().echo(a文字列) in skip;
end TestCaseComm

class TestCaseUT0001 is subclass of TestCaseComm
operations

public TestCaseUT0001: seq of char ==> TestCaseUT0001
TestCaseUT0001(name) == setName(name);

public test01: () ==> ()
test01 () == (
    let w図書館 = new 図書館1(),
```

## MyTestCase.vpp その2



```
分野1 = "ソフトウェア",  
分野2 = "工学",  
分野3 = "小説",  
著者1 = "佐原伸",  
著者2 = "井上ひさし",  
本1 = mk_図書館1`本("デザインパターン",著者1,{分野1,分野2}),  
本2 = mk_図書館1`本("紙屋町桜ホテル",著者2,{分野3}),  
蔵書1 = {mk_token(101) |-> 本1},  
蔵書2 = {mk_token(102) |-> 本1},  
蔵書3 = {mk_token(103) |-> 本2},  
蔵書4 = {mk_token(104) |-> 本2},  
利用者1 = mk_token("利用者1"),  
利用者2 = mk_token("利用者2"),  
職員1 = mk_token("職員1")  
  
in (  
  w図書館.蔵書を追加する(蔵書1);
```

## MyTestCase.vpp その3



```
w図書館.蔵書を追加する(蔵書2);
w図書館.蔵書を追加する(蔵書3);
w図書館.蔵書を追加する(蔵書4);
assertTrue("test01 蔵書の追加がおかしい。",
    w図書館.蔵書を得る() = {mk_token(101) |-> 本1, mk_token(102) |-> 本1,
                             mk_token(103) |-> 本2, mk_token(104) |-> 本2}
);
let w見つかった本1 = w図書館.本を検索する("井上ひさし"),
    w見つかった本2 = w図書館.本を検索する("工学")
in
assertTrue("test01 本の検索がおかしい。",
    w見つかった本1 = {mk_token(103) |-> 本2, mk_token(104) |-> 本2} and
    w見つかった本2 = {mk_token(101) |-> 本1, mk_token(102) |-> 本1}
);
```

## MyTestCase.vpp その4



```
w図書館.本を貸す({mk_token(101) |-> 本1}, 利用者1, 職員1);  
w図書館.本を貸す({mk_token(103) |-> 本2}, 利用者1, 職員1);  
w図書館.本を貸す({mk_token(104) |-> 本2}, 利用者2, 職員1);  
assertTrue("test01 本の貸出がおかしい。",  
    let w貸出 = w図書館.貸出を得る() in  
    w貸出 = {利用者1 |-> {mk_token(101) |-> 本1, mk_token(103) |-> 本2},  
             利用者2 |-> {mk_token(104) |-> 本2}}  
);
```

## MyTestCase.vpp その5



```
w図書館.本を返す({mk_token(103) |-> 本2}, 利用者1, 職員1);  
w図書館.本を返す({mk_token(104) |-> 本2}, 利用者2, 職員1);  
assertTrue("test01 本の返却がおかしい。",  
    let w貸出 = w図書館.貸出を得る() in  
    w貸出 = {利用者1 |-> {mk_token(101) |-> 本1}}  
);  
w図書館.蔵書を削除する({mk_token(104) |-> 本2});  
assertTrue("test01 蔵書の削除がおかしい。",  
    w図書館.蔵書を得る() =  
    {mk_token(101) |-> 本1, mk_token(102) |-> 本1,  
    mk_token(103) |-> 本2}  
);  
);  
end TestCaseUT0001
```

## MyTestCase.vpp その6(想定したエラー発生)



```
class TestCaseUT0002 is subclass of TestCaseComm
operations

public TestCaseUT0002: seq of char ==> TestCaseUT0002
TestCaseUT0002(name) == setName(name);

public test01: () ==> ()
test01 () == (
  trap <RuntimeError> with
    print("¥ttest01 意図通り、最大貸出数を超えたエラー発生。¥n")
  in
    let w図書館 = new 図書館1(),
```

## MyTestCase.vpp その7(想定したエラー発生)



```
分野1 = "ソフトウェア",
分野2 = "工学",
分野3 = "小説",
著者1 = "佐原伸",
著者2 = "井上ひさし",
本1 = mk_図書館1`本("デザインパターン",著者1,{分野1,分野2}),
本2 = mk_図書館1`本("紙屋町桜ホテル",著者2,{分野3}),
蔵書1 = {mk_token(101) |-> 本1},
蔵書2 = {mk_token(102) |-> 本1},
蔵書3 = {mk_token(103) |-> 本2},
蔵書4 = {mk_token(104) |-> 本2},
利用者1 = mk_token("利用者1"),
職員1 = mk_token("職員1")
in (
  w図書館.蔵書を追加する(蔵書1);
  w図書館.蔵書を追加する(蔵書2);
  w図書館.蔵書を追加する(蔵書3);
  w図書館.蔵書を追加する(蔵書4);
```



## MyTestCase.vpp その8(想定したエラー発生)



```
assertTrue("test01 蔵書の追加がおかしい。",
  w図書館.蔵書を得る() =
    {mk_token(101) |-> 本1, mk_token(102) |-> 本1, mk_token(103) |-> 本2, mk_token(104) |-> 本2}
);
let w見つかった本1 = w図書館.本を検索する("井上ひさし"),
    w見つかった本2 = w図書館.本を検索する("工学")
in
assertTrue("test01 本の検索がおかしい。",
  w見つかった本1 = {mk_token(103) |-> 本2, mk_token(104) |-> 本2} and
  w見つかった本2 = {mk_token(101) |-> 本1, mk_token(102) |-> 本1}
);
w図書館.本を貸す({mk_token(101) |-> 本1}, 利用者1, 職員1);
w図書館.本を貸す({mk_token(102) |-> 本1}, 利用者1, 職員1);
w図書館.本を貸す({mk_token(103) |-> 本2}, 利用者1, 職員1);
w図書館.本を貸す({mk_token(104) |-> 本2}, 利用者1, 職員1);
assertTrue("test01 本の貸出がおかしい。",
  let w貸出 = w図書館.貸出を得る() in
    w貸出 = {}
);
```

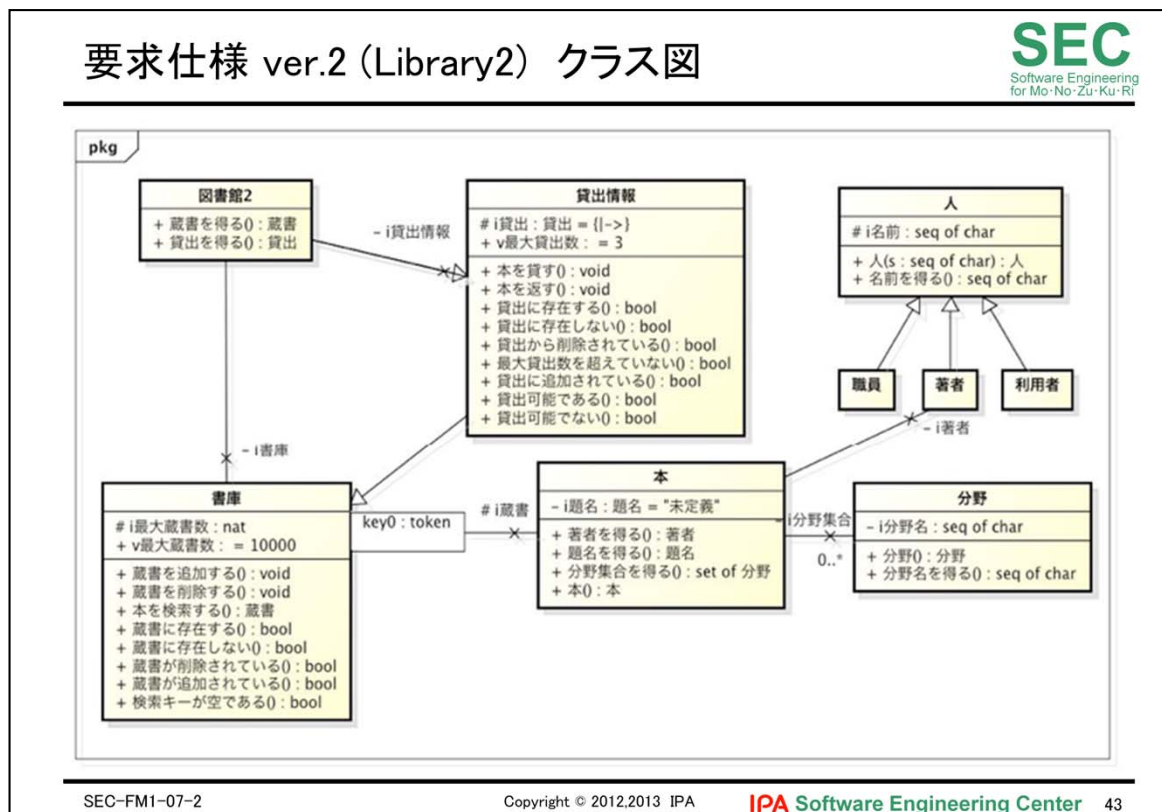
## 7. 要求仕様 ver.2 (Library2) オブジェクトモデル



### ■ ソースを参照のこと

- Library2.vpp
- ファイル名に2が付いた.vdmppファイル
- BookStacks.vdmpp
- 回帰テストは、MyTest2.vppとMyTestCase2.vpp
  - 回帰テスト実行
    - `debug new TestApp2().run()`

ver.2 は、オブジェクト指向に基づくモデル化であり、ver.1 に基づいて詳細化している。  
ver.2 における図書館を表す構造が『図書館2』クラスである。



ここでは一覧性のためにクラス図を示しているが、VDM 記述において、クラス図を描く必然性があるわけではない。

## 要求仕様 ver.2 (Library2) オブジェクトモデル



### ■ オブジェクトモデルの問題点は何か？

## オブジェクトモデルの問題点



- 蔵書の型を変えると、他のクラスに修正が波及する。
  - 蔵書を型ではなく、クラスとすべきである？
  
- 書庫クラスと、貸出情報クラスと図書館2クラスの情報隠蔽が 不十分である
  - 図書館2クラスを貸出情報のサブクラスにすべきではない？

## サマリー



図書館の例に対してVDMを用いたモデル化を行う事例を通して、実際の問題に対して自身でVDMを適用するための準備状況を確認する  
モデル化の一般的手順

### ■ モデル化の手順の実践例とそのポイント



---

実務家のための形式手法

厳密な仕様記述を志すための形式手法入門

実践法:モデル化の課題例  
－ 図書館システム －

独立行政法人情報処理推進機構  
技術本部 ソフトウェア・エンジニアリング・センター  
統合系システム・ソフトウェア信頼性基盤整備推進委員会  
上流品質技術部会 人材育成WG(編)  
2013年3月 第二版発行

記載されている個々の情報に関しての著作権及び商標はそれぞれの権利者に帰属するものです  
なお、本書の内容は将来予告なしに変更することがあります