第 1 回 UML ゼミ

ゼミ担当者　：　千田　智治，李　翠敏，上村　英里沙，松下　知明
指導院生　　：　宮崎　真，小林　賢二
開催日　　：　2006 年 5 月 19 日

ゼミ内容: 本ゼミでは，プログラミング演習に向けて，UML の概念とオブジェクト指向，そして EclipseUML の使い方について学ぶ

1　はじめに

近年オブジェクト指向技術は幅広く使われるようになってきており，それに伴いモデルを表現するための様々な方法論が開発されてきた．しかし，このような方法論の乱立はシステムを開発する際に，各々の開発者の間での意思の疎通に混乱を招く．そこで，開発者の間で共通の理解を得ることができる統一的な方法が重要となる．本ゼミでは，統一的なモデリング手法である UML(UML:Unified Modeling Language) を Eclipse の Plugin である EclipseUML を用いて学ぶ.

2　UML

2.1　UML とは

UML はオブジェクト指向システムを多面的に表現するための表記方法で，実際に目には見えないシステムの構造や振る舞いを，目に見える「図 (ダイアグラム)」という形で表現することである．このようにオブジェクト指向の考えを基本とすることで，複雑なシステムの構造も容易に理解でき，システム開発を効率的に進めることを可能にするためのモデリングができる．このモデリングをするためには，システムが担う業務の構成要素をオブジェクトとして捉え，オブジェクトの動きやオブジェクト間の関係を整理，分析することが重要である．

2.2　UML のメリット

- 表現力が高度であるため情報が正確に伝えることができる
- ビジュアルであるため容易に理解できる
- 1 つのシステムを多角的な視点から見ることができる
- 国際的に共通したコミュニケーションができる
- 開発者とユーザー間のコミュニケーションも滑らかになる
- 構築した UML モデルを次の開発にも利用できる

3　オブジェクト指向

UML を考えるにはオブジェクト指向が重要である．オブジェクト指向は，内部処理を考えずに外から見える事象について考え，それぞれが固有の役割を持って独立しており，他の事象とはメッセージのやり取りのみで協調をとるという考え方である．今まで用いられてきた構造化の考え方ではプログラムの処理ごとに分割していたが，大規模になると分割したくとも処理の分け方が分からなくなるからである．

3.1　オブジェクトとは

オブジェクト指向では，関連するデータの集合と，それに対するメソッド（振る舞い）をまとめて「オブジェクト」とし，それを組み合わせることでソフトウェアを構築できる．また，オブジェクトをより一般的なものにすることを抽象化といい，抽象化したものをクラスと呼ぶ．特に，これは大規模なソフトウェア開発で効果的な考え方である．オブジェクトの 4 つの特性には属性，振る舞い，関係，アイデンティティがある．

属性（データ）とは，オブジェクトの固有の姿・形・性質などの状態を，属性（attribute）の値で表現する．具体的には，物理的なオブジェクトの属性は「大きさ」「重さ」「色」などがあり，人の属性では Fig. 1 のように「名前」「生年月日」「身長」「体重」「血液型」などがである．

振る舞い（メソッド）とは，各オブジェクトは固有の振る舞いを持つことである．物理的なオブジェクトの振る舞いとして、Fig. 1 のように「人」のオブジェクトでは「話す」「歩く」「食べる」「寝る」「働く」などがある．

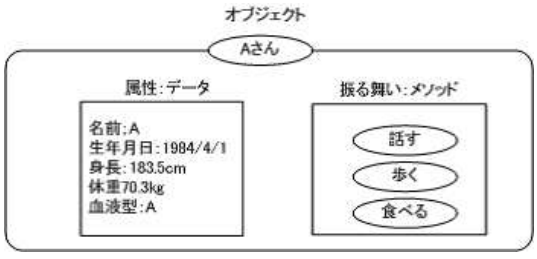

Fig. 1　属性と振る舞いの関係 (出展：自作)

関係とは，オブジェクトは単独で存在せず，Fig. 2 のように必ずほかのオブジェクトと関係を持っていること

である．

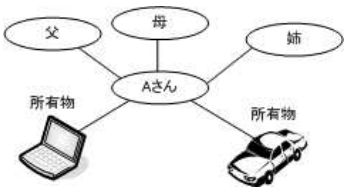

Fig. 2　オブジェクト同士の関係 (出展：自作)

アイデンティティとは，各オブジェクトは，ほかのオブジェクトから自らを識別できる自己同一性を持っていることである．

3.2　オブジェクト指向の特徴

オブジェクト指向の特徴としては，カプセル化，メッセージパッシング，継承，多態性などである．これらにより対象の中身の構造を知らない状態でも操作できることである．具体例としては，テレビの電源ボタンを押せばテレビの電源を入れることが可能なことが挙げられる．なぜなら，テレビで多くのメーカーが異なる構造で内部を作ったとしても，テレビの操作方法が変わらず，また部品を取り替えても動作に影響を与えないからである．

カプセル化とは，Fig. 3 にあるようにオブジェクトに含まれるデータを外部から隠蔽し，公開されている関数やメソッドからのみアクセスできるようにする仕組みである．この利点はオブジェクトの独立性を高め，内部の仕様変更などが外部に影響しにくくなる．簡単に言えば，すでに存在しているオブジェクトは，内部構造や動作原理を詳細に知らない状態でも，外部からのメッセージに対して何が返ってくるかを知っていればいいのである．

メッセージパッシングとは，オブジェクトに対して指示を与える唯一の手段である．このときやり取りされる命令や応答のことをメッセージという．

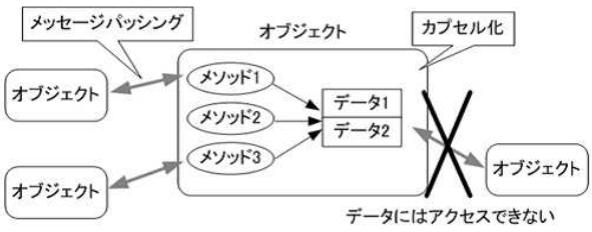

Fig. 3　オブジェクト指向の仕組み (出展：自作)

継承とは，Fig. 4 のようにあるオブジェクトが他のオブジェクトの特性を引き継ぐ場合に出来る関係のことである．継承されたオブジェクトをスーパークラス，継承したオブジェクトをサブクラスという．

また最後にオブジェクト指向の大きな特徴である．ポリモフィズム（多態性）とは，同じ名前のメッセージを

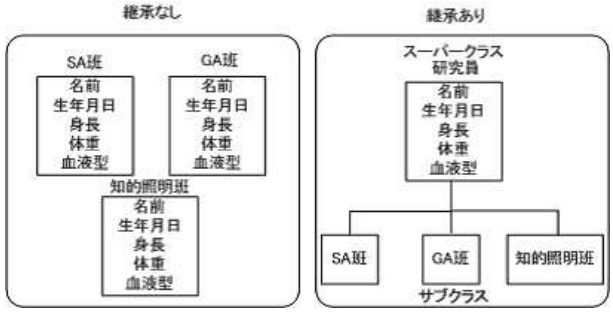

Fig. 4　継承の比較 (出展：自作)

送っても受け手によって適切なメソッドが呼び出されることである．簡単に言えば，Fig. 5 のように同じ質問でも応答はするが，それぞれで答えが違うのである．

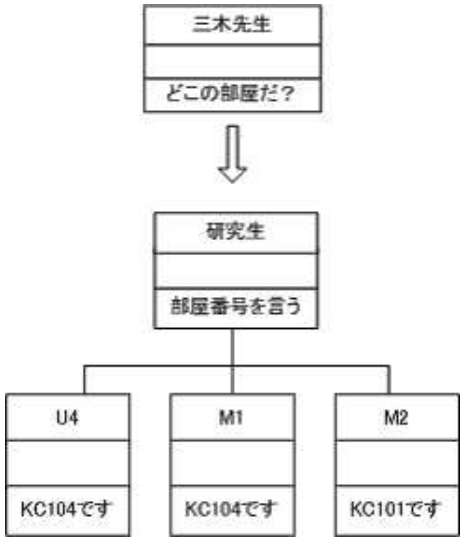

Fig. 5　多態性の具体例 (出展：自作)

3.3　オブジェクト指向の目的

オブジェクト指向で作ったプログラムは再利用したり，修正しやすくなっているので，複雑かつ大規模化するソフトウェアを整理，再編し，ソフトウェアの生産性上げるために不可欠になる．このようにオブジェクト指向を用いることで信頼性向上などの様々な問題に対処できるようになる．

4　UML による設計手法

4.1　オブジェクト指向で用いられる UML

オブジェクト指向で用いられる UML は Table 1 のように多岐にわたる．

Table 1　UML の種類

視点	図の種類	ダイヤグラム
モデル化対象に対する要求を表す	要求図	ユースケース図
モデル化対象の静的・論理的な構造を表す	構造図	クラス図
		オブジェクト図
		パッケージ図
モデル化対象の動的な振る舞いを表す	振る舞い図	シーケンス図
		コミュニケーション図
		アクティビティ図
		ステートマシン図
モデル化対象の物理的な構造を表す	実装図	コンポーネント図
		配置図

システムの構成要素とその相互関係を静的に表現することができるクラス図，オブジェクト間のやりとりを時系列で表現するシーケンス図などがある．

4.2　モデル化

4.2.1　モデル化とは

モデル化とは，オブジェクト指向において現実世界にある物を，「データ」と「振る舞い」という 2 つの構成要素でモデル化して，コンピュータ上に表現する．「データ」とは，モデル化の対象となる物の性質や状態などを表す情報である．また，「振る舞い」とは，その物が提供する機能や，データを使用する手段を表す．

4.2.2　クラスの作成手順

Fig. 6 のシナリオからクラスをみつける．手順としては以下の通りである．

- 名詞と動詞をすべて列挙する．
- 同義語をまとめる．
- 名詞はクラス，インスタンス，属性などに分類する．

- 動詞は操作（メソッド）に対応する．
- システムに関係の無いものを排除して，メソッドの元になる名詞を探し，クラスを確定する．

Fig. 6，Fig. 7 より，以下のようにクラスを作成する．

「A さん」　→　Member クラス

「本」　→　Book クラス

システムは一個しかないのでクラスにはしない．

「販売サイト」　→　Site クラス

4.3　クラス図の作成

4.3.1　EclipseUML によるクラス図の作成

EclipseUML を用いてクラス図描いていく．Eclipse を起動し，Fig. 8 のように「ファイル」-「新規」-「プロジェクト」を選択する．

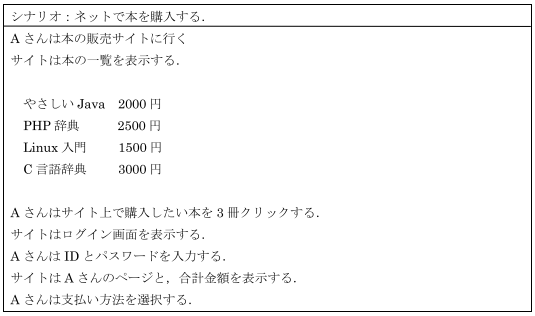

Fig. 6　シナリオ (出展：自作)

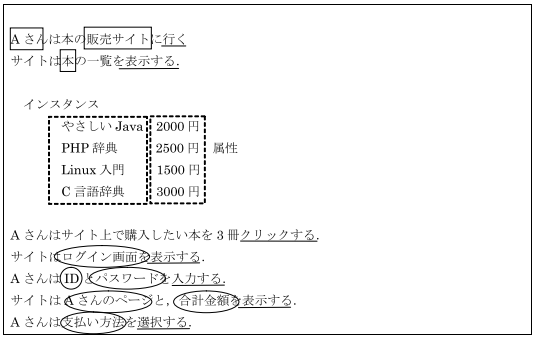

Fig. 7　クラスの作成 (出展：自作)

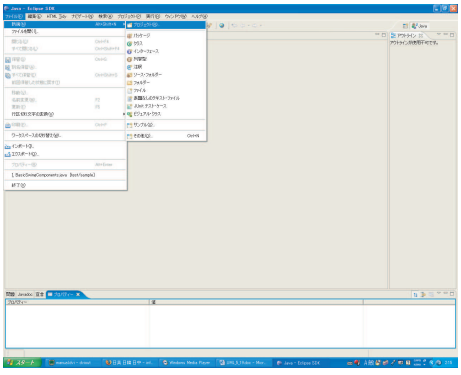

Fig. 8　プロジェクトの作成 (出展：自作)

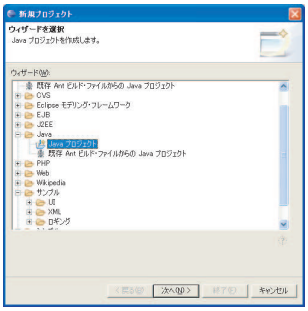

Fig. 9　プロジェクトの選択 (出展：自作)

Fig. 9 に示すように，「Java」-「Java プロジェクト」を選択して「次へ」をクリックする．

「プロジェクト名」に Sample_UML を入力して，Fig. 10 のように，終了する．

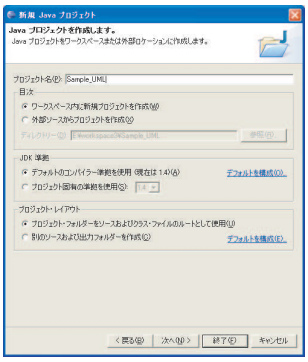

Fig. 10　プロジェクト名の入力 (出展：自作)

Fig. 11 に示すように，「ファイル」-「新規」-「その他」を選択する．

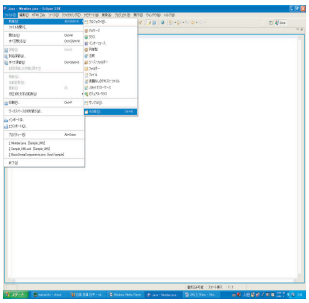

Fig. 11　その他の選択 (出展：自作)

「UML Diagram」-「UML Class Diagram」を選択して，Fig. 12 のように，「次へ」をクリックする．

「親フォルダーを入力または選択」で Sample_UML と入力する（Fig. 13）．

Fig. 14 のような画面になることを確認する．

Fig. 15 に示すように，ツールバーにある緑色で C と書かれた「create a class」ボタンを押す．

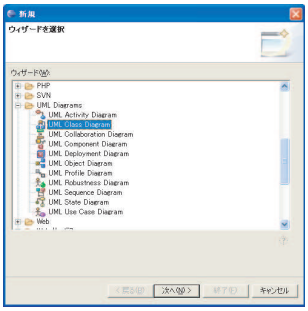

Fig. 12　ウィザードを選択 (出展：自作)

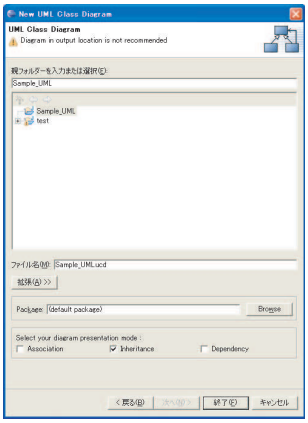

Fig. 13　親フォルダーの入力 (出展：自作)

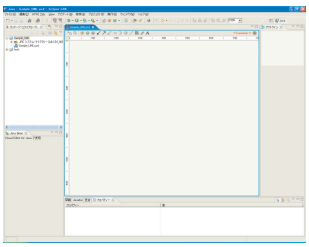

Fig. 14　ファイルの確認 (出展：自作)

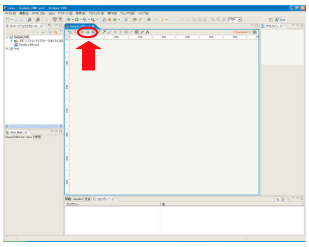

Fig. 15　ツールバーの選択 (出展：自作)

Fig. 16 に示すように，スケッチ領域内でドラッグして範囲を選択する．

「名前」に Member と入力する．Fig. 17 に示すように，他のチェックボックスを空欄にして「終了」ボタンをクリックする．

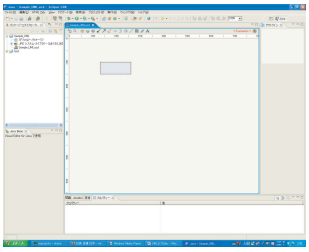

Fig. 16　範囲の選択 (出展：自作)

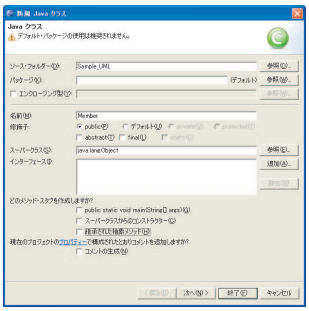

Fig. 17　クラス名の入力 (出展：自作)

Fig. 18 のように，クラス図をダブルクリックすると，Fig. 19 のプロパティビューに対応するコードが出現する．

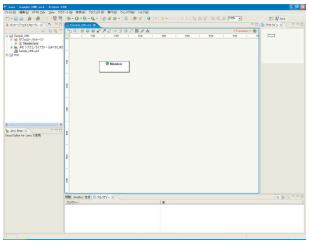

Fig. 18　Member のクラス図 (出展：自作)

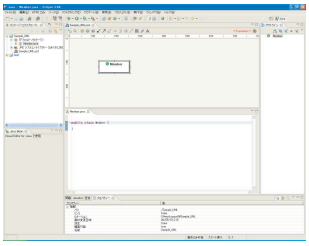

Fig. 19　コードの出現 (出展：自作)

同様に「Book」クラスと「Site」クラスも作成し，それらをダブルクリックする．Fig. 20 の「Book」クラスと「Site」クラスのプロパティビューに対応するコードを出現させる．

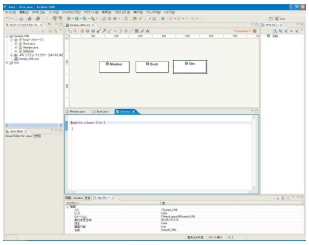

Fig. 20　全クラス図の出力 (出展：自作)

4.3.2　クラスの作成

Fig. 21 に示すように,「ファイル」-「新規」-「クラス」を選択する．

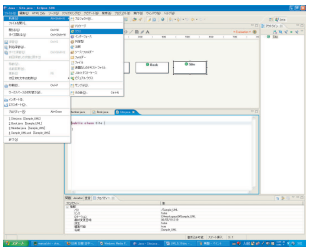

Fig. 21　クラスの作成 (出展：自作)

「名前」に，App と入力して App.java を作成する．「public static void main(String[] args)」のチェックボックスにチェックを入れて,「終了」をクリックする（Fig. 22）．

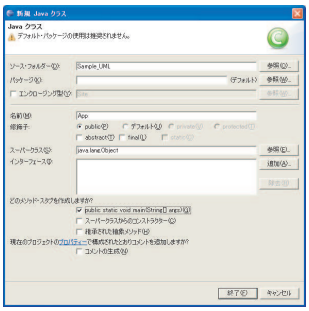

Fig. 22　App.java の作成 (出展：自作)

Fig. 23 に示すように，パッケージエクスプローラーの App.java をスケッチ領域内にドラッグ＆ドロップを行い，コードからクラス図を作成する．

App.java において，main 文に Fig. 24 のプログラムを書き加えて，Fig. 25 のように，保存を行う．

Book.java で Fig. 26 のプログラムを書き加える．Fig. 27 に示すように，保存を行う．

App.java を開き,「実行」-「Java アプリケーション」を選択する（Fig. 28）．

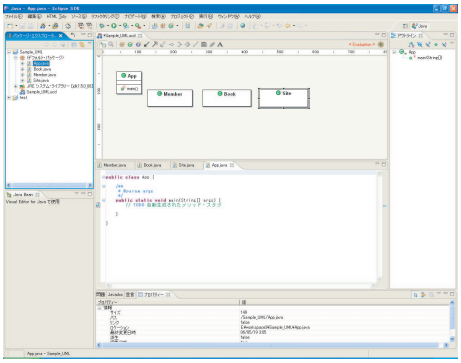

Fig. 23　コードからクラス図を作成 (出展：自作)

```
Book book = new Book();
System.out.println("ブッククラスのインスタンスが作成されました。");

Member member = new Member();
System.out.println("会員クラスのインスタンスが作成されました。");

Site site = new Site();
System.out.println("サイトクラスのインスタンスが作成されました。");
```

Fig. 24　App.java(出展：自作)

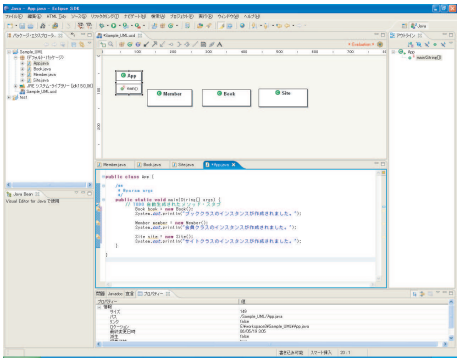

Fig. 25　main 文の書き換え (出展：自作)

```
private String name;

public void setName(String _name){
        this.name = _name;
}

public String getName(){
        return this.name;
}
```

Fig. 26　Book.java(出展：自作)

Fig. 29 の，コンソールビューに以下のような文章が出力される．

```
ブッククラスのインスタンスが作成されました。
会員クラスのインスタンスが作成されました。
サイトクラスのインスタンスが作成されました。
```

Book クラスで右クリックを行い，Fig. 30 のように

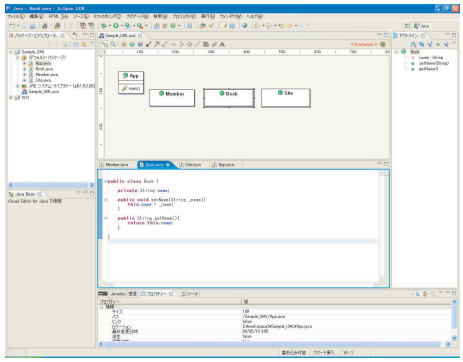

Fig. 27　Book.java の書き換え (出展：自作)

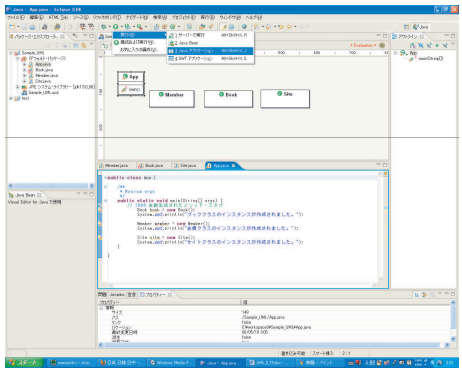

Fig. 28　Java アプリケーションの選択 (出展：自作)

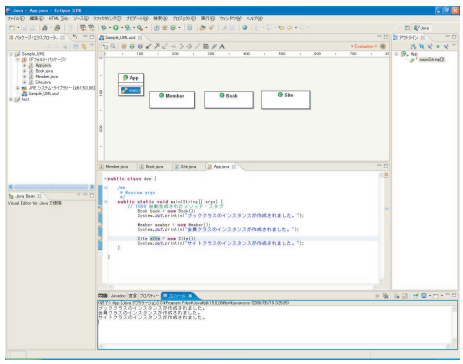

Fig. 29　コンソールビュー (出展：自作)

「view selector」を選択する．

Fig. 31 に示すように，「Attributes」タブの「name:String」のチェックボックスにチェックを入れる．

「Methods」タブの「getName():String」と「setName(String)」のチェックボックスにチェックを入れる．また，「Signature」の「Display parameter type」と「Display parameter name」,「Display return type」のチェックボックスにチェックを入れる．「OK」ボタンをクリックする（Fig. 32）．

この実行結果を Fig. 33 に示す．

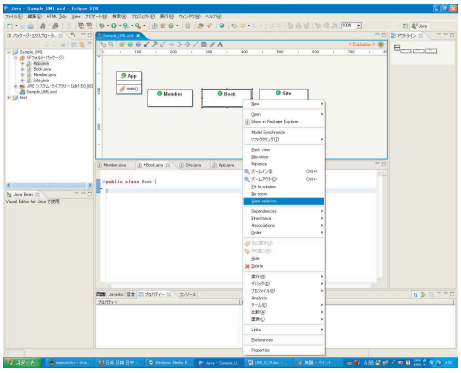

Fig. 30　view selector の選択 (出展：自作)

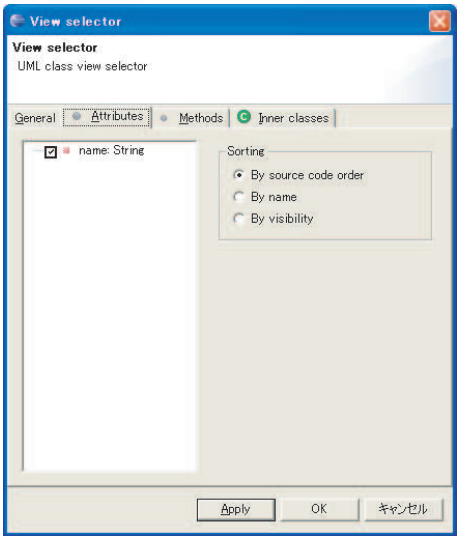

Fig. 31　Attributes(出展：自作)

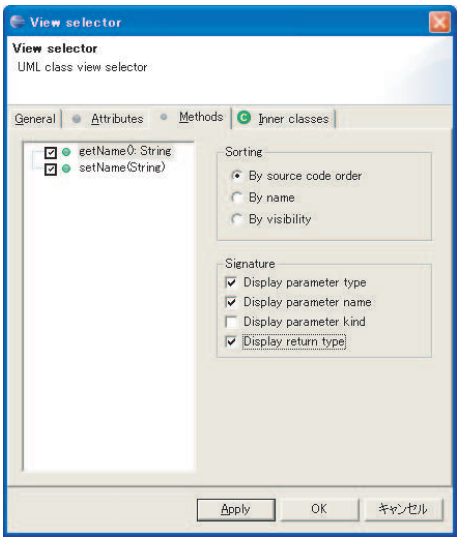

Fig. 32　Methods(出展：自作)

クラス図とコードの対応は Fig. 34 の通りである．

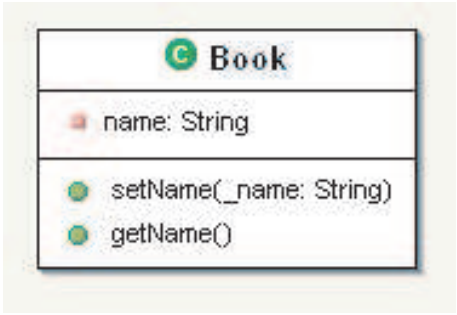

Fig. 33　Book クラス (出展：自作)

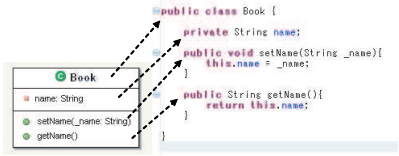

Fig. 34　クラス図とコードの対応 (出展：自作)

4.4　メソッドの作成

Fig. 6 のシナリオ通りにメソッドを作成し，本の合計金額を求めるプログラムを作成する．まず，App.java の変更を行う．プログラムを Fig. 35 のように変更を行った．

```
public class App {

/**
 * @param args
 */
public static void main(String[] args) {
        // TODO 自動生成されたメソッド・スタブ
        Book book1 = new Book("やさしい Java",2000);
        Book book2 = new Book("PHP 辞典",2500);
        Book book3 = new Book("Linux 入門",1500);
        Book book4 = new Book("C 言語辞典",3000);

        Member member = new Member();
        Site site = new Site();

        int book1_price = book1.getPrice();
        int book2_price = book2.getPrice();
        int book3_price = book3.getPrice();
        int book4_price = book4.getPrice();

        int sum = 0;
        sum = book1_price + book2_price + book3_price;

        System.out.println("1 冊目に購入した書籍の名前は"+book1.getName());
        System.out.println("値段は"+book1.getPrice()+"円です");
        System.out.println("2 冊目に購入した書籍の名前は"+book2.getName());
        System.out.println("値段は"+book2.getPrice()+"円です");
        System.out.println("3 冊目に購入した書籍の名前は"+book3.getName());
        System.out.println("値段は"+book3.getPrice()+"円です");
        System.out.println("お買い上げ頂いた書籍の総額は"+sum+"円になります");
        }
}
```

Fig. 35　App.java(出展：自作)

このプログラムによって，App.java は Book.java で定義した本の名前と金額（インスタンス）を持ってきている．次に，Book.java のプログラムを書き換える．変更後のプログラムを Fig. 36 に載せる．

この実行結果は，以下の通りである．

```
public class Book {
                **
                 * @uml.property   name="name"
                */
                private String name;
                **
                 * @uml.property   name="price"
                 */
                rivate int price;

                public Book(String _name,int _price){
                        his.name = _name;
                        his.price = _price;

                **
                 * @param name   設定する name。
                * @uml.property   name="name"
                /
                ublic void setName(String _name){
                this.name = _name;
                }
                /**
                * @return   name を戻します。
                 * @uml.property   name="name"
                */
                public String getName(){
                        eturn this.name;
                }
                /**
                * @return   price を戻します。
                 * @uml.property   name="price"
                */
                public int getPrice(){
                        return this.price;

}
```

Fig. 36　Book.java(出展：自作)

```
1 冊目に購入した書籍の名前はやさしい Java
値段は 2000 円です
2 冊目に購入した書籍の名前は PHP 辞典
値段は 2500 円です
3 冊目に購入した書籍の名前は Linux 入門
値段は 1500 円です
お買い上げ頂いた書籍の総額は 6000 円になります
```

参考文献

1) はじめて学ぶ UML, ナツメ社，竹政照利，2003 年.
2) UML 徹底活用, 翔泳社，井上樹，2005 年.
3) すいすい習得 UML モデリング, 技術評論社，岡村敦彦，2006 年.