4～6 月 Java 教育報告書

H19 年 6 月 29 日
三重野　恵

1.目的

Java 言語の理解、及びプログラミングを繰り返し行うことで、現場レベルでのプログラミングが行えるスキル及び必要知識を身に付けることを目的とする。

2.方法

参考書、Web サイトを活用し、サンプルプログラム、練習問題等で Java 言語でのプログラミングに必要な知識の習得を行い、出題された演習問題についてアルゴリズムの思案、プログラムを作成する。また、見出された不明な点、理解できていなかった点について参考書・参考 Web サイトを用い知識の再習得を行う。

3.結果

参考書・Web サイトでの例題、練習問題を実施することでプログラミングに必要な知識、スキルの習得ができた。また、まとめとしての演習問題で、習得しきれていなかった知識、スキルの習得、及び理解しきれていなかった知識の再習得が行えた。

4.結論

Java 言語の理解、プログラム作成を繰り返し行うことで、必要知識及びスキルの習得ができた。また、演習問題を行うことで現場レベルのプログラミングを行えるための基礎知識及び基礎スキルの習得ができた。今後、更に知識・スキルを高めていき、現場レベルでのプログラミングを行えるように精進する。

5.コメント

3 ヶ月という期間で Java 言語、オブジェクト指向プログラミングの基礎から始めたので、習得した範囲の知識・スキルで現場レベルのプログラミングを行えるレベルまでは達することはできていないが、その基礎部分は作れた。引き続き知識・スキルの習得を行って、現場レベルのプログラミングスキルの習得を目指したい。

目次

頁
[1] 目的 1
[2] 対象システム 1
[3] 方法 1
3.1 Java 基本文法(構造化プログラミングレベル)について 1
3.2 アルゴリズムについて 2
3.3 Java 基本文法(オブジェクト指向型プログラミングレベル)について 2
3.4 Java による DB 操作および SQL について 3
3.5 デザインパターンについて 4
3.6 JRE の API の使い方について 4
3.7 コーディング規約について 4
[4] 結果 5
4.1 Java 基本文法(構造化プログラミングレベル)について 5
4.2 アルゴリズムについて 8
4.3 Java 基本文法(オブジェクト指向型プログラミングレベル)について 9
4.4 Java による DB 操作および SQL について 11
4.5 デザインパターンについて 12
4.6 JRE の API の使い方について 15
4.7 コーディング規約について 19
[5] 結論 20
[6] 参考文献 21

[1] 目的

現在、情報技術のローエンドからハイエンドまで幅広く利用され、非常に多くの分野で使用されているニーズの高い言語である Java は、構文の多くを C 及び C++から引き継いでいる。また、狭義でオブジェクト指向プログラミング言語であり、プラットフォームに依存しないアプリケーションソフトウェアの開発、配備が可能である。
この Java 言語について以下の項目を理解し、現場レベルでのプログラミングが行えるスキル及び必要知識を身に付けることを目的とする。

- Java 基本文法(構造化プログラミングレベル)
- アルゴリズム
- Java 基本文法(オブジェクト指向型プログラミングレベル)
- Java による DB 操作および SQL
- デザインパターン
- JRE の API の使い方
- コーディング規約

[2] 対象システム

Java 教育で使用する PC 及び、開発環境を以下に示す。

PC ： Microsoft Windows 2000 5.00.2195 Service Pack 4
IDE ： eclipse-SDK-3.2.2-win32
MySQL ： MySQL Server 5.2
MySQL Connector Java ： mysql-connector-java-5.0.5

[3] 方法

3.1 Java 基本文法(構造化プログラミングレベル)について

参考書（新 Java 言語入門　ビギナー編）を熟読する。また、サンプルプログラムを作成し、実行することにより以下の構文の動作を確認する。

- 基礎知識(文字列出力、数値出力、繰り返し処理、条件処理)
- 変数と定数(変数、定数)
- よく使われる演算子(算術演算子、関係演算子、論理演算子、インクリメント/デクリメント演算子)
- 制御文(if 文、for 文、while 文、do-while 文、switch 文、break 文(ラベル有無)、

continue 文)

- 配列(多次元配列)
- コンソールへの入出力(文字列、整数、浮動小数点、例外処理)

また、Java 教育(基本編)演習問題を行うことで理解度を確認する。演習問題では、以下の 2 つを作成する。

1. 数当て[0～99 の数を当てるゲーム]
2. 数当て[3 桁の数字を当てるゲーム(マスターマインド)]

1 及び 2 はコンソールへ数を直接入力して行う数当てゲームでこれらのフローチャート、及びプログラムの作成により習熟度の確認し、理解しきれていない部分の知識習得を行う。

3.2　アルゴリズムについて

参考 Web サイト（目指せプログラマー!のアルゴリズム入門）を用い、以下の項目についての練習問題でフローチャート、及びプログラムを作成することにより、知識習得を行う。

- アルゴリズム(アルゴリズムとは、フローチャート)
- 順次構造(代入、計算、入力、出力)
- 分岐構造(条件分岐、単一分岐、多重分岐、複合条件、多方向分岐)
- 反復構造(前判定型、後判定型、多重反復処理(ネスト))
- 配列(1 次配列、2 次配列)
- 検索・ソート(サブルーチン、シーケンシャルサーチ、バイナリサーチ、ルックアップテーブル、選択法、交換法、バブルソート、挿入法)
- 文字列(文字列処理、文字列コピー、文字列の比較・連結・検索)

3.3　Java 基本文法(オブジェクト指向型プログラミングレベル)について

参考書（Java 言語プログラミングレッスン　上/下）を熟読する。また、サンプルプログラムを作成し、実行することにより以下の構文の動作を確認する。

- メソッド(宣言と呼び出し、引数と戻り値)
- 配列(宣言・初期化と代入・参照、要素・長さ、添字、二次元配列)
- オブジェクト指向へ向けて(クラス、インスタンス、フィールド、メソッド、コンストラクタ)
- クラスとインスタンス(クラス、インスタンス、フィールド、メソッド、コンストラクタ、クラスフィールド、クラスメソッド、修飾子)
- スーパークラスとサブクラス(継承、final クラス、抽象クラス)
- 例外(try…catch…finally 文、throw 文、throws 節、コンパイラにチェックされない例外(RuntimeException 、 Error) 、チェックされる例外(RuntimeException 以外の Exception))
- インターフェース(実装、extends と implements、クラスとインターフェース

の比較)

参考 Web サイト（ソフトウェア構成論）を用い、以下の項目についての例題、練習問題を通してプログラミングを行うことにより知識習得を行う。

- クラスと継承(継承、オーバーライド)
- インターフェース
- インターフェースとコレクション(抽象データ型、具体的なデータ型、コレクション(Set、List、Map))
- コレクション(続き)(primitive 型、wrapper クラス、Map、Iterator と foreach 文、Map と Iterator)
- まとめ(継承、インターフェース、コレクションを利用してポーカーのシミュレーションプログラム)

3.4　Java による DB 操作および SQL について

参考 Web サイト（ようこそアクセスデータベース講座へ）の例題より、Microsoft Access を使用しながら、データベースについての知識習得を行う。

参考 Web サイト（MySQL 4.1 リファレンスマニュアル）に沿って文法等の知識習得を行う。

- データベース作成
- テーブル作成
- データの Insert(追加)
- データの Update(更新)
- データの Select(取得)
- データの Delete(削除)

参考 Web サイト（JDBC プログラミングの基本）に沿って、Java から MySQL のデータベースへアクセスする方法、文法等の知識習得を行う。

- Insert(データの追加)
- Update(データの更新)
- Select(データの取得)
- Delete(データの削除)

演習問題_Java 教育(DB 編)を行う。なお、演習問題はユーザ情報の管理を行うシステム開発であり、実行用クラス、アカウント管理クラス、テーブルクラス、ユーザ認証クラスから構成されている。これらのクラスを作成することで、以下の項目についての知識を習得する。

- SQL 基礎知識(Insert、Update、Select、Delete の構文)
- Java からの DB 操作

3.5　デザインパターンについて

参考 Web サイト（矢沢久雄の早わかり GoF デザインパターン）を用いて、GoF デザインパターン全 23 種の概要の習得を行う。

- オブジェクトの生成に関するパターン(Abstract Factory、Builder、Factory Method、Prototype、Singleton パターン)
- プログラムの構造に関するパターン(Adapter、Bridge、Composite、Decorator、Facade、Flyweight、Proxy パターン)
- オブジェクトの振る舞いに関するパターン(Chain of Responsibility、Command、Interpreter、Iterator、Mediator、Memento、Observer、State、Strategy、Template Method、Visitor パターン)

オブジェクト指向プログラミングについてより深く知識(抽象クラス、インターフェースの使い方)の習得を行うために、Java 教育(DB 編)で作成したユーザ管理システムを以下のデザインパターンを用いて作成する。また、Facade パターンを用いて Insert(追加)、Update(更新)、Select(取得)、Delete(削除)が簡単に実装できるようにする。

- Facade(Data Access Object:DAO)パターン
- Template Method パターン
- Factory Method パターン
- Iterator パターン

3.6　JRE の API の使い方について

参考書（Java 言語プログラミングレッスン　下）を用い、スレッドについての知識を習得する。また、Polygon 図形が回転しながら移動するアニメーションのプログラムの作成を JRE の API を用いて行う。

さらにイベントの概念を習得するために、マウス操作で Polygon 図形を操作し、平行移動、回転移動、拡大縮小できるプログラムを作成する。なお、マウス操作で発生するイベントは以下の通りである。

- mouseClicked(コンポーネント上でクリック)
- mouseEntered(コンポーネント上に入る)
- mouseExited(コンポーネント上から出る)
- mousePressed(コンポーネント上でマウスボタンを押す)
- mouseReleased(コンポーネント上でマウスボタンを離す)
- mouseDragged(ボタンを押しながら動かす(ドラッグ))
- mouseMoved(ボタンを押さずに動かす)

3.7　コーディング規約について

3.1 章～3.6 章を実施する過程において、プログラムのコーディング時にクラス名、変数

名等の命名規約について調査し、その知識習得を行う。

[4] 結果

4.1　Java 基本文法(構造化プログラミングレベル)について

以下の項目について、テキストの熟読、及び例題・練習問題を通してどのようにこれらを用いてプログラムを作成するかのスキルを習得した。

また、Java 教育(基本編)演習問題でアルゴリズムの思案後、フローチャートを作成し、プログラムの作成を行い、習得した知識・スキルをしっかりと固めることができた。

・アプリケーション：Java のプログラムはアプリケーションと呼ばれる。アプリケーションは任意のクラスを作ることができ、そのクラスの main メソッドから実行を開始する。アプリケーションは独立した 1 つのプログラムとして動作する。

・文字列出力：System.out.print(“出力したい文字列”); 又は System.out.println(“”);
出力したい文字列に「”」がある場合は、文字列内の「”」を「¥”」と記述する。

- 文字列→「” “」
- 文字→「’ ‘」

・変数：何かを入れておく箱のようなもの。変数に対してできることは、

1. 変数を作る(変数宣言)
2. 値を入れる(代入)
3. 値を見る(参照)

・演算子：・算術演算子(+ - * / % など)
・ビット演算子(& | ^ ~ など)
・論理演算子(! && || など)
・関係演算子(== != < > <= >= など)
・シフト演算子(>> << >>> など)
・インクリメント演算子(++)…指定された変数の値に 1 加算する
・デクリメント演算子(--)…指定された変数の値に 1 減算する

・if 文：「もしも…ならば」条件式は boolean 型、その値は true か false。

```
if (条件式) {
        条件が成り立つときの処理
}
```

・if …else 文：「もしも…ならば…さもなくば」true のときの処理か、false のときの処理、どちらか一方は必ず行われる。

```
if (条件式) {
```

条件式が true のときの処理

else {

条件式が false のときの処理

}

・switch 文：多くの選択肢から 1 つを選んで実行するための構文であり、図 4.1.1 に構成を示す。case として明示的にかかれていない値ならば default 以下が実行される。

・break 文：for や while の繰り返しを中断する仕組み(ループの中断)。しかし、switch 文中の break 文は繰り返しの中断ではない。

for や while が二重になっている場合、break で中断するのは、内側の for や while 文だけである。

ラベル付き break 文は二重ループ(多重ループ)から一気に脱出するときに使用する。ラベルのついたループから抜け出すことが可能となる(図 4.1.1 参照)。

ラベルは、識別子(名前)の後にコロン(：)をつけて指定する。

・continue 文：ループを次に進めるための仕組み。Break 文同様にラベル付きの continue 文がある。ラベルのついたループを次に進めることが可能となる(図 4.1.1 参照)。

switch 文の構成	ラベル付き break 文	ラベル付き continue 文

switch (式) {
case 定数式 1:
処理 1
break;
 	
case 定数式 2:
処理 2
break;
 	
default:
処理 3
break;
}
	outer:
	while (・・・){
	while(・・・) {
	if (条件式) {
	break outer;
	}
	}
	}
	対応するラベルの付いた while 文の外まで脱出
		outer:
		while (・・・) {
		while (・・・) {
		if (条件式) {
		continue outer;
		}
		}
		}
		対応するラベルの付いた while 文を次に進める

図 4.1.1　switch 文、ラベル付き break 文・continue 文

・for 文：

for (初期化; 条件式; 次の一歩) {

繰り返す処理

}

初期化は繰り返しの準備で一度だけ実行される。条件式は繰り返しを続ける条件

を表現した式で、次の一歩は繰り返す処理が一度終わった後、繰り返しを次に進めるための処理。

・while 文:ある処理を繰り返して行いたいときに使う構文(ある条件を満たす間繰り返す)。

```
while (条件式) {
      繰り返す処理
}
```

・for 文と while 文:

図 4.1.2 に for 文と while 文の流れを示す。

- for 文では、初期化を行った後…
  - ➢ 条件式のチェック
  - ➢ 繰り返す処理の実行
  - ➢ 次の一歩を実行
- while 文では、初期化の部分はないので…
  - ➢ 条件式のチェック
  - ➢ 繰り返す処理の実行

をそれぞれ繰り返し行っている。

for 文は、繰り返す回数がわかっているときにつかう繰り返し文である。

while 文は、何度繰り返すかはわからないが、繰り返し処理を行うときにつかう繰り返し文である。

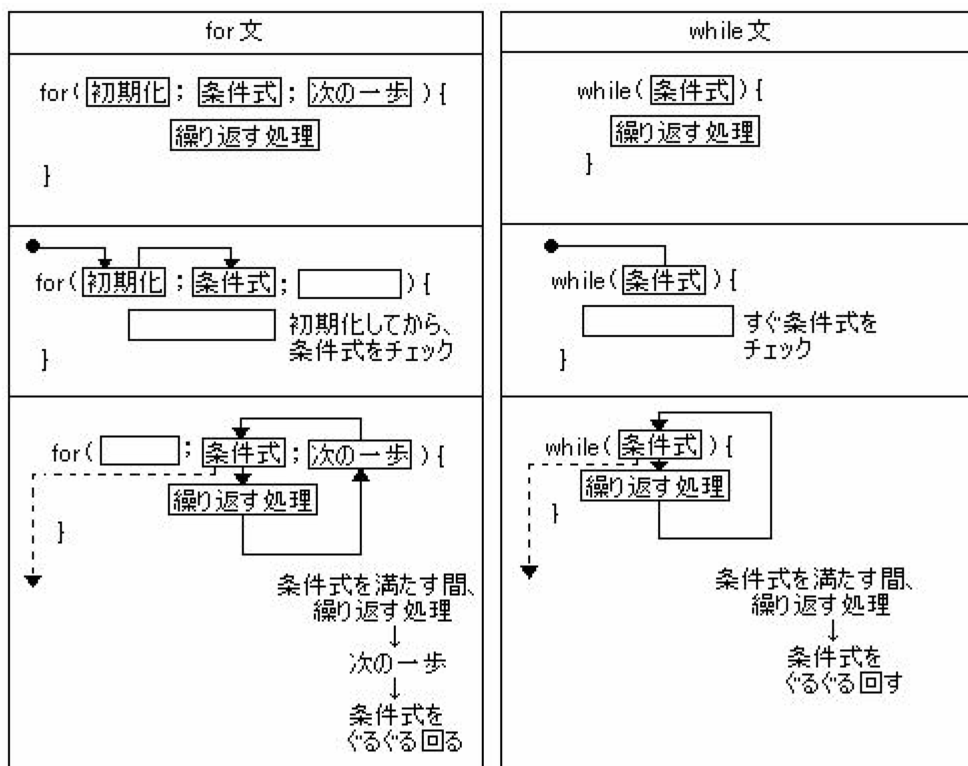

図 4.1.2 for 文と while 文の比較

・do-while 文

- while 文　……0 回以上の繰り返し(前判定型)
- do-while 文……1 回以上の繰り返し(後判定型)

```
do {
    繰り返す処理
} while (条件式);
```

・配列：変数に番号をつけて並べたもの、共通の目的を持つ複数の変数を番号をつけてまとめるという仕組み。

- new 型名[要素の個数]で配列の確保。
- 配列の要素は必ず 0 番目から始まる。
- 配列の長さは…配列.length で求まる。
- 配列の初期化…配列の型[] 配列 ={要素，要素，要素，要素，要素};
- 配列にまとめて代入…配列 =new 配列の型[] {要素，要素，要素，要素，要素};

・2 次元配列：配列の配列のことを言う。2 次元配列の場合、各要素となっている配列の長さは一定である必要はない。

4.2　アルゴリズムについて

参考 Web サイトに沿って学習を行った結果、以下の項目の知識を習得した。また、アルゴリズムの思案、及びフローチャート、プログラムの作成を行うことでアルゴリズム及び、Java 基本文法(構造化プログラミングレベル)についてのスキルを習得した。

・アルゴリズム：何か問題を解決するときの考え方のこと。

・フローチャート：アルゴリズムをわかりやすく図示したもの。

- 図で示されているので、文章だけよりもわかりやすく、プログラムにしやすい
- バージョンアップなど改良する際に効率よくできる
- １つのプログラムを複数人で作るときに、手分けしやすい
- 同じような処理は、再利用できる

・分岐構造：・単一分岐…　処理を２つに分ける。もっとも単純な分岐
　　　　　　・多重分岐…　単一分岐を複数組み合わせたもの
　　　　　　・多方向分岐…複数の条件の中から当てはまる１つの処理を実行

・反復構造：ある処理を条件を満たす間もしくは、満たすまで繰り返す構造。
　　　　　　・前判定型…処理の前に反復条件を設置
　　　　　　・後判定型…処理の実行後に反復条件を設置

・検索：目的のデータを探し出すこと。
　　　・シーケンシャルサーチ…最初からもしくは最後からデータを１つ１つ探す手法。
　　　・バイナリサーチ…上限・下限を狭めていく手法。(ただしデータが昇順に並んで

いる場合のみ可能)

・ソート：データをある順序に並べ替える。

- 選択法…最も小さな値を探し、その値を最初に持ってくることを要素分繰り返す
- 交換法…選択法を発展させ、選択した値とそれが入る順番の要素のデータを交換しながら処理する
- バブルソート…最初の値と隣接する値を比較し降順になっていなければ値を交換しながらすべてが降順になるまで繰り返す
- 挿入法…整列済みの配列に次のデータを挿入していき、データがなくなるまで繰り返す

4.3　Java 基本文法(オブジェクト指向型プログラミングレベル)について

参考書、参考 Web サイトに沿って、例題・練習問題でプログラム作成を繰り返し行い学習した結果、以下の項目についての知識、スキルを習得した。

・フィールド：型と名前を持つ

int field…インスタンスフィールド/static int field…クラスフィールド

・メソッド：型と名前と引数列を持つ

int method(引数列){	static int method(引数列){
……	……
} インスタンスメソッド	}　クラスメソッド

・スーパークラスのコンストラクタの呼び出し：super(“引数”);

・自分のクラスのコンストラクタの呼び出し：this(“引数”);

・オーバーライド(override)：継承したフィールドやメソッドを変更すること。スーパークラスのメソッドと同じメソッドを宣言することで行われる(メソッド名、引数列が等しい)。

・Throwable：throw 文で投げることができ、catch 文で受け止められるクラス。
Throwable は Error、Exception に分けられる。

・Error：もはや動作を継続するのは期待できないときに投げられるクラス。

・Exception：正しく例外処理を行って、動作が継続することを期待するときに投げられるクラス。
Exception は RuntimeException、RuntimeException 以外の Exception に分けられる。

・RuntimeException：実行中に起こって、コンパイラにより前もってチェックされない例外を表すクラス。

・上以外の Exception：コンパイラによって前もってチェックされる例外を表すクラス。

・try…catch…finally：

- try ブロックの中で例外が投げられてもそうでなくても、finally ブロックは実

行される。

- try ブロックの中で return する場合でも実行される。
- try ブロックで行った事の後始末を書く。
- return ブロックを書いてはいけない。

・メソッドの中でエラーが起きたことを示す：

(1) 戻り値でエラーを示す。

(2) 例外でエラーを示す。

(3) 上とは別の方法でエラーを示す。

(1)の例として、java.io.InputStreamReader というクラスの read メソッド。

(2)の例として、java.io.DataInputStream というクラスの readChar メソッド。

(3)の例として、java.io.PrintWriter というクラス。

エラーは戻り値や例外でなく、checkError メソッドを呼び出し調べる。

・printStackTrace：コールスタックが表示され、どのメソッドの中から例外が投げられたのかはっきりわかることが可能。

・クラスとインターフェースの相違点、類似点：クラスとインターフェース概要を表 4.3.1 に示す。

表 4.3.1　クラスとインターフェースの比較

	クラス	インターフェース
インスタンス	生成可	生成不可
メソッド	様々	public abstract
フィールド	様々	public static final
スーパークラス	1つ	なし
スーパーインターフェース	複数指定可能(implements)	複数指定可能(extends)

・抽象クラスとインターフェースの類似点および相違点：

・類似点

- インスタンスを生成不可。
- 抽象クラスはサブクラスに対して、インターフェースは実装クラスに対して、メソッドの実装を要請する。

・相違点

- 抽象クラスはクラス階層の一ヶ所にしか位置できないが、インターフェースは階層のあちこちに実装クラスを作成可能。
- 抽象クラスは属しているメソッドのうち少なくとも 1 つが抽象メソッドであれば他のメソッドは抽象メソッドでなくても良い。また、フィールドは定数でなくても良い。インターフェースはすべてのメソッドが抽象メソッドであり、フィールドは全て定数。

・インヘリタンス(継承)とオーバーライド：Java ではあるクラスをもとに、機能を拡張し

たり、変更したクラスを定義できる。もとのクラスをスーパークラス、それをもとに新しく定義したクラスをサブクラスと呼ぶ。スーパークラスのメソッドは基本的に全てサブクラスにそのまま引き継がれる(継承される)が、一部のメソッドだけ再定義(オーバーライド)して機能を変えることができる。

・抽象データ型と具体的なデータ型：一般に「集合」「列」「写像」といった抽象的なデータ(やりたい事・仕様)と「線形リスト」「配列」「ハッシュ表」「木構造」といった具体的なデータ構造(どうやって実現するか・実装)を切り離しておいた方が変更に強く、再利用しやすいプログラムを作れる。Java では抽象的なデータを表すためにインターフェースを用いることが多い。抽象的なデータの実装(implementation)はインターフェースを implement したクラスによって表す。

・コレクション：Java では「集合」「列」「写像」などのデータの集まり(コレクション)を表すインターフェースが用意されている。

- java.util.Set：集合(重複のないコレクション)を表すインターフェース

実装するクラス…HashSet、TreeSet、LinkedHashSet

- java.util.List：列(ある順序で並んだ重複を許すコレクション)を表すインターフェース

実装するクラス…ArrayList、LinkedList

- java.util.Map：写像(キーから値への対応づけの場合)を表すインターフェース

実装するクラス…HashMap、TreeMap

4.4 Java による DB 操作および SQL について

以下の項目についての知識を習得し、その知識を用いて演習問題(Java 教育(DB 編))を行うことで、DB 及び SQL の基本を習得した。簡易ユーザ管理システムの構築を行いプログラム作成のためのスキル習得を行えた。なお、演習問題は、DB で管理されているユーザ情報を操作するシステムの開発である。

・DB：

- 問い合わせ言語：クエリー言語 SQL(Structed Query Language)という言語が標準。
- SQL：データの取り出し(検索)だけでなく、テーブルの作成、設定、データの追加・削除・更新のような操作もサポートしているが、データベースへ問い合わせるための専用言語であるのでアプリケーションを作れない。

・テーブルに情報追加(Insert)：mysql> INSERT INTO テーブル名 VALUES ('各カラムに追加したい情報・値');

・テーブルの情報取得(Select)：mysql> SELECT カラム名 FROM テーブル名 WHERE 取得対象レコードの満たすべき条件;

・テーブルの情報更新(Update)：mysql> UPDATE テーブル名 SET カラム名 = '修正内容' WHERE 最初のカラム名 ='入力されている内容';

・テーブルの情報削除(Delete)：mysql> DELETE FROM テーブル名 WHERE カラム名 = '入力してある内容';

・Java から DB へアクセスするプログラミング：

1.JDBC ドライバの登録：DriverManager クラスを使用し、利用する DBM を登録する。(登録されたドライバはメモリ上にロードされる。)

2.DB へアクセスするための SQL 文を変数に代入。

3.DB とのコネクションの確立：DB への接続には Connection オブジェクトを使用。(メソッド DriverManager.getConnection か DriverManager の代替として DataSource インターフェース利用可)登録したドライバがデータソースを認識/接続確立できるように、データソースを識別する URL を指定しておく。

4.SQL ステートメント・オブジェクトの作成：DB を操作するための SQL 文を DB へ送るために、PreparedStatement オブジェクトを使用、SQL 文の実行のためのコンテナの役割をする。

5.Select(取得)の場合：ResultSet の受信：ResultSet は発行された SQL 文の実行結果を含む Java オブジェクト。指定されたデータ型で返され Java アプリケーション内で処理される。

Insert(追加)、Update(更新)、Delete(削除)の場合：executeUpdate()の実行

6.DB のクローズ：必要がなくなったらできるだけ早く明示的に全ての Statement をクローズ、全ての Connection をクローズする。ガーベッジコレクションが暗黙的にリソースを開放する前に不要なリソースを開放し、結果的にパフォーマンス/可用性を向上させる。

4.5 デザインパターンについて

以下の項目について知識の習得を行い、また演習問題_デザインパターン編で、デザインパターンを用いてプログラミングを行うことで、抽象クラスやインターフェースを利用した再利用性の高いプログラムの作成の知識、及びスキルの習得ができた。

・UML クラス図：クラス図は図 4.5.1 の様に記述する。

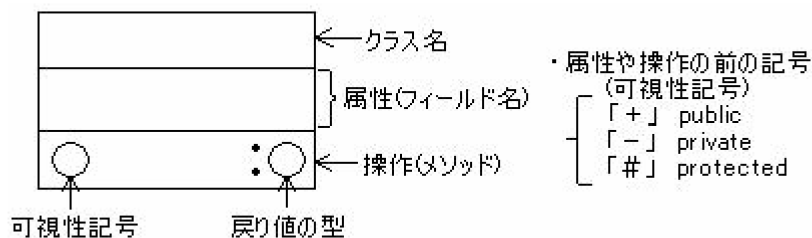

図 4.5.1 UML クラス図

・GoF デザインパターン：GoF：The Gang of Four(4 人の奴等)デザインパターン考案者、エリック・ガンマ、リチャード・ヘルム、ラルフ・ジョンソン、ジョン・ブリシディーズの 4 人を指す。全 23 種類のパターンがある。

・Mediator(オブジェクトの関連を整理する)パターン

・Iterator(オブジェクト間の関連をシンプルにする)パターン：for 文で i++と i を 1 ずつ増加させていくと、現在注目している要素を「次」「その次」「そのまた次」と進めていることになる。要素全体を最初から順番にスキャン(走査)していることになる。ここで使われている変数 i の働きを抽象化し、一般化したもののこと。図 4.5.2 にクラス図を示す。

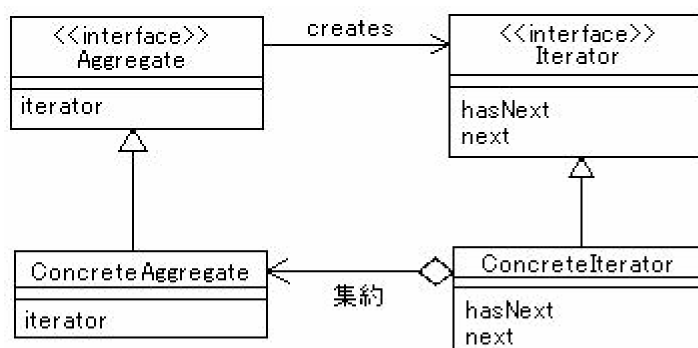

図 4.5.2　Iterator パターンのクラス図

関連しているパターンとしては、Visitor、Composite、Factory Method パターン。

・Adapter(つながらないクラスをつなげてしまう)パターン：変換コネクタのようなもので直接つながらないコネクタと差込口であっても変換コネクタを置くことで結び付けられる。コネクタと差込口には改造が不要である。

・Factory Method(一緒に使ってほしいオブジェクトを生成する)パターン：Template Method パターンでは、スーパークラス側で処理の骨組みを作り、サブクラス側で具体的な処理の肉付けを行った。このパターンをインスタンス生成の場面に適用したパターン。図 4.5.3 にクラス図を示す。

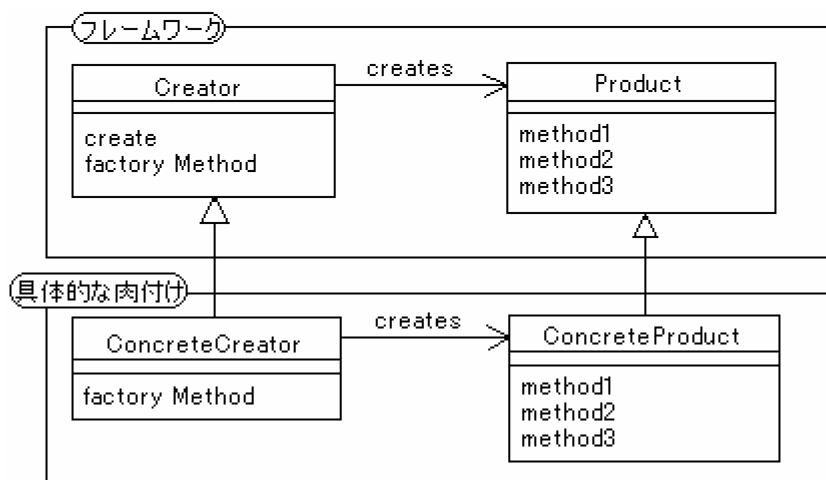

図 4.5.3　Factory Method パターンのクラス図

関連しているパターンとしては、Template Method、Singleton、Composite、Iterator パターン。

・Abstract Factory(プログラムの一部を丸ごと変換する)パターン

・Facade(1 つのメッセージを複数のクラスで処理する)パターン：大きなプログラムを使って処理を行うためには、関係しあっているたくさんのクラスを適切に制御しなければならないので、その処理を行うための「窓口」を用意しておくのが便利である。たくさんのクラスを個別に制御しなくても、その「窓口」に対して要求をだすだけで仕事がすむ。その「窓口」を用意しておくこと。

関連しているパターンとしては、Abstract Factory、Singleton、Mediator パターン。

＊基礎に「ファサードによる層の分割」があり、その分割された層のうちの「データアクセス層」は、Java に特化したパターンとして、サーバサイド Java のパターンを集めた J2EE パターンの中の DAO パターンとなる。

・DAO(Data Access Object)パターン：データアクセス部分をまとめて整理整頓するためのもので図 4.5.4 にクラス図を示す。

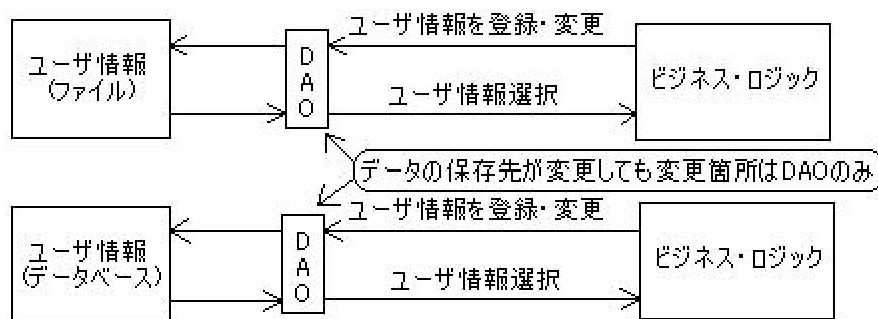

図 4.5.4　DAO パターンのクラス図

・Chain of Responsibility(1 つのメッセージを伝言ゲームのように処理する)パターン

・Builder(様々なオブジェクトを作ってくれる)パターン：様々なバリエーションでオブジェクトを生成する「ビルダクラス」と「ディレクタクラス」を用意しておく。このパターンはカプセル化(データの処理手順の隠蔽)することがポイントである。

・Prototype(自分のクローンを生成する)パターン：同じクラスでいくつかのバリエーションのオブジェクトを作るときに便利。あらかじめそれぞれのバリエーションのオブジェクトを 1 つずつ作っておき、このオブジェクトがプロトタイプ(原型)で必要に応じてプロトタイプのクローンを作っていく。クラスの中に自分のクローンを返すメソッドを用意する。クローンとはフィールドの値が等しいオブジェクトのことである。

・State(if 文を使わずに状態に応じた処理を行える)パターン：状態における処理を if 文を使わずに、状態を表す複数のクラスを作る。

・Observer(複数のオブジェクトに状態の変化を通知する)パターン

・Singleton(オブジェクトを 1 つだけしか生成させない)パターン：オブジェクトの数を 1 つではなく、2 つや 3 つのように特定の数に制限できる。static の場合には 1 つに制限す

ることしかできない。オブジェクトの生成タイミングを制御できる。

- Flyweight(軽量オブジェクトを使いまわす)パターン：小さなオブジェクトを数多くロードする状況では、ロード済みのオブジェクトを使いまわした方がプログラムのパフォーマンスが向上する。
- Bridge(拡張から実装に橋を架ける)パターン
- Decorator(オーバーライドせずに機能を追加する)パターン
- Composite(再帰とインターフェースを応用する)パターン
- Proxy(インターフェースとメッセージフックを応用する)パターン：メッセージとはオブジェクトから他のオブジェクトに送られる通知のこと。メソッド呼び出しがメッセージに相当する。このパターンではオブジェクトに送られるメッセージをフックして任意の処理を挿入するためのもの。そのためフック用のクラスを作る。
- Command(命令 1 つのオブジェクトで表す)パターン：命令を表すオブジェクト(コマンドオブジェクト)をメソッドの引数にする。オブジェクトはデータと処理の集合体なので 1 つの引数だけで複数のデータとその処理方法を伝えられる。
- Strategy(アルゴリズムを汎化して交換可能にする)パターン：プログラムの実行時にアルゴリズムを切り替える。クラスを使う人のコードを変更することなくアルゴリズムを変えられる。
- Template Method(複数の抽象メソッドを呼び出すメソッドを作る)パターン：スーパークラスで処理の枠組みを定め、サブクラスでその具体的内容を定める。
  関連しているパターンとしては、Factory Method と Strategy パターン。
- Visitor(2 つのオブジェクトを 1 つに結びつける)パターン：オブジェクトに動的に機能を追加する。プログラムの実行時にオブジェクトに何らかの機能を追加する。
- Interpreter(ツリー構造で構文解析結果を表す)パターン：言語構文の追加に用意に対応する。
- Memento(形見を残してアンドゥ機能を実現する)パターン：オブジェクトの現在の状態を保存し、後で元の状態に復元するための工夫。
- 集約：「持っている」関係を「集約」(Aggregation)と呼ぶ。インスタンスを持っていれば個数に関わらずその関係は集約である。

4.6　JRE の API の使い方について

以下の項目についての知識、API の調査方法の習得を行い、スレッドや必要な API を用いてプログラミングを行うことでスキルの習得をできた。

- アプレット：applet は他のアプリケーション上で動作する小さなプログラムである。「他のアプリケーション」というのは通常は、Microsoft Internet Explorer のような Web ブラウザのこと。アプレットは JDK に付属しているアプレットビューア(appletviewer)と

いうアプリケーション上でも動作する。java.applet パッケージの Applet クラスのサブクラスを作らなければならない。アプレットは init メソッドで初期化され、start メソッドで実行を開始する。アプレットを読み込んで動かすために HTML ファイルが必要になる。

・アプレットの作成と実行：

① Applet クラスの拡張クラスを作る(import java.applet.Applet;)

② init/start/stop/destroy メソッドを書く

- init メソッドは初期化
- start メソッドは起動
- stop メソッドは停止
- destroy メソッドは終了処理

③ その他必要なメソッドを書く

④ コンパイルをする

⑤ HTML ファイルを作る

⑥ アプレットビューア又はブラウザで HTML ファイルを読み込む

・AffineTransform クラス：線の直線性と平行性を保ったままで 2 次元座標間の線形マッピングを実行する 2 次元アフィン変換を表現する。アフィン変換は一連の平行移動、スケーリング、反転、回転、変形により構成される。

・Graphics2D クラス：Graphics クラスを拡張して、ジオメトリ、座標変換、カラー管理、およびテキスト配置について高度な制御を行う。このクラスは Java プラットフォームで 2D の図形、テキスト、およびイメージを描画するための基本クラス。

・GeneralPath クラス：直線、2 次曲線、および 3 次(ベジェ)曲線から作成されたジオメトリックパスを持てる。

・スレッドを作る方法：スレッドの作成は以下の 2 つの方法がある。この 2 つの方法の概要を表 4.6.1 に示す。

1. Thread クラスの拡張クラスを作る
2. Runnable インターフェースを実装したクラスを作る

表 4.6.1　スレッドの作成方法

	Thread の拡張クラス	Runnable の実装クラス
用途	簡単なクラス向け	複雑なクラス向け
スーパークラス	Thread クラス	何でも可
クラスの宣言	Thread クラスの拡張	Runnable インターフェースの実装

スレッドの起動	class (クラス名) extends Thread{
	・・・・・
	}
		class (クラス名) implements Runnable{
		・・・・・
		}

実行開始点	run メソッド	run メソッド

1 の方法では、スーパークラスは 1 つしか持てないので、Thread クラスの拡張クラスに

してしまうと、他のクラスの拡張クラスにすることは不可能なのでクラス階層の下の方に位置するクラスをスレッドにすることができない。2 の方法は 1 の方法の問題を解決するために用意されている。クラス階層の下の方に位置するクラスでも Runnable インターフェースの run メソッドを実装することでスレッド化が可能。スレッドを実際に動かすには別途 Thread クラスのインスタンスを生成して、その start メソッドを呼び出す必要がある。

・repaint()メソッド：paint()メソッドは内部的に呼び出されるメソッドなのでプログラム内で直接呼び出すことはできない。代わりに repaint()メソッドを呼び出すと強制的にアプレットの再描画の依頼を出すことができる。その結果 paint()メソッドが呼び出されることになる。(repaint メソッドで update メソッドを呼び出すとコンポーネントが全て背景色で消されてしまう)

1. repaint をプログラム内で呼び出す。
2. update メソッドが内部的に呼び出され画面をクリアした後で paint メソッドを呼び出す。
3. paint メソッドによってアプレットが描画される。

・ダブルバッファリング：画像更新の画面のちらつきを回避するための技術。画面がちらつくのはコンポーネント更新時に一度背景色で塗りつぶすためである。この現象は update メソッドをオーバーライドすることで回避できるが画面を何かつけ足していくようなプログラムの場合はこれで解決可能だが、画面の一部を動的に更新する場合には、この方法だけでは解決できないので、ダブルバッファリングを用いる。

- オフスクリーンという見えないウィンドウを作る
- プログラムは見えないウィンドウを更新する
- オフスクリーンをコンポーネントに drawImage で描画することで画面のクリアからイメージ操作の全ての過程をウィンドウに写すことなく全ての作業を終えてから表示することでちらつきを防止可能となる。

＊オフスクリーンとなるバッファの生成：Component クラスの createImage()メソッドを使用。

```
public Image createImage(int width, int height)
```

width、height はオフスクリーンとなるバッファのサイズを指定。このオフスクリーンを操作するためにはグラフィックコンテキストが必要必要なので、グラフィックコンテキストを得るために Image クラスの getGraphics()メソッドを使用。

```
public abstract Graphics getGraphics()
```

オフスクリーンイメージに描画するためのグラフィックコンテキストを作成する。

・イベント：ユーザ動作のこと。イベントの種類は様々でボタンを押したり、キーボードを押したりなどである。Java のイベントモデルに代行(委譲)イベントモデルが存在する。イベント発生の対象は Applet クラス、つまりアプレットそのもので、イベントを受ける

のには、Component クラスのメソッドを利用する。
例えばマウスイベントを受けるのには、addMouseListener() メソッドや addMouseMotionListener()メソッドを利用する。addMouseListener()メソッドには MouseListener インターフェースを実装しているクラスのオブジェクトを渡す。これはコンポーネントからのイベントの監視を追加するもので、MouseListener インターフェースで定義できるイベントタイプは次の通りである。

- public abstract void mouseClicked(MouseEvent e)→コンポーネント上でクリック。
- public abstract void mouseEntered(MouseEvent e)→コンポーネントに入る。
- public abstract void mouseExited(MouseEvent e)→コンポーネントから出る。
- public abstract void mousePressed(MouseEvent e)→コンポーネント上でマウスボタンを押す。
- public abstract void mouseReleased(MouseEvent e)→コンポーネント上でマウスボタンを離す。

addMouseMotionListener()メソッドには MouseMotionListener インターフェースを実装しているクラスのオブジェクトを渡す。これは、addMouseListener()ではマウスの動きを追う、つまりドラッグ対応できないのでその点をカバーするもので、MouseMotionListener インターフェースで定義できるイベントタイプは次の通りである。

- public abstract void mouseDragged(MouseEvent e)→ボタンを押しながらマウスを動かす(ドラッグ)。
- public abstract void mouseMoved(MouseEvent e)→ボタンを押さずにマウスを動かす。

インターフェースを実装すると、たとえ必要としないイベントであっても空実装しなくてはコンパイルできなくなる。この空実装は時として非常に面倒で、さらにソースも長くなり、可読性の低下にもつながる。そこでアダプタクラスなるものが各イベントには用意されている。

- アダプタクラス : 目的のインターフェースをインプリメントしているクラスでこのクラスを拡張し、メソッドをオーバーライドすることで目的のプログラムを作ることができる。
  - MouseAdapter クラス : MouseListener クラスをインプリメントしている。
  - MouseMotionAdapter クラス : MouseMotionListener クラスをインプリメントしている。

しかし、アダプタクラスを用いても、クラスをもう 1 つ定義しなければならず、アダプタクラスを使用しないのとあまり変わらなくなってしまう。そこで内部クラスを使用する。内部クラスとアダプタクラスを使用することで上で示したような欠点が解決するのである。

(例)addMouseListener()メソッドを使用した場合

```
addMouseListener(new MouseAdapter() {
        public void 〜実装したいメソッド {
                ・・・・・
        }
});
```

＊内部クラスの利用法としてはイベントハンドラが代表される。

4.7　コーディング規約について

コーディング規約を熟読し、各演習問題で以下に示すことを考慮しながらコーディングすることで、コーディング・スキルの習得ができた。

・コーディングの心得5カ条：

1. 見易さを重視する：良いコードは他の人が読んでもわかりやすいと感じられるコードである。フォーマット、ロジックの簡潔さ API の常識的な使い方によって、良いコードは作られる。
2. ネーミングはわかりやすくする：ネーミングは対象の本質を表すような名前を考える作業である。
3. サンプルを鵜呑みにはしない：サンプルコードを活用する場合には、そのコードの内容や背景をしっかり理解することが必要である。
4. 同じコードを2度書かない:同じコードが現れるようならまとめて1つにして、外に出してコールするような書き方をすべきである。
5. 役割は1つにする：メソッドの役割が明確かつ1つであれば、単体テストが行いやすくなり、コードの試験性が向上する。また、コード変更の際に修正箇所がわかりやすいので障害修正に要する時間が短くなるので、コードの保守性が向上する。

・全般：名前をつけるときは全て英語を基本とする。

・パッケージ名：

- 特に取り決めがない場合、パッケージ名は全て小文字で統一する。
- できるだけ、内容が連想可能名前にする。
- 多少長くなってもできるだけ省略せずにわかりやすい名前を使用する。

・クラス名：

- 役割を表すような意味のある文字列を使用する。
- 単語の先頭を大文字にする。
- インターフェース名、抽象クラス名、実装クラス名はクラス名に準ずるものにする。

・メソッド：

- メソッド名は 1 つの単語の場合は全て小文字で、複数の単語での構成の場合は区切りのを大文字にする。
- オブジェクトを生成するメソッドは”create”＋オブジェクト名にする。
- 変換メソッドは”to”+オブジェクト名にする。
- ゲッターメソッドは”get”+属性名にする。
- セッターメソッドは”set”+属性名にする。
- boolean 変数を返すメソッドは true/false の状態がわかるようにする：Yes、No を表す疑問形の形式”is”+名詞、”can”+名詞や”has”+名詞を用いる。

・変数全般：

- boolean 変数は true/false の状態がわかるようにする：Yes、No を表す疑問形の形式”is”+名詞、”can”+名詞や”has”+名詞を用いる。
- 定数は全て static final 宣言をし、変数名は全て大文字、定数名が複数の単語で構成されている場合は、各単語の間は”_”で区切る。

・コーディング規約全般：

- 使われていない private メソッド、変数、あるいはローカル変数はコードの可読性を低下させるので必要ないものは削除するようにする。
- 継承されないクラス、オーバーライドされないメソッド、値の変わらない変数等、変化のないもの/変化させたくないものは final で宣言する。

[5] 結論

以下の項目についての必要な知識の習得、及びプログラム作成を繰り返し行ったことで現場レベルでのプログラミングが行えるための基礎知識及び基礎スキルの習得ができた。今後、更に知識・スキルを高めていき、現場レベルでのプログラミングが行えるように精進する。

- Java 基本文法(構造化プログラミングレベル)
- アルゴリズム
- Java 基本文法(オブジェクト指向型プログラミングレベル)
- Java による DB 操作および SQL
- デザインパターン
- JRE の API の使い方
- コーディング規約

[6] 参考文献

参考書

・新 Java 言語入門　ビギナー編
・Java 言語プログラミングレッスン　上
・Java 言語プログラミングレッスン　下
・Java 言語で学ぶデザインパターン入門
・やさしい Java　入門編
・コンピュータ・グラフィックスの基礎

参考 Web サイト

・目指せプログラマー　[http://www5c.biglobe.ne.jp/~ecb/index.html]
・MySQL4.1 リファレンスマニュアル
　[http://dev.mysql.com/doc/refman/4.1/ja/index.html]
・MySQL クイック・リファレンス
　[http://www.bitscope.co.jp/tep/MySQL/quickMySQL.html]
・ソフトウェア構成論　[http://www.ialab.is.tsukuba.ac.jp/~maeda/class/06/sc/]
・矢沢久雄の早わかり GoF デザインパターン
　[http://itpro.nikkeibp.co.jp/article/COLUMN/20051201/225570/]
・サルでもわかる 逆引きデザインパターン
　[http://www.nulab.co.jp/designPatterns/designPatterns1/designPatterns1-1.html]