

知能プログラミング演習II 課題2

グループ8

29114003 青山周平

2019年10月29日

提出物 29114003.pdf, group08.zip

グループ グループ8

メンバー	学生番号	氏名	貢献度比率
	29114003	青山周平	null
	29114060	後藤拓也	null
	29114116	増田大輝	null
	29114142	湯浅範子	null
	29119016	小中祐希	null

1 課題の説明

必須課題 2-1 Matching クラスまたは Unify クラスを用い、パターンで検索可能な簡単なデータベースを作成せよ。

必須課題 2-2 自分たちの興味ある分野の知識についてデータセットを作り、上記 2-1 で実装したデータベースに登録せよ。また、検索実行例を示せ。

発展課題 2-3 上記システムの GUI を作成せよ。

2 発展課題 2-3

上記システム (Matching クラスまたは Unify クラスを用いた, パターンで検索可能な簡単なデータベース) の GUI を作成せよ.
データの追加, 検索, 削除を GUI で操作できるようにすること.
登録されたデータが次回起動時に消えないよう, 登録されたデータをファイルへ書き込んだり読み込んだりできるようにすること.

私の担当箇所は, 発展課題 2-3 の GUI 全般の Swing を用いた実装である.

2.1 手法

GUI を実装するにあたり, 以下のような方針を立てた.

1. データベースとデータのやりとりをするためのクラスやメソッドを作る.
2. 検索・追加・削除のためのテキストフィールドやボタン, リストを表示する.
3. 表示した各種コンポーネントを動作させる. データベースからデータを受け取って GUI に反映する.

1. に関して, 班員と協力してタスクを分割し, データベースとの直接のやり取りは Presenter クラスに任せた. 自分は Presenter からデータを受け取るための View クラス等を作成して用いることで, より構造化されたデータのやり取りを可能とした.

2. に関して, GridBagLayout を用いてコンポーネントの配置を行うことで, ユーザがより直感的に利用できるよう工夫した. また, データベースの一覧を表示することで, データの追加・削除・検索の視覚的な確認を行えるような仕様とした. また, 削除を一覧から選択して実行できるように, 一覧の表示には JList クラスを用いた.

3. に関して, View クラスを介することで, コンソールを通じて GUI に正しく反映できているかを確認できるような仕様とした. また, ボタンを押したと同時に GUI 上のリストを更新するために, DefaultListModel クラスを利用した.

2.2 実装

実装にあたり，主に下記のサイトを参考にした．

TATSUO IKURA：『Swingを使ってみよう - Java GUI プログラミング』 <https://www.javadrive.jp/tutorial/> (2019/10/29 アクセス)

GUIに大きく関連するプログラムとして，UnifyGUI.java, Presenter.java, TextModel.java, ViewInterface.java が挙げられる．各プログラムの説明については以下の通りである．

UnifyGUI.java には以下のクラスが含まれる．

- SearchGUI: メソッド main, actionPerformed, クラス myListener を実装したクラス．
- View: インターフェース ViewInterface を実装したメソッド，各種ゲッターを実装したクラス．

Presenter.java には以下のクラスが含まれる．

- Presenter: メソッド start, finish, addData, searchData, deleteData, fetchData を実装したクラス．

TextModel.java には以下のクラスが含まれる．

- TextModel: データベースの ID とテキストを一元的に保持するためのクラス．ID とテキストのゲッターを実装している．

ViewInterface.java には以下のクラスが含まれる．

- ViewInterface: メソッド successStart, successFinish, successAddData, showSearchResult, successDeleteData, showResultList, showError, showNoData を持つインターフェース．

2.2.1 データベースとデータのやりとりをするためのクラスやメソッドを作るまで

他の班員が作った Presenter クラスを介してデータを受け取るために，まず ViewInterface インターフェースを実装する必要があった．初めは

UnifyGUI 自体に実装しようと考えたが、Presenter のインスタンスの引数に ViewInterface を実装したクラスを渡す必要があったため、こうすると UnifyGUI の main メソッドを実行しようとしたとき、static であるため自身を引数として渡すことができなくなるという問題が発生した。

そこで、ViewInterface を実装するためのクラスとして View クラスを別に作った。View クラスでは Presenter とのやり取りのたびにコンソールに結果が出力されるため、デバッグ等で活用しやすいものにできた。

また、Presenter クラス側から、プログラム開始時に start メソッド、終了時に finish メソッドを呼び出してほしいという要求があったため、WindowAdapter クラスを拡張した myListener において、windowOpened メソッドと windowClosing メソッドの実装によりこれらを実現した。これにより、ウィンドウが最初に表示されたときに start メソッドが呼び出され、ウィンドウを閉じようとしたときに finish メソッドが自動的に呼び出されるように実装した。myListener クラスをソースコード 1 に示す。

ソースコード 1: myListener クラス

```
1 public class myListener extends WindowAdapter {
2     public void windowOpened(WindowEvent e) {
3         view = new View();
4         presenter = new Presenter(view);
5         presenter.start();
6         presenter.fetchData();
7         textList = view.getRl();
8         for (TextModel text : textList) {
9             lModel.addElement(text);
10        }
11    }
12
13    public void windowClosing(WindowEvent e) {
14        presenter.finish();
15    }
16 }
```

2.2.2 検索・追加・削除のためのテキストフィールドやボタン、リストを表示するまで

検索や追加には入力が必要なため、JTextField を用いて入力を可能とし、JButton クラスで対応するボタンの表示を行った。また、検索結果の

表示と、データベースの要素の表示には JList クラスを用いた。

これらのコンポーネントの表示には、ユーザが直感的に扱いやすくするためのレイアウトを考える必要があった。そこで今回の形式に最も適しているような GridBagLayout クラスを用いることとした。また、細かな配置を行うために GridBagConstraints クラスを用いた。このクラスのフィールドである gridx や gridy でセルを設定したり、anchor でセル内の配置を調整することで、ユーザにわかりやすいレイアウトの構築ができた。これをソースコード 2 に示す。

ソースコード 2: UnifyGUI コンストラクタの一部

```
1      gbc.gridx = 0;
2      gbc.gridy = 0;
3      layout.setConstraints(text, gbc);
4
5      gbc.gridy = 1;
6      gbc.anchor = GridBagConstraints.NORTHEAST;
7      layout.setConstraints(btnPanel, gbc);
8
9      gbc.gridy = 2;
10     gbc.anchor = GridBagConstraints.CENTER;
11     layout.setConstraints(searchSp, gbc);
12     ...
13     gbc.gridx = 1;
14     gbc.gridy = 0;
15     gbc.gridheight = 3;
16     layout.setConstraints(listSp, gbc);
17
18     gbc.gridy = 3;
19     gbc.gridheight = 1;
20     gbc.anchor = GridBagConstraints.NORTHEAST;
21     layout.setConstraints(delButton, gbc);
```

2.2.3 表示した各種コンポーネントを動作させたり、データベースからデータを受け取って GUI に反映したりするまで

ボタンが押下されたときに入力データを引数として Presenter からメソッドを呼び出すことを、ActionListener インターフェースの actionPerformed メソッドの実装により実現した。このとき、どのボタンが押されたかを判別するために、getActionCommand メソッドを用いた。

削除においては、リストから選択して行えるように、getSelectedIndex メソッドを利用して、どの項目が選択されているかの取得を行い、Presenter への引数として渡した。

これらを行う actionPerformed メソッドをソースコード 3 に示す。

ソースコード 3: actionPerformed メソッド

```
1 public void actionPerformed(ActionEvent e) {
2     String cmd = e.getActionCommand();
3     String arg = text.getText();
4
5     if (cmd.equals("検索")) {
6         presenter.searchData(arg);
7         sModel.clear();
8         searchList = view.getSr();
9         for (String text : searchList) {
10             sModel.addElement(text);
11         }
12
13     } else if (cmd.equals("追加")) {
14         presenter.addData(arg);
15         lModel.clear();
16         presenter.fetchData();
17         textList = view.getRl();
18         for (TextModel text : textList) {
19             lModel.addElement(text);
20         }
21
22     } else if (cmd.equals("削除")) {
23         if (!listPanel.isSelectionEmpty()) {
24             int index = listPanel.getSelectedIndex();
25             TextModel val = (TextModel) listPanel.
                getSelectedValue();
26             presenter.deleteData(val.getUUID());
27             lModel.remove(index);
28         } else {
29             System.out.println("削除失敗(未選択のため
                ) ");
30         }
31     }
32 }
```

このソースコードに示されたように、リストへの表示は DefaultList-

Model クラスの add メソッドを用いて行った。また、ソースコード 1 の start メソッド内でも同様の処理が行われており、リストの初期状態、すなわちデータベースの初期状態がウィンドウ表示時に反映されている。

また、このソースコードから分かるように、データの要求は Presenter に対して行っている一方で、データの受け取りは View を介して行っている。これはデータベースの保守性を高めるための、Presenter クラス担当者の意向によるものである。

もしリストの中身が多くなり、画面内に全ての要素が一度で表示しきれないときにスクロールバーを表示するために、JScrollPane クラスを用いた。JList インスタンスをこのコンポーネント内に入れることで、期待通りの実装ができた。この際、コンポーネントを思った通りのサイズにするために、setPreferredSize メソッドで調整した。これをソースコード 4 に示す。

ソースコード 4: UnifyGUI コンストラクタの一部

```
1      sModel = new DefaultListModel();
2      searchPanel = new JList(sModel);
3      JScrollPane searchSp = new JScrollPane();
4      searchSp.getViewPort().setView(searchPanel);
5      searchSp.setPreferredSize(new Dimension(200,
           300));
6      ...
7      mainPanel.add(searchSp);
```

2.3 実行例

UnifyGUI を実行したところ、下図のような画面が得られる.

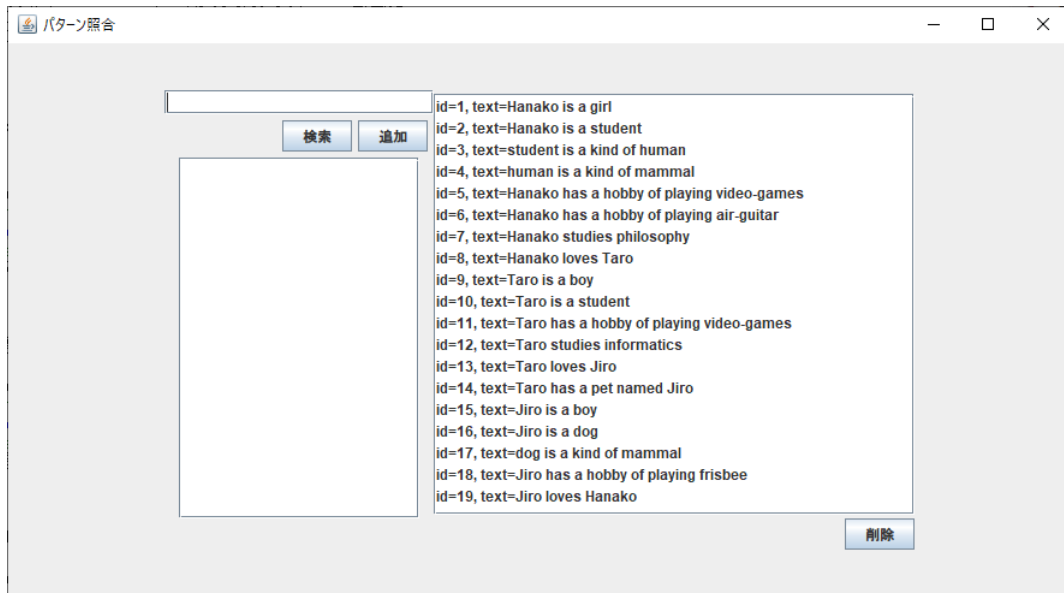


図 1: 初期状態

Matching に関する質問文「?x has a hobby of playing video-games」を入力し、検索ボタンを押したところ、下図のような画面が得られる。

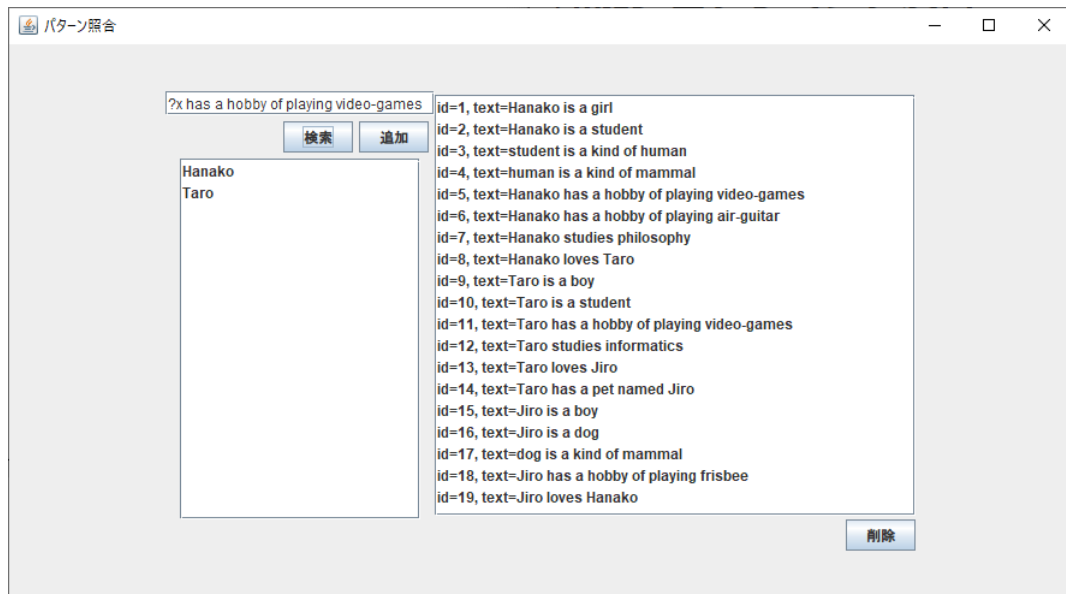


図 2: Matching の検索

Unifyに関する質問文「?x is a boy,?x loves ?y」を入力し，検索ボタンを押したところ，下図のような画面が得られる．

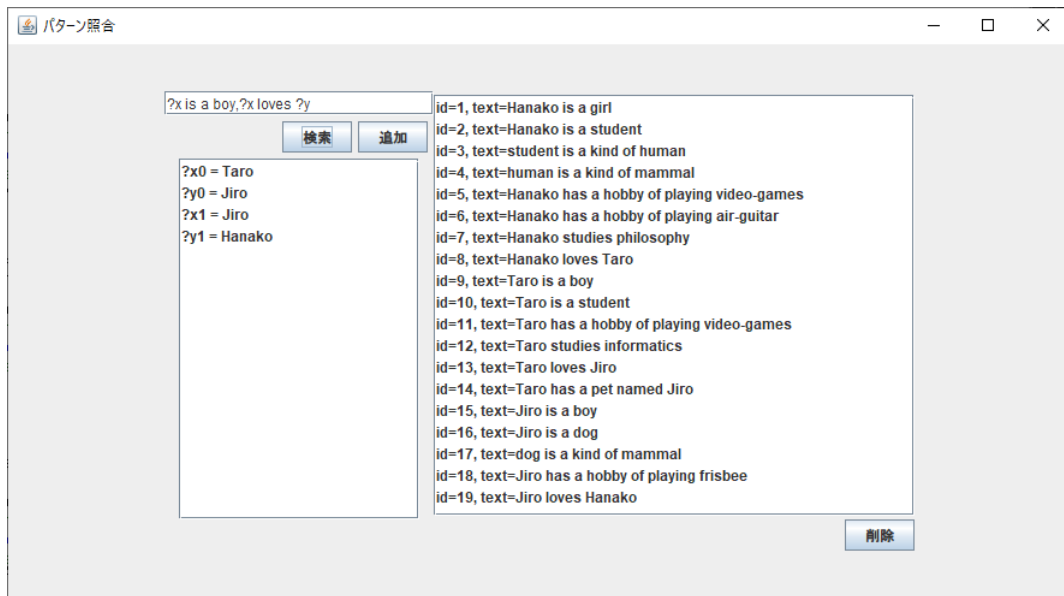


図 3: Unify の検索

データ「Shuheï is a boy」を入力し、追加ボタンを押したところ、下図のような画面が得られる。

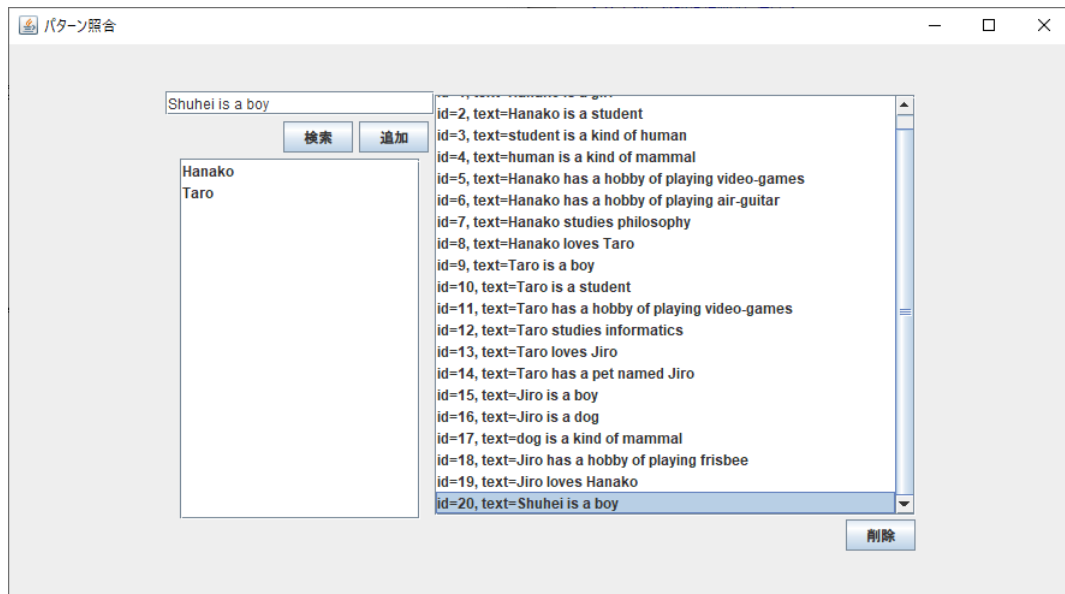


図 4: データの追加

右のリストから「id=19,text=Jiro loves Hanako」を選択し，削除ボタンを押したところ，下図のような画面が得られる．

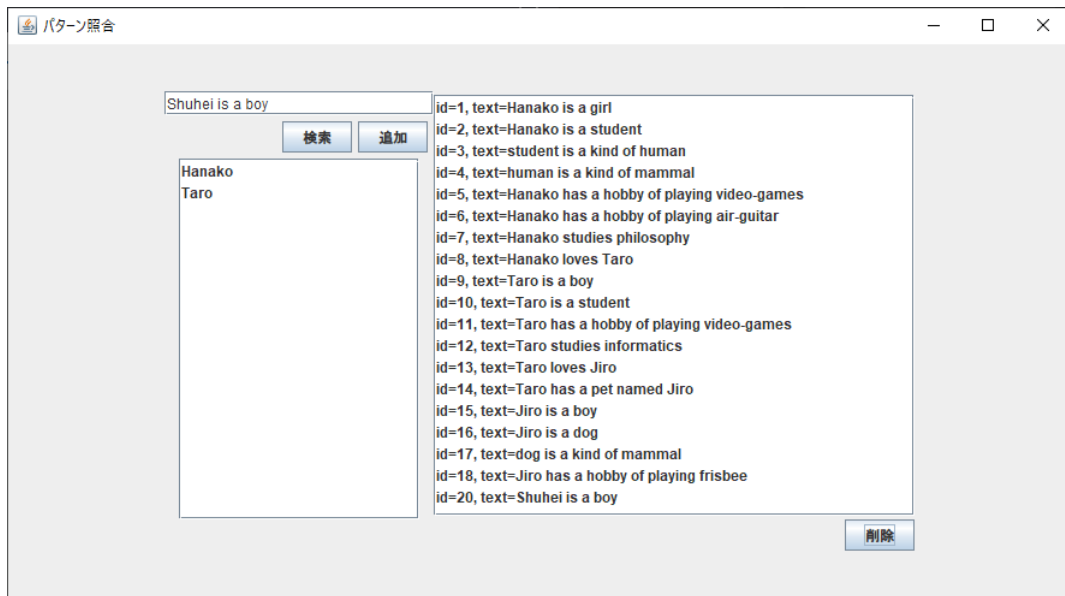


図 5: データの削除

右上の「×」ボタンからプログラムを終了し、再度 UnifyGUI を実行して起動したところ、下図のような画面が得られ、前回の追加・削除したデータが保持されていることが分かる。

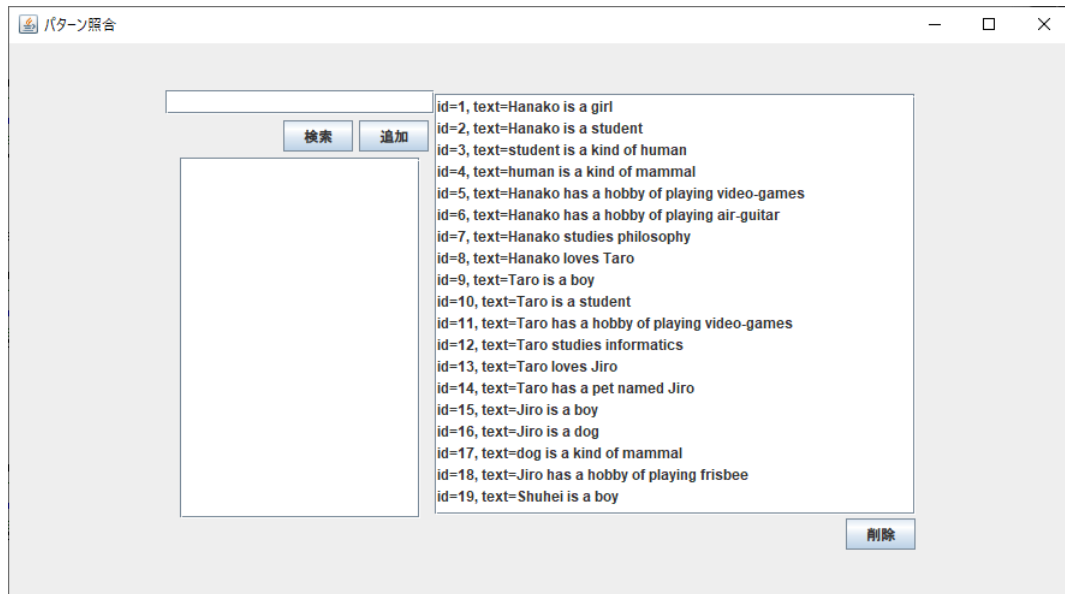


図 6: プログラムの再起動

2.4 考察

ViewInterfaceを実装するためのクラスとしてViewクラスをUnifyGUIクラスとは別に作ったが、これによりPresenterからの値をGUI実行時に使えるようになっただけでなく、UnifyGUIクラスを介さずともUnifyクラス等の単体テスト時にもPresenterクラスを利用できるようになったため、結果として独立したクラスでViewInterfaceを実装してよかったと考えられる。

また、Presenter担当者からの要求としてstart,finishメソッドの実行をプログラム開始・終了時に求められたが、finishメソッドの呼び出しにおいて、windowClosingメソッドの他に、windowClosedメソッドの利用が候補として挙げられた。しかし、windowClosedメソッドではfinishメソッドの呼び出しが行われなかったため、windowClosingメソッドを用いることとなった。windowClosingメソッドはウィンドウをクローズしようとしたときに呼び出される機能である一方で、windowClosedメソッドはウィンドウがクローズされたときに呼び出される機能であるが、いずれにおいてもウィンドウはクローズされるのに、なぜwindowClosedメソッドではfinishメソッドが呼び出されないのかは、ウィンドウがクローズした後finishメソッドを呼び出す前にUnifyGUIプログラムが終了してしまうためだと考えられる。要するに、「×」ボタンが押された時にプログラムを終了させるために設定したsetDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)の方が、windowClosedメソッドよりも実行優先度が上であるためだと考えられる。また、コンソールから「Ctrl + C」でプログラムを強制終了した場合も、finishメソッドは呼び出されなかったため、強制終了はwindowClosingメソッドよりも優先度が高いことが分かった。

GUIの表示において、レイアウトの設定が大変であった。setPreferredSizeやsetMaximumSize等のメソッドを駆使しても、コンポーネントを意図した通りの配置や大きさにすることが難しかった。そこで、GridBagLayoutクラスを用いたところ、これらの問題を一同に解消することができた。GridBagLayoutは表形式でありながらも相対的な位置を高い自由度で配置することができるため、今後も自由な表形式のGUIを作りたいときに重宝すべきものであると考えられた。

JListにスクロールバーを追加するためにJScrollPaneクラスを用いたが、これによりペインにも様々な種類があることを認識できた。JScrollPaneクラスを用いて実際に実装してみることで、ペインはパネルを多機能化したものというような認識を得ることができた。

今回のプログラムでは、データベース担当、GUI 担当、これらの仲介担当といったようにグループ内で分担を行ったが、この並行作業は大変なものであった。自分の知らないところでプログラムの構造が想定していたものから変わっていることがあり、グループ内で作業進捗の共有をより密に行う必要があったと感じたと同時に、プログラムが大規模になってくると、どうしても自分の干渉できない部分が出てくることは仕方のないことと割り切り、自分の想定していたものとは異なっているにもかかわらず相手に合わせてあげるといったことも重要だと考えられる。

しかし、今回の他の班員が作ったデータベースの構成についてはもう少し言及すべきだったと反省している。今回のデータベースはキャッシュとして扱われており、これは永続的な記憶を目的とするデータベースの本来の用途とは異なっていると考えられる。また、データベースの構造自体も、各行に ID を割り振って 1 つのレコードとして取り扱うという、文が構造化されて分解されたものだとはいえ到底言い難いものとなってしまったと考えられる。

3 感想

ViewInterface は班員からの指定で実装したものだったが、結局このインターフェースの目的が何だったのかはよく分からなかった。無かったことにしようとしても、Presenter クラスとの不整合が発生してしまうため、仕方なく実装したが、今後このインターフェースのような不可解なインターフェースやクラス・メソッド等の実装を班員から求められたら、その意図や実装してほしい内容を、早い段階から聞いて意思の疎通を図るとともに、自分から他の班員にそのような実装を求めるときは、説明を充分にしてから求めることとして、プログラムの分担と結合をより円滑に行えるようにしてゆきたい。

他の班員が作ったデータベースについては、考察でも述べたように納得できるものではなかったが、制作期間の問題もあり、後から突き詰めることはしなかった。しかし、データベースとは本来どのようなものかという認知を予め説明し、自分でも調べるようお願いすべきだったと反省している。

前回の GUI が図形式で表示する必要があった一方で、今回の GUI は表形式で示せる内容だったため、GridBagLayout クラスのおかげで比較的楽に作ることができた。しかし、データベースとのデータのやり取りが

大変であり、いかに情報を保持したまま形式を変換するか、例えばデータベースから受け取った情報を保持したまま JList に格納する点などで手こずった。しかし、この点においては TextModel クラスを用いて一元的にデータを管理することで解決でき、上手にクラスを活用できたと思っている。

また、グループ内でタスクを細かく分割したことによるやりとりが本当に大変であった。今後はメンバーのソースコードも早いうちから読んで、認知の共有を密に行い、作業の効率化を図ってゆきたい。

前回に引き続き GitHub を活用したが、各々が同じプログラムを複製して作業したために、最後の統合作業が大変になった。次回はこの反省を活かし、より GI t Hub による体系立ったグループワークを実現してゆきたい。