知能プログラミング演習 II 課題 2

グループ 8
29114116 増田大輝

2019 年 10 月 7 日

■提出物　rep2

■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	NoData
29114060	後藤拓也	NoData
29114116	増田大輝	NoData
29114142	湯浅範子	NoData
29119016	小中祐希	NoData

1　課題の説明

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.

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. どのような方法でデータセットを登録しても構わない.

課題 2-3　上記システムの 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.

2　課題 2-3

上記システムの 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.

私の担当は全体のシステム設計と Presenter の構築に留まるため, 実行例は無い.
実装と考察に関しては適宜, 各種手法の詳細説明において言及する.
また, 必要に応じてソースコードを明示する場合もある.

2.1　手法

今回の課題では, データベースの利用や GUI の設計などシステムを構成する要素が前回課題よりも多いと考えられる. したがって, 機能区分を明確に行うことによって, システムとしての保守性やソースコードの可読性, 依存関係の明確化を狙うことが重要であると考えた. これらの目的を達成するために, 以下のような一般的にシステム・アプリケーション開発で取り入れられている設計概念を採用することとした.

MVP アーキテクチャ
DAO パターン
抽象化による疎結合な関係性の構築

2.2　MVP アーキテクチャの導入

課題 2-3 では,GUI の使用やデータベースの応用が必要となるため, 明確な機能区分を与えることで全体の設計指針が立つと考えられる. そこで, 今回はアプリケーションアーキテクチャとして代表的な MVP アーキテクチャを採用することによって, 機能の切り分けを行った. 以下に MVP アーキテクチャの各構成要素について述べる.

Model　データ処理機構を担っている. 今回はデータを格納するデータベースやデータベースへのアクセスを担う DAO が該当する.View や Presenter に依存しない.

Presenter　View から受けた処理に基づいて Model からデータを取得し, 画面反映を行うための View メソッドを呼び出す. これにより,View と Model 間の円滑なデータフローと画面制御を行う.

View　ユーザーインターフェースを担う. ユーザー入力を受け取り,Presenter に通知し, 処理結果を出力として画面に表示する. 今回は, 入出力に対応するロジックで構成される GUI が該当する.

以下に MVP アーキテクチャの概念図を示す.

図 1　MVP アーキテクチャの概念図

2.3　DAO パターンの導入

今回は, データベースへのアクセスを一括して担うデザインパターンである DAO を導入した. 全てのデータ処理において, 必ず DAO を通すことによって, 他のクラスにデータベースアクセスメソッドが分散することを防ぐことができる. さらに,Model としての機能を担っていると考えられるため,Presenter のみが DAO インスタンスを握るように設計した. ただし, テキストファイルの使用が課題の条件として提示されていたため, 通常アプリケーション開発では行われないテキストファイル-データベース間の処理機構も必要となった. これに対処するために, 今回は通常の DAO に加え, テキストファイル-データベース間の処理を担当する TextDAO を設計した. これら二種類の DAO の分類は, 以下

の考えに基づいて明確に規定される.

DAO　通常のデータベースアクセス処理を担い, 今回はデータの追加・取得・削除を行うメソッドを取りまとめている.

TextDAO　データベースをテキストファイルの一次キャッシュとみなし, テキストファイル-データベース間の読み書きを担当する. 具体的には,GUI 起動時にテキストファイルからデータベースへのデータを読み込み,GUI 終了時にデータベースからテキストファイルへのデータ書き込みを行う. 結果として, テキストファイルとデータベースの一貫性を保つこととなる.

以下に DAO パターンの概念図を示す.

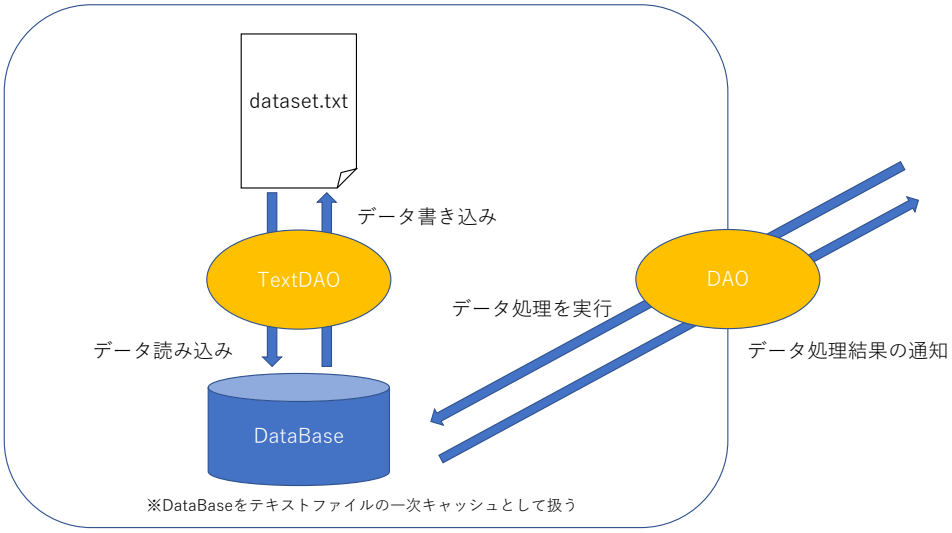

図 2　DAO パターンの概念図

今回,Presenter は TextDAO に対して以下のメソッドを要求する.

readTextFile メソッド　テキストファイルからデータベースへのデータ読み込み
writeTextFile メソッド　データベースからテキストファイルへのデータ書き込み

Presenter は DAO に対して以下のメソッドを要求する.

addData メソッド　データベースにデータを追加する

fetchDataList メソッド　データベースからデータを取得する
deleteData メソッド　データベースからデータを削除する

2.4　抽象化による疎結合な関係の構築

まず,MVP アーキテクチャにおける各要素の間の依存関係について述べる. 上述した通り,Model クラスは Presenter や View に全く依存していない. 一方で,Presenter が Model インスタンスを握っていることから,Presenter は Model に依存する関係にある. 加えて,MVP アーキテクチャ含め, 一般的な MVC 系統のアーキテクチャにおいては,View が Controller 部のインスタンスを握ることから, 今回は View が Presenter インスタンスを握ることとなる.
ここで,MVP アーキテクチャの特徴として,Presenter はユーザー入力を Model に通知するだけで無く, 直接　View への反映命令を行う点があげられることに注目する. 後者の機能の実現には, 何らかの形で Psesenter から View メソッドを呼び出す必要性が生じる. したがって,Presenter 内において,View メソッドを持ちうるインスタンスの生成が不可避的となる.
しかし,View と Presenter が直接互いのインスタンスを握り合うことは望ましく無い. 何故ならば, これらの要素が互いのインスタンスをにぎり合うことにより, 強結合と呼ばれる相互的な依存関係が生まれ, アーキテクチャを採用することによる機能分割の意味が薄れるためである. 実際, 互いに直接インスタンスを握り合うのであれば, アーキテクチャという観点からはクラスを統一した場合と意義的にはほぼ等しい.
この問題を解決するための手法が, インターフェースを利用した抽象化による疎結合な関係の構築である. 実装としては,Presenter から View を管理するメソッドを集めた ViewInterface を定義し,Presenter 側では ViewInterface 型インスタンスを持つようにする.

ソースコード 1　ViewInterface の定義

```
1    interface ViewInterface {
2        \\データベース初期化完了メソッド
3        void successStart();
4        \\テキストファイル記録完了メソッド
5        void successFinish();
6        \\データ追加完了メソッド
7        void successAddData();
```

```
8           \\検索結果反映メソッド
9           void showSearchResult(List<TextModel> resultList);
10          \\データ削除メソッド
11          void successDeleteData();
12          \\一覧表示メソッド
13          void showResultList(List<TextModel> resultList);
14          \\例外処理表示メソッド
15          void showError(String errorText);
16          \\データ無し表示メソッド
17          void showNoData();
18      }
```

ソースコード 2　Presenter における ViewInterface 型インスタンスの生成

```
1       class Presenter {
2           ...
3           private ViewInterface view;
4
5           public Presenter(ViewInterface view) {
6               this.view = view;
7           }
8           ...
9       }
```

View 側の実装クラスでは, インターフェースを実装し, オーバーライドされたメソッドの具体的な処理を記述する.

ソースコード 3　GUI における Presenter インスタンス生成

```
1       class GUI implement ViewInterface {
2       ...
3       Presenter presenter;
4       init() {
5           presenter = new Presenter(this);
6       }
7       ...
8   }
```

また,View は生成後に Presenter コンストラクタに自身を渡すことで,Presenter インスタンスを生成する. 反対に View の終了時には,Presenter インスタンスを離すように設計す

る.

以上により,View が Presenter インスタンスを握ることで依存しながらも,Presenter 側は直接明示的に View インスタンスを握らずとも View メソッドを呼び出せる状態が完成する. すなわち,View が Presenter を持つという一方的な依存関係としての疎結合な関係の構築が達成される.
結果的に, 一般的にアーキテクチャを採用する際に好ましいとされる緩やかな関係性を持たせることができるのである.
以下に,View-Presenter 間における疎結合な関係を示す.

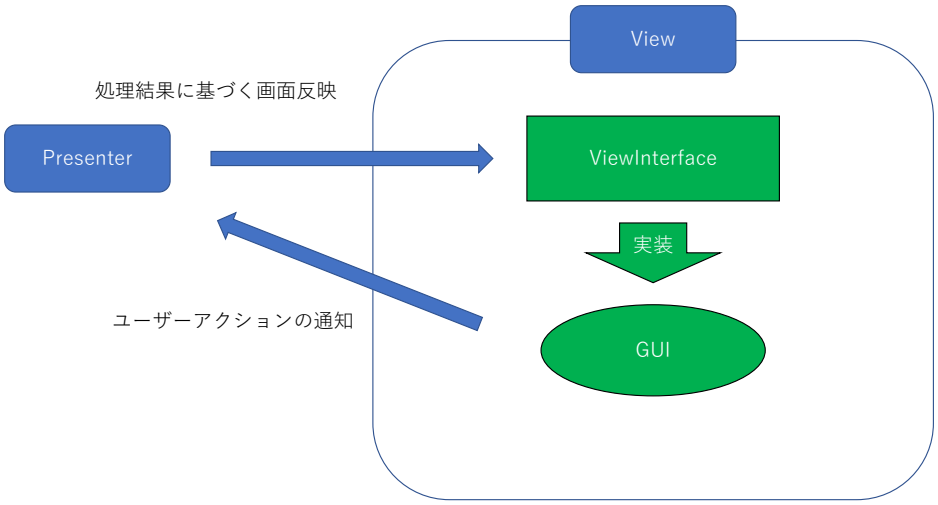

図 3　View 内部と Presenter の依存関係の概念図

2.5　システムの全体像

上述した 3 種類の技術を用いて, 今回の課題のシステムを実現した. その全体像を以下に示す.

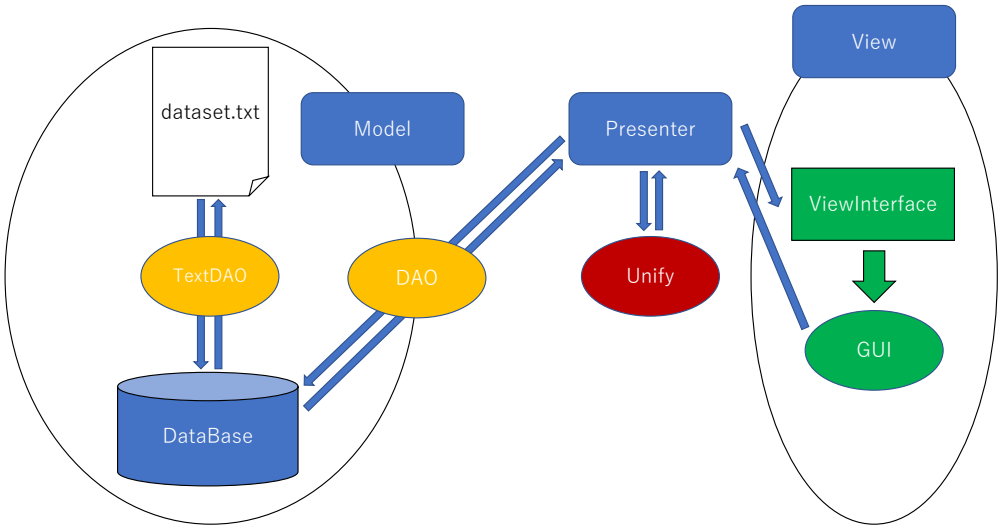

図 4　システム全体の概念図

2.6　データ追加・削除機能の実装

View がユーザー入力を受け取り,Presenter に通知する. Presenter は DAO に問い合わせ, 結果により呼び出す View メソッドを切り替えて View に反映指示を出す.

2.7　データ検索・一覧表示機能の実装

View がユーザー入力を受け取り,Presenter に通知する. Presenter は DAO に問い合わせ, 結果として受け取ったデータリストとユーザー入力から得られた文字列を Unify 内部のメソッドに渡すことで検索を行う. Unify による検索結果から, 呼び出す View メソッドを切り替えて View に反映指示を出す.

2.8　テキストファイル-データベース間の読み込み/書き込み機能の実装

View がユーザー入力を受け取り,Presenter に通知する．Presenter は TextDAO に問い合わせ, 実行結果により呼び出す View メソッドを切り替えて View に反映指示を出す.

2.9　考察

ここでは, 全体的な考察について述べることとする.
まず, アプリケーション/システムとして重要なのは機能を明確に区分することであると考えた．次に, チーム開発を行うに当たって, 保守性や可読性を高めることに重点を置くべきだと考えた．これら二つの考えを基として,MVP アーキテクチャや DAO パターンの導入が適切であると判断した．その反面, 概念的な要素が多く, チームメンバーに対しての説明を要した．しかし,GitHub 上でソースコード管理を行っているので, コンフリクションを防止するためにもアーキテクチャやデザインパターンにより役割分担を行うことは非常に有効な手段であると考えられる.

さて, 機能の切り分けと分担を考える上では非常に良い課題であると感じた反面, 今回の課題について, 非常に違和感を感じた点がある．それは, 課題 2-3 において, テキストファイルへのデータ読み込みと書き出しによりデータの保管を行う指示があることだ．私は, 今まで Android アプリケーションの開発に携わってきたが, まずデータ保管のためにテキストファイルを用いるのは一般的では無いとの認識を持っている．通常, サーバーに管理を任せるか,OS 標準搭載のデータベースを用いる．今回であれば, 課題 2-1 でデータベースの作成を要求されているので, データベースを利用するのみで十分であると考えられる．また, データベースの永続性の観点から, わざわざテキストファイルへの保存を行うのは冗長であると感じられる．課題の指示内容の解釈に苦しんだので, グループメンバーが TA に質問して提案された内容を重んじ, データベースをテキストファイルの一次キャッシュのような形にすることを決定した．上述した通り, データベースの利用法としては非常にナンセンスだと感じられる.
次に,Unify とデータベースの検索機能が競合する点である．Unify は読み込んだデータを元に指定された文字列とのマッチングを行うことで検索を行う．一方で, データベースに動詞ごとのテーブルを作り, 主格と目的格を属性として持たせ,SELECT 文を実行することで同様の機能を実装できる．すなわち, 検索機能に関して二つの実装方式が可能であり,

問題文からは Unify を用いる実装, 一般的にはデータベースを用いる実装があげられる. 前者を採用した場合は, データベースはただ一つのみのテーブルを持ち, そのまま文章を格納するだけの機能となり, 十分にデータベースを活用できているとは言えない. 後者を採用した場合は, そもそも Unify を利用する必要性が無くなり, 課題の趣旨に反する可能性がある. 結果的に, 今回の課題テーマにあるパターンマッチングを活かすために前者を採用したが, 現実的な開発においては後者が最適であると個人的に強く感じる.
これらのような課題指示の曖昧性と現実的なシステムからの乖離は学生を非常に混乱させ, メンバー間の意思疎通をいたずらに困難とすると考えられる. もちろん, 実装課題であるので, 受け手側の受け取り方による多少の揺らぎは許容されるべきであるが, 学生の知識や経験にそぐわない実装を手段として選ばざるを得ないような表現にはいささか疑問が残る. 課題内容の吟味と説明の徹底を行うことを強く推奨すると共に, このように学生が解釈に手間取る可能性を十分に考慮して, 課題内容を早期に公開すべきであると考える.

2.10 感想

以上のような疑問を抱えたまま, グループワークを行うと統制が取れなくなるだけでなく, システム設計者として大変苦労することが理解できた. 提出期限が約一週間という短期間であることを考慮すると、他メンバーの進捗等を考慮してなるべく早めに全体の方針を打ち出す必要があるので, 急いで作業を行うようにしている. 結果的に, 課題の曖昧性を吟味することやメンバーの理解状況の把握することに非常に骨が折れる. さらに, 今回の課題では, 曖昧性の中でこのような実装を採用した根拠をメンバーに説明する義務が生じる. そのために, 徹夜を余儀なくされ, 本来の予定をキャンセルするなど生活に実害が及んでいる. もう一度強調したいが, 極めて計画的に行動していてもこのような負担を強いられているのが現状である. したがって, 課題の出題方法・内容に対して非常に不満を感じた.

参考文献

[1] 新入社員におくる GitHub でのプロジェクト管理の初歩 –著：hayato ki https://qiita.com/gumimin/items/63dcb36d4730213bd63a （2019 年 10 月 7 日アクセス）.