知能プログラミング演習II 課題2

グループ8
29114142 湯浅範子

2019年10月29日

提出物　rep2(29114142.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	
29114060	後藤拓也	
29114116	増田大輝	
29114142	湯浅範子	
29119016	小中祐希	

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．どのような方法でデータセットを登録しても構わない．

発展課題 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．

今回私は3つ全ての課題に関わるプログラム等を作成したため，それぞれについて順に記述していく．

2　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．

私の担当箇所は，データセットからデータを読み取り，データベースに格納するためのメソッドの作成である．

2.1　手法

課題内容の実装のために，以下のように改良を加えた．

1. 与えられたデータセットを DB に格納する．
2. DB にアクセスするための DAO を作成する．
3. DB から得られたデータと，検索文を比較して検索を行えるよう Matching.java と Unify.java を改良する．

1. に関して，テキストファイルと DB 間でデータのやり取りを行うため，テキストファイルにアクセスするための java ファイル TextCon.java を作成した．そして 1. の実現のため，テキストファイルからデータを読み込むためのメソッド readTextFile を作成した．これは私が担当した．

2. に関しては，DB アクセスを行う際のデザインパターンである DAO の作成のため，TextDAO.java を作成した．具体的には table(texts) を作成し，column として uuid(int 型) と line(text 型) を定め，そこにテキストファイルの中身が入るようにした．これも私が担当した．

3. に関しては，与えられた Unify.java では変数入力の際に値を返すことが出来ないため，変数検索にも対応可能になるようプログラムの書き換えを行った．これは後藤君と小中君が担当した．

2.2　実装

まずテキストファイルから DB を利用するためのプログラムを作成した．始めに DB とその table を作成し，その内容を初期化する．その後与えられた「テキストファイルに対する操作のプログラム」を利用してテキストファイルから一文ずつ取り出し DB に格納する．

このとき，テキストファイルを読み込むためのメソッドは readTextFile，DB とその table を作成するメソッドが createTab，DB の table 内部を初期化するメソッドが deleteData，読み込んだデータを DB に格納するメソッドが insertTextTab である．

テキストファイルアクセスとDBアクセスを分けるため，それぞれTextCon.java，TextDAO.javaというファイルを作成し，readTextFileメソッドはTextCon.javaに，createTab，deleteData，insertTextTabメソッドはTextDAO.javaに含まれている．

また，DBアクセスプログラムは，3年生前期の授業で取ったプログラミング応用の課題で作成したDAOなどのプログラムも参考にして作成を行った．

テキストファイルを読み込んでDBアクセスを指示するreadTextFileメソッドの実装をソースコード1に示す．

ソースコード 1: readTextFileメソッド

```
public void readTextFile() throws FileNotFoundException{
        String empty = "";
        //text ファイル処理
        try {
                // table の作成【DB】
                TextDAO.createTab();
                TextDAO.conCom();
                // table の中身の初期化【DB】
                TextDAO.deleteData();

                // ファイル読み込み操作
                        …
                        if (!line.equals(empty)) {
                                // 更新【DB】
                                TextDAO.insertTextTab(line);
                        }
                        //【DB】
                        TextDAO.conCom();
                }
                in.close(); // ファイルを閉じる
        } catch (IOException e) {
                e.printStackTrace();
        }
        //【DB】
        TextDAO.conCom();
        TextDAO.closeConn();
}
```

DBのtable作成のためのcreateTabメソッドをソースコード2に示す．

ソースコード 2: createTabメソッド

```
public static Connection conn = null;
public static final String connDB = "jdbc:sqlite:data.db";
// 格納用table を作成
public static void createTab(){
        PreparedStatement pStmt = null;
```

```
6         String sql;
7         try{
8                 Class.forName("org.sqlite.JDBC");
9                 conn = DriverManager.getConnection(connDB);
10                conn.setAutoCommit(false);
11                sql = "create table texts(uuid int, line text)";
12                pStmt = conn.prepareStatement(sql);
13                pStmt.executeUpdate();
14        }catch{
15                ...
16        }
17 }
```

DB の table の中身の初期化のための deleteData メソッドをソースコード 3 に示す.

ソースコード 3: deleteData メソッド

```
1  // DBtable の中身の削除
2  public static void deleteData() {
3         conn = null;
4         PreparedStatement pStmt = null;
5         try{
6             Class.forName("org.sqlite.JDBC");
7             if(conn == null){
8                         conn = DriverManager.getConnection(connDB
                              );
9                         conn.setAutoCommit(false);
10                }
11        String sql = "delete from texts";
12        pStmt = conn.prepareStatement(sql);
13                pStmt.executeUpdate();
14        }catch{
15                ...
16        }
17 }
```

DB にデータを格納するための insertTextTab メソッドをソースコード 4 に示す.

ソースコード 4: insertTextTab メソッド

```
1  // table に追加
2  public static void insertTextTab(String line) {
3         PreparedStatement pStmt = null;
4         String sql;
5         try{
6                 Class.forName("org.sqlite.JDBC");
7                 if(conn == null){
8                         openConn();
9                 }
10                sql = "insert into texts values(?, ?)";
```

```
11                  pStmt = conn.prepareStatement(sql);
12                  pStmt.setInt(1, id);
13                  pStmt.setString(2, line);
14                  pStmt.executeUpdate();
15                  id++;
16          }catch{
17                  …
18          }
19  }
```

またこれらの動作のため，DB へのコネクション OPEN として openConn メソッド，CLOSE として closeConn メソッド，COMMIT として conCom メソッドを新たに作成している．これらは TextDAO.java と TextCon.java のどちらからも利用される．

2.3 実行例

作成した DB への格納プログラムが正しく動作しているかを確認するために Test.java を作成し，readTextFile メソッドを実行した．すると，data.db という名前の DB が作成される．この中身を確認したところ，以下のように表示された．

```
1  C:\Users\Owner>sqlite3 data.db
2  SQLite version 3.28.0 2019-04-16 19:49:53
3  Enter ".help" for usage hints.
4  sqlite> .table
5  texts
6  sqlite> select * from texts;
7  1|Hanako is a girl
8  2|Hanako is a student
9  3|student is a kind of human
10 4|human is a kind of mammal
11 5|Hanako has a hobby of playing video-games
12 6|Hanako has a hobby of playing air-guitar
13 7|Hanako studies philosophy
14 8|Hanako loves Taro
15 9|Taro is a boy
16 10|Taro is a student
17 11|Taro has a hobby of playing video-games
18 12|Taro studies informatics
19 13|Taro loves Jiro
20 14|Taro has a pet named Jiro
21 15|Jiro is a boy
22 16|Jiro is a dog
23 17|dog is a kind of mammal
24 18|Jiro has a hobby of playing frisbee
25 19|Jiro loves Hanako
26 sqlite>
```

この結果はテキストファイルに含まれていたデータセットの内容と一致することから，作成したプログラムが正しく動作し，DB が作成できていることが確認できた．

2.4 考察

この課題は課題文の解釈の方法がいくつか考えられたため，どのように DB を作成するかを考えるのにかなりの時間を要した．

まず，DB の作成を行うため，データセットであるテキストファイルからどのように DB を作成するかを考えた．この時大きく分けて 2 種類の作成方法がまず考えられた．それを以下に示す．

1. DB にデータセットと同様の内容を格納する．
2. DB に Unify で処理した検索文 (変数を含む文章) とその出力結果を格納する．

この 2 種類の考えは，課題文の解釈の違いによって考えられた．
「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.」という課題であるが，Matching クラスや Unify クラスをどの段階で使用するのかが明確に判断できず，DB 作成後に Matching クラスや Unify クラスを利用するのか，Matching クラスや Unify クラスを利用して得られた結果を DB に格納するのかが分からなかったからである．Matching クラスや Unify クラスを使用する前に DB を作成する場合は 1. の DB の形式にし，Matching クラスや Unify クラスを使用した後に DB を作成する場合は 2. の DB の形式にすると考えられたが，どちらを採用するかがかなり難しい判断であった．

さらに，DB をデータセットと同様の内容とした場合はさらに 3 種類の作成方法が考えられた．それを以下に示す．

1. DB に一文をそのまま text 型として格納する．
2. DB に一文を各単語毎に分割して格納する．
3. DB で動詞毎に table を作成し，それぞれの動詞で分類分けした状態で主語とそれ以外 (動詞も除く) に分割して格納する．

DB をより効率よく利用する方法を考え，これらのような格納方法を考えた．

DB の作成方法に複数の方法が考えられたが，これは必須課題の 1 つ目であり，作成期間も一週間と短く，出来るだけ早く方針を固めなければならない．そこで私たちの班では，DB のこれ以降の課題との関係性や検索方法と課題文の内容にどれだけ沿うことが出来るかなどを考慮してプログラムを作成することとした．

ここで，先に挙げた DB の格納方法とその具体例を表形式にまとめ，それぞれの利点と欠点を上げていく．

1 つ目は「DB に一文をそのまま text 型として格納する.」方法である．

表 1: DB 例 1(table:texts)

uuid(int)	line(text)
1	Hanako is a girl
2	Hanako is a student
…	…
8	Hanako loves Taro

表 1 の利点は，DB への格納が単純であるため比較的楽にプログラムすることが出来る点にある．また，Unify.java 等がデータを受け取るときにテキストファイルを読み込む形と殆ど変わらずにプログラム出来るため，改良の手間が少し減る点もあげられる．

これに対し欠点は，格納方法がテキストファイルと同様の形式であるため，DB へ格納した恩恵をあまり得られない点にある．データの削除や追加はテキストファイルよりも楽に行うことが出来るが，検索に関しての手間はテキストファイルと変わらないため，あまり意味のない DB になってしまう可能性も考えられる．

2 つ目は「DB に一文を各単語毎に分割して格納する.」方法である．

表 2: DB 例 2(table:texts)

uuid(int)	namea(text)	nameb(text)	namec(text)	named(text)
1	Hanako	is	a	girl
2	Hanako	is	a	student
…	…	…	…	…
8	Hanako	loves	Taro	NULL

表 2 の利点は，検索が DB を用いて行うことが出来るために，マッチング操作にかかる手間が大幅に減ることがあげられる．

これに対し欠点は，一文に含まれている単語の数が一定でないため，上の表のようにフィールドによっては NULL になってしまい DB への格納が複雑になることがあげられる．一文が分割される単語数もテキストファイルに含まれる行数も可変であるため，可変長二次元配列を活用して格納を行わなければならなくなってしまい，かなり複雑なプログラムとなることが考えられる．またこの格納方法では，検索時に DB 参照によって検索結果を得られるため，使用するように指定

されている Matching クラスや Unify クラスが利用できなくなってしまうことも欠点としてあげられる．

3 つ目は「DB で動詞毎に table を作成し，それぞれの動詞で分類分けした状態で主語とそれ以外 (動詞も除く) に分割して格納する.」方法である．

表 3: DB 例 3(table:iss)

uuid(int)	namea(text)	nameb(text)
1	Hanako	a girl
2	Hanako	a student
···	···	···

表 4: DB 例 3(table:lovess)

uuid(int)	namea(text)	nameb(text)
8	Hanako	Taro
···	···	···

表 3, 表 4 の利点は，table を複数作成することで，テキストファイルには出来ない単語毎の分類が可能になるので，DB を利用する利点が明確になることである．同時に，分類分けが行われることで検索時のマッチング操作にかかる手間が減ることも考えられる．

これに対し欠点は，分割を行って動詞毎に table を作成するため，DAO が非常に複雑になってしまうことがあげられる．実際にこのプログラムは始めに作成したが，デバッグ作業にかなりの時間を要することになった．また各 table 毎に検索を行うため，Unify.java に引数としてデータを与える際に工夫が必要になることも，実行効率は良いが実装効率が悪いことを示している．さらに調べたところ，table として指定できない特定の文字列が存在し，それが is が該当していると分かった．そのため動詞毎で table を作成する場合は全ての動詞に's' を付けるなどの工夫が必要になることが課題を解く中で分かった．これも，プログラムが複雑になる原因の一つとして考えられる．

最後に,「DB に Unify で処理した検索文 (変数を含む文章) とその出力結果を格納する.」方法である．

表 5: DB 例 4(table:answers)

uuid(int)	question(text)	answer(text)
1	?x has a hobby of playing video-games	Hanako, Taro
2	Hanako is a ?y	girl
…	…	…
8	?x is a boy, ?x loves ?y	(Taro, Jiro), (Jiro, Hanako)

表 5 の利点は，必須課題で求めらているプログラムに最も近いと考えられるプログラムであることがあげられる．また，データセットと DB の役割の違いがはっきりするため，テキストファイルと DB を有効に活用できている部分も利点と考えられる．

これに対し欠点は，格納のためのプログラムが複雑になってしまうことと，発展課題を行う際に追加と削除のデータが DB に反映させられないことがあげられる．DB を利用することで，テキストファイルよりも追加や削除が簡単になるが，この方法で実装を行うとこれが出来ない．それにより追加や削除命令に応じてテキストファイルを直接操作することになるが，これは非常に手間がかかり効率も悪い．

これらの利点と欠点，そして実装期間を考え，私たちの班は表 1 の方法を採用することとした．これらを踏まえ作成したものが先に述べたプログラムである．そのため，結果として DB はテキストファイルと形式を殆ど変えることなく作成することとなった．ただしこのとき，改行は DB に含まないことにしたため，DB では空行は含まれずに格納される．

これにより実装を分担して行う今回のような課題は実装方法が複雑にならなかったことから大幅に時間をかけることなくプログラムを作成することが出来た．

3　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．どのような方法でデータセットを登録しても構わない．

必須課題 2-2 は実装を伴わない課題であるため，実装以外を記述する．

3.1 手法

私たちの班では，必須課題 2-1 で DB を表 1 のように作成したため，与えられたテキストファイルと同様の形式でデータセットを作成した．この課題は私が担当した．

興味のある分野については，ラグビーワールドカップが日本で開催され日本チームが活躍していたことから，ラグビーについて調べ以下のようにデータセットを作成した．

表 6: 作成したデータセット (rugby.txt)

Rugby is a sport
RugbyWorldCup is held in Japan
Japan is ranked 8th in the world
Japan won against Russia
…
England lose to SouthAfrica
JamieJoseph is the coach of Japan
YuuTamura scored 51 points
…
MichaelLeitch tackled 44 times
England is ranked 1st in the world
NewZealand is ranked 2nd in the world
Wales is ranked 3rd in the world

3.2 実行例

まず，必須課題 2-1 と同じ要領で作成したデータセットから DB を作成した．DB の名前や読み込むテキストファイルについてはプログラムを直接書き換えることで実行した．その結果作成された DB の中身を出力した結果を以下に示す．

```
1 C:\Users\Owner>sqlite3 rugby.db
2 SQLite version 3.28.0 2019-04-16 19:49:53
3 Enter ".help" for usage hints.
4 sqlite> .table
5 texts
6 sqlite> select * from texts;
```

```
1|Rugby is a sport
2|RugbyWorldCup is held in Japan
3|Japan is ranked 8th in the world
4|Japan won against Russia
5|Japan won against Ireland
6|Japan won against Samoa
7|Japan won against Scotland
8|England lose to SouthAfrica.
9|JamieJoseph is the coach of Japan
10|YuuTamura scored 51 points
11|KotaroMatsushima scored 25 points
12|KenkiFukuoka scored 20 points
13|PieterLabuschagne tackled 68 times
14|JamesMoore tackled 67 times
15|ShotaHorie tackled 58 times
16|KazukiHimeno tackled 50 times
17|KeitaInagaki tackled 48 times
18|LukeThompson tackled 47 times
19|MichaelLeitch tackled 44 times
20|England is ranked 1st in the world
21|NewZealand is ranked 2nd in the world
22|Wales is ranked 3rd in the world
sqlite>
```

これは作成したテキストファイルと同様の内容であることが確認できた.

次に，このDBを用いて検索を行った結果を以下に示す．今回のデータセットでは，Unifyクラスの比較方法は使用しないため，Matchingクラスにmainメソッドを作成し，そこからプログラムを呼び出しテキストファイルからDBへの値の格納・マッチングを行う．これらのプログラムの連動関係に関しては増田君がアーキテクチャを考えたため，そちらのレポートを参考にされたい.

以下にMatchingクラスのmainメソッドを用いて作成したデータセット読み出して動作させた場合の検索実行例を示す.

```
C:··· >javac -encoding UTF-8 -cp sqlite-jdbc-3.21.0.jar;. Matching
    .java
C:··· >java -cp sqlite-jdbc-3.21.0.jar;. Matching "Rugby is a ?x"
Successfully started
targetData = Rugby is a ?x
検索結果を取得
answer = sport
C:··· >java -cp sqlite-jdbc-3.21.0.jar;. Matching "Japan is ranked
     ?y in the world"
Successfully started
targetData = Japan is ranked ?y in the world
```

```
検索結果を取得
answer = 8th
C:··· >java -cp sqlite-jdbc-3.21.0.jar;. Matching "Japan won
    against ?z"
Successfully started
targetData = Japan won against ?z
検索結果を取得
answer = Russia
answer = Ireland
answer = Samoa
answer = Scotland
C:··· >
```

作成したデータセットと比較しても，正しい結果が得られていることが確認できたので，DB から受け取った値と検索文との比較検索が正しく行われ，正確な結果が得られていることが分かった．

3.3 考察

データセットの登録方法はどのような方法でも良いと記述されていたため，テキストファイルに直接書き込む形で作成を行った．記述に際しては，今回は複雑になりすぎないよう Matching クラスを利用してマッチングを行うことを考え，似たような語彙や記述が入るように工夫した．また，国名や人名はスペースを空けると別の語として扱われてしまうため，これを防ぐためスペースを空けずに記述することとした．しかし，今回のデータセットでは中々解 (answer) が複数になる結果が得られるようなデータセットにならなかったため，今後データセットを作成する際には解が複数になるような可能性も多く考えられるようなデータセットを選択したい．また今回のデータセットでも，選手のポジション等を利用して作成すればよりよいデータセットにすることが出来たのではないかと実行を行って感じた．

形式は必須課題 2-1 と同じであるため，DB への格納等はテキストファイルが変わっても正しく動作することも確認できた．また今回は読み込むテキストファイルや DB を変更する場合は直接プログラムを書き換える必要がある．これは入力の操作でこれらを変更できたりするようプログラムを書き加えることで，プログラムを直接触ることなく読み取りファイルの変更が可能になると考えたが，実装する時間を取ることが出来なかった．しかしプログラム方法自体は比較的容易だと考えられるので，時間に余裕があれば取り組んでみたいと考えた．

4　発展課題 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.

私は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むことが出来るようにするためのメソッドの作成と，GUI からの DB アクセスのために作成した Dao.java の実装を行った.

4.1　手法

必須課題 2-1 で，テキストファイルからデータセットを読み込むときに利用した TextCon.javat と TextDAO.java を利用して新しい機能を追加した．追加したものは以下のようになっている.

1. GUI からの命令で書き換えが行われた DB から，データを取得する.
2. DB から取得したデータを，テキストファイルに書き込む.

これらの実装を私が担当した.

4.2　実装

書き換え終了後に DB からその内容を取得するメソッド getDBData，取得したデータをテキストファイルに書き戻すメソッド writeTextFile の作成を行った．それぞれのメソッドは，getDBData メソッドが TextDAO.java に，writeTextFile メソッドが TextCon.java に含まれている.

取得したデータをテキストファイルに書き戻すメソッド writeTextFile の実装をソースコード 5 に示す.

ソースコード 5: writeTextFile メソッド

```
1 public void writeTextFile() throws FileNotFoundException{
2         try { // ファイル読み込みの操作
3                 …
4                 // データの取得【DB】
5                 TextData = TextDAO.getDBData();
6                 TextDAO.conCom();
7                 // ファイルへの書き込み
8                 for (int i = 0 ; i < TextData.size() ; i++){
9                         out.println(TextData.get(i));
```

```
10                  }
11                  out.close(); // ファイルを閉じる
12          } catch (IOException e) {
13                  e.printStackTrace();
14          }
15          TextDAO.closeConn();
16  }
```

DB に格納されているデータを取得するメソッド getDBData の実装をソースコード 6 に示す.

ソースコード 6: getDBData メソッド

```
1  // DB のデータを取得
2  public static ArrayList<String> getDBData() {
3          conn = null;
4          ArrayList<String> DBList = new ArrayList<String>();
5          PreparedStatement pStmt = null;
6          ResultSet rs = null;
7          String sql;
8          try{
9                  Class.forName("org.sqlite.JDBC");
10                 if(conn == null){
11                         conn = DriverManager.getConnection(connDB);
12                         conn.setAutoCommit(false);
13                 }
14                 sql = "select line from texts";
15                 pStmt = conn.prepareStatement(sql);
16                 rs = pStmt.executeQuery();
17                 while (rs.next()) {
18                         String text = rs.getString("line");
19                         DBList.add(text);
20                 }
21                 rs.close();
22                 pStmt.close();
23         }catch{
24         …
25         }
26         return DBList;
27 }
```

これらに加え，増田君が作成してくれたたDB へのアクセスを行う Dao.java のプログラムに対して実際の動作が正しく行われるように修正をしたり，Presenter.java の DB アクセス指令の改良やデバッグなども行った.

4.3 実行例

GUI の作成は青山君が作成したため，ここでは私が実装したファイルの書き込み部分のみに着目して実行例を確認する．必須課題 2-1 で作成した DB 作成テスト時に

使用したTest.javaを用い，テキストファイルを読み込んでDBに格納し，値の更新や削除を行った後変更したDBからテキストファイルへと上書きまでの動作が正しく行われたかを確認する．使用するデータセットは，与えられたdataset_example.txtとした．

dataset_example.txtを操作するためのText.javaをソースコード7に示す．

ソースコード 7: Text.javaのmainメソッド

```
1  public static void main(String arg[]){
2          // DBの作成
3          Presenter presenter = new Presenter(new View());
4          presenter.start();
5          // DBの書き換え
6          presenter.addData("Mike is a boy");
7          presenter.deleteData(1);
8          presenter.addData("Mike loves Hanako");
9          presenter.deleteData(5);
10         presenter.addData("Mike has a hobby of playing tennis");
11         presenter.deleteData(20);
12         presenter.addData("Mike is a boy");
13         // テキストファイルへの上書き
14         presenter.finish();
15 }
```

addDataでデータの追加を行い，deleteDataでデータの削除を行う．これらはそれぞれ引数にStringとintを持つ．またstart()でテキストファイルからDBへの書き込みを，finish()でDBからテキストファイルへの上書きを行う．これらはPresenterを経由してからDAOへと命令が伝えられる．これを実行するが，このとき変更された内容は繰り返し利用され，2回目の実行時には1回目の変更内容が反映された状態で新しくプログラムの更新を行わなければならない．それを踏まえ実装を行った実行結果が以下のようになる．

上のTest.javaの内容を3回繰り返して得られた結果を以下に順に示す．この時示すのは，結果が反映されたテキストファイルとする．

ソースコード 8: 実行前のdataset_example.txt

```
1  Hanako is a girl
2  Hanako is a student
3  student is a kind of human
4  human is a kind of mammal
5  Hanako has a hobby of playing video-games
6  Hanako has a hobby of playing air-guitar
7  Hanako studies philosophy
8  Hanako loves Taro
9
10 Taro is a boy
```

```
Taro is a student
Taro has a hobby of playing video-games
Taro studies informatics
Taro loves Jiro
Taro has a pet named Jiro

Jiro is a boy
Jiro is a dog
dog is a kind of mammal
Jiro has a hobby of playing frisbee
Jiro loves Hanako
```

ソースコード 9: 1 回実行後の dataset_example.txt

```
Hanako is a student
student is a kind of human
human is a kind of mammal
Hanako has a hobby of playing air-guitar
Hanako studies philosophy
Hanako loves Taro
Taro is a boy
Taro is a student
Taro has a hobby of playing video-games
Taro studies informatics
Taro loves Jiro
Taro has a pet named Jiro
Jiro is a boy
Jiro is a dog
dog is a kind of mammal
Jiro has a hobby of playing frisbee
Jiro loves Hanako
Mike loves Hanako
Mike has a hobby of playing tennis
Mike is a boy
```

ソースコード 10: 2 回実行後の dataset_example.txt

```
student is a kind of human
human is a kind of mammal
Hanako has a hobby of playing air-guitar
Hanako loves Taro
Taro is a boy
Taro is a student
Taro has a hobby of playing video-games
Taro studies informatics
Taro loves Jiro
Taro has a pet named Jiro
Jiro is a boy
Jiro is a dog
```

```
13 dog is a kind of mammal
14 Jiro has a hobby of playing frisbee
15 Jiro loves Hanako
16 Mike loves Hanako
17 Mike has a hobby of playing tennis
18 Mike is a boy
19 Mike loves Hanako
20 Mike has a hobby of playing tennis
21 Mike is a boy
```

ソースコード 11: 3 回実行後の dataset_example.txt

```
1 human is a kind of mammal
2 Hanako has a hobby of playing air-guitar
3 Hanako loves Taro
4 Taro is a student
5 Taro has a hobby of playing video-games
6 Taro studies informatics
7 Taro loves Jiro
8 Taro has a pet named Jiro
9 Jiro is a boy
10 Jiro is a dog
11 dog is a kind of mammal
12 Jiro has a hobby of playing frisbee
13 Jiro loves Hanako
14 Mike loves Hanako
15 Mike has a hobby of playing tennis
16 Mike is a boy
17 Mike loves Hanako
18 Mike is a boy
19 Mike is a boy
20 Mike loves Hanako
21 Mike has a hobby of playing tennis
22 Mike is a boy
```

4.4 考察

Test.java では，4 つの文の追加と 3 つの文の削除を行っている．このことから実行結果を確認すると，実行回数が増えるたびにテキストファイルに入っている文章の数が一つずつ増えていることが確認できる．また追加では Mike に関する情報を加えているが，実行回数が増えるたびに Mike に関する文章の数が増えていることも確認できる．これらのことから，このプログラムは前回実行時の内容を保持したまま正しく次の実行が行えていることが分かった．

DB 格納時に空行は飛ばして格納するようにしたことから，DB の内容をテキストファイルに格納する場合も空行は残らずに上書きが行われることも分かった．さらに，データの削除は DB 格納時に各文章に与えられる id 番号によって行われる

が，実行するたびにテキストファイルの一番上にある行が消えていることから，DB に格納が行われるたびに与えられる id 番号が正しく与え直され，その時のテキストファイルの一文目が削除されてることが確認できた．

発展課題ではあったものの，私が担当した範囲は必須課題 2-1 で作成したプログラムと DB に対しての操作を行うため，実装方法自体は大きく悩むことなく取り組むことが出来た．また，GUI との連動に必要なメソッドは Presenter.java によって操作が行われるため，私自身は GUI 本体を操作することなくプログラムを動作させることが出来た．これはプログラム作成時のアーキテクチャを明確にしていたことが影響しているため，プログラムを作成する前にアーキテクチャを明確にすることの大切さを強く理解した課題となった．

5　DB 関連についての全体の考察

ここまで，各課題に対する考察を行ってきたため，ここからは私が担当した DB 全体に対する考察を行う．

今回の課題は DB を作成するとの課題であったことから，プログラミング応用の講義で利用した sqlite を使用して DB を作成することを考えた．課題の要件を満たすような DB の形として，最終的に表 1 の格納方法を使用した．この理由としては，実装難度と実装期間と発展課題でデータの追加や削除を行う必要があると考えたため，DB を利用するとこれらの実装が行えると考えたからである．しかしこれでは，動作が終了すると DB で読み込んだ内容を最終的にテキストファイルへと保存することになり，DB はデータを保存するという意味では機能しなくなってしまう．DB は本来テキストファイルよりもデータの保存に適したものであるのにも関わらず，DB をテキストファイルのキャッシュのように使用することになってしまい，これでは DB を利用する利点が無くなってしまうと考えられた．

さらに，DB は table やカラムを持つことが出来ることもテキストファイルよりも保存方法に幅がある．しかしこの格納方法ではほとんどテキストファイルと同様の保存方法になっており，これも DB を有効に活用できていないと感じられる原因となった．

これらを解決するためにも，単語毎で分割し動詞毎で table を作成したり，単語毎でフィールドに追加していく方法も考えた (必須課題 2-1 考察参照)．しかしこの方法を利用すると，検索時に Matching クラスや Unify クラスを利用せずとも，変数が単語の分割された何個目に含まれているかという情報と各単語の内容で比較を行うことで検索が出来てしまうと考えられる．これでは課題の内容を満たしているとは言えないため，この方法は使用しないことを決めた．

Matching クラスや Unify クラスの利用の観点から考えると，先に検索を行い，

その際の検索文とその答えを DB に格納する方法が課題内容も満たし DB も活用できているため最も良いようにも考えられる．しかしこれでは発展課題のデータの追加や削除が DB で行えないため容易には実装出来なくなってしまう．追加や削除が行われたデータセットを最終的に保存しなければならないことを考えると，この方法も課題内容を満たさないと考えた．

このような考えから，今回は表 1 のように実装を行ったが，やはり DB を活用できているようにはあまり感じられない結果となった．そのため，時間に余裕があれば他の方法での実装も検討してみたいと考えた．例えば，検索文とその答えの格納を DB に行う場合は，初めにテキストファイルを読み込んだ際にその内容を List などに格納しておき，検索や追加，削除はその List を利用して行うようにし，最終的には変更された List の内容をテキストファイルに上書きする形を取ることで，発展課題の内容も満たすことが出来るのではないかと考えた．

また，データの追加や削除と課題にあったが，この「データ」とはデータセットを指すのか，検索文 (パターン) を指すのかも明確に指定されていないように感じた．今回私たちの班は「データ」をデータセットと考えたが，パターンと考えて実装を行うことも可能であると考えられた．そのため，検索文 (パターン) とその答えを格納する DB の作成は，課題の内容をより満たしたプログラムの作成となるのではないかとも考えた．

ここまでは DB の定義の仕方について考察したが，ここからは実際に作成した DB の実装について考察する．

DB の作成でまず工夫した点は，id 番号を引数で指定することなく決定することが出来るようにしたことである．テキストファイルから読み込んで一文ずつ DB に格納する際には 1 で初期化された変数を順にインクリメントすることで実現できる．しかし追加では別の DAO から DB にアクセスするため変数を取得することが難しい．そこで table のなかで最大の id を取得する関数 max(id) を利用するメソッド getNo を作成することで今使用されている最大の id 番号を取得し，それを 1 増やした値を追加されたデータの id 番号とした．

また，作成された DB の扱い方についても工夫を行った．初めに作成したプログラムでは，同じ table 名を作成しようとするとエラーが発生するため，一度実行して作成した DB は次の実行の前に自分で削除しなければならなかった．これを解決するため，まず作成した DB を実行終了と同時に消すことを考えた．しかし sqlite では DB そのものを消す命令が存在していなかったため，この方法は実現させることが出来なかった．

次に，実行終了時に DB ではなく table を消すことを考えた．これは drop table という命令で可能であることが分かったので，これを実装したところ，二回目以降の実行では table が消えているためエラーが発生しなくなった．しかしこれでは，1 回目の実行時に別の main で実行して残っていたプログラムが存在しているとエ

ラーが発生することが分かった．また，実行終了時に table を消してしまうため，実行終了時に DB にはデータが残らなくなってしまう．そのため，この方法よりも良い実装を考えた．

その結果考えたものが，実行開始時に table を消去する方法である．しかしこれは table が存在しなかった時にエラーが発生してしまう．そのため table の存在の有無を判定しようと調べたところ，table が存在していないときにだけ table を新しく作成することが出来ることが分かった．ここから，table を作成した後にその中身を削除することを考えた．そこで deleteData メソッドを作成し，table の中身を全て削除してからテキストファイルのデータを書き込むことが出来るようになった．よって DB の有無にかかわらず，正しく読み込んだテキストファイルの内容を格納することが出来るプログラムを作成することが出来た．

これは求めたい結果が得られるまでにかなり遠回りをしてしまったが，プログラム改良のプロセスを体感できたように感じた．

今回作成した DB では，追加の際に既に入っているデータをもう一度追加すると，同一内容でもはじくことなく新しいデータとして格納してしまう．これではデータセットの内容の重複が認められてしまうことになるため，同一内容の文章ははじくための処理を入れるべきだった．Matching クラスや Unify クラスを用いれば内容が一致しているかどうかの判断は可能だと感じた．今回は時間がなくできなかったが，追加処理を行う際は現在入っているデータと追加文が重複しているかを確認し，重複していないときにのみデータへの追加を行うように実装したい．

また，Moodle の質問掲示板の返信から，そもそも既存の DB を利用せずに課題を解かなければならなかったことが分かった．そのため，今までの考え方を大きく変え自作 DB の実装方法を考えなければならなかったが，今回は時間がなく考えることが出来なかった．自作 DB の実装方法はまだどのように行えばよいのか明確に分かっていないため，今後のためにもどのような方法が利用できるのか考えていきたい．

6　感想

今回は，実装期間が一週間と短かったため，実現させたかったことをすべて実現させることが出来なかったのが残念だった．また，講義の時間以外に全員で集まることが中々出来なかったため，それぞれの考えを統一するのに時間がかかってしまった．そのため，講義の時間などを利用して積極的に話し合いをし，全員の考えと方向性を統一させることが大切であると感じた．

また今回は全員で一つのプログラムを分担して実装したため，それぞれの担当箇所を接続する際にエラーが発生することが非常に多かった．その上，お互いが別の部分を担当していることから互いの担当範囲について詳しくないため，デバッ

グ作業にかなりの時間を要してしまった．プログラムを分担して作成する大変さも感じた．

しかしそのため，お互いに協力したりそれぞれで調べるなど，課題に対する知識は増えたように感じられる．特に私は DB に関連するプログラムの作成を行ったため，DB に関連する知識が以前よりも増えたと感じられる．DB の知識を身に着けることが出来たのはとても良かった．

参考文献

[1] プログラミング応用で作成した DAO 等のプログラム

[2] RUGBY WORLD CUP, https://www.rugbyworldcup.com/ （2019 年 10 月 29 日アクセス）.

[3] DBOnline SQLite 入門, https://www.dbonline.jp/sqlite/ （2019 年 10 月 29 日アクセス）.

[4] memocarilog, https://memocarilog.info/php-mysql/5355 （2019 年 10 月 29 日アクセス）.

[5] プログラミングマガジン　【デザインパターン】DAO/DTO パターン, www.code-magagine.com/?p=1311 （2019 年 10 月 29 日アクセス）.

[6] CMO クラウドアカデミー　データベースの用語を理解しよう「テーブル」「レコード」「カラム」「フィールド」とは？, https://academy.gmocloud.com/know/20160425/2259 （2019 年 10 月 29 日アクセス）.