

システム開発プロジェクト総合実践・補助資料

2020 年 5 月 18 日～5 月 22 日

(Java プログラミング)



Web Applications
Windows Store Apps Using C#

2020 年 5 月 7 日



Windows Azure Developer
Windows Phone Developer
Windows® Developer 4
Enterprise Application Developer 3.5

ニュートラル株式会社
林 広宣

<http://www.neut.co.jp/>
hayashi.hironori@neut.co.jp

複製を禁ず

(このページが、表紙の裏になるように印刷をお願いします。)

ご意見・ご質問は下記まで

ニュートラル株式会社 林広宣
hayashi.hironori@neut.co.jp

目 次

システム開発プロジェクト総合実践・補助資料	1
目 次	3
第1章. はじめに	5
1.1. 課題の説明	6
第2章. データベース構造	9
2.1. データベース全体の構造	10
2.2. 各テーブルの構造	11
第3章. データベースプログラミング	17
3.1. 事前準備	18
3.2. 簡単なプログラム	22
3.3. オブジェクトに読み込む	29

(空白ページ)

第1章. はじめに

1.1. 課題の説明

1.1.1.概要

フライトチケットの予約をシミュレーションするプログラムを作成する。

1.1.2.一例

あくまで例であるが、下記のような機能を持ったアプリケーション。

```
予約したい日付の「月」を入力してください。(1--12, END=999) >5
予約したい日付の「日」を入力してください。(1--31, END=999) >30

出発する空港を選んでください。
1..札幌・新千歳
2..福岡
3..東京・羽田
4..大阪・伊丹
5..大阪・関西
6..名古屋・中部
7..成田

番号は？(END=999)>2

=====
出発地は、FUK-福岡 です。

目的地の空港を選んでください。
1..札幌・新千歳
2..福岡
3..東京・羽田
4..大阪・伊丹
5..大阪・関西
6..名古屋・中部
7..成田

番号は？(END=999)>6

=====
目的地は、NGO-名古屋・中部 です。

人数は何人ですか？
1..1人 2..2人 3..3人 4..4人 5..5人
番号は？(1--5, END=999) >3
```

どの便を利用しますか？

- 1.. AB012 09:10-10:40
- 2.. AB014 21:10-22:40

番号は？ (END=999) >1

選択したフライトは、AB012 (2020-05-30) 福岡 09:10 名古屋・中部 10:40 です。

どのクラスを利用しますか？

- 1.. スーパーバリュー ¥11200/人 (3名 ¥33600)
- 2.. 普通席 ¥16000/人 (3名 ¥48000)
- 3.. プレミアム ¥24000/人 (3名 ¥72000)

番号は？ (1--5, END=999) >3

空席を確認中です...

予約可能です。

クレジットカードの番号を入力してください。 (END=999) >111122223333

予約者の氏名を入力してください。 (END=999) >紫式部

予約できました。

下記の予約番号で搭乗券と引き換えてください。

予約番号: 4

予約者氏名: 紫式部

予約情報

AB012 便 2020年5月30日 福岡 09:10 発 - 名古屋・中部 10:40 着
プレミアム 3名 ¥72000

終了します。

続行するには何かキーを押してください...

この結果、データベース (flight.accdb) の [D_予約] というテーブルにデータが1件追加される。
(下図参照)

D_予約							
予約番号	予約者氏名	クレジットカード番号	搭乗者数	単価	フライト日付	フライト番号	料金区分
1	徳川家康	123456789012	3	¥25,000	2020/05/22	AB013	PS
2	上杉謙信	987654321098	3	¥24,000	2020/06/02	AB014	PS
3	源頼朝	500212349876	1	¥30,000	2020/06/03	AB026	NS
4	紫式部	111122223333	3	¥24,000	2020/05/30	AB012	PS

1.1.3.制約

国内線の予約のみとし、次の 7 つの空港の便に限定する。

1. . 札幌・新千歳
2. . 福岡
3. . 東京・羽田
4. . 大阪・伊丹
5. . 大阪・関西
6. . 名古屋・中部
7. . 成田

予約できる便の期間は 2020/5/15 から 2020/6/30 までとする。

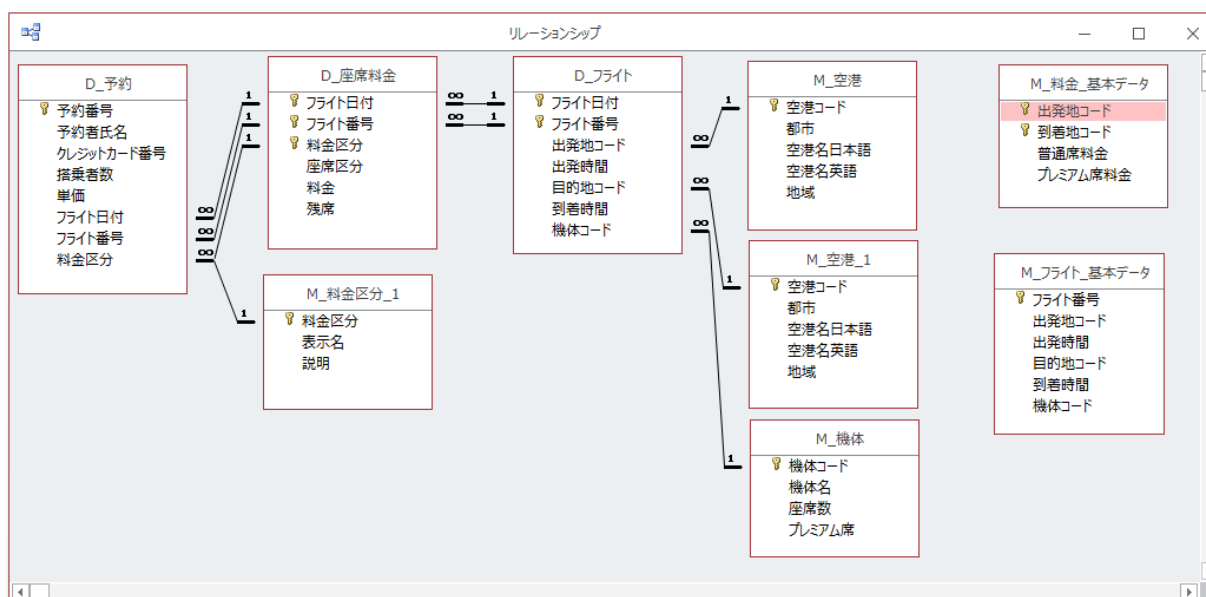
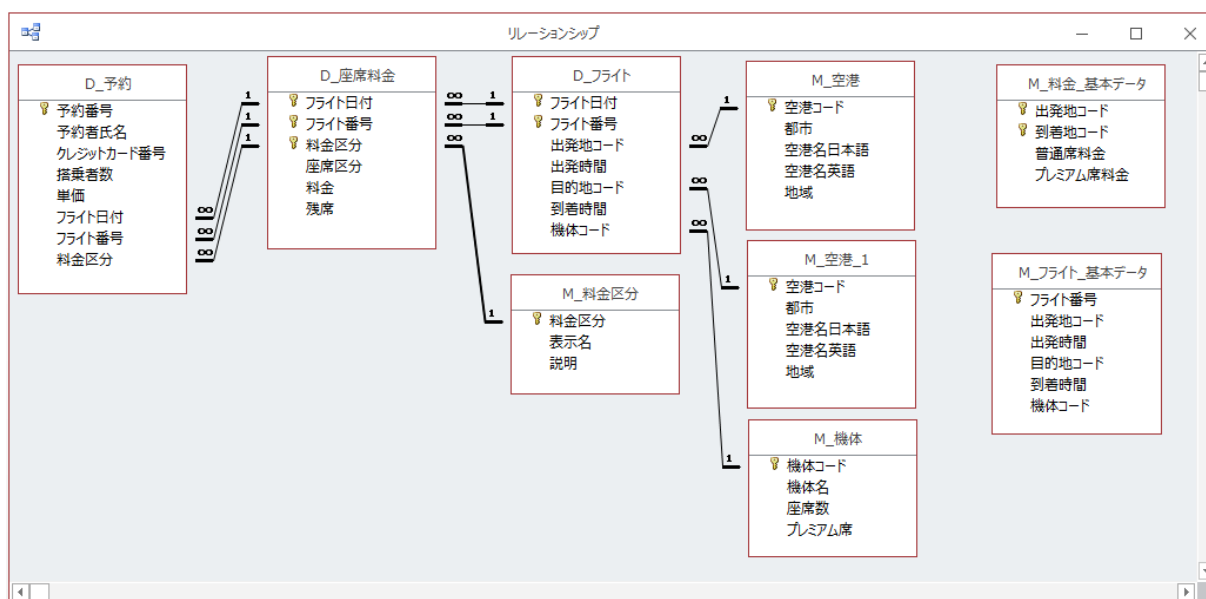
1.1.4.目標

できる範囲までの機能を実装すればよい。

例えば、予約機能まで作成できなくとも、「日付と空港コードを入力すると、その日に発着するフライトの一覧を表示する」というような部分的な機能が、正しく動作すればよい。

第2章. データベース構造

2.1. データベース全体の構造



「M_料金_基本データ」と「M_フライト_基本データ」は、実習では使用しない。
 これらは、「D_座席料金」や「D_フライト」を生成するためのものである。

2.2. 各テーブルの構造

2.2.1.D_予約

今回作成するアプリケーションが最終的に生成するデータがこのデータとなる。

「誰がいつのどのフライトを予約し、人数何人でいくらだったか」というような情報を格納する。

フィールド名	データ型	説明(オプション)
予約番号	数値型	int
予約者氏名	短いテキスト	String
クレジットカード番号	短いテキスト	String
搭乗者数	数値型	int
単価	通貨型	int
フライト日付	日付/時刻型	LocalTime
フライト番号	短いテキスト	String
料金区分	短いテキスト	String

予約番号	予約者氏名	クレジットカード番号	搭乗者数	単価	フライト日付	フライト番号	料金区分
1	徳川家康	123456789012	3	¥25,000	2020/05/22	AB013	PS
2	上杉謙信	987654321098	3	¥24,000	2020/06/02	AB014	PS
3	源頼朝	500212349876	1	¥30,000	2020/06/03	AB026	NS
* 0			0	¥0			

レコード: 4 / 4 フィルターなし 検索

2.2.2.D_座席料金

予約したフライトの料金と残席数を格納しているテーブルである。

D_座席料金		
フィールド名	データ型	説明(オプション)
フライト日付	日付/時刻型	
フライト番号	短いテキスト	
料金区分	短いテキスト	NS..普通席通常料金、ND..普通席割引料金、PS..プレミアム席通常料金
座席区分	短いテキスト	N..普通席、P..プレミアム席
料金	通貨型	
残席	数値型	

D_座席料金						
フライト日付	フライト番号	料金区分	座席区分	料金	残席	
2020/05/15	AB001	NS	N	¥18,000	39	
2020/05/15	AB001	PS	P	¥28,000	0	
2020/05/15	AB002	NS	N	¥15,000	39	
2020/05/15	AB002	PS	P	¥25,000	0	
2020/05/15	AB003	NS	N	¥18,000	70	
2020/05/15	AB003	PS	P	¥28,000	0	
2020/05/15	AB004	NS	N	¥15,000	70	
2020/05/15	AB004	PS	P	¥25,000	0	
2020/05/15	AB005	NS	N	¥18,000	39	
2020/05/15	AB005	PS	P	¥28,000	0	
2020/05/15	AB006	NS	N	¥15,000	39	
2020/05/15	AB006	PS	P	¥25,000	0	
2020/05/15	AB011	NS	N	¥17,000	186	
2020/05/15	AB011	PS	P	¥25,000	8	
2020/05/15	AB012	ND	N	¥11,200	20	
2020/05/15	AB012	NS	N	¥16,000	166	
2020/05/15	AB012	PS	P	¥24,000	8	
2020/05/15	AB013	NS	N	¥17,000	112	
2020/05/15	AB013	PS	P	¥25,000	8	
2020/05/15	AB014	ND	N	¥11,200	20	
2020/05/15	AB014	NS	N	¥16,000	92	
2020/05/15	AB014	PS	P	¥24,000	8	
2020/05/15	AB021	NS	N	¥30,000	112	
2020/05/15	AB021	PS	P	¥40,000	8	

2.2.3.M_料金区分

座席によって料金が異なる。その種類を定義したテーブルである。

金額はフライトと日付によって異なるので、前述の D_座席料金テーブルに定義されている。

M_料金区分		
フィールド名	データ型	説明(オプション)
料金区分	短いテキスト	
表示名	短いテキスト	
説明	短いテキスト	

M_料金区分		
料金区分	表示名	説明
ND	普通料金	普通席 通常料金
NS	スーパーバリュー	普通席 割引料金
PS	プレミアム	プレミアム席 通常料金
*		

レコード: 4 / 4 フィルターなし 検索

2.2.4.D_フライト

フライトとは、フライト番号と日付が主キーとなっているテーブルである。

出発地と目的地、出発時間と到着時間、機体の種類が格納されている。

D_フライト		
フィールド名	データ型	説明(オプション)
フライト 日付	日付/時刻型	
フライト 番号	短いテキスト	
出発地コード	短いテキスト	[M_空港]の[空港コード]
出発時間	日付/時刻型	
目的地コード	短いテキスト	[M_空港]の[空港コード]
到着時間	日付/時刻型	
機体コード	短いテキスト	[M_機体]の[機体コード]

D_フライト							
フライト 日付	フライト 番号	出発地コード	出発時間	目的地コード	到着時間	機体コード	
2020/05/15	AB001	NGO	6:00:00	NRT	7:00:00	Q82	
2020/05/15	AB002	NRT	9:00:00	NGO	10:00:00	Q82	
2020/05/15	AB003	NGO	12:00:00	NRT	13:00:00	CR7	
2020/05/15	AB004	NRT	15:00:00	NGO	16:00:00	CR7	
2020/05/15	AB005	NGO	18:00:00	NRT	19:00:00	Q82	
2020/05/15	AB006	NRT	21:00:00	NGO	22:00:00	Q82	
2020/05/15	AB011	NGO	6:10:00	FUK	7:40:00	321	
2020/05/15	AB012	FUK	9:10:00	NGO	10:40:00	321	
2020/05/15	AB013	NGO	18:10:00	FUK	19:40:00	73P	
2020/05/15	AB014	FUK	21:10:00	NGO	22:40:00	73P	
2020/05/15	AB021	NRT	6:00:00	FUK	8:00:00	73P	
2020/05/15	AB022	FUK	9:00:00	NRT	11:00:00	73P	
2020/05/15	AB023	NRT	12:00:00	FUK	14:00:00	CR7	
2020/05/15	AB024	FUK	15:00:00	NRT	17:00:00	CR7	
2020/05/15	AB025	NRT	18:00:00	FUK	20:00:00	789	
2020/05/15	AB026	FUK	21:00:00	NRT	23:00:00	789	
2020/05/16	AB001	NGO	6:00:00	NRT	7:00:00	Q82	
2020/05/16	AB002	NRT	9:00:00	NGO	10:00:00	Q82	
2020/05/16	AB003	NGO	12:00:00	NRT	13:00:00	CR7	
2020/05/16	AB004	NRT	15:00:00	NGO	16:00:00	CR7	
2020/05/16	AB005	NGO	18:00:00	NRT	19:00:00	Q82	
2020/05/16	AB006	NRT	21:00:00	NGO	22:00:00	Q82	
2020/05/16	AB011	NGO	6:10:00	FUK	7:40:00	321	
2020/05/16	AB012	FUK	9:10:00	NGO	10:40:00	321	

2.2.5.M_空港

空港を表すテーブル。

M_空港		
フィールド名	データ型	説明(オプション)
空港コード	短いテキスト	
都市	短いテキスト	
空港名日本語	短いテキスト	
空港名英語	短いテキスト	
地域	短いテキスト	

M_空港				
空港コード	都市	空港名日本語	空港名英語	地域
CTS	札幌・新千歳	新千歳空港	New Chitose Airport	北海道
FUK	福岡	福岡空港	Fukuoka Airport	福岡県
HND	東京・羽田	東京国際空港(羽田空	Tokyo International Airport (Haneda)	東京都
ITM	大阪・伊丹	大阪国際空港・伊丹空	Osaka International Airport - Itami Airport	大阪府
KIX	大阪・関西	関西国際空港	Kansai International Airport	大阪府
NGO	名古屋・中部	中部国際空港	Chubu Centrair International Airport	愛知県
NRT	成田	成田国際空港	Narita International Airport	千葉県
*				

レコード: 1 / 7 フィルターなし 検索

2.2.6.M_機体

機体の種類（エアバス XXX、ボーイング XXX など）とその期待の座席数を格納したテーブル。

M_機体		
フィールド名	データ型	説明 (オプション)
機体コード	短いテキスト	
機体名	短いテキスト	
座席数	数値型	普通席の座席数は[座席数]－[プレミアム席]で求められる。
プレミアム席	数値型	

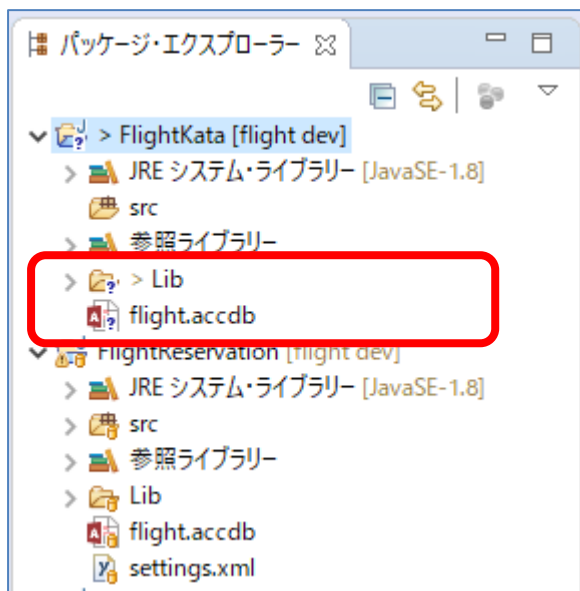
M_機体				
機体コード	機体名	座席数	プレミアム席	
+ 320	エアバス A321	166	0	
+ 321	エアバス A321	194	8	
+ 735	ボーイング 737-500	126	0	
+ 738	ボーイング 737-800	167	8	
+ 73P	ボーイング 737-700	120	8	
+ 789	ボーイング 787-9	395	18	
+ CR7	ボンバルディア CRJ-700	70	0	
+ Q82	ボンバルディア DHC8-Q2000	39	0	
*		0	0	
レコード: 9 / 9 フィルターなし 検索				

第3章. データベースプログラミング

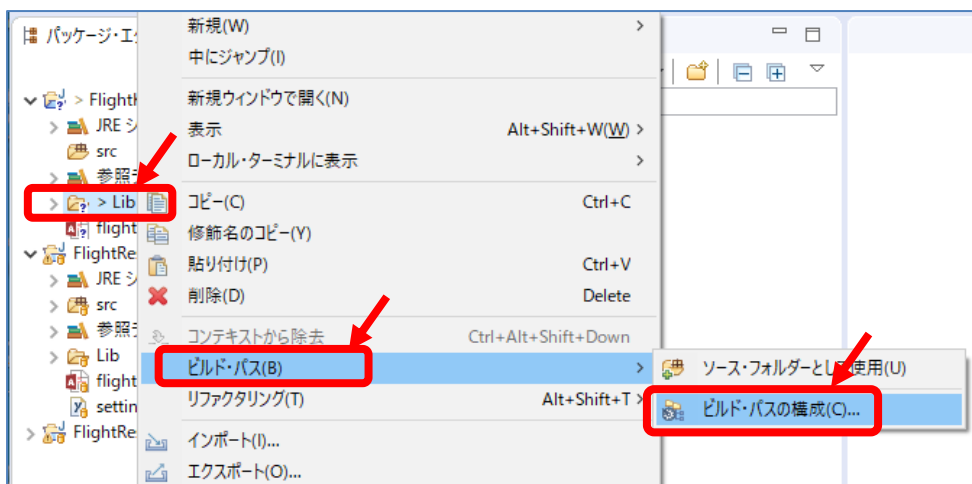
3.1. 事前準備

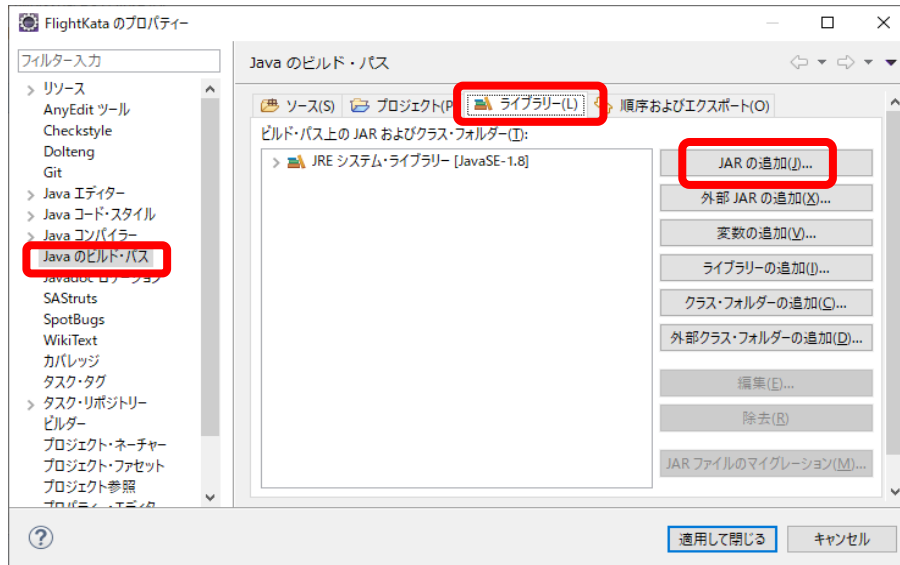
データベース (flight. accdb) を読み書きできるようにするための準備。

Eclipse でプロジェクトを作成し、プロジェクト直下に (pre_req 内の) Lib フォルダをコピーする。
同様に、flight. accdb ファイルもコピーする。



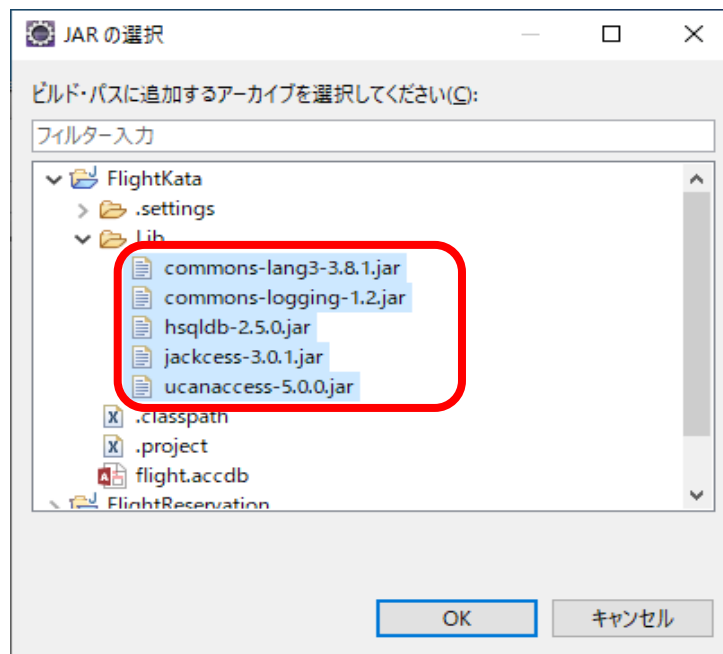
プロジェクトの Lib フォルダを右クリックし、[ビルド・パス] — [ビルド・パスの構成] とたどっていく。





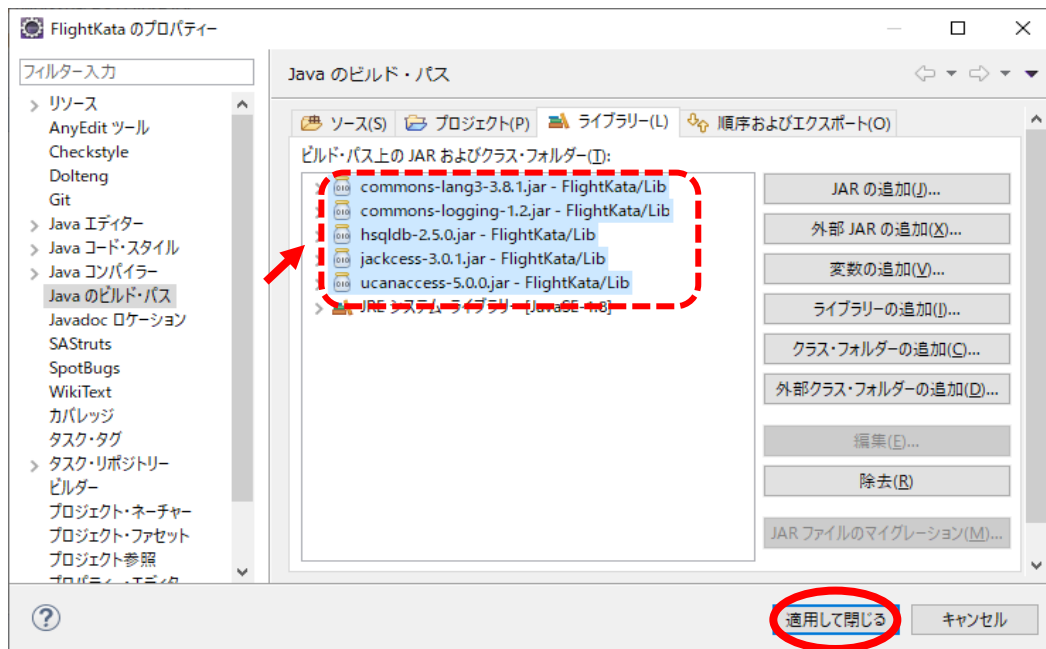
[Shift] キーを押しながら、Lib フォルダー内の JAR ファイルをすべて選択し、[OK] ボタンをクリックする。

全て選ぶのではなく、1 つずつ選んで、この処理を繰り返し、すべて (5 つ) の JAR ファイルが追加されるようにしてもよい。



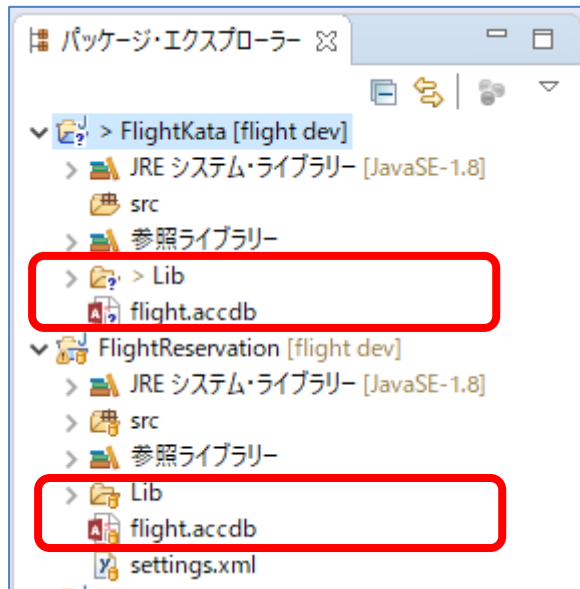
第3章.データベースプログラミング(3.1.事前準備)

下図のように JAR ファイルが追加されていることを確認し、[適用して閉じる] ボタンをクリックする。



以上の作業は、プロジェクトを作成するたびごとに実施すること。

下図は2つのプロジェクト (FlightKata と FlightReservation) に対して、同じ作業が実施されたことを示している。



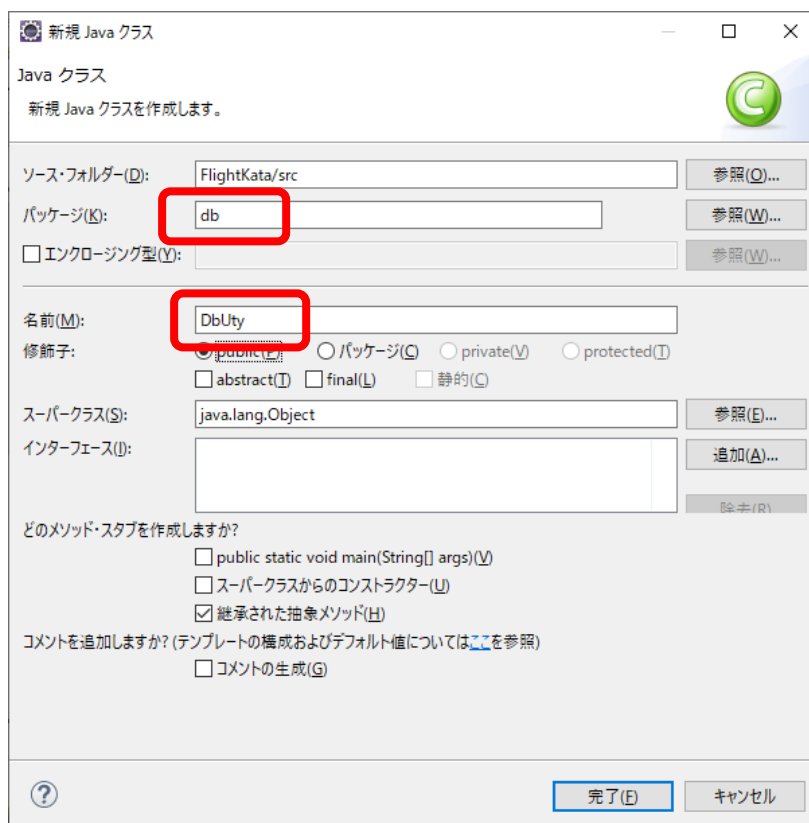
3.2. 簡単なプログラム

3.2.1.注意事項

1. パッケージを必ず指定しよう

クラスを作成するときは、必ずパッケージ名も指定して作成するようにしよう。
パッケージは、クラスの機能の役割分担だけでなく、チーム内のプログラマーの役割分担にも利用できる。

下記は、db.DBQty というクラスを Eclipse で作成する場合の例。



3.2.2.とにかくテーブルの内容を一覧表示する。

1. クラス main.Sample01

```
package main;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class Sample01 {
    public static void main(String[] args) {

        try {
            String prefix = "jdbc:ucanaccess://";
            String fname = "flight.accdb";
            // このプログラムが起動された場所（フォルダー）を取得する。
            String dir = System.getProperty("user.dir");
            //
            String url = prefix + dir + "/" + fname;
            //
            // 上記までの結果、url は、例えば次のような文字列となる。
            //
            //      jdbc:ucanaccess://C:/Java/db/flight.accdb
            //
            //
            String sql_select = "SELECT * FROM M_機体";
            //
            Connection conn = DriverManager.getConnection(url);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(sql_select);
            while (rs.next()) {
                System.out.println(
                    String.format(
                        "%s %s %s",
                        rs.getString(1),
                        rs.getString(2),
                        rs.getString(3)
                    )
                );
            }
        } catch (SQLException e) {
            System.out.println("err: " + e.getMessage());
        }

    }
}
```

2. 実行結果

```
320 エアバス A321 166
321 エアバス A321 194
735 ボーイング 737-500 126
738 ボーイング 737-800 167
73P ボーイング 737-700 120
789 ボーイング 787-9 395
CR7 ボンバルディア CRJ-700 70
Q82 ボンバルディア DHC8-Q2000 39
```


3.2.3.何度も利用するコードは1カ所にまとめよう

1. クラス db.DbUty

```
package db;

public class DbUty {

    //
    // 起動ディレクトリの flight.accdb を指し示す JDBC 用の URL の取得
    //
    public static String getDbUrl() {
        //
        // 下記の4行（コメントを入れると6行）は、Sample01 からコピーしてよい。
        //
        String prefix = "jdbc:ucanaccess://";
        String fname = "flight.accdb";
        // このプログラムが起動された場所（フォルダー）を取得する。
        String dir = System.getProperty("user.dir");
        //
        String url = prefix + dir + "/" + fname;
        //
        return url;
    }
}
```

2. クラス main.Sample02

```
package main;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import db.DbUty;

public class Sample02 {
    public static void main(String[] args) {

        try {
            String url = DbUty.getDbUrl();
            //
            String sql_select = "SELECT * FROM M_空港";
            //
            Connection conn = DriverManager.getConnection(url);
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(sql_select);
            while (rs.next()) {
                System.out.println(
                    String.format(
                        "%s %s %s",
                        rs.getString(1),
                        rs.getString(2),
                        rs.getString(3)
                    )
                );
            }
        } catch (SQLException e) {
            System.out.println("err: " + e.getMessage());
        }

    }
}
```

3. 実行結果

```
CTS 札幌・新千歳 新千歳空港
FUK 福岡 福岡空港
HND 東京・羽田 東京国際空港（羽田空港）
ITM 大阪・伊丹 大阪国際空港・伊丹空港
KIX 大阪・関西 関西国際空港
NGO 名古屋・中部 中部国際空港
NRT 成田 成田国際空港
```

3.2.4.大量のデータから一部だけ抽出しよう

1. クラス main.Sample03

```

package main;

import java.sql.Connection;
import java.sql.Date;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.LocalDate;

import db.DbUty;

public class Sample03 {
    public static void main(String[] args) {

        try {
            String url = DbUty.getDbUrl();
            //
            String sql_select =
                "SELECT * FROM D_フライト " +
                " WHERE フライト日付=?" +
                " AND 出発地コード=?" +
                "";

            //
            // パラメータを補完するデータ (フライト日付と出発地コード)
            // (このデータを変えて、結果を確認してみよう)
            //
            LocalDate flightDate = LocalDate.of(2020, 5, 15);
            String fromAirport = "NGO";
            //
            Connection conn = DriverManager.getConnection(url);
            PreparedStatement ppst = conn.prepareStatement(sql_select);
            ppst.setDate(1, Date.valueOf(flightDate));
            ppst.setString(2, fromAirport);
            ResultSet rs = ppst.executeQuery();

            while (rs.next()) {
                System.out.println(
                    String.format(
                        "%s %s %s %s %s %s %s",
                        rs.getString(1),
                        rs.getString(2),
                        rs.getString(3),
                        rs.getString(4),
                        rs.getString(5),
                        rs.getString(6),
                        rs.getString(7)
                    )
                );
            }
        } catch (SQLException e) {
            System.out.println("err: " + e.getMessage());
        }
    }
}

```

2. 実行結果

フライト日付 : 2020/5/15、出発地 : NGO の場合

```
2020-05-15 00:00:00.000000 AB001 NGO 1899-12-30 06:00:00.000000 NRT 1899-12-30 07:00:00.000000 Q82
2020-05-15 00:00:00.000000 AB003 NGO 1899-12-30 12:00:00.000000 NRT 1899-12-30 13:00:00.000000 CR7
2020-05-15 00:00:00.000000 AB005 NGO 1899-12-30 18:00:00.000000 NRT 1899-12-30 19:00:00.000000 Q82
2020-05-15 00:00:00.000000 AB011 NGO 1899-12-30 06:10:00.000000 FUK 1899-12-30 07:40:00.000000 321
2020-05-15 00:00:00.000000 AB013 NGO 1899-12-30 18:10:00.000000 FUK 1899-12-30 19:40:00.000000 73P
```

1899-12-30 という日付には意味がない。この列で意味があるのは、時間の部分(時分秒)だけである。

フライト日付 : 2020/5/30、出発地 : NRT の場合

```
2020-05-30 00:00:00.000000 AB002 NRT 1899-12-30 09:00:00.000000 NGO 1899-12-30 10:00:00.000000 Q82
2020-05-30 00:00:00.000000 AB004 NRT 1899-12-30 15:00:00.000000 NGO 1899-12-30 16:00:00.000000 CR7
2020-05-30 00:00:00.000000 AB006 NRT 1899-12-30 21:00:00.000000 NGO 1899-12-30 22:00:00.000000 Q82
2020-05-30 00:00:00.000000 AB021 NRT 1899-12-30 06:00:00.000000 FUK 1899-12-30 08:00:00.000000 73P
2020-05-30 00:00:00.000000 AB023 NRT 1899-12-30 12:00:00.000000 FUK 1899-12-30 14:00:00.000000 CR7
2020-05-30 00:00:00.000000 AB025 NRT 1899-12-30 18:00:00.000000 FUK 1899-12-30 20:00:00.000000 789
```

3.3. オブジェクトに読み込む

3.3.1.概要

データをデータベースから読み込んで画面に表示する、あるいは画面で入力されたデータをデータベースに格納するというような処理の流れの場合、データを格納するオブジェクトには3種類考えられる。

1. ビジネス・エンティティ (Business Entity)

DB や画面に依存せず、ビジネスモデルを素直に表現したオブジェクト。
社内の他のプロジェクトでも使用される。

データベースから読み書きする場合は、DTO を経由して行われる。
画面に表示／画面から入力される場合は、ビュー・モデルを経由して行われる。
「顧客オブジェクト」「商品オブジェクト」「注文明細オブジェクト」など。

2. DTO (Data Transfer Object データ・トランスファー・オブジェクト)

データベースの構造に即したオブジェクト。
プログラムしやすいようにテーブルのカラムと、ほぼ1対1対応するように設計することが多い。

なお、DB から DTO へ読み込んだり、DTO から DB へ書き込んだりする処理を行うオブジェクトを、DAO (Data Access Object) と呼ぶ。

例えば、ビジネス・エンティティとして「注文オブジェクト」があったとする。
「注文オブジェクト」はさらに「顧客オブジェクト」と「商品オブジェクト」を含んでいるとする。
(〇〇さんが▲▲商品を注文した)
このような場合は、オブジェクト指向では注文オブジェクトの内部に直接「顧客オブジェクト」や「商品オブジェクト」を含めることができるが、リレーショナルデータベースではそれができないので、その代わりに「顧客 ID」「商品 ID」というインデックス (コード番号) を保持する。
(いわゆる外部キー)

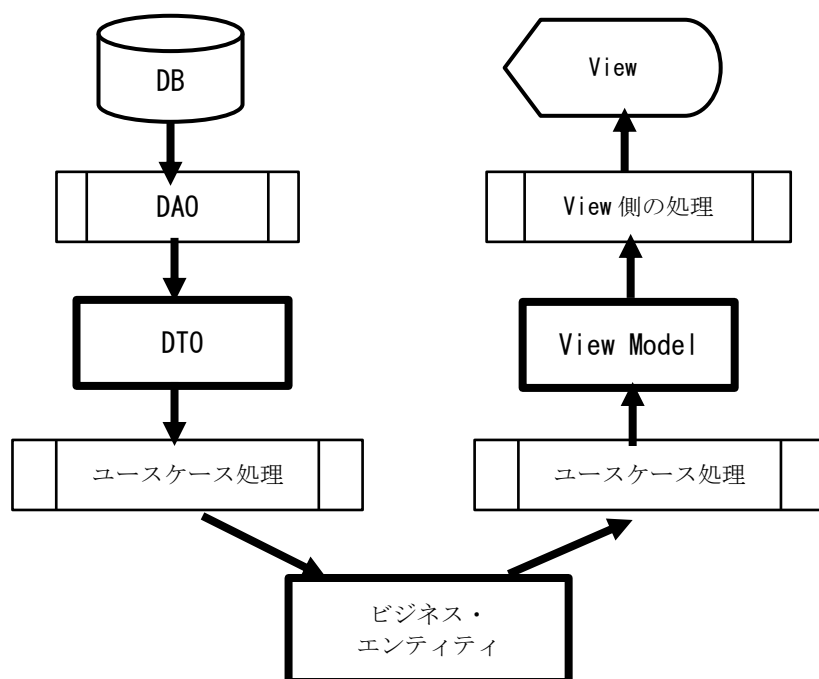
このように、オブジェクト指向とリレーショナルデータベースでは、データの保持の方法が異なることがあるので、データベースから読み込んだデータは、直接ビジネス・エンティティに保持するのではなく、いったん DTO に保持するということがよく行われる。

3. ビュー・モデル (View Model)

画面 (ビュー) に表示しやすいように、また、画面の入力を受け取りやすいように設計されたオブジェクト。

ビジネス・エンティティには色や文字列フォーマットなどの情報は存在しないが、ビュー・モデルには存在する。
例えば、ビジネス・エンティティに「金額」という情報があつたとすると、ビュー・モデルには「金額」だけでなく、「マイナスの場合は赤色、プラスの場合は黒色」とか「3桁ごとにカンマ区切りをして通貨記号を付ける」というような情報も保持する。

矢印の向きは、データの流れを示す。(依存関係や処理の流れを示すものではない)
なお、処理の流れは、この矢印とは正反対の向きになることが多い。



3.3.2.3 種類は多すぎるとする場合

データの内容としては1種類のデータを格納するとに、形式としては3種類も定義するのは煩雑すぎる場合がある。

例えば、DTO もビジネス・エンティティも **ViewModel** もほとんど同じ形式で構わないような場合である。

そのような場合にまで、わざわざ3種類のデータを定義する必要はない。
最も重要なデータ形式は、ビジネス・エンティティなので、まずそれを定義し、DTO と **ViewModel** は省略してかまわない。

その場合は、**DAO** は直接ビジネス・エンティティに読み込んだり書き込んだりする。また、画面は直接ビジネス・エンティティの情報を表示したり格納したりする。

DTO だけ省略する、**ViewModel** だけ省略するというのもしばしば行われる。