

学習目標

今回は、いよいよ、Javaのプログラムからデータベースにアクセスする方法を学びます。今回の学習モジュールを修了すると、次のようなことができるようになるはずです。そうなるように学習しましょう。

- EclipseでMySQLを操作するJavaプログラムを作成する際に必要なコネクタを設定できる。
- MySQLを操作するJavaのプログラムの大まかな処理の流れを説明できる
- JavaプログラムでMySQLに接続する処理を記述できる
- MySQLのテーブルからデータを検索するJavaプログラムを記述できる
- MySQLのテーブルにデータを挿入するJavaプログラムを記述できる

プログラミング例題の実行に際して

プログラミング例題では全体のソースリストを示すようにしていますが、まずは、解説を読み、自分でコードを入力しながら実習をするようにしましょう。その方が理解が進みます。

Eclipseでは文法エラーの場所がわかりますので、その部分がどうしてもわからない場合は、ソースコードを確認するようにするとよいでしょう。

こうした姿勢は、第9回以降で例題プログラムが出てきたときも同様です。

学習の準備

Eclipseの準備

この実習では、Eclipseを使ってJavaプログラムを作成します。プログラミング1～4で利用したものがあればそれを使えばよいでしょう。科目等履修生等で、Eclipse をインストールしていない場合は、まずはEclipseをインストールしてください。

容量が大きいのですが、以下のサイトからPleiades All in One をダウンロードすると、最初から日本語化されています。

- <http://mergedoc.sourceforge.jp/>

どのバージョンでもよいですが、この授業ではEclipse 3.5 Galileo のJavaに沿って説明をしていきます。

MySQLコネクタの準備

次に、JavaでMySQLを操作するためにはコネクタが必要になりますので、次のサイトから、mysql-connector-javaをダウンロードしておきます。Windowsの場合は、Source and Binaries (zip) の方をダウンロードすればよいでしょう。

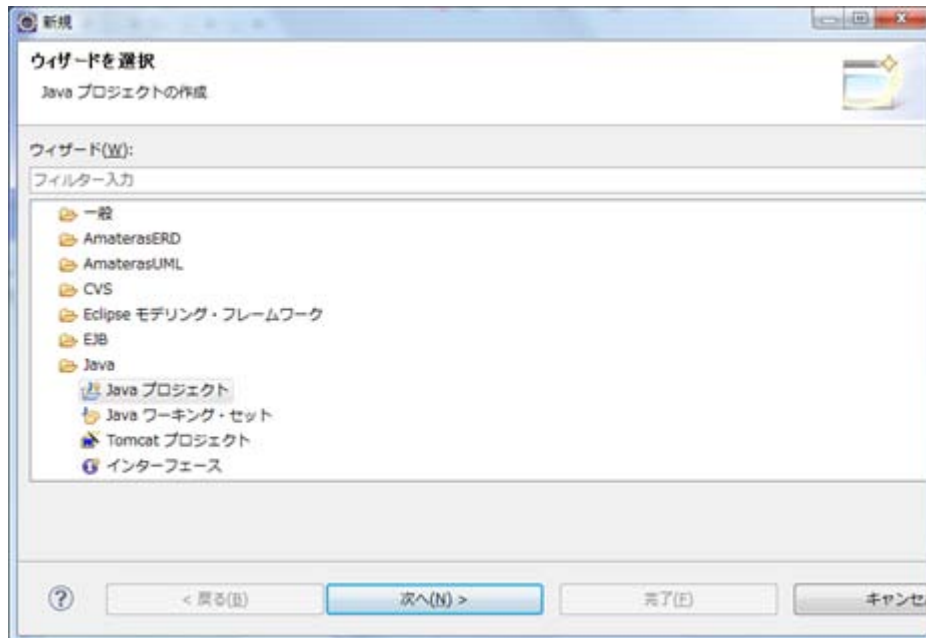
- <http://dev.mysql.com/downloads/connector/j/5.0.html>

zipファイルを展開して、ワークスペースのフォルダに mysql-connector-java-[バージョン]のフォルダをコピーしておきます。

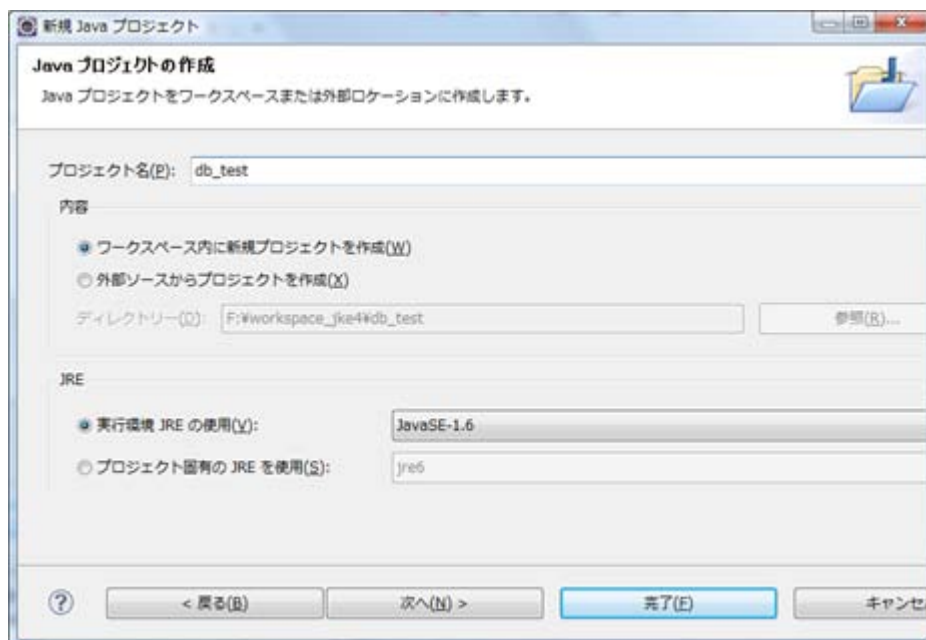
プロジェクトの作成

まず、最初にプロジェクトを作成します。種類はJavaプロジェクトです。Eclipse 3.5 では以下のような手順になります。

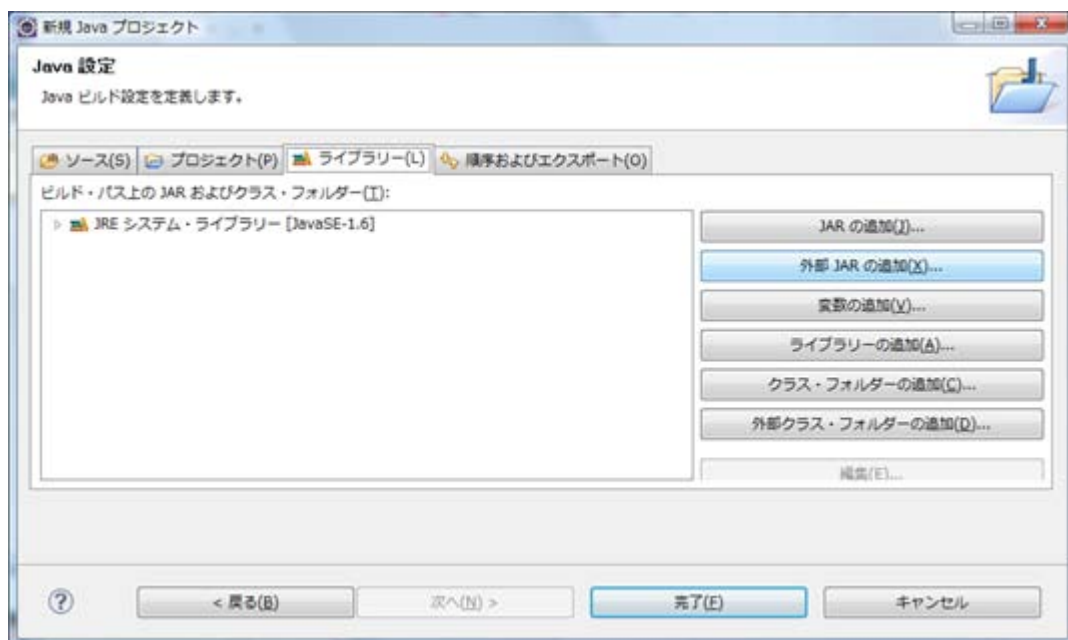
「ファイル」 → 「新規作成」 → 「その他」 で、Java の下のJavaプロジェクト を選択し、次に進みます。



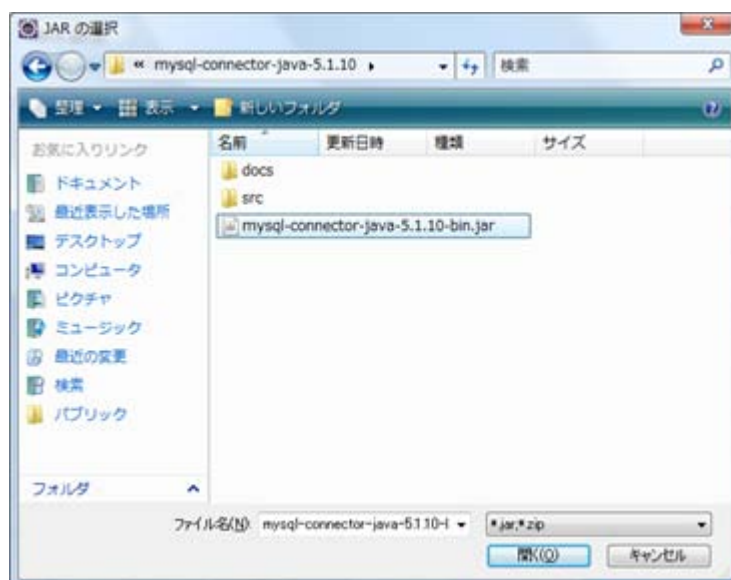
プロジェクト名を db_test として次に進みます。プロジェクト名は好きな名称にしてもかまいません。



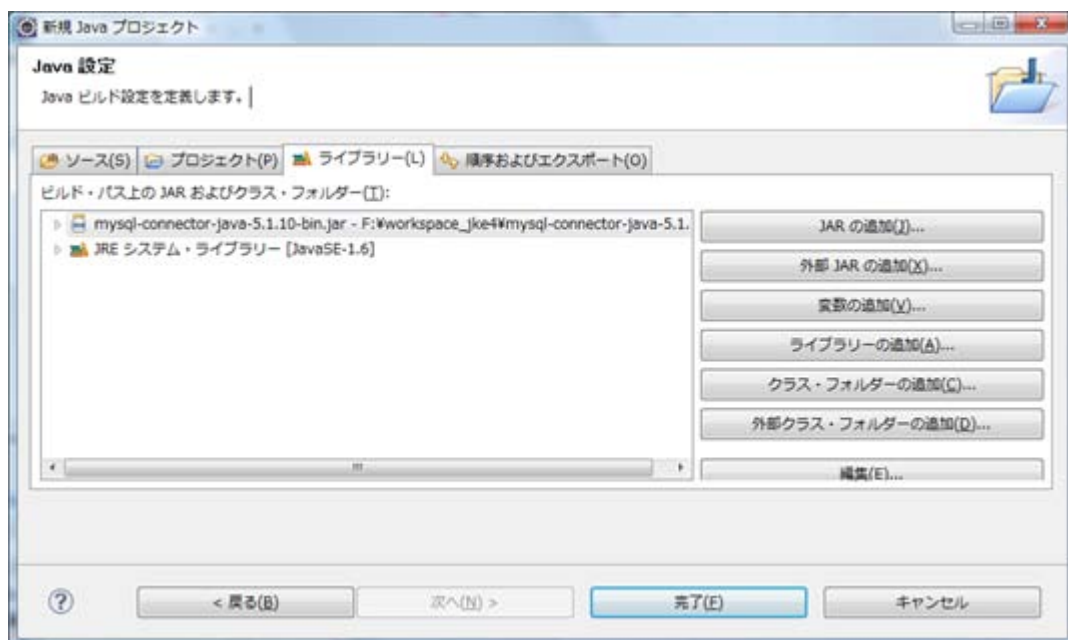
次に、mysql-connector-javaを登録します。「ライブラリー」タグをクリックし、「外部JARの追加」を選択して、先にダウンロードしておいたmysql-connecotr-java-[バージョン]-bin.jar といったファイルを選択します。



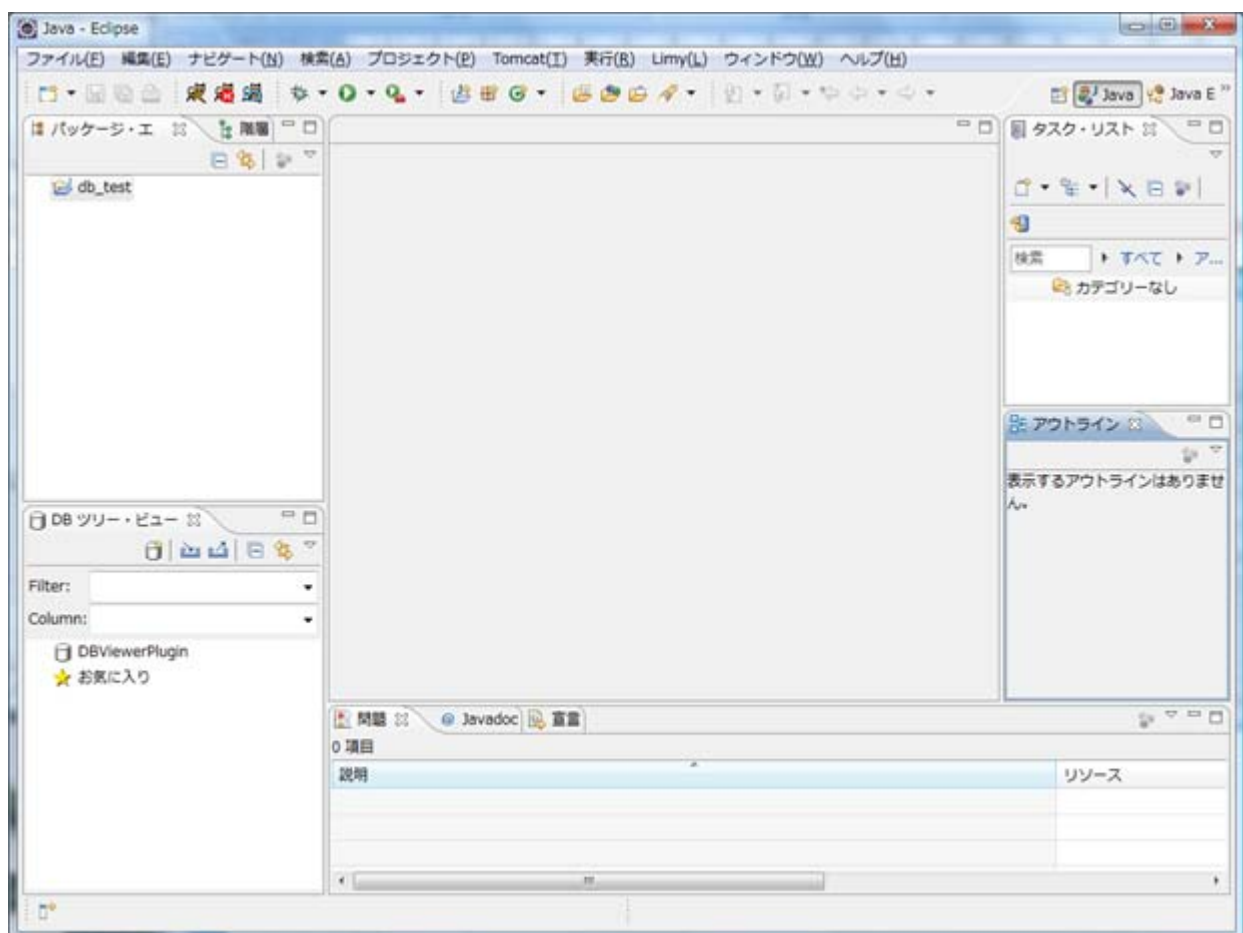
ファイルの選択画面の例は以下の通りです。



以上の操作で、ライブラリーにmysql-connector-javaが追加されますので、「完了」をクリックしてプロジェクトを作成します。



プロジェクト作成直後のEclipseのウィンドウは以下ようになります。



MySQLを操作するJavaプログラムを作成する際には、以上のような方法でプロジェクトを作成することになります。

プロジェクト作成時に外部jarファイルの登録を忘れた場合は、以下の方法で登録することもできます。

プロジェクト名で右クリック → 「プロパティ」 → 「Javaのビルドパス」で、上で説明したJava設定の画面と同様な画面が表示されますので、「ライブラリー」タグを選択して、「外部JARの追加」をクリックします。

MySQLを操作するためのユーザの設定

JavaプログラムからMySQLを操作するために、対象となるデータベースへのアクセス権限を持ったMySQLのユーザを登録します。前回のSQLモニタを使った実習ではrootユーザを使用しました。Javaプログラムからの操作の際にもrootユーザを使っても良いのですが、セキュリティへの配慮やプログラミングの際の誤操作の範囲を最小限にするためにも、対象となるデータベースに対して特定の場所からだけアクセスできるユーザアカウントを作成して使用するようにすべきです。

ユーザの登録とデータベースへのアクセス権の付与はGRANT文で一度にできます。GRANT文の構文は以下の通りです。

```
GRANT 実行できるコマンドをカンマ区切りで記述 ON データベース TO ユーザ IDENTIFIED BY 'パスワード';
```

データベースの指定は、データベース名.*と記述します。たとえば、practice.* といった形です。これはpracticeというデータベースの全てのテーブルが対象となることを示しています。ユーザの指定は、ユーザ名@ホスト名と記述して、アクセス可能なホスト(コンピュータ)も指定します。たとえば、puser@localhost と記述すれば、MySQLが動作しているコンピュータ(localhost)からのみpuserというユーザ名でアクセスできるようになります。puser@host.teikyo-u.ac.jpと記述すれば、host.teikyo-u.ac.jpからのみアクセスできます。puser@'%'と記述すると全てのホストからアクセスできます。この実習ではMySQLが動作しているコンピュータでJavaプログラムを実行するので、localhost と指定します。

GRANT文でアクセス権を設定したら、その設定を反映させるために以下のコマンドを実行します。

```
FLUSH PRIVILEGES;
```

以下の画面は、show databases で存在するデータベースを確認した後、practiceというデータベースの全てのテーブルに対して、SELECT, INSERT, UPDATE文をlocalhostのpuserに許可しています。Puserのパスワードは1234になります。GRANT文の次にFLUSH文を実行している点にも注意しておきましょう。

```
コマンドプロンプト - mysql -u root -p
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.1.38-community MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| practice |
| practice2 |
+-----+
4 rows in set (0.00 sec)

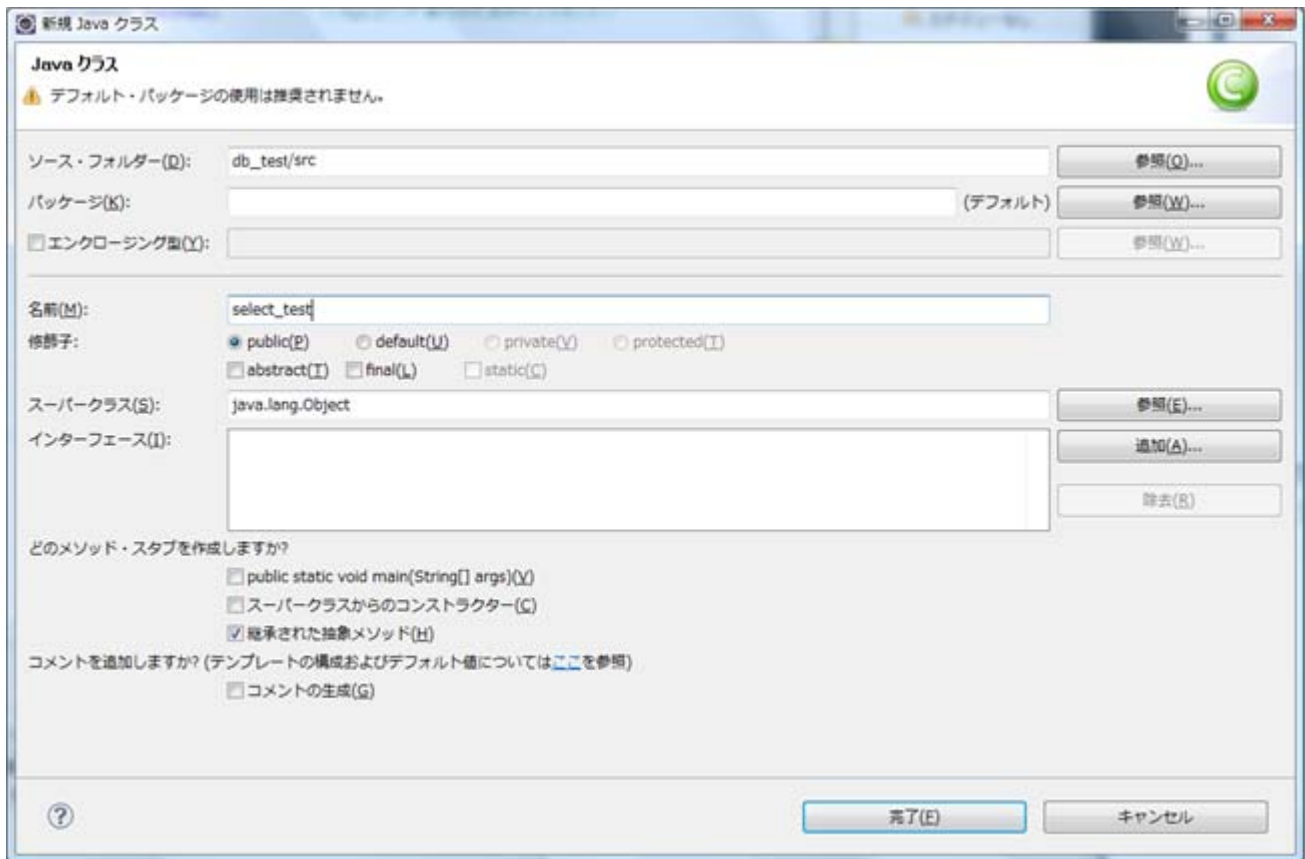
mysql> GRANT SELECT,INSERT,UPDATE ON practice.* TO puser@localhost
-> IDENTIFIED BY '1234';
Query OK, 0 rows affected (0.00 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.02 sec)

mysql>
```


Javaプログラミング データベースへの接続と切断

具体例として、JavaのSelectコマンドを実行してみるselect_testというクラスを作成しながら説明をしていきます。プロジェクトに select_testという新規クラスを作成します。パッケージ・エクスプローラでsrcというパッケージの上にマウスカーソルを持って行って、右クリック、「新規」→「クラス」で以下のようなウィンドウが表示されますので、その中の名前にselect_testと入力して「完了」をクリックします。



新しくできたselect_test.javaというファイルのプログラムの上の方には以下のように入力します。

```
import java.sql.*;
public class select_test {
    static String userid = "puser"; //ユーザID
    static String password = "1234"; //パスワード
    static Connection sqlCon; //MySQLへの接続
    static Statement sqlStmt; //SQLコマンド実行のためのインスタンス
}
```

最初に、MySQLの操作をするクラスを使うためにjava.sqlをインポートしておきます。それがimport java.sql.*;の部分です。

次に、MySQLの操作のための変数を定義します。MySQLを操作するクラス内に定義しておきます。ユーザIDとパスワードは、先に定義したpuserと1234を定義しています。ConnectionクラスはMySQLに接続するためのクラス、StatementクラスはSQLのコマンドを実行し、結果を取得するために使用するクラスです。

MySQLの操作は、(1)準備としてMySQLへの接続とStatementクラスの生成を行い、(2)Statementク
SQL (3)MySQL

スを使って コマンドを実行し、 への接続を切断する といった 段階で行います。この章では(1)準備の部分と(3)切断の部分を説明します。

MySQLへの接続とStatementの準備

MySQLへの接続とStatementクラスの生成を行うメソッドを以下に示します。上に記述した変数が同じクラス内に宣言されていることを前提としています。

```
> // MySQLを操作する準備を行うメソッド<
> private static void connectDB() {<
>     try{<
>         // ドライバクラスをロード<
>         Class.forName("org.gjt.mm.mysql.Driver"); // MySQLの指定<
<
>         // データベースへのURLを作成<
>         String url = "jdbc:mysql://localhost?useUnicode=true&characterEncoding=SJIS";<
>         // ユーザIDとパスワードを設定して接続<
>         sqlCon = DriverManager.getConnection(url,userid, password);<
<
>         // ステートメントオブジェクトを生成<
>         sqlStmt = sqlCon.createStatement();<
>     } catch (Exception e) {<
>         e.printStackTrace();<
>     }<
> }<
..
```

例外が生じた場合に備えて、全体を try で囲んでおきます。その中で、ドライバクラスをロードし、urlにMySQLへのURL(localhost)を作成します。そのURLにユーザIDとパスワードを指定して接続するインスタンスを生成し、sqlConに入れておきます。最後にsqlConに対してcreateStatement()を実行してStatementクラスのインスタンスを生成し、sqlStmtに入れておきます。

MySQLからの切断

MySQLに対する操作が終了したら、接続を切断します。以下は切断をするためのメソッドです。例外に備えてtryで囲んでありますが、実質はsqlStmtとsqlConに対して、close()を実行しているだけです。

```
> // MySQLへの接続を解除するメソッド<
> private static void closeDB() {<
>     try{<
>         sqlStmt.close();<
>         sqlCon.close();<
>     } catch (Exception e) {<
>         e.printStackTrace();<
>     }<
> }<
..
```

SELECT文で検索したレコードの表示

以下のプログラムリストはreservationテーブルのすべてのレコードの予約日、学生ID、施設名を表示するmainメソッドです。

```
public static void main ( String argv[]){  
    >    //(1) MySQLを使用する準備  
    >    connectDB();  
  
    >    //(2) MySQLの操作(SELECT文の実行)  
    >    try {  
    >        // 予約情報を取得するクエリ  
    >        String sql = "SELECT * FROM practice.reservation;";  
    >        // クエリーを実行して結果セットを取得  
    >        ResultSet rs = sqlSmt.executeQuery(sql);  
    >        // 検索結果からレコードを1つずつ取り出し、3つのカラムの値を表示  
    >        while(rs.next()){  
    >            >            String student = rs.getString("student_id");  
    >            >            String facility_name = rs.getString("facility_name");  
    >            >            String date = rs.getString("date");  
    >            >            String res = date + " " + student + " " + facility_name ;  
    >            >            System.out.println(res);  
    >        }  
    >    } catch (Exception e) {  
    >        >        e.printStackTrace();  
    >    }  
  
    >    //(3) MySQLへの接続切断  
    >    closeDB();  
    > }
```

全体としては、(1)MySQLを使用する準備、(2)MySQLの操作、(3)MySQLの切断といった流れになっています。これらのうち、(1)と(3)は先に示したメソッドを呼び出しているだけです。

(2)のMySQLの操作の部分詳しく見てみましょう。例外に備えて全体をtryで囲んでいます。まずString型のsqlにSQLのコマンドを入れておきます。reservationテーブルのレコードをすべて表示するコマンドです。テーブル名を指定する際に、データベース名.テーブル名となっていることに注意しましょう。先にSQLモニタを使用したときは、「use データベース名」を実行したので、テーブル名だけ指定すればよかったのですが、ここではデータベース名も指定する必要があります。

変数sqlに設定したSQLコマンドを、StatementクラスのsqlSmtに対してexecuteQueryで実行し、その結果をResultSetクラスのrsに入れます。ここにSQLモニタでSELECT文を実行した結果が入るイメージです。

rsに得た結果をwhile文を使ってレコードを1つずつ取り出して、表示を行う処理をしています。rs.next()で次のレコードが設定されます。次にレコードがあればtrueを返しますし、なければfalseを返します。それでwhile文の条件部にrs.next()を書いておくことで、次のレコードがなければ繰り返しを抜けることになります。

while文の中の処理では、レコードが設定されているrsに対して、カラム名を引数としてgetStringを実行することでそのカラム名に対する値を得ます。これをstudent_id, facility_name, dateに対して実行し、それらをresに一つの文字列としてまとめ、System.out.printlnで表示しています。

6.1 プログラムリストの全体

以下に、select_testクラスのソースリストの全体を示しておきます。

```

import java.sql.*;

public class select_test {

    static String userid = "puser"; //ユーザID
    static String password = "1234"; //パスワード
    static Connection sqlCon; //MySQLへの接続
    static Statement sqlStmt; //SQLコマンド実行のためのイン
    スタンス

    public static void main ( String argv[]){

        //(1) MySQLを使用する準備
        connectDB();

        //(2) MySQLの操作(SELECT文の実行)
        try {
            // 予約情報を取得するクエリ
            String sql = "SELECT * FROM practice.reservation;";
            // クエリーを実行して結果セットを取得
            ResultSet rs = sqlStmt.executeQuery(sql);
            // 検索結果からレコードを1つずつ取り出し、3つのカラムの値を表示
            while(rs.next()){
                String student = rs.getString("student_id");
                String facility_name = rs.getString("facility_name");
                String date = rs.getString("date");
                String res = date + " " + student + " " +
                facility_name ;
                System.out.println(res);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

        //(3) MySQLへの接続切断
        closeDB();
    }

    //MySQLを操作する準備を行うメソッド
    private static void connectDB(){
        try{
            // ドライバクラスをロード
            Class.forName("org.gjt.mm.mysql.Driver"); // MySQLの指定

            // データベースへのURLを作成
            String url = "jdbc:mysql://localhost?
            useUnicode=true&characterEncoding=SJIS";
            // ユーザIDとパスワードを設定して接続
            sqlCon = DriverManager.getConnection(url,userid, password);

            // ステートメントオブジェクトを生成
            sqlStmt = sqlCon.createStatement();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    //MySQLへの接続を解除するメソッド
    private static void closeDB(){
        try{
            sqlStmt.close();
            sqlCon.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

INSERT文を使ったデータの挿入

以下のプログラムリストはキーボードから、学生ID、施設名、予約日(年月日)を入力して、それらを予約情報としてreservationテーブルに挿入するmainメソッドです。

```
public static void main ( String argv[]){  
  
    > String student_id = "";  
    > String facility_name = "";  
    > String date = "";  
  
    > // (0) 挿入するデータの列の値を入力する  
    try {  
        > BufferedReader r =  
        >     new BufferedReader(new InputStreamReader(System.in), 1);  
        // データ入力準備  
  
        > // 学生IDの入力  
        > System.out.print("学生ID? "); // プロンプトの出力  
        > student_id = r.readLine(); // 値の入力  
  
        > // 施設名の入力  
        > System.out.print("施設名? "); // プロンプトの出力  
        > facility_name = r.readLine(); // 値の入力  
  
        > // 予約日の入力  
        > System.out.print("日付 (YYYY-MM-DDの形式で)? "); // プロンプトの出力  
        > date = r.readLine(); // 値の入力  
  
    } catch (IOException e1) {  
        > e1.printStackTrace();  
    }  
  
    > // (1) MySQLを使用する準備  
    > connectDB();  
  
    > // (2) MySQLの操作 (INSERT文の実行)  
    try {  
        > // 予約情報を挿入するクエリ  
        > String sql = "INSERT INTO practice.reservation ( student_id, facility_name, date ) VALUES ('";  
        > sql += student_id + "',' + facility_name + "',' + date + "')";  
        > // クエリを実行して結果セットを取得  
  
        > int rs_int = sqlStmt.executeUpdate(sql);  
        > System.out.println(rs_int+ "行登録しました");  
    } catch (Exception e) {  
        > e.printStackTrace();  
    }  
  
    > // (3) MySQLへの接続切断  
    > closeDB();  
  
}
```

(0)の部分で、キーボードからデータを入力しています。キーボードからの入力については、プログラミング4で学習しましたね。まず、データ入力のために、BufferedReaderクラスのインスタンスを生成し、rに入れておきます。学生ID、施設名、予約日のそれぞれに対して、System.out.printで入力すべき項目をプロンプトとして表示し、r.readLineで文字列変数に読み込んでいます。

(1)と(3)はselectのときと同じで、MySQLを使用する準備とMySQLの切断を別に定義したメソッドconnectDBとcloseDBを呼び出すことで実行しています。それらのメソッドは前の章で示した同名のメソッドと同じものを使います。

(2)の部分でMySQLのテーブルにINSERTの操作を行っています。SELECT文を実行するときには、executeQueryメソッドを使っていましたが、INSERT文やUPDATE文のようにテーブルの内容を変更するSQLコマンドをJavaから実行する場合は、executeUpdateメソッドを使用することに注意しましょう。(2)の処理では最初に文字型の変数sqlにSQLコマンドを設定しています。このとき、キーボードから変数student_id, facility_name, dateに入力した値を使うために、演算子を使って文字列をつ

なぐことでコマンドの全体を作成します。次に、sqlStmtに対してexecuteUpdateメソッドでコマンドを実行し、結果を整数変数のrs_intに入れています。executeUpdateは処理をしたレコード数を返します。これはSQLモニタでINSERT文を実行したときに「Query OK, 1 row affected」等と表示されますが、ここで表示される数です。

プログラムリストの全体

insert_testというクラスを作成し、上で述べたmainメソッドを追加します。connectDBメソッド、closeDBメソッドはクラスselect_testと同じものを追加しておきます。以下に、insert_testクラスのソースリストの全体を示します。入力で使用するBufferedReaderクラスのために、java.io.*もインポートしておきます。

```
import java.io.*;
import java.sql.*;

public class insert_test {

    static String userid = "puser"; //ユーザID
    static String password = "1234"; //パスワード
    static Connection sqlCon; //MySQLへの接続
    static Statement sqlStmt; //SQLコマンド実行のためのイン
    スタンス

    public static void main ( String argv[]){

        String student_id = "";
        String facility_name = "";
        String date = "";

        //(0) 挿入するデータのカラムの値を入力する
        try {
            BufferedReader r =
                new BufferedReader(new InputStreamReader(System.in), 1);
            // データ入力の準備

            // 学生IDの入力
            System.out.print("学生ID ? "); // プロンプトの出力
            student_id = r.readLine(); // 値の入力

            // 施設名の入力
            System.out.print("施設名 ? "); // プロンプトの出力
            facility_name = r.readLine(); // 値の入力

            // 予約日の入力
            System.out.print("日付 (YYYY-MM-DDの形式で) ? "); // プロンプトの出力
            date = r.readLine(); // 値の入力

        } catch (IOException e1) {
            e1.printStackTrace();
        }

        //(1) MySQLを使用する準備
        connectDB();

        //(2) MySQLの操作(INSERT文の実行)
        try {
            // 予約情報を挿入するクエリ
            String sql = "INSERT INTO practice.reservation (
student_id, facility_name, date ) VALUES ('";
            sql += student_id + "','" + facility_name + "','" + date +
            "')";

            // クエリーを実行して結果セットを取得

            int rs_int = sqlStmt.executeUpdate(sql);
            System.out.println(rs_int+ "行登録しました");
        } catch (Exception e) {
            e.printStackTrace();
        }

        //(3) MySQLへの接続切断
        closeDB();
    }
}
```

```
}

///// MySQLを操作する準備を行うメソッド
private static void connectDB(){
    try{
        // ドライバクラスをロード
        Class.forName("org.gjt.mm.mysql.Driver"); // MySQLの指定

        // データベースへのURLを作成
        String url = "jdbc:mysql://localhost?
useUnicode=true&characterEncoding=SJIS";
        // ユーザIDとパスワードを設定して接続
        sqlCon = DriverManager.getConnection(url,userid, password);

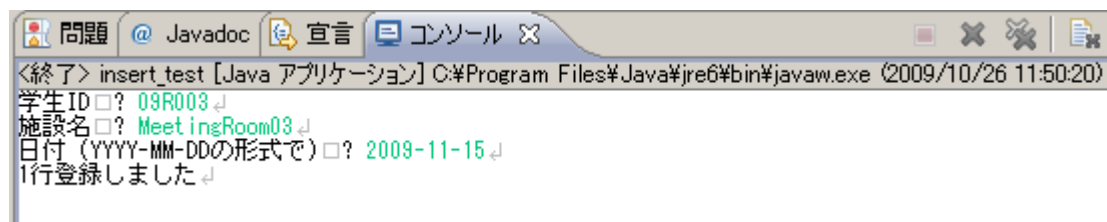
        // ステートメントオブジェクトを生成
        sqlStmt = sqlCon.createStatement();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

///// MySQLへの接続を解除するメソッド
private static void closeDB(){
    try{
        sqlStmt.close();
        sqlCon.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

}
```

実行例

以下に insert_test クラスを実行した例を示します。緑色の文字がキーボードから入力した文字です。



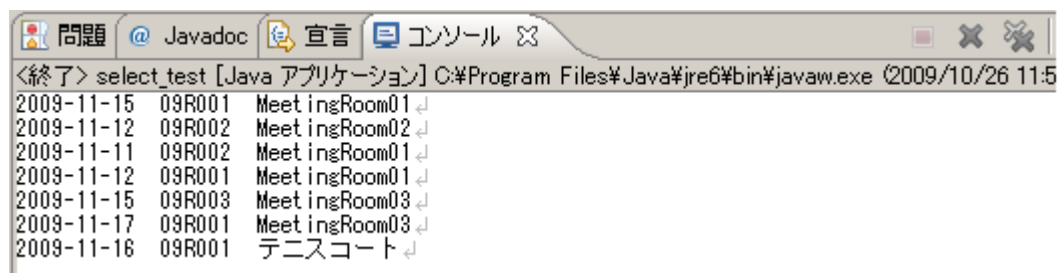
```
<終了> insert_test [Java アプリケーション] C:\Program Files\Java\jre6\bin\javaw.exe (2009/10/26 11:50:20)
学生ID □? 09R003 ↓
施設名 □? MeetingRoom03 ↓
日付 (YYYY-MM-DDの形式で) □? 2009-11-15 ↓
1行登録しました ↓
```

以下のように日本語を入れることもできます。Eclipseで実行する場合は、入力用カーソルが行の先頭に行ってしまうことがあるので、表示されているプロンプトの後をマウスでクリックしてから入力するとよいでしょう。



```
<終了> insert_test [Java アプリケーション] C:\Program Files\Java\jre6\bin\javaw.exe (2009/10/26 11:56:35)
学生ID □? 09R001 ↓
施設名 □? テニスコート ↓
日付 (YYYY-MM-DDの形式で) □? 2009-11-16 ↓
1行登録しました ↓
```

以上のような形でいくつかのレコードを追加した後、select_test クラスを実行した例を以下に示します。日本語で入力した施設名「テニスコート」もきちんと表示されていることがわかります。



```
<終了> select_test [Java アプリケーション] C:\Program Files\Java\jre6\bin\javaw.exe (2009/10/26 11:5
2009-11-15 09R001 MeetingRoom01 ↓
2009-11-12 09R002 MeetingRoom02 ↓
2009-11-11 09R002 MeetingRoom01 ↓
2009-11-12 09R001 MeetingRoom01 ↓
2009-11-15 09R003 MeetingRoom03 ↓
2009-11-17 09R001 MeetingRoom03 ↓
2009-11-16 09R001 テニスコート ↓
```

この状態で、SQL モニタを使ってSELECT 文でレコードを表示させると、以下のように日本語は文字化けしてしまいます。これはSQL モニタの文字コードの設定が原因なので特に気にする必要はありません。上で示したように日本語で入力した値は、きちんと日本語で表示されます。



```
mysql> select * from reservation;
+-----+-----+-----+-----+
| reservation_id | student_id | facility_name | date |
+-----+-----+-----+-----+
| 1 | 09R001 | MeetingRoom01 | 2009-11-15 |
| 2 | 09R002 | MeetingRoom02 | 2009-11-12 |
| 3 | 09R002 | MeetingRoom01 | 2009-11-11 |
| 4 | 09R001 | MeetingRoom01 | 2009-11-12 |
| 5 | 09R003 | MeetingRoom03 | 2009-11-15 |
| 6 | 09R001 | MeetingRoom03 | 2009-11-17 |
| 7 | 09R001 | ?????? | 2009-11-16 |
+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```


まとめ

このモジュールでは以下のことを学びました。

- EclipseでMySQLを操作するJavaプログラムを作成する際のmysql-connectorの設定方法
 - MySQLを操作するJavaのプログラムの大まかな処理の流れ
 - JavaプログラムでMySQLを操作する準備の処理と切断の処理
 - MySQLのテーブルからデータを検索するJavaプログラムを記述方法
 - MySQLのテーブルにデータを挿入するJavaプログラムを記述方法
-