

Java プログラミング初級

2019 年 1 月 21 日～1 月 25 日

(実習テキスト)



Web Applications
Windows Store Apps Using C#

2019 年 1 月 18 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

JAVA プログラミング初級.....	1
目 次	3
第 1 章. はじめに	5
1.1. 概要	6
第 2 章. 基本的なプログラム（条件判断と繰り返し）	7
2.1. 簡単なログオンプログラム（条件判断）	8
2.2. 繰り返し処理.....	15
第 3 章. 配列とメソッド（関数）	21
3.1. 複数のアカウントを用意する（配列）	22
3.2. 認証部分を関数化（メソッド化）する	26
第 4 章. オブジェクト指向	29
4.1. ユーザーアカウントオブジェクトの導入.....	30
4.2. コンストラクターの定義	36
4.3. 新たなメソッドの定義.....	42
第 5 章. オブジェクトを機能ごとに分ける.....	47
5.1. アカウント管理オブジェクトの導入.....	48
5.2. アカウントを配列からコレクションに変更する	52
第 6 章. アカウント追加・一覧表示機能の追加.....	57
6.1. 機能追加の準備	58
6.2. クラスの分離.....	63
6.3. 追加機能と一覧表示機能の実装	68
第 7 章. エラー（例外）処理の追加	75
7.1. パスワードに長さの制限を設定する	76
第 8 章. アカウントをファイルに保存する（参考）	83
8.1. ユーザーアカウントをファイルに保存する（参考）	84
8.2. ファイルからユーザーアカウントを読み込む	92

(空白ページ)

第1章. はじめに

1.1. 概要

1.1.1.説明

このプリントは「**Java** プログラミング初級」の内容を網羅した実践的な実習テキストである。

「ユーザー名とパスワードを入力してログオンする」という認証機能を、非常に簡単なプログラムから作り始め、最終的には実務で耐えうるコードにまで仕上げることを目標にしている。

作成するアプリケーションは、**GUI** や **Web** 形式ではなく、コンソールアプリケーションである。これは、できるだけ **Java** の純粋な文法やコーディング方法に焦点を当てたいからである。

第2章. 基本的なプログラム（条件判断と繰り返し）

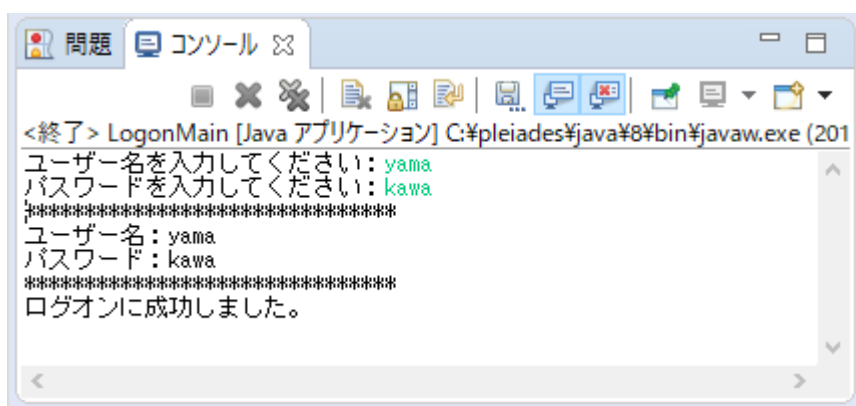
2.1. 簡単なログオンプログラム（条件判断）

2.1.1.概要

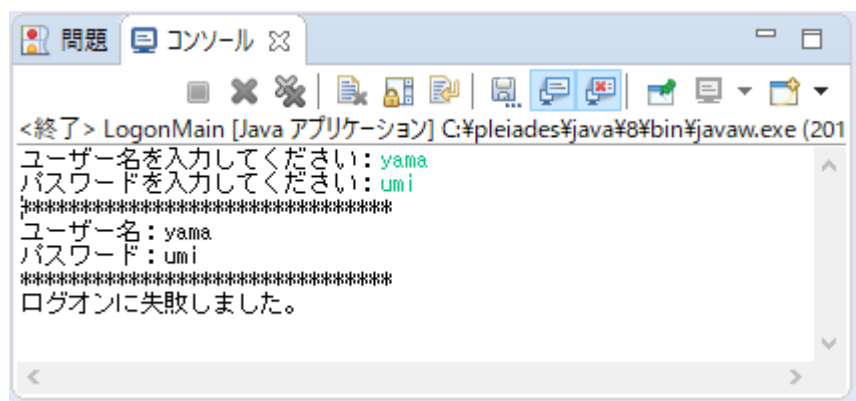
キーボードからユーザー名とパスワードを入力させ、登録されたユーザー名とパスワードと同一かどうか判断するプログラムを作成する。

登録されたものと同じ場合は「ログオン成功」、異なる場合は「ログオン失敗」とし、適切なメッセージを表示する。

このアプリケーションは、今後、少しずつバージョンアップし、最終的に、オブジェクト指向に基づいて、複数のユーザーアカウント（ユーザー名とパスワードの組み合わせ）を管理・認証できるアプリケーションへとバージョンアップしていく。



```
<終了> LogonMain [Java アプリケーション] C:\pleiades\java\8\bin\javaw.exe (201
ユーザー名を入力してください: yama
パスワードを入力してください: kawa
*****
ユーザー名: yama
パスワード: kawa
*****
ログオンに成功しました。
```



```
<終了> LogonMain [Java アプリケーション] C:\pleiades\java\8\bin\javaw.exe (201
ユーザー名を入力してください: yama
パスワードを入力してください: umi
*****
ユーザー名: yama
パスワード: umi
*****
ログオンに失敗しました。
```


2.1.2.プロジェクトの作成

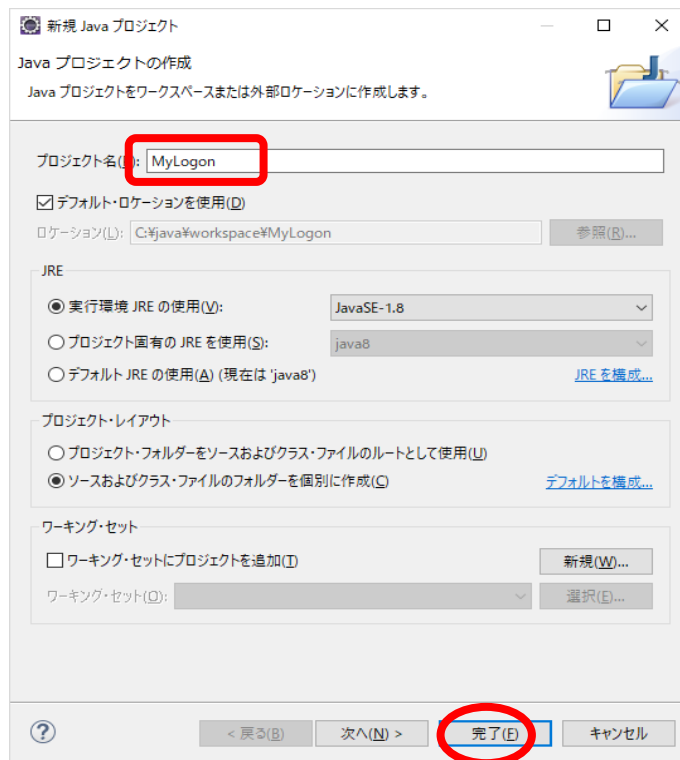
MyLogon という名前のプロジェクトを作成する。

Eclipse のパースペクティブを [Java] に設定する。

Eclipse のメニューから [ファイル] – [新規] – [Java プロジェクト] とたどっていく。

(パースペクティブが [JavaEE] の場合は、[ファイル] – [新規] – [プロジェクト] または [ファイル] – [新規] – [その他] とたどっていてもよい。その場合は、さらに、[Java プロジェクト] を選択する)

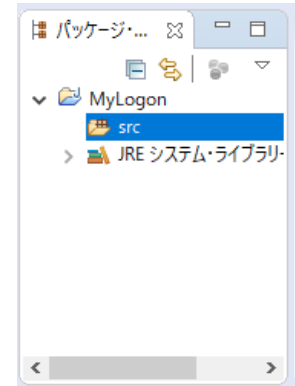
下図のように、プロジェクト名に [MyLogon] と入力し、[完了] ボタンをクリックする。



2.1.3.クラスの追加

これまでの作業の結果、パッケージエクスプローラー（[JavaEE] パースペクティブの場合は、プロジェクトエクスプローラー）に、MyLogon というプロジェクトが作成される。

右図のように MyLogon プロジェクトを広げ、src フォルダを右クリックする。



[新規] - [クラス] とたどっていく。

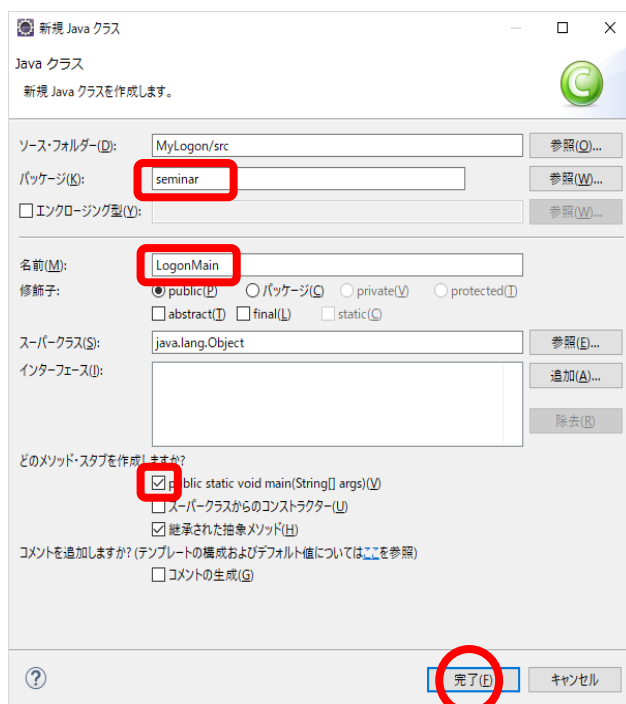
下記のウィンドウが表示されるので、3箇所入力する。

[パッケージ] seminar

[名前] LogonMain

[どのメソッドスタブを作成しますか?]

public static void main(String[] args)
にチェックを入れる。



2.1.4.ソースコードの確認

下記のようなソースコードが自動生成される。

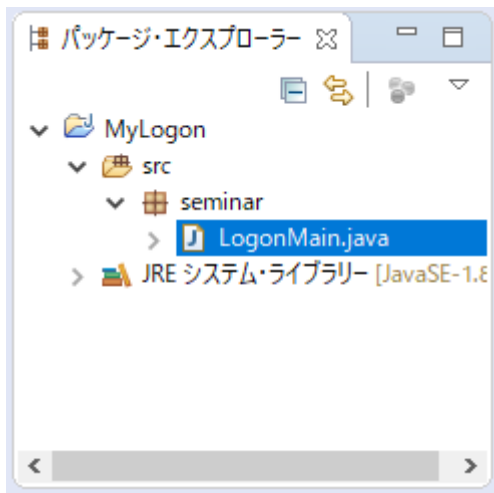
```
package seminar;

public class LogonMain {

    public static void main(String[] args) {
        // TODO 自動生成されたメソッド・スタブ
    }

}
```

パッケージエクスプローラーが下記のようにになる。



2.1.5. ソースコードの修正

自動生成されたコードを下記のように書き換える。(下線部の `throws IOException` を忘れないように)

```
package seminar;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class LogonMain {

    // throws IOException を忘れないように！
    public static void main(String[] args) throws IOException {
        // TODO 自動生成されたメソッド・スタブ
        //
        final String USERNAME = "yama";
        final String PASSWORD = "kawa";
        //
        InputStreamReader isr = new InputStreamReader(System.in); // 1文字だけの入力
        BufferedReader br = new BufferedReader(isr); // 複数文字（文字列）の入力
        //
        System.out.print("ユーザー名を入力してください：");
        String user = br.readLine();
        //
        System.out.print("パスワードを入力してください：");
        String pass = br.readLine();
        //
        //入力された情報の確認表示
        //
        System.out.println("*****");
        System.out.println("ユーザー名：" + user);
        System.out.println("パスワード：" + pass);
        System.out.println("*****");
        //
        if (user.equals(USERNAME) && pass.equals(PASSWORD)) {
            System.out.println("ログオンに成功しました。");
        }
        else {
            System.out.println("ログオンに失敗しました。");
        }
    }
}
```

2.1.6.ソースコードのコンパイル

Eclipse では、ソースコードを保存するときに、同時にコンパイルも行われる。

[ファイル] - [保管] あるいは [ファイル] - [すべて保管] で、保存とコンパイルが行われる。

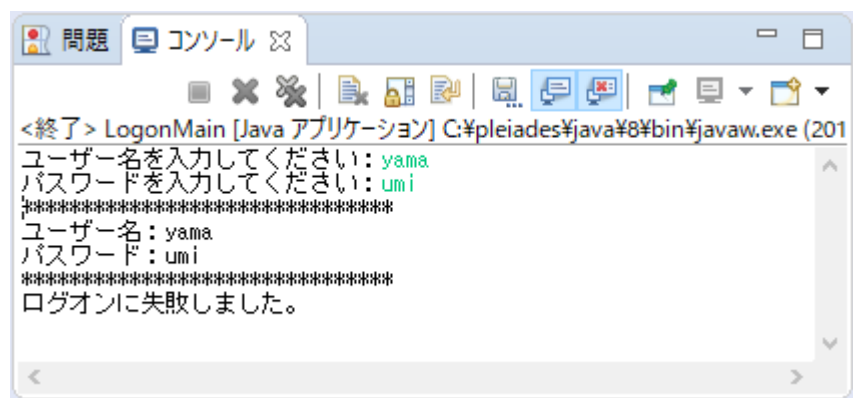
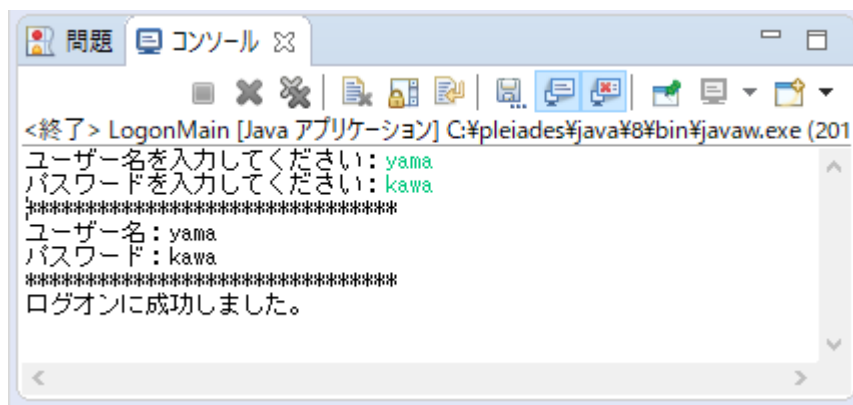
2.1.7.アプリケーションの実行

`public static void main()` メソッドが存在するクラスは、**Java** アプリケーションとして実行することができる。

パッケージエクスプローラーで、**LogonMain** クラスを右クリックし、[実行] - [Java アプリケーション] とたどっていく。

コンソールウィンドウに「ユーザー名を入力してください:」と表示されるので、例えば下記のように入力して、動作を確認する。

上の図は、ユーザー名に **yama**、パスワードに **kawa** と入力した例で、下の図は、ユーザー名に **yama**、パスワードに **umi** と入力した例。



2.1.8.ソースコードの説明

最初のサンプルであるが、ひとつとおり動作するものを作成したため、かなり多くの文法を使用している。

詳細は研修時に説明するが、主な点を列挙すると下記のようなになる。

1. package 文
2. import 文
3. throws キーワード
4. final キーワード
5. InputStreamReader と BufferedReader
6. if 文
7. &&演算子
8. equals メソッド

2.2. 繰り返し処理

2.2.1.概要

現状では、1回の認証しか行えないので、認証に失敗した場合、繰り返し入力できるように変更する。

while 文と break 文を理解することが目標。

2.2.2.プログラムのバージョンアップの準備

1. パッケージの作成

新たなプロジェクトを作成してもよいが、プロジェクトの数が増えすぎて管理しづらくなるので、新たな別の「パッケージ」（フォルダーのようなもの）を作成し、以前のソースコードをコピーしてバージョンアップする。

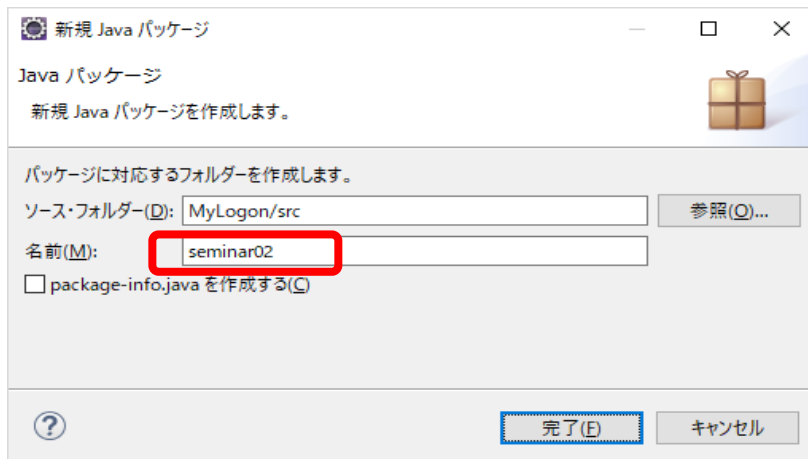
これは、研修をスムーズに行うための方法であり、実際の開発業務では、新たなプロジェクトを作成する場合が多い。

パッケージエクスプローラーで MyLogon の src フォルダーを右クリックする。

[新規] - [パッケージ] とたどっていく。

下記のウィンドウが表示されるので、1箇所入力する。

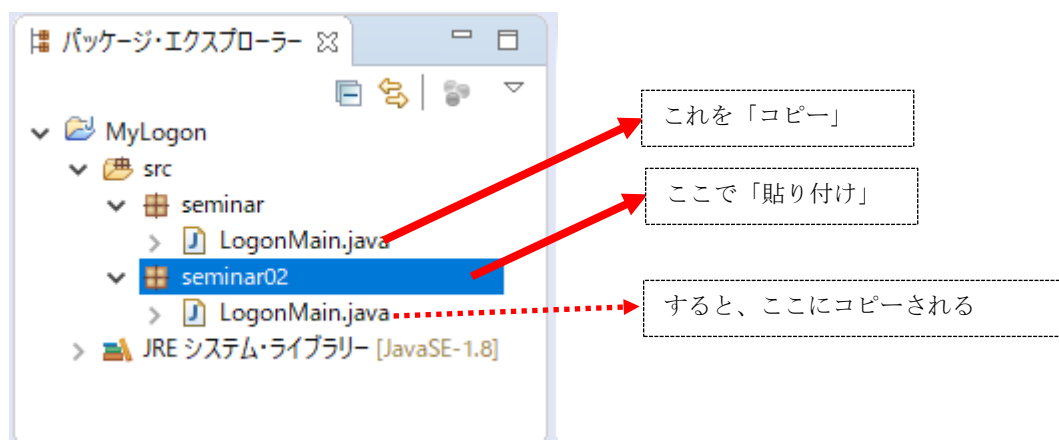
[名前] seminar02



今後、パッケージは、seminar03, seminar04… と増やしていく。

2. ソースファイルのコピー

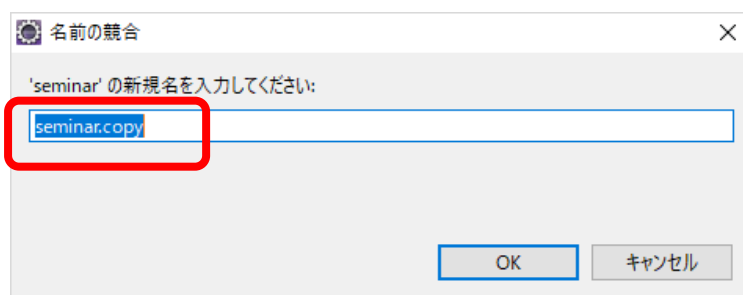
次に、seminar パッケージ内の LogonMain.java ファイルを、seminar02 パッケージにコピーする。



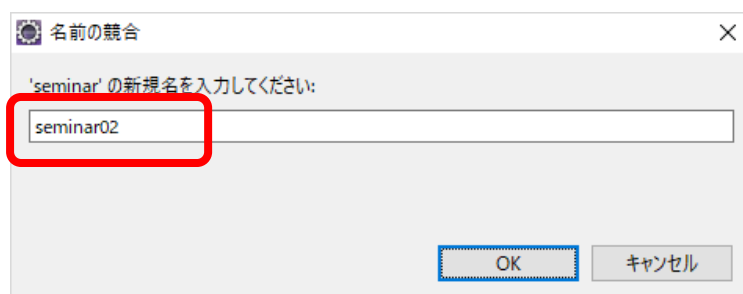
3. 別の方法

seminar パッケージをクリックし、キーボードから [CTRL]+C でコピーし、[CTRL]+V で貼り付けるという順で操作すると、パッケージをまるごと複製できる。

① 貼り付け直後の状態：



② 上記の seminar.copy の部分を、seminar02 に書き換える：



4. ソースコードの確認

コピーされたソースコードは、最初の行の `package` 文が、正しく `seminar02` となっていることを確認しておこう。

```
package seminar02;                                //この部分に注目!!!

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class LogonMain {

    public static void main(String[] args) throws IOException {
        // TODO 自動生成されたメソッド・スタブ
        //
        final String USERNAME = "yama";
        final String PASSWORD = "kawa";
        //
        InputStreamReader isr = new InputStreamReader(System.in); // 1文字だけの入力
        BufferedReader br = new BufferedReader(isr);             //複数文字（文字列）の入力
        //
        //
        // この部分にソースコードがありますが、紙面の都合で省略します。
        //
        //
    }
}
```

2.2.3.ソースコードの修正

```

package seminar02;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class LogonMain {
    public static void main(String[] args) throws IOException {
        final String USERNAME = "yama";
        final String PASSWORD = "kawa";
        final String END_MARK = "9999"; //プログラム終了の目印
        //
        InputStreamReader ips = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ips);
        //
        while (true) {
            System.out.println("終了するときは、ユーザー名に" + END_MARK +
                               "と入力してください。");

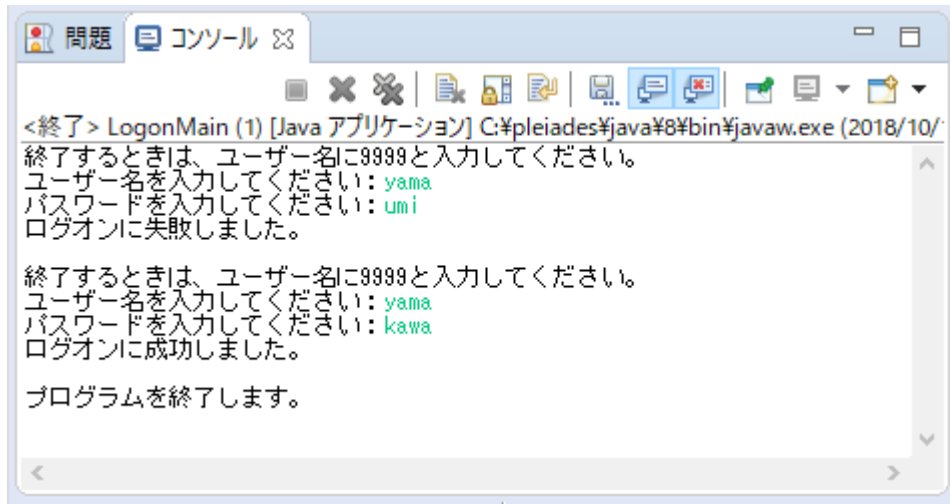
            System.out.print("ユーザー名を入力してください：");
            String user = br.readLine();
            //
            if (user.equals(END_MARK)) {
                break;
            }
            //
            System.out.print("パスワードを入力してください：");
            String pass = br.readLine();
            //
            //下記のコードは残しておいてもいいのですが、
            //表示が煩雑になるので、削除（コメント）することにしました。
            //
            System.out.println("*****");
            System.out.println("ユーザー名：" + user);
            System.out.println("パスワード：" + pass);
            System.out.println("*****");
            //
            if (user.equals(USERNAME) && pass.equals(PASSWORD)) {
                System.out.println("ログオンに成功しました。");
                break;
            }
            else {
                System.out.println("ログオンに失敗しました。");
            }
            System.out.println(""); //空白行を1行出力。
        }
        System.out.println("¥n プログラムを終了します。");
    }
}

```

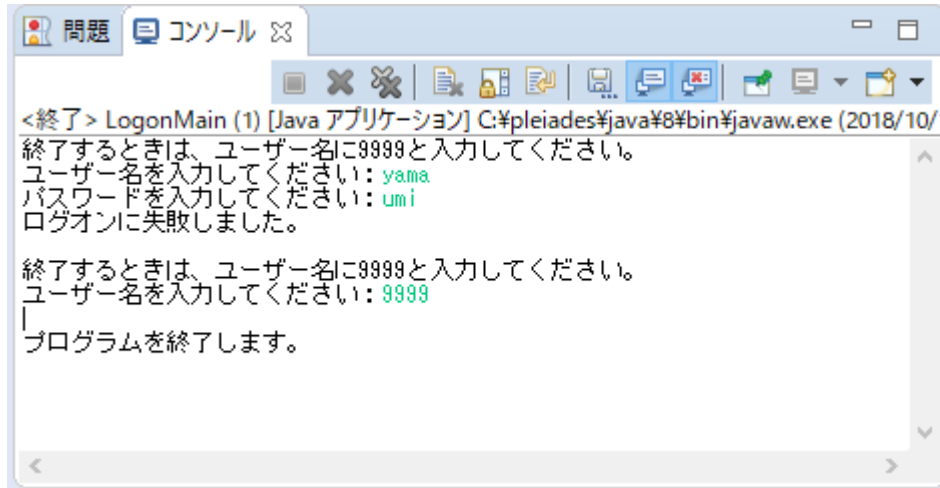
2.2.4.アプリケーションの実行

seminar02 配下の LogonMain を右クリックし、[実行] - [Java アプリケーション] とたどっていく。

(seminar の LogonMain を実行しないように注意しよう)



ユーザー名に 9999 と入力して、途中で終了する場合：



(空白ページ)

第3章. 配列とメソッド（関数）

3.1. 複数のアカウントを用意する（配列）

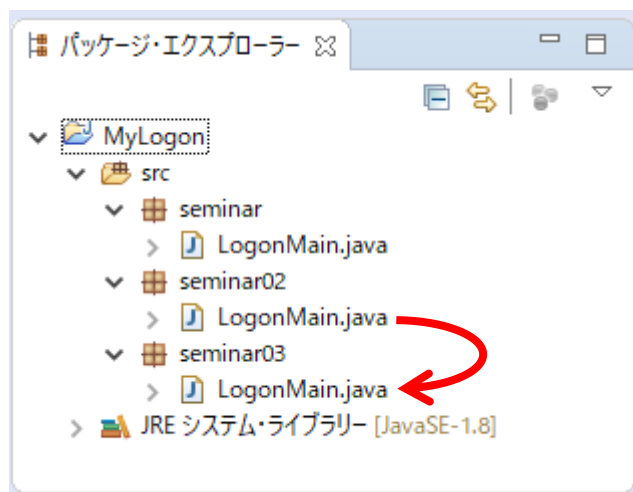
3.1.1.概要

現状では、yama/kawa という 1 つのユーザーアカウントしか存在しない。そこで、複数のアカウントを用意し、その中のどれかに一致すればログオンできるように変更する。

配列とその使い方を理解することが目標。

3.1.2.事前準備

semina03 パッケージを作成し、seminar02 の LogonMain をコピーする。



3.1.3. ソースコードの修正

```

package seminar03;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class LogonMain {
    public static void main(String[] args) throws IOException {
        //final String USERNAME = "yama";
        //final String PASSWORD = "kawa";
        //変数名の綴りが複数形になっています。注意してください。
        final String[] USERNAMES = {"yama", "java", "duke", "hello", "james"};
        final String[] PASSWORDS = {"kawa", "cafe", "mascot", "world", "gosling"};
        final String END_MARK = "9999";
        //
        InputStreamReader ips = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ips);
        //
        while (true) {
            //
            System.out.println("終了するときは、ユーザー名に" + END_MARK +
                               "と入力してください。");

            System.out.print("ユーザー名を入力してください：");
            String user = br.readLine();
            //
            if (user.equals(END_MARK)) {
                break;
            }
            //
            System.out.print("パスワードを入力してください：");
            String pass = br.readLine();
            //
            boolean found = false;
            for (int i = 0; i < USERNAMES.Length; i++) {
                if (user.equals(USERNAMES[i]) && pass.equals(PASSWORDS[i])) {
                    found = true;
                    break;
                }
            }
            //
            if (found == true) {
                System.out.println("ログオンに成功しました。");
                break;
            }
            else {
                System.out.println("ログオンに失敗しました。");
            }
            System.out.println("");
        }
        System.out.println("¥n プログラムを終了します。");
    }
}

```

3.1.4.アプリケーションの実行

seminar03 配下の LogonMain を右クリックし、[実行] – [Java アプリケーション] とたどっていく。

(seminar や seminar02 の LogonMain を実行しないように注意しよう)

終了するときは、ユーザー名に **9999** と入力してください。

ユーザー名を入力してください : duke

パスワードを入力してください : muscat

ログオンに失敗しました。

終了するときは、ユーザー名に **9999** と入力してください。

ユーザー名を入力してください : duke

パスワードを入力してください : mascot

ログオンに成功しました。

プログラムを終了します。

(空白ページ)

3.2. 認証部分を関数化（メソッド化）する

3.2.1. 概要

認証する部分を独立させ、isCertified というメソッドとする。

関数（メソッド）を理解することが目標。

3.2.2. 事前準備

semina04 パッケージを作成し、seminar03 の LogonMain をコピーする。

3.2.3. ソースコードの修正

public static void main() よりも上の方に、isCertified というメソッドを追加する。

isCertified の内部には、配列の定義と繰り返し（for 文）の部分を、main() からコピーして貼り付け、下記のように修正する。

```
package seminar04;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class LogonMain {

    ##### 004 main() 関数からコピーし、一部修正。
    private static boolean isCertified(String user, String pass) {
        final String[] USERNAMES = {"yama", "java", "duke", "hello", "james"};
        final String[] PASSWORDS = {"kawa", "cafe", "mascot", "world", "gosling"};
        //
        for (int i = 0; i < USERNAMES.length; i++) {
            if (user.equals(USERNAMES[i]) && pass.equals(PASSWORDS[i])) {
                return true;
            }
        }
        //
        return false;
    }

    public static void main(String[] args) throws IOException {
        ##### 004 上記 isCertified へコピーし、この部分は削除（またはコメント）。
        //変数名の綴りが複数形になっています。注意してください。
        //final String[] USERNAMES = {"yama", "java", "duke", "hello", "james"};
        //final String[] PASSWORDS = {"kawa", "cafe", "mascot", "world", "gosling"};
        //次の行は残す。
        final String END MARK = "9999";
        //
    }
}
```

(次のページへ)

```

InputStreamReader ips = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(ips);
//
while (true) {
    //
    System.out.println("終了するときは、ユーザー名に" + END_MARK +
                        "と入力してください。");

    System.out.print("ユーザー名を入力してください：");
    String user = br.readLine();
    //
    if (user.equals(END_MARK)) {
        break;
    }
    //
    System.out.print("パスワードを入力してください：");
    String pass = br.readLine();
    //

    ##### 004 大部分を削除 (コメント化) し、一部修正。
    //
    boolean found = false;
    boolean found = isCertified(user, pass);
    //
    for (int i = 0; i < USERNAMES.length; i++) {
    //
        if (user.equals(USERNAMES[i]) && pass.equals(PASSWORDS[i])) {
    //
            found = true;
    //
            break;
    //
        }
    //
    }
    //
    if (found == true) {
        System.out.println("ログオンに成功しました。");
        break;
    }
    else {
        System.out.println("ログオンに失敗しました。");
    }
    System.out.println("");
}
System.out.println("¥n プログラムを終了します。");
}
}

```

3.2.4.アプリケーションの実行

今回の修正は、外部から見た動作は一切変わらないので、前回の実行結果と同じになる。

外部から見た動作を一切変更せず、内部のソースコードだけを改善することを「リファクタリング *refactoring*」と呼ぶ。

(空白ページ)

第4章. オブジェクト指向

4.1. ユーザーアカウントオブジェクトの導入

4.1.1.概要

ユーザー名とパスワードの組み合わせを1つのオブジェクトとして表現してみる。

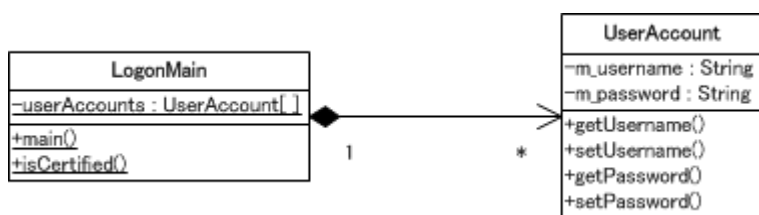
簡単なクラスを作成し、利用できるようにすることが目標。

なお、この段階では、いままでのコードよりも量が増え、煩雑なコードに見えてしまうが、今後改良する。従って、この節のコードは、学習過程での一時的なものである。

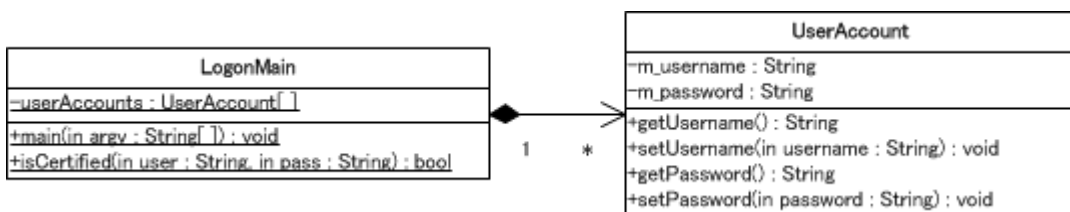
4.1.2.説明

今までのコードは、ユーザー名とパスワードが別の配列に格納されていたので、ユーザー名とパスワードの組み合わせがずれてしまったり、数の過不足があるような場合のチェックが行いにくくなっていた。

そこで、**UserAccount** というオブジェクトを導入し、**LogonMain** クラスから呼び出して使用するようソースコードを変更する。(これは、機能追加や機能変更ではなく、リファクタリングである)



メソッドのパラメーター（引数）も含めた詳細なクラス図を掲載すると下記のようなになる。



4.1.3.UserAccount クラスの追加

これまでと同じように新しいパッケージに **LogonMain** クラスをコピーする。
つまり、**seminar05** に **Logonmain** ができあがる。

次に、パッケージ **seminar05** を右クリックし、[新規] - [クラス] とたどっていき、**UserAccount** クラスを作成する。

4.1.4.ソースコード

1. UserAccount クラス

```
package seminar05;

public class UserAccount {
    //=====
    //   フィールド変数 (メンバー変数)
    //=====
    private String m_username;
    private String m_password;

    //=====
    //   メソッド
    //=====
    //
    //   Username (get/set)
    //
    public String getUsername() {
        return this.m_username;
    }
    public void setUsername(String username) {
        this.m_username = username;
    }
    //
    //   Password (get/set)
    //
    public String getPassword() {
        return this.m_password;
    }
    public void setPassword(String password) {
        this.m_password = password;
    }
}
```

ここでいったんコンパイル（保存）し、エラーがないことを確認して、次へ進む。

2. LogonMain クラス

```
package seminar05;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class LogonMain {

    //これは、LogonMain クラスのメンバー変数となる。
    private static UserAccount[] userAccounts = new UserAccount[5];

    private static boolean isCertified(String user, String pass) {
        //    final String[] USERNAMES = {"yama", "java", "duke", "hello", "james"};
        //    final String[] PASSWORDS = {"kawa", "cafe", "mascot", "world", "gosling"};

        userAccounts[0] = new UserAccount();
        userAccounts[0].setUsername("yama");
        userAccounts[0].setPassword("kawa");
        //
        userAccounts[1] = new UserAccount();
        userAccounts[1].setUsername("java");
        userAccounts[1].setPassword("cafe");
        //
        userAccounts[2] = new UserAccount();
        userAccounts[2].setUsername("duke");
        userAccounts[2].setPassword("mascot");
        //
        userAccounts[3] = new UserAccount();
        userAccounts[3].setUsername("hello");
        userAccounts[3].setPassword("world");
        //
        userAccounts[4] = new UserAccount();
        userAccounts[4].setUsername("james");
        userAccounts[4].setPassword("gosling");
        //
        for (int i = 0; i < userAccounts.length; i++) {
            if (user.equals(userAccounts[i].getUsername()) &&
                pass.equals(userAccounts[i].getPassword())) {
                return true;
            }
        }
        //
        return false;
    }

    public static void main(String[] args) throws IOException {
        final String END_MARK = "9999";
        //
        InputStreamReader ips = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ips);
        //
```

(次のページへ)


```
while (true) {
    System.out.println("終了するときは、ユーザー名に" + END_MARK +
                        "と入力してください。");
    System.out.print("ユーザー名を入力してください：");
    String user = br.readLine();
    //
    if (user.equals(END_MARK)) {
        break;
    }
    //
    System.out.print("パスワードを入力してください：");
    String pass = br.readLine();
    //
    boolean found = isCertified(user, pass);
    if (found == true) {
        System.out.println("ログオンに成功しました。");
        break;
    }
    else {
        System.out.println("ログオンに失敗しました。");
    }
    System.out.println("");
} //while

System.out.println("¥n プログラムを終了します。");

} //main
} //class
```

4.1.5. アプリケーションの実行

ソースコードは大幅に変更したが、外部からみた振る舞いは一切変えていない。

従って、リファクタリングしただけである。

つまり、実行結果は、いままでと同じになることを確認しておこう。

終了するときは、ユーザー名に **9999** と入力してください。

ユーザー名を入力してください: duke

パスワードを入力してください: muscat

ログオンに失敗しました。

終了するときは、ユーザー名に **9999** と入力してください。

ユーザー名を入力してください: duke

パスワードを入力してください: mascot

ログオンに成功しました。

プログラムを終了します。

終了するときは、ユーザー名に **9999** と入力してください。

ユーザー名を入力してください: java

パスワードを入力してください: coffee

ログオンに失敗しました。

終了するときは、ユーザー名に **9999** と入力してください。

ユーザー名を入力してください: java

パスワードを入力してください: cafe

ログオンに成功しました。

プログラムを終了します。

(空白ページ)

4.2. コンストラクターの定義

4.2.1.概要

前回のコードは、オブジェクト指向にしたにもかかわらず、かえって今までのコードよりも煩雑になっていた。

そこで、よりシンプルなコードになるように改良する。

今回の場合は、クラスにコンストラクターという初期化処理を追加することで解決できる。

この節の目標は、コンストラクターの書き方と利用方法である。オブジェクト配列の初期化方法も理解できるようにする。

4.2.2.事前準備

seminar06 パッケージを作成し、既存のファイルをコピーしておく。

4.2.3. ソースコード

1. UserAccount クラス

```
package seminar06;

public class UserAccount {
    //=====
    // フィールド変数 (メンバー変数)
    //=====
    private String m_username;
    private String m_password;

    //=====
    // コンストラクター
    //=====
    public UserAccount(String username, String password) {
        this.setUsername(username);
        this.setPassword(password);
        //
        // 今の段階では、まだ、次のように書いてもよい。
        //
        // this.m_username = username;
        // this.m_password = password;
        //
    }

    //=====
    // メソッド
    //=====
    //
    // Username (get/set)
    //
    public String getUsername() {
        return this.m_username;
    }
    public void setUsername(String username) {
        this.m_username = username;
    }
    //
    // Password (get/set)
    //
    public String getPassword() {
        return this.m_password;
    }
    public void setPassword(String password) {
        this.m_password = password;
    }
}
```

2. LogonMain クラス

```

package seminar06;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class LogonMain {
    //=====
    // フィールド変数 (メンバー変数)
    //=====
    private static UserAccount[] userAccounts = new UserAccount[5];

    //=====
    // メソッド (isCertified)
    //=====
    private static boolean isCertified(String user, String pass) {
        userAccounts[0] = new UserAccount("yama", "kawa");
        userAccounts[1] = new UserAccount("java", "cafe");
        userAccounts[2] = new UserAccount("duke", "mascot");
        userAccounts[3] = new UserAccount("hello", "world");
        userAccounts[4] = new UserAccount("james", "gosling");
        //
        for (int i = 0; i < userAccounts.length; i++) {
            if (user.equals(userAccounts[i].getUsername()) &&
                pass.equals(userAccounts[i].getPassword())) {
                return true;
            }
        }
        //
        return false;
    }

    //=====
    // メソッド (main)
    //=====
    public static void main(String[] args) throws IOException {
        final String END_MARK = "9999";
        //
        InputStreamReader ips = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ips);
        //
        while (true) {
            //
            System.out.println("終了するときは、ユーザー名に" + END_MARK +
                               "と入力してください。");

            System.out.print("ユーザー名を入力してください : ");
            String user = br.readLine();
            //
            if (user.equals(END_MARK)) {
                break;
            }
            //
            System.out.print("パスワードを入力してください : ");
            String pass = br.readLine();
            //
            boolean found = isCertified(user, pass);

```

(次のページへ)

```
        if (found == true) {
            System.out.println("ログオンに成功しました。");
            break;
        }
        else {
            System.out.println("ログオンに失敗しました。");
        }
        System.out.println("");
    }
    System.out.println("¥n プログラムを終了します。");
}
}
```

4.2.4.配列の初期化の方法を使用する

配列の初期化の方法を適用して、少しでもシンプルなソースコードにしてみる。

パッケージを改めて、seminar07 にコピーしてから作業を行う。

1. LogonMain クラス

```
package seminar07;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class LogonMain {
    //=====
    // フィールド変数 (メンバー変数)
    //=====
    private static UserAccount[] userAccounts = new UserAccount[] {
        new UserAccount("yama", "kawa"),
        new UserAccount("java", "cafe"),
        new UserAccount("duke", "mascot"),
        new UserAccount("hello", "world"),
        new UserAccount("james", "gosling"),
    };

    //=====
    // メソッド (isCertified)
    //=====
    private static boolean isCertified(String user, String pass) {
        //
        // ここにあったコードは、フィールド変数で初期化するように変更しました。
        //
        for (int i = 0; i < userAccounts.length; i++) {
            if (user.equals(userAccounts[i].getUsername()) &&
                pass.equals(userAccounts[i].getPassword())) {
                return true;
            }
        }
        //
        return false;
    }

    //=====
    // メソッド (main)
    //=====
    public static void main(String[] args) throws IOException {
        final String END_MARK = "9999";
        //
        InputStreamReader ips = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ips);
        //
    }
}
```

(次のページへ)


```

        while (true) {
            System.out.println("終了するときは、ユーザー名に" + END_MARK +
                               "と入力してください。");

            System.out.print("ユーザー名を入力してください：");
            String user = br.readLine();
            //
            if (user.equals(END_MARK)) {
                break;
            }
            //
            System.out.print("パスワードを入力してください：");
            String pass = br.readLine();
            //
            boolean found = isCertified(user, pass);
            if (found == true) {
                System.out.println("ログオンに成功しました。");
                break;
            }
            else {
                System.out.println("ログオンに失敗しました。");
            }
            System.out.println("");
        }
        System.out.println("¥n プログラムを終了します。");
    }
}

```

4.2.5.アプリケーションの実行

今回も、外部からみた振る舞いは一切変えていない。

リファクタリングしただけである。

そこで、実行結果は、いままでと同じになることを確認しておこう。

4.3. 新たなメソッドの定義

4.3.1. 概要

isCertified メソッドのコードのうち、下記の強調表示した部分は「リクエストされた文字列 (user と pass) が登録されているアカウントの中に存在しているかどうかを調べる」処理、つまり一言でいうと、isCertified メソッドのかなめの「認証」処理を行っている部分である。

```
for (int i = 0; i < userAccounts.length; i++) {  
    if (user.equals(userAccounts[i].getUsername()) &&  
        pass.equals(userAccounts[i].getPassword())) {  
        return true;  
    }  
}
```

しかし、そのこと（認証を行っているということ）がややわかりづらいコードとなっている。

また、この「認証」処理は、UserAccount クラスを使用する場面では、ほぼ必須の処理といえる。

以上2つの理由から、この「認証」処理を、UserAccount クラスのメソッドとして作成し、ここからは (isCertified メソッドからは) そのメソッドを呼び出すようにしてみる。

UserAccount クラスに作成する新しいメソッドは、isIdenticalWith とする。

この作業により、次の2点のメリットが得られる。

1. 認証ロジックが UserAccount クラスに存在するので、それを呼び出すだけでよい。
2. isCertified メソッドのコードが読みやすくなる。

ここでの目標は、上記の内容を理解することである。

4.3.2. 事前準備

既存コードを、新しいパッケージ seminar08 にコピーしてから作業を行う。

4.3.3. ソースコード

1. UserAccount クラス

```
package seminar08;

public class UserAccount {
    //=====
    // フィールド変数 (メンバー変数)
    //=====
    private String m_username;
    private String m_password;

    //=====
    // コンストラクター
    //=====
    public UserAccount(String username, String password) {
        this.setUsername(username);
        this.setPassword(password);
    }

    //=====
    // メソッド (getter/setter)
    //=====
    //
    // Username (get/set)
    //
    public String getUsername() {
        return this.m_username;
    }
    public void setUsername(String username) {
        this.m_username = username;
    }
    //
    // Password (get/set)
    //
    public String getPassword() {
        return this.m_password;
    }
    public void setPassword(String password) {
        this.m_password = password;
    }

    //=====
    // メソッド (一般)
    //=====
    public boolean isIdenticalWith(String username, String password) {
        boolean ret = false;
        if (username.equals(this.getUsername()) &&
            password.equals(this.getPassword())) {
            ret = true;
        }
        return ret;
    }
}
```

2. LogonMain クラス

isCertified の if 文の条件の部分を修正する。

```
package seminar08;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class LogonMain {
    //=====
    // フィールド変数 (メンバー変数)
    //=====
    private static UserAccount[] userAccounts = new UserAccount[] {
        new UserAccount("yama", "kawa"),
        new UserAccount("java", "cafe"),
        new UserAccount("duke", "mascot"),
        new UserAccount("hello", "world"),
        new UserAccount("james", "gosling"),
    };

    //=====
    // メソッド (isCertified)
    //=====
    private static boolean isCertified(String user, String pass) {
        for (int i = 0; i < userAccounts.length; i++) {
            //##### 008 変更箇所
            if (userAccounts[i].isIdenticalWith(user, pass)) {
                return true;
            }
        }
        //
        return false;
    }

    //=====
    // メソッド (main)
    //=====
    public static void main(String[] args) throws IOException {
        final String END_MARK = "9999";
        //
        InputStreamReader ips = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ips);
        //
        while (true) {
            System.out.println("終了するときは、ユーザー名に" + END_MARK +
                               "と入力してください。");

            System.out.print("ユーザー名を入力してください: ");
            String user = br.readLine();
            //
            if (user.equals(END_MARK)) {
                break;
            }
            //
        }
    }
}
```

(次のページへ)

```
        System.out.print("パスワードを入力してください：");
        String pass = br.readLine();
        //
        boolean found = isCertified(user, pass);
        if (found == true) {
            System.out.println("ログオンに成功しました。");
            break;
        }
        else {
            System.out.println("ログオンに失敗しました。");
        }
        System.out.println("");
    }
    System.out.println("¥n プログラムを終了します。");
}
}
```

4.3.4.アプリケーションの実行

今回も、リファクタリングしただけである。

実行結果が、いままでと同じになることを確認しておこう。

第5章. オブジェクトを機能ごとに分ける

5.1. アカウント管理オブジェクトの導入

5.1.1.概要

LogonMain クラスの、①フィールド変数 (UserAccount の配列データ) と、②isCertified メソッドは、キーボード入力や画面表示処理から切り離されたため、LogonMain クラスから独立させることができる。

独立させることにより、他のアプリケーションに対しても、「複数アカウントのログオン処理の機能」を提供することができる。(他のアプリケーションからも再利用できるようになる)

そこで、LogonMain クラスの、①フィールド変数 (UserAccount の配列データ) と、②isCertified メソッドの部分を、UserAccountManager というクラスとして独立させる。

今までは、図1のような構成であった。これを図2のような構成に変更する。

図1:

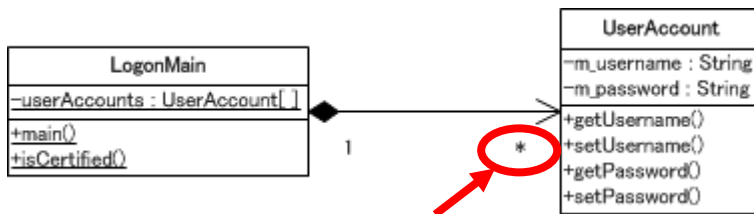
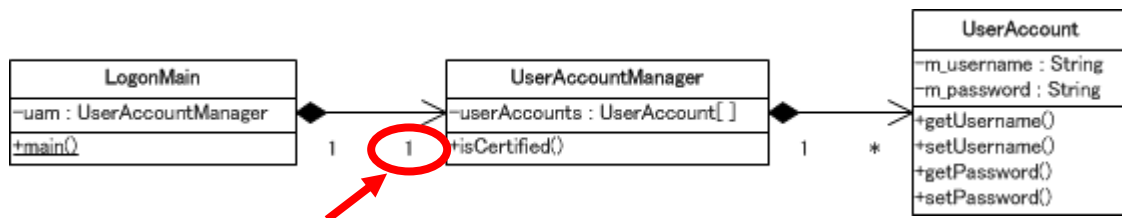


図2:



上記の楕円で囲んだ部分に注目する。LogonMain から見た場合、ユーザーを「複数形」で扱わなくてはならないが、図にの場合は「単数形」で扱うことができる。
つまり、mainの側のコードを単純化できる。

ここの節の目標は、上記の内容を理解することである。

5.1.2.事前準備

既存コードを、新しいパッケージ seminar09 にコピーしてから作業を行う。

5.1.3. ソースコード

1. UserAccount クラス

(このクラスに変更はない)

2. UserAccountManager クラス

このクラスは新たに作成する。

そして、LogonMain からコピーし、不要な部分は削除（あるいはコメントに）する。

```
package seminar09;

public class UserAccountManager {
    //=====
    //   フィールド変数 (メンバー変数)
    //=====
    private /*static*/ UserAccount[] userAccounts = new UserAccount[] {
        new UserAccount("yama", "kawa"),
        new UserAccount("java", "cafe"),
        new UserAccount("duke", "mascot"),
        new UserAccount("hello", "world"),
        new UserAccount("james", "gosling"),
    };

    //=====
    //   メソッド
    //=====
    public /*static*/ boolean isCertified(String user, String pass) {
        for (int i = 0; i < userAccounts.length; i++) {
            if (userAccounts[i].isIdenticalWith(user, pass)) {
                return true;
            }
        }
        return false;
    }
}
```

3. LogonMain クラス

```

package seminar09;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class LogonMain {

    //=====
    // フィールド変数 (メンバー変数)
    //=====
    //##### 009 追加
    private static UserManager uam = new UserManager();

    // private static UserAccount[] userAccounts = new UserAccount[] {
    //     new UserAccount("yama", "kawa"),
    //     new UserAccount("java", "cafe"),
    //     new UserAccount("duke", "mascot"),
    //     new UserAccount("hello", "world"),
    //     new UserAccount("james", "gosling"),
    // };

    // //=====
    // // メソッド (isCertified)
    // //=====
    // private static boolean isCertified(String user, String pass) {
    //     for (int i = 0; i < userAccounts.length; i++) {
    //         if (userAccounts[i].isIdenticalWith(user, pass)) {
    //             return true;
    //         }
    //     }
    //     return false;
    // }

    //=====
    // メソッド (main)
    //=====
    public static void main(String[] args) throws IOException {
        final String END_MARK = "9999";
        //
        InputStreamReader ips = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ips);
        //
        while (true) {
            System.out.println("終了するときは、ユーザー名に" + END_MARK +
                               "と入力してください。");
            System.out.print("ユーザー名を入力してください: ");
            String user = br.readLine();
            //
            if (user.equals(END_MARK)) {
                break;
            }
            //
        }
    }
}

```

(次のページへ)

```
        System.out.print("パスワードを入力してください：");
        String pass = br.readLine();
        //
//##### 009 修正
        boolean found = uam.isCertified(user, pass);
        if (found == true) {
            System.out.println("ログオンに成功しました。");
            break;
        }
        else {
            System.out.println("ログオンに失敗しました。");
        }
        System.out.println("");
    }
    System.out.println("¥n プログラムを終了します。");
}
}
```

5.1.4.アプリケーションの実行

今回も、リファクタリングしただけである。

実行結果が、いままでと同じになることを確認しておこう。

5.2. アカウントを配列からコレクションに変更する

5.2.1.概要

これまでは、配列を使用していたため、ユーザーアカウントの数を増やしたり減らしたりすることができなかった。

そこで、配列の代わりに **ArrayList** オブジェクトを導入して、ユーザーアカウント数を増減させられるようなデータ構造とする。

なお、ユーザーアカウントの追加機能自体は、次の章で実装する。

コレクションフレームワークに属するクラスのうち、**ArrayList** の使い方を理解することが目標である。

5.2.2.事前準備

既存コードを、新しいパッケージ **seminar10** にコピーしてから作業を行う。

5.2.3.ソースコード

1. UserAccount クラス

(このクラスに変更はない)

2. UserManager クラス

```

package seminar10;

//##### 010 追加
import java.util.ArrayList;

public class UserManager {
    //=====
    // フィールド変数 (メンバー変数)
    //=====
    //##### 010 追加
    private ArrayList<UserAccount> m_userAccounts = new ArrayList<UserAccount>();

    // private UserAccount[] userAccounts = new UserAccount[] {
    // new UserAccount("yama", "kawa"),
    // new UserAccount("java", "cafe"),
    // new UserAccount("duke", "mascot"),
    // new UserAccount("hello", "world"),
    // new UserAccount("james", "gosling"),
    // };

    //=====
    // コンストラクター
    //=====
    //##### 010 追加
    public UserManager() {
        this.m_userAccounts.add(new UserAccount("yama", "kawa"));
        this.m_userAccounts.add(new UserAccount("java", "cafe"));
        this.m_userAccounts.add(new UserAccount("duke", "mascot"));
        this.m_userAccounts.add(new UserAccount("hello", "world"));
        this.m_userAccounts.add(new UserAccount("james", "gosling"));
    }

    //=====
    // メソッド
    //=====
    public boolean isCertified(String user, String pass) {
    //##### 010 修正
        for (int i = 0; i < m_userAccounts.size(); i++) {
            if (m_userAccounts.get(i).isIdenticalWith(user, pass)) {
                return true;
            }
        }
        return false;
    }
}

```

3. LogonMain クラス

変更箇所はないが、無駄なコメントなどを削除したコードを掲載する。

```
package seminar10;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

import seminar09.UserAccountManager;

public class LogonMain {
    //=====
    // フィールド変数 (メンバー変数)
    //=====
    private static UserAccountManager uam = new UserAccountManager();

    //=====
    // メソッド (main)
    //=====
    public static void main(String[] args) throws IOException {
        final String END_MARK = "9999";
        //
        InputStreamReader ips = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ips);
        //
        while (true) {
            System.out.println("終了するときは、ユーザー名に" + END_MARK +
                               "と入力してください。");

            System.out.print("ユーザー名を入力してください：");
            String user = br.readLine();
            //
            if (user.equals(END_MARK)) {
                break;
            }
            //
            System.out.print("パスワードを入力してください：");
            String pass = br.readLine();
            //
            boolean found = uam.isCertified(user, pass);
            if (found == true) {
                System.out.println("ログオンに成功しました。");
                break;
            }
            else {
                System.out.println("ログオンに失敗しました。");
            }
            System.out.println("");
        }
        System.out.println("¥n プログラムを終了します。");
    }
}
```

5.2.4.アプリケーションの実行

今回も、リファクタリングしただけである。

実行結果が、いままでと同じになることを確認しておこう。

(ユーザーの追加機能は、次の章で実装する)

(空白ページ)

第6章. アカウント追加・一覧表示機能の追加

6.1. 機能追加の準備

6.1.1.概要

この章では、ユーザーアカウントの一覧表示と追加ができるような機能を実装する。
この節では、まずは、そのための準備を行う。

具体的には、アプリケーションを起動した直後のメッセージを次のように変更する。

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>

このうち「1..ログオン」を選択した場合、今までのログオン処理が実行されるようにする。

一方、「2..アカウント一覧」と「3..アカウント追加」は、現段階では、ダミーのメッセージを表示だけにする。

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>2
(未完成です)

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>3
(未完成です)

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>1
終了するときは、ユーザー名に 9999 と入力してください。
ユーザー名を入力してください: hello
パスワードを入力してください: world
ログオンに成功しました。

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>9

プログラムを終了します。

6.1.2.事前準備

既存コードを、新しいパッケージ seminar11 にコピーしてから作業を行う。

6.1.3.ソースコード

1. UserAccount クラス

(このクラスに変更はない)

2. UserAccountManager クラス

このクラスも変更箇所はないが、無駄なコメントなどを削除したコードを掲載する。

```
package seminar11;

import java.util.ArrayList;

public class UserAccountManager {
    //=====
    // フィールド変数 (メンバー変数)
    //=====
    //Java8 から右辺のジェネリック型は省略できるようになった。
    private ArrayList<UserAccount> m_userAccount = new ArrayList<>();

    //=====
    // コンストラクター
    //=====
    public UserAccountManager() {
        this.m_userAccount.add(new UserAccount("yama", "kawa"));
        this.m_userAccount.add(new UserAccount("java", "cafe"));
        this.m_userAccount.add(new UserAccount("duke", "mascot"));
        this.m_userAccount.add(new UserAccount("hello", "world"));
        this.m_userAccount.add(new UserAccount("james", "gosling"));
    }

    //=====
    // メソッド
    //=====
    public boolean isCertified(String user, String pass) {
        for (int i = 0; i < m_userAccount.size(); i++) {
            if (m_userAccount.get(i).isIdenticalWith(user, pass)) {
                return true;
            }
        }
        return false;
    }
}
```

3. LogonMain クラス

```

package seminar11;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class LogonMain {
    //=====
    // フィールド変数 (メンバー変数)
    //=====
    private static UserAccountManager uam = new UserAccountManager();
    //##### 011 追加
    private static BufferedReader reader;

    //=====
    // メソッド (機能ロジック別)
    //=====
    //(1) ログオン認証機能
    //##### 011 メソッド追加 (main からコピーして、注意深く修正する)
    private static void Logon() throws IOException {
        //
        // (1) 従来のmain()関数のコードをここに貼り付ける。
        // (2) 不要な部分を削除する (コメントにする)。
        // (3) br を reader に書き換える
        //
        final String END_MARK = "9999";
        //
        // InputStreamReader ips = new InputStreamReader(System.in);
        // BufferedReader br = new BufferedReader(ips);
        //
        while (true) {
            //
            System.out.println("終了するときは、ユーザー名に" + END_MARK +
                               "と入力してください。");

            System.out.print("ユーザー名を入力してください：");
            String user = reader.readLine();
            //
            if (user.equals(END_MARK)) {
                break;
            }
            //
            System.out.print("パスワードを入力してください：");
            String pass = reader.readLine();
            //
            boolean found = uam.isCertified(user, pass);
            if (found == true) {
                System.out.println("ログオンに成功しました。");
                break;
            }
            else {
                System.out.println("ログオンに失敗しました。");
            }
            System.out.println("");
        }
        //
        System.out.println("¥n プログラムを終了します。");
    }
}

```

(次のページへ)

```

// (2) アカウント一覧表示機能
##### 011 追加
private static void ListUserAccounts() {
    System.out.println(" (未完成です) ");
}

// (3) アカウント追加機能
##### 011 追加
private static void addUserAccount() throws IOException {
    System.out.println(" (未完成です) ");
}

//=====
// メソッド (main)
//=====
##### 011 修正
public static void main(String[] args) throws IOException {
    InputStreamReader ips = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(ips);
    reader = br;
    //
    while (true) {
        System.out.println(
            "1.. ログオン、2.. アカウント一覧、3.. アカウント追加、9.. 終了");
        System.out.print("番号>");
        String cmd = br.readLine();
        //
        if (cmd.equals("9")) {
            break;
        }
        else if (cmd.equals("1")) {
            Logon();
        }
        else if (cmd.equals("2")) {
            ListUserAccounts();
        }
        else if (cmd.equals("3")) {
            addUserAccount();
        }
        else {
            System.out.println("正しい番号を入力してください。");
        }
        System.out.println("");
    }
    System.out.println("¥n プログラムを終了します。");
}
}

```

6.1.4.アプリケーションの実行

アプリケーションの機能が変更されたので、動作確認をしっかりとこよう。

1.. ログオン、2.. アカウント一覧、3.. アカウント追加、9.. 終了
番号>2
(未完成です)

1.. ログオン、2.. アカウント一覧、3.. アカウント追加、9.. 終了
番号>3
(未完成です)

1.. ログオン、2.. アカウント一覧、3.. アカウント追加、9.. 終了
番号>1
終了するときは、ユーザー名に 9999 と入力してください。
ユーザー名を入力してください: hello
パスワードを入力してください: goodbye
ログオンに失敗しました。

終了するときは、ユーザー名に 9999 と入力してください。
ユーザー名を入力してください: hello
パスワードを入力してください: world
ログオンに成功しました。

1.. ログオン、2.. アカウント一覧、3.. アカウント追加、9.. 終了
番号>9
プログラムを終了します。

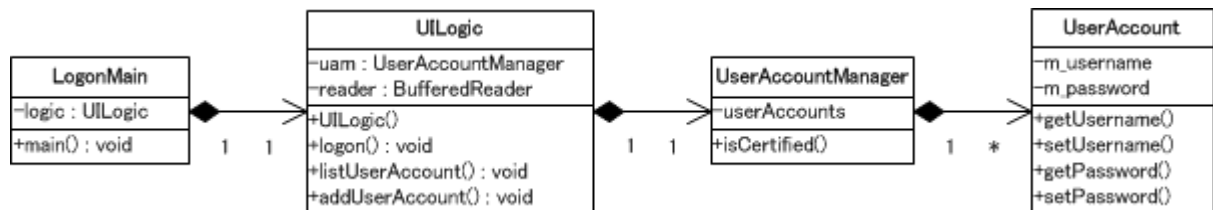
6.2. クラスの分離

6.2.1.概要

UILogic という新しクラスを作成し、LogonMain クラス内の3つのメソッド(logon, listUserAccount, addUserAccount) を、そのクラスへ移動する。

つまり、Logon クラスは、main メソッドでメインメニューの制御だけを行うことにして、各メニュー以降の処理を UILogic というクラスに任せることにする。

図にすると、下記のようなになる。



6.2.2.事前準備

既存コードを、新しいパッケージ seminar12 にコピーしてから作業を行う。

6.2.3.ソースコード

1. UserAccount クラス

(このクラスに変更はない)

2. UserAccountManager クラス

(このクラスも変更はない)

3. UILogic クラス

LogonMain クラスの main() 関数からコピー・貼り付けを行い、下記の修正を行う。

- ① インスタンスとして使用できるようにするため、static を取り除く。
- ② 公開するメンバーは public にする。
- ③ BufferedReader を受け取るためのコンストラクターを追加する。

```
package seminar12;

import java.io.BufferedReader;
import java.io.IOException;

public class UILogic {
    //=====
    // フィールド変数 (メンバー変数)
    //=====
    //##### 012 修正
    private /*static*/ UserManager uam = new UserManager();
    private /*static*/ BufferedReader reader;

    //=====
    // コンストラクター
    //=====
    //##### 012 追加
    public UILogic(BufferedReader br) {
        this.reader = br;
    }

    //=====
    // メソッド (機能ロジック別)
    //=====
    //(1) ログオン認証機能
    //##### 012 修正
    public /*static*/ void logon() throws IOException {
        final String END_MARK = "9999";
        //
        while (true) {
            System.out.println("終了するときは、ユーザー名に" + END_MARK +
                               "と入力してください。");

            System.out.print("ユーザー名を入力してください: ");
            String user = reader.readLine();
            //
            if (user.equals(END_MARK)) {
                break;
            }
            //
            System.out.print("パスワードを入力してください: ");
            String pass = reader.readLine();
        }
    }
}
```

(次のページへ)


```

        //
        boolean found = uam.isCertified(user, pass);
        if (found == true) {
            System.out.println("ログオンに成功しました。");
            break;
        }
        else {
            System.out.println("ログオンに失敗しました。");
        }
        System.out.println("");
    }
}

//(2) アカウント一覧表示機能
//##### 012 修正
    public /*static*/ void listUserAccounts() {
        System.out.println("（未完成です）");
    }

//(3) アカウント追加機能
//##### 012 修正
    public /*static*/ void addUserAccount() throws IOException {
        System.out.println("（未完成です）");
    }
}

```

4. LogonMain クラス

UILogic クラスに移動した部分を削除（コメント化）し、UILogic の機能呼び出すように修正する。

```
package seminar12;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class LogonMain {
    // //=====
    // // フィールド変数 (メンバー変数)
    // //=====
    // private static UserAccountManager uam = new UserAccountManager();
    // private static BufferedReader reader;
    //
    // //=====
    // // メソッド (機能ロジック別)
    // //=====
    // //(1) ログオン認証機能
    // private static void Logon() throws IOException {
    //     final String END_MARK = "9999";
    //     //
    //     while (true) {
    //         System.out.println("終了するときは、ユーザー名に" + END_MARK +
    //             "と入力してください。");
    //         System.out.print("ユーザー名を入力してください: ");
    //         String user = reader.readLine();
    //         //
    //         if (user.equals(END_MARK)) {
    //             break;
    //         }
    //         //
    //         System.out.print("パスワードを入力してください: ");
    //         String pass = reader.readLine();
    //         //
    //         boolean found = uam.isCertified(user, pass);
    //         if (found == true) {
    //             System.out.println("ログオンに成功しました。");
    //             break;
    //         }
    //         else {
    //             System.out.println("ログオンに失敗しました。");
    //         }
    //         System.out.println("");
    //     }
    // }
    //
    // //(2) アカунト一覧表示機能
    // private static void ListUserAccounts() {
    //     System.out.println(" (未完成です) ");
    // }
}
```

(次のページへ)

```
//
// //(3) アカウント追加機能
// private static void addUserAccount() throws IOException {
//     System.out.println(" (未完成です) ");
// }

//=====
// メソッド (main)
//=====
public static void main(String[] args) throws IOException {
    InputStreamReader ips = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(ips);
//##### 012 削除と追加
    //reader = br;
    UILogic logic = new UILogic(br);
    //
    while (true) {
        System.out.println(
            "1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了");
        System.out.print("番号>");
        String cmd = br.readLine();
        //
        if (cmd.equals("9")) {
            break;
        }
        else if (cmd.equals("1")) {
            logic.login();                //##### 012 修正
        }
        else if (cmd.equals("2")) {
            logic.listUserAccounts();    //##### 012 修正
        }
        else if (cmd.equals("3")) {
            logic.addUserAccount();      //##### 012 修正
        }
        else {
            System.out.println("正しい番号を入力してください。");
        }
        System.out.println("");
    }
    System.out.println("¥n プログラムを終了します。");
}

}
```

6.2.4.アプリケーションの実行

今回は、リファクタリングしただけである。

実行結果が、いままでと同じになることを確認しておこう。

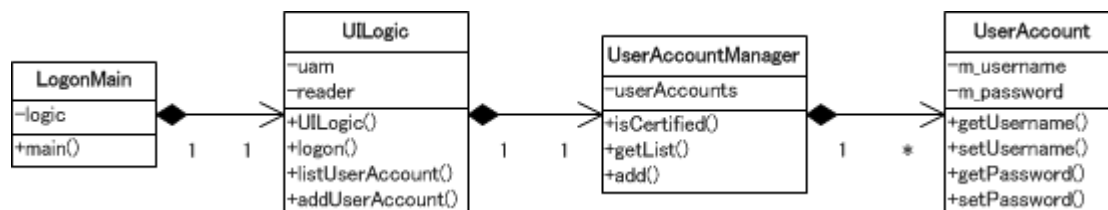
(ユーザーの追加機能は、ようやく次で実装する)

6.3. 追加機能と一覧表示機能の実装

6.3.1.概要

ようやく、ユーザーアカウントの一覧表示と新規追加の機能を実装する準備が整ったので、コードを実装する。

そのためには、UILogic クラスだけでなく、UserAccountManager クラスにも、一覧の取得メソッド (getList()) とアカウントの追加メソッド (add()) が必要なので、そのコードから記述していく。



この節の目標は、これらのコードを理解することである。

6.3.2.事前準備

既存コードを、新しいパッケージ `seminar13` にコピーしてから作業を行う。

6.3.3.ソースコード

1. UserAccount クラス

(このクラスに変更はない)

2. UserAccountManager クラス

```

package seminar13;

import java.util.ArrayList;

public class UserAccountManager {
    //=====
    // フィールド変数 (メンバー変数)
    //=====
    private ArrayList<UserAccount> m_userAccount = new ArrayList<>();

    //=====
    // コンストラクター
    //=====
    public UserAccountManager() {
        this.m_userAccount.add(new UserAccount("yama", "kawa"));
        this.m_userAccount.add(new UserAccount("java", "cafe"));
        this.m_userAccount.add(new UserAccount("duke", "mascot"));
        this.m_userAccount.add(new UserAccount("hello", "world"));
        this.m_userAccount.add(new UserAccount("james", "gosling"));
    }

    //=====
    // メソッド
    //=====
    //(1) 認証
    public boolean isCertified(String user, String pass) {
        for (int i = 0; i < m_userAccount.size(); i++) {
            if (m_userAccount.get(i).isIdenticalWith(user, pass)) {
                return true;
            }
        }
        return false;
    }

    //(2) 一覧の取得
    public String getList() {
        StringBuilder ret = new StringBuilder();
        for (UserAccount ac : this.m_userAccount) {
            ret.append("user=" + ac.getUsername() + ", pass=" + ac.getPassword() + "¥n");
        }
        return ret.toString();
    }

    //(3) アカウントの追加
    public void add(String user, String pass) {
        UserAccount ac = new UserAccount(user, pass);
        this.m_userAccount.add(ac);
    }
}

```

3. UILogic クラス

```

package seminar13;

import java.io.BufferedReader;
import java.io.IOException;

public class UILogic {
    //=====
    // フィールド変数 (メンバー変数)
    //=====
    private UserManager uam = new UserManager();
    private BufferedReader reader;

    //=====
    // コンストラクター
    //=====
    public UILogic(BufferedReader br) {
        this.reader = br;
    }

    //=====
    // メソッド (機能ロジック別)
    //=====
    //(1) ログオン認証機能
    public void logon() throws IOException {
        final String END_MARK = "9999";
        //
        while (true) {
            System.out.println("終了するときは、ユーザー一名に" + END_MARK +
                               "と入力してください。");
            System.out.print("ユーザー名を入力してください: ");
            String user = reader.readLine();
            //
            if (user.equals(END_MARK)) {
                break;
            }
            //
            System.out.print("パスワードを入力してください: ");
            String pass = reader.readLine();
            //
            boolean found = uam.isCertified(user, pass);
            if (found == true) {
                System.out.println("ログオンに成功しました。");
                break;
            }
            else {
                System.out.println("ログオンに失敗しました。");
            }
            System.out.println("");
        }
    }

    //(2) アカунト一覧表示機能
    public void listUserAccounts() {
        //System.out.println(" (未完成です) ");
        System.out.println(uam.getList());
    }
}

```

(次のページへ)

```
//(3) アカウント追加機能
public void addUserAccount() throws IOException {
    //System.out.println(" (未完成です) ");
    System.out.print("ユーザー名を入力してください:");
    String user = reader.readLine();
    //
    System.out.print("パスワードを入力してください:");
    String pass = reader.readLine();
    //
    uam.add(user, pass);
    //
    System.out.println("ユーザーアカウントを登録しました。");
}
}
```

4. LogonMain クラス

変更箇所はないが、無駄なコメントなどを削除したコードを掲載する。

```
package seminar13;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class LogonMain {
    //=====
    //   メソッド (main)
    //=====
    public static void main(String[] args) throws IOException {
        InputStreamReader ips = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(ips);
        UILogic logic = new UILogic(br);
        //
        while (true) {
            System.out.println(
                "1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了");
            System.out.print("番号>");
            String cmd = br.readLine();
            //
            if (cmd.equals("9")) {
                break;
            }
            else if (cmd.equals("1")) {
                logic.logon();
            }
            else if (cmd.equals("2")) {
                logic.listUserAccounts();
            }
            else if (cmd.equals("3")) {
                logic.addUserAccount();
            }
            else {
                System.out.println("正しい番号を入力してください。");
            }
            System.out.println("");
        }
        System.out.println("¥n プログラムを終了します。");
    }
}
```


6.3.4.アプリケーションの実行

機能追加を行ったので、動作確認をしっかりとっておこう。

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>2

```
user=yama, pass=kawa
user=java, pass=cafe
user=duke, pass=mascot
user=hello, pass=world
user=james, pass=gosling
```

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>3

ユーザー名を入力してください: admin
パスワードを入力してください: P@ssw0rd
ユーザーアカウントを登録しました。

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>2

```
user=yama, pass=kawa
user=java, pass=cafe
user=duke, pass=mascot
user=hello, pass=world
user=james, pass=gosling
user=admin, pass=P@ssw0rd
```

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>1

終了するときは、ユーザー名に 9999 と入力してください。
ユーザー名を入力してください: admin
パスワードを入力してください: password
ログオンに失敗しました。

終了するときは、ユーザー名に 9999 と入力してください。
ユーザー名を入力してください: admin
パスワードを入力してください: P@ssw0rd
ログオンに成功しました。

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>9

プログラムを終了します。

(空白ページ)

第7章. エラー（例外）処理の追加

7.1. パスワードに長さの制限を設定する

7.1.1.概要

「パスワードの長さは4文字以上とする」という制約を設定してみよう。
ユーザー名に対しても「空白のユーザー名は許可しない」という設定を追加する。

既存コードを、新しいパッケージ `seminar14` にコピーしてから作業を行う。

【ルール】

ユーザー名： 空のユーザー名、空白文字ばかりのユーザー名は許可しない
パスワード： 4文字未満のパスワードは許可しない

この節の目標は、例外処理の理解である。

7.1.2.事前準備

既存コードを、新しいパッケージ `seminar14` にコピーしてから作業を行う。

7.1.3.ソースコード

1. UserAccount クラス

```

package seminar14;

public class UserAccount {
    //=====
    // フィールド変数 (メンバー変数)
    //=====
    private String m_username;
    private String m_password;

    //=====
    // コンストラクター
    //=====
    //##### 014 一部追加
    public UserAccount(String username, String password) throws Exception {
        this.setUsername(username); //this.m_username = username は不可。
        this.setPassword(password); //this.m_password = password は不可。
    }

    //=====
    // メソッド (getter/setter)
    //=====
    // Username (get/set)
    public String getUsername() {
        return this.m_username;
    }
    //##### 014 追加
    public void setUsername(String username) throws Exception {
        if (username.trim().equals("")) {
            throw new Exception("ユーザー名が空です。");
        }
        this.m_username = username;
    }

    // Password (get/set)
    public String getPassword() {
        return this.m_password;
    }
    //##### 014 追加
    public void setPassword(String password) throws Exception {
        if (password.trim().length() < 4) {
            throw new Exception("パスワードが4文字未満です: " + password);
        }
        this.m_password = password;
    }

    //=====
    // メソッド (一般)
    //=====
    public boolean isIdenticalWith(String username, String password) {
        boolean ret = false;
        if (username.equals(this.getUsername()) &&
            password.equals(this.getPassword())) {
            ret = true;
        }
        return ret;
    }
}

```

2. UserAccountManager クラス

```

package seminar14;

import java.util.ArrayList;

public class UserAccountManager {
    //=====
    // フィールド変数 (メンバー変数)
    //=====
    private ArrayList<UserAccount> m_userAccount = new ArrayList<>();

    //=====
    // コンストラクター
    //=====
    //##### 014 追加
    public UserAccountManager() {
        try {
            this.m_userAccount.add(new UserAccount("yama", "kawa"));
            this.m_userAccount.add(new UserAccount("java", "cafe"));
            this.m_userAccount.add(new UserAccount("duke", "mascot"));
            this.m_userAccount.add(new UserAccount("hello", "world"));
            this.m_userAccount.add(new UserAccount("james", "gosling"));
        }
        catch (Exception ex) {
            ; //サンプルデータなので、エラーは無視します。
            //そもそもエラーの発生しないデータばかり用意してあります。
        }
    }

    //=====
    // メソッド
    //=====
    //(1) 認証
    public boolean isCertified(String user, String pass) {
        for (int i = 0; i < m_userAccount.size(); i++) {
            if (m_userAccount.get(i).isIdenticalWith(user, pass)) {
                return true;
            }
        }
        return false;
    }

    //(2) 一覧
    public String getList() {
        StringBuilder ret = new StringBuilder();
        for (UserAccount ac : this.m_userAccount) {
            ret.append("user=" + ac.getUsername() + ", pass=" + ac.getPassword() + "¥n");
        }
        return ret.toString();
    }

    //(3) 追加
    //##### 014 一部追加
    //このエラーは無視せず、上位に伝播するようにします。
    public void add(String user, String pass) throws Exception {
        UserAccount ac = new UserAccount(user, pass);
        this.m_userAccount.add(ac);
    }
}

```

3. UILogic クラス

アカウント追加メソッド (addUserAccount) を修正する。

```
package seminar14;

import java.io.BufferedReader;
import java.io.IOException;

public class UILogic {
    //=====
    // フィールド変数 (メンバー変数)
    //=====
    private UserAccountManager uam = new UserAccountManager();
    private BufferedReader reader;

    //=====
    // コンストラクター
    //=====
    public UILogic(BufferedReader br) {
        this.reader = br;
    }

    //=====
    // メソッド (機能ロジック別)
    //=====
    //(1) ログオン認証機能
    public void logon() throws IOException {
        final String END_MARK = "9999";
        //
        while (true) {
            System.out.println("終了するときは、ユーザー名に" + END_MARK +
                               "と入力してください。");

            System.out.print("ユーザー名を入力してください：");
            String user = reader.readLine();
            //
            if (user.equals(END_MARK)) {
                break;
            }
            //
            System.out.print("パスワードを入力してください：");
            String pass = reader.readLine();
            //
            boolean found = uam.isCertified(user, pass);
            if (found == true) {
                System.out.println("ログオンに成功しました。");
                break;
            }
            else {
                System.out.println("ログオンに失敗しました。");
            }
            System.out.println("");
        }
    }
}
```

(次のページへ)

```
//(2) アカウント一覧表示機能
public void listUserAccounts() {
    System.out.println(uam.getList());
}

//(3) アカウント追加機能
public void addUserAccount() throws IOException {
    System.out.print("ユーザー名を入力してください: ");
    String user = reader.readLine();
    //
    System.out.print("パスワードを入力してください: ");
    String pass = reader.readLine();
    //
    ##### 014 追加
    try {
        uam.add(user, pass);
        System.out.println("ユーザーアカウントを登録しました。");
    }
    catch (Exception ex) {
        System.out.println("エラー発生: " + ex.getMessage());
    }
}
}
```

4. LogonMain クラス

(このクラスには、今回、変更箇所はない)

7.1.4.アプリケーションの実行

機能追加を行ったので、動作確認をしっかりとっておこう。

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>3

ユーザー名を入力してください：

パスワードを入力してください：12345

エラー発生：ユーザー名が空です。

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>3

ユーザー名を入力してください：abcde

パスワードを入力してください：123

エラー発生：パスワードが4文字未満です：123

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>3

ユーザー名を入力してください：abcde

パスワードを入力してください：1234

ユーザーアカウントを登録しました。

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>2

user=yama, pass=kawa

user=java, pass=cafe

user=duke, pass=mascot

user=hello, pass=world

user=james, pass=gosling

user=abcde, pass=1234

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>9

プログラムを終了します。

(空白ページ)

第8章. アカウントをファイルに保存する（参考）

8.1. ユーザーアカウントをファイルに保存する（参考）

8.1.1.概要

実行中に追加されたユーザーアカウントも含め、アプリケーション内で保持しているアカウントはアプリケーションを終了すると消えてなくなってしまう。

そこで、ファイルに保存する機能を追加する。

保存したファイルからデータを復元する機能は、次の節で追加する。

8.1.2.事前準備

既存コードを、新しいパッケージ `seminar15` にコピーしてから作業を行う。

8.1.3. ソースコード

1. UserAccount クラス

ファイルに保存したり、ネットワーク転送したりするためには、保存・転送するオブジェクトが `java.io.Serializable` インターフェースを実装する必要がある。

```
package seminar15;

import java.io.Serializable;

public class UserAccount implements Serializable {
    //=====
    // フィールド変数 (メンバー変数)
    //=====
    private String m_username;
    private String m_password;

    //=====
    // コンストラクター
    //=====
    public UserAccount(String username, String password) throws Exception {
        this.setUsername(username);
        this.setPassword(password);
    }

    //=====
    // メソッド (getter/setter)
    //=====
    //
    // Username (get/set)
    //
    public String getUsername() {
        return this.m_username;
    }
    public void setUsername(String username) throws Exception {
        if (username.trim().equals("")) {
            throw new Exception("ユーザー名が空です。");
        }
        this.m_username = username;
    }
    //
    // Password (get/set)
    //
    public String getPassword() {
        return this.m_password;
    }
    public void setPassword(String password) throws Exception {
        if (password.trim().length() < 4) {
            throw new Exception("パスワードが 4 文字未満です: " + password);
        }
        this.m_password = password;
    }
}
```

(次のページへ)

```
//=====
//   メソッド (一般)
//=====
public boolean isIdenticalWith(String username, String password) {
    boolean ret = false;
    if (username.equals(this.getUsername()) &&
        password.equals(this.getPassword())) {
        ret = true;
    }
    return ret;
}
}
```

2. UserAccountManager クラス

```

package seminar15;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.ArrayList;

public class UserAccountManager {
    //=====
    // 定数
    //=====
    //##### 015 追加
    private final static String FILE_NAME = "UserAccounts.dat";

    //=====
    // フィールド変数 (メンバー変数)
    //=====
    private ArrayList<UserAccount> m_userAccount = new ArrayList<>();

    //=====
    // コンストラクター
    //=====
    public UserAccountManager() {
        try {
            this.m_userAccount.add(new UserAccount("yama", "kawa"));
            this.m_userAccount.add(new UserAccount("java", "cafe"));
            this.m_userAccount.add(new UserAccount("duke", "mascot"));
            this.m_userAccount.add(new UserAccount("hello", "world"));
            this.m_userAccount.add(new UserAccount("james", "gosling"));
        }
        catch (Exception ex) {
            ; //サンプルデータの追加なので、エラーは無視します。
            //そもそもエラーの発生しないデータばかり用意してあります。
        }
    }

    //=====
    // メソッド
    //=====
    //(1) 認証
    public boolean isCertified(String user, String pass) {
        for (int i = 0; i < m_userAccount.size(); i++) {
            if (m_userAccount.get(i).isIdenticalWith(user, pass)) {
                return true;
            }
        }
        return false;
    }

    //(2) 一覧の取得

```

(次のページへ)

```
public String getList() {
    StringBuilder ret = new StringBuilder();
    for (UserAccount ac : this.m_userAccount) {
        ret.append("user=" + ac.getUsername() + ", pass=" + ac.getPassword() + "¥n");
    }
    return ret.toString();
}

//(3) アカウントの追加
//このエラーは無視せず、上位に伝播するようにします。
public void add(String user, String pass) throws Exception {
    UserAccount ac = new UserAccount(user, pass);
    this.m_userAccount.add(ac);
}

//(4) データをファイルに保存
//##### 015 追加
public void save() throws IOException {
    FileOutputStream fos = new FileOutputStream(FILE_NAME);
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(this.m_userAccount);
    fos.close();
}
}
```


3. UILogic クラス

```

package seminar15;

import java.io.BufferedReader;
import java.io.IOException;

public class UILogic {
    //=====
    // フィールド変数 (メンバー変数)
    //=====
    private UserManager uam = new UserManager();
    private BufferedReader reader;

    //=====
    // コンストラクター
    //=====
    public UILogic(BufferedReader br) {
        this.reader = br;
    }

    //=====
    // メソッド (機能ロジック別)
    //=====
    //(1) ログオン認証機能
    public void logon() throws IOException {
        final String END_MARK = "9999";
        //
        while (true) {
            System.out.println("終了するときは、ユーザー名に" + END_MARK +
                               "と入力してください。");

            System.out.print("ユーザー名を入力してください：");
            String user = reader.readLine();
            //
            if (user.equals(END_MARK)) {
                break;
            }
            //
            System.out.print("パスワードを入力してください：");
            String pass = reader.readLine();
            //
            boolean found = uam.isCertified(user, pass);
            if (found == true) {
                System.out.println("ログオンに成功しました。");
                break;
            }
            else {
                System.out.println("ログオンに失敗しました。");
            }
            System.out.println("");
        }
    }
}

```

(次のページへ)

```
}

//(2) アカウント一覧表示機能
public void listUserAccounts() {
    System.out.println(uam.getList());
}

//(3) アカウント追加機能
public void addUserAccount() throws IOException {
    System.out.print("ユーザー名を入力してください：");
    String user = reader.readLine();
    //
    System.out.print("パスワードを入力してください：");
    String pass = reader.readLine();
    //
    try {
        uam.add(user, pass);
//##### 015 追加
        uam.save();
        System.out.println("ユーザーアカウントを登録しました。");
    }
    catch (Exception ex) {
        System.out.println("エラー発生：" + ex.getMessage());
    }
}

}
```

4. LogonMain クラス

(このクラスには、今回、変更箇所はない)

8.1.4.アプリケーションの実行

機能追加を行ったので、動作確認をしっかりとっておこう。

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>3

ユーザー名を入力してください：abcde

パスワードを入力してください：1234

ユーザーアカウントを登録しました。

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>2

user=yama, pass=kawa

user=java, pass=cafe

user=duke, pass=mascot

user=hello, pass=world

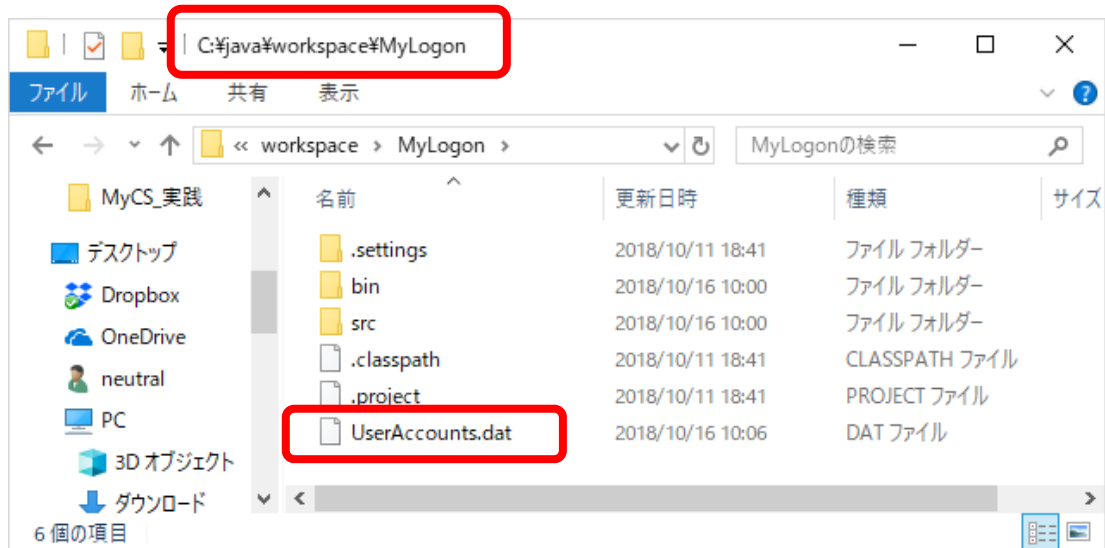
user=james, pass=gosling

user=abcde, pass=1234

1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了
番号>9

プログラムを終了します。

ファイルが作成されているかどうか、確認する。



8.2. ファイルからユーザーアカウントを読み込む

8.2.1.概要

ユーザーアカウントが保存されたファイルを読み込み、データを復元する処理を追加する。

これにより、今後は、メモリーではなく、ファイルにアカウント情報を保存し、利用できるようになる。

なお、アカウントの削除機能は、これまでに実装してこなかったので、時間に余裕がある場合は、メインメニュー以降に、アカウントの削除機能を追加してみよう。(これは、仕様を考え、自分で設計することから始めてみるとよい)

8.2.2.事前準備

この実習は、新しいパッケージを作らず、**seminar15** パッケージのものをバージョンアップする。

(もし、**seminar16** などというパッケージで作業してしまうと、**seminar15** で作成したファイルを読み込む時に「クラス名が違う」というエラーが発生するため。詳細に説明すると、**seminar15.UserAccount** クラスとして保存したものを、こちらでは **seminar16.UserAccount** として読み込もうとしてしまうから)

8.2.3.ソースコード

1. UserAccount クラス

(このクラスには、変更箇所はない)

2. UserAccountManager クラス

```

package seminar15;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;

public class UserAccountManager {
    //=====
    // 定数
    //=====
    private final static String FILE_NAME = "UserAccounts.dat";

    //=====
    // フィールド変数 (メンバー変数)
    //=====
    private ArrayList<UserAccount> m_userAccount = new ArrayList<>();

    //=====
    // コンストラクター
    //=====
    //#### 015b 修正
    @SuppressWarnings("unchecked")
    public UserAccountManager() {
        try {
            //
            this.m_userAccount.add(new UserAccount("yama", "kawa"));
            //
            this.m_userAccount.add(new UserAccount("java", "cafe"));
            //
            this.m_userAccount.add(new UserAccount("duke", "mascot"));
            //
            this.m_userAccount.add(new UserAccount("hello", "world"));
            //
            this.m_userAccount.add(new UserAccount("james", "gosling"));
            FileInputStream fis = new FileInputStream(FILE_NAME);
            ObjectInputStream ois = new ObjectInputStream(fis);
            this.m_userAccount = (ArrayList<UserAccount>)ois.readObject();
            fis.close();
        }
        catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

    //=====
    // メソッド
    //=====
    //(1) 認証
    public boolean isCertified(String user, String pass) {
        for (int i = 0; i < m_userAccount.size(); i++) {
            if (m_userAccount.get(i).isIdenticalWith(user, pass)) {
                return true;
            }
        }
    }
}

```

(次のページへ)

```
        }
    }
    return false;
}

//(2) 一覧の取得
public String getList() {
    StringBuilder ret = new StringBuilder();
    for (UserAccount ac : this.m_userAccount) {
        ret.append("user=" + ac.getUsername() + ", pass=" + ac.getPassword() + "¥n");
    }
    return ret.toString();
}

//(3) アカウントの追加
//このエラーは無視せず、上位に伝播するようにします。
public void add(String user, String pass) throws Exception {
    UserAccount ac = new UserAccount(user, pass);
    this.m_userAccount.add(ac);
}

//(4) データをファイルに保存
public void save() throws IOException {
    FileOutputStream fos = new FileOutputStream(FILE_NAME);
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(this.m_userAccount);
    fos.close();
}
}
```

3. UILogic クラス

(このクラスには、今回、変更箇所はない)

4. LogonMain クラス

(このクラスには、今回、変更箇所はない)

8.2.4.アプリケーションの実行

機能追加を行ったので、動作確認をしっかりとこよう。

さきほど追加したアカウントが、アプリケーション起動と同時に読み込まれているはずである。

```
1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了  
番号>2  
user=yama, pass=kawa  
user=java, pass=cafe  
user=duke, pass=mascot  
user=hello, pass=world  
user=james, pass=gosling  
user=abcde, pass=1234
```

```
1..ログオン、2..アカウント一覧、3..アカウント追加、9..終了  
番号>9
```

プログラムを終了します。