はじめに

プロになるJava

Javaの勉強を始める人の多くが、エンジニアとして仕事がしたい、よりよい条件で仕事がしたいということで勉強を始めているのではないかと思います。つまり、プロになるということです。エンジニアとしてプロになるために大事なのは、プログラムが組めること、アプリケーションが作れることです。そのために、Javaについてある程度知っておく必要があります。「全部同じでは」と思うかもしれませんが、実際は少しずつ違います。

「プログラムが組める」というのは、コンピュータでのデータの表現や操作を知って、そのような操作を並べて手続きを組めること、組んだ手続きを構成してまとめられることです。「アプリケーションが作れる」というのは、だれかがやりたいことをコンピュータで実現するために、入力を加工して出力するプログラムが作れることです。そして、プログラムを組んでアプリケーションを作ることをJavaを使ってできるのが、「Javaを知る」ということになります。

Javaの入門書は多くありますが、多くの本では「Javaを知る」の部分に主眼が置かれて、「プログラムが組める」という部分はあまり書かれていません。またユーザーインタフェースの解説がないために、「アプリケーションを作る」というところまで到達しにくくなっています。そこでこの本では、プログラムを組むことを目的に、手続きの組み方や構成を解説して、そのために必要なJavaの文法を紹介するという形をとっています。また、アプリケーションに関しては、世の中のアプリケーションがWeb主体になっているため、Webの仕組みを説明したあとでフレームワークを使ったWebアプリケーションの解説を行います。

プログラムを組んでアプリケーションを作る作業は頭を使う作業です。時間をかけて構築していく必要もあります。間違いの混入も防がないといけません。また、他の人が作ったプログラムを組み込んだり、一緒に作業したりということもあります。そのような中でも、機械的にできることは多く、そういったことはツールに任せることで、考える作業に集中することができます。プロとしてのプログラミングでは、ツールを活用して考えることに集中して間違いの少ないプログラムを作るということも大切です。この本では、プログラミングを助けるツールや考え方についても解説しています。

入門書としては難しいことまで書いてあるかもしれませんが、必要な知識でもあります。がんばって勉強してください。

2022年2月吉日

著者一同

```
jshell> var b = new JButton("Upper")
b ==> javax.swing.JButton[,0,0,0x0,invalid, ... text=Upper,defaultCapable=true]
```

JButtonオブジェクトを生成して変数bに割り当てました。
このボタンを"Center"の位置に配置しましょう。

```
jshell> f.add("Center", b)
$18 ==> javax.swing.JButton[,0,0,0x0,invalid, ... text=Upper,defaultCapable=true]
```

validateメソッドを呼び出すとウィンドウの中央にボタンが表示されます（図6.9）。

```
jshell> f.validate()
```

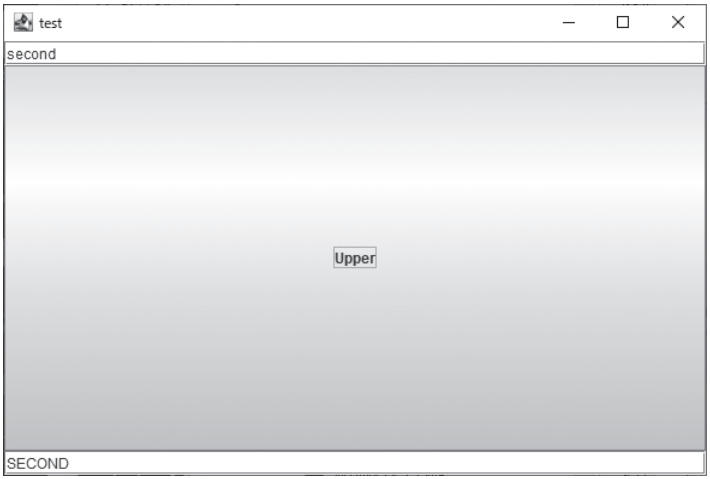

図6.9 ● ボタンを配置

ボタンが表示されましたが、まだ何も処理を設定していないのでこのボタンを押しても何も起きません。

では、ボタンがクリックされたときの処理を設定してみましょう。ボタンがクリックされたときの処理を設定するのはaddActionListenerメソッドです。「addAc」まで入力して［Tab］キーを押すと続きが入力されます。引数には次のように入力します。

```
jshell> b.addActionListener(ae -> t.setText(t2.getText().toUpperCase()))
```

この時点では何も起きません。下側の入力領域に「hello」と入力してボタンをクリックすると、上側の入力領域に「HELLO」と表示されます（**図6.10**）。

図6.10 ● ボタンをクリックすると処理が実行

このとき、ボタンの処理として**ラムダ式**と呼ばれる書き方で渡しています。「ae」の部分はなんでもかまわないのですが、ここでは`ActionEvent`オブジェクトが渡されてくることになっているので頭文字を取って`ae`としています。

ラムダ式は次のような形になっています。

構文 ラムダ式

```
変数名 -> 処理
```

「必要なときにこの処理を呼び出してください」というときにラムダ式で処理を渡します。今回は、下部の入力領域への入力を大文字にして上部に表示するという処理を、ボタンが押されたときの処理として`addActionListener`メソッドで渡しています。`addActionListener`メソッドの時点では何も起きません。

ラムダ式については第10章の「10.2.3 ラムダ式」や第11章の「11.2 ラムダ式とメソッド参照」で詳しく説明します。

6.1.6 クラスとオブジェクト、インスタンス

GUI部品だけではなく、文字列や日付などもオブジェクトだという話をしました。文字列は`String`クラスのオブジェクト、日付は`LocalDate`クラスのオブジェクトです。クラスはデータ

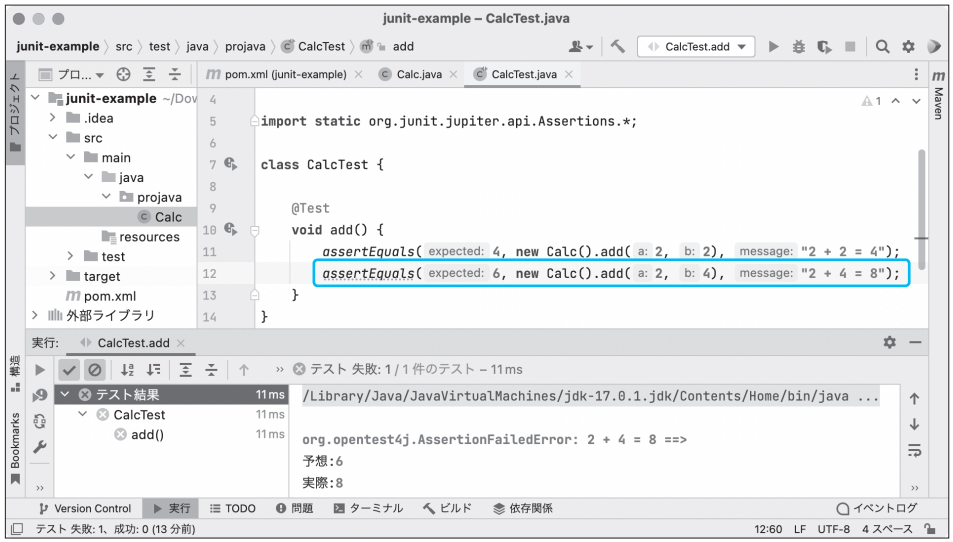

図18.7 ● テストが失敗した状態

今度はテストが失敗しました。これで、足し算の代わりにかけ算を行ってしまっているバグを検出できるテストが書けたことになります。

あとはCalcクラスに移動して実装を修正していきます。Calcクラスへ移動するときは、テストを作成する際に使った［Ctrl］＋［Shift］＋［T］（［Shift］＋［Command］＋［T］）キーを使うと瞬時に戻ることができます。「a * b」とかけ算を行っている箇所を「a + b」と修正しましょう。再度テストを実行する場合、前に使ったショートカットキー［Ctrl］＋［Shift］＋［F10］（［Control］＋［Shift］＋［R］）は使えません。これは開いているファイルのテストやmainメソッドを実行するコマンドで、Calcクラスには存在しないmainメソッドを実行しようとしてしまうからです。しかしテストクラスに戻らなくても、Calc.javaを開いたまま［Shift］＋［F10］（［Control］＋［R］）キーを押す（Runコマンド）と「最後に実行した設定で再度実行」できます。今回は図18.8のようにテストが成功するはずです。

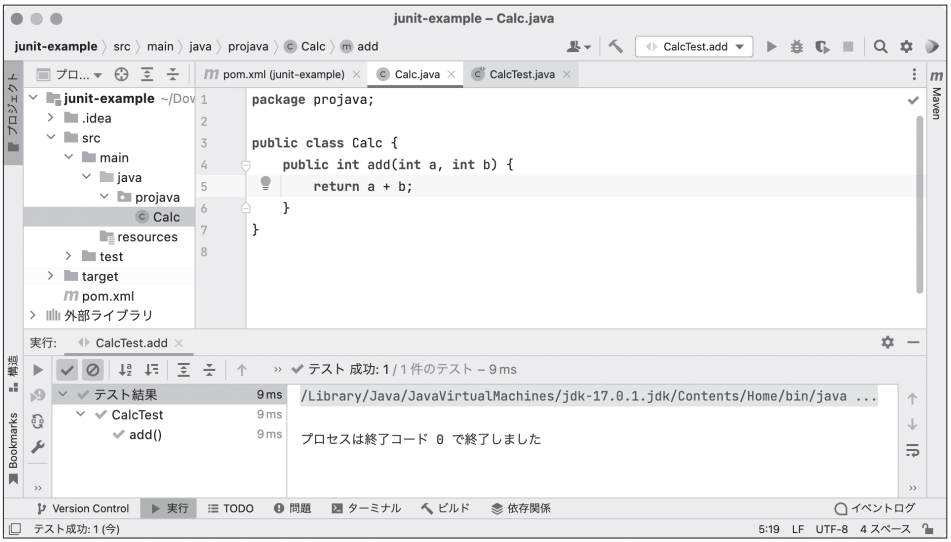

図18.8 ● テストが成功した状態

JUnitによる自動テストの実装、実行方法は以上のとおりです。テストクラス、テストメソッドはいくつあってもかまいません。多くの場合、テストクラスはテスト対象のクラスと対になる形で「テスト対象クラス名Test」という名前で作成します。テストメソッドはいくつ実装してもよいので、テスト対象メソッド1つに対して複数実装してかまいません。確認する項目が多い場合や、確認する内容・性質が異なる場合は積極的にテストメソッドを分けましょう。テストメソッドを分けたほうが、実装中やデバッグ中に「何が動作して何が動作していないのか」がわかりやすくなります。

18.1.3 その他のアサーションメソッド

前項では実行結果が期待した値と等しいことを確認するassertEqualsメソッドを紹介しました。アサーションメソッドの種類はたくさんあるのですが、**表18.1**に挙げる代表的なアサーションメソッドを覚えておけば十分にテストを書くことができます。

表18.1 ● 代表的なアサーションメソッド

アサーションメソッド	確認内容
assertEquals(期待値, 式 [, メッセージ])	式が期待値と一致することを確認する
assertNotEquals(期待値, 式 [, メッセージ])	式が期待値と異なることを確認する
assertTrue(式 [, メッセージ])	式がtrueを返すことを確認する
assertFalse(式 [, メッセージ])	式がfalseを返すことを確認する
assertNull(式 [, メッセージ])	式がnullを返すことを確認する

ユーザーインタフェースには次のように［削除］ボタンが追加されました（図22.14）。

図22.14 ● タスクの削除機能を追加した例

削除したいタスクの［削除］ボタンをクリックすると、サーバー側にリクエストが送られて、対象のタスク情報が削除されます。

22.3.7 タスク情報の更新機能を追加する

続いて、すでに登録されているタスク情報を更新できるようにしてみましょう。

TaskListDaoクラスにタスク情報を更新するメソッドを追加

削除機能のときと同じように、まずはTaskListDaoクラスに、指定したレコードの登録内容を更新するためのupdateメソッドを追加します。

Java
```
public int update(TaskItem taskItem) {
    int number = jdbcTemplate.update(
            "UPDATE tasklist SET task = ?, deadline = ?, done = ? WHERE id = ?",
            taskItem.task(),
            taskItem.deadline(),
            taskItem.done(),
            taskItem.id());
    return number;
}
```

updateメソッドは、引数に更新したいタスク情報をTaskItemオブジェクトとして受け取ります。そして、deleteメソッドと同様にその内容を使ってSQLのUPDATE文を作成し、JdbcTemplateのupdateメソッドを使ってデータベースに発行します。戻り値は、updateメ

ソッドから返された更新されたレコードの行数です。

ちなみに、JdbcTemplateのupdateメソッドに渡すSQL文の中では、この例のように「?」を複数回使うこともできます。その場合、実際の値は第2引数以降で「?」の数に合わせて順番に渡します。これはちょうど、String.formatメソッドの%dなどと同様の使い方になっています。

コントローラにタスク情報を更新するエンドポイントを追加

HomeControllerクラスには、Webサイトから更新のリクエストを受け取るエンドポイントとしてupdateItemメソッドを追加します。

Java
```
@GetMapping("/update")
String updateItem(@RequestParam("id") String id,
                  @RequestParam("task") String task,
                  @RequestParam("deadline") String deadline,
                  @RequestParam("done") boolean done) {
    TaskItem taskItem = new TaskItem(id, task, deadline, done);
    dao.update(taskItem);
    return "redirect:/list";
}
```

ここでは、受け取ったパラメータをもとにしてTaskItemのオブジェクトを作成し、それを引数としてTaskListDaoのupdateメソッドを呼び出しています。addItemメソッドではタスクのIDは自動で新規に生成していましたが、今回は既存のタスク情報を更新したいので、IDも含めた4項目の値をすべてパラメータで受け取るようになっています。更新が完了したら「/list」にリダイレクトします。

タスク情報の更新リクエストを送れるようにHTMLテンプレートを修正

Webページ側ですが、削除機能は単に対象タスクのIDだけを送ればよかったのに対して、今回はタスクの内容や期限、完了の状態を入力できるようにしなければなりません。そこでhome.htmlには、まずbodyタグ内の最後（</body>の直前）にタスク情報を更新するための次のようなdiv要素を追加します。

HTML
```
<div id="updateDialog">
    <div class="task_form">
        <h2>タスクの更新</h2>
        <form action="/update">
            <input id="update_id" name="id" type="hidden" />
            <label>タスク</label>
            <input id="update_task" name="task" type="text" />
            <label>期限</label>
```