3-2 プロジェクトを使いこなす

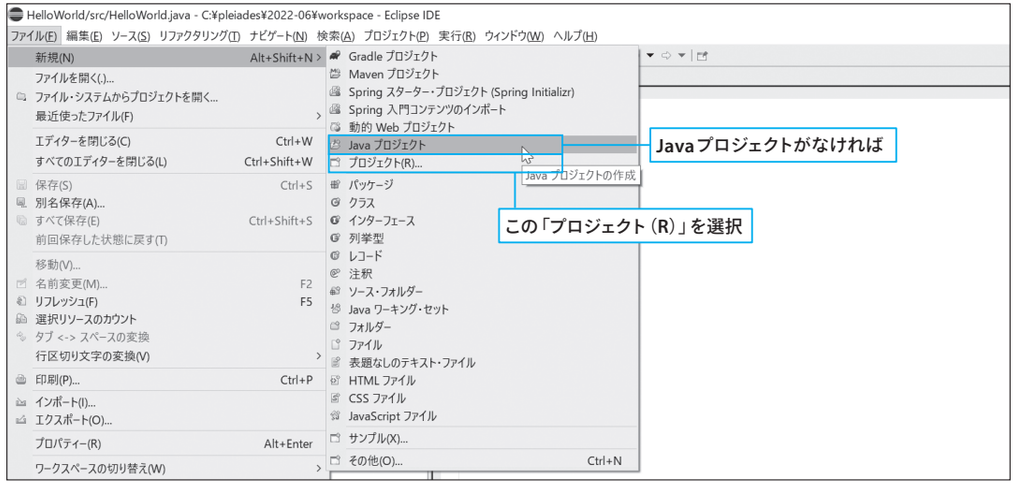
実際の開発の主体となるものが「プロジェクト」です。ここでは、基本的なプロジェクトの作成手順と、よく使われる機能やその設定について紹介します。

プロジェクトを作成する

それでは、基本的なJavaプロジェクトの作成を通して、プロジェクトの概要をみてみましょう。

プロジェクトを作成する場合は、Eclipseのメニューから「ファイル(F)」→「新規(N)」を選択します(**図3.17**)。

● **図3.17　Javaプロジェクトとプロジェクトメニュー**

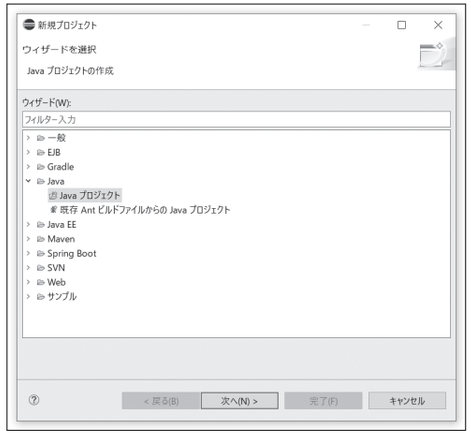


ONEPOINT

プロジェクトという言葉は、「企画」や「計画」を意味します。プロジェクトは通常、複数人のチームで遂行されますが、Eclipseのプロジェクトも、複数人のメンバーでプロジェクトを共有して開発を進めることが大半です。

図3.17で示したように、「新規(N)」メニューの先に「Javaプロジェクト」があればそのメニューを、見当たらない場合は「プロジェクト(R)」をクリックして、「新規プロジェクト」ダイアログボックスから「Java」内の「Javaプロジェクト」をクリックしてください(**図3.18**)。

● **図3.18　「新規プロジェクト」ダイアログボックス**



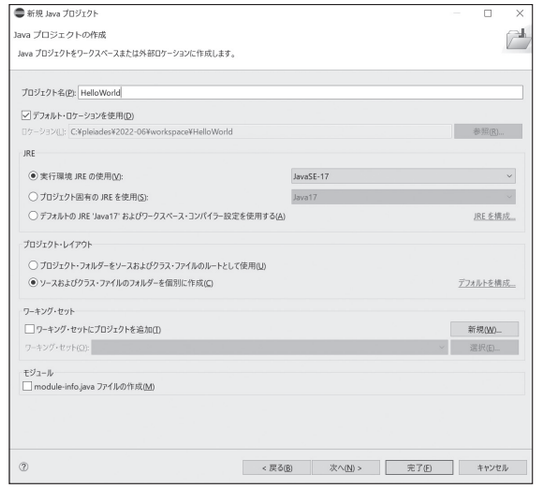
ONEPOINT

Eclipseのパッケージが公式サイトの「Java Developers」の場合は、「新規(N)」の先に「Javaプロジェクト」メニューがあります。「Java EE Developers」などの場合は、「プロジェクト(R)」から**図3.18**の「新規プロジェクト」ダイアログボックスを表示して、「Javaプロジェクト」を選択してください。

「新規Javaプロジェクト」ダイアログボックスが表示されたら、次の手順でJavaプロジェクトを作成してください(**図3.19**)。

1. 「Javaプロジェクト」ダイアログボックスにある「プロジェクト名(P)」欄に任意のプロジェクト名を入力する(ここではHelloWorld)。
2. 「デフォルト・ロケーションを使用(D)」にチェックが付いていることを確認する。
3. 「JRE」欄の「実行環境JREの使用(V)」と「プロジェクト・レイアウト」欄の「ソースおよびクラス・ファイルのフォルダー

● **図3.19　「新規Javaプロジェクト」ダイアログボックス**



プロジェクトの種類を知る

Javaプロジェクトについては前述してきましたが、ここでは本書でベースとしている「Pleiades All in One」にあるその他のプロジェクトについて触れておきましょう（**表3.2**）。

● **表3.2　主なプロジェクト**

プロジェクトの種類	説明
Gradleプロジェクト	JavaプロジェクトにビルドツールのGradleを追加したプロジェクト
Mavenプロジェクト	JavaプロジェクトにビルドツールのMavenを追加したプロジェクト
Springスターター・プロジェクト	Spring Frameworkをベースとした Webアプリケーションを手軽に作成するためのプロジェクト
Javaプロジェクト	Javaアプリケーション開発用のプロジェクト
動的Webプロジェクト	動的なWebコンテンツを作成するプロジェクト
プロジェクト	ディレクトリーとファイルの管理機能が主のシンプルなプロジェクト

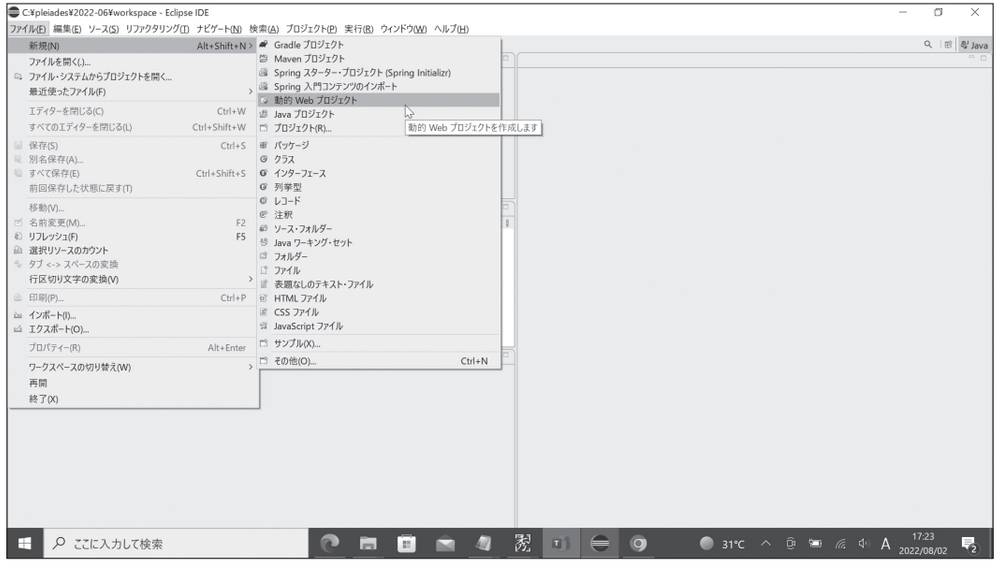
表3.2にあるプロジェクトには、Webアプリケーションに関するものがありました。本書がベースとする「Pleiades All in One Java Full Edition」には、デフォルトでこれらのプロジェクトが含まれますが、Eclipseの公式サイトでは、Eclipseのパッケージが「Java Developers」のものには含まれず、「Enterprise Java and Web Developers」に含まれます。

それでは、「Pleiades All in One」で、2種類のWebプロジェクトの作成手順を紹介しましょう。

以下は、Pleiades All in Oneでの「動的Webプロジェクト」の作成手順です。なお、本書では、SpringBootでのWebアプリケーション構築を主体にしているため、SpringBootでの手順は**第10章**で取り上げています。

1. Eclipseのメニューから、「ファイル（F）」→「新規（N）」を選択する。
2. 表示されたメニューから「プロジェクト（R）」を選択し、次に表示された新規プロジェクトダイアログボックス内の「ウィザード（W）」欄の「Web」から、「動的Webプロジェクト」を選択して、「次へ（N）」ボタンをクリックする（**図3.39**）。

● **図3.39　メニューから「動的Webプロジェクト」を選択する**



3　次の「新規動的Webプロジェクト」ダイアログボックスでは、「プロジェクト名（M）」欄に任意のプロジェクト名を入力し、「完了（F）」ボタンをクリックする（**図3.40**）。

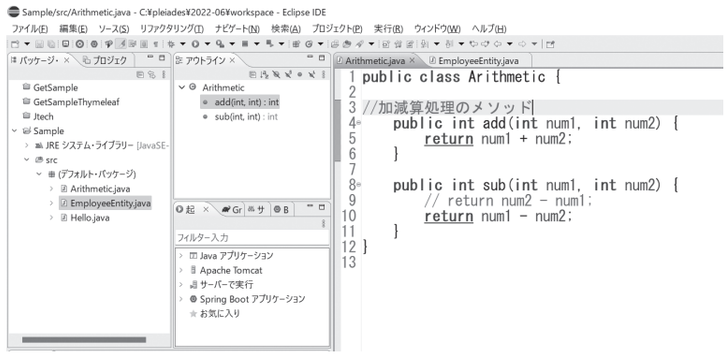
● **図3.40　プロジェクト名を入力する**

ズームイン・ズームアウト

Eclipseのメインメニューの「ウインドウ(W)」→「エディタ」→「ズームイン」、「ズームアウト」を使えば、エディター内の文字を大きくしたり、小さくしたりすることができます(**図5.16**)。なお、ズームイン、ズームアウトは、以下のショートカットキーを使うと便利です。

- ズームイン・・・Ctrl ＋ Shift ＋「＋(プラス)」／Ctrl ＋「＋(プラス)」
- ズームアウト・・・Ctrl ＋「－(マイナス)」

● **図5.16　ズームインでソースコードの文字を大きくした例**



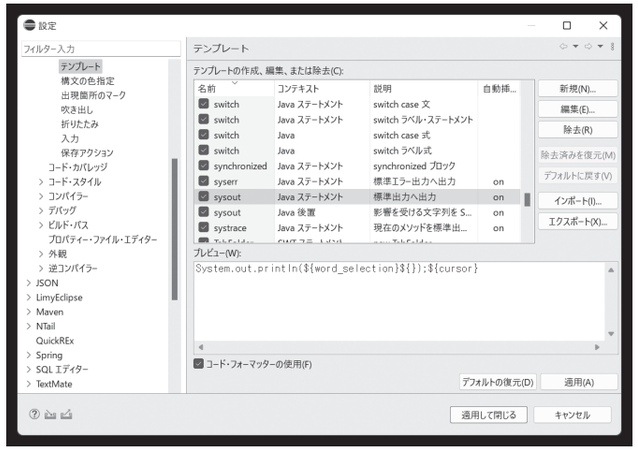
5-2 エディターのテンプレート機能を使いこなす

ソースコードの編集では、エディターに搭載されている機能をうまく使えば、効率よくソースコードが作成できるため、4章で紹介したデバッグ作業も楽になります。ここでは、コーディングに役立つエディターの機能を紹介していきましょう。

エディターで使える基本的なテンプレート機能

P.142では、Ctrl ＋ スペース キーでコード補完ができる「コードアシスト」について紹介しました。Javaエディターには、コードアシストの他にも、コード補完が可能な「テンプレート」と呼ばれる機能があります。Eclipseのメニューから「ウィンドウ(W)」→「設定(P)」で「設定」ダイアログボックスを表示させ、左側の項目から「Java」→「エディター」→「テンプレート」と進むと、**図5.17**に示す「テンプレート」が表示されます。

● **図5.17　Javaエディターの「テンプレート」**



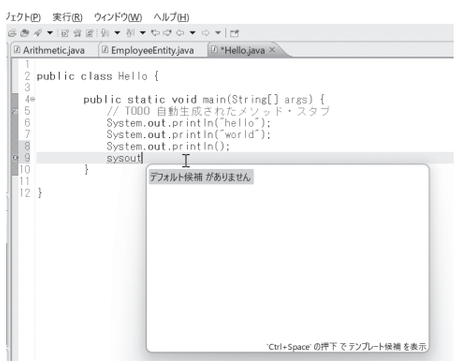
それでは、代表的なテンプレートをいくつか使ってみましょう。まずは、「sysout」です。**図5.17**で表示されている「sysout」をエディター上に入力し、Ctrl ＋ スペース を押下すれば、

```
System.out.println();
```

が生成されます。

なお、**図5.17**で示した画面からは、文字入力システムの単語登録やオフィスソフトのオートコンプリート機能などと同様に、新たなテンプレートを登録したり、テンプレートの編集や削除、無効化などが可能です。例えば、先のsysoutを無効にするには、テンプレートの左側のチェックを外してください。そうすると、エディター上で、sysoutと入力してCtrl ＋ スペース を押下しても何も生成されなくなります(**図5.18**)。

● **図5.18　sysoutを無効にした例**



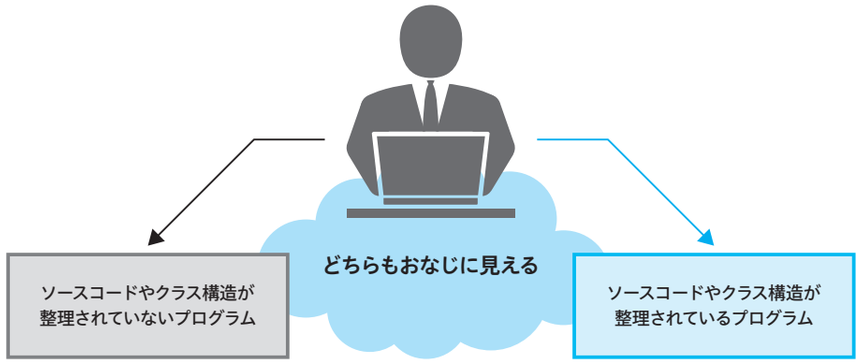
6-1 リファクタリングの目的

まずは、リファクタリングの必要性について考えながら、リファクタリングが、誰にどのようなメリットを及ぼすのかについて明確にしていきましょう。

なぜリファクタリングが必要なのか

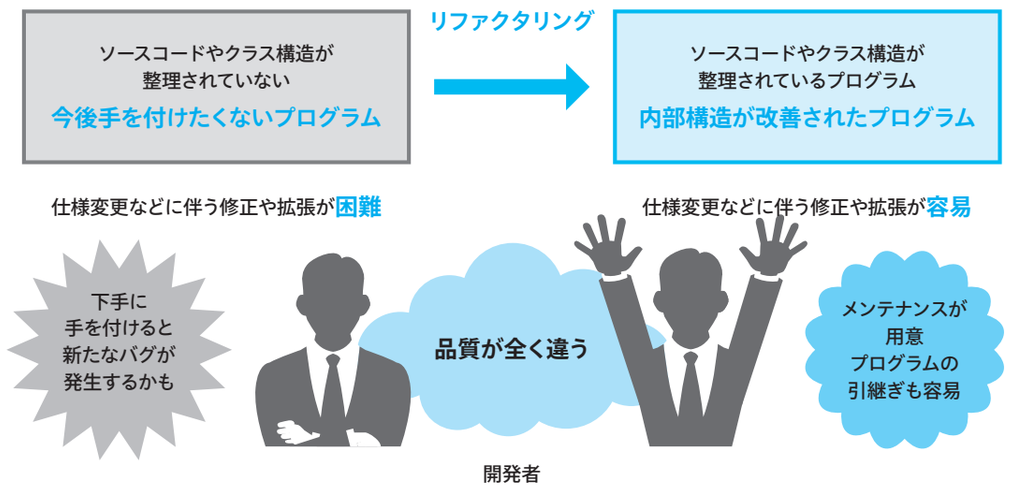
第4章でも述べたように、リファクタリング（refactoring）とは、現在動作しているプログラムの機能、仕様を保ちつつ、内部構造を見直すことです。しかし、リファクタリングは、新たな機能や操作を追加することではないため、システムの利用者（ユーザー）にとって表面的にわかるものではありません（**図6.1**）。

● 図6.1　リファクタリングはユーザーの目に見えない作業



開発者から見て、明らかにリファクタリングが必要なプログラムで作られたシステムをユーザーが利用していたとしても、プログラムの構造やソースプログラムは、ユーザーから見えないし、見えなくても業務に支障はないため、直接的な影響はありません。しかし、ユーザーの要望に伴う仕様変更によって、プログラムの改編が必要となった場合、内部構造に問題のあるプログラムは、下手に手を加えるとバグが発生する可能性もあり、拡張そのものが不可能となることもあり得るのです（**図6.2**）。

● 図6.2　リファクタリングしておかなければ、仕様変更に対応できない



家を建てる場合、子供の成長にあわせて、リビングを区切って子供部屋を作りやすく設計するなどといったことがあるようですが、同様に、システムでも将来の拡張に備えた内部構造を保っておくことは重要です。特に企業向けのシステムでは、システムを利用する企業（ユーザー）をとりまく環境の変化に伴い、仕様変更や機能拡張が起こる可能性が高いため、ユーザーからの仕様変更等の依頼に即座に対応するためにも、リファクタリングが必要となるのです。

リファクタリングの目的

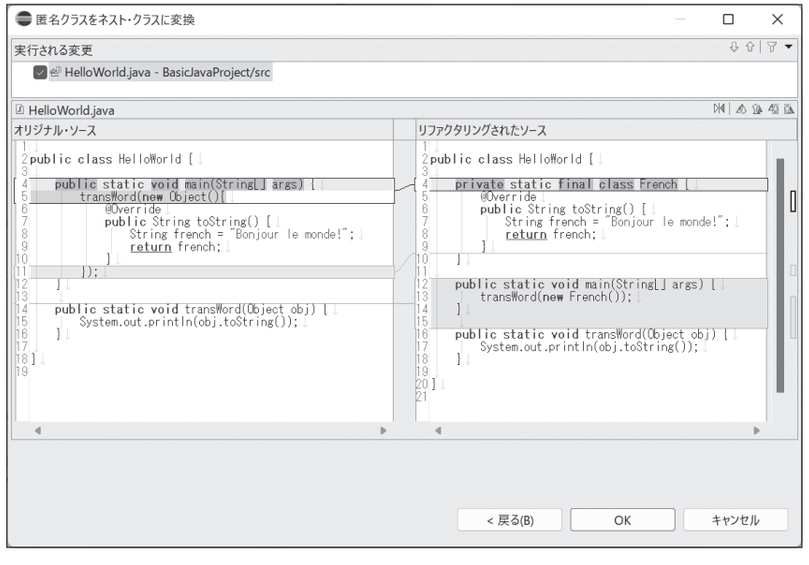
リファクタリングは、ユーザーの要望に迅速に対応するためだけでなく、プログラムの品質を保つために重要な作業です。以下に、開発者にとってのリファクタリングの目的についてあげてみましょう。

プログラムの品質を向上させる

メンテナンスが困難なプログラムに共通する点は、一般的に重複部分が多く、また、重複部分の記述が点在して、プログラムの構造が複雑になっていることが多いと言われています。さらに、重複部分が多ければ、ソースコードのボリュームも増えるため、読み解くには時間がかかり、生産性は著しく低下します。仮に現状のままのソースコードを読み解いて、修正や拡張を行えたとしても、ほとんどの場合、さらにメンテナンスが困難なプログラムと化してしまい、いずれは手を付けられないプログラムになってしまう可能性が高くなります。

もしリファクタリングを行って、プログラム制作の早期から、ソースコードの構造を改善し

● 図6.26　「匿名クラスをネスト・クラスに変換」のプレビュー画面



6-3 リファクタリングの実際

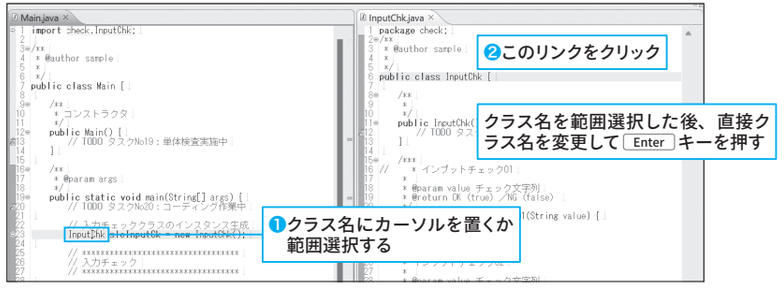
次は、前述したいくつかのリファクタリング機能を、異なるクラスを対象としたり、連続的に使用したりするなどといった、実践的な作業について紹介していきます。

異なるクラスにあるクラス名を変更する

リファクタリングによる「名前変更」の基本的な手順については、P.182で取り上げました。そこで、ここでは、「名前変更」が単なる文字列の置換ではない確定的な事例として、クラス名の変更を見ていくことにします。以下は、リファクタリングでクラス名「InputChk」を「InputCheck」に変更する手順です。

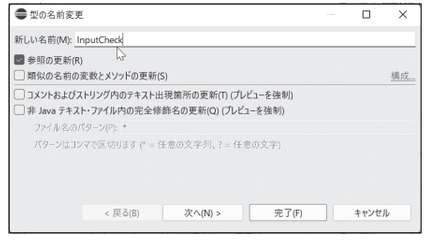
1. ソースプログラム「Main.java」内の「InputChk」クラス部分にカーソルを置くか、範囲選択する（**図6.27**）。
2. Eclipseのメニューから、「リファクタリング(T)」→「名前変更(N)」をクリックする。

● 図6.27　InputChkクラスを選択する

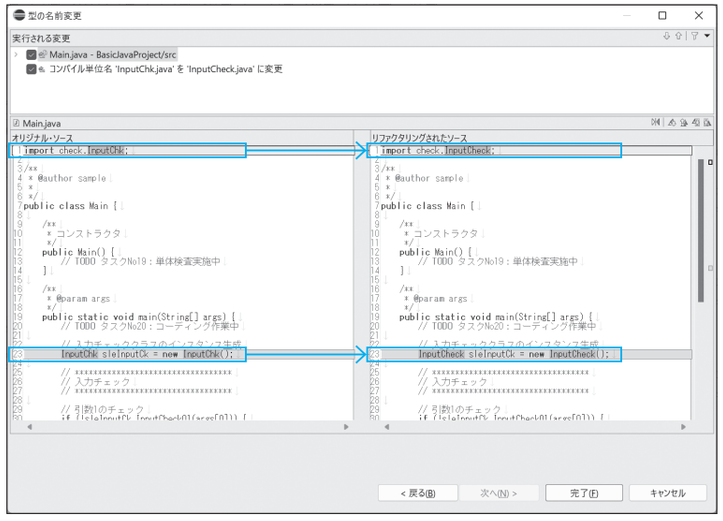


3. 「Enter」を押してリファクタリングします」の右側にある「オプション」のリンクをクリックする。
4. 「型の名前変更」ダイアログボックスが表示されたら、「新しい名前(M)」欄に変更後の「InputCheck」を入力して、「次へ(N)」ボタンをクリックする（**図6.28**）。
5. 「型の変更」ダイアログボックスでは、変更後の状態を確認して、「完了(F)」ボタンをクリックする（**図6.29**）。

● 図6.28　「型の名前変更」ダイアログボックス



● 図6.29　変更後の状態の確認



C2:条件網羅

全ての条件式の真偽をテストすればC2は100%となるため、以下の2パターンのいずれかとなる。

●パターン1:

- X=0が真とY=0が真のケース
- X=0が偽とY=0が偽のケース

●パターン2:

- X=0が真とY=0が偽のケース
- X=0が偽とY=0が真のケース

ONEPOINT

網羅条件のうち、分岐網羅（branch coverage）の具体例は、P.242で取り上げています。

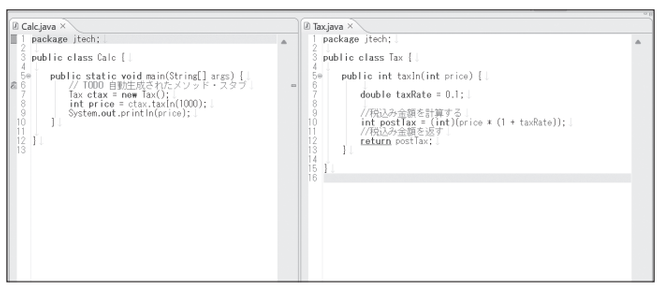
7-2 JUnitの設定と基本操作

テストの概要を理解したところで、次は、JUnitの基本的な使い方やテストケースの生成手順について紹介していきましょう

元のソースプログラム

まずは今回のテスト対象となるクラスをあげておきましょう。今回は、Taxクラスにあるtax Inメソッドをテストすることにします。なお、tax Inメソッドは、Calcクラスから呼び出されているという構成になっています（**図7.4**）。

● **図7.4　今回のテスト対象となるクラスと呼び出し元となるクラス**

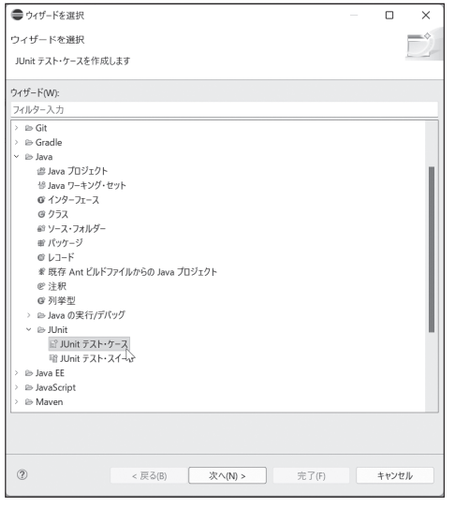


テストケースを作成する

それではテストケースを作成しましょう。

1 「パッケージ・エクスプローラー」内のTax.javaを右クリックして、ショートカットメニューから「新規（W）」→「その他（O）」からダイアログボックスを表示させ、「Java」の先にある「Junit」→「Junitテスト・ケース」を選択して「次へ（N）」をクリックする（**図7.5**）。

● **図7.5　ショートカットメニューから「Junitテスト・ケース」を選択する**

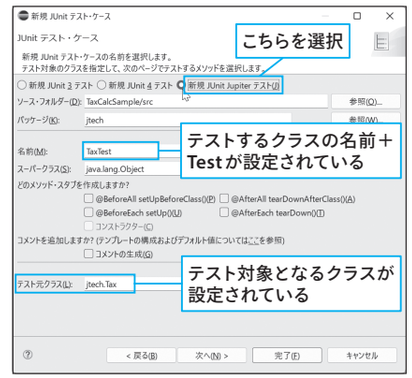


ONEPOINT

ショートカットメニューの「その他（O）」をクリックして、ウィザードのダイアログボックスから「Java」→「JUnit」→「Junitテスト・ケース」を選択してください。

2 「新規JUnitテスト・ケース」ダイアログボックスでは、「新規JUnit Jupiterテスト（J）」を選択し、「名前（M）」欄には、テスト対象となるクラスファイルの名前を入力（デフォルトのままでOK）して、「次へ（N）」ボタンをクリックする（**図7.6**）。

● **図7.6　「新規JUnitテスト・ケース」ダイアログボックス**



ONEPOINT

「JUnit Jupiter」がJUnit5に相当します。名前はデフォルトでテストしたいクラスファイルに「Test」という文字列を付けた名前（今回ならTax.java）の「Tax」に「Test」を付けた「TaxTest」になるため、問題なければ入力する必要はありません。

3 「テスト・メソッド」の画面では、テストするメソッドを選択してチェックを付け、「完了（F）」ボタンをクリックする（**図7.7**）。

ONEPOINT

BeforeEach、AfterEachは、JUnit4の@Before、@Afterに、BeforeAll、AfterAllは、@BeforeClass、@AfterClassに相当します。

JUnit5のアノテーションを検証する

それでは、はじめに、前述のJUnit5で使用できる、代表的なアノテーションを使ったテスティングの構成について見てみましょう。ここでは、**リスト7.1**に示す「TryJunit.java」をテスト対象にします。

● **リスト7.1　テスト対象となる「TryJunit.java」**

```
package junit;

public class TryJunit {

    static {
        System.out.println("staticイニシャライザが呼ばれました!");
    }

    TryJunit() {
        System.out.println("コンストラクタが呼ばれました!");
    }

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

}
```

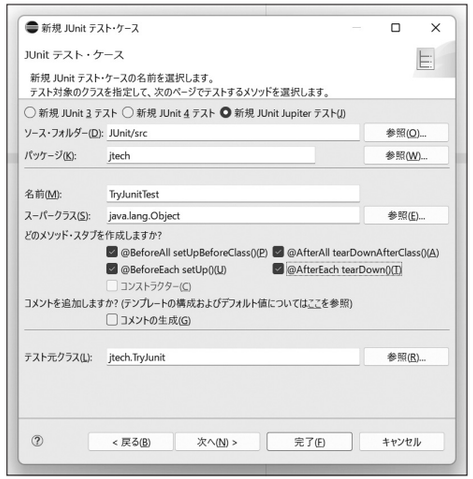
それでは、「TryJunit.java」のテストケースを作成します。P.218で紹介したように、「TryJunit.java」を右クリックして、「新規(W)」→「その他(O)」から「Java」→「JUnit」→「Junitテスト・ケース」を選択しますが、今回は、「Junitテストケース」ダイアログボックスの「どのメソッド・スタブを作成しますか？」にある4項目にチェックを付けて「次へ(N)」ボタンをクリックします(**図7.25**)。

ONEPOINT

スタブ(stub)とは、プログラムをテストする際に、呼び出されるモジュールの代用となる部品です。

次に表示される「テスト・メソッド」ダイアログボックスでは、対象となるクラス「TryJunit.java」にあるすべてのメソッドをチェックして、「完了(F)」ボタンをクリックしてください(**図7.26**)。

● **図7.25　「どのメソッド・スタブを作成しますか?」の項目にチェックを付ける**



● **図7.26　「テスト・メソッド」ダイアログボックス**

この後、「JUnit5がビルド・パス上にありません。追加しますか？」というメッセージが表示されたら、「OK」ボタンをクリックしてください。

これで、テストケース「TryJunitTest.java」が生成されます。**リスト7.2**に「TryJunitTest.java」の内容をあげておきましょう。

● **リスト7.2　生成されたテストケース「TryJunitTest.java」**

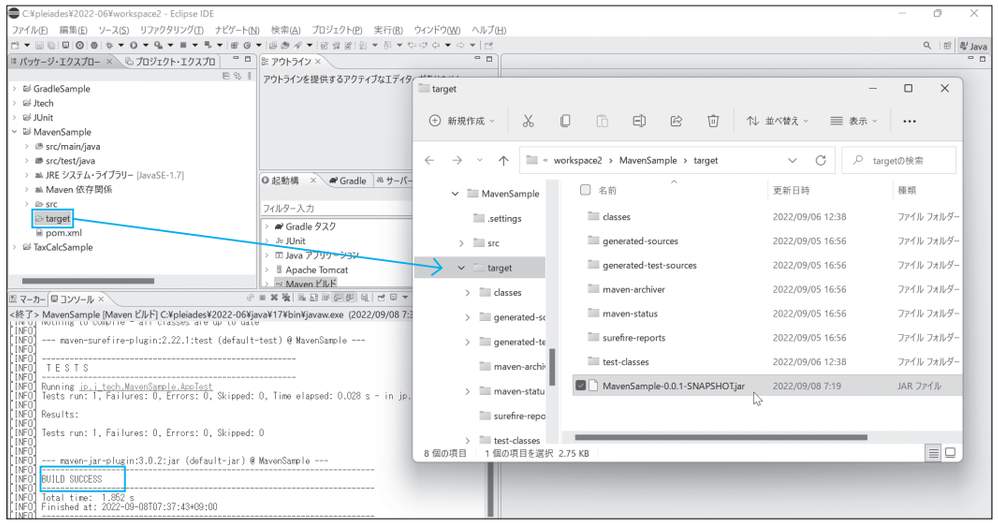
```
:
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class TryJunitTest {

    @BeforeAll
    static void setUpBeforeClass() throws Exception {
```

● 図8.13　「target」を右クリックして「Windowsエクスプローラー」を選択



●jarファイルを実行する（javaコマンドのパスが通っていない場合の例）

- javaコマンドの保存先・・・c:\pleiades\2022-09\java\7\bin
- jarファイルの保存先・・・C:\pleiades\2022-09\workspace\MavenSample\target

1　javaコマンドのある場所へ移動する。

```
cd c:\pleiades\2022-09\java\7\bin
```

2　絶対パス指定でjarファイルを指定して実行する。

```
java -classpath C:\pleiades\2022-09\workspace\MavenSample\target\MavenSample-0.0.1-SNAPSHOT.
jar jp.j_tech.MavenSample.App
```

これでjarファイルを実行することができます。なお、Eclipse JEEと同様に、「-jar」オプションで実行するためには、pom.xmlにマニュフェストの記述を追加する必要があります。P.260で示したように、pom.xml内の「maven-jar-plugin」の記述を変更すれば、実行可能なjarファイルとして利用可能となります。

8-2 Gradleの設定と基本操作

Eclipseで利用できるビルドツールの最新版がGradleです。ここでは、まずGradleをEclipseで使うための基本手順について見ていくことにしましょう。

Gradleの特徴

ビルドツールの草分け的存在のMakeやAntは「手続き型」で、前述のMavenとここで紹介するGradleは「規約型」のビルドツールであると言えます。「手続き型」では、ソースプログラムの場所やビルドしたファイルの出力先を逐次指定する必要がありますが、「規約型」では、「JARファイルを生成する」「Webアプリケーションをビルドする」といったような、プロジェクトの定義が明確で、あらかじめ決められたルールに則ってソースプログラムなどを配置していくため、処理がより簡潔になります。

以下に、Gradleの主な特徴をあげておきましょう。

- Groovyを利用
  AntやMavenのようにXMLでビルド処理を記述するのではなく、Groovyと呼ばれるJavaライクなスクリプト言語を使用する
- タスクによる処理
  タスクと呼ばれる「作業単位」で、ビルド処理を記述する
- Mavenのセントラルリポジトリに対応
  デフォルトでMavenのセントラルリポジトリに対応している（これまでデフォルトリポジトリだったJCenterは2022年2月で終了）

ONEPOINT

リポジトリ（repository）とは、貯蔵庫や倉庫を意味する単語で、ライブラリなどが一元管理されている場所を指します。

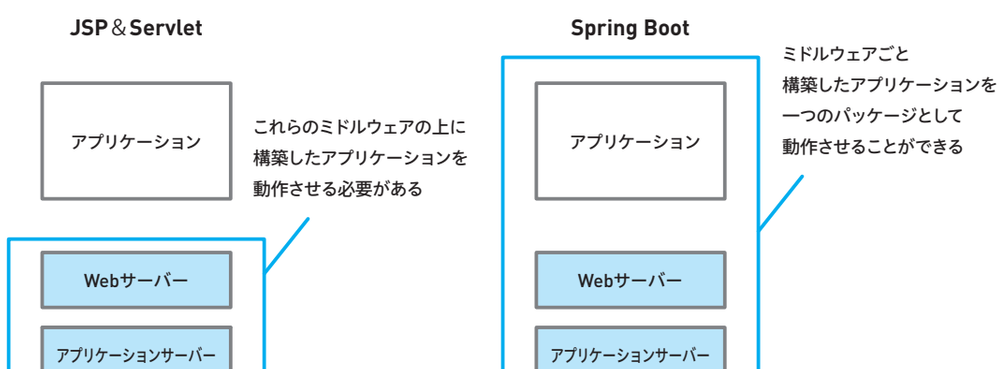
Gradleプロジェクトを作成する

Gradleによるビルドを行う場合、通常なら、先にGradleの公式サイトより、Gradleをインストールする必要があります（**図8.14**）。

JSPやServletでWebアプリケーションを構築する際には、Eclipseの公式サイトにあるEclipse JEEを入手して、まず「Tomcat」と呼ばれるアプリケーションサーバーの導入や設定等を行う必要がありました。

Spring Bootでは、構築するアプリケーションと、それらを動作させるためのTomcatなどのミドルウェアを一つのパッケージとしてまとめているため、開発環境も容易に構築することが可能です（**図10.3**）。

● **図10.3　Spring Bootを利用する場合**



ONEPOINT

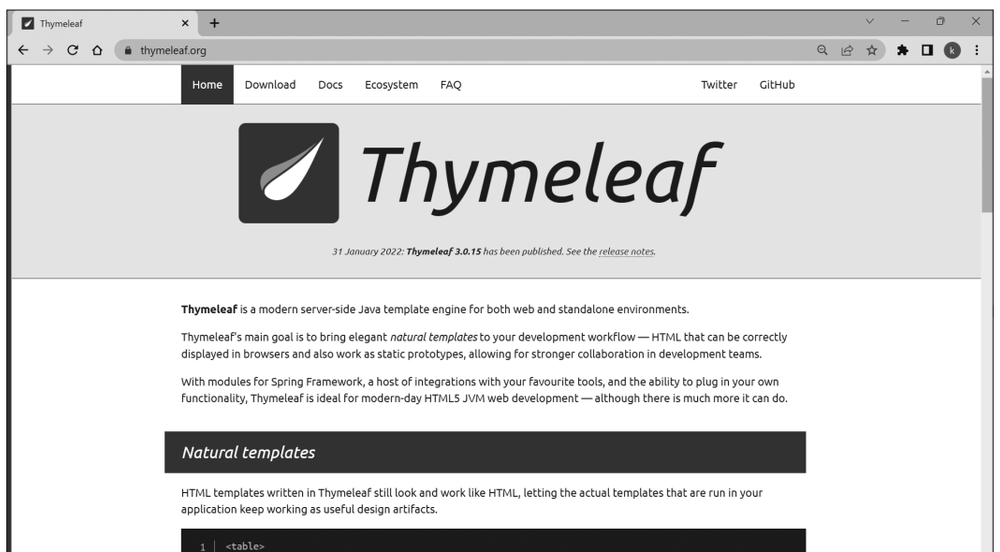
JSPやServletでWebアプリケーションを構築する場合も、Pleiades All in OneのJava Full Editionを使えば、Tomcatなどの構築に必要なものはデフォルトで用意されていましたが、設定は必要でした。

JSPに代わるThymeleafとは

JSP（Java Server Page）は、HTMLファイル内にJavaのコードを埋め込み、HTML形式の結果をクライアントへ返す技術として人気を博し、これまでの代表的なJavaテクノロジーとして普及しました。しかし、近年は「テンプレートエンジン」と呼ばれる、テンプレート（ひな形）とデータを合わせて文字列を出力する仕組みが主流であり、Spring Bootでは、「Thymeleaf（タイムリーフ）」というテンプレートエンジンの利用を推奨しています（**図10.4**）。

Thymeleafは、HTMLファイルを解析し、Webコンテンツを生成するテンプレートエンジンであり、Spring Bootには、Thymeleafがデフォルトで搭載されています。なお、Spring BootでThymeleafを使用する例はP.329で取り上げています。Thymeleafの基本文法については巻末付録を参照してください。

● **図10.4　Thymeleafの公式ページ（https://www.thymeleaf.org/）**



ONEPOINT

Spring Bootでは、JSPの使用に制限があり非推奨となっています。

Lombokとは

Lombok（ロンボック）は、エディターとビルドツールに自動的にプラグインして、Javaのソースコードを簡潔にするためのライブラリです（**図10.5**）。

Javaプログラミングでは、メンバー変数にアクセスするためのgetterやsetterメソッド、そしてequalsメソッド等に代表される、「ボイラープレートコード」と呼ばれるコードが存在します。

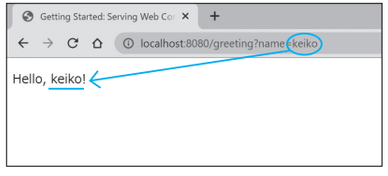
ボイラープレートコードは、定型的かつ冗長なコードなのですが、仕様上省略できません。しかし、Lombokを導入すればボイラープレートコードの実装を排除して、簡潔なコードを記述することが可能になります。

任意のパラメータ（文字列）を指定する場合は、以下のようにURLを入力します。

http://localhost:8080/greeting?name=任意のパラメータ（文字列）

文字列クエリを指定した場合は、**図10.26**のように、任意の文字列がgreeting.htmlの文字列と連結されて表示されます。

● **図10.26　文字列クエリを指定した実行結果**



10-2 Spring Bootでチーム開発

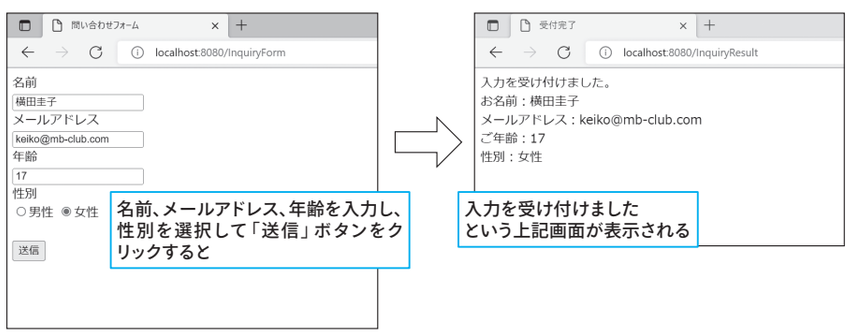
本書の最後に、Spring Bootを用いた基本的なWebアプリケーションの構築手順を取り上げますが、単なる構築手順ではなく、バージョン管理を行いながらチームで開発する形式で紹介していきます。

チーム開発を行うWebアプリケーションの概要

まずは、ここで取り上げるWebアプリケーションの概要について紹介しましょう。ここでは、フォームに氏名やメールアドレスなどを入力して送信するWebアプリケーションを構築することにします。先に完成例（**図10.27**）をあげておきましょう（最終的には結果表示に登録日時と番号を追加します）。

また、完成例（**図10.27**）のWebアプリケーションの概要は**図10.28**の通りです。

● **図10.27　Webアプリケーションの完成例**



● **図10.28　Webアプリケーションの概要**

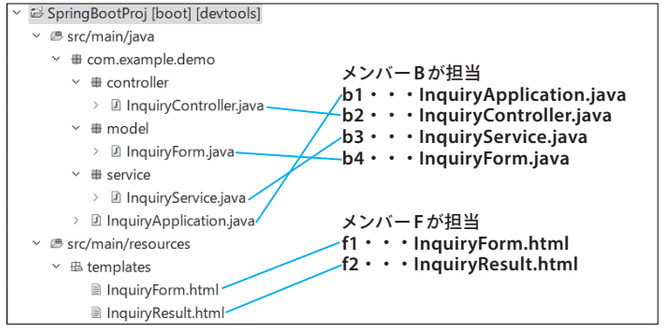
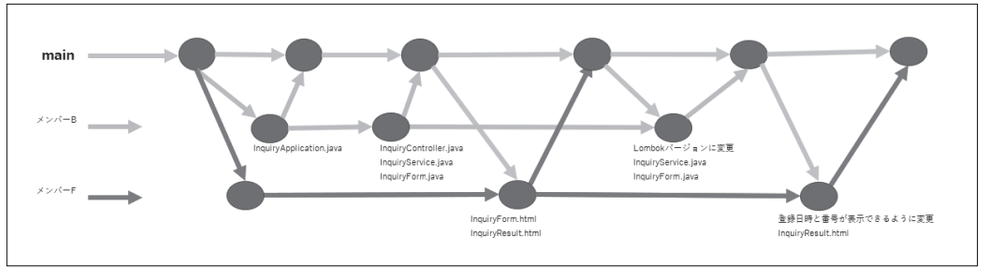


図10.28で示したように、f1、f2のフロントエンド処理をメンバーFが、b1～b4のバックエンド（サーバーサイド）処理をメンバーBが担当することにします。なお、このプロジェクトは、**第9章**で紹介したGitによるバージョン管理を行いながらチームで開発を進めていきます。担当ごとの作業の流れを、バージョン管理を主体にあげておきましょう（**図10.29**）。

● **図10.29　今回の作業の流れ**



1. メンバーB　b1をリファクタリングしてプッシュ
2. メンバーB　b2、b3、b4を作成してプッシュ
3. メンバーF　f1、f2を作成し、2をプルしてプッシュ
4. メンバーB　3をプルして、b3、b4をLombokバージョンに変更してプッシュ
5. メンバーF　4をプルして、f2に機能を追加してプッシュ