

🔗

🏠

🕒

📖

📄

🔍

✎

🔊

教材

🔍 検索

所属チーム ▼ 🔔¹ 👤

本文 目次 質問一覧 2件

ホーム 教材 Javaの基礎を学ぼう Javaの基本的な使い方を理解しよう

3章 Javaの基本的な使い方を理解しよう

Javaの基本的な書き方のルールとJavaプログラムの実行方法を学びます。

🕒 60分 🏆 - 未読

3.1 本章の目標

本章では以下を目標にして学習します。

- Javaの基本的な書き方のルールを知ること
- Javaプログラムの実行方法を知ること

本章からは、Javaでのプログラム開発を学んでいきます。まずJavaの基本的な書き方のルールを学んだうえで、[前章](#)で構築したJavaの開発環境でプログラムを実行します。

Javaのプログラムを自分で動かせるようになれば、実践しながらスムーズに文法を学べます。これから学習を加速できるように、本章でJavaのイロハを押さえましょう。

3.2 書き方の基本ルール

まずは以下のサンプルコードを見て、Javaの基本的な書き方のルールを知りましょう。これは「Hello World」と表示するだけの、最も初歩的なプログラムです。

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         // Hello Worldと表示
4         System.out.println("Hello World");
5     }
6 }
7
```

上記を踏まえて、Javaを書くときの基本ルールとして以下の4つを押さえておきましょう。

- 処理はクラスの中を書く
- main メソッドに中核となる処理を書く
- コメントを付ける方法は2種類
- 処理と処理はセミコロン (;) で区切る

順番に説明します。

+ 質問する



処理はクラスの中に書く

Javaプログラムにおける処理（命令）は、「**クラス**」の中に書いていくのが基本です。クラスとは、「**オブジェクト指向**」においてオブジェクトのもとになる「ひな型」のこと。

オブジェクト指向では、クラスから多様なモノ（オブジェクト）を作ります。たとえば「商品」クラスから、「シャンプー」や「コーヒー」などの商品オブジェクトを作れます。

クラスについて詳しくは[14章](#)で学びますが、形だけは知っておきましょう。クラスを定義して、その中に処理を書くのが基本です。冒頭のサンプルコードでは、最初と最後にある

```
1 public class HelloWorld {
2
3 }
4
```

の部分でクラス「HelloWorld」を定義し、中括弧 `{ }` ではさまれた部分に色々な処理を書いていますね。「public」は**アクセス修飾子**というものですが、詳しくは[5章](#)で学びます。

Javaのプログラミングでは、クラスが処理の入れ物となることを覚えておきましょう。

mainメソッドに中核となる処理を書く

Javaのプログラムには、必ず1つの「**mainメソッド**」が必要です。main メソッドは、Javaプログラムを実行したときに、処理が始まる「入り口」のようなもの。

main メソッドには、プログラムの中核となる処理を書きましょう。たとえば電卓プログラムを作るなら、計算を行うためのクラスを作り、それを main メソッドで呼び出します。

冒頭のサンプルコードでは、2行目・5行目にある

```
1 public static void main(String[] args) {
2
3 }
4
```

の部分が main メソッドです。今はこのように書くものと覚えておきましょう。「メソッド」については、[12章](#)で詳しく学びます。

main メソッドは、必ずクラス内に書きます。中括弧 `{ }` ではさまれた部分に処理を書くのは、クラスと同様です。

main メソッドから処理が始まることは、しっかり覚えておいてください。

コメントを付ける方法は2種類

Javaに限らず大半のプログラミング言語では、「**コメント**」というメモ書きをプログラムに残せます。コメントはプログラムの補足情報と見なされ、処理としては実行されません。

Javaのプログラムにコメントを付ける方法は、以下の2つです。



用途	コメントを付ける方法
1行だけコメント化したい場合	頭にスラッシュ2つ（//）を付ける
複数行をコメント化したい場合	<code>/* ~ */</code> ではさむ

頭にスラッシュ2つ（//）を付けると、**その行のスラッシュ以降がすべてコメント化**されます。冒頭のサンプルコードでは、3行目にある

```
1 // Hello Worldと表示
2
```

の部分がコメントです。「//」以降の「Hello Worldと表示」がコメントとして扱われます。

また、「/」と「/」ではさむと、**はさまれた部分すべてがコメント化**します。複数行にわたるコメントを付けたい場合は、こちらの方法で書きます。たとえば、

```
1 /* Hello
2   World
3   と表示 */
4
```

のように書くと、3行すべてがコメント化されます。

チーム開発では、自分の変更箇所をコメントで補足説明すると、他の人が読みやすくなります。また、一時的に無効化したい処理をコメント化（**コメントアウト**）するのも便利です。

コメントはJavaに限らずプログラミングで重宝するため、書き方を覚えておきましょう。

処理と処理はセミコロン（;）で区切る

Javaで処理と処理を区切るときは、セミコロン（;）を使います。冒頭のサンプルコードでは、4行目にある

```
1 System.out.println("Hello World");
2
```

の文末に、セミコロンがあります。セミコロンを付けることで、Javaのコンパイラが「ここで処理が終わるんだな」と処理の区切りを判断できるのです。

なお、**System.out.println(“メッセージ”);**と書くことで、任意のメッセージを表示できます。よく使うため覚えておきましょう。

通常は、1行の最後にセミコロンを1つ付ければOKです。

```
1 System.out.println("Hello World1");
2 System.out.println("Hello World2");
3 System.out.println("Hello World3");
4
```

ただし、以下のように**1行に複数のセミコロンが含まれるケース**もあります。





```
1 for(int i = 0; i < 100; i++) {  
2
```

上記のセミコロンで区切られた

- int i = 0
- i < 100
- i++

の3つは、それぞれ別の処理ということです。なお上記の文法は、同じ処理を繰り返す「for文」です。詳しくは9章で学ぶため、ここでは気にしなくて構いません。

行の最後でなくても、別の処理に切り替わるときにはセミコロンが必要です。セミコロンは、あくまで「処理と処理の区切り」である点に注意しましょう。

3.3 実際にJavaを使ってみよう

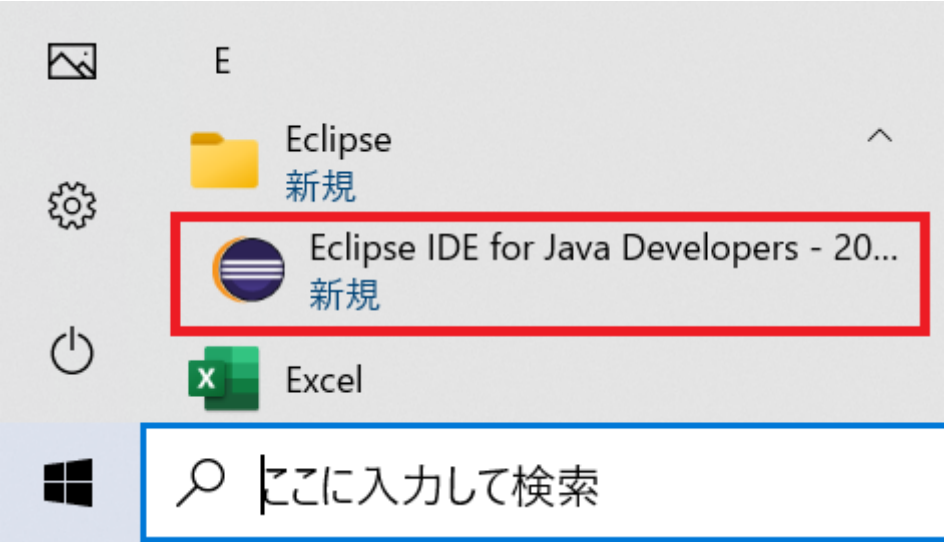
前節の基本ルールを踏まえて、実際にJavaを使ってみましょう。ここでは、前章でインストールした開発環境「Eclipse」を用いて、Javaのプログラムを動かす手順を学びます。

(1) Eclipseの起動

まずはEclipseを起動しましょう。

【Windowsの場合】

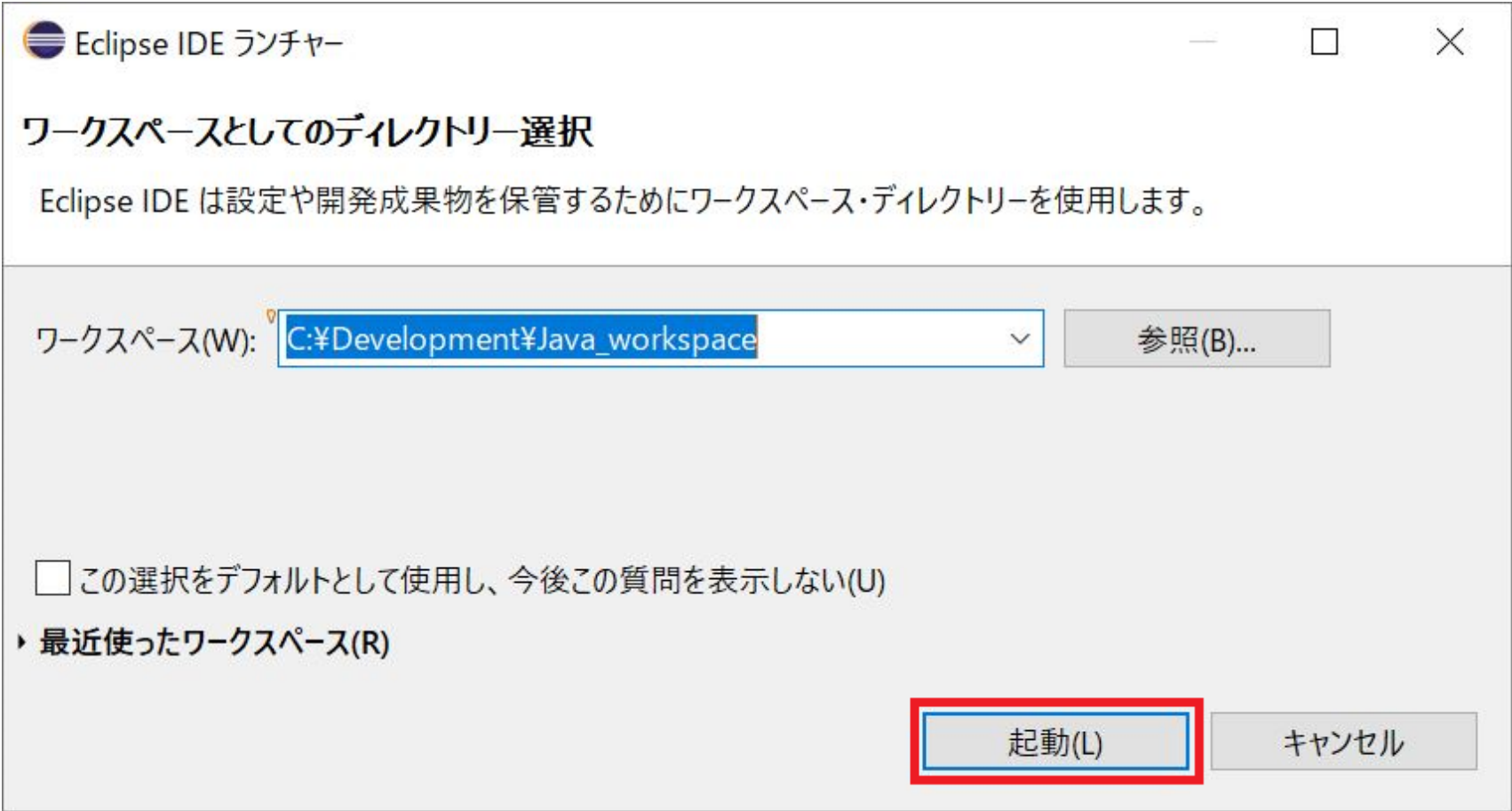
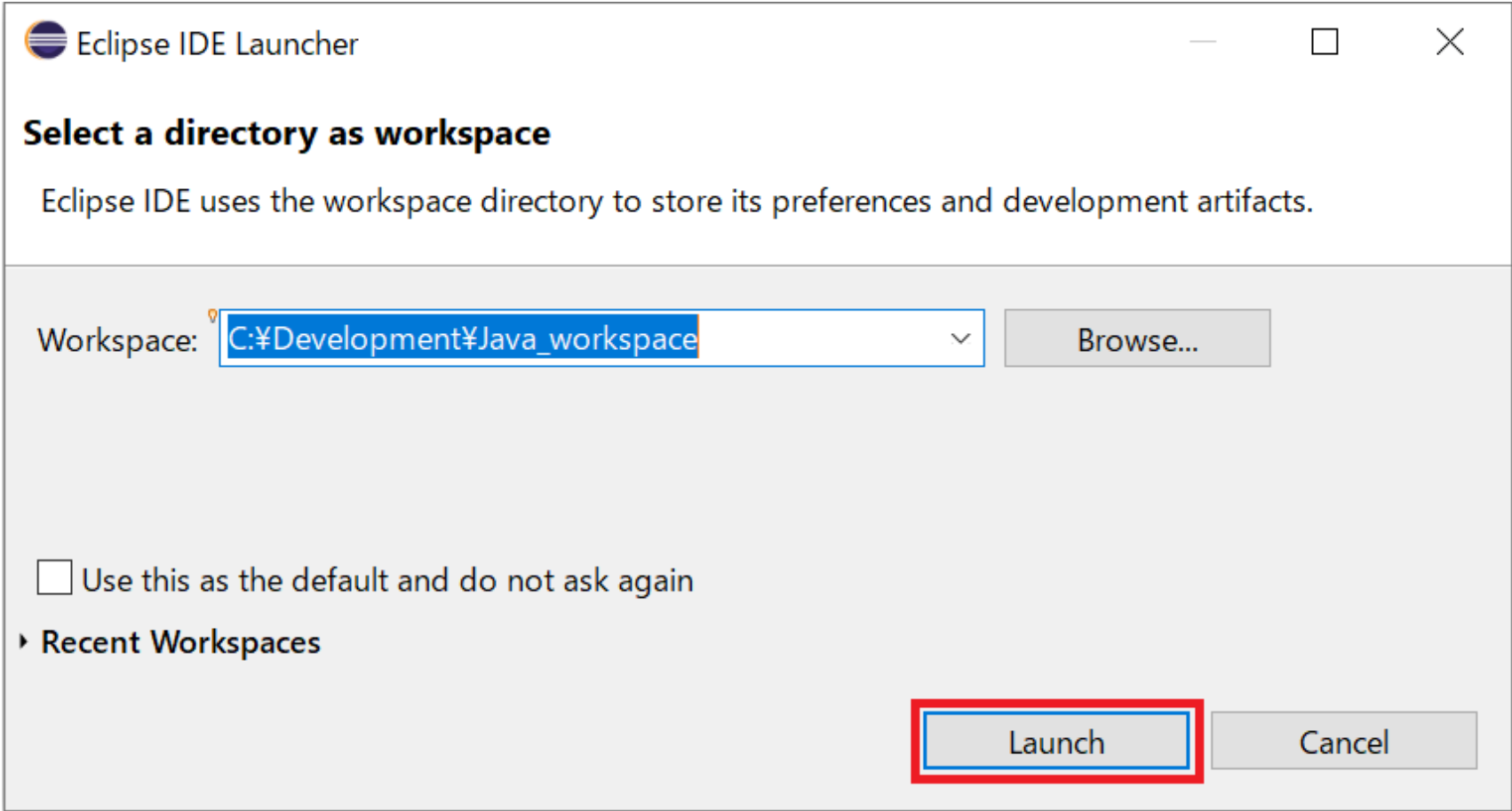
デスクトップのショートカットをダブルクリック、あるいはスタートメニューからEclipseをクリックしてください。



【Macの場合】

ファインダーで「アプリケーション」を選択し、アプリ一覧からEclipseをダブルクリックしましょう。

ワークスペース（開発するプログラムの管理場所）を聞かれた場合は、そのまま「起動」をクリックすればOKです。ワークスペースは、前章で作成したものを使いましょう。

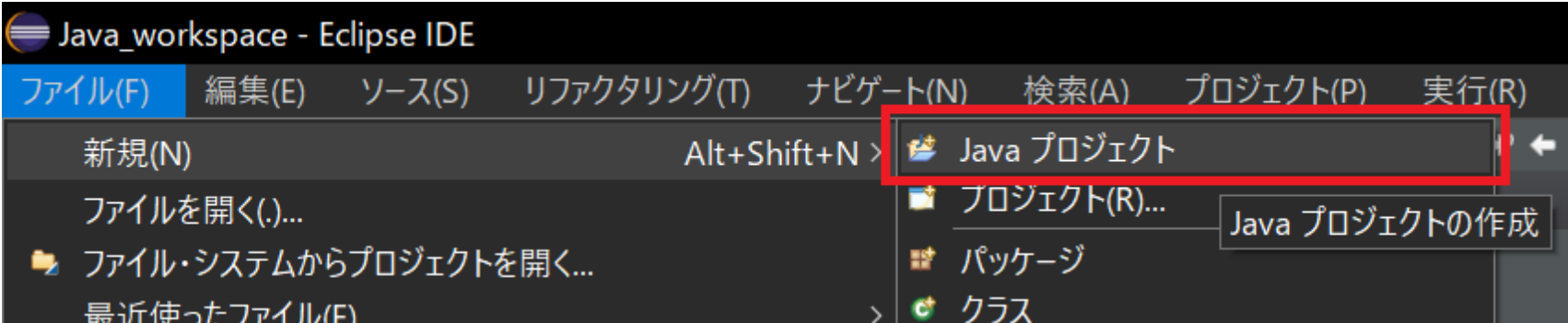


(2) プロジェクトの作成

Javaでプログラムの開発を始める場合、まず「プロジェクト」を作成します。プロジェクトとは、Eclipseで開発するアプリやシステムを管理する単位です。

たとえば、電卓アプリを作りたい場合は、電卓アプリ用のプロジェクトを作成します。新たに何かを作る場合は、新しいプロジェクトを作成するのが基本です。

Eclipse画面左上のメニューから、ファイル→新規→Javaプロジェクトを選択してください。





(3) プロジェクト情報の設定

プロジェクト情報を設定しましょう。各項目については、下表のとおりです。

項目名	説明
プロジェクト名	プロジェクトの名前。開発するものに合わせて自分で決める。
ロケーション	プロジェクト用のフォルダを作成する場所。基本的には、初期設定で問題ない。
JRE (Java実行環境)	どの実行環境を使うか。複数バージョンの実行環境があり、プロジェクトごとに使う環境を区別したい場合などに設定する。なお、JREにはJVM（Java仮想マシン）が含まれている。
プロジェクト・レイアウト	作成するプロジェクトのフォルダ構成。基本的には、初期設定で問題ない。
ワーキング・セット	作成するプロジェクトを、「ワーキングセット」に追加するか。ワーキングセットとは、ワークスペース内のプロジェクトをグループ化してまとめる機能のこと。
モジュール	「モジュール」の設定。モジュールとは、開発するプログラムをまとめる機能のこと。

最低限、設定が必要なのは**プロジェクト名**です。ほかの項目は、特別な理由がなければ初期設定でも問題ありません。プロジェクト名は、以下のルールに沿って指定しましょう。

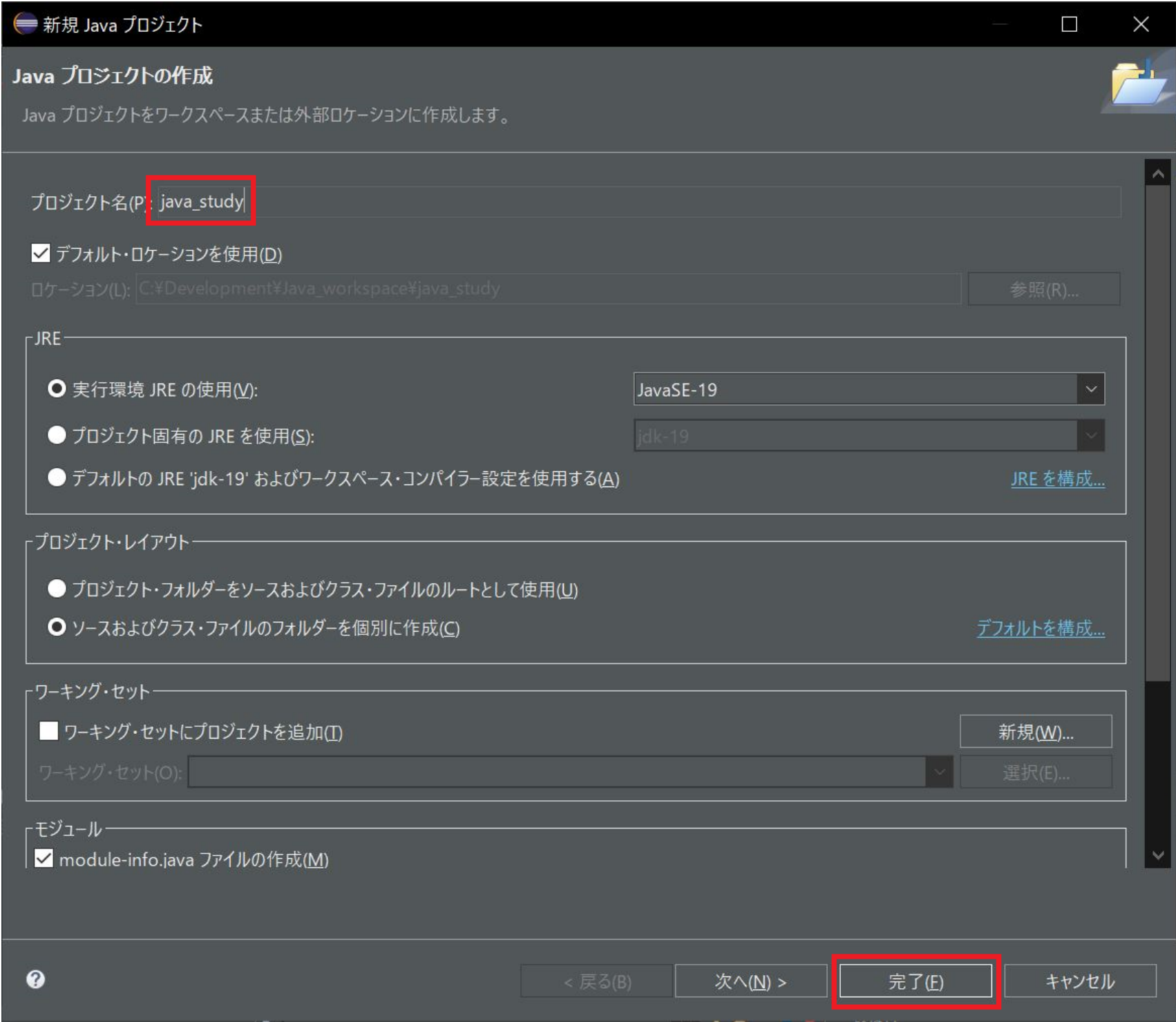
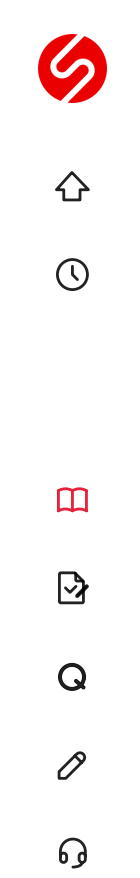
- 英文字・数字・アンダースコア（`_`）のみ使用可
- **先頭は英小文字を推奨**（英大文字は非推奨、数字は使用不可）
- 予約語（後述）と同名は不可

プロジェクト名は、先頭を英小文字とするのが慣例となっています。先頭が英大文字だとEclipse上でも警告されるため、英小文字から始まる名前を考えましょう。

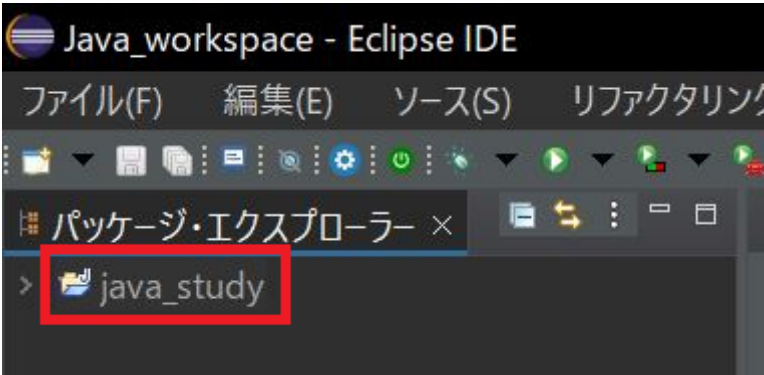
「**予約語**」とは、文法として特別な意味を持つキーワードのことです。たとえば、サンプルコードにある「`class`」はクラス定義用の予約語のため、プロジェクト名にできません。

今回は文法の勉強用として、「`java_study`」というプロジェクトを作成します。簡単に動かして文法を確認したいときは、このプロジェクトにプログラムを書けばOKです。

プロジェクト名を設定したら、「完了」をクリックしましょう。なお、「実行環境JREの使用」の部分はインストール方法によって変わる場合がありますが、そのまま構いません。



画面左部のパッケージ・エクスプローラーに「java_study」が追加されれば、プロジェクトの作成は完了です。

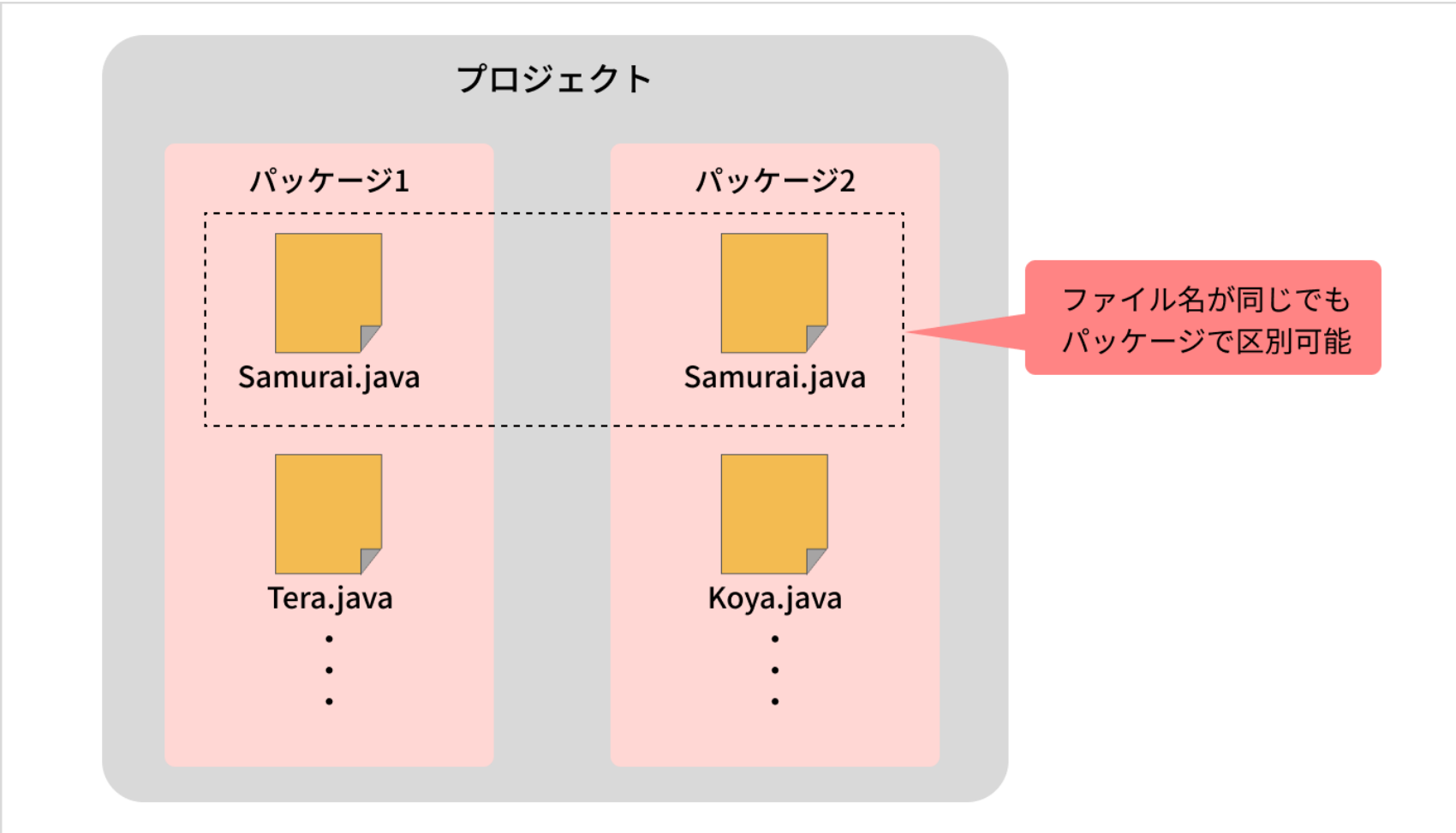


(4) パッケージの作成

プロジェクトを作成したら、「パッケージ」を作成しましょう。Javaにおけるパッケージとは、複数のソースファイル（プログラムが書かれたファイル）をまとめる機能のこと。

パッケージは、WindowsやMacでいう「フォルダ」のようなものです。さまざまなファイルをフォルダごとに整理すると、ファイルを管理しやすくなりますよね。

それと同様に、ソースファイルをパッケージでまとめると、機能や役割で区別して管理できます。また、同名ファイルが複数あっても、以下のようにパッケージで区別が可能です。



Javaのソースファイル名は、ファイル内で中核となるクラス名に合わせます。たとえば、クラス「Samurai」を定義するファイルの名前は「Samurai.java」となります。

Samurai.java ×

```
1 package syntaxbase;
2 ↓
3 public class Samurai {
4     public static void main(String[] args) {
5     ↓
6     |
7     ↓
8     }
9 ↓
10 }
11
```

名前を合わせる

ただし実際の開発だと、ときには同名ファイルを複数管理しなければなりません。たとえば、同じソフトウェアでも国内向けと海外向けで、仕様が変わるようなケースです。

もし Samurai.java が2つあるとすれば、そのままではJavaのコンパイラが区別できません。その点、上図のようにパッケージで分けられ

- パッケージ1のSamuraiクラス（Samurai.java）
- パッケージ2のSamuraiクラス（Samurai.java）

と、それぞれで異なるSamuraiクラスを定義できるのです。

前置きが長くなりましたが、パッケージの作成はパッケージエクスプローラーで行います。プロジェクト名を右クリックし、新規→パッケージを選択しましょう。



(5) パッケージ名の指定

パッケージ名の入力を求められるため、以下のルールに沿って指定しましょう。

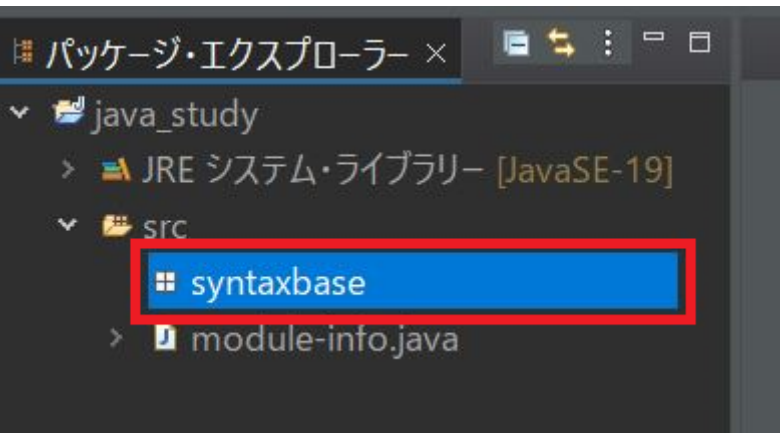
- 英小文字・数字・アンダースコア (_) のいずれかを組み合わせて使う **(英大文字は非推奨)**
- 先頭に数字は使用不可
- 予約語と同名は不可

英大文字は使用不可ではないものの、一般的には推奨されないため避けましょう。

今回は、「文法 (syntax) の基礎 (base) を学ぶ」という意味で「syntaxbase」というパッケージを作成します。「名前」にパッケージ名を指定し、「完了」をクリック。



すると、以下のようにsyntaxbaseパッケージが追加されます。

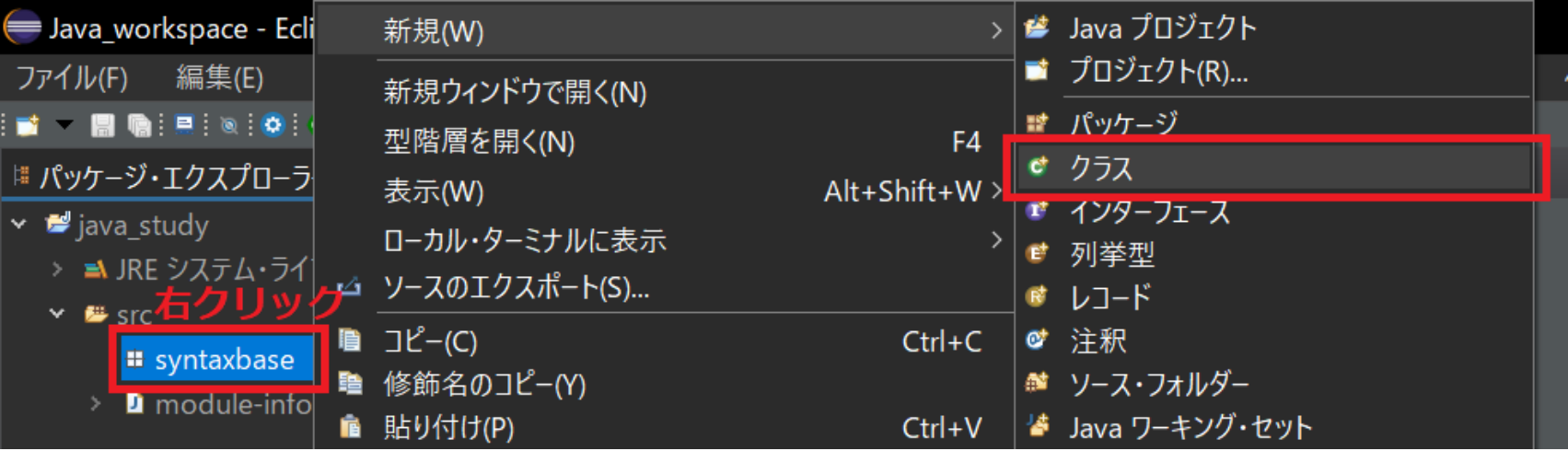




(6) ソースファイルの追加

作成したパッケージに、プログラムを書くためのソースファイルを追加しましょう。そして、前節で紹介した「Hello World」と表示するプログラムを書いていきます。

パッケージ名で右クリックし、新規→クラスを選択してください。



(7) クラス情報の設定

ソースファイルに追加するクラス情報を設定しましょう。オブジェクト指向のJavaでは、ソースファイル内にクラスが必須となるため、その設定が必要です。

ただし、大半はオブジェクト指向やクラスを学ばないと理解できない内容のため、細かい説明は割愛します。最低限、以下の2つを設定してください。

項目名	説明
名前	作成するクラス名を指定する。
public static void main(String[] args)	mainメソッドが必要な場合、チェックを入れる。

今回は前節のHelloWorldクラスを追加するため、名前は「HelloWorld」とします。ここで、クラス名を付けるときの基本ルールを押さえておきましょう。

- 英文字・数字・アンダースコア（`_`）のみ使用可
- 単語は先頭のみ英大文字、それ以降は英小文字（例：`hello`→`Hello`、`world`→`World`）
- 先頭に数字は使用不可
- 予約語と同名は不可

また、今回は `main` メソッドも追加するため、「`public static void main(String[] args)`」のチェックも入れてください。設定が済んだら、「完了」をクリックしましょう。



新規 Java クラス

Java クラス

新規 Java クラスを作成します。

ソース・フォルダー(D):

java_study/src

参照(O)...

パッケージ(K):

syntaxbase

参照(W)...

■ エンクロージング型(Y):

参照(W)...

名前(M):

HelloWorld

修飾子:

☐ public

☐ パッケージ(C)

☐ private

☐ protected

☐ abstract(I)

☐ final(L)

☐ static(C)

☒ なし(E)

☐ sealed(S)

☐ non-sealed(N)

☐ final(L)

スーパークラス(S):

java.lang.Object

参照(E)...

インターフェース(I):

追加(A)...

除去(R)

どのメソッド・スタブを作成しますか?

☒ public static void main(String[] args)(V)

☐ スーパークラスからのコンストラクター(U)

☒ 継承された抽象メソッド(H)

コメントを追加しますか? (テンプレートの構成およびデフォルト値については[ここ](#)を参照)

☐ コメントの生成(G)

?

完了(F)

キャンセル

すると以下のように、パッケージ内に「HelloWorld.java」が追加されます。その右側に表示されているのは、ソースファイルの内容です。最低限のソースコードは自動生成されます。

パッケージ・エクスプローラー ×

java_study

> JRE システム・ライブラリー [JavaSE-19]

> src

> syntaxbase

> HelloWorld.java

> module-info.java

HelloWorld.java ×

```
1 package syntaxbase;
2
3 public class HelloWorld {
4
5     public static void main(String[] args) {
6         // TODO 自動生成されたメソッド・スタブ
7
8     }
9
10 }
11
```

「クラス名に合わせる」という前述のルールにしたがい、ソースファイル名は**クラス名.java**となっています。クラス名・ファイル名が違くとエラーになるため注意してください。



なお、ソースコードの1行目にある

```
1 package syntaxbase;
2
```

は、HelloWorldクラスがsyntaxbaseパッケージに含まれることを明示しています。また、 HelloWorld.java のソースコードにある

```
1 // TODO 自動生成されたメソッド・スタブ
2
```

というコメントは、Eclipseでメソッドを自動生成したときに追加されるものです。このコメントは、不要なら削除してください。

なお「スタブ」は、プログラムの中身がちゃんと書かれていないメソッドのようなものです。自動生成された main メソッドの中身は空のため、スタブと表現されています。

(8) プログラムの記述

前節のサンプルコードにならって、「Hello World」と表示するプログラムを書いてみましょう。 main メソッドの中（中括弧ではさまれた部分）に、

```
1 // Hello Worldと表示
2 System.out.println("Hello World");
3
```

という処理を書けばOKです。ただし、上記ソースコードをコピーアンドペーストせずに自分で書くことが大切です。そのほうが、プログラムを書くスキルが定着しやすくなります。

またコピーアンドペーストだと、「l（小文字エル）」と「I(大文字アイ)」のような紛らわしい文字を誤認しても気付けません。しかし自分で書けば、勘違いにすぐ気付けます。

なお、ソースファイルを変更すると、ファイル名の頭に「*（アスタリスク）」が付きます。**変更を保存するときは、Windowsでは「Ctrl + S」キー、Macでは「command + S」キー**をそれぞれ押しましょう。



*HelloWorld.java ×

ファイルに変更あり

```
1 package syntaxbase;
2
3 public class HelloWorld {
4
5     public static void main(String[] args) {
6         // Hello Worldと表示
7         System.out.println("Hello World");
8     }
9
10 }
11
```

変更が済んだら保存する

以下のようにバツ印（×）と赤波線が表示される場合、ソースコードに問題があるため修正が必要です。この状態では、プログラムを実行しようとしてもエラーで失敗します。

```
3 public class HelloWorld {
4
5     public static void main(String[] args) {
6         // Hello Worldと表示
7         System.out.println("Hello World");
8     }
9
10 }
11
```

問題あり

問題箇所

上記ソースコードでは、抜けているセミコロンを追加すれば問題が解決します。

補足：Eclipseの補完機能を有効活用しよう

Eclipseには便利な「補完機能」があるため、有効活用しましょう。補完機能とは、部分的に入力したキーワードを、完全な形に補ってくれる機能のことです。

補完機能を使うときは、キーワードの入力途中で以下のキーを押してください。

使用環境	ショートカットキー
Windowsの場合	Ctrl + スペース
Macの場合	control + スペース

【注意】
Macで補完機能が使えない場合、「control + スペース」が別のショートカットキー（Spotlight検索など）に割り当てられています。その場合は、「control + スペース」を使っているショートカットキーの変更が必要です。
Macのショートカットキーを変更する方法は、以下の記事を参考にしてください。

Macのショートカットキー早見表92個 | 効かないときの対処法も紹介します | 侍エンジニアブログ (sejuku.net)

たとえば、「System」というキーワードを「Sys」まで入力したとします。



```
*HelloWorld.java ×
1 package syntaxbase;
2
3 public class HelloWorld {
4
5     public static void main(String[] args) {
6         // Hello Worldと表示
7         Sys
8     }
9
10 }
11
```

ここで上記のキーを押すと、以下のように「Sys」で始まる入力候補が表示されます。これが補完機能です。入力したい「System」をダブルクリックしましょう。

```
*HelloWorld.java ×
1 package syntaxbase;
2
3 public class HelloWorld {
4
5     public static void main(String[] args) {
6         // Hello Worldと表示
7         Sys
8     }
9
10 }
11
```

System - java.lang

SysexMessage - javax.sound.midi

SystemColor - java.awt

SystemEventListener - java.awt.desktop

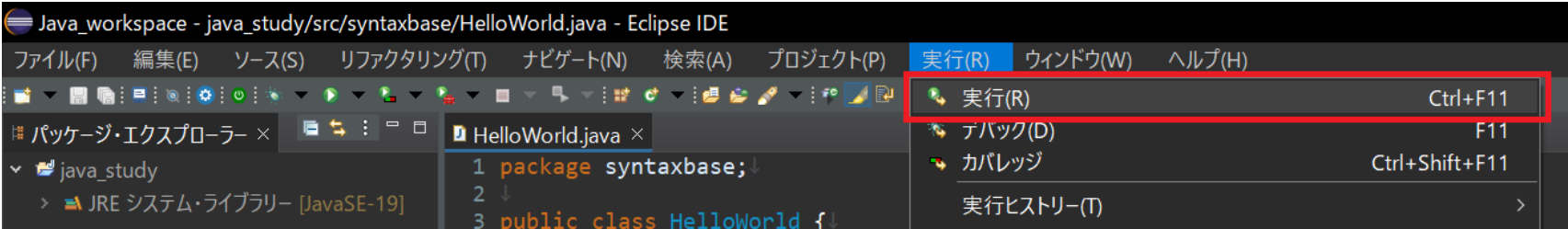
すると以下のように、選択した「System」が入力されました。

```
*HelloWorld.java ×
1 package syntaxbase;
2
3 public class HelloWorld {
4
5     public static void main(String[] args) {
6         // Hello Worldと表示
7         System
8     }
9
10 }
11
```

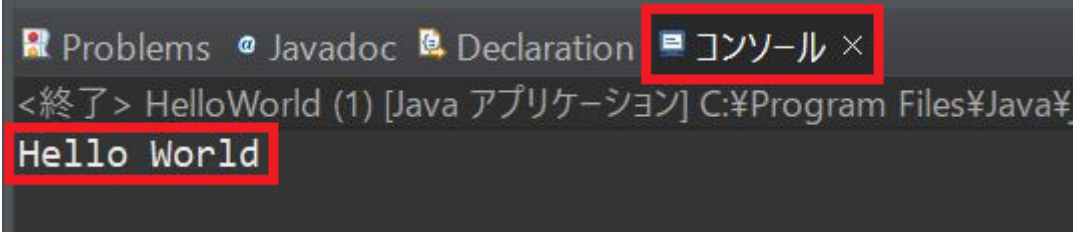
このように、補完機能を使うことでプログラミングをスピードアップでき、負担も減らせます。Eclipseのような統合開発環境を使うときは、積極的に活用しましょう。

(9) プログラムの実行

ソースファイルの変更が済んだら、プログラムを実行しましょう。Eclipse画面上部のメニューから、実行→実行を選択してください。



実行に成功すると、Eclipse画面下部の「**コンソール**」に**実行結果が表示**されます。「Hello World」と表示されていれば、実行は成功です。



2章では、Javaプログラムの実行前にコンパイル（変換処理）が必要だと学びましたね。しかし、**Eclipseの「実行」にはコンパイルも含まれる**ため、上記の操作だけで問題なく実行できます。

補足：日本語が文字化けする場合の対処法

Windows環境の場合、**日本語が文字化けして表示される場合**があります。文字化けとは、本来の文字とは異なる文字として扱われてしまうことです。

試しに、以下のように「こんにちは」と表示するプログラムを実行してみましょう。

HelloWorld.java

```
1 package syntaxbase;
2
3 public class HelloWorld {
4     public static void main(String[] args) {
5         // 日本語を表示できるか確認
6         System.out.println("こんにちは");
7     }
8 }
9
```

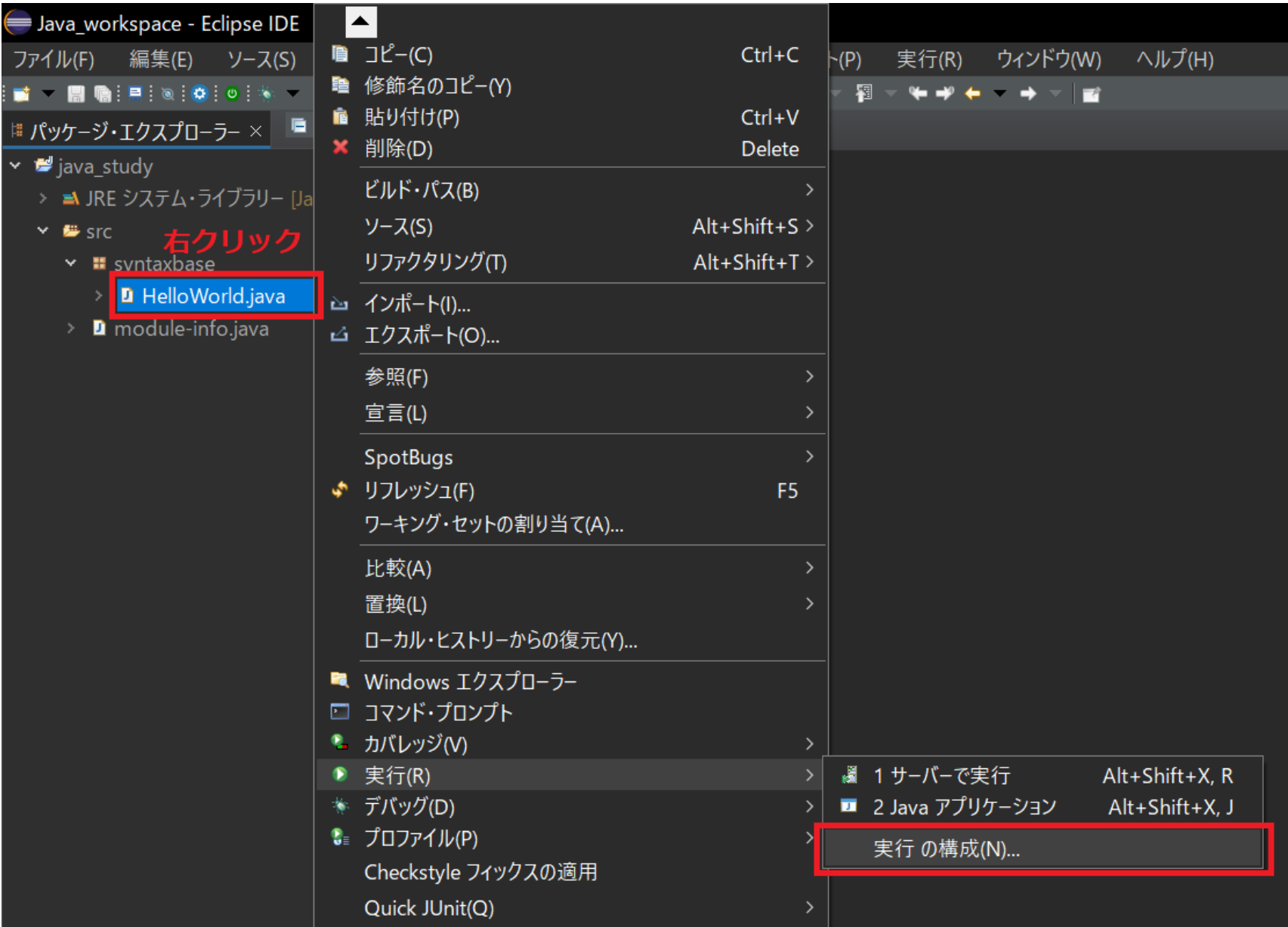
「こんにちは」と表示されるなら、何も問題ありません。しかし、以下のように文字化けして表示される場合は、追加の設定が必要です。



文字化けの原因は、エンコード方式（データ処理方法）がパソコンとEclipseで上手くかみ合わないためです。Eclipseの初期設定では「UTF-8」ですが、パソコンがこれに対応していないと、文字化けしてしまいます。

実行時に文字化けが発生するときは、以下の手順を実施してください。

1. 実行したいファイルを右クリックし、実行→実行の構成を選択します。



2. 「共通」タブの「エンコード」を「システム・エンコーディングを使用（windows-31j）」にします。windows-31j（MS932）は、Windowsの標準的なエンコード方式です。

エンコード方式を変更したら、「実行」をクリックしてください。

名前(N): HelloWorld (1)

メイン

引数

JRE

依存関係

ソース

環境

共通

プロトタイプ

保存

☐ ローカル・ファイル(O)

☒ 共用ファイル(H): ¥java_study参照(B)...

お気に入りのメニューに表示(R)

☒ デバッグ

☒ カバレッジ

☒ 実行

エンコード

☒ システム・エンコーディングを使用(S) (windows-31j)

☐ デフォルト - プロジェクトとワークスペースから継承(U) (UTF-8)

☐ その他(E) ISO-8859-1

標準入出力

☒ コンソールを割り当て(A) (入力に必要)

☐ 入力ファイル(E):

ワークスペース...ファイル・システム...変数...

☐ 出力ファイル(L):

ワークスペース(W)...ファイル・システム(S)...変数...

☐ 追加(P)

☐ 標準出力とエラー出力をマージする(M) (エラー出力の色付けを使用不可にする)

☒ バックグラウンドでの起動(K)

☒ 起動されたプロセスを終了する場合は子プロセスを終了する(I)

コマンド・ラインの表示(W)

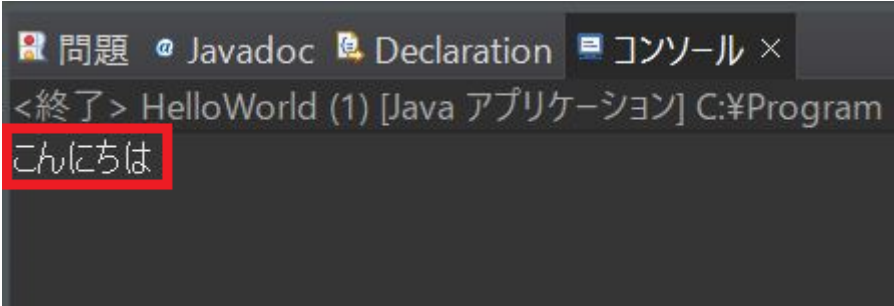
前回保存した状態に戻す(V)

適用(Y)

実行(R)

閉じる

3. エンコード方式を変更したうえでプログラムが実行されます。以下のように、文字化けせず「こんにちは」とコンソールに表示されれば、設定は完了です。



文字化けが発生したら、上記と同様の手順を行いましょう。

本章の学習は以上です。お疲れさまでした。

まとめ

本章では以下の内容を学習しました。

- 書き方の基本ルール
 - 処理はクラスの中を書く
 - クラス：プログラムに必要なモノを作るもとなる「ひな型」のこと
 - main メソッドに中核となる処理を書く
 - main メソッド：Javaプログラムの実行時に、処理が始まる場所
 - コメント（プログラムのメモ書き）を付ける方法は2種類
 - 1行だけコメント化したい場合：頭にスラッシュ2つ（//）を付ける



- 複数行をコメント化したい場合： `/* ~ */` ではさむ
- 処理と処理はセミコロン（`;`）で区切る
- 任意のメッセージを表示する場合→ `System.out.println(“メッセージ”);`
- EclipseでJavaプログラムを実行するまでの流れ
 - （1）Eclipseの起動
 - （2）プロジェクトの作成
 - （3）プロジェクト情報の設定
 - （4）パッケージの作成
 - （5）パッケージ名の指定
 - （6）ソースファイルの追加
 - （7）クラス情報の設定
 - （8）プログラムの記述
 - ソースファイルの保存
 - Windowsの場合：「Ctrl + S」キー
 - Macの場合：「command + S」キー
 - 補完機能の使用
 - Windowsの場合：入力途中で「Ctrl + スペース」キー
 - Macの場合：入力途中で「control + スペース」キー（場合によってはショートカットキーの設定が必要）
- （9）プログラムの実行

本章ではJavaの基本的な書き方から、プログラムの実行方法まで学びました。覚えることが多く大変だったと思います。お疲れ様でした。


プログラミング上達のコツは、積極的に実践を繰り返すことです。そのためには、Javaの基本的な使い方の理解が欠かせません。

始めは慣れないかもしれませんが、少しずつ自分の中にインプットしていきましょう。


次章では、Javaにおけるデータの扱い方について学びます。

理解度を選択して次に進みましょう


ボタンを押していただくと次の章に進むことができます



～50%



50～80%



80～100%

最後に確認テストを行いましょう

下のボタンを押すとテストが始まります。

教材をみなおす

テストをはじめる



前に戻る

3 / 31 ページ

次に進む

く 一覧に戻る

改善点のご指摘、誤字脱字、その他ご要望はこちらからご連絡ください。

© SAMURAI Inc. [利用規約](#) [法人会員利用規約](#) [プライバシーポリシー](#) [運営会社](#)