

オブジェクト指向言語 Java

博士(情報理工学) 小林 貴訓
博士(工学) Htoo Htoo 共著
工学博士 大澤 裕

コロナ社

まえがき

このテキストは、プログラム言語 Java について解説しています。Java はオブジェクト指向という考え方をベースとしてつくられた言語です。オブジェクト指向言語では、プログラムの部品化を容易に行え、再利用しやすくなります。最近の多くのプログラミング言語は、Java と同様にオブジェクト指向言語です。したがって、Java を習得すれば C# や Visual Basic .NET、C++ など、他のオブジェクト指向言語を学ぶときに、理解が容易になります。

プログラミングは難しい作業ではありません。一度基本を覚えてしまえば、使用する言語が変わっても、容易に順応できるようになります。しかし、最初のプログラミング言語を覚えるときには若干の障壁を感じるかもしれません。その障壁を乗り越えるためには訓練が必要です。プログラミングを初めて学ぶ人にとっての訓練とは、このテキストのような教材に載っているプログラムを自分で実際にタイプし、コンパイルし、実行してみるという、一見単純作業の積み重ねが重要と考えます。自分でタイプすることにより、慣用的な表現が記憶に残ります。また、プログラムのタイプミスが生じるかもしれません。その結果、コンパイル時にエラーが出力されます。もし、コンパイルエラーが生じたら新しいことを学ぶチャンスと考えてください。コンパイルエラーメッセージから原因を推測し、テキストのプログラムと見比べてみてください。エラーが発見できないときもあきらめず、友達と検討してください。最後には教員に聞き、必ず原因を理解してください。このような地道な訓練がプログラミング言語の基礎レベルの学修に非常に有益です。

Java をはじめ、多くのプログラミング言語のテキストにはソースファイルが CD や DVD で付録として付けられています。すでにプログラミングの経験がある人が新しい言語を学ぶときには、それらは有効と思いますが、ビギナーのうちには必ず自分でタイプすることが重要です。

大規模なプログラミングにおいては、Eclipse や NetBeans のような統合開発環境 (IDE) を使います。それらにより、タイプが容易になったり、タイプミスの発見が早くなり、またデバッグも容易になります。しかし、初学者のうちには、これらは使わず是非テキストエディタとコマンドプロンプトでプログラミングをしてください。統合開発環境はプログラミングの基礎を習得した後で使えば効率化の面でたいへん有益です。しかし初学者のうちには、考えて、調べて、聞くという一見非効率な作業が、習得には有益と考えます。

インターネット上には Java に関する情報があふれています。コンパイルエラーの意味や理

由がわからないときは，検索エンジンにコンパイルエラーメッセージをコピー，ペーストして尋ねることにより解決することもよくあります．プログラミング言語の習得に，インターネットも活用してみてください．

2016 年 9 月

大澤 裕

[†] 本書に掲載の Java プログラムソースコードは，コロナ社 Web ページの本書の紹介ページ <http://www.coronasha.co.jp/np/isbn/9784339028652/> にアーカイブデータをアップしております．必要に応じて，ダウンロードの上，ご活用ください．アーカイブデータ解凍の際のパスワードは以下のとおりです．

OBjJaVa028652CorOnA

また，各章末にある章末問題の解答も併せて上記 Web ページに用意しておりますので，必要に応じてご利用ください．

目 次

1. とりあえず Java を使ってみる

1.1 プログラミング言語 Java	1
1.2 簡単なプログラム例	3
1.3 本書の構成	5
章末問題	6
演習	6

2. Java の 基 礎

2.1 Java プログラムの構成	7
2.2 基本データ型と変数名	9
2.3 演 算 子	12
2.3.1 算術演算で用いられる演算子	12
2.3.2 関係演算子と論理演算子	13
2.3.3 ビット演算子	14
2.3.4 文字列と文字列結合演算子	14
2.3.5 その他の演算子と演算子の優先順位	16
2.4 配 列	17
2.5 制 御 構 造	19
2.5.1 条 件 分 岐	19
2.5.2 while 文と do-while 文	21
2.5.3 for 文	22
2.5.4 break 文と continue 文	23
2.5.5 コ メ ン ト	24
2.6 変数や定数の宣言とスコープ	24
2.7 データ型の変換	25

2.8 列 挙 型	26
2.9 メ ソ ッ ド	27
2.10 簡単な入出力	30
章 末 問 題	31

3. クラスとJavaプログラムの基本

3.1 Javaプログラムの基礎	33
3.2 クラスおよびオブジェクトとインスタンス	36
3.3 フィールドとメソッド	37
3.4 基本データ型とクラスオブジェクトとの違い	40
3.5 コンストラクタ	42
3.6 クラス変数	45
3.7 クラスメソッド	47
3.8 String クラス	50
3.9 ラッパークラス	52
章 末 問 題	54
演 習	56

4. クラスの拡張

4.1 クラス拡張の準備	58
4.2 クラスの拡張	60
4.3 クラス拡張における留意点	62
4.4 ポリモーフィズム	64
4.5 アクセス修飾	68
4.6 Object クラス	70
4.7 内部クラス	72
4.8 アノテーション	74
章 末 問 題	76
演 習	79

5. 抽象クラスとインタフェース

5.1 抽象クラスが必要になる状況	80
5.2 抽象クラス	82
5.3 インタフェース	83
5.4 final 修飾子による拡張の制限	90
5.5 総称型	90
5.6 総称型クラスの限定適用	92
5.7 匿名クラス	94
5.8 Lambda (ラムダ) 式	97
章末問題	99
演習	100

6. パッケージと例外処理

6.1 パッケージ	101
6.2 パッケージの作成	103
6.3 jar	105
6.4 例外処理	106
6.5 例外クラスの定義法	112
章末問題	114

7. GUI プログラム

7.1 JavaFX による簡単なプログラム	116
7.2 コントロールの配置	119
7.3 イベント処理の基本	121
7.4 レイアウトの方式	123
7.4.1 HBox	124
7.4.2 BorderPane	124
7.4.3 GridPane	126

7.4.4	FlowPane	127
7.4.5	Pane を組み合わせたレイアウト	127
7.5	色やフォントの設定	129
章 末 問 題		132
演 習		133

8. さまざまなコントロール

8.1	チェックボックス	135
8.2	ラジオボタン	137
8.3	テキストフィールド	139
8.4	テキストエリア	140
8.5	コンボボックス	141
8.6	プルダウンメニュー	143
8.7	ス ラ イ ダ	144
8.8	FXML	146
章 末 問 題		148
演 習		149

9. 図形の描画

9.1	Shape	150
9.2	GraphicsContext2D を用いた描画	151
9.3	マウスイベントの処理	154
9.4	キーボードイベントの処理	156
9.5	ウィンドウアプリケーションの実際	157
9.5.1	プログラムの動作	157
9.5.2	Polygonクラス	158
9.5.3	プログラムの説明	159
9.6	画像の表示	160
章 末 問 題		162
演 習		162

10. ファイルの入出力

10.1 基本的な入出力	163
10.2 Scanner による入力	166
10.3 PrintStream を用いた出力	169
10.4 ファイルに関する属性を知る	170
10.5 バイトストリーム	172
10.6 ランダムアクセスファイル	174
10.7 ファイルチューザ	177
章 末 問 題	179
演 習	180

11. クラスライブラリー

11.1 Math クラス	182
11.2 Arrays	184
11.3 時間と日付	185
11.4 コレクションクラス	187
11.5 コレクションクラスのインタフェース	188
11.6 コレクションクラスの例	191
11.6.1 ArrayList クラス	191
11.6.2 Stack クラス	193
11.6.3 HashMap クラス	194
11.6.4 PriorityQueue クラス	195
11.6.5 TreeSet	195
11.6.6 拡張 for 文とイテレータ	196
11.6.7 その他のコレクションクラス	197
11.7 Stream	197
章 末 問 題	199
演 習	200

12. マルチスレッド

12.1 Thread クラスによるマルチスレッドの実現	201
12.2 Runnable インタフェースによるマルチスレッドの実現	204
12.3 スレッドへの割込み	205
12.4 スレッドの終了を待つ	207
12.5 スレッド間の同期	209
12.6 スレッド間通信	211
章 末 問 題	214
付録 Java のドキュメント	215
索 引	218

前章までで扱ったプログラムは、コンソールに対して文字の入出力を行うものでした。このようなアプリケーションプログラムは CUI (console user interface) プログラムと呼ばれます。一方、本章から 9 章まででは GUI (graphical user interface) プログラムの作成法について述べます。まず、本章では GUI アプリケーションの基本について述べます。8 章では、GUI を構成する部品 (コントロールと呼ばれます) と、その使い方について述べます。9 章では、グラフィックスの表示法やマウスを用いた座標の取得法について述べます。

7.1 JavaFX による簡単なプログラム

GUI アプリケーションを作成するためには、GUI ライブラリーを使います。この GUI ライブラリーには、AWT、Swing、そして本書で扱う JavaFX などがあります。JavaFX は Java8 から標準のライブラリーとなりました。しかし、以前の Java では AWT と Swing がよく使われてきました。そこで現在広く使われているアプリケーションでも AWT や Swing が使われています、しかし今後作成されるアプリケーションは JavaFX で書かれるものが増えると考えられますので、本書では JavaFX について解説します。

最初に、JavaFX の機能を用いた GUI アプリケーションの例をプログラム 7.1 に示します。

[プログラム 7.1: ch7/WindowApp.java]

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.layout.StackPane;
4  import javafx.stage.Stage;
5
6  public class WindowApp extends Application {
7      @Override
8      public void start(Stage stage) throws Exception {
9          StackPane root = new StackPane();
10         Scene scene = new Scene(root, 300, 250);
11         stage.setTitle("WindowApp");
12         stage.setScene(scene);
13         stage.show();
14     }
15 }
```

```
16     public static void main(String[] args) {
17         launch(args);
18     }
19 }
```

このプログラムをつぎのようにコンパイルして、実行します。

```
[実行例]
% javac WindowApp.java
% javaw WindowApp
```

実行例の 2 行目で、いままでの java コマンドの代わりに javaw というコマンドを使っています。これは、いままでどおり java でもよいのですが、java コマンドの場合、プログラムが終了するまで、コマンドプロンプトが出されません。GUI プログラムでは、ユーザと対話しながら処理を行うため、プログラムが起動してからその終わりまでの時間が長いのが普通です。一方、java コマンドで起動した場合、プログラムが終了するまでコンソールが占領されてしまいます。javaw コマンドでプログラムを起動した場合には、コンソールにすぐにつぎのコマンドを受け付けるプロンプトが出されますので、GUI プログラムではこちらが好まれます。

このプログラムの表示例を図 7.1 に示します。ここに見られるように、つくられたウィンドウの中にはなにも入っていません。章が進むにつれ、コントロールと呼ばれる GUI 部品がウィンドウ内に配置されていきます。まずは、このなにもないウィンドウのつくられ方について説明します。

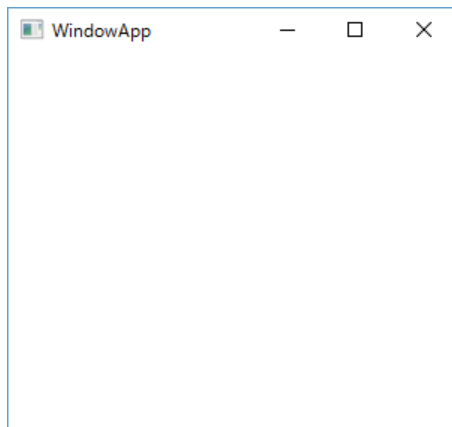


図 7.1 WindowApp の表示例

JavaFX を用いた GUI プログラムも、javaw コマンドに与えられたクラス（この例では WindowApp）の main メソッド（16 行目）から実行が始まります。しかし、main メソッドに書かれているのは launch(args) の 1 行のみです。このメソッドは、main と同じクラス内にある start メソッドを起動します。実質的には、JavaFX で書かれた GUI プログラムは、この start メソッドから実行が開始すると考えておけばよいでしょう。

プログラム 7.1 の 6 行目で、クラス定義の先頭に `public` 修飾子が付けられています。この `public` は必要です。理由は、Application クラスにおいて、このクラスを参照しているためです。

さて、プログラムに書かれたクラス `WindowApp` は `Application` クラスを拡張してつくられています。このクラスは抽象クラスであり、`start` メソッドの実装が要求されます。この `start` メソッドの中に複数の行が書かれていますが、ウィンドウを表示するために必要なのは、13 行目の `stage.show()` だけです。ために、`start` メソッドから `stage.show()` だけを残してすべて消してしまうと、大きななにもないウィンドウが表示されることになります。

残りの行を説明する前に、JavaFX の GUI プログラムの構成について簡単に説明しておきます。物理的に表示されるウィンドウに対応するのは、`Stage` クラス (`javafx.stage.Stage`) です。このクラスはトップレベルのコンテナと呼ばれます。コンテナとは、GUI を構成するさまざまな部品を入れる入れ物のことです。この `Stage` クラスは、ウィンドウのサイズや背景の色などを指定することができませんので、それらを行うために `Scene` (`javafx.scene.Scene`) というクラスを使います。さらに、この `Scene` 内にさまざまな部品を配置するために、`Pane` というクラスを用います。このクラスはレイアウト、またはコンテナとも呼ばれます。後で扱う `Button` や `Label` などの GUI 部品 (コントロール) は、`Pane` を用いて適切な位置に配置します。`Pane` には多数の種類があり、本章の後半で説明します。ここでは、単純な `StackPane` (`javafx.scene.layout.StackPane`) というクラスを用いています。

以上で述べた処理を、プログラム 7.1 に対応づけます。まず、9 行目は `StackPane` のオブジェクトをつくり、`root` という変数に代入しています。10 行目では、`Scene` クラスのオブジェクトをつくり変数 `scene` に代入しています。コンストラクタの最初の引数には、`StackPane` のオブジェクトを、つぎの二つの引数には、ウィンドウの横と縦のサイズを指定します。この指定では、横方向に 300 画素、縦方向に 250 画素の大きさのウィンドウが作られます。11 行目では `start` メソッドに引数として渡された `stage` オブジェクトに対して、タイトルバーに表示するメッセージを指定しています。ここで指定した文字列が、ウィンドウのタイトルバーに表示されます。12 行目では先ほどの `scene` を `stage` に登録しています。最後に 13 行目の `stage.show()` でウィンドウが画面に表示されます。

以上で述べたことを図 7.2 に示します。Stage がウィンドウの土台にあり、その上に Scene と Pane が順に配置されます。後で述べる `Button` や `Label` などの GUI 部品は `Pane` の上に配置されます。

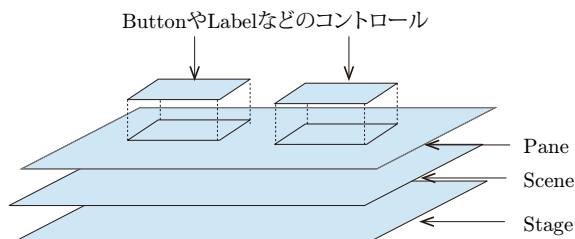


図 7.2 JavaFX の構成

7.3 イベント処理の基本

Java では、画面にボタンやプルダウンメニューなどのコントロールを配置した GUI を容易に作成することができます。例えばボタンでは、それがマウスでクリックされたときにイベントが発生し、そのイベントに対してあらかじめ登録されている処理が実行されます。このような処理を記述したものをイベントハンドラと呼びます。

Java でのイベント処理は、委任イベントモデル (delegation event model) という方法で行われます。イベントの種類には、マウスの移動やボタンの押下、キーボード上のキーの押下、GUI 上に配置されたボタンの選択、メニューの選択などたくさんの種類が考えられます。委任イベントモデルとは、イベントが発生したとき実行させたいメソッドをイベントを発生させるコントロールに登録しておき、実際のイベント発生により自動的にそれらのメソッド (イベントハンドラ) が呼び出される方式です (図 7.4)。

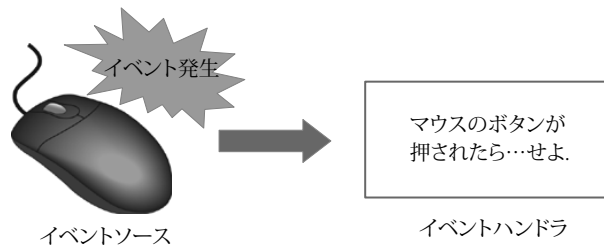


図 7.4 委任イベントモデル

ボタンやマウスのように、イベントを発生させるものをイベントソースと呼びます。そのイベントは、イベントの種類に応じて記述されるイベントハンドラと呼ばれるメソッドで処理されます。したがって、委任イベントモデルでは

1. 個々のイベントを処理するイベントハンドラを定義・宣言しておく
2. そのイベントソースにハンドラを登録する

という手順がとられます。

さて、プログラム 7.2 の例に戻ります。ボタンが押されたとき、`ActionEvent` というイベントが発生します。そこで、ボタンには `ActionEvent` に対する動作を記述したメソッドを登録することになります。これを行うためには、`EventHandler(javafx.event.EventHandler)` インタフェースの `handle` メソッドをオーバーライドします。ここで行う内容は、ラベルに対して `Hello World!` という文字列を設定することです。

プログラム 7.2 では匿名クラスを用いていますが、これを通常のクラスで書き換えるとプログラム 7.3 のようになります。

```

3  import javafx.event.EventHandler;
4  import javafx.scene.Scene;
5  import javafx.scene.control.Button;
6  import javafx.scene.control.Label;
7  import javafx.scene.layout.VBox;
8  import javafx.stage.Stage;
9
10 public class GUISample2 extends Application {
11
12     @Override
13     public void start(Stage stage) {
14         Label lb = new Label();
15         lb.setPrefWidth(250); lb.setPrefHeight(50);
16         Button btn = new Button();
17         btn.setPrefWidth(250); btn.setPrefHeight(50);
18         btn.setText("Click Me");
19         btn.setOnAction(event -> lb.setText("Hello World!"));
20         VBox root = new VBox();
21         root.getChildren().addAll(lb, btn);
22         Scene scene = new Scene(root, 250, 100);
23         stage.setTitle("GUISample");
24         stage.setScene(scene);
25         stage.show();
26     }
27 }

```

7.4 レイアウトの方式

JavaFX による GUI アプリケーションには、画面をつくる Stage があり、その Stage には Scene が登録されることを述べました。また、GUI に配置するコンポーネントは Pane というものを用いて配置し、それを Scene に登録することにより、画面上に表れることとなります。

さまざまなコンポーネントの配置はシーングラフ (scene graph) で管理されています。シーングラフは木構造をしています。つまり、一つの根ノード (root) をもち、根ノードと中間ノードには Pane が配置され、葉ノードにボタンなどのコントロールが配置されたグラフです。この木は次数は不定です。つまり、各ノードのもつ子ノードの数は定まっていません。言い換えると、各 Pane に登録されるコントロール (や別の Pane) の数を自由に決めることができます。

Pane への子ノードの登録は、プログラム 7.2 ではつぎのように行いました。

```

VBox root = new VBox();
root.getChildren().addAll(lb, btn);

```

VBox 型の変数名が root になっているのは、それがシーングラフの根ノードになっていることを示しています。どの Pane でも共通の操作は、Pane の `getChildren()` メソッドで子ノードの参照を得て、そこに `addAll` メソッドで引数に指定されているコントロールや別の Pane を登録することです。この操作を組み合わせることによりシーングラフがつけられます。

いままでの例で、プログラム 7.1 ではレイアウトとして StackPane を、プログラム 7.2 では VBox を使いました。以下では、よく使われるレイアウトについて説明します。ただし、プログラムの構造を単純なものにするためにコントロールにはすべて Button を用います。

7.4.1 HBox

HBox によるレイアウトは、コントロールを横一列に並べます。レイアウトに HBox を用いている他には、このプログラム 7.5 には特に新しいものはありません。11 行目から 13 行目で四つの Button をつくり、14 行目で HBox レイアウトをつくり、15 行目では四つのボタン（配列に入れられています）を登録しています。16 行目で Scene を作成する際に、大きさを指定していませんが、大きさが省略された場合には適当なサイズでウィンドウがつけられます。もちろん scene の大きさを指定してもかまいません。このプログラムの実行画面を図 7.5 に示します。

[プログラム 7.5: ch7/HBoxSample.java]

```

1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.scene.layout.HBox;
5  import javafx.stage.Stage;
6
7  public class HBoxSample extends Application {
8      @Override
9          public void start(Stage stage) {
10             stage.setTitle("HBoxSample");
11             Button[] button=new Button[4];
12             for(int i=0; i<4; i++)
13                 button[i]=new Button("button-"+i);
14             HBox root=new HBox();
15             root.getChildren().addAll(button);
16             stage.setScene(new Scene(root));
17             stage.show();
18         }
19     }

```

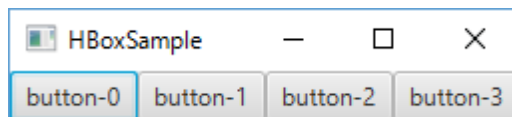


図 7.5 HBox を用いたレイアウト

7.4.2 BorderPane

ボーダーレイアウト (BorderLayout) は、コンポーネントを配置するコンテナ上で、上 (TOP)、下 (BOTTOM)、左 (LEFT)、右 (RIGHT)、および中央 (CENTER) の位置を指定して配置する方式です。ただし、これらの 5 種類をすべて指定する必要はなく、必要なもののみを指定します。ボーダーレイアウトを用いた簡単な例をプログラム 7.6 に示します。

[プログラム 7.6: ch7/BorderPaneSample.java]

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.control.Button;
4  import javafx.scene.layout.BorderPane;
5  import javafx.stage.Stage;
6
7  public class BorderPaneSample extends Application {
8      @Override
9      public void start(Stage stage) throws Exception {
10         Button buttonC=new Button("Center");
11         buttonC.setPrefWidth(200); buttonC.setPrefHeight(200);
12         Button buttonT=new Button("Top");
13         buttonT.setPrefWidth(300); buttonT.setPrefHeight(50);
14         Button buttonB=new Button("Bottom");
15         buttonB.setPrefWidth(300); buttonB.setPrefHeight(50);
16         Button buttonL=new Button("Left");
17         buttonL.setPrefWidth(50); buttonL.setPrefHeight(200);
18         Button buttonR=new Button("Right");
19         buttonR.setPrefWidth(50); buttonR.setPrefHeight(200);
20         BorderPane root=new BorderPane();
21         root.setCenter(buttonC);
22         root.setTop(buttonT);
23         root.setBottom(buttonB);
24         root.setLeft(buttonL);
25         root.setRight(buttonR);
26         Scene scene=new Scene(root,300,300);
27         stage.setTitle("BorderPaneSample");
28         stage.setScene(scene);
29         stage.show();
30     }
31 }
```

プログラム 7.6 の実行画面を図 7.6 に示します。

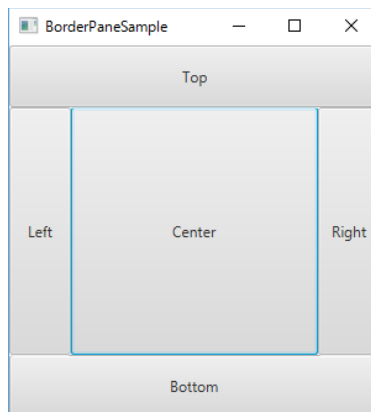


図 7.6 BorderPaneSample
の実行例

このプログラムでは、10 行目で Button のインスタンスをつくり、11 行目でそのボタンのサイズを指定しています。これを後 4 回繰り返しています。20 行目で BorderPane のインスタンスをついています。21 行目から 25 行目までは、作成された Button を BorderPane に表示位置を指定して登録しています。


```
i= 1, val= 9
i= 0, val= 10
```

10.7 ファイルチューザ

7章から10章で多数のJavaFXのコントロールについて説明しましたが、ファイル操作時に頻繁に使われるコントロールについて説明していませんでした。アプリケーションプログラムでは、ファイルを開き、そのファイルの内容を読み込んだり、ファイルにデータを書き出したりします。ここで、本章で行ったようにファイル名をプログラム中に記述したり、コマンドラインの引数で受け取ることも可能ですが、多くのWindowsプログラムにおけるファイル操作ではファイルチューザ (FileChooser) と呼ばれる画面が表れ、フォルダやファイルの選択をGUIにより容易に行えます。以下では、このFileChooser (javafx.stage.FileChooser) のJavaでの利用について説明します。

以下で説明するプログラム10.12の実行画面を図10.2に示します。プログラムを起動すると左側上の画面が表れます。中央の「ファイル選択」というボタンを押下すると、右側のファイルチューザが表れます。ここでファイルを選択し、「開く」ボタンを押すと、ファイル名などの情報がアプリケーションに渡ります。このプログラムでは、ボタンの下にラベルを配置し、そこに選択されたファイル名を表示しています。

[プログラム 10.12: ch10/FileChooserSample.java]

```
1  import java.io.File;
2  import javafx.application.Application;
3  import javafx.scene.Scene;
4  import javafx.scene.control.Button;
5  import javafx.scene.control.Label;
6  import javafx.scene.layout.VBox;
7  import javafx.stage.FileChooser;
8  import javafx.stage.Stage;
9
10 public class FileChooserSample extends Application {
11     Label lb=new Label();
12     @Override
13     public void start(Stage stage) {
14         Button btn = new Button("ファイル選択");
15         btn.setOnAction(event -> fileOpen(stage));
16         VBox root = new VBox();
17         root.getChildren().addAll(btn, lb);
18         Scene scene = new Scene(root, 200, 100);
19
20         stage.setTitle("FileChooser");
21         stage.setScene(scene);
22         stage.show();
23     }
24
25     void fileOpen(Stage stage) {
26         FileChooser fc=new FileChooser();
27         fc.setInitialDirectory(new File("C:\\home\\abc"));

```

```

28     File file=fc.showOpenDialog(stage);
29     if(file != null && file.isFile()) {
30         lb.setText(file.getName());
31     }
32 }
33 }

```

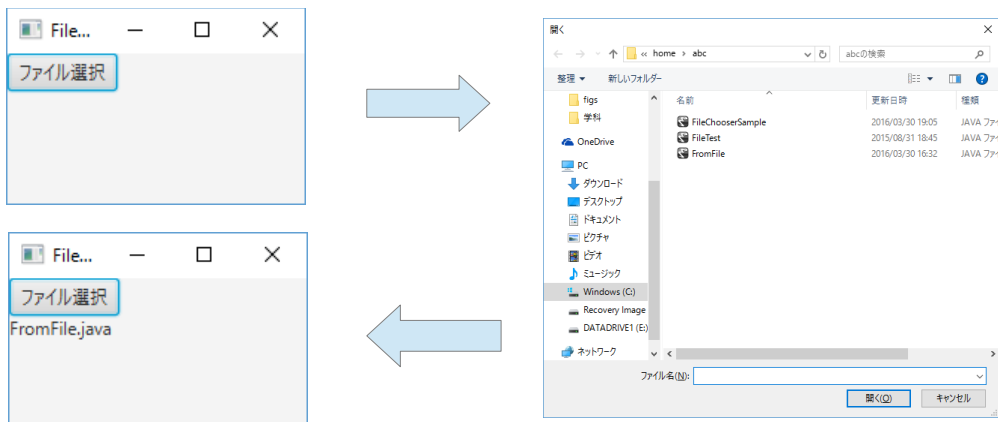


図 10.2 FileChooser の例

15 行目に示すように、ボタンが押されると、25 行目以下の `fileOpen` が実行されます。26 行目で `FileChooser` のオブジェクトをつくっています。27 行目では、ファイルチューザに最初に表示されるフォルダを設定しています。この設定が行われていないとき、Windows10 では PC の画面がファイルチューザに表示されます。

28 行目の処理でファイルチューザが画面に表れます (図 10.2)。その画面から目的とするファイルを選び「開く」ボタンを押すと、選ばれたファイルのオブジェクトが `file` に返されます。29 行目では、戻り値が `null` でなく、かつ選ばれたものがファイル (他にフォルダがあります) であれば、そのファイル名を得て、ラベルに表示しています。ファイルチューザで「取り消し」ボタンが押された場合、`file` には `null` が返されます。

プログラム 10.12 では、ファイルチューザに指定したフォルダ内のすべての種類のファイルが表示されます。しかし、アプリケーションごとに対象とするファイルの種類が決まっており、表示されるファイルも種類もアプリケーションが対象とするものに絞り込みたい場合があります。そのためには、ファイルチューザに `ExtensionFilter` (`javafx.shape.FileChooser.ExtensionFilter`) を設定します。

いま、フォルダ内の `java` ソースファイル (拡張子は `.java`) と、テキストファイル (拡張子は `.txt`) のみを表示したいとき、26 行目の下に、つぎの文を追加します。

```

fc.getExtensionFilters().addAll(
    new ExtensionFilter("Text Files","*.txt");
    new ExtensionFilter("Java Source","*.java"));

```

この追加を行った後の表示画面を図 10.3 に示します。ExtensionFiler のコンストラクタの第 1 引数は右下の楕円で囲んだ部分に表示される文字列です。ファイルの種類を判別しやすい文字列を指定します。また、第 2 引数には*につづけてそのファイルの拡張子を指定します。楕円で囲んだ部分は、ドロップダウンリストになっており、上で述べたように指定したファイルの種類を選択できます。このドロップダウンリストに表示される順番は、addAll メソッドに記述した順番になります。

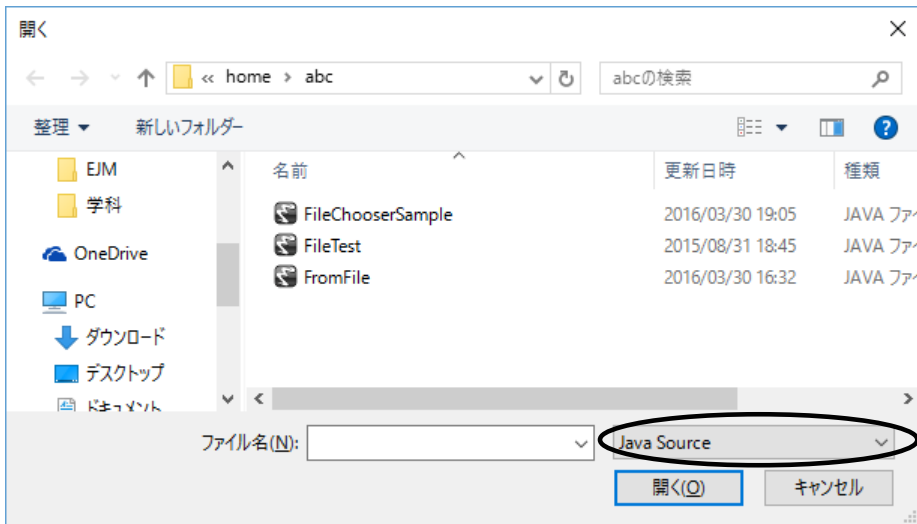


図 10.3 拡張子による絞り込み

章 末 問 題

- 【 1 】 以下の文章のうち、誤っている文章を選び、番号で答えなさい。
- (1) System.out.printf というメソッドは、import System.*; という宣言をファイルの先頭に記述することにより、out.printf で呼び出すことができる。
 - (2) System.in.read() メソッドは、例外を発生する可能性があるため、try-catch 文の中に記述するか、それを呼び出すメソッドが例外を throw しなければならない。
 - (3) Scanner クラスで入力を行うとき、整数データの読み込みは hasNextInt() メソッドを用いて行う。
 - (4) ファイル名の変更は、java.nio.Files クラスを用いて行える。
 - (5) バイナリーデータの出力には java.io.DataOutputStream を用いる。
 - (6) ランダムアクセスファイルにおいて、ファイル中での読み込み位置を指定するためには、getFilePointer メソッドを用いる。
- 【 2 】 つぎのプログラムは、java.io.PrintWriter クラスを用いて、カレントフォルダの text.txt という名前のファイルに 1 行の文字列を書き込み、ファイルを閉じる一連の流れが記述されている。空白の (A) ~ (C) を埋めてプログラムを完成させなさい。

索引

【あ】		
アクセス修飾	68	
アノテーション	75, 148	
アプリケーション	3	
アンボクシング	53	
【い】		
委任イベントモデル	121	
イベント	121	
イベントソース	121	
イベントハンドラ	73, 121	
インクリメント演算子	12	
インスタンス	36	
インスタンスフィールド	45	
インスタンス変数	45	
インタフェース	83	
——の実装	84	
インライン CSS	129	
【え】		
演算子	12	
——の優先順位	16	
【お】		
オーバーライド	61	
オーバーロード	38	
オブジェクト	36	
親クラス	58, 60	
【か】		
外部クラス	73	
拡張 for 文	22, 196	
型引数	91	
可変個の引数	29	
可変長引数	29	
仮引数	27	
関係演算子	13	
関数型インタフェース	98, 122	
【き】		
キーボードイベントの処理	156	
擬似乱数	182	
基本データ型	9	
キャスト	26, 66	
【く】		
空白文字	166	
区切り文字	168	
クラス	7, 33	
——の拡張	58, 60	
——の継承	58	
クラスフィールド	46	
クラス変数	46	
クラスメソッド	47	
クラスライブラリー	101, 182	
グラフィックコンテキスト	151	
繰り返し	21	
グリッドレイアウト	126	
【け】		
ゲッター	69	
【こ】		
コアクラス	2	
子クラス	58, 60	
コメント	24	
コレクション	187	
コレクション型	191	
コレクションクラス	187	
コンストラクタ	34, 42	
コンテナ	118	
コントロール	119	
コンパレータ	95	
コンボボックス	141	
【さ】		
サブクラス	58, 60	
サブパッケージ	101	
算術演算子	12	
参照	41	
【し】		
シーングラフ	123, 148	
識別子	10	
シグネチャ	39, 80	
自然な順序づけ	191	
実引数	28	
条件演算子	16	
条件分岐	19	
【す】		
数学関数	182	
スーパークラス	58, 60	
スクロールバー	140	
図形描画	150	
ストリーム	163	
スライダ	145	
スレッド	201	
——の終了を待つ	207	
スレッド間通信	212	
スレッド間の同期	209, 211	
【せ】		
正規表現	168	
制御構造	19	
静的インポート	102	
セッター	69	
線の太さ	153	
【そ】		
総称型	90	
ソーティング	95	
【た】		
ダイヤモンド演算子	92	
ダイヤモンド記法	192	
多重継承	80, 83	
多重定義	38	
単項演算子	13	
【ち】		
チェックボックス	135	
抽象クラス	82	
抽象メソッド	82	
【て】		
定数	10	
テキストエリア	140	

テキストフィールド	139	フォント	129	メッセージパッシング	35
デクリメント演算子	12	浮動小数点型	10	【も】	
デフォルト値	42	プルダウンメニュー	143	文字ストリーム	163
デフォルトのコンストラクタ	42	ブロック	9	文字定数	10
デフォルトパッケージ	101	【へ】		文字の色	129
【と】		並列型ストリーム	197	文字列結合演算子	14
統合開発環境	76	変数のスコープ	24	【ゆ】	
動的結合	66	【ほ】		優先順位	16
匿名クラス	96	ボーダーレイアウト	124	【よ】	
【な】		ボクシング	53	予約語	11
内部クラス	72	ポリモーフィズム	64, 82	【ら】	
【は】		【ま】		ラジオボタン	137
背景色	129	マウス	154	ラッパークラス	53
バイトコード	1	マウスイベント	154	ラベル付き break 文	23
バイトストリーム	163, 172	——の処理	154	ラムダ式	122, 136, 140, 198
配列	17	マッピング	197	ランダムアクセスファイル	175
パッケージ	101, 103	マルチスレッド	201	【れ】	
パッケージ内	68	【む】		レイアウト	118
ハッシュコード	191	無名クラス	96	例外	107
バッファリング処理	163	無名内部クラス	73, 96	列挙型	26
【ひ】		無名のパッケージ	101	【ろ】	
ビット演算子	14	【め】		論理演算子	13
標準入出力	163	命名規則	11	【わ】	
標準ライブラリー	101	メインスレッド	201	割込み	205
【ふ】		メソッド	7, 27, 34, 37		
ファイルチューザ	177	——のシグネチャ	39		
フィールド	33, 37	メソッド参照	199		

【A】		BufferedOutputStream	172	ComboBox	141
abstract	82	Button	120	Comparable	94, 96, 190
abstract 修飾子	83	byte	9	Comparator	95, 189
ActionEvent	121	【C】		continue 文	23
Application クラス	118	Canvas	151	CSS	129
Arc	150	catch ブロック	109	CUI	116
ArithmeticException	108	char	9	【D】	
ArrayList クラス	191	CharSequence クラス	92	DataInputStream	172
Arrays	184	CheckBox	135	DataOutputStream	172
Arrays クラス	94	ChoiceBox	143	double	9
【B】		Circle	150	do-while 文	21
boolean	9	class	1, 7	【E】	
BorderLayout	124	CLASSPATH	106	Ellipse	150
break 文	23	classpath	105	enum 型	26
		Color クラス	153		

Error	107	jar	2, 105	PrintWriter	170
Exception	107	java	2	PriorityQueue クラス	195
Exception クラス	108	Java 仮想マシン	1	private	68
extends	61	javac	2	protected	68
ExtensionFilter	178	JavaFX	116	public	68
	【F】	Java SE	2		【Q】
File	171	javaw	117	QuadCurve	150
FileChooser	177	JDK	2	Queue	188
FileOutputStream	172	JIT コンパイラ	2		【R】
Files	171	JRE	1		
final	10		【K】	RadioButton	137
final 修飾子	38, 90	KeyEvent	156	Random	182
finally ブロック	109		【L】	RandomAccessFile	175
float	9	Label	120	Rectangle	150
FlowPane	127	Lambda 式	97	Runnable	201, 204
for 文	22	Line	150		【S】
FXML	146	LinkedList	197	Scanner	166
	【G】	List	188, 192	Scene	118
GC	151	LocalDateTime	186	Set	188
Generics	90	long	9	setOnAction	136
GraphicsContext	151		【M】	Shape	150
GridLayout	126	main メソッド	4	sleep	205
Group	151	Map	188	slider	145
GUI	116	Math クラス	182	Stack クラス	193
GUI アプリケーション	116	MouseButton	154	StackPane	118
	【H】	MouseEvent	154	Stage クラス	118
HashMap	197		【N】	start	117
HashMap クラス	194	new 演算子	35	static	46, 48, 98
HashTable	197	notify	213	Stream	197
HBox	124	notifyAll	213	String クラス	50
	【I】	NullPointerException	108	StringBuffer	89
if 文	19		【O】	sub-class	60
implements	84	Object クラス	70	super	61
import	102	outer class	73	super class	60
inheritance	58		【P】	switch 文	20
inner class	72	Pane	118	synchronized 修飾子	211, 213
InputStream	164	Path	168, 170	System.err	164
instanceof	67	Paths	168, 170	System.in	164
instanceof 演算子	86	Polygon	150	System.out	164
Instant クラス	186	Polyline	150		【T】
Integer クラス	53	polymorphism	64	TextArea	140
InterruptedException	206	print	15	TextField	139
Iterable	196	printf	15	this	59
Iterator	197	println	15	Thread クラス	201
	【J】	PrintStream	164, 169	Throwable	108
JapaneseDate	187			throws	111
				ToggleGroup	137
				TreeSet	189, 195

try ブロック	109	Vector	197		
try-catch 文	107			【記号】	
try-with-resources	173	【W】		@Deprecated	75
		wait		213 @FXML	148
【U】		while 文		21 @Override	75
Unicode	9	【数字】		@SuppressWarnings	76
		2 項算術演算子			
【V】			12		
VBox	120				

— 著者略歴 —

小林 貴訓 (こばやし よしのり)

2000年 電気通信大学大学院修士課程修了 (情報システム運用学専攻)
2000年 三菱電機株式会社
2007年 東京大学大学院博士課程修了 (電子情報学専攻)
博士 (情報理工学)
2007年 埼玉大学助教
2014年 埼玉大学准教授
現在に至る

Htoo Htoo (とう とう)

2004年 ヤンゴンコンピュータ大学修士課程修了 (計算機科学専攻)
2004年 ヤンゴンコンピュータ大学助教
~08年
2013年 埼玉大学大学院博士後期課程修了 (数理電子情報専攻)
博士 (工学)
2013年 埼玉大学助教
現在に至る

大澤 裕 (おおさわ ゆたか)

1978年 信州大学大学院修士課程修了 (電子工学専攻)
1982年 東京大学生産技術研究所助手
1985年 工学博士 (東京大学)
1989年 埼玉大学助手
1990年 埼玉大学助教
1998年 埼玉大学教授
現在に至る

オブジェクト指向言語 Java

Object Oriented Programming Language Java

© Yoshinori Kobayashi, Htoo Htoo, Yutaka Ohsawa 2016

2016年11月2日 初版第1刷発行



検印省略

著者 小林 貴訓

Htoo Htoo

大澤 裕

発行者 株式会社 コロナ社

代表者 牛来真也

印刷所 三美印刷株式会社

112-0011 東京都文京区千石 4-46-10

発行所 株式会社 コロナ社

CORONA PUBLISHING CO., LTD.

Tokyo Japan

振替 00140-8-14844・電話 (03) 3941-3131 (代)

ホームページ <http://www.coronasha.co.jp>

ISBN 978-4-339-02865-2 (金) (製本: 愛千製本所)

Printed in Japan



本書のコピー、スキャン、デジタル化等の無断複製・転載は著作権法上での例外を除き禁じられています。購入者以外の第三者による本書の電子データ化及び電子書籍化は、いかなる場合も認めておりません。

落丁・乱丁本はお取替えいたしません