JavaFX: JavaFX UIコンポーネントの操作

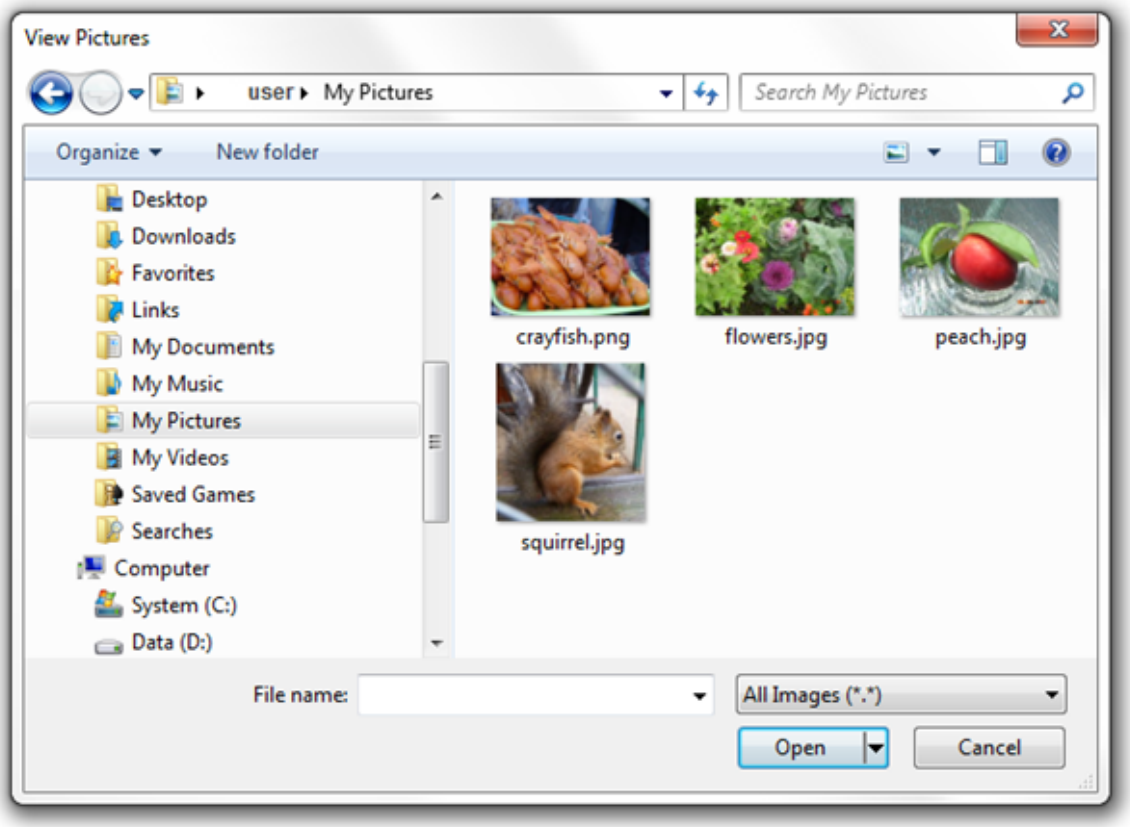
28 ファイル・チューザ

この章では、ユーザーがファイル・システムをナビゲートできるようにするためにFileChooserクラスを使用する方法について説明します。この章で示すサンプルでは、1つ以上のファイルを開く方法、ファイル・チューザ・ダイアログ・ウィンドウを構成する方法、およびアプリケーションのコンテンツを保存する方法について説明します。

他のユーザー・インタフェース・コンポーネント・クラスとは異なり、FileChooserクラスはjavafx.scene.controlsパッケージに属していません。ただし、このクラスは一般的なGUIアプリケーション機能の1つであるファイル・システムのナビゲーションをサポートしているため、JavaFX UIコントロールのチュートリアルで説明されます。

FileChooserクラスは、Stage、WindowおよびPopupなどの他の基本ルート・グラフィカル要素とともにjavafx.stageパッケージ内に格納されています。図28-1の「View Pictures」ウィンドウは、Windowsのファイル・チューザ・ダイアログの例です。

図28-1 ファイル・チューザ・ウィンドウの例



「図28-1 ファイル・チューザ・ウィンドウの例」の説明 (img_text/file-chooser-sample.htm)

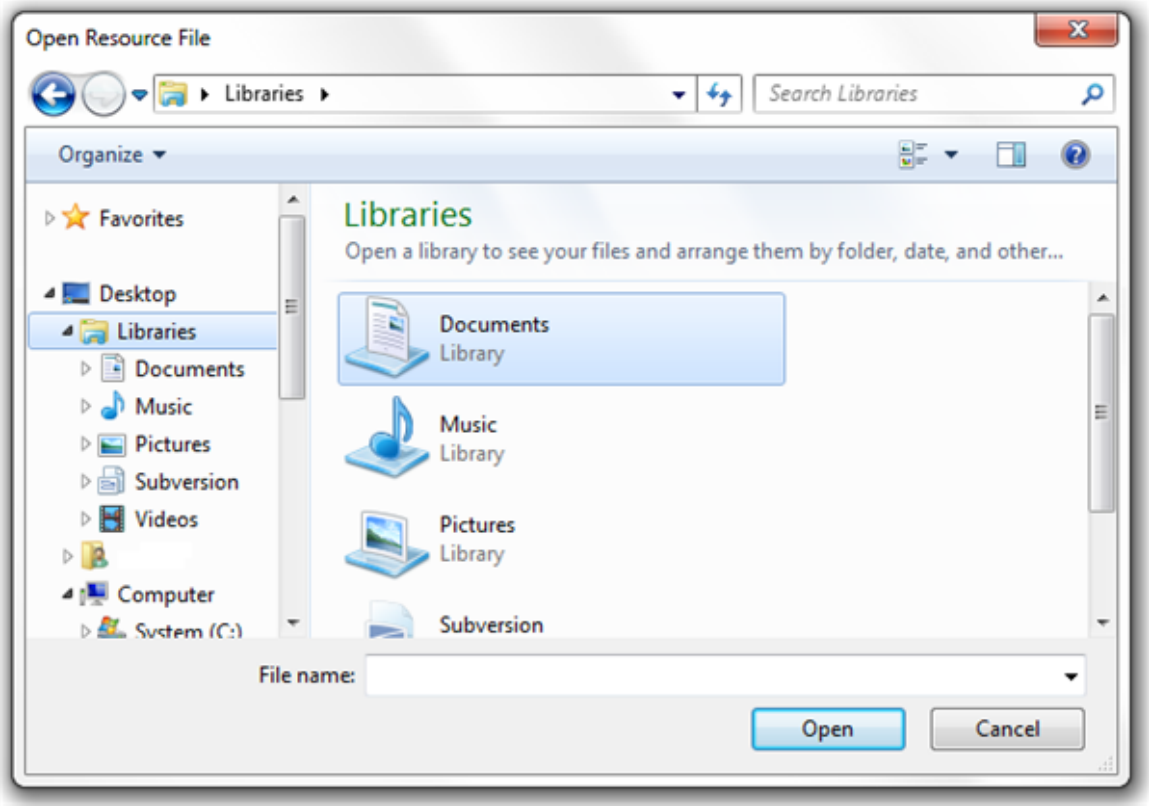
ファイル・チューザを使用して、単一のファイルまたは複数のファイルを選択するためのオープン・ダイアログ・ウィンドウを呼び出したり、ファイルの保存ダイアログ・ウィンドウを有効にすることができます。ファイル・チューザを表示するには、通常、FileChooserクラスを使用します。例28-1に、アプリケーションでファイル・チューザを有効にする最も簡単な方法を示します。

例28-1 ファイル・チューザの表示

```
FileChooser fileChooser = new FileChooser();
fileChooser.setTitle("Open Resource File");
fileChooser.showOpenDialog(stage);
```

例28-1のコードをJavaFXアプリケーションに追加した後、図28-2に示すように、アプリケーションが起動すると同時にファイル・チューザ・ダイアログ・ウィンドウが表示されます。

図28-2 簡単なファイル・チューザ

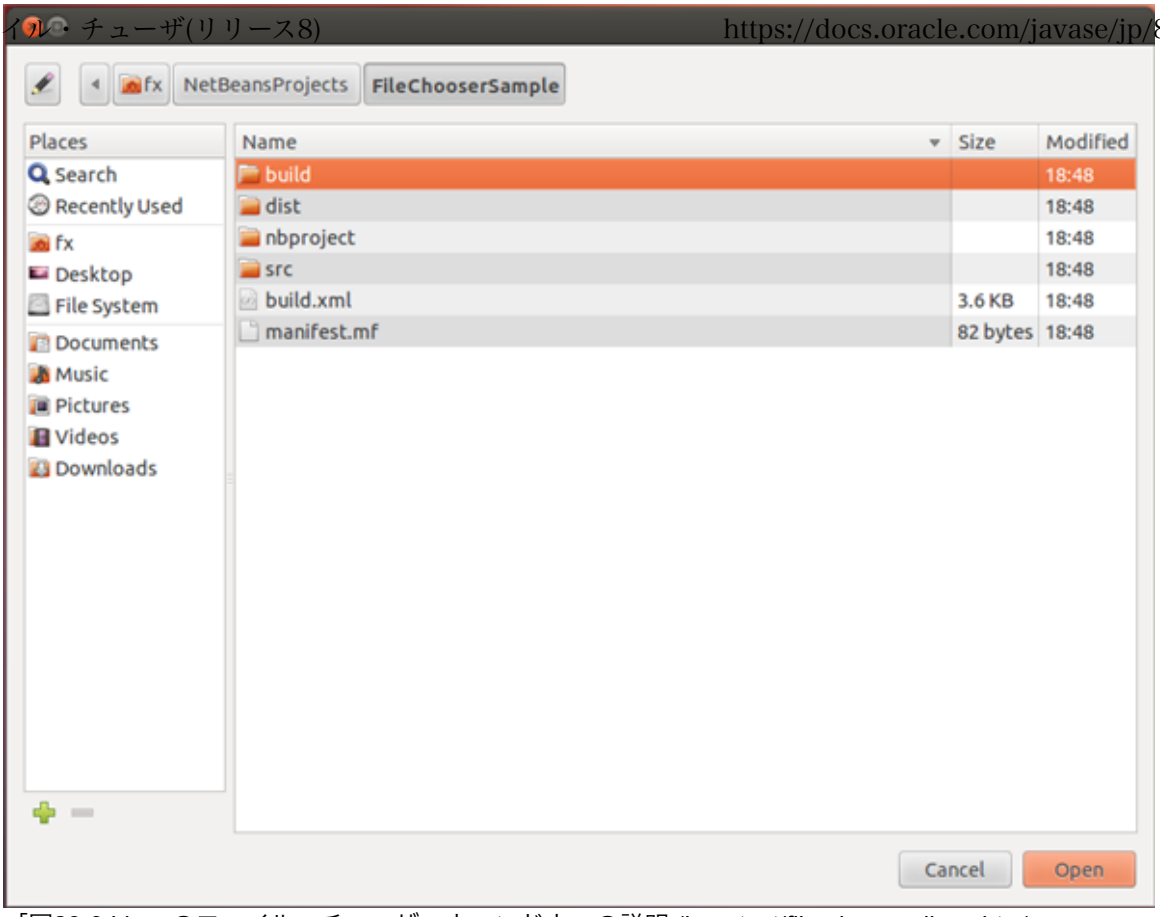


「図28-2 簡単なファイル・チューザ」の説明 (img_text/file-chooser-simple.htm)

注意:

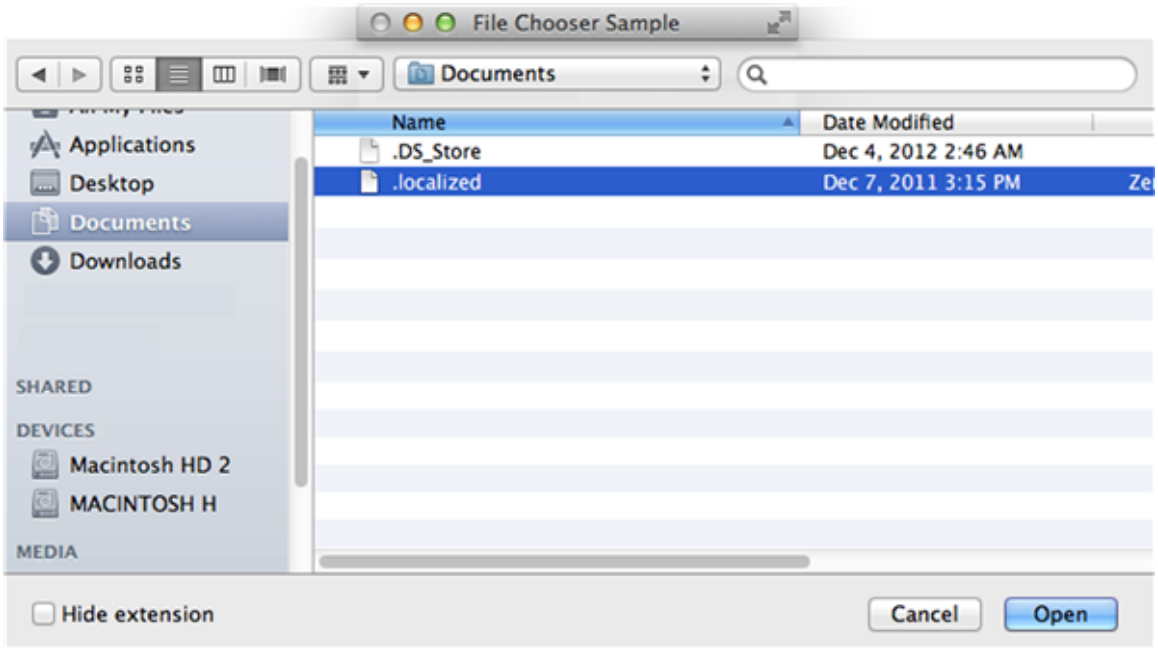
図28-2は、Windowsのファイル・チューザを示しています。この機能をサポートしている他のオペレーティング・システムでファイル・チューザを開くと、別のウィンドウが表示されます。図28-3および図28-4は、LinuxおよびMac OSのファイル・チューザ・ウィンドウの例を示しています。

図28-3 Linuxのファイル・チューザ・ウィンドウ



「図28-3 Linuxのファイル・チューザ・ウィンドウ」の説明 (img_text/file-chooser-linux.htm)

図28-4 Mac OSのファイル・チューザ・ウィンドウ



「図28-4 Mac OSのファイル・チューザ・ウィンドウ」の説明 (img_text/file-chooser-mac.htm)

前述の例ではアプリケーションの起動時にファイル・チューザが自動的に表示されましたが、より一般的なアプローチの場合、対応するメニュー項目を選択するか専用ボタンをクリックしてファイル・チューザを呼び出します。このチュートリアルでは、ユーザーがボタンをクリックしてファイル・システム内にある1つ以上の画像を開くことができるようにするアプリケーションを作成します。例28-2に、このタスクを実装するFileChooserSampleアプリケーションのコードを示します。

例28-2 単一選択および複数選択用のファイル・チューザを開く

```
import java.awt.Desktop;
import java.io.File;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

public final class FileChooserSample extends Application {

    private final Desktop desktop = Desktop.getDesktop();

    @Override
    public void start(final Stage stage) {
        stage.setTitle("File Chooser Sample");

        final FileChooser fileChooser = new FileChooser();

        final Button openButton = new Button("Open a Picture...");
        final Button openMultipleButton = new Button("Open Pictures...");

        openButton.setOnAction(
            (final ActionEvent e) -> {
                File file = fileChooser.showOpenDialog(stage);
                if (file != null) {
                    openFile(file);
                }
        });
        openMultipleButton.setOnAction(
            (final ActionEvent e) -> {
                List<File> list =
                    fileChooser.showOpenMultipleDialog(stage);
                if (list != null) {
                    list.stream().forEach((file) -> {
                        openFile(file);
                    });
                }
        });

        final GridPane inputGridPane = new GridPane();

        GridPane.setConstraints(openButton, 0, 0);
        GridPane.setConstraints(openMultipleButton, 1, 0);
        inputGridPane.setHgap(6);
        inputGridPane.setVgap(6);
        inputGridPane.getChildren().addAll(openButton, openMultipleButton);
```

```
        final Pane rootGroup = new VBox(12);
        rootGroup.getChildren().addAll(inputGridPane);
        rootGroup.setPadding(new Insets(12, 12, 12, 12));

        stage.setScene(new Scene(rootGroup));
        stage.show();
    }

    public static void main(String[] args) {
        Application.launch(args);
    }

    private void openFile(File file) {
        EventQueue.invokeLater(() -> {
            try {
                desktop.open(file);
            } catch (IOException ex) {
                Logger.getLogger(FileChooserSample.
                        class.getName()).
                        log(Level.SEVERE, null, ex);
            }
        });
    }
}
```

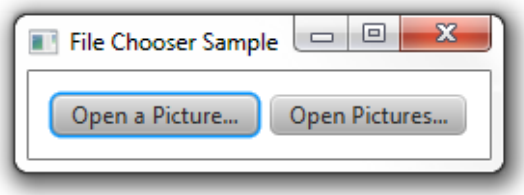
例28-2では、「Open a Picture」ボタンを使用して、ユーザーが単一選択用のファイル・チューザを開けるようにし、「Open Pictures」ボタンを使用して、ユーザーが複数選択用のファイル・チューザを開けるようにしています。これらのボタンの`setOnAction`メソッドはほぼ同一です。唯一の違いは、`FileChooser`を呼び出すために使用されるメソッド内にあります。

- `showOpenDialog`メソッドは、1つのファイルを選択できる新しいファイル・オープン・ダイアログを表示します。このメソッドは、ユーザーによって選択されたファイルを示す値、または選択が行われていない場合は`null`を返します。
- `showOpenMultipleDialog`メソッドは、複数のファイルを選択できる新しいファイル・オープン・ダイアログを表示します。このメソッドは、ユーザーによって選択されたファイルのリストを示す値、または選択が行われていない場合は`null`を返します。返されたリストは変更できず、変更が試みられるたびに`UnsupportedOperationException`がスローされます。

どちらのメソッドでも、表示されたオープン・ダイアログ・ウィンドウが閉じられるまで(つまり、ユーザーが選択をコミットするか取り消すまで)は結果が返されません。

FileChooserSampleアプリケーションをコンパイルして実行すると、図28-5に示すウィンドウが生成されます。

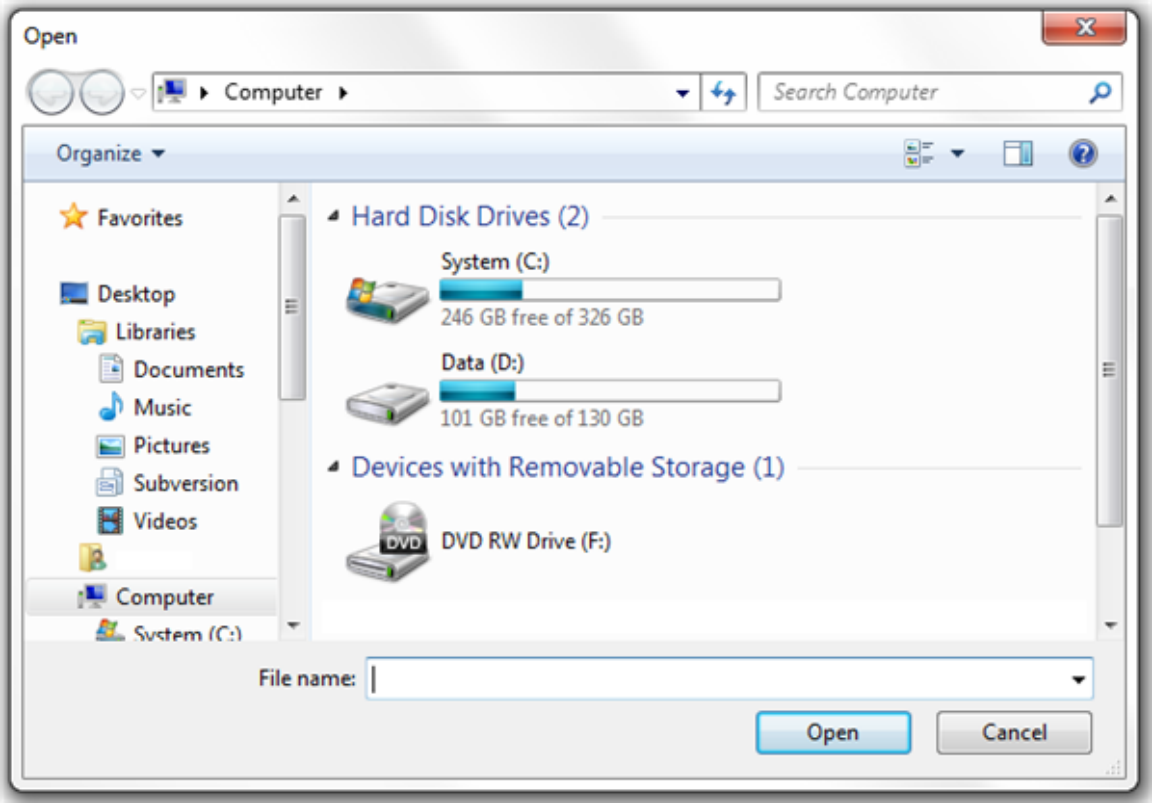
図28-5 2つのボタンがあるFileChooserSample



「図28-5 2つのボタンがあるFileChooserSample」の説明 (img_text/file-chooser-with-buttons.htm)

いずれかのボタンをクリックすると、図28-6に示すダイアログ・ウィンドウが表示されます。開いたファイル・チューザ・ダイアログ・ウィンドウには、オペレーティング・システムのデフォルトの場所が表示されます。

図28-6 デフォルトのファイル・チューザ・ウィンドウ



「図28-6 デフォルトのファイル・チューザ・ウィンドウ」の説明 (img_text/file-chooser-default.htm)

FileChooserSampleアプリケーションのユーザーは、画像が含まれているディレクトリにナビゲートし、画像を選択できます。ファイルを選択すると、関連するアプリケーションで開きます。コード例では、`java.awt.Desktop`クラスの`open`メソッドである`desktop.open(file);`を使用してこれを実装しています。

注意:

`Desktop`クラスの可用性はプラットフォームに依存します。`Desktop`クラスの詳細は、APIドキュメント (../../docs/api/)を参照してください。また、`isDesktopSupported()`メソッドを使用して、それがシステム内でサポートされているかどうかを確認することもできます。

このアプリケーションのユーザー操作性を改善するには、ファイル・チューザ・ディレクトリを、画像が含まれている特定のディレクトリに設定します。

ファイル・チューザの構成

`FileChooser`オブジェクトの`initialDirectory`および`title`プロパティを設定することにより、ファイル・チューザ・ダイアログ・ウィンドウを構成できます。例28-3に、画像をプレビューしたり開くための初期ディレクトリおよび適切なタイトルを指定する方法を示します。

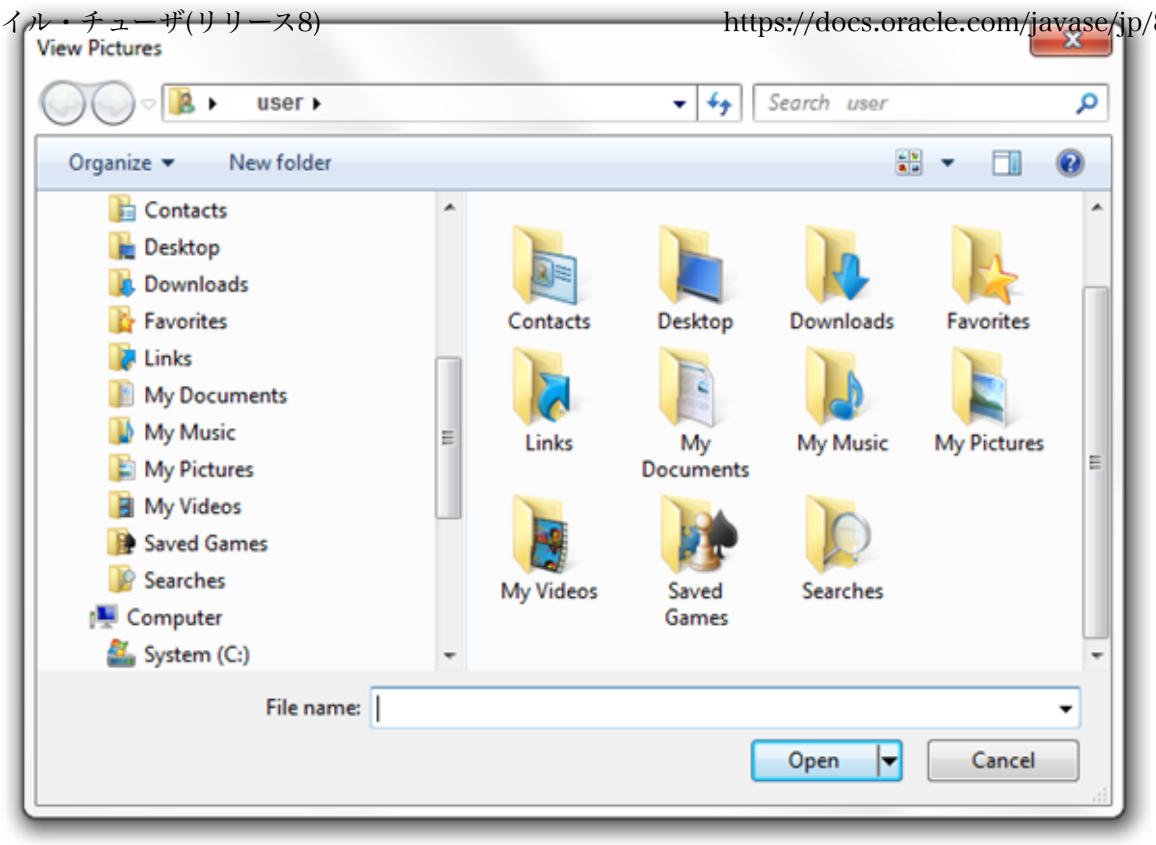
例28-3 初期ディレクトリおよびウィンドウ・タイトルの設定

```
import java.awt.Desktop;
import java.io.File;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

public final class FileChooserSample extends Application {

    private final Desktop desktop = Desktop.getDesktop();

    @Override
    public void start(final Stage stage) {
        stage.setTitle("File Chooser Sample");

        final FileChooser fileChooser = new FileChooser();

        final Button openButton = new Button("Open a Picture...");
        final Button openMultipleButton = new Button("Open Pictures...");

        openButton.setOnAction(
            (final ActionEvent e) -> {
                configureFileChooser(fileChooser);
                File file = fileChooser.showOpenDialog(stage);
                if (file != null) {
                    openFile(file);
                }
        });        openMultipleButton.setOnAction(
            (final ActionEvent e) -> {
                configureFileChooser(fileChooser);
                List<File> list =
                        fileChooser.showOpenMultipleDialog(stage);
                if (list != null) {
                    list.stream().forEach((file) -> {
                        openFile(file);
                    });
                }
        });

        final GridPane inputGridPane = new GridPane();

        GridPane.setConstraints(openButton, 0, 0);
        GridPane.setConstraints(openMultipleButton, 1, 0);
        inputGridPane.setHgap(6);
        inputGridPane.setVgap(6);
```

```
        inputGridPane.getChildren().addAll(openButton, openMultipleButton);

        final Pane rootGroup = new VBox(12);
        rootGroup.getChildren().addAll(inputGridPane);
        rootGroup.setPadding(new Insets(12, 12, 12, 12));

        stage.setScene(new Scene(rootGroup));
        stage.show();
    }

    public static void main(String[] args) {
        Application.launch(args);
    }

    private static void configureFileChooser(final FileChooser fileChooser){
        fileChooser.setTitle("View Pictures");
        fileChooser.setInitialDirectory(
            new File(System.getProperty("user.home"))
        );
    }

    private void openFile(File file) {
        EventQueue.invokeLater(() -> {
            try {
                desktop.open(file);
            } catch (IOException ex) {
                Logger.getLogger(FileChooserSample.
                        class.getName()).
                        log(Level.SEVERE, null, ex);
            }
        });
    }
}
```

configureFileChooserメソッドは、「View Pictures」タイトル、および「マイ ピクチャ」サブディレクトリがあるuserホーム・ディレクトリへのパスを設定しています。FileChooserSampleをコンパイルして実行し、いずれかのボタンをクリックすると、図28-7に示すファイル・チューザが表示されます。

図28-7 画像ライブラリを開く



「図28-7 画像ライブラリを開く」の説明 (img_text/file-chooser-pictures.htm)

また、DirectoryChooserクラスを使用して、ユーザーがターゲット・ディレクトリを指定できるようにすることもできます。例28-4に示すコード・フラグメントでは、browseButtonをクリックするとdirectoryChooser.showDialogメソッドが呼び出されます。

例28-4 DirectoryChooserクラスの使用

```
final Button browseButton = new Button("...");
browseButton.setOnAction(
    (final ActionEvent e) -> {
        final DirectoryChooser directoryChooser
            new DirectoryChooser();
        final File selectedDirectory =
                directoryChooser.showDialog(stage);
        if (selectedDirectory != null) {
            selectedDirectory.getAbsolutePath();
        }
});
```

選択を実行した後、fileChooser.setInitialDirectory(selectedDirectory);のように、ファイル・チューザに対応する値を割り当てることができます。

拡張子フィルタの設定

次の構成オプションとして、例28-5に示すように、ファイル・チューザで開くファイルを決定するための拡張子フィルタを設定できます。

例28-5 イメージ・タイプ・フィルタの設定

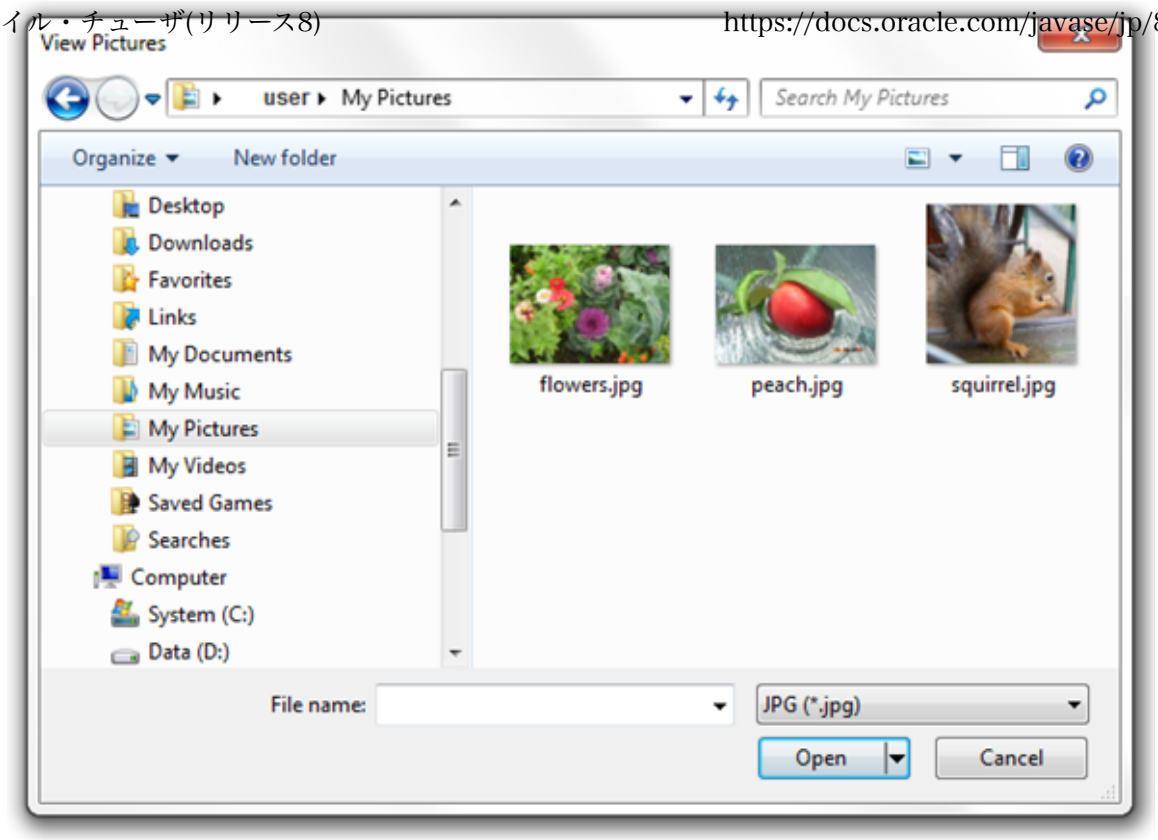
```
import java.awt.Desktop;
import java.io.File;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Pane;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

public final class FileChooserSample extends Application {

    private final Desktop desktop = Desktop.getDesktop();

    @Override
    public void start(final Stage stage) {
        stage.setTitle("File Chooser Sample");

        final FileChooser fileChooser = new FileChooser();
        final Button openButton = new Button("Open a Picture...");
        final Button openMultipleButton = new Button("Open Pictures...");

        openButton.setOnAction(
            (final ActionEvent e) -> {
                configureFileChooser(fileChooser);
                File file = fileChooser.showOpenDialog(stage);
                if (file != null) {
                    openFile(file);
                }
        });

        openMultipleButton.setOnAction(
            (final ActionEvent e) -> {
                configureFileChooser(fileChooser);
                List<File> list =
                        fileChooser.showOpenMultipleDialog(stage);
                if (list != null) {
                    list.stream().forEach((file) -> {
                        openFile(file);
                    });
                }
        });

        final GridPane inputGridPane = new GridPane();

        GridPane.setConstraints(openButton, 0, 1);
        GridPane.setConstraints(openMultipleButton, 1, 1);
        inputGridPane.setHgap(6);
```

```
        inputGridPane.setVgap(6);
        inputGridPane.getChildren().addAll(openButton, openMultipleButton);

        final Pane rootGroup = new VBox(12);
        rootGroup.getChildren().addAll(inputGridPane);
        rootGroup.setPadding(new Insets(12, 12, 12, 12));

        stage.setScene(new Scene(rootGroup));
        stage.show();
    }

    public static void main(String[] args) {
        Application.launch(args);
    }

    private static void configureFileChooser(
        final FileChooser fileChooser) {
            fileChooser.setTitle("View Pictures");
            fileChooser.setInitialDirectory(
                new File(System.getProperty("user.home"))
            );
            fileChooser.getExtensionFilters().addAll(
                new FileChooser.ExtensionFilter("All Images", "*.*"),
                new FileChooser.ExtensionFilter("JPG", "*.jpg"),
                new FileChooser.ExtensionFilter("PNG", "*.png")
            );
    }

    private void openFile(File file) {
        EventQueue.invokeLater(() -> {
            try {
                desktop.open(file);
            } catch (IOException ex) {
                Logger.getLogger(FileChooserSample.
                        class.getName()).
                        log(Level.SEVERE, null, ex);
            }
        });
    }
}
```

例28-5では、FileChooser.ExtensionFilterを使用してファイル選択用のオプションとして「All images」、「JPG」および「PNG」を定義し、拡張子フィルタを設定しています。

例28-5のFileChooserSampleコードをコンパイルして実行し、いずれかのボタンをクリックすると、ファイル・チューザ・ウィンドウに拡張子フィルタが表示されます。ユーザーがJPGを選択すると、ファイル・チューザにJPGタイプの画像のみが表示されます。図28-8は、「マイ ピクチャ」ディレクトリ内でJPG画像を選択した瞬間を捉えています。

図28-8 ファイル・チューザでのJPGファイルのフィルタ処理



「図28-8 ファイル・チューザでのJPGファイルのフィルタ処理」の説明 (img_text/file-chooser-jpg.htm)

ファイルの保存

ファイルを開いたりフィルタ処理する以外にも、FileChooser APIには、アプリケーションによって保存されるファイルのファイル名(およびファイル・システム内の場所)をユーザーが指定できるようにする機能が用意されています。FileChooserクラスのshowSaveDialogメソッドは、保存ダイアログ・ウィンドウを開きます。他の表示ダイアログ・メソッドの場合と同様、showSaveDialogメソッドは、ユーザーによって選択されたファイル、または選択が行われていない場合はnullを返します。

例28-6に示すコード・フラグメントは、メニュー (menu_controls.htm#BABGHADI)・サンプルの追加コードです。これは、表示されているイメージをファイル・システムに保存するコンテキスト・メニューの項目を追加実装します。

例28-6 FileChooserクラスを使用したイメージの保存

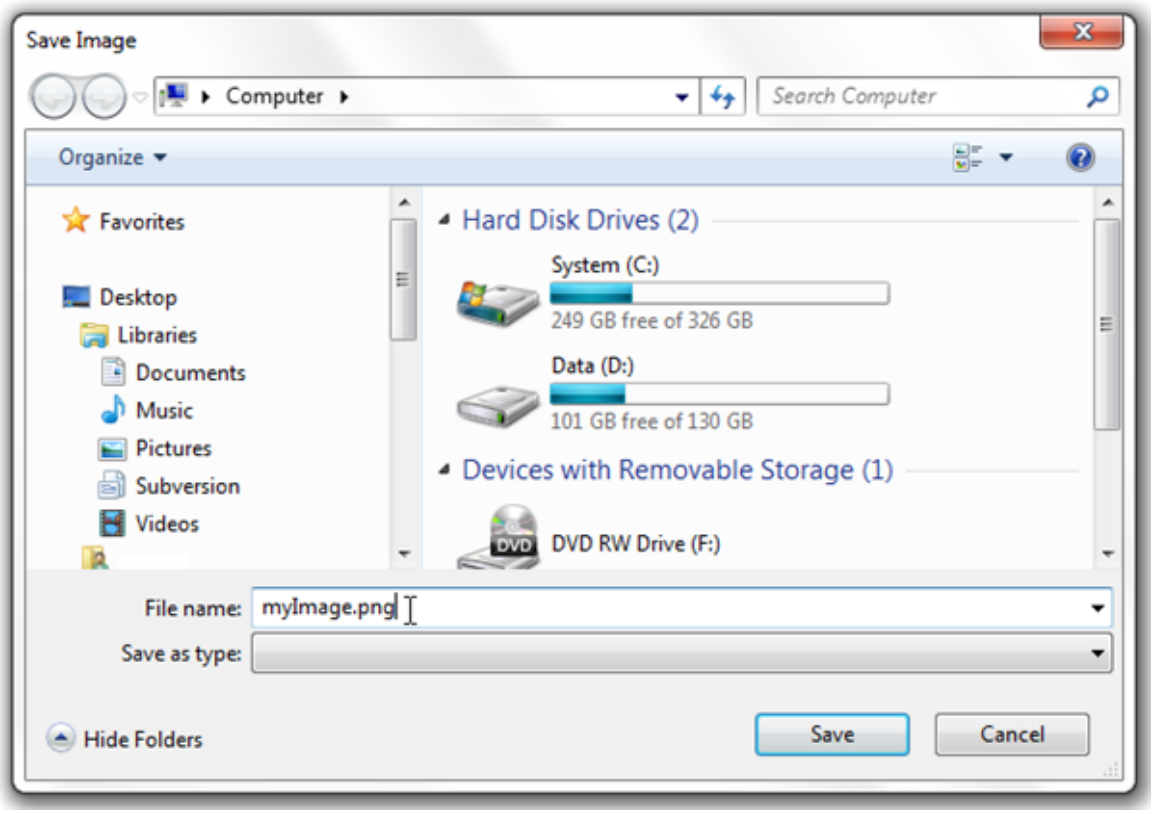
```
MenuItem cmItem2 = new MenuItem("Save Image");
cmItem2.setOnAction((ActionEvent e) -> {
    FileChooser fileChooser1 = new FileChooser();
    fileChooser1.setTitle("Save Image");
    System.out.println(pic.getId());
    File file = fileChooser1.showSaveDialog(stage);
    if (file != null) {
        try {
            ImageIO.write(SwingFXUtils.fromFXImage(pic.getImage(),
                    null), "png", file);
        } catch (IOException ex) {
             System.out.println(ex.getMessage());
        }
    }
});
```

例28-6をMenuSampleアプリケーション(アプリケーション・ファイル内でソース・コードを検索)に追加し、コンパイルして実行すると、図28-9に示すように、「Save Image」メニュー項目が有効になります。

「図28-9 イメージの保存」の説明 (img_text/file-chooser-save.htm)

「Save Image」項目を選択すると、図28-10に示す「Save Image」ウィンドウが表示されます。

図28-10 「Save Image」ウィンドウ



「図28-10 「Save Image」ウィンドウ」の説明 (img_text/file-chooser-save-image.htm)

「Save Image」ウィンドウは、保存ダイアログ・ウィンドウの一般的なユーザー操作性に対応しています。つまり、ユーザーはターゲット・ディレクトリを選択し、保存ファイルの名前を入力し、「保存」をクリックします。

関連APIドキュメント

- `FileChooser` (../api/javafx/stage/FileChooser.html)
- `DirectoryChooser` (../api/javafx/stage/DirectoryChooser.html)