

Main.java

このコードは、[Triangle](#) と [Circle](#) という 2 つの形状の面積を計算し、その結果を表示するものです。これらの形状は、[Shape](#) というインターフェースを実装しています。

まず、[Triangle](#) クラスのインスタンス [triangle](#) が作成され、その基底と高さはそれぞれ 10 と 3 に設定されます。次に、[Circle](#) クラスのインスタンス [circle](#) が作成され、その半径は 10 に設定されます。

次に、[Shape](#) 型の変数 [shape1](#) と [shape2](#) に、それぞれ [triangle](#) と [circle](#) が代入されます。これにより、[Shape](#) インターフェースを通じて、これらの形状の面積を計算することができます。

最後に、[System.out.println](#) メソッドを使用して、それぞれの形状の面積を表示します。面積は double 型で計算されますが、表示する際には(int)を使用して整数にキャスト(型変換)しています。これは、小数点以下の値を切り捨て、整数部分だけを表示するためです。

Shape.java

このコードは、Java で定義されたインターフェース [Shape](#) を示しています。インターフェースは、クラスが実装すべきメソッドを定義するためのテンプレートのようなものです。この場合、[Shape](#) インターフェースは [getArea](#) というメソッドを定義しています。

[public double getArea\(\);](#)という行は、このインターフェースを実装するすべてのクラスが [getArea](#) メソッドを持つことを要求しています。このメソッドは、戻り値として double 型の値を返すことが期待されています。

この [Shape](#) インターフェースは、例えば [Circle](#) や [Rectangle](#) など、さまざまな形状を表すクラスに実装されるかもしれません。それぞれのクラスは、その形状の面積を計算して返すための [getArea](#) メソッドを提供する必要があります。

Triangle.java

このコードは、Java で定義された [Triangle](#) というクラスを示しています。このクラスは [Shape](#) インターフェースを実装しています。これは、[Triangle](#) クラスが [Shape](#) インターフェースで定義されたメソッドをすべて持つことを保証します。

[Triangle](#) クラスには、[base](#) と [height](#) という 2 つの double 型のパブリックフィールドがあります。これらは、三角形の底辺と高さを表します。

次に、[Triangle](#) クラスのコンストラクタが定義されています。このコンストラクタは、底辺

と高さの値を引数として受け取り、それらをクラスのフィールドに設定します。this キーワードは、現在のインスタンスを参照します。したがって、this.base = base; と this.height = height; は、引数として渡された値をそれぞれのフィールドに設定します。

最後に、[Shape](#) インターフェイスで定義された [getArea](#) メソッドが [Triangle](#) クラスでオーバーライドされています。このメソッドは、三角形の面積を計算して返します。面積の計算は、 $0.5 * \text{base} * \text{height}$ という式で行われ、これは三角形の面積を求める一般的な公式です。

Circle.java

このコードは、[Shape](#) インターフェイスを実装した [Circle](#) クラスを定義しています。[Circle](#) クラスは、円の半径を表す [radius](#) というパブリックな double 型のフィールドを持っています。

```
public double radius;
```

次に、[Circle](#) クラスのコンストラクタを見てみましょう。このコンストラクタは、引数として double 型の [radius](#) を受け取り、それをインスタンスの [radius](#) フィールドに設定します。

```
Circle(double radius) {  
    this.radius = radius;  
}
```

最後に、[Shape](#) インターフェイスで定義された [getArea](#) メソッドをオーバーライドしています。このメソッドは、円の面積を計算し、その結果を返します。円の面積は、半径の二乗に π を掛けたもので計算されます。

```
@Override
```

```
public double getArea() {  
    return Math.PI * radius * radius;  
}
```

以上が、このコードの全体的な説明です。

Main.java

このコードは、CSV ファイルから動物の名前と学名を読み込み、ユーザーが入力した動物の名前に対応する学名を出力する Java プログラムです。

まず、[CSVReader](#) クラスのインスタンスを作成し、"動物名.csv"という CSV ファイルを読み込みます。この CSV ファイルは、動物の名前とその学名がカンマで区切られた形式で保存されています。[readCSV](#) メソッドを呼び出すことで、CSV ファイルの内容を [CSVItem](#) オブジェクトのリストとして読み込みます。

次に、読み込んだ [CSVItem](#) のリストを [Reader](#) クラスのインスタンスに渡します。この [Reader](#) クラスは、動物の名前から学名を取得するためのメソッドを提供します。

その後、[Scanner](#) クラスを使用してユーザーからの入力を受け付けます。ユーザーが動物の名前を入力すると、その名前がカタカナであるかどうかをチェックします。もしカタカナでなければ、[NotKatakanaException](#) がスローされ、エラーメッセージが出力されます。

カタカナであれば、[getScientificName](#) メソッドを使用して、入力された動物の名前に対応する学名を取得します。学名が存在すればそれを出力し、存在しなければ該当する動物が見つからなかったというメッセージを出力します。

最後に、finally ブロックで [Scanner](#) オブジェクトを閉じます。これは、リソースリークを防ぐための重要なステップです。

アップです。

CSVItem.java

このコードは、[CSVItem](#) というクラスを定義しています。このクラスは、動物の名前と学名を 1 セットとして保持することができます。具体的には、[animalName](#) と [scientificName](#) という 2 つのパブリックな文字列フィールドを持っています。

コンストラクタ [CSVItem\(String animalName, String scientificName\)](#) は、動物の名前と学名を引数として受け取り、それぞれを対応するフィールドに設定します。このとき、`this` キーワードを使用して、クラスのフィールドと引数の名前が同じ場合でも、それぞれを区別しています。

CSVReader.java

このコードは、[CSVItem](#) というクラスを定義しています。このクラスは、動物の名前と学名を 1 セットとして保持することができます。具体的には、[animalName](#) と [scientificName](#) という 2 つのパブリックな文字列フィールドを持っています。

コンストラクタ [CSVItem\(String animalName, String scientificName\)](#) は、動物の名前と学名を引数として受け取り、それぞれを対応するフィールドに設定します。このとき、`this` キーワードを使用して、クラスのフィールドと引数の名前が同じ場合でも、それぞれを区別しています。

また、このクラスには [getAnimalName\(\)](#) と [getScientificName\(\)](#) という 2 つのパブリックなゲッターメソッドがあります。これらのメソッドは、それぞれ動物の名前と学名を返します。これにより、クラスの外部からこれらのフィールドの値を安全に取得することができます。ただし、このコードではフィールドがパブリックに設定されているため、クラスの外部から直接アクセスして値を変更することが可能です。これは、オブジェクト指向プログラミングの原則に反しています。フィールドをプライベートに設定し、値の変更は専用のメソッドを通じてのみ行うようにすることで、データの整合性を保つことができます。

NotKatakanaException.java

このコードは、[NotKatakanaException](#) というカスタム例外クラスを定義しています。このクラスは Java の組み込みクラスである `Exception` クラスを継承しています。これにより、このクラスは例外を表現するための基本的な機能を継承します。

[NotKatakanaException\(String message\)](#) というコンストラクタは、エラーメッセージを引数として受け取り、スーパークラスのコンストラクタに渡します。これにより、例外がスローされたときに表示されるメッセージを設定することができます。

このクラスは、カタカナ以外の文字が入力されたときにスローするためのものです。例えば、ユーザーからの入力をカタカナのみに制限したい場合などに使用できます。このようなカスタム例外を作成することで、特定のエラー状況をより具体的に表現し、エラーハンドリングをより柔軟に行うことができます。

Reader.java

このコードは、[Reader](#) というクラスを定義しています。このクラスは、CSV ファイルから取り出した情報を [CSVItem](#) オブジェクトに変換する役割を持っています。[CSVItem](#) は、動物の名前とその学名を保持するオブジェクトと考えることができます。

[Reader](#) クラスは、`List<CSVItem>` 型の [items](#) というフィールドを持っています。これは、[CSVItem](#) オブジェクトのリストを保持します。[Reader](#) クラスのコンストラクタは、この [items](#) フィールドを初期化します。

[getScientificName\(String animalName\)](#) メソッドは、指定された動物の名前に対応する学名を返します。このメソッドは、[items](#) リストを走査し、動物の名前が一致する [CSVItem](#) オブジェクトを探します。一致するオブジェクトが見つかった場合、そのオブジェクトの学名

を返します。ただし、動物の名前がカタカナでない場合は、[NotKatakanaException](#) をスローします。これは、動物の名前がカタカナであることを強制するためのエラーチェックです。[isKatakana\(String s\)](#)メソッドは、与えられた文字列が全てカタカナであるかどうかをチェックします。文字列を文字配列に変換し、各文字がカタカナの Unicode 範囲に含まれるかどうかをチェックします。全ての文字がカタカナであれば true を、そうでなければ false を返します。