7　その他の知識：パッケージ・例外処理・いくつかのキーワード

7.1　package

Java 言語では、1 つのクラスに対して 1 つのファイル、という体制で開発を進めます。大きなプログラムでは、クラスが数百個、という状況も発生します。これを全て一括管理するのは大変なので、クラスをいくつかのグループに分類して保管する仕組みが必要になります。それはちょうど、たくさんのファイルを、ディレクトリで分類して保管するのと同じような仕組みに相当します。

パッケージとは

Java 言語では、**パッケージ**という考え方により、クラスを分類することができます。

例えば、いくつかのクラスを、a という名前のパッケージに格納したとします。このとき、a に格納されたクラスの 1 行目には、

```
package a;
```

というように記載する必要があります。

ディレクトリと同様に、パッケージも二重三重に作成することが可能です。例えば a というパッケージの中に b というパッケージを作成したとします。このとき、親パッケージを a とし、子パッケージを b とすると、a と b をピリオドで連結して a.b というように記載します。このような場合には、a.b に所属するクラスの 1 行目には、

```
package a.b;
```

というように記載する必要があります。

パッケージとディレクトリの関係

Java 言語では、自分でパッケージをつくって、自分でつくったクラスを管理するときには、パッケージとディレクトリに対応関係がとれていないといけません。このときの対応関係について、以下の 3 つのルールがあります。

ルール 1:　パッケージに属するクラスのファイルは、そのパッケージの構造と同じ構造のディレクトリに属する必要があります。例えば a というパッケージに属するクラスのファイルは、a というディレクトリを作り、その中に置かなければなりません。例えば a.b というパッケージに属するクラスのファイルは、a というディレクトリの中に b というディレクトリを作り、その中に置かなければなりません。

ルール 2:　パッケージに属するクラスを実行するには、パッケージに対応するディレクトリの上のディレクトリから実行しなければならない。例えば program というディレクトリに a というディレクトリを作り、a というパッケージに属するクラスのファイルを置いたときには、java 命令は program というディレクトリから実行しなければならない。

ルール 3:　java 命令では、パッケージ名をつけてクラス名を表現する必要がある。例えば a.b というパッケージに属する Hello というクラスを実行するときは、

```
java a.b.Hello
```
というように実行する必要がある。

7.2 import

Java 言語では、同一のパッケージに属するクラスどうし、あるいはどのパッケージにも属さないクラスどうしは、実際に同じディレクトリにファイルが置いてあれば、クラス名を指定するだけで互いに参照できます。

例えば 4.4 節で紹介した BmiMethod1 クラスは、同じディレクトリに置いてある BmiMethod2 クラスを、クラス名を指定するだけで参照できています。

しかし異なるパッケージに属するクラスを参照する際には、そのクラスがどのパッケージに属するかを含めて指定しないといけません。

例えば 4.4 節で紹介した BmiMethod2 クラスが、a.b というパッケージに属していたとします。この BmiMethod2 クラスを BmiMethod1 クラスから参照する際には、a.b.BmiMethod2 というように、パッケージ名とクラス名をピリオドで連結して表記しないといけません。

でも、毎回いちいちパッケージ名を記載するのは面倒くさいので、最初にあらかじめ、そのクラスがどのパッケージに属するものであるかを宣言する構文が生まれました。それが **import** 文です。

import 文の記述方法

一例として、java.io というパッケージの中に入っている InputStreamReader というクラスを利用する可能性があるときには、プログラムの冒頭に

```
import java.io.InputStreamReader;
```

と記述します。この記述によって InputStreamReader というクラスは、それ以降はパッケージ名を指定しなくても使えるようになります。

また別の import 文の例として、6.2 節に紹介したものを説明します。

```
import java.io.*;
```

この記述は、java.io というパッケージに属する全てのクラスを利用する、ということを意味します。ここで「*」は「全ての」という意味で利用されます。このような表現を、**正規表現**といいます。

6.2 節のプログラムでは、InputStreamReader および BufferedReader というクラスが使われていますが、これらはいずれも java.io というパッケージに属するものです。

最初から使えるパッケージ

ところで、java.io というパッケージは、皆さんが自分で作った覚えはないだろうと思います。このように Java 言語では、皆さんが自分で作っていなくても最初から使えるパッケージが、いくつか用意されています。本資料では java.io の他に、8.1 節のプログラムにて、java.util というパッケージが import されています。

さらには、import しなくても最初から使えるパッケージとして、java.lang というパッケージが用意されています。典型的な例として、これまでに何度も出てきた以下のクラスは、java.lang パッケージの中にありますので、import 文を一切書かなくても最初からクラス名を指定するだけで利用できます。

表 8: java.lang パッケージに属するクラスのうち本資料で用いるもの。

クラスの名前	クラスの説明
String	文字列を扱うクラス。
Integer	整数に関する処理のためのクラス。
	本資料では Integer.parseInt() などを用いる。
Double	浮動小数点数に関する処理のためのクラス。
	本資料では Double.parseDouble() などを用いる。
Math	数式演算に関するクラス。
	本資料では Math.rand() などを用いる。
System	コンピュータシステムに関わるクラス。
	本資料では System.out.println() などを用いる。

7.3 例外処理：try と catch

プログラムを実行しているときに、いろいろな例外が起こることがあります。例えば「整数をゼロで割ろうとした」「配列の大きさ以上の添え字を使おうとした」といったものが典型的な例外です。一般的にプログラムは、例外が起こると実行を停止してしまいます。

Java 言語では、例外が起こったときに、実行を停止しない代わりに、その例外から正常な状況に戻るための処理をさせる、というプログラミングが可能です。このような処理を **例外処理** といいます。

例外処理のための構文に、**try - catch 文** があります。 この文では、

○○という処理中に、△△という例外が発生したら、□□を実行する

という処理を、

try { ○○ } catch(△△) { □□ }

というように記述します。

6.2節のプログラムを例にすると、try の後の { から } までの間で例外が発生すると、その後の catch(Exception e) にて Exception クラスの変数 e に例外の内容が記録されます。そしてその後の System.out.println(e); にて、例外の内容が標準出力されます。

```
...
     try {
       InputStreamReader isr = new InputStreamReader(System.in);
...
     }
     catch (Exception e) {
       System.out.println(e);
     }
...
```

例外処理が必須である場合・必須でない場合

例外には「例外処理を記述しなくてもいいもの」と「例外処理を記述しなくてはいけないもの」があります。

一例として、以下のようなメソッドは、例外処理を記述しなくてはなりません。

- BufferedReader クラスの readLine() メソッド、close() メソッド（6.2 節参照）
- BufferedWriter クラスの write() メソッド、flush() メソッド、newLine() メソッド（8.2 節参照）

このようなメソッドを実行する部分は、try 文に続いて { と } で囲む必要があります。

7.4 public と private

Java 言語では、クラス、メソッド、変数の「公開度」を設定することができます。この公開度とは、異なるクラスやパッケージから、そのクラス、メソッド、変数がアクセス可能であるかどうか、を示すものです。

この設定のために Java 言語では、**アクセス修飾子**という用語を使います。ここではその代表例として、publuc と private を紹介します。

public

いままで多くのプログラムにおいて、多くの場面で public という単語が使われてきましたが、長らく説明を省略していました。

public という単語は、クラスやメソッドの名前の接頭に用いられてきました。これらとは別に、クラスやメソッドだけでなく、変数の名前の接頭にも用いることができます。

表 9: public を接頭に用いる例。

クラス	public class Bmi
メソッド	public static void main(String args[])
変数	public int i = 1;

public とは、そのクラス、メソッド、変数が、たとえ違うパッケージからでも自由に呼び出せることを表しています。ただし、変数に public を指定することは、あまり推奨されていません。

private

public の代わりに private をつけると、メソッドや変数を「そのクラス内からしか呼び出せない」という状態になります。private は、クラス外部に対して最も隠蔽された状態を表します。

public も private も指定しなければ

public も private も指定していないメソッドや変数は「同じパッケージ内の別のクラスからのみ呼び出せる」という状態になります。

7.5 static

いままで説明してきたプログラムでは、クラスを利用する際には、new 命令などを用いてインスタンスを生成してきました。例えば 4.4 節で紹介したプログラムでは、BmiMethod2 クラスを使用する際に、

```
BmiMethod2 method2 = new BmiMethod2();
```

というように、new 命令によって BmiMethod2 クラスのインスタンスを生成し、それを変数 method2 に代入していました。

これに対して、インスタンスを生成しないで変数やメソッドを利用する方法もあります。この場合には **static** という修飾子を使う必要があります。

一例として、4.4 節で紹介した BmiMethod1 クラスを、以下のように書き換えてみてください。

```
public class BmiMethod1 {
   public static void main(String[] args) {
     BmiMethod2.weight = 80.0;
     BmiMethod2.height = 1.80;
     double bmi = BmiMethod2.calculateBmi();
     System.out.println("BMI=" + bmi);
   }
}
```

さらに BmiMethod2 クラスを、以下のように書き換えてみてください。

```
public class BmiMethod2 {
   static double weight = 67.0;
   static double height = 1.78;
   public static double calculateBmi() {
     double result = weight / (height * height);
     return result;
   }
}
```

そして、この 2 つのクラスをコンパイルしてください。

```
javac BmiMethod1.java
javac BmiMethod2.java
```

そして、main メソッドのある BmiMethod1 クラスを実行してください。

```
java BmiMethod1
```

どうでしょうか？ ちゃんと実行できましたでしょうか？

実は今までにも使われている static

実はいままでにも本資料では、static を使って定義された多くのメソッドを使っています。例えば、

```
Integer.parseInt();
Double.parseDouble();
```

はそれぞれ、Integer クラス、Double クラスの中で static を使って定義された parseInt() メソッド、parseDouble() メソッドを表しています。また、

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

は、System クラスの中に static を使って定義された out という変数があり、その中で static を使って定義された println() というメソッドがある、ということを表しています。