Javaプログラミング講座

javaファイルの書式

・パッケージ宣言: package フォルダ.フォルダ…;

・インポート宣言: import フォルダ.フォルダ…;

・クラス(またはインターフェース, 列挙型)の定義

パッケージ: プログラムを分類して管理するための仕組み。

(実質的にはフォルダ(ファイルシステム)として管理)

パッケージ宣言: 定義したクラスが、どのパッケージに属するものなのかを宣言する。

インポート宣言: 異なるパッケージに属するクラスを使用したい場合に指定する。

Mainクラスの定義:

public class クラス名 {

public static void main(String[] args) {

// 処理

}

}

基本的にこれをいじっていく。

式(文)

・定義文(宣言, 初期化[定義式+代入式としても])

修飾子 型 変数名;

・代入式

変数名 = インスタンス/値;

・初期化式

修飾子 型 変数名 = インスタンス/値;

・処理式(命令, 実行, 演算)

(クラス.)関数(メソッド)名;

・ブロック文(複文): 複数の処理が一つの文として書ける。

※変数の名前空間に注意: ブロックの外からはその変数を参照、利用できない。(未定義と同じになる)

{ //処理 }

・条件式[boolean](論理, 判定, 評価)

boolean型を返すメンバ

・if-else文

if (条件式) { //処理 }

else if (条件式) { //処理 }

…

else { //処理 }

・switch文

switch (変数\*) {

case 値:

//処理

break;

…

default: //処理

}

\*: 整数型, 列挙型, 文字列型

・while文

while (条件式) { //処理 }

・do-while文

do { //処理 } while (条件式);

・for文

for (定義式; 条件式; 代入式) { //処理 }

・for-each文

for (型 変数名 : 変数\*) { //処理 }

\*: Iterableを実装している型の変数

・try-catch文: 例外処理用

・try-catch-resouce文: I/O用

== インスタンス ==

インスタンス: クラスがメモリ上に展開されたもの。

インスタンス化(クラス生成, 初期化):インスタンスを生成すること。下記の文でできる。

型 変数 = new クラス名(引数);

また、インスタンス生成とクラス定義を組み合わせた無名クラス(下記)という構文もある。

<加えて覚えること>

null: インスタンスがないことを表す値。

static: 別記する。

== コンストラクター ==

クラスのインスタンス化が行われる際に呼ばれ、その際に必要な処理を記述するメソッド。

修飾子 class クラス名 {

修飾子 クラス名(引数) {

// 処理

}

}

== 修飾子 ==

public: 全てのクラスから見える

private: 同一クラスの中からしか見えない(他のクラスから見られない)

protected: 同一クラス、およびその継承関係にあるクラス(子クラス)からしか見えない

→ 基本的にはprivateと言われているが、プロジェクト規模にもよるので一概には言えない。

詳細: http://java-code.jp/134

Collection

Collection: 要素のオブジェクトを管理(格納、取得)することを可能にするAPI

・List(リスト)

配列のように格納される要素が順番を持つ。

順番に並んでいるため、リストの要素はindex(インデックス)で参照したり、

追加したりすることができる。

配列(Array)との違い:

要素数が確定していなくてもできる。(配列はListのほぼ下位互換)

・インデックスで管理

・要素は重複が可能

・Set(セット)

格納される要素に順番も重複もない。

・要素は順不同

・要素は重複不可

・Map(マップ)

・Key(キー)とValue(値)のペアで管理

・Keyは重複不可

・Queue(キュー)

・待ち行列として管理

・先頭または末尾の要素のみ追加、削除可能

Iterator(イテレーター, 反復子):

・Collectionの要素を1個ずつ処理するためのクラス

・Iterableインターフェースを実装していなければならない

・拡張for文はこれがあると使える

Generics(ジェネリクス):

・汎用的な型を使うための機能

・クラス名<任意の型> ←これのこと (型とクラスはたぶんほぼ同じ)

Stream(ストリーム): (下記する)

・配列とCollectionを元にして、それらのオブジェクトを集合体として扱えるようにしたAPI

・選ばれた配列やCollectionの要素自体に変更を加えたり、出力したりできます。

・ラムダ式(関数型インターフェース)を使って、楽に配列とCollectionを操作できる。(下記する)

従来はただのfor文, Iteratorを用いて行っていたが、これを用いると簡潔に記述できるようになる。(ex: forEach)

Streamは次の3つの処理の種別に分かれる。

・生成処理: 配列やCollectionのオブジェクトを元にStreamオブジェクトを生成する。

・中間処理: Streamオブジェクトに含まれる要素を選別、加工して、新たなStreamを作る。

なお、この処理はメソッドチェーン(メソッドを続けて記述)することができ、視覚的にもわかりやすい。

・終端処理: 最終結果を取得します。要素数の取得、forEachメソッドの実行がこれに該当します。

なお、終端処理を実行するとそのStreamと元になったStreamはcloseされ、

それ以降の中間処理、終端処理が行えなくなります。(実行すると例外が発生する)

この3つの処理を組み合わせて目的の処理を行います。

さらに、この処理の中ではラムダ式を用いることができ、今までより格段に書きやすくなっています。

== 拡張for構文 ==

for (型※1 変数※2 : 変数※3) {

//処理

}

※1: 変数※3の内部の型

※2: 変数※3の内容が順次入る

※3: Iteratableを実装したクラス(Collection関係のクラス)

== ラムダ式 / forEachメソッド ==

Java8で追加された関数型を受けて追加されたもので、便利。

変数※1.forEach((型 変数) -> {

任意の関数(変数);

})

1. 変数の型の省略
2. 変数部分の()の省略 (※変数が一個の時)
3. 複文(ブロック文)表記の省略 (※処理が一行の時)
4. メソッド参照 (※変数と、任意の関数の引数が対応している時?)

詳細: Javaラムダ式メモ(Hishidama's Java8 Lambda Expression Memo) –

<http://www.ne.jp/asahi/hishidama/home/tech/java/lambda.html>

== 無名クラス(匿名クラス) ==

new スーパークラス/インターフェース() {

// 継承したメソッドなど

}

クラス名のないクラスを定義できる。これを使うことで新たなクラスを宣言することなく、

メソッド内でクラスの定義や、インターフェースの実装ができる。

なお、特定のスーパークラスを指定しない場合には、Objectクラスを指定する。

(Objectクラスは全てのクラスのスーパークラスに当たるため)

== ラムダ式による無名クラスの置き換え ==

() -> {

// ※(参照)

}

※: 抽象メソッドが1つだけ定義されているインターフェース(関数型インターフェース)のメソッドを

実装することができ、従来の無名クラスをつかっていた処理がシンプルに記述できるようになる。

== static(静的) ==

static: インスタンス生成しなくても、そのメソッドや変数にアクセスできるようになる。

しかし、全てのインスタンスに影響を及ぼす。たとえば、あるクラスに定義されている

staticフィールド(変数)と非staticなフィールド(変数)の違いは、次のようになる。

class A {

public static String name = “A”;

public String cat = “C”;

public void test() {

A.name; // printlnとかで出力する

cat; //printlnとかで出力する

}

}

class Main {

public static void main() {

A.name; //printlnとかで出力する

A a = new A();

a.cat; // printlnとかで出力する

a.test();

}

}

詳細: <https://java-reference.com/java_basic_static.html>

また次のようなものも併せて紹介する。

class Main {

static String text;

static {

text = LocalDateTime.now().toString();

System.out,println(text);

}

}

Mapの初期化: <http://daichan4649.hatenablog.jp/entry/20090611/1244650579>

なんかおもしろいの:

<http://d.hatena.ne.jp/satosystems/20120905/1346805620>, <https://ts0818.hatenablog.com/entry/2018/06/02/194653>

File I/O関係

ファイルに書き込みたい！ファイルを読みたい！とかいうことをする。

・FileIOってなんだよ？

FileSystemのI (Input / 入力) と O (Output / 出力) を行えるよ！

・FileSystemってなんやねん！！

FileSystemっていうのは、OSにおいてデータを保管するファイル(File)とそれをまとめ上げて管理するディレクトリー(Directory/Folder)で作られる世界のこと。

・FileIOで何ができるん？

ファイルとディレクトリーに対して、delete, create, exist,…,write, readなどいろいろ、

私達がOSで行うこと(たぶん)すべてができる！

・じゃぁJavaでそれを行うにはどうすればいいの？

→NewIOというライブラリがおすすめですよ！

旧来はFile(java.io系)というクラスを中心として構成されていたが、現在では様々な利点から、

New I/O(java.nio系)というパッケージを用いて行うのがこれからの主流となっていきます。

<New I/O>

NewIOでは一般的に次の2つのクラスを基底として使います。

Path: Stringを変換してファイルのパスを表すようにしたクラス

Paths: Pathを作るためのクラス

Files: 実際にファイル操作をするクラス

※FileクラスはNewIOではなく、古のもの(?)のため注意すべし。

・Pathの取得(例) … ファイルの場所！

Path path = Paths.get(“C:\\”, “a”);

※Pathはnewでは作れません！

※Pathsのgetメソッドはstaticメソッドです！

※getメソッドの引数は、可変長引数で、引数ごとで、ファイルの階層を表すことができます。

※パス(普通に文字列)の取得は、ファイルを選択して、Shift+右クリックで「パスのコピー」を行うと楽。

※パスは、エスケープシークエンスとダブルクオーテーションに気をつける。

・Filesの利用の例 … ファイル/ディレクトリーに対する操作を行う。

Files.exsit(path);

※戻り値はbooleanなのでこれをif文で使うことができる！

●ファイルへの読み書き

基本的に次の2つのクラスを使うと良い。(ほかのクラスは基本的に古のものが多いが、他人のプログラムでは古でも使われることが多いので注意？してください)

BufferedWriter: バッファを用いて効率的に、ファイルに書き込みを行うクラス

BufferedReader: バッファを用いて効率的に、ファイルに読み込みを行うクラス

・Buffer(バッファ)って？

メモリ空間上に一時的に置かれたデータのこと。tmp/temp(テンプ)ともいいます。

●BufferedWriter/BufferedReaderの(インスタンスの)生成

<古の方法/自分もよくわからん>

たぶん、(NewIOでは)あんまり使わないです。

BufferedWriter bw = new BufferedWriter(Writer);

BufferedReader br = new BufferedReader(Reader);

<NewIOを用いた、BufferedWriter/BufferedReaderの生成>

BufferedWriter bw = Files.newBufferedWriter(path);

<BufferedWriter/BufferedReaderの利用>

bw.write(String);

String line = br.readLine();

※bw, brはそれぞれclose()メソッドを呼び出さないと行けない。

(bwでclose()を呼び出さないと出力されない)

よって以下のようにする必要がある。

try {

BufferedWriter bw = Files.newBufferedWriter(path);

bw.write(“A”);

bw.close();

} catch (IOException ex) {

ex.printStackTrace();

}

クソ長い

↓

<BufferedWriter/BufferedReaderの利用(改良) ～ try-catch-resouce！>

try (BufferedWriter bw = Files.newBufferedWriter(path)) {

bw.write(“A”);

}

神

<write()の注意？>

br.write(text)を呼び出しただけでは、改行されず、

AAAAAAA

みたいなことになってしまう。

改行するには、行と捉える文字列の最後に改行文字をつければ良い。

bw.write(text + System.lineSeparator());

※System.lineSeparator();はOS毎に正しい改行文字を返してくれるメソッド。

CRLF(\r\n), LF(\n)とかの差異に気をつけなくていい！

<課題>

1. ファイルに文字列を書き込む
2. 「1.」の複数行version (Listを用いて)
3. ファイルの読み込み
4. 「3.」の複数行version (Listを用いて行の表現), なお全ての行でも良いし、任意の範囲の行でも良い。

<コラム>

私達が今まで使ってきたいつもの構文

System.out.println(“Hello world!”);

には実は、FileIOもといStreamの概念のようなものが紛れ込んでいる。

実は、System.out (のout)は、OutputStreamなのである！

つまりFileIOと本当に似たようなものなのです。

同様に、System.in というものもあり、これはInputStreamです。

しかしながら、これは考えれば自然で、コンソールに文字を書くことって、実際はファイルに文字を書くこととあまり変わりないですよね？そういうことなのです。

<参考/Reference>

Oracleの公式解説ページ群: https://docs.oracle.com/cd/E26537\_01/tutorial/essential/io/index.html

java.io.File のコードを java.nio.Path と java.nio.Files を使って書き直す:

<https://qiita.com/toastkidjp/items/5500521ff5dc0346c2b1>

Javaのファイル入出力関係のクラス/インタフェースについて整理する:

<https://qiita.com/anqooqie/items/0ac128f52e2e5153f727>

== 明示的な書き方 ==

空白

改行

インデント

処理順

== オブジェクト指向 ==

・クラス

・メンバ

・フィールド

・メソッド

・オブジェクト

(・インスタンス: クラス, オブジェクト)

・static？

クラス関係:

・継承([extends])

class:

・クラスの定義の拡張。

・単一継承。

・親クラスのメソッド、フィールドを利用できる。

・親クラスとほぼ同一として扱われる。(キャスト)

・実装(abstract[extends], interface[implements])

abstract(抽象):

classとほぼ同様だが、抽象である。

・クラスのメソッド、フィールドの実装を定義しない。

・abstract(抽象)クラスは、インスタンス生成できず、

継承クラスで実装されなければならない。

interface(インターフェース):

・機能の宣言。

・クラスのメソッドの実装を定義しない。(フィールドはできる？)

・多重継承可能。

++++最後に参考に+++++

おすすめサイト: <https://manga.crocro.com/?cat=java&pg=index> (漫画でわかりやすくて触りには最高)