Java 入門編まとめ

教科書や授業中に出てきた用語をまとめてみました。

用語	説明	書き方	重要度
変数	値を入れる箱 箱には「型」があり、代表的なものは以下の通り 数値型・・・int 小数点・・・double 文字列・・・String 真偽値・・・boolean 変数は「宣言」、「代入」、「参照」がある 【宣言】変数を使うことをプログラムに予告する 【代入】変数に値を入れる 【参照】変数の中身を見る	【宣言】変数の型 変数名 例）int param; 【代入】変数名 = 値 例）param = 10; 【参照】変数名 例）System.out.println(param);	超◎

if 文
else if 文
	条件を指定して分岐処理を書くときに使用する
	【構文】
	if(質問１){
	　質問１が真のときの処理
	} else if(質問２){
	　質問２が真のときの処理
	} else {
	　それ以外のときに処理
	}
		int param = 1;
		
		if(param == 1){
		　System.out.println(“param は１”);
		}else if(param == 2){
		　System.out.println(“param は２”);
		}else{
		　System.out.println(“それ以外”);
		}
			超◎

for 文	繰り返し処理を行う。主に回数が決まっているときに使用することが多い
	【構文】
	for(初期値; 継続条件; 増減値){
	　何回も行いたい処理
	}
		5 回「こんにちは」と表示したい時
		
		for(int i=0; i < 5; i++){
		　System.out.println(“こんにちは”);
		}
			超◎

配列	変数の集まりを１つの変数として扱える。 配列も変数の１種なので、「宣言」「代入」「参照」がある。通常の変数との違いは以下の通り 【宣言】 配列の個数（要素数）を指定しなければならない new キーワードを使うか、直接値を書くかする。 【代入】 配列の何番目に代入するかを添え字で指定する必要がある	【宣言】変数の型[] 変数名 例）int[] arry = new int[10]; 　int[] arry = { 0,1,2,3,4,5,6,7,8,9 }; 【代入】変数名[添え字] = 値 例）arry [0] = 10; 【参照】変数名[添え字] 例）System.out.println(arry[0]);	超◎

<table>
<tr>
<td></td>
<td>【参照】
配列の何番目を参照するか添え字で指定する必要がある
※添え字は 0 から始まるので注意が必要。</td>
<td></td>
<td></td>
</tr>
<tr>
<td>メソッド</td>
<td>クラスの中で何かの処理を行う「職人さん」のようなもの。Java のプログラムは全てメソッドが実行される。基本的に以下のような構文になる
アクセス修飾子 戻り値 メソッド名(引数){
　　メソッドの処理の中身
}
【アクセス修飾子】
<table><tr><td>public</td><td>公開するという意味。他のクラスからも使って OK の場合につける</td></tr><tr><td>private</td><td>隠すという意味。他のクラスから隠蔽する（使わせない）ときにつける</td></tr><tr><td>protected</td><td>private と public の中間。基本的に他のクラスに使わせないが、派生クラスからは使って OK の場合につける</td></tr></table>【戻り値】職人が返す「結果の値」
メソッドが何らかの処理をして、その結果を返す「型」を指定する。何も返さない場合は「void」を指定する。メソッドを返す場合は返す値の「型」を指定する
【メソッド名】仕様的な決まりは無いが習慣として以下のようにつけることが多い。
・最初の字は小文字で、文節を大文字
・動詞＋名詞の順にする
例えば、名前を取得するメソッドの名前は
　getName
とすることが多い
【引数】職人に渡す「インプット」
つまり、メソッドが処理をするのに必要な情報を受け渡すクチ。書き方はメソッド名の後に（）内に変数の</td>
<td>アクセス修飾子：public（公開）
戻り値：なし
引数：なし
メソッド名：method1
処理内容：「こんにちは」を表示
の場合
　public void method1( ) {
　　System.out.println(“こんにちは”);
　}

アクセス修飾子：private（非公開）
戻り値：整数値
引数：2 つの整数値
メソッド名：method2
処理内容：引数の値を足した数を返す
の場合
private int method2(int p1,int p2) {
　　int total = p1 + p2;
　　return total;
}</td>
<td>超◎</td>
</tr>
<tr>
<td>クラス</td>
<td>ある「モノ」について「情報（フィールド）」と「機能（メソッド）」を持つ「設計図」</td>
<td>public class クラス名{
　　（クラスの中身）
}
例）
public class Person{</td>
<td>超◎</td>
</tr>
</table>

		}	
インスタンス	クラスが「設計図」であることにたいしてインスタンスは「実体」。 基本的にクラスはインスタンスがないと動かない	クラス名 変数名 = new クラス名(); 例） Person p = new Person();	◎

インスタンス変数
（メンバ変数）
	インスタンスごとに作られる変数。クラスのフィールド（情報）にあたる。通常の変数と同じように、宣言・代入・参照する。
	インスタンス毎に別の変数となることに注意
		例）
		【宣言】
		public class Person{
		　private **int param;**
		}
		【代入】
		param = 10
		【参照】
		System.out.println(param);
			◎

インスタンス
メソッド
	インスタンスごとに存在するメソッド。クラスの「機能」にあたる。
	通常のメソッドと同じように戻り値、引数などを持つ。
	メソッドの呼び出しは、
	インスタンス.メソッド名
	で呼び出す
		例）
		【宣言】
		public class Person{
		**　public void method(){**
		**　　・・・メソッドの処理・・・**
		**　}**
		**　public void method2(int data){**
		**　　・・・メソッドの処理・・・**
		**　}**
		**　public int method3(int data){**
		**　　・・・メソッドの処理・・・**
		**　}**
		}
		【呼び出し】
		Person p = new Person();
		p.method();
		p.method2(10);
		int result = p.method3(10);
			◎

コンストラクタ
（構築子）
	インスタンス作成時に自動で呼び出され、主に初期化処理などを行う。
	メソッドと書き方が似ており、引数は指定できるが、名前はクラス名と必ず同じにしなければならない。戻り値は指定できない。処理の内容については通常のメソッドと同じ。
	また、引数違いの異なるコンストラクタを定義することが出来る。
		例）
		public class Person{
		　public Person(){
		　　・・・処理・・・
		　}
		　public Person(int param){
		　　・・・処理・・・
		　}
		}
			◎

スーパークラス サブクラス	継承関係にある親がスーパークラス。子供がサブクラス。	例） public class ClassA{	超◎

	継承は extends というキーワードを使って
	public class サブクラス名 extends スーパークラス名
	のように書く
		}
		public class ClassB extends ClassA{
		}
		この場合、ClassA がスーパークラス
		ClassB がサブクラス	

オーバーライド	スーパークラスが持つメソッド（機能）をサブクラス側で上書きすること。
	実装としては、スーパークラスと同じ　戻り値、名前、引数を持つメソッドを定義すれば OK
		例）
		public class ClassA{
		　public int method1(String str){
		　}
		}
		public class ClassB extends ClassA{
		　public int method1(String str){
		　}
		}
		この場合、ClassB で method1 をオーバーライドしているという
			超◎

クラスメソッド
静的メソッド
static メソッド
	インスタンスがなくても使用可能なメソッド（機能）。宣言については static という修飾子がつく以外は、インスタンスメソッドと同じ。呼び出す際は
	クラス名.メソッド名
	となる。
		例）
		【宣言】
		public class CryptUtil{
		public **static** String encode(String str){
		}
		}
		【呼び出し】
		String str = CryptUtil.encode(“aaaa”);
			○

クラス変数
静的変数
static 変数
	インスタンスが不要な変数。複数のインスタンスがあってもクラス変数は１つだけとなる。
	クラスで定義した定数などで使用したりシングルトン実現のために使われる。
		例）
		public class ClassA{
		　private static int param;
		}
			△

ポリモーフィズム	多態性。様々な性質の振る舞いができるインスタンス。現段階では、スーパークラスの変数にサブクラスのインスタンスが代入できるということを覚えておこう	public class Animal{
		}
		public class Dog extends Animal{
		}
		public class Cat extends Animal{
		}
		
		Animal a1 = new Dog();
		Animal a2 = new Cat();
			◎

抽象クラス	ポリモフィズムを実現するひとつの手順。 abstract キーワードが付いたクラス又はメソッド。 継承を考えた時、親クラス側で実体を持つ必要が	Animal というクラスの子クラスにかならず Walk というメソッドをオーバーライドさせたい場合 public abstract class Animal{	△

	無い場合は抽象クラスにすることある。（しなくてもよい）
	また、作成したクラスのメソッドを必ず子クラスでオーバーライドさせたいときには抽象メソッドを用いることがある（しなくてもよい）
	抽象クラスと子クラスとの関係は「IS」の関係であるべき。
		public abstract void walk( );
		}
		public class Human extends Animal{
		public void walk( ){
		System.out.println(“walk!”);
		}
		}	

インターフェース	ポリモフィズムを実現するひとつの手順。
	クラスとは違い、メソッドの戻り値、引数、名前だけを定義し、処理自体は持たない。
	あるものに対して「規格（ルールブック）」を定義するイメージ。
	interface キーワード使って宣言する。
	インターフェースを使う側は「実装クラス」と呼ばれる。（インターフェース自体は処理が無く規格だけなので、規格に沿って実装するため総呼ばれる）
	実装クラスは implements というキーワードを使って実装する。
	実装クラスでは、インターフェースに定義されたメソッドを必ず実装（処理を記述する）しなければならない。
	インターフェースと実装クラスの関係は「CAN DO」であるべき。
	なお、継承（extends）は１つのクラスしか出来ないが、インターフェースの実装（implementts）は、複数可能
		Swimable というインターフェイスとその実装クラスの書き方。
		
		public interface Swimable {
		public void swim ( );
		}
		public class Flog implements Swimable {
		public void swim ( ){
		System.out.println(“swim!”);
		}
		}
			△

例外	実行時に何らかのエラーが発生した場合に、throw（投げる）するもの。
	投げられた例外を catch（キャッチ）してエラー時の処理を行う。
	例外をキャッチするためには、例外が発生する恐れがある部分を try ブロックで囲む必要がある。
	例えば 「0 の除算」を行うと ArithmeticException という例外が発生するが try で囲っていない場合は、キャッチされず、発生した時点で、プログラムが終了する。
	また、キャッチした場合も、例外が発生した以降のプログラムは実行されないことに注意。
		例）配列の不正アクセスが発生した時に処理を続けたい場合
		int[] arry = new int[3];
		try{
		for(int i = 0; i < 4; i++){
		//3 つしかない配列なのに添え字
		//「3」でアクセスする時に
		//例外（エラー）が発生する
		arry[i] = 0;
		}
		//try ブロック中のココより下の命令は
		//例外発生時には実行されない
		System.out.println(“実行されない”);
		} catch(ArrayIndexOutOfBoundsException e){
			超◎

<table>
<tr><td></td><td></td><td>//例外が発生したのでココに処理が飛ぶ
System.out.println(“例外発生”);
}</td><td></td></tr>
<tr><td>finally 句
（例外）</td><td>例外をキャッチする try – catch には finally ブロックをつけることが出来る。
finally ブロックは、例外発生しようが、しまいが必ず実行される。
用途としては、ファイルをオープンした時に例外が発生した場合もしなかった場合もクローズ（閉じる）処理を行う必要があるときに使う</td><td>try{
int a = 10/ 0; //例外が起こる
//try ブロックのココより下は実行されない
System.out.println(“実行されない”);
}catch(ArithmeticException e){
System.out.println(“例外発生”);
}finally{
System.out.println(“必ず実行される”);
}</td><td>○</td></tr>
<tr><td>例外クラス
（例外）</td><td>例外はあくまで「例外クラス」であり、投げられている例外は「インスタンス」です。
全ての例外クラスは以下の３つから派生しています。それぞれに特徴があります。
<table><tr><td>クラス名</td><td>特徴</td></tr><tr><td>RuntimeException</td><td>catch しなくてもよい例外</td></tr><tr><td>Exception</td><td>catch しなければならない例外</td></tr><tr><td>Error</td><td>catch してはダメな例外</td></tr></table>Exception クラスから派生した例外を catch しなかったり、Error クラスから派生した例外を catch してしまうと、コンパイルエラーになります。</td><td>Java のネイティブな例外クラスも定義を見ると左のいずれかから派生しています。

// ArithmeticException の定義↓
public class ArithmeticException
extends RuntimeException</td><td>◎</td></tr>
<tr><td>throw
（例外）</td><td>例外は、自分で投げる（発生させる）こともできます。その場合は、throw 命令をつかって投げます。あくまで「インスタンス」を投げるので new することを忘れないようにしましょう。
丁寧に書くと
ArithmeticException e =
new ArithmeticException();
throw e;
ですが、通常は省略して
throw new ArithmeticException();
と書きます。</td><td>例）割り算をするときに、割る値が 0 だったら自分で ArithmeticException を投げる例

int a = 10;
int b = 0;

if(b == 0){
//割る値（変数 b）が 0 なので例外を投げる
throw new ArithmeticException();
}
int c = a / b;</td><td>超◎</td></tr>
</table>

例外クラスの自作	例外クラスは自分で作って構いません。 その場合は、RuntimeException、Exception クラスのどちらかを継承して作ります。 （これらのクラスを継承しないと throw できません） 自作の例外クラスを作る意図としては、システムを作る際、独自のエラー処理をしたい場合などです。 例えば、何かの販売サイトを作るとして、購入しようとした時に在庫が無い場合、「在庫が無い」という意味の例外クラス 「NoStockException」を作成して、その例外をキャッチして在庫が無い場合の処理をするときなどです。		◎

catch 句	catch はいくつもかけます。
	例えば、0 の除算の時と配列の不正添え字の参照時で、異なる処理をしたい場合などのときには catch を複数書きます。
		例）0 の除算の時と配列の不正添え字の参照時で、異なる処理をしたい場合
		
		try{
		　//何らかの処理
		} catch (ArithmeticException e){
		　//0 の除算の時の処理
		} catch (ArrayIndexOutOfBoundsException e){
		　//配列の不正添え字参照時の処理
		}
			◎

拡張 for 文	通常の for 文にはカウンタとなる変数がありますが拡張 for ぶんでは不要です。
	配列などを for 文の中で値参照したいときに便利です。
	拡張 for 文の構文
	
	for(要素を受け取る変数の宣言 ： 配列など){
	　処理の中身
	}
		配列を全て表示する場合
		【通常の for】
		int[] arry = { 0,1,2,3 };
		for(int i = 0; i < arry.length; i++){
		　System.out.println(arry[i]);
		}
		
		【拡張 for】
		int[] arry = { 0,1,2,3 };
		for(int elment : arry){
		　System.out.println(elment);
		}
		
		【文字列配列の場合】
		String[] arry = { “a”,”b”,”c” };
		for(String str : arry){
		　System.out.println(str);
			◎

		}	