第二週

クラス、コンストラクタ、変数の型、static

オブジェクト

今回は C にはなかったクラスというものを学びます。Java を学ぶにはオブジェクト指向の考え方を避けては通れません。まずオブジェクトとはどういうものなのかを理解しましょう。

オブジェクトは日本語で「物体」という意味ですが、Java でも同様に「物体」として考えます。オブジェクトは

- 状態（変数）
- 振る舞い（関数）

を持っています。例えば「動物」という**オブジェクト**は、「名前」や「年齢」、「性別」などといった**状態**と、「歩く」や「食べる」、「寝る」などといった**振る舞い**を持っています。また、目に見えるものだけでなく、抽象的な概念までオブジェクトとみなすことが出来ます。

クラス

クラスはオブジェクトの設計書みたいなものであり、オブジェクトの状態や振る舞いを１つの型に集めたものです。Java では、オブジェクトの状態を表す変数をフィールド、振る舞いを表す関数をメソッドといいます。またこれらを総称してメンバといいます。クラスの定義は以下のように記述します。（アクセス修飾子はカプセル化の時に説明します。）

```
アクセス修飾子 class クラス名 {

        フィールド定義

        メソッド定義

}
```

Java では基本的に 1 つの JAVA ファイルに 1 つのクラスを作成し、ファイル名とクラス名は一致させる必要があります。また、クラス名は大文字から始めるルールがあります。

では、実際にクラスを作成しましょう。ただし、クラスを定義してもそれ単体では動作しません。

```
// クラス定義

public class Animal {

        // フィールド定義

        String name;

        int age;



        // メソッド定義

        void method(){

                System.out.println(name+"は"+age+"歳です.");

        }

}
```

Animal **クラス**に String 型の name、int 型の age という**フィールド**と返値の型が void 型の method という**メソッド**が定義されています。

インスタンス

クラスは定義しただけでは動作せず、まずインスタンス（実体）を生成する必要があります。これをインスタンス化といいます。インスタンス化は以下のように行います。

```
クラス名 変数名 ＝new クラス名（引数);

もしくは

クラス名 変数名;

変数名 ＝new クラス名（引数);
```

new 演算子を使って、メモリ領域に割り当てられたインスタンスを格納するための領域の参照情報を代入しています。

クラスをインスタンス化したら、定義したフィールドやメソッドを呼び出すことが出来ます。次のように呼び出すことが出来ます。

```
変数名.フィールド;

変数名.メソッド;
```

構造体のように「.」（ドット）を付けることで扱うことが出来ます。ただし、フィールドにアクセスするのは良くありません。この話は今後の講座のカプセル化でやります。では、実際にクラスを扱ってみましょう。

```
public class Main {

	public static void main(String[] args) {

		// インスタンス化

		Animal dog = new Animal();

		Animal cat = new Animal();



		// フィールドに値を代入

		dog.name = "犬";

		dog.age = 5;

		cat.name = "猫";

		cat.age = 3;

		// フィールドを呼び出す

		System.out.println("名前:"+dog.name+",¥t 年齢:"+dog.age);

		System.out.println("名前:"+cat.name+",¥t 年齢:"+cat.age);



		// メソッドを呼び出す

		dog.method();

		cat.method();

	}

}
```

実行結果は次のようになります。

```
名前:犬, 年齢:5

名前:猫, 年齢:3

犬は 5 歳です.

猫は 3 歳です.
```

このように基本的に他のクラスからオブジェクトを生成して利用します。また、１つのクラスから複数のインスタンスを生成することが出来ます。

コンストラクタ

コンストラクタはインスタンス生成時に自動的に呼び出されるメソッドです。主に初期化などに使われます。コンストラクタは以下のように記述します。

```
クラス名（引数）{

        処理

}
```

コンストラクタの名前はクラス名と同じであり、返値の型を書きません。また、コンストラクタはオーバーロードすることが出来ます。コンストラクタを省略、もしくは引数なしで何もしないデフォルトコンストラクタが生成されます。クラス内に１つでもコンストラクタを定義するとデフォルトコンストラクタは生成されません。では、先ほどのプログラムをコンストラクタを使って実装してみましょう。

```
public class Animal {

	// 以下を書き加える

	// コンストラクタ

	Animal(String name, int age){

		this.name = name;

		this.age = age;

	}

}
```

```
public class Main {

	public static void main(String[] args) {

		Animal dog = new Animal("犬", 5);

		Animal cat = new Animal("猫", 3);



		System.out.println("名前:"+dog.name+",¥t 年齢:"+dog.age);

		System.out.println("名前:"+cat.name+",¥t 年齢:"+cat.age);



		dog.method();

		cat.method();

	}

}
```

Animal クラスのコンストラクタ内に用いられている **this キーワード**は、そのオブジェクト自身を指示している。例えば、this.name で Animal クラスの name であるということを明示している。これはメソッドでフィールドと同名の変数が引数として使われている場合、区別する時などで使われる。

プリミティブ型とオブジェクト型

Java には大きく分けて２つの型が存在します。

- プリミティブ型（基本データ型）

メモリに格納されているデータそのものを表す型です。boolean、char、byte、short、int、long、float、double の８つがこれにあたる。プリミティブ型は先頭文字が小文字である。

- オブジェクト型（参照型）

メモリに格納されているデータの場所を表す型です。プリミティブ型以外はすべてオブジェクト型にあたります。String 型や配列型などもオブジェクト型です。

プリミティブ型は変数を代入する時は、受け渡し側と受け取った側がそれぞれ別のデータを持つ。一方、オブジェクト型の変数を他のメソッドを渡すときは、渡された側が受け渡し側の実態を操作できるようになる。以下にプログラムを確認してみましょう。

```
public class Main {

	public static void main(String[] args) {

		// プリミティブ型は値渡し

		int x = 0;

		System.out.println(x);

		int y = x;

		y = 10;

		// 渡した側のデータの値は変わらない

		System.out.println(x);



		// オブジェクト型は参照

		Animal dog = new Animal("犬", 5);

		dog.method();

		Animal wolf = dog;

		wolf.name = "狼";

		// 渡した側のデータの値が変わっている

		dog.method();

	}

}
```

実行結果は以下の通りである。

```
0

0

犬は 5 歳です.

狼は 5 歳です.
```

int 型の x を int 型の y に渡して、y の値を変えても x の値に変化はありませんでした。しかし、Animal 型の dog を Animal 型の wolf に渡して、wolf のフィールド name の値を変えたら dog の name が変化してしまいました。

static 修飾子

static なフィールドやメソッドは、インスタンスごとはなくクラスに対してただ 1 つだけ生成されます。static なフィールド（**クラス変数**）やメソッド（**クラスメソッド**）を**静的メンバ**といいます。ちなみに static でないフィールドを**インスタンス変数**、メソッドを**インスタンスメソッド**といいます。静的メンバは以下のように定義します。

```
static 型 フィールド名

static 型 メソッド名(引数) {

        処理

}
```

また、静的メンバは以下のように呼び出します。

```
クラス名.フィールド名;

クラス名.メソッド名(引数);
```

静的メンバを利用する際の注意として、static なメソッドからクラス内の非 static なメソッドを呼び出せません。また、static なメソッドからクラス内の非 static なフィールドを操作できません。では以下のプログラムを実行してください。

```
public class Animal {

	// 以下を追加する

	static int num = 0;		// クラス変数



	// クラスメソッド

	static void printNum(){

		System.out.println("現在の動物の数は"+num+"匹です.");

	}

}
```

```
public class Main {

	public static void main(String[] args) {

		Animal.printNum();		// クラスメソッドの呼び出し

		Animal dog = new Animal("犬", 5);

		Animal.printNum();

		Animal cat = new Animal("猫", 3);

		Animal.printNum();

	}

}
```

以下に実行結果を示す。

```
現在の動物の数は 0 匹です.

現在の動物の数は 1 匹です.

現在の動物の数は 2 匹です.
```

演習問題

1. 次の要件を満たす Character クラスを実装しなさい。

- 以下のフィールドを持つ。
    I. 名前(name)：String 型
    II. 体力(hp)：int 型
    III. 攻撃力(power)：int 型
- 以下のメソッドを持つ。
    I. 名前と体力と攻撃力の値を表示する(status)：void 型
- 以下のような実行結果になる。（値は好きに決めてよい）

名前　：プレイヤー 体力　：100 攻撃力：20

2. 1 で作成した Character クラスをコンストラクタを用いて名前と体力と攻撃力の初期値を設定できるようにつくりかえなさい。
3. 文字列型の要素数 5 配列 strs1 を作成し、同様の配列 strs2 に strs1 の値を代入し、strs2 に辞書順に並び替えなさい。ただし、並び替えた際に strs1 の値が変わってはいけない。（String クラスに 2 つの文字列を辞書的に比較するメソッドがあるので、このような Java に用意されているクラスやメソッドを利用しましょう）