OPEN STREAM

212.オブジェクト指向(1)
オブジェクト指向・クラスとインスタンス

Make IT your Success

目次（1/2）

目次（2/2）

この講義について

目的：

オブジェクト指向の考え方や、基本的な用法

- クラスとフィールド、メソッドとは何か
- クラスとインスタンスの違い
- カプセル化

について学ぶ。

ゴール：

オブジェクト指向の基本概念を理解し、クラスとインスタンスを用いたプログラムが書くことが出来る。

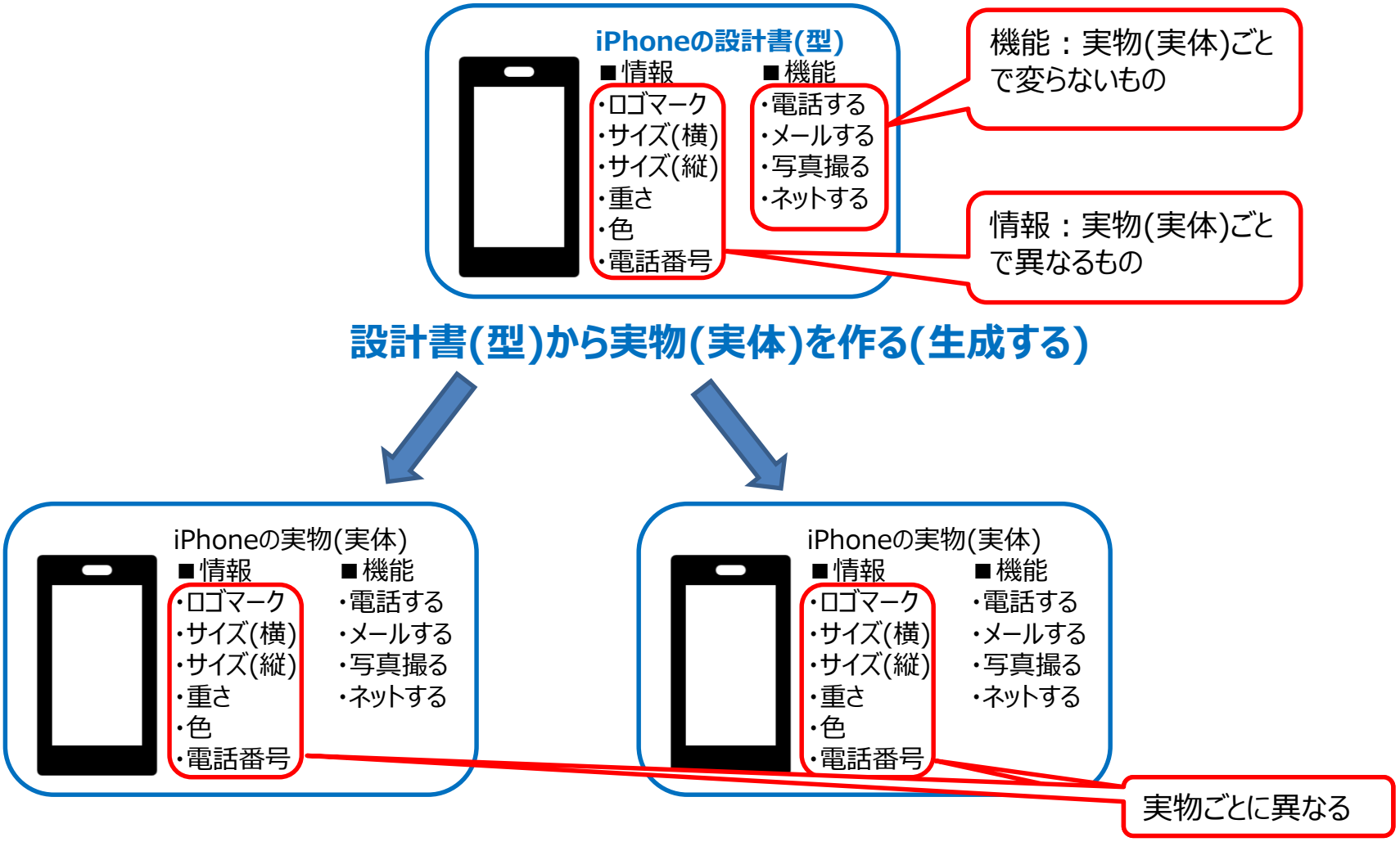
オブジェクト指向

私たちが作成するプログラムは現実世界とかけ離れたものではない。(手作業で行っているものをシステム化するとか、自然現象をシミュレートするシステムを作るとか、現実世界の仕事や仕組みをシステムに落とし込む事が多い)

現実世界にあるものを模して作成した方が、作成しやすく理解しやすい。現実世界のようなものを表現するのにオブジェクト指向は都合が良い。

オブジェクト指向

スマートフォンで例えると



オブジェクト指向

スマートフォンの型を**クラス**、クラスから生成された実体を**インスタンス(オブジェクト)**という。

クラスは**属性(フィールド)**と**機能(メソッド)**を持つ。

※ 前ページの情報→属性

オブジェクト指向

このような考えでプログラム開発を進めていく考え方が
オブジェクト指向プログラミング(OOP:object-oriented programming**)**という。

オブジェクト指向プログラミングには以下の特徴がある。

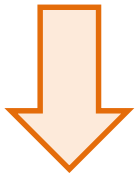
① カプセル化　　**本日の講義**で解説する。
② 継承
③ 多態性

実際の開発

Javaでは複数のクラスを組み合わせて、１つのプログラムを作成する。

研修でもクラスを作成しますが、自らが作成するクラスだけで完結することはない。

Javaが提供するクラス群や第三者が提供するクラス群と、自らが作成するクラスを組み合わせて作成することになる。



よって

クラスの作り方、クラスの使い方を覚える必要がある。

クラスの構成要素

オブジェクト指向プログラミング(OOP)の基本的な構成要素である、クラス(設計書(型))の構成要素を解説する。

① クラス・・・・・・・・・・・・・インスタンスの設計書。雛形。
② フィールド・・・・・・・・・・情報（データ）
③ メソッド・・・・・・・・・・・・機能（データの操作など）
④ アクセス修飾子・・・・アクセス指定

クラス - クラスの構成要素

クラス

オブジェクト指向の説明でもあったようにクラスは設計書（型）であり、
属性（フィールド）と機能（メソッド）を持つ。
※ 情報→属性

書き方：

```
アクセス修飾子 class 識別子 {
    ・・・・・・
    ・・・・・・
}
```

※ アクセス修飾子は後述。

フィールド

クラスの属性を変数・定数で記述する。

書き方：変数

```
アクセス修飾子 class 識別子 {
    アクセス修飾子　データ型　識別子；
    ・・・・・・
}
```

定数

```
アクセス修飾子 class 識別子 {
    アクセス修飾子　final データ型　識別子；
    ・・・・・・
}
```

※static(クラス変数、クラス定数)に関しては、後の講義で解説する。

クラス - クラスの構成要素

メソッド

クラスの機能を記述する。

書き方：

```
アクセス修飾子 class 識別子 {
    アクセス修飾子　データ型　識別子；
    アクセス修飾子　戻り値のデータ型　識別子(引数のデータ型　識別子、・・・・) {
        処理
        return ……　// 戻り値がある場合return文を入れる
    }
    ・・・・・・
}
```

※メソッドに渡される変数のことを引数という。

※フィールドとメソッドのことをあわせてメンバと表現することもある。

※メソッドについては同じ名称のものを作成しても良い。

→ シグネチャ(メソッド名と、変数の数と変数の型が同じ)が異なる必要がある。

→ オーバーロードについては今後の講義で行う。

メソッドと引数

データ型・・・・・・「基本型（プリミティブ型）」と「参照型（オブジェクト型）」がある。

メソッドに渡される変数が「基本型（プリミティブ型）」である場合：
メソッドの中で値を変更しても呼び出し元の方は変わらない。**値渡し**

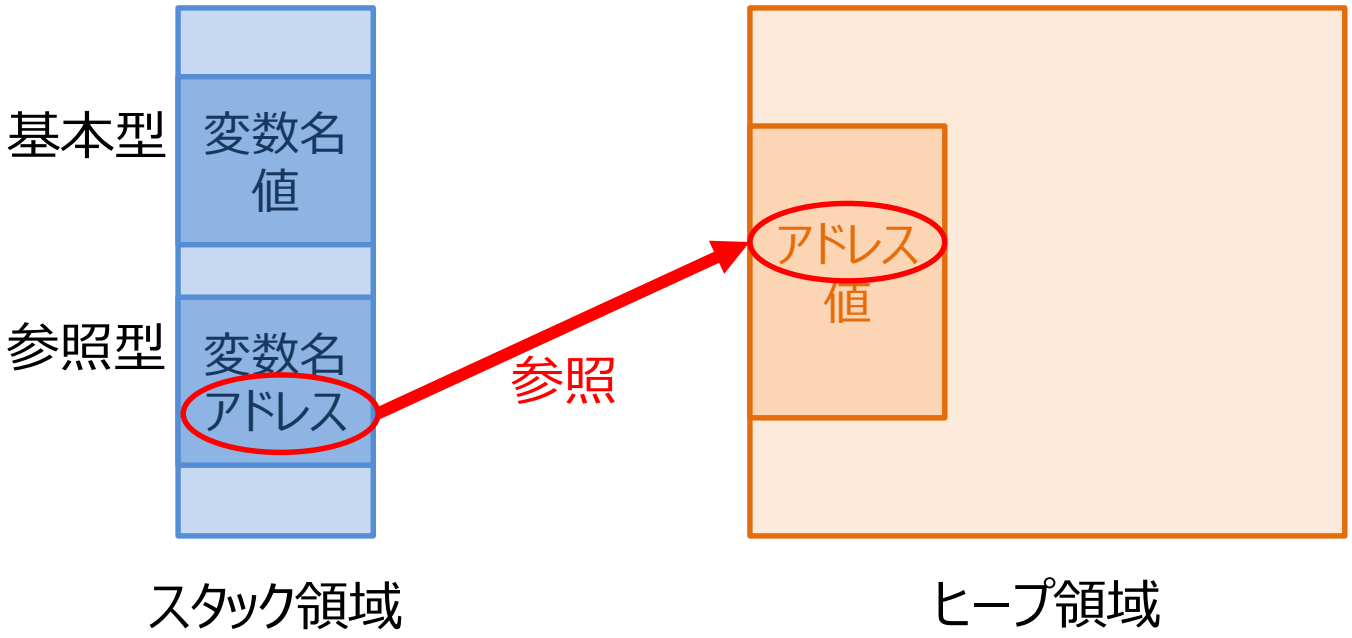
メソッドに渡される変数が「参照型（オブジェクト型）　」である場合：
メソッドの中で値を変更すると呼び出し元の方でも値が変わる。**参照渡し**

復習：
スタックとヒープ
　→引数として何がメソッドに渡されているのか

復習：変数が保持される場所

「基本型（プリミティブ型）」と「参照型（オブジェクト型）」で保持され方が異なる。

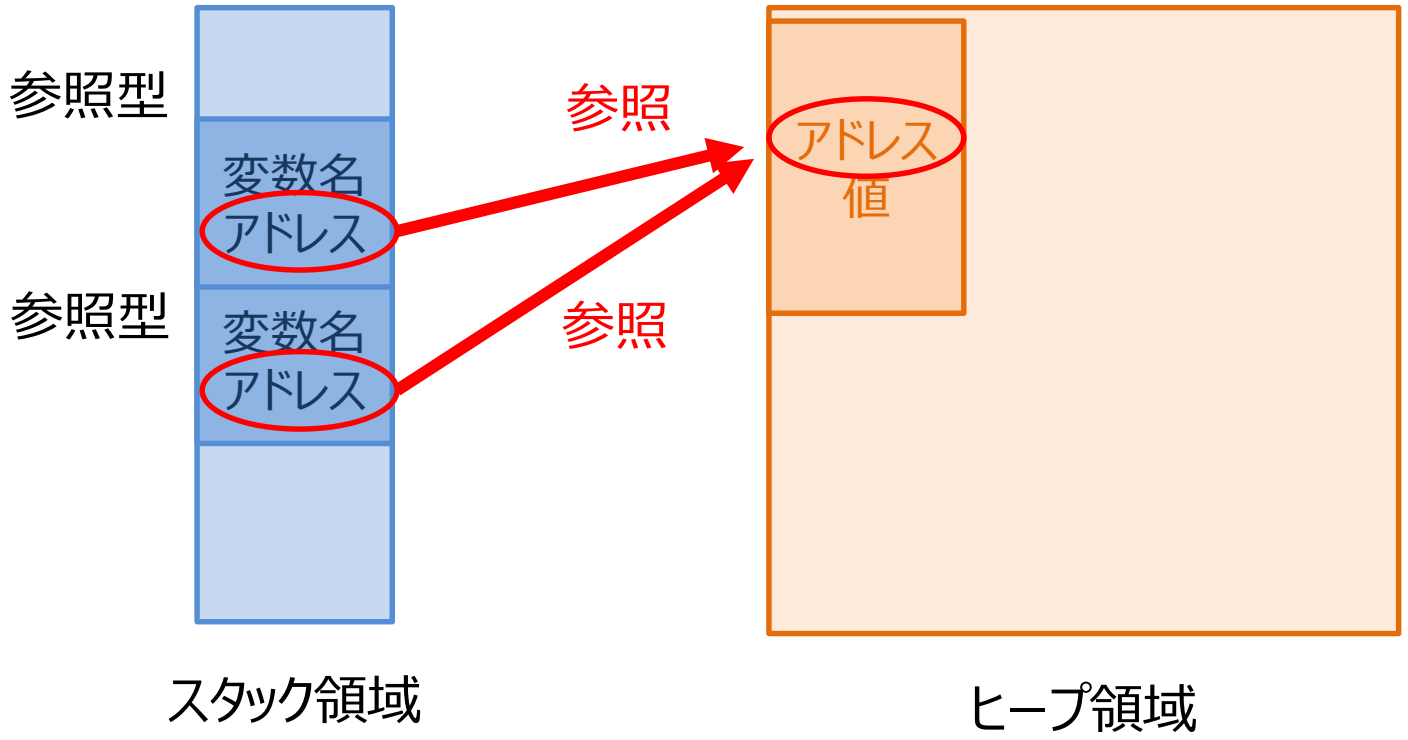
「基本型（プリミティブ型）」→スタック
「参照型（オブジェクト型）」→スタック＋ヒープ



復習：変数が保持される場所

参照渡し

値ではなく参照(アドレス)を渡す為、別の変数で同じ参照先になる



アクセス修飾子

クラス・フィールド・メソッドに「アクセス修飾子」と出てきたが、「アクセス修飾子」はアクセス出来る範囲を指定する。

アクセス出来る範囲が広い方から以下となる。

① public　どこからでもアクセス出来る。（アクセス制限無し）
② protected 同じパッケージかサブクラスからアクセス出来る。
③ 指定無し　同じパッケージのみアクセスできる。（package private とも言う）
④ private　同じクラスのみアクセス出来る。

※　サブクラスの説明は後ほど。
※　トップレベルクラスに付けられるのは「public」と「指定なし(package private)」のみ。

基本的にアクセス出来る範囲が最も狭いprivateを利用して、外部に公開するものだけをpublicにする。

→オブジェクト指向の特徴で挙げた**カプセル化**。
→なぜカプセル化するのか？

クラス - クラスの構成要素

特殊なメソッド **アクセサ**

アクセスを許可していないフィールドから値の取得、値の設定を行うメソッド。

値の設定：setter
値の取得：getter

アクセサの命名規則。

クラス - クラスの構成要素

特殊なメソッド **main**

今まで何気なく使っていたmainメソッドは、クラスを指定して実行すると、

そのクラスのmainメソッドが実行される。

Javaではクラスのstaticな「main」というメソッドをエントリーポイント（一番最初に呼ばれるメソッド）としている。

```
public static void main(String[] args){
    ・・・・・・
}
```

コマンドライン引数をString型の配列argsで受け取る。

可変長引数

引数の型の直後に「...」（ピリオド3つ）を付けると、そのメソッドを呼び出す側はその型の引数をいくつでも書けるようになる。

各メソッドにつき1つだけ、そのメソッドの最後の引数にだけ、指定可能。

可変長引数は、**糖衣構文**である（syntax sugar：複雑でわかりにくい書き方を、シンプルでわかりやすい書き方で表現できるもの）

```
public void method1(String... args){
    String val0 = args[0];
}
```

```
private int method2(String str, int... vals){
    int val1 = vals[1];
}
```

呼び出し方

```
method1();              //可変長部分は省略可
method2(“a”);           //可変長部分は省略可
method1(“a”, ” b”);
method1(“a”);
method2(“a”, 20,30,40);
```

※配列として値を取得できる。

もちろん、引数がセットされていない状態でargs[0]のようにアクセスすると、ArrayIndexOutOfBoundsExceptionが起こるので注意。

インスタンスとは

インスタンス（オブジェクト）

オブジェクト指向の説明でもあったようにインスタンスはクラスから生成されたデータである。

書き方：

```
クラス型 識別子 = new クラス型();
```

※メンバを利用する場合は「識別子.メンバ名」とします。

インスタンス – コンストラクタ

特殊なメソッド **コンストラクタ**

インスタンス（オブジェクト）はクラスから生成されたもの。

コンストラクタはインスタンスが生成される際に自動的に呼び出されるメソッド。

インスタンスを初期化する際に利用する。

書き方(引数が無い場合)：

```
アクセス修飾子 クラス名(){
    ……
}
```

or(引数がある場合)

```
アクセス修飾子 クラス名(データ型 引数名, …){
    ……
}
```

インスタンス – コンストラクタ

暗黙的コンストラクタの呼び出し

コンストラクタを定義しないと暗黙的に引数なしのコンストラクタが作られる。

→ コンパイラが自動生成する

引数ありのコンストラクタを定義すると引数なしのコンストラクタが呼べなくなる。

その他の修飾子 **final**

final修飾子を付与したフィールド、メソッドは変更できない。

変数：1度だけしか初期化出来ない＝定数
メソッド：上書きすることが出来ない＝オーバーライド出来ない。今後の講義で説明。

クラス：クラスの修飾子にfinalを使うことも出来る。その場合、finalなクラスは継承できない。
→ 継承の説明は後日

修飾子

その他の修飾子 **static**

static修飾子を付与したフィールド・メソッドのことを以下と呼ぶ。

クラス変数・静的変数

クラスメソッド・静的メソッド

対してstatic修飾子を付与しないフィールド・メソッドのことを以下と呼ぶ。

インスタンス変数

インスタンスメソッド

クラスメソッドからはクラス変数・クラスメソッドにアクセス出来るが、インスタンス変数・インスタンスメソッドにはアクセス出来ない。

インスタンスからはクラス変数・クラスメソッドにアクセス出来る。

パッケージ

クラスを分類・整理する仕組み。
パッケージはそれが含むタイプに関してユニークな名前空間を提供する。
同じパッケージにあるクラス群はお互いに保護された(protected)メンバにアクセスできる。

書き方：

```
package 識別子xxx;
```

このクラスはxxxに属する。

パッケージを指定しないことも可能ではあるが、業務でプログラムを作成する際にパッケージを決めないことは稀。

パッケージ

また、利用するクラス・パッケージを指定しなければならない。

書き方：

```
import パッケージ名;
```

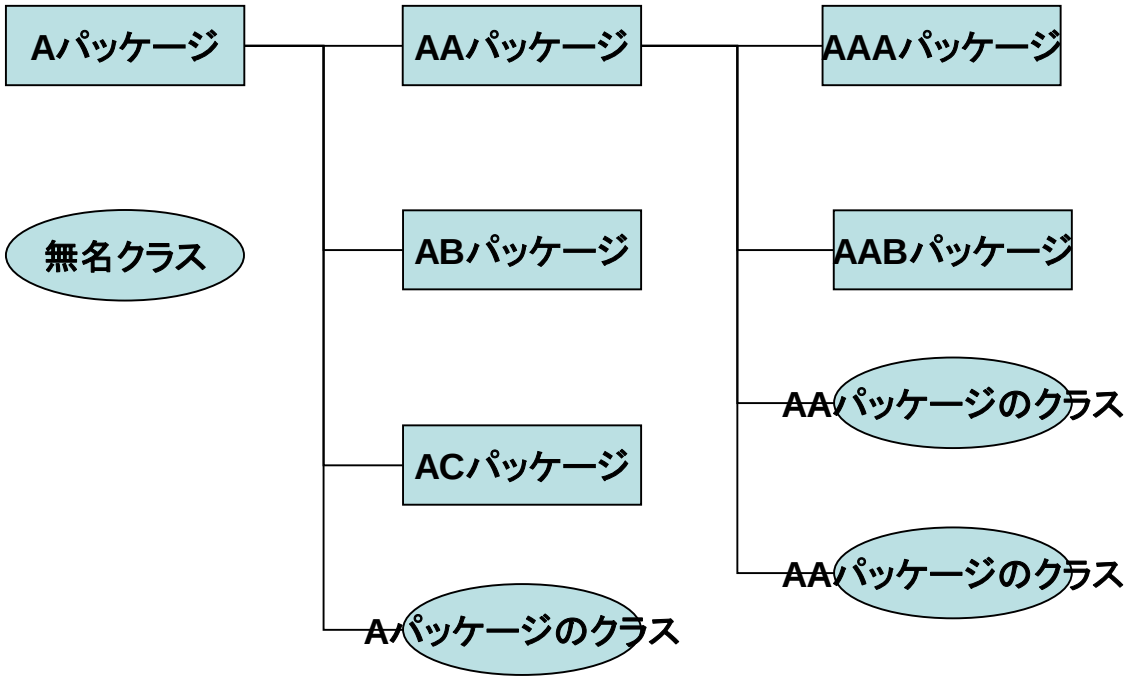
指定したパッケージ名に属するクラスを利用する。

import パッケージ名.クラス名;
指定したクラスを利用する。

※java.langパッケージはimportで記述する必要はありません。
※「*」で記述してもサブパッケージは利用できません。

パッケージ

パッケージの階層図



パッケージ

作成したクラス群を配布する際はクラス毎にばらばらの状態ではなく、
パッケージを「アーカイブ」というまとめた形とするのが一般的。

①JAR（Java ARchive）

jarコマンドでパッケージをアーカイブする。
ライブラリ

②WAR（Web application ARchive）

jarコマンドでパッケージをアーカイブする。
WEBアプリケーション