オブジェクト指向

「情報II」第6章

Contents

クラス：　　番号：　　氏名：

オブジェクト指向

多くのプログラミング言語でオブジェクト指向と呼ばれる考え方がよく用いられています。オブジェクト指向とは、実世界に存在する「モノ」の構造や振る舞いに着目し、「モノ」そのものや「モノ同士の関係」をソフトウェアで実現する方法です。

（教科書II：p.91 , p.116 – p.117）

■ オブジェクト指向の考え方

オブジェクト指向とは

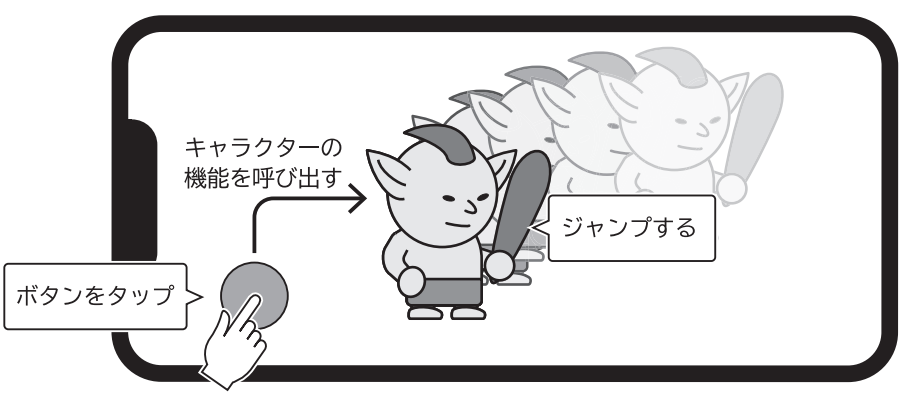
オブジェクト指向＝ オブジェクトというものを中心にプログラムを構成する考え方

オブジェクト＝ 機能と属性（性質）をひとまとめにしたもの

オブジェクト同士がメッセージ交換することにより機能を動かしたりする
同じような機能と属性をもったオブジェクトを再利用して生成することもでき効率的

実際のプログラムでは

ボタンやキャラクターなどを**オブジェクト**として画面上に配置
例）ボタンを押す → キャラクター（オブジェクト）の機能を呼び出す



どんなオブジェクトを用意し、それらをどのように関係させるかを考えよう

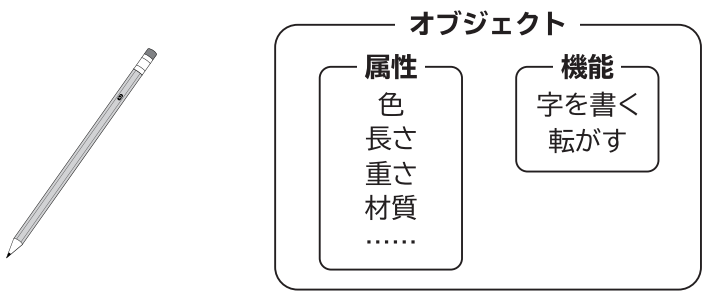
オブジェクトのカプセル化

カプセル化＝ オブジェクトの**属性**（**プロパティ**）と**機能**（**メソッド**）をまとめること

属性と機能

属性（**プロパティ**）＝ 色や形、位置など、オブジェクトの持っている特徴のこと

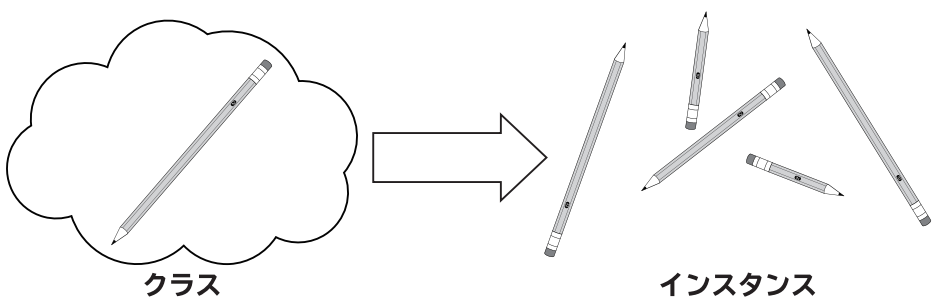
機能（**メソッド**）＝ オブジェクトの持っている機能のこと



クラスとインスタンス

クラス＝ 共通の属性（プロパティ）と機能（メソッド）を持つオブジェクトをまとめて一般化・抽象化

インスタンス＝ クラスを実体化したもので、具体的な値を持つオブジェクト



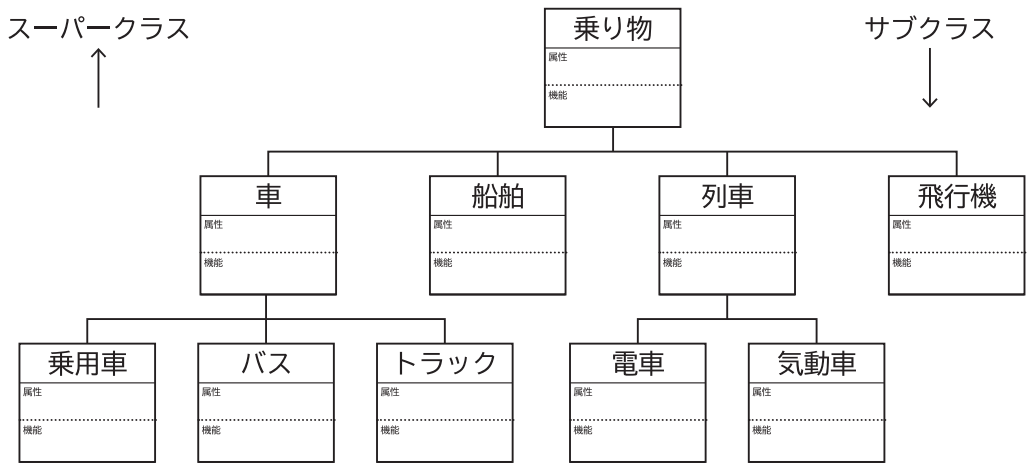
※一般的なえんぴつという概念 ＝ **クラス**

※いま手に持っているこのえんぴつと、筆箱に入っているそのえんぴつは別もの
→それぞれの具体的なえんぴつ ＝ **インスタンス**

継承（インヘリタンス）

継承＝既に定義されたクラスから、拡張や変更を加えて新たなクラスを定義すること

新しく定義されたクラスは、元のクラスの属性（プロパティ）や機能（メソッド）を**継承**する



継承する元のクラスを**スーパークラス**（**親クラス**）という

継承し新たに定義したクラスを**サブクラス**（**子クラス**）という

考えてみよう1

上の図の例で、トラックの性質（属性（プロパティ）、機能（メソッド））を考えてみよう

①乗り物に共通した機能（メソッド）には何があるだろう？

1
2

②車に共通した機能（メソッド）には何があるだろう？

3
4

③トラック独自の機能（メソッド）には何があるだろう？

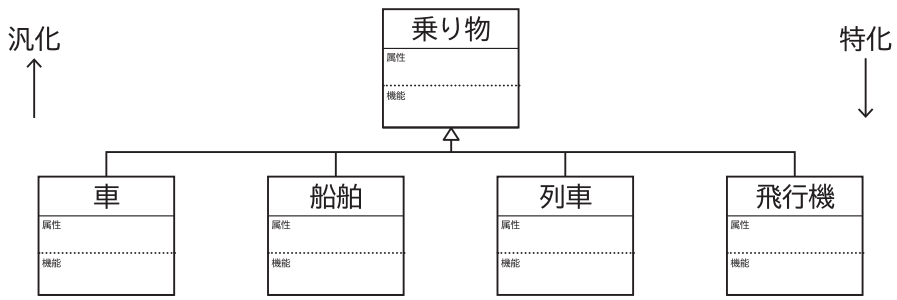
5

④トラック独自の属性（プロパティ）には何があるだろう？

6

汎化と特化（is-a関係）

「～は・・・である」という関係にサブクラス、スーパークラスにまとめる



例：「列車 is a 乗り物」＝「列車は乗り物である」

汎化 ＝ クラスの共通する性質をひとまとめにし、スーパークラスにすること

特化 ＝ スーパークラスの性質を継承して新たなサブクラスを生成すること

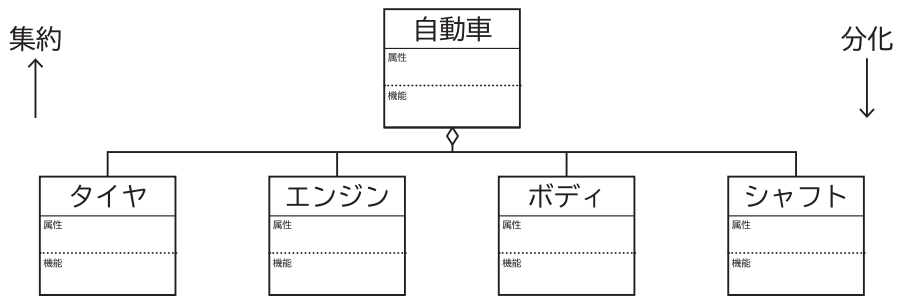
考えてみよう2

クラスを7つ以上作成し、is-a関係に図解してください。

7

集約と分化（part-of関係）

「～は・・・の一部である」という関係にクラスの関係をまとめる



例：「エンジン is a part of 自動車」＝「エンジンは自動車の一部である」

集約：いくつかの部分がひとまとまりとなって一つのオブジェクトを構成すること

分化：複雑な構造をしているオブジェクトを単純なオブジェクトに分解すること

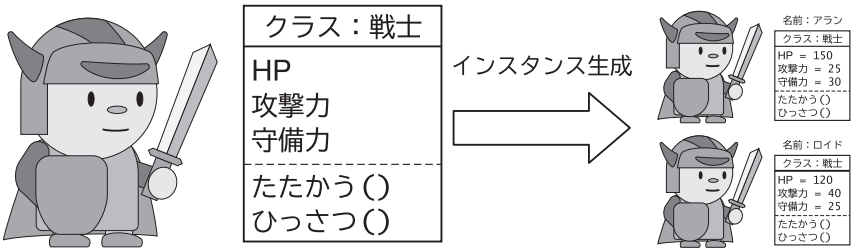
考えてみよう3

クラスを4つ以上作成し、part-of関係に図解してください。

8

ゲームを例に考える

クラスをもとに具体的に生成されたものがインスタンス
クラスは、キャラクターの職業のようなもの
インスタンスは、実際に登場するキャラクター



※戦士が複数登場する場合もある
※モンスターも同様にクラスとして設定 → 登場モンスターはインスタンス

オブジェクト指向という目でゲームを見てみよう

振り返り

次の各観点が達成されていれば□を塗りつぶしましょう。
□オブジェクト指向におけるオブジェクトとはどのようなものかを理解できた
□オブジェクトには属性と機能があることを理解できた
□クラスとインスタンスの概念を理解できた
□クラスの継承の概念を理解し、is-a関係、part-of関係の例を考えることができた

今日の授業を受けて思ったこと、感じたこと、新たに学んだことなどを書いてください。

オブジェクト指向プログラミング

前節では、そもそもオブジェクト指向とはどのようなものであるか、その考え方について学びました。ここでは、実際にオブジェクト指向の考え方を使ってプログラミングをしてみたいと思います。

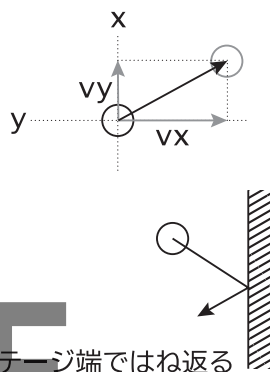
（教科書II：p.91）

■ オブジェクト指向プログラミング

クラスの設定

Ball（ボール）というクラスを次のように定義する

クラス名：**Ball**		
属性（プロパティ）	**x**	ボールのx座標
	y	ボールのy座標
	d	ボールの直径
	vx	ボールのx方向の速さ
	vy	ボールのy方向の速さ
	c	ボールの色
機能（メソッド）	**move()**	ボールを動かす
	display()	ボールを画面に表示する



ステージ端ではね返る

クラスの定義

```
 9 class Ball {                  ここからクラスBallを定義
10   x = random(0,width);       x座標の初期値を乱数で決定
11   y = random(0,height);      y座標の初期値を乱数で決定
12   vx = random(-5,5);         x方向の速さを乱数で決定
13   vy = random(-5,5);         y方向の速さを乱数で決定
14   d = 10;                    ボールの直径を10に設定
15   c = 255;                   ボールの色を白（255）に
16
17   move(){                    ここからmove()メソッドを定義
18                              move()メソッドの内容を書く
19   }
20
21   display(){                 ここからdisplay()メソッドを定義
22                              display()メソッドの内容を書く
23   }
24 }
```

※**width** , **height** はキャンバスの幅（**width**）、高さ（**height**）が格納された変数

インスタンスの生成

インスタンス = クラスを実体化したもので、具体的な値を持つオブジェクト

インスタンスを格納するための配列を用意する → 今回は**balls**[]という配列を用意

setup()関数内にforループでインスタンスを準備する

```
 1 let num = 10;                          インスタンスの個数をnumに設定
 2 let balls = [];                        ballsを配列として用意する
 3
 4 function setup(){                      実行時に一度だけ実行される関数
 5   createCanvas(400,400);               キャンバスサイズを設定
 6   for(let i = 0; i < num; i++){        iを0からnum回繰り返す
 7     balls[i] = new Ball();             インスタンス生成、ballsに格納
 8   }
 9 }
```

※まだ現段階では再生しても何も表示されない

display()メソッドの定義

ボールを画面上に表示させるために、**display()**メソッドを実装してみよう

オブジェクトが持つ変数を指定する場合、this.xのように表記する

```
27   display(){
28     fill(this.c);                      塗り色をcにする（255）
29     circle(this.x, this.y, this.d);    円を描く
30   }
```


display()メソッドの呼び出し

上で定義した**display()**メソッドを呼び出してみよう

インスタンスのメソッドを呼び出す場合、インスタンス名.メソッド名()のように表記

→たとえば**ball**[0]のdisplay()メソッド → **ball**[0].display()となる

```
11 function draw(){                       実行中ずっと繰り返される関数
12   background(220);                     背景を薄いグレーにする（220）
13   for(let i = 0; i < num; i++){        iを0からnum回繰り返す
14     balls[i].display();                ballsのdisplay()を実行
15   }
16 }
```

※この時点で、ようやく画面上に10個のボールが表示される

例題1

numの値を100にして、100個のボールを表示させてみよう

例題2

move()メソッドを実装してみよう

まずは、初速度のまま移動するだけのものを実装してみよう

→画面外に

```
27   move(){
27     this.x += this.vx;
28     this.y += this.vy;
29   }
```

塗り色を**c**にする（255）
円を描く

例題3

move()メソッドを呼び出してみよう

→**display()**を呼び出すよりも前に**move()**を呼び出すようにしよう

```
13   for(let i = 0; i < num; i++){
14                    ;
15     balls[i].display();
16 }
```

balls[i]のmove()を呼び出す

例題4

move()メソッド内に、ボールのはね返りを実装してみよう

this.**x**が0より小さい **または** this.**x**が**width**より大きい場合

→this.**vx**に−1を掛ける

this.**y**が0より小さい **または** this.**y**が**height**より大きい場合

→this.**vy**に−1を掛ける

```
29
30     if(              ||                  ){
31       this.vx        ;
32     }
33     if(              ||                  ){
34       this.vy        ;
35     }
```

||は **または** の意味
this.**vx**を−1倍に

this.**vy**を−1倍に

課題

最低限、［例題4］まで完成した状態のものを提出してください。
下に示す改良方針に従って、更に改良していこう。

改良方針

①ボールの速さの調整 → 小さい値にするとゆっくり動くようになる
　→いろいろな速さにして違いを確かめてみよう
②ボールの直径を乱数で決める → 大きさは6 ～ 20くらいまで？
　→大きさはいろいろと試行錯誤してみよう
③ボールの色を設定する → **cr** ， **cg** ， **cb** 変数を用意し、乱数で0 ～ 255を生成
　→`fill(this.cr, this.cg, this.cb);`
　→乱数の値を128 ～ 255など、大きな値にすると、明るい色にできる
④壁にめり込まずにはね返るようにする → **x** ， **y**座標は円の中心
　→`this.x`が`0+this.d/2`より小さい
　　または `this.x`が`width-this.d/2`より大きい場合
　→`this.y`が`0+this.d/2`より小さい
　　または `this.y`が`width-this.d/2`より大きい場合

振り返り

次の各観点が達成されていれば□を塗りつぶしましょう。
□属性（プロパティ）と機能（メソッド）をクラスの中に定義することができた
□インスタンスを生成することができた
□各メソッドを定義し、メソッドを呼び出すことができた
□プログラミングの知識を思い出し、一つの作品を作り上げることができた

今日の授業を受けて思ったこと、感じたこと、新たに学んだことなどを書いてください。

おまけ

■ 雪を降らせるプログラムを作ってみよう

ゴール

大きさや落下速度の異なる円（雪）がランダムに降ってくるアニメーションをつくりたい
画面幅に対して10分の1くらいの量を降らせたい
速さはy方向のみで、1 ～ 2の乱数で設定
雪の大きさは、6 ～ 20の乱数で設定

キャンバスサイズと、雪の降る量の設定

```
 1 let snows = [];                                  インスタンス格納
 2
 3 function setup(){
 4   createCanvas(windowWidth, windowHeight);       画面の幅、高さ
 5   let snowNum = int(width/10);                   幅の1/10の量
 6 }
 7
 8 function draw(){
 9   background(0);                                 背景を黒（0）に
10 }
```


特殊変数

windowWidth	画面の**幅**が格納されている特殊変数
windowHeight	画面の**高さ**が格納されている特殊変数
width	createCanvas()で設定したキャンバスの**幅**
height	createCanvas()で設定したキャンバスの**高さ**

Snowクラスの定義

属性（プロパティ）の定義

```
12 class Snow {
13   x = random(0,width);      x座標は0からキャンバス幅までの乱数
14   y = random(-20,-1000);    y座標は画面のかなり上に設定しておく
15   d = random(6,20);         雪の直径を6 ～ 20の乱数で
16   v = random(1,2);          雪の速さは1 ～ 2の乱数で
17
```

機能（メソッド）の定義

```
18   move(){
19     this.y += this.v;                  yを座標をvだけ移動
20     if(this.y > height + 30){          キャンバス下限を超えたら
21       this.y = random(-20,-300);       キャンバス上に移動
22     }
23   }
24
25   display() {
26     noStroke();                        周囲の線をなしにする
27     fill(255);                         塗り色を白にする
28     circle(this.x, this.y, this.d);    円を描く
29   }
30 }
```

インスタンスの生成

```
3  function setup(){
4    createCanvas(--略--);
5    let snowNum = int(width/10);
6    for(let i = 0; i < snowNum; i++){   雪の数だけ繰り返す
7      snows.push(new Snow());           snows配列に追加していく
8    }
9  }
```


配列操作

配列に対して.push()メソッドを実行すると、引数が配列の最後に追加される

→**snows**.push(new Snow());は、**snows**配列に新しいSnowインスタンスを追加

move()、display()メソッドを使用

```
11 funcrion draw(){
12   background(0);
13   for(let snow of snows){   snows配列を順に実行、実行中はsnowに
14     snow.move();            move()メソッドを実行
15     snow.display();         display()メソッドを実行
16   }
17 }
```

配列に対する繰り返し

snow of snowsは、**snows**配列を0番目から順に全て繰り返す

→現在の繰り返し番目の値を**snow**という変数に格納する

章末問題

[問題1]

次のそれぞれは、オブジェクト指向におけるis-a関係か、part-of関係のいずれですか。

(1) "会社"と"社員"

(2) "自動車"と"エンジン"

(3) "図形"と"三角形"

(4) "人間"と"頭"

(5) "パン"と"メロンパン"

(6) "哺乳類"と"ゾウ"

[問題2]

次の各組み合わせのうち、オブジェクト指向におけるクラスとインスタンスの関係である組み合わせには○、そうでない組み合わせには×を書いてください。

(1) 公園 , ぶらんこ

(2) 公園 , 中之島公園

(3) 駅 , 近江八幡駅

(4) 駅 , 改札口

[問題3]

次のようなクラス"生徒"からインスタンス"太郎くん"が生成されたとして、インスタンス"太郎くん"に次のような動作をさせる際、または属性を取り出す際の記述方法を書いてください。

クラス：生徒
学級 性別 部活
授業を受ける() 弁当を食べる() 登校する()

(1) 授業を受ける

(2) 学級

(3) 弁当を食べる

コラム～ポリモーフィズム

■ ポリモーフィズム

オブジェクト指向三大要素

- ♦**カプセル化**　＝　属性と機能を一つにまとめる考え方
- ♦**継承**　＝　属性や機能をサブクラスに引き継ぐ
- ♦**ポリモーフィズム**　＝　？

ポリモーフィズムとは

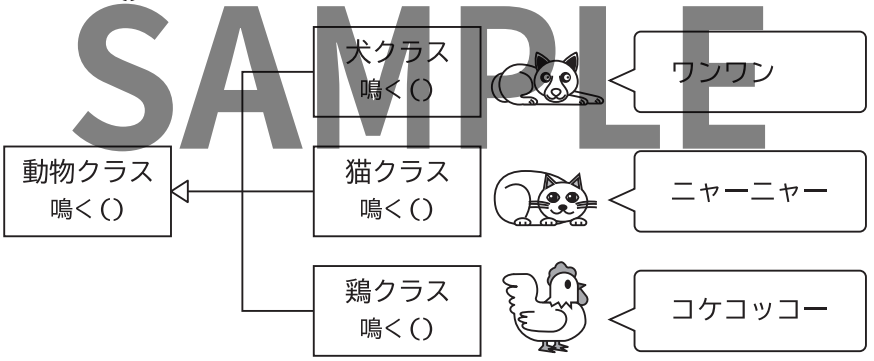
ポリモーフィズム＝ 同じメソッドを使用してもオブジェクト毎に異なる動作をさせる

例

動物クラスに鳴く()メソッドがあるとする

動物クラスのサブクラスに、犬クラス、猫クラス、鶏クラスがあるとする

→それぞれに鳴く()メソッドが独自に定義されている



それぞれ、犬.鳴く()、猫.鳴く()、鶏.鳴く()で実行することができる

ここで、鳴く()は共通して持っているから、まとめてしまおうという発想

```
動物 = {犬, 猫, 鶏};
for(動物){
  動物.鳴く();
}
```

動物の中に入るものが変わっても、鳴く()は共通している

→別の処理にすることができる

条件分岐を使わなくても分岐のような振る舞いをさせることができ、プログラムが単純化

SAMPLE

2025年度「情報Ⅱ」ワークブック見本
CQ出版社