

第一回 手続き型言語とオブジェクト指向言語

1. かんたん！手続き型言語

かつてプログラミングをする上では、「手続き型言語」と呼ばれる形のプログラミング言語が主流でした。手続き型言語とは、名前が指し示す通り、処理の手続きを記述する方式です。

たとえば、「1～300の合計値を求める」プログラムを考えるとします。この問題を解くための「手続き」を簡単に書くと、

- ①合計値を求めるための変数 sum を用意する
- ②ループ用変数 i を用意する
- ③ i を 1 から始めて 300 になるまで④を繰り返す
 - ④sum に i を足す
- ⑤sum の値が 1～300 までの合計値となる

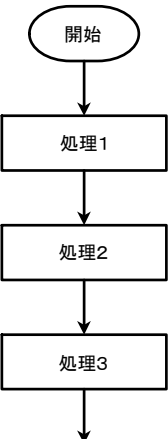

・・・という風には書けるとおもいます。この考え方はシンプルかつ直観的なので、恐らくちょっとでもプログラミングをしたことある人なら、誰でも簡単に理解できるとおもいます。手続き型言語は、問題を解くための手順(=手続き)を一つ一つ事細かに書くことで、プログラムが解くべき問題を解決するのが特徴です。手続き型の考え方のメリットは「簡単である」ことです。人間がソースコードを見たときプログラムの処理の流れを掴みやすく、一目でそのプログラムが何をしたいのかが分かる※1という優れモノです。

しかし！そんな簡単な手続き型言語が既にあるにも関わらず、これからこの資料で説明する「オブジェクト指向言語」は、現在、多くのプログラマーが用いる言語となっているのです。

2. “恐怖の”オブジェクト指向言語

手続き型言語はシンプルで分かりやすい、という話をしました。一方のオブジェクト指向言語とはどのようなものなのでしょうか。オブジェクト指向言語を一言で表すと、「処理の対象をオブジェクト単位で分割し、オブジェクト同士のメッセージのやり取りにより処理を記述する」プログラミング言語です。

「？」と思っても結構です。というか、知らない人が聞いても絶対に訳がわからないと思います。手続き型言語と比べると、オブジェクト指向はとても新しい考え方で、実は誕生してから20～30年程の概念です。つまり、オブジェクト指向とはコンピュータが誕生してから※2長い間研究されていた技術の粋を集めた考えであり、それ故に高度な考えとして理解するのが難しいものとなってしまったのです。(ちょっと大げさ)

手続き型言語	オブジェクト指向言語
 <pre> graph TD Start([開始]) --> P1[処理1] P1 --> P2[処理2] P2 --> P3[処理3] P3 --> End[] </pre>	
<p>処理を矢印で結んでいく形。 直観的でイメージしやすい</p>	<p>視覚的な形にしにくい (可能だが高度な知識が必要) 抽象的でイメージしにくい</p>

とは言っても、「オブジェクト指向ってそんな難しいものなのか」と身構えなくても大丈夫です。オブジェクト指向は決して難しいのではなく、「理解しにくい」だけであって、一度理解できれば簡単なのです。そして、一度理解することができればオブジェクト指向のメリットを活かして、より簡潔で、人間にとって理解しやすいプログラムを書けるようになるはずです。

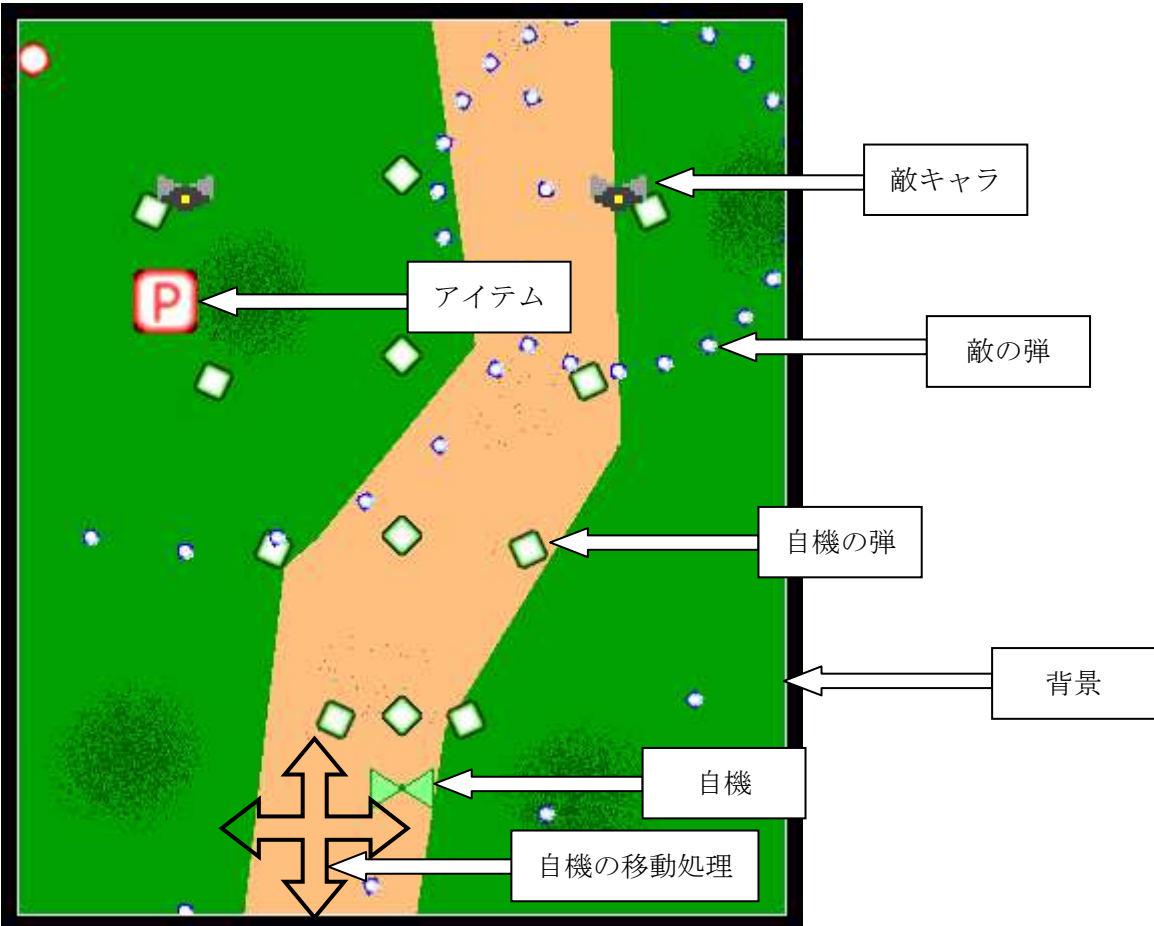
3. 手続き型言語の限界

「手続き型」はシンプルで分かりやすい、ということは理解してもらえたと思います。では、なぜ現代のソフトウェア開発でわざわざ理解しにくいオブジェクト指向を使っているのでしょうか？実は、手続き型言語にも欠点があります。そして、その欠点を解決する方法こそが、オブジェクト指向を用いることのメリットにも繋がっているのです。

今、あなたは「シューティングゲームを作りたい」と思ったとき、どのように考えますか？おそらくシューティングゲームを構成するためには最低限、以下の要素が必要になると思います。

- ・ゲームのメインループ
- ・プレイヤーの操作（移動、弾の発射）
- ・敵キャラの出現および行動
- ・自機との当たり判定
- ・アイテムによる自機の強化
- ・ステージデータの読み込み
- ・プレイヤー、敵、弾、背景等の描画
- ・スコアの計算

などなど。そして、これらの処理を全て手続きとして記述するのを想像してみてください。



シューティングゲームには、少なくともこれくらいの要素がある。さらにこれら全てにそれぞれ処理が必要なのを考えると・・・

※どうでもいいですが、この画像は私が昔作ったゲームの画面です。

・・・正直考えるのも嫌になるような量になりますね。しかも、実際には弾は1種類だけじゃないし、※3敵だって何種類も存在します。そして、これらを構成する大量の変数も必要になります。そうなってくると、デバッグも大変だし、どこから手をつければいいのか訳がわからなくなりそうです。

現代のソフトウェア開発というのは、このように複数の要素が複雑に絡み合って全体を構成しなければならないものが大半です。実はそのような処理を記述する上でオブジェクト指向は手続き型言語より遥かに優れているのです。

4. オブジェクト指向のメリット・デメリット

オブジェクト指向のメリットは実に多彩です。ここではキーワードだけ挙げて、具体的な解説は次回以降にしていこうと思います。(そもそもオブジェクト指向とは何か？という解説もしてないし・・・)

- ・再利用性が高まる
- ・(大規模開発において)開発工程を減らせる
- ・継承、カプセル化、多態性などの考え

一方、オブジェクト指向はいいことづくめではなく、デメリットもあります。勉強する前からデメリット勉強してどうすんだ！と言われそうですが、大事なこともあるので先に紹介しておきます。こちら具体的な解説は次回以降で。

- ・プログラムの設計が大変
- ・小規模開発においては、かえって開発工数が増える
- ・ソースコードの量自体は増える

5. オブジェクト指向言語には何がある？

現在実際にソフトウェア開発の現場では多数のプログラミング言語が使われています。その中でもオブジェクト指向言語として代表的なものとしては、

- ・C++
- ・Java
- ・C#
- ・Objective-C

といった言語が挙げられます。ただし、本講義ではできるだけ具体的なプログラミング言語は使わず、あくまで概念の説明のみに留まろうと思います。記述の仕方というのは本質的ではないし、記述方法だけが先行してオブジェクト指向の本質が曖昧なまま覚えてしまわないようにするためです。(ちなみに筆者も C++でオブジェクト指向を学んだが、形だけを覚えてしまった結果オブジェクト指向の正しい使い方を理解できていなかった)

それでも便宜上どうしても必要になる場合があるので、その場合は「C#」準拠で書くことを明記しておきます。

6. おわりに

はい、今回はこれで終わりです。・・・実は「オブジェクト指向って何？」という基本的な内容すらやっていません。決して手抜きではなく、しっかりとオブジェクト指向の出来たいきさつを説明してから解説をしたほうが良いと考えたからです。というわけで、次回からが本題、オブジェクト指向の最も基本的な定義の説明をしていこうと思います。

※1 : 3にも記述したように、これはあくまでプログラムが簡単である場合。実際にはこの限りではない。

※2 : 1955年に初めて誕生したENIACが人類史上初のコンピュータ。

※3 : インベーダーみたいなゲームならともかく、今時のシューティングは大抵数十～数百の弾データがある。