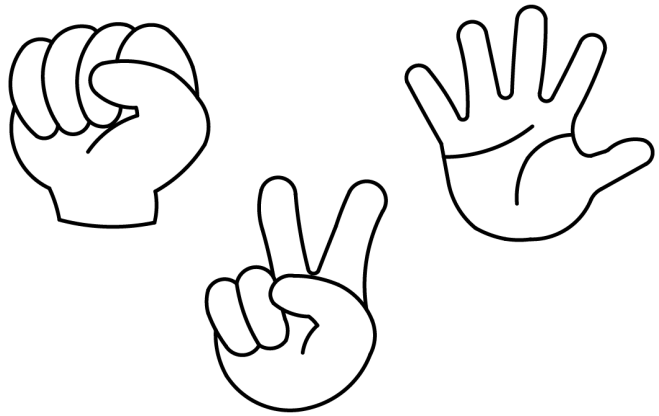
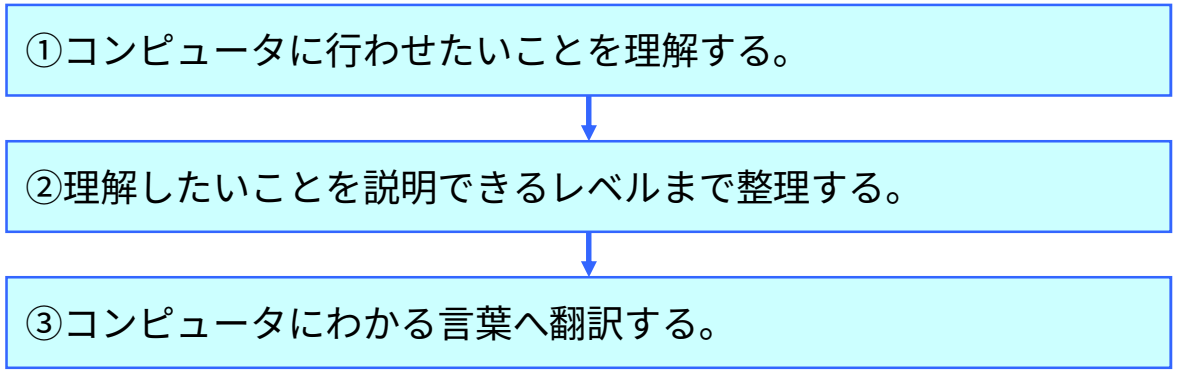
Java言語 基礎課題②　～ジャンケン～



プログラムを作成するためには、下記のような流れをきちんと押さえることが重要です。したがって、まずは「コンピュータに行わせたいことを理解する」ことが必要です。



上記のようにプログラミングとは「人間の理解を、プログラミング言語を用いてコンピュータに伝えること」と言えます。その従来の方法が難しいのは「人間の理解を、プログラミング言語で表すこと」が難しいからなのです。では、もう少し楽に我々の考えたものをコンピュータに伝える方法はないのでしょうか・・・。

実はこの、人間の理解をできるだけそのままコンピュータへ伝えるための方法として考えられたものが

オブジェクト指向

なのです。

演習１：現実世界からオブジェクト指向の世界へ

左側に現実世界でのジャンケンがどのように行われているかを整理したものがあります。この左のジャンケンゲームの様子に従って、それぞれの登場人物は何をしているかを右の表にまとめてください。

現実世界のジャンケンの様子

これから村田さんと山田さんの２人で実際にジャンケンをしてもらいます。斉藤さんには審判役をお願いします。それではどうぞ！

斉藤：「それでは村田さん対山田さんのジャンケン３回勝負をはじめます。」

斉藤：「まず１回戦目。ジャンケン・ポン！」

村田：「パー！」

山田：「チョキ！」

斉藤：「はい、パーとチョキで山田さんの勝ちです」

斉藤：「続いて２回戦目。ジャンケン・ポン！」

村田：「グー！」

山田：「チョキ！」

斉藤：「はい、グーとチョキで村田さんの勝ちです」

斉藤：「それでは３回戦目。ジャンケン・ポン！」

村田：「グー！」

山田：「パー！」

斉藤：「はい、グーとパーで山田さんの勝ちです」

斉藤：「これで３回勝負が終わりました。２人の勝った回数を聞いてみましょう」

斉藤：「村田さん何回勝ちましたか？」

村田：「１回です」

斉藤：「山田さんは何回勝ちましたか？」

山田：「２回です」

斉藤：「はい、２対１で山田さんの勝ちです！」

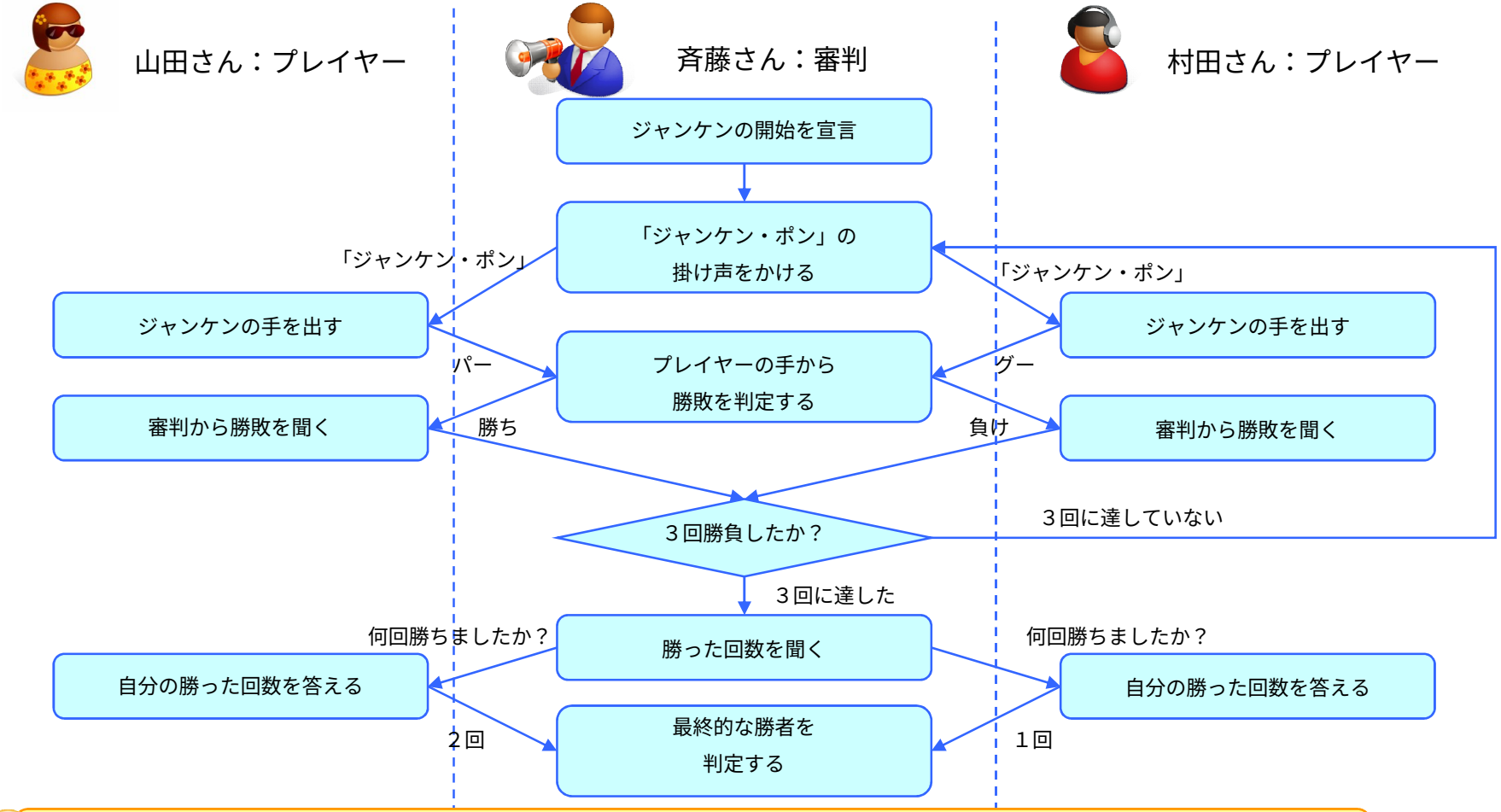
村田さんの行ったこと	
1	（「ジャンケン・ポン」の掛け声と同時に）ジャンケンの手を出す
2	審判から勝敗を聞く
3	自分の勝った回数を答える
4	
5	

山田さんの行ったこと	
1	
2	
3	
4	
5	

斉藤さんの行ったこと	
1	
2	
3	
4	
5	

登場人物を明確にしたジャンケンの流れ

これで登場人物が何を行うかが明確になりましたよね。これを元にして、ジャンケンの流れをわかりやすく表現したものが下記の図になってます。縦に引かれた点線で、左から山田さん、斉藤さん、村田さんが行うことを分けてあります。登場人物はそれぞれが行うことが決まっていて、自分の役割に徹しています。



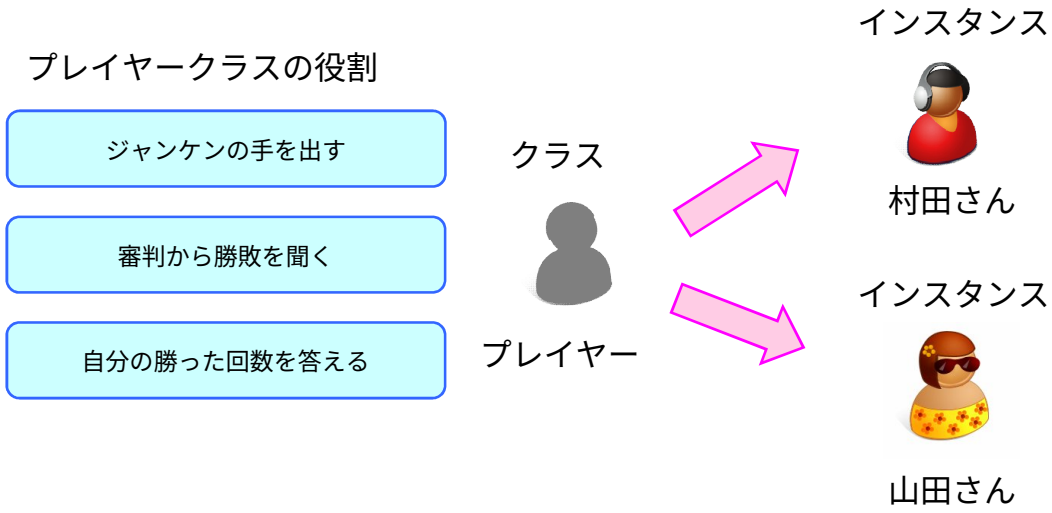
上記のように役割分担を明確にするということがオブジェクト指向による考え方の第一歩になります。

クラスとインスタンス

前ページの村田さんと山田さんの「行った」ことは同じですよね。村田さんと山田さんは別々の人物にもかかわらず、「行ったこと」という観点で見ると実は同じなのです。つまり「プレイヤー」という同じ役割を演じているといえます。

このことより、今考えているジャンケンで必要な役割は「審判」と「プレイヤー」の２種類であることがわかります。

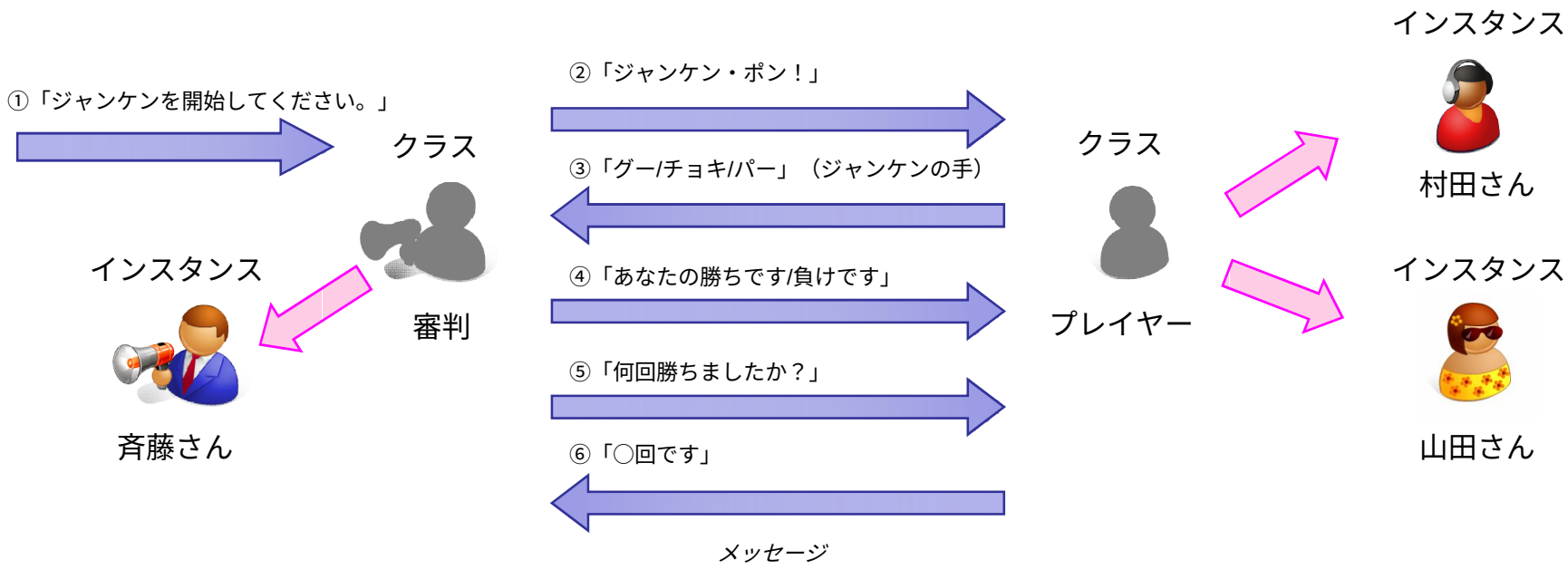
オブジェクト指向では、この役割をクラス（Class）と呼びます。また、「村田さん」や「山田さん」といった実際の登場人物を、クラスに対してインスタンス（instance：実体）と呼んでます。



上記の図はプレイヤークラスにおけるクラスとインスタンスの関係を示したものです。インスタンスはより広い意味でオブジェクト（object：実体が存在するもの）と呼ばれています。オブジェクト指向の「オブジェクト」とは、これが元になっているのです。

メッセージ・パッシング

登場人物を明確にしたジャンケンの流れをもう１度見てください。それぞれの登場人物がお互いの動作をきっかけとして、自分の役割をこなしていることがわかると思います。例えば、プレイヤーである村田さんと山田さんは、審判である斉藤さんの「ジャンケン・ポン」というかけ声をきっかけとして、ジャンケンの手を出しています。オブジェクト指向では、この「きっかけ」をメッセージと呼びます。



オブジェクト同士がメッセージをやりとりしてジャンケンを行う様子がより明確になりましたね。このように、オブジェクト同士がメッセージをやりとりすることを、メッセージ・パッシング（message passing）と呼びます。このようにオブジェクト指向による考え方は、私の普段の考え方に近く、シンプルであることがわかってもらえると思います。

クラスに対する「操作」という考え方

「クラスとインスタンスの関係」と「メッセージ・パッシング」は、オブジェクト指向の根底にある考え方です。しっかり覚えておきましょう。もう少しクラスについて勉強しましょう。

下記の表は、プレイヤークラス・審判クラスのメッセージと振る舞いを表したもので、各クラスは受け取ったメッセージをきっかけとして、それぞれのオブジェクトがどのように振る舞うかを明確にしています。これはクラスの持つ機能であると考えることもできます。

プレイヤークラスの役割		
	きっかけとするメッセージ	オブジェクトの振る舞い
1	「ジャンケン・ポン！」	ジャンケンの手を出す
2	「あなたの勝ちです」「あなたの負けです」	勝った回数を覚えておく
3	「何回勝ちましたか？」	自分の勝った回数を答える

審判クラスの役割		
	きっかけとするメッセージ	オブジェクトの振る舞い
1	「ジャンケンを開始してください」	「ジャンケン・ポン」と声をかける

このようなオブジェクトの振る舞いを、オブジェクト指向の世界では操作と呼びます。

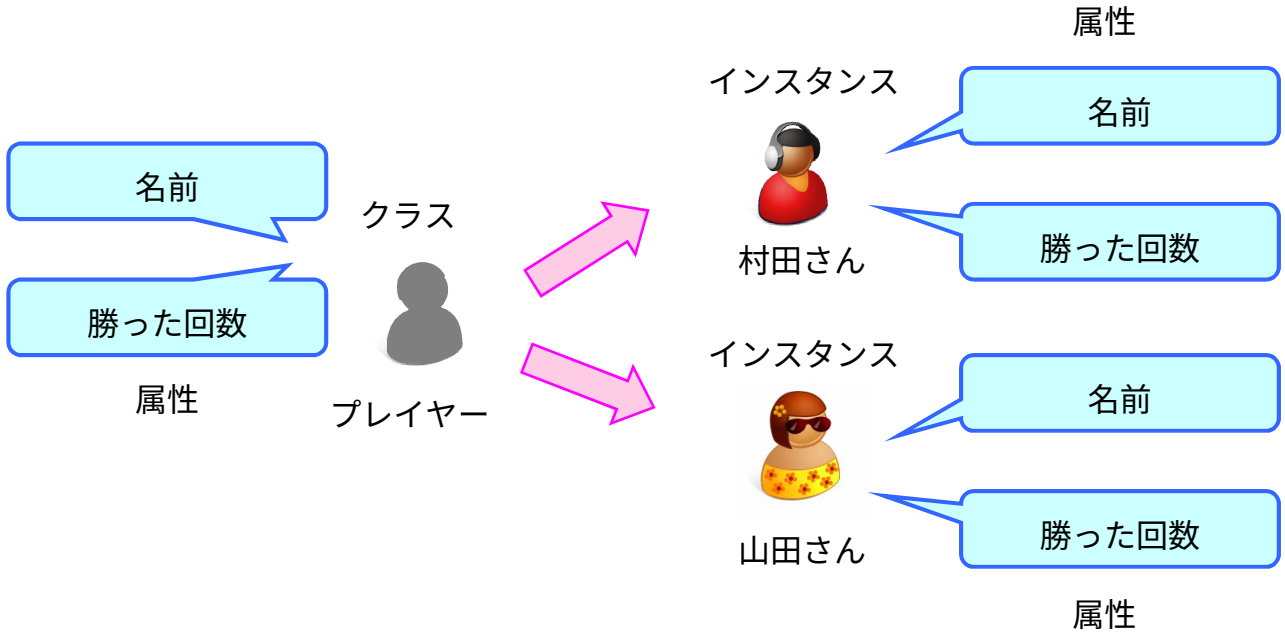
※ここでは「操作」という言葉の持つ本来の意味にはあまりとらわれずに以下のように考えてください。

オブジェクト指向でいう「操作」とは、「メッセージをきっかけとした、オブジクトの振る舞い」に「操作」という名前をつけているにすぎない

オブジェクトの状態を管理するための方法①

現実世界のジャンケンを分析することで、「審判」と「プレイヤー」という２つのクラスが必要であることがわかりました。今回のジャンケンでは、プレイヤークラスに対して「村田さん」・「山田さん」という２つのインスタンスが登場していましたね。それではこの村田さんと山田さんの違いは何でしょうか。「身長」・「体重」・「服装」・「容姿」…と色々挙げることができるはずです。このようなそれぞれのインスタンス固有の性質を属性といいます。　※属性はクラスに対して定義します。

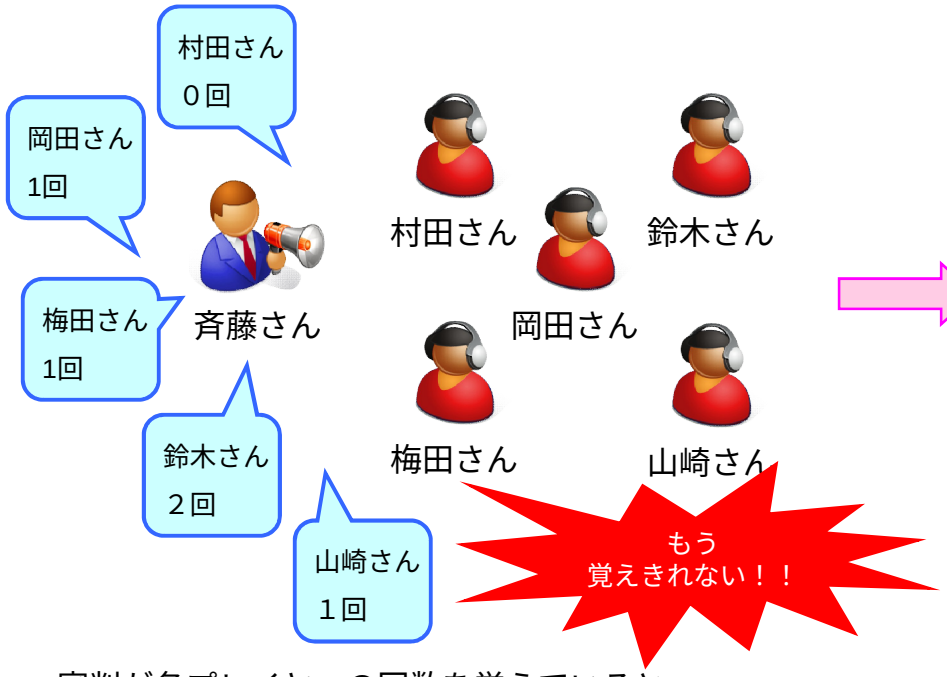
今考えているプレイヤークラスの場合は「名前」や「勝った回数」が属性であり、その状態（何という名前か、何回勝ったか）は、インスタンスごとに違うというわけです。



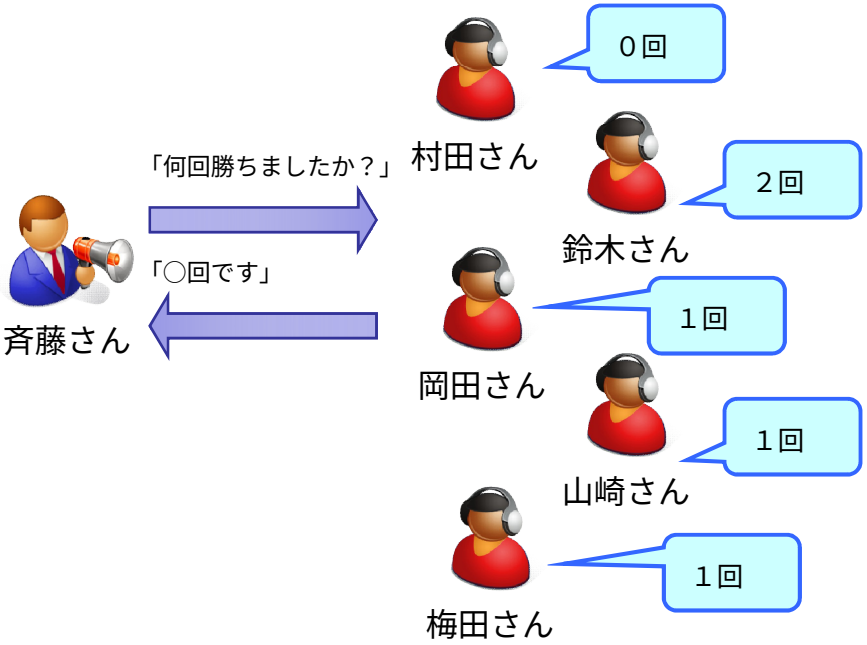
オブジェクトの状態を管理するための方法②

前のページのようにインスタンス固有の状態を表す属性という考え方を取り入れることで、オブジェクトごとの状態管理がしやすくなりました。属性という考え方は比較的理解しやすいでしょう。しかし、「これって、実際どう役立つの？」と思う人は多いかもしれません。

例えば現実世界のジャンケンで、審判が各プレイヤーの勝った回数を覚えていなければならないとしたら、どうでしょうか？２人ならまだしも、３人、４人と増えるにしたがって、とても覚えてはいられなくなることでしょう。そこで、各プレイヤーには自分の勝った回数を自分で覚えておいてもらい、審判はあとから「何回勝ちましたか？」と聞く方が合理的だと思いませんか。



審判が各プレイヤーの回数を覚えていると・・・

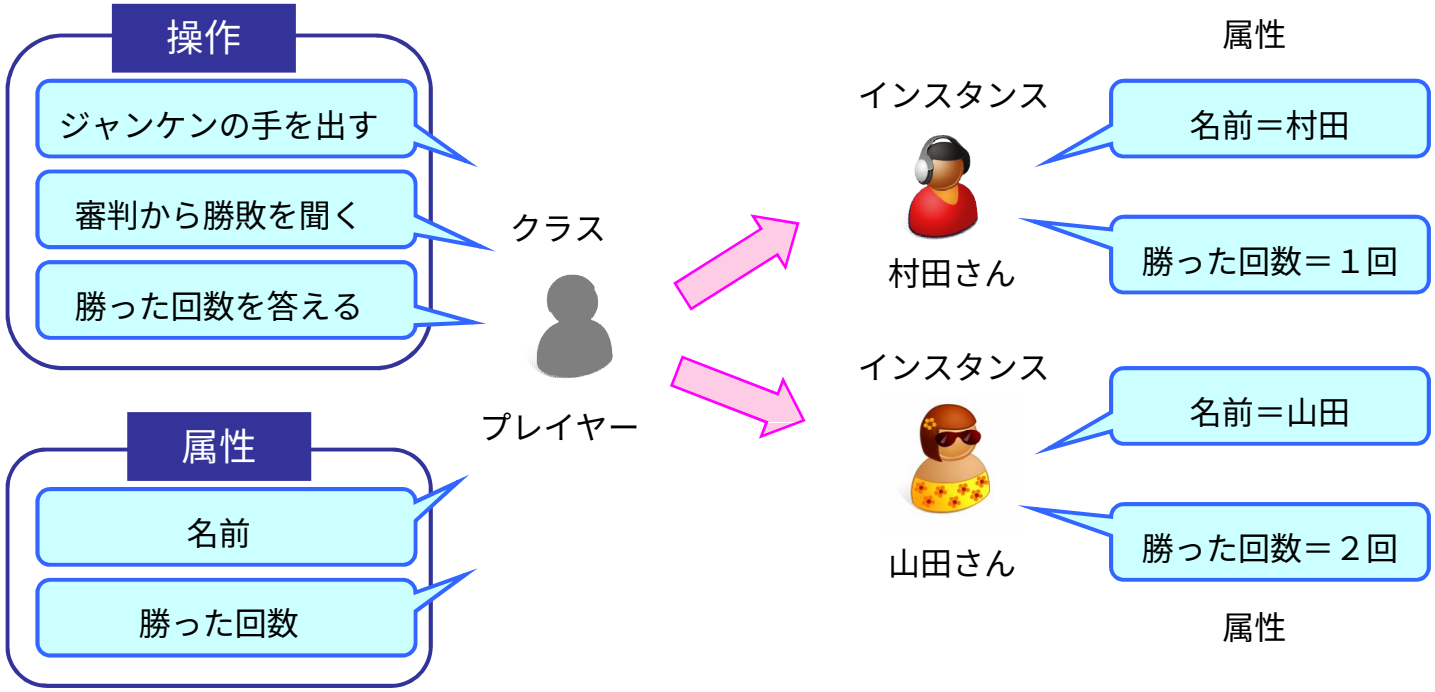


プレイヤーに属性を持たせると・・・

オブジェクトの状態を管理するための方法③

前のページのプレイヤーのようにクラスに属性を持たせることで、現実世界と同じように自分に関する情報は自分で管理させるようにしたのです。逆に言えば、あるオブジェクトの属性はそのオブジェクト自身しか知りません。オブジェクトの状態が知りたければ、オブジェクトに直接聞くしかないのです。
ジャンケンの例では、プレイヤーに対して「何回勝ちましたか？」と聞くことによって、そのプレイヤーの勝った回数を知ることができましたね。

このように、オブジェクトの内部の状態を隠し、ほかのオブジェクトからは直接知ることができないようにすることをカプセル化（encapsulation）といいます。



演習２：オブジェクト指向版ジャンケンプログラムの作成

　オブジェクト指向プログラミングでは、オブジェクト同士がメッセージ・パッシングによってお互いの操作を呼び合う、というオブジェクト指向の考え方を、そのままプログラムとして表すことができます。

　非オブジェクト指向では、「コンピュータの物事の手順を説明する」というアプローチのプログラミングでしたが、オブジェクト指向では「コンピュータで物事そのものを表現する」というアプローチのプログラミングになります。それでは実際に手を動かしてジャンケンプログラムを作成しましょう！



ジャンケンプログラム作成手順

１．プレイヤークラスを作成する Player

２．審判クラスを作成する Judge

３．ジャンケンプログラム ObjectJanken というクラスを新たに作成し実行する。