前回のレッスンでは、アニメーションを実装するために、ウェルカムスクリーンステートクラスにSingle TicketProviderStateMixinと呼ばれるものを追加する必要があり、「with」キーワードを使用してそれを行いました。

したがって、このレッスンでは、これらのミックスインとは何か、なぜ必要なのかを詳しく説明します。さて、最初に心に留めておくべきことは、Dart言語ツールで見つけたこの文であり、mixinは複数のクラス階層でクラスのコードを再利用する方法であることを示しています。

したがって、このレッスンを進めている間は、この文章を続けてください。 mixinがどのように機能するかを説明するために、コードに直接飛び込みます。

それでは、新しいクラスを作成して、それをAnimalと呼びましょう。

今、動物は共有するこの1つの行動を持ち、それは動物が動くことができるという事実です。

それでは、動物のクラスの動作、つまり正しく動く方法を記述するために、動物クラスのメソッドを構築しますか？

もしあなたが動物であるなら、あなたが動くかもしれないと、例えば、簡単にするために、あなたは正しい変化した位置を印刷するでしょう？

したがって、移動するたびに、おそらく自分の位置が変わってしまうでしょう。

したがって、新しい動物を作成する場合、animal.moveと言います。それをコンソールに配置すると、動物の位置が正しく変更されます。

それは、移動するように指示したときに、実行するように指示していることです。

ここで別のクラスを作成し、これが魚のクラスであり、このクラスが動物のクラスを継承するとします。

Dartのキーワードは拡張されているため、魚のクラスは動物のクラスから拡張されます。これは、動物のクラスにあるすべての動作にアクセスできるようになったことを意味します。

つまり、新しい魚を作成できることを意味し、移動がどのように正しく機能するかを定義したコードを書き直すことなく、魚がすぐに移動できると言えますか？

これで、メイン関数でfish.moveと言うと、位置も変わることがわかります。

そのため、このクラスから継承できるため、クラスのisコードを再利用しています。

では、別のクラスがあったとしましょう。

これは鳥のクラスであり、鳥も移動することができ、動物からも受け継ぐとしましょう。

しかし、私たちがそれを考えると、魚の動き方と鳥の動き方はまったく違いますか？

したがって、動物から継承したmoveメソッドをオーバーライドすることもできます。

そのため、@ overrideアノテーションを使用してそれを行うことができ、オーバーライドする移動メソッドを指定できます。

そして、スーパークラスが行うことを実行してみましょう。

したがって、移動のスーパークラス定義では、変更位置を出力します。しかし、それを行った後、それを呼び出した後、独自のことも行います。

水泳で位置を変えて印刷するかもしれません。

そして、私たちの鳥のクラスでは、まったく同じことをします。

単純に位置を変更するだけの親バージョンのmoveをオーバーライドしますが、その後、飛行して印刷します。

魚が動き、泳ぐことができ、鳥が動き、飛ぶことができます。

そして、fish.moveを印刷することを決めたとき、泳いで位置を変えると言うことがわかります。

そして、この場合、bird.moveを印刷すると、飛行することで位置を変更します。

これで、どのように継承できるかがわかりました。

別のクラスを作成しましょう。

さて、このクラスはアヒルのクラスになるでしょう？

そして、少なくとも野生のアヒルは、飛ぶことも泳ぐこともできます。

では、正確に何を継承しますか？魚を受け継ぐのか、鳥から受け継ぐのか？泳ぐために、また飛ぶためにアヒルが必要だからです。そのため、Dartおよび最新のオブジェクト指向プログラミング言語の多くには、単一の継承パターンがあります。

したがって、1つのクラスのみを拡張できます。

あなたが選ぶことができます。

動物を選択することも、魚を選択することも、鳥を選択することもできますが、魚を拡張して鳥を拡張して動物を拡張することはできません。

拡張できるクラスは1つだけです。

だから私たちは選択をしなければなりません。

さて、魚や鳥を広げることができたとしても、アヒルの移動方法を実装する場合は、鳥の移動方法を使用して飛行するのか、それとも魚から継承した移動を使用するのか泳ぐ？

そのmoveメソッドを2つのクラスから継承する場合、どちらのメソッドもmoveメソッドを持っているため、実際にはどちらのメソッドを取得しますか？

これが、mixinが実際に解決するものです。

moveメソッドを持つ魚のクラスとmoveメソッドを持つ鳥のクラスを持つ代わりに、実際に代わりにmixinを作成し、CanSwimと呼ぶことができます。したがって、CanSwimには基本的にswimというメソッドがあります。

そして、これを使用すると、水泳によって変化する位置を印刷します。そして、別のミックスインを作成できます

CanFlyなので、CanFlyにflyメソッドを追加し、飛行によって位置を変更できます。

したがって、アヒルは動物のクラスから継承して継承することができます。おそらく、アヒルがちょうど動き回って位置を変える基本的な動きを使用できるようにしたいのですが、mixinを追加することでCanFlyおよびCanSwim機能を与えることもできます。そして、クラス拡張の後にキーワード「with」を追加することにより、ミックスをクラスに組み込み、ミックスインの名前を指定します。

したがって、CanSwimミックスインの後にカンマが続き、CanFlyミックスインを追加できます。

そして今、私たちのアヒルクラスにはmoveメソッドがあるため、ここで書いたこのコードを再利用するだけでなく、泳いだり飛んだりすることもできます。

そのため、さまざまなクラスのコードを再利用しています。

それで、もし私たちがアヒルオブジェクトを作成するなら、それは動くだけでなく、飛ぶことも泳ぐこともできます。

これの利点は、クラスのコードを複数のクラス階層で再利用できるようになったことです。そのため、誰から継承するかは問題ではありません。誰からも継承する必要さえありません。

他のクラス用に作成したコードを、さまざまなクラスで再利用できます。

たとえば、後で飛行機という別のクラスを作成しますか？そして、飛行機は単にCanFlyのmixinを追加するだけです。

そして、すでに私の飛行機は、私が余分なコードを書かずに、ここで書いたコードを繰り返すことなく飛行できます。

さて、これらはすべて非常に些細な例です。なぜなら、私はただ1行の印刷ステートメントを入力しているだけです。水や、何かを羽ばたかせて飛ばす方法を見つけようとするとさらに悪いことになります。これには多くのコードが含まれており、モジュール式に保ちたいと思いますか？

特定のクラスに属するメソッドとプロパティ、および物の実装を1つのクラスにまとめ、それを単にミックスインとして必要とするクラスに追加するだけで、これが実装を維持するキーとなりますミックスインを使用します。 flyメソッドとswimメソッドでコーディングしたすべての機能を保持します。

これは、クラスに機能を追加する場合のドキュメントの意味です。

したがって、ミックスインは非常に強力であり、多くの場合、サブクラスを使用する代わりの方法を提供します。たとえば、魚に親のmoveメソッドをオーバーライドさせるのではなく、fishクラスをmixinとともにキーワードを使用して泳ぐことができるクラスに変更できます。

なぜこれが便利なのでしょうか？

かなり大きな利点は、いくつかの機能を共有する必要がある多くのクラスを持つ大規模なプログラムにありがちです。

その共有機能をmixinに分割すると、保守も簡単になります。

たとえば、swimメソッドを変更する必要がある場合、魚とアヒルのクラスを変更することなく、アヒルと魚の両方がその変更を認識します。

したがって、機能とプロパティを共有する多くのクラスを持つ大きなモジュールを作成し、この定義ミックスインに戻ると、複数の階層でコードとしてクラスを再利用できるようになります。

だから、あなたが継承するものやクラス階層のどこにいるのかはわかりません。両親が誰であるか、兄弟が誰であるか、子供が誰であるかは関係ありません。

これらの機能はいつでも追加できます。

新しいスキルを習うようなものですか？

あなたの両親はあなたが医者になる方法を学ぶことができるように医者である必要はありませんか？

それはあなたが受け継がなければならない何かの代わりに持つことができるスキルです。