それで、私たちは最後の柱に進みます。それは多型または変化する形状です。

これはどのように作動しますか？

プログラムを抽象化し、個別のロール用に個別のモジュールを作成しているとしましょう。

そして、親クラスから多くの動作を継承しています。

しかし、多くの継承を行うと、基本的に親クラスが実行できることのコピーを取得することがわかります。

したがって、私たちの電気自動車は運転でき、電気自動車にも同じ特性があります。しかし、私たちのプログラムでクローンの束だけになってしまうことは本当に望ましくありません。それは素晴らしいことではありませんか？

顧客の行動も定義できるようにしたいと考えています。

それで、私たちのシェフとパティシエの場合、おそらく、これら2つの役割の間には類似点があります。

彼らはおそらくオーブンで何かを焼くことができ、おそらく鍋でソースを作ることができます。

しかし、彼らはそれらの振る舞いについて少し異なる方法で行くかもしれません。

だから、おそらくパティシエは彼が働いていることに対してもう少し繊細である必要があり、そしてシェフは多分、彼女は台所で完全な騒乱であるため、彼女は少し速く物事をしなければならないでしょう。

しかし、ポイントは、親クラスから継承している場合、おそらく時間を節約するために、再作成する必要がなく、単純に継承できることです。

ただし、継承する動作の一部、メソッドの一部は、独自のニーズに合わせてカスタマイズすることができます。

これを行うには2つの方法があります。

車の類推を続けると、単にLevitatingCarがあり、このLevitatingCarはCarクラスから継承します。

Carクラスを拡張するように記述します。これは、おそらく空中浮揚車にも5席あるため、車とほとんど同じように見えるためです。同じ特性を共有するかもしれませんが、少なくとも私の考えでは、空中浮揚車には車輪がありません。

代わりに、空中浮揚車の駆動機能を実装する場合、おそらく少し異なります。ドライブメソッドを持つCarクラスから既に継承しているため、同じメソッドを使用する場合は、それをオーバーライドする必要があります。そのためには、@記号を使用して、非常に重要な単語が来ることを指定します。単語は上書きされます。そして、次の行では、独自のバージョンのドライブを作成して、空のドライブを書き込むことができ、以前とまったく同じように見えます。しかし、今では、車輪を動かすのではなく、独自の駆動方法と空中浮揚車です。非常に上品に、単に前方に滑るだけです。

浮上車を作成し、浮上車であるMagLevを運転するように要求すると、ドライブの親実装を使用する代わりに、前方に滑空する必要があることがわかります。ですから、私は両親から多くの特性と行動を引き継いでいますが、私は彼らとは違うことをしたり、違うやり方をしたいと思っています。

たとえば、コンピューターの使用方法では、コンピューターを使用してプログラミングするのに対して、両親はそれを使用してFacebookにアクセスする場合があります。

しかし、ポイントは、親の動作をオーバーライドして独自のカスタムバージョンを提供できるようになったことですが、実際にはそれ以上のことを行うことができます。

実際に親クラスからいくつかの良い部分を取り出すことができますが、単に追加するだけです。

私の言いたいことをお見せしましょう。

新しいクラスを作成しましょう。これは自動運転車になります。

そしてもちろん、それは車です。

そのため、車のクラスを拡張します。

駆動方法があり、5つの座席があります。とても簡単。完了しました。

しかし、運転して車輪を回すことができることに加えて、

また、自動運転車も操縦できるようにしたいです。単に車輪を回すだけでなく、自動運転車で目的地に向かって運転します。

そこで、destinationという新しいプロパティを作成します。この目的地は、自動運転車から新しいオブジェクトを作成するときに設定されます。

カスタムコンストラクターを定義するので、自動運転車を作成するときは、目的地を指定する必要があります。そして、そのuserSetDestinationを呼び出します。

そして、自動運転車を作成したら、destinationプロパティを、新しいオブジェクトを作成したときに提供されたユーザー設定の宛先に等しく設定します。

今、この自動運転車の内部で、私たちがそれを運転することに決めたとき、私たちは一種の車輪の動きである運転の機能を継承したいと思います。

しかし、私たちもそれに追加したいと思います。親メソッドのコードドライブを再びオーバーライドできます。

したがって、まったく同じに見える必要があります。

また、空のドライブの空の括弧です。

そして、スーパークラスまたはparent.driveを表すsuperを呼び出して、親の動作をトリガーします。

これにより、スーパークラスのドライブメソッドの動作が実行されます。

しかし、その後、私たちは自分のことをするつもりです。私たちは、ステアリングが目的地に向かって私たちを導くだろうと言うつもりです。

メインでこのメソッドを実際に呼び出すと、次のようになります。したがって、新しい自動運転車を作成できます。これをmyWaymoと呼びます。そして、それは新しい自動運転車と同等になるでしょう。そして、myWaymoを新しい自動運転車として作成する場合、目的地を指定する必要があります。

それでは、1 Hacker wayに向かってみましょう。

そして今、myWaymo.driveがそのドライブメソッドを呼び出していると言うと、runを押すと、2つのことが起こることがわかります。

1つは、ドライブの親バージョンを実行することです。これは単に「ホイールターン」を印刷することですが、目的地に向かって操縦する独自のことも行います。

そして、これはすべて、その1つのドライブメソッドを呼び出すだけです。

これが実際のポリモーフィズムです。

両親から継承することはできますが、改善することもできます。

メソッドをオーバーライドすることで、少し異なるように変更できます。また、Flutterコードに戻ると、buildメソッドを呼び出すたびにそれを行っていることがわかります。

そのため、ステートレスウィジェットにはすでにbuildメソッドの実装がありますが、コードではそれをオーバーライドし、代わりにやりたいことを提供します。

このように、継承している場合でも、少し異なるものにしたり、ステートレスウィジェットやステートフルウィジェットの動作をカスタマイズしたりできます。

したがって、最良の部分を取得するだけでなく、これらの継承されたクラスを使用してカスタム処理を実行します。