抽象化とカプセル化についてすでに説明し、実際に動作しているのを見たので、次はOOPの次の柱である継承です。継承とは、クラスが親からプロパティとメソッドを継承できることを指すプログラミングのことです。

たとえば、あなたは両親から目の色を引き継いだり、きちんとしたフリークのように特定の行動を引き継いだり、本当にすべてをきちんと保つ必要があります。

お母さんやお父さんからもらったかもしれません。

同様に、すべてのクラスはスーパークラスまたは親クラスから継承でき、メソッドだけでなくプロパティも継承できます。

そして、これは重要です。プログラムを抽象化し、多くの異なるモジュールを作成し、すべてが特定のタスクを処理する多くの異なるコンポーネントを作成しているからです。

機能が重複している可能性があります。

したがって、たとえば、パティシエとシェフには、おそらくいくつかの類似の機能があります。

おそらく彼らは両方とも水を沸かす方法を知っています。

彼らはおそらく何かを焼くための方法を持っていますが、わずかに異なる方法でそれを行うかもしれません。

したがって、多くの異なるクラスと個別のモジュールに抽象化するときに大量のコードを書き直さなくて済むように、オブジェクトを作成するときに継承が本当に重要なのはこのためです。これをデモするために、空のDartPadに入り、新しいクラスを作成します。

Carクラスを作成し、Carクラスにいくつかのプロパティがあるとします。

すべての車は、おそらく多くの座席の所有権で始まりますか？

そして、デフォルトとして5に設定します。

私が作るすべての車は、5席から始めます。

かなり標準的なようです。

そして、すべての車にも駆動方法があります。そうでない場合、なぜ車があるのですか？そして、駆動メソッドを呼び出すとき、それは単に車輪を回すだけです。

そこで、「車輪が回る」という印刷文を作成します。そして今、Carクラスからcarオブジェクトを作成したら、新しいcarを作成しましょう。それをmyNormalCarと呼びましょう。少し後で、明らかにこれをさらに充実させます。そのため、myNormalCarは沼地の標準的な車のクラスから作成されます。

したがって、たとえばmyNormalCar .numberOfSeatsを印刷すると、5が得られることがわかります。myNormalCarと言うと。駆動すると、駆動機能が実行され、ホイールが回転し始めます。

これはすべて非常に標準的なものであり、クラスオブジェクトがこのように機能する方法に精通しています。

しかし、別のタイプの車を作成したい場合はどうなりますか？

電気自動車を作りたい場合はどうなりますか？

電気自動車は車に似ていますよね？

また、おそらく5つの座席があり、おそらく同様に駆動できますが、他の機能を備えている可能性があります。

先に進み、新しいクラスを作成してみましょう。このクラスはElectricCarになります。

しかし、私は本当にこれらすべてを再度入力する必要はありません。

普通の車と同じです。

しかし、通常の車とどう違うのかを説明することに時間を集中したい。

それで、車で手に入れたすべてのものをどのように継承できますか？

Dartでは、キーワードは拡張されています。

それで、私の電気自動車は私の​​Carクラスを拡張します。

そしてこれが意味することは、デフォルトで私のElectricCarrは、車が持っているすべてを持っているということです。

だから私は先に進んで新しい電気自動車を作ることができました。

それをmyTeslaと呼び、新しい電気自動車として作成しましょう。そして今、私はまっすぐに言うことができます

myTesla.driveを実行すると、ホイールが回転することがわかります。

私たちの車に関連していた以前のものをコメントアウトして、コンソールで混乱しないようにしましょう。このmyTeslaドライブは、その電気自動車のすべての機能を継承しているため、電気自動車が車輪を回転させるのをトリガーできますその親クラスであるCarクラス、および開始時のデフォルトの座席数もあります。

しかし、Carクラスを継承することで、電気自動車を引き起こすためによくあることをすべて繰り返し入力する必要がなくなり、異なるコードを作成することに時間を集中できます。

たとえば、私の電気自動車にはバッテリー残量があり、100％で起動します。しかし、ダウンしたら電気自動車を充電できるので、充電と呼ばれる方法を使用できます。

これにより、バッテリーレベルが100％にリセットされるので、Teslaドライブだけでなく、充電もできるようになり、車が持っていることを繰り返す必要がなくなりました。

拡張キーワードを使用するだけで、プロパティとメソッドを組み込むことができました。

これが継承の仕組みであり、コードでもこれを確認できます。

新しいステートフルウィジェットまたはステートレスウィジェットを作成するたびに、ステートレスまたはステートフルウィジェットのすべてのプロパティまたは機能を作成する必要はありません。

親クラスをステートフルまたはステートレスウィジェットに拡張するクラスを作成することで、簡単にそれを利用することができました。

また、WindowsでCOMMANDまたはCONTROLを押したままそのクラスをクリックすると、さまざまなプロパティとメソッドがあり、それらを拡張しているために単純に使用できることがわかります。私たちはそれを継承しており、親クラスとは異なるすべてのものを記述し、画面にウィジェットを描画する方法などの基本的な退屈なことではなく、カスタム機能の構築に焦点を当てるだけです。

別の色にするにはどうすればいいですか？

形状とサイズをどのように変更しますか？

実際にどのようにピクセルを移動しますか？

これらはすべてFlutter SDKによって整理されており、その機能はすべて、継承可能なこれらの親ウィジェットにパッケージ化されています。次のレッスンでは、最後の柱であるポリモーフィズムについて少しお話したいと思います。

それで、そのすべてについて、次のレッスンでお会いしましょう。