4.3 継承

継承とは、もとのデータ型である基底クラスを使って、部分的に機能を変えたり、拡張をしたりするための機能です。継承はオブジェクト指向プログラミングにおいて重要な概念ですが、本書では継承の概念については説明せず、Pythonでの継承の使い方について説明します。

4.3.1 標準ライブラリの継承の例

Pythonの標準ライブラリでも継承が使われています。ここでは、「11.1　ファイルパス操作を直観的に行う ― pathlib」（p.246）で紹介するpathlibライブラリを例に説明します。

pathlibライブラリは、OSごとに異なるファイルシステムのパスを基本的に同じインターフェースで操作できるようになっているライブラリです。このライブラリの基底クラスは、PurePathというものです。この基底クラスから継承した、Windows用のPureWindowsPathと、Windows以外向けのPurePosixPathがあります。

継承関係を示す図がpathlibの公式ドキュメントにありますので、参照してください。

- https://docs.python.org/ja/3/library/pathlib.html

◆親クラスと子クラス

継承関係にある継承クラスは、基底クラスのデータ属性やメソッドをすべて引き継ぐことになります。よって、基底クラスを「親クラス」、継承クラスを「子クラス」と呼ぶことができます。

PurePathクラスが親クラスで、PureWindowsPathクラスやPurePosixPathクラスが子クラスとなります。つまり、PureWindowsPathクラスやPurePosixPathクラスは、PurePathクラスのすべての機能を有し、子クラスごとに特化した機能を持っているクラスが定義されています。これは、共通するデータ属性やメソッド定義を親クラスの1ヵ所に集約できることになります。さらに子クラスに特化した機能のみを子クラスに定義するということになります。

親クラスのPurePathクラスに定義されている`parts`データ属性や`is_absolute`メソッドは、子クラスのPureWindowsPathやPurePosixPathからも利用できます。

4.3.2 子クラスの定義

独自に定義したクラスやライブラリの既存クラスを親クラスとして継承した、新たなクラスを定義できます。子クラスに必要な範囲のみを実装することで、少ないコード量で目的の機能を実現できます。

◆子クラスの定義方法

子クラスの定義にもclassキーワードを使用して新たなクラスを定義します。その際、親クラス名を子クラス名のあとのカッコ内に記述します。構文は以下のとおりです。

継承構文

```
class 子クラス名(親クラス名):
    属性 = 値

    def メソッド名(self):
        メソッドの処理など
```

属性を定義する場合、親クラスに存在する属性は値が上書きされ、親クラスに存在しない属性は子クラスにのみ存在する新たな属性となります。メソッドも同様で、親クラスに存在する場合はメソッド定義が上書きされ、そうでない場合は新たなメソッドとなります。

◆子クラスの具体例

ここでは、「16.2　ユニットテストフレームワークを利用する — unittest」（p.371）で紹介するテストフレームワーク unittest.TestCase を継承する例を用います。

unittest を使う場合には、以下のような子クラスを定義します。

unittest を使う場合には本来実装コードをテストするのですが，ここではクラス継承の説明をするためだけに必ず unittest が成功するコード例を示します。

unittest を継承した独自テスト

```
import unittest

class TestSample(unittest.TestCase):  unittest.TestCaseを親クラスとした子クラスを定義
    def setUp(self):  unittest.TestCaseが提供するテスト事前準備用メソッドを上書き
        self.target = 'foo'

    def test_upper(self):  新たなメソッドを定義し、テスト実行対象のメソッドとなる
        self.assertEqual(self.target.upper(), 'FOO')  unittest.TestCaseが提供するassertEqu
alメソッドを使ってテストする
```

子クラスの定義方法に従って、class キーワードのあとに子クラス名を書き、カッコ内に親クラスを設定しています。次の setUp() メソッドは、親クラスの unittest.TestCase に定義されているテストの初期化処理を行うメソッドです。このメソッドを今回定義するクラス用に上書きします。さらに、新規のメソッドとして、test_upper() を定義します。これは unittest.TestCase がテスト対象にするために、メソッド名を test_ から始めています。このメソッド内では、self.assertEqual() メソッドが使われています。このメソッドは、親クラス unittest.TestCase に定義されているメソッドです。

◆super() 関数

子クラスでメソッドを上書きする際に、親クラスに実装されているメソッドも実行したい場合があります。

unittest.TestCase の setUp() メソッドを共通化する例を示します。ここでは、継承関係と super() 関数の説明をするための例を示しています。setUp() メソッドの一部を共通化して部分的に子クラスに必要な初期化を行い、tearDown() メソッドは親クラスで共通化しています。

mypy.ini

```
[mypy]
follow_imports = silent
ignore_missing_imports = True
disallow_any_generics = True
disallow_untyped_calls = True
disallow_untyped_defs = True
warn_unused_ignores = True
warn_return_any = True
check_untyped_defs = True
```

特定のモジュールでのみ有効なオプションを設定する場合は「mypy- モジュール名」セクションに記載します。以下の例では、サードパーティのライブラリであるaiohttpにfollow_importsを設定し、同じくサードパーティのライブラリであるSQLAlchemyにignore_missing_importsを設定しています。

mypy.ini

```
[mypy-aiohttp.*]
follow_imports = skip

[mypy-sqlalchemy.*]
ignore_missing_imports = True
```

5.2.5 mypy：よくある使い方

mypyの利用方法はいくつかあります。開発環境やリリース方法などに合わせて利用しやすい方法を選択しましょう。

- コマンドラインで利用する方法
- IDEやエディターで利用する方法
- pre-commitやtox、CIを組み合わせ自動的に利用する方法

5.2.6 mypy：ちょっと役立つ周辺知識

◆mypy以外の静的型チェックツール

mypy以外にも静的型チェックツールがあります。ここではmypy以外のPythonの静的型チェックツールを紹介します。

表：静的型チェックツール

ツール名	リポジトリ管理	URL
Pyright	Microsoft	https://github.com/microsoft/pyright
Pyre	Facebook	https://pyre-check.org/
pytype	Google	https://google.github.io/pytype/

5.2.7 mypy：よくあるエラーと対処法

サードパーティのライブラリは型ヒントに対応していないことがよくあります。ここではサードパーティのライブラリ利用時によく発生する2つの型ヒントに関するエラーの対処法を紹介します。

- `error: Library stubs not installed for "サードパーティライブラリ名"`
- `error: Cannot find implementation or library stub for module named "サードパーティライブラリ名"`

それぞれ日本語では以下のような内容です。

- エラー：ライブラリのスタブファイルがインストールされていない
- エラー：型の実装、またはライブラリのスタブファイルが見つからない

上記のエラーメッセージに出てくるスタブファイルとは、型ヒントを定義している型定義ファイル（拡張子が.pyiのファイル）です。型ヒントはソースとは別に、外部ファイルであるスタブファイルに型情報を定義できます。mypyはこのスタブファイルも静的型チェックに利用します。

サードパーティのライブラリの一部は、型定義のリポジトリであるtypeshed（https://github.com/python/typeshed）にスタブファイルが登録されています。

◆ライブラリのスタブファイルがインストールされていない

このエラーは、型定義のリポジトリであるtypeshedにスタブファイルが存在する場合に出力されます。エラーの対処法は、pipコマンドや「`mypy --install-types`」コマンドで該当のライブラリの型定義をインストールすることです。

◆型の実装、またはライブラリのスタブファイルが見つからない

このエラーは、型の実装がないか、ライブラリのスタブファイルが見つからない場合に出力されます。このエラーが出力された場合、以下の順に対応を検討しましょう。

1. 該当のライブラリを最新にアップデートする
2. 該当のライブラリにスタブファイルがあるか確認する
3. 該当ライブラリをmypyのチェックから除外する

該当ライブラリをmypyのチェックから除外するには、以下のように記述します。

```
import example  # type: ignore
```