

2.9.8 メソッドのオーバーライドと super()

基底クラスで定義したメソッドと同じ名前のメソッドを拡張クラス側で定義すると、拡張クラス側のメソッドが基底クラスのメソッドをオーバーライドする。すなわち、拡張クラス側で実行されるメソッドは、そのクラスで定義されたものであり、基底クラス側のメソッドは無効になる。ただし `super` を用いると基底クラス側のメソッドを呼び出すことができる。このことを次の `overrideTest1.py` を例に挙げて解説する。

プログラム：`overrideTest1.py`

```
1 class A:
2     def method1(self):
3         print('method1 of class A')
4
5 class B(A):
6     def method1(self):
7         super().method1()
8         print('method1 of class B')
9
10 class C(B):
11     def method1(self):
12         super().method1()
13         print('method1 of class C')
14
15 print('クラス A のインスタンスに対する method1() の実行。')
16 a = A()
17 a.method1()
18
19 print('クラス B のインスタンスに対する method1() の実行。')
20 b = B()
21 b.method1()
22
23 print('クラス C のインスタンスに対する method1() の実行。')
24 c = C()
25 c.method1()
```

このプログラムでは、クラスが順に $A \rightarrow B \rightarrow C$ と派生しており、全てのクラスで同一の名前のメソッド `method1` が定義されている。各クラスのインスタンスにこのメソッドを実行すると、当該クラスの `method1` が実行される。しかし、`b`、`c` に対して `method1` を実行すると、その定義に記述された

```
super().method1()
```

によって、1つ上の基底クラスの `method1` も実行される。「`super()`」は **super オブジェクト**であり、ここでは、当該インスタンス (`self`) を 便宜的に1つ上の基底クラスのものとして解釈できる。詳しくは後の「2.9.8.1 `super` オブジェクト」(p.159) で解説する。

`super()` によって、オーバーライドされて無効になった基底クラスのメソッドを実行することができ、`overrideTest1.py` を実行すると次のような出力が得られる。

実行例.

```
クラス A のインスタンスに対する method1() の実行.
method1 of class A
クラス B のインスタンスに対する method1() の実行.
method1 of class A      ←親クラスの method1 を遡及実行
method1 of class B      ←自クラスの method1 による出力
クラス C のインスタンスに対する method1() の実行.
method1 of class A      ←先祖クラスの method1 を遡及実行
method1 of class B      ←親クラスの method1 を遡及実行
method1 of class C      ←自クラスの method1 による出力
```

クラス B のインスタンスに `method1` を実行すると、クラス A に定義された `method1` が実行されていることがわかる。また、クラス C のインスタンス `c` に対して `method1` を実行すると、クラス A の `method1` まで遡及して実行されていることがわかる。

`super()` はクラスメソッドにおいても使用できる。このことを次の `overrideTest2.py` で示す。

プログラム：overrideTest2.py

```
1 class A:
2     @classmethod
3     def method1(cls):
4         print('method1 of class A')
5
6 class B(A):
7     @classmethod
8     def method1(cls):
9         super().method1()
10        print('method1 of class B')
11
12 class C(B):
13     @classmethod
14     def method1(cls):
15         super().method1()
16         print('method1 of class C')
17
18 print('クラス A に対する method1() の実行. ')
19 A.method1()
20
21 print('クラス B に対する method1() の実行. ')
22 B.method1()
23
24 print('クラス C に対する method1() の実行. ')
25 C.method1()
```

このプログラムでは、先の overrideTest1.py と類似の考え方で上位のクラスのクラスメソッド method1 を遡及して実行する。overrideTest2.py を実行すると次のような出力が得られる。

実行例.

```
クラス A に対する method1() の実行.
method1 of class A
クラス B に対する method1() の実行.
method1 of class A
method1 of class B
クラス C に対する method1() の実行.
method1 of class A
method1 of class B
method1 of class C
```

2.9.8.1 super オブジェクト

先に super オブジェクトのことを「当該インスタンス (self) を便宜的に 1 つ上の基底クラスのものとして解釈できる」と解説したが、正確には、祖先のクラスを参照するための特殊なオブジェクトである。実際に super() が返す値のクラスは super である。これは次のプログラム overrideTest3.py で確認できる。

プログラム：overrideTest3.py

```
1 class A:
2     def method1(self):
3         print('method1 of class A')
4
5 class B(A):
6     def method1(self):
7         print('type(super()) :', type(super()))
8         super().method1()
9         print('method1 of class B')
10
11 b = B()
12 b.method1()
```

このプログラムでは、クラス B のメソッド method1 の中で

```
type( super() )
```

として super オブジェクトのクラスを調べ、それを出力する。このプログラムを実行すると次のような出力が得られる。

実行例.

```
type(super()) : <class 'super'>          ← super オブジェクトのクラス (型)
method1 of class A
method1 of class B
```

このプログラムからもわかるように、「`super()`」は クラス `super` のコンストラクタ である。またこのコンストラクタは、実行される状況から適切に基底クラスを参照する。また、クラス `super` のコンストラクタには次のように引数を与えることもできる。

書き方： `super( クラス, インスタンス )`

「クラス」「インスタンス」の情報から、「クラス」の基底クラスを参照する `super` オブジェクトを作成して返す。これに関して次のようなプログラム `overrideTest4.py` を示して解説する。

プログラム： `overrideTest4.py`

```
1 class A:
2     def method1(self):
3         print('method1 of class A')
4
5 class B(A):
6     def method1(self):
7         super().method1()
8         print('method1 of class B')
9
10 class C(B):
11     def method2(self):
12         super(B,self).method1()    # クラス B の親 (つまりクラス A) を参照
13         print('method2 of class C')
14
15 print('クラス C のインスタンスに対する method1 () の実行. ')
16 c = C()
17 c.method1()
18
19 print('クラス C のインスタンスに対する method2 () の実行. ')
20 c.method2()
```

このプログラムでは、クラス `C` のメソッド `method2` の中に

```
super(B,self)
```

という記述がある。これは「クラス `B` の基底クラスのインスタンス」と解釈することができ、結果としてクラス `A` のインスタンスと見なして `method1` を実行している。このプログラムを実行すると次のような出力が得られる。

実行例.

```
クラス C のインスタンスに対する method1 () の実行.
method1 of class A          ←これはクラス B から継承された method1
method1 of class B          による出力
クラス C のインスタンスに対する method2 () の実行.
method1 of class A          ←クラス A の method1 を直接的に実行
method2 of class C
```

実行結果から、クラス `C` 内からクラス `B` の `method1` を実行することなく、クラス `A` の `method1` を実行していることがわかる。