第5回：『科学的モデリング』継承⑤～継承パラドックス

第5回の話題～「継承パラドックス」

第4回のコラムでクラスよりも「タイプ(型)」を意識して継承を考えることの重要性を解説しました。

そして、スーパークラスとサブクラス間に「is-a」関係／「a-kind-of」関係が成立するか否かで、継承関係の正当性をチェック方法は非科学的なガイドラインであり、判断基準(ガイド)とは言えないと解説しました。

「is-a」関係／「a-kind-of」関係による判断基準(ガイド)が、非科学的なガイドラインであることの理由とそれに代わる科学的な方法についての解説は、今回(第6回)を含めて今後のコラムで連続して行います。

今回は科学的な方法の解説のプロローグとして、ちょっとした継承の問題提起を行います。

この問題提起は：

「継承パラドックス(inheritance paradox)」

と呼ばれるものです。

今回のコラムのテーマは：

「継承パラドックス(inheritance paradox)」から「継承の科学的判定基準」の必要性を理解する

です。

正方形と長方形の継承関係を考える

図形の長方形と正方形をとりあげ、簡単なクラスを作成し検討してみます。第4回でクラスよりもクラスの「タイプ(型)」を意識して継承関係を考えると良いと解説しましたが、ここからしばらくは、「タイプ(型)」を意識することはひとまず脇においておきます。

図形の「種類」に注目して継承を考える

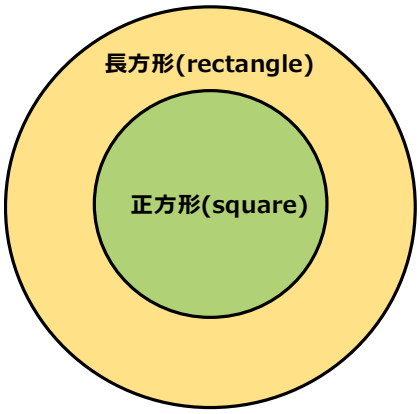
長方形と正方形の関係は、数学的な図形の「種類」の視点からベン図を描いて表現すれば【図5-2】のようになります。

正方形は長方形の一種ですから、【図5-2】のような包含関係が成立します。

これは長方形と正方形の間に「is-a」関係／「a-kind-of」関係が成立するということを意味します。

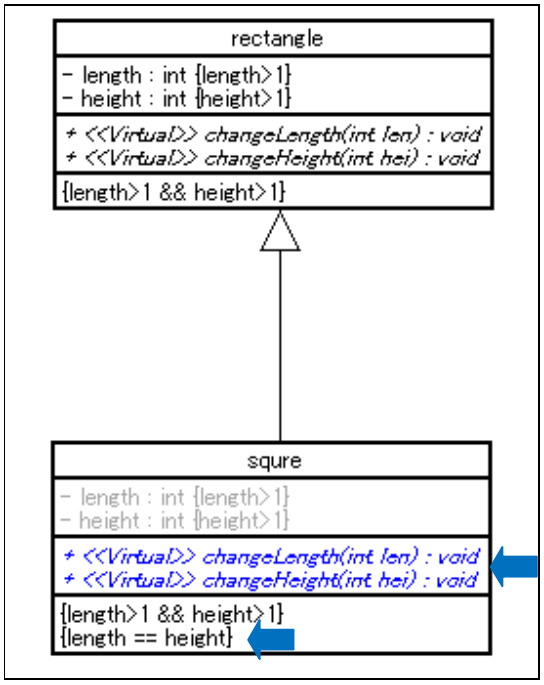
多くの書籍やセミナーでは、２つのクラス間に「is-a」関係／「a-kind-of」関係が成立しなければならないと解説しています。それでは、この【図5-2】のような長方形と正方形の間に継承関係を持ち込めば正しい継承関係となるのでしょうか？

それでは検討していきましょう。



【図5-2】

【図5-2】の関係に沿ってクラス「rectangle(長方形)」とクラス「square(正方形)」の継承関係を表現すると【図5-3】のようになります。



【図5-3】

【図5-3】のクラス図の継承関係を見て直ぐに気づくのは、クラス「square(正方形)」の属性にクラス「rectangle(長方形)」から継承した属性「length」と「height」があることです。

正方形は辺の長さが等しい訳ですから、辺の長さを表現する属性は本来１つで済みます。よって、これは不自然と言えるでしょう。

第3回で特性(プロパティ)は全て継承されると解説しました。

そのため、【図5-3】のクラス「square(正方形)」は、クラス「rectangle(長方形)」から「属性と属性の不変条件」「操作」「クラス不変条件」の全てを継承しています。しかし、クラス「square(正方形)」にとって辺の長さを表現する属性は本来であれば１つで済みます。そのため、属性名「length」「height」も特性を適切に表現している名前とは言えません。

操作について見てみると、クラス「rectangle(長方形)」から２つの操作「changeLenght(int len):void」と操作「changeHeight(int len):void」を継承しています。

この２つの操作は引数に変更する長さを取り長方形の辺の長さを変更する操作です。既に述べた様に正方形は辺の長さが等しい訳ですから、辺の長さを変更する操作は本来１つで済みます。

以上を考慮するとサブクラスであるクラス「square(正方形)」の定義が、ゆがんで不自然になっている印象がいなめません。

この原因はサブクラスであるクラス「square(正方形)」が、スーパークラスであるクラス「rectangle(長方形)」の特性(プロパティ)を継承していることが原因であることは明白です。

特に注目するべき点は属性「length」「height」をそれぞれ異なる値(長さ)に出来る長方形とは異なり、正方形は辺の長さが等しくなる必要があるので、「クラス不変条件(class invariant)」に{length == height}を追加せざるおえなくなっています。なお、「クラス不変条件(class invariant)」については第3回のコラムを参照してください。

図形の「特性(プロパティ)」に注目して継承を考える

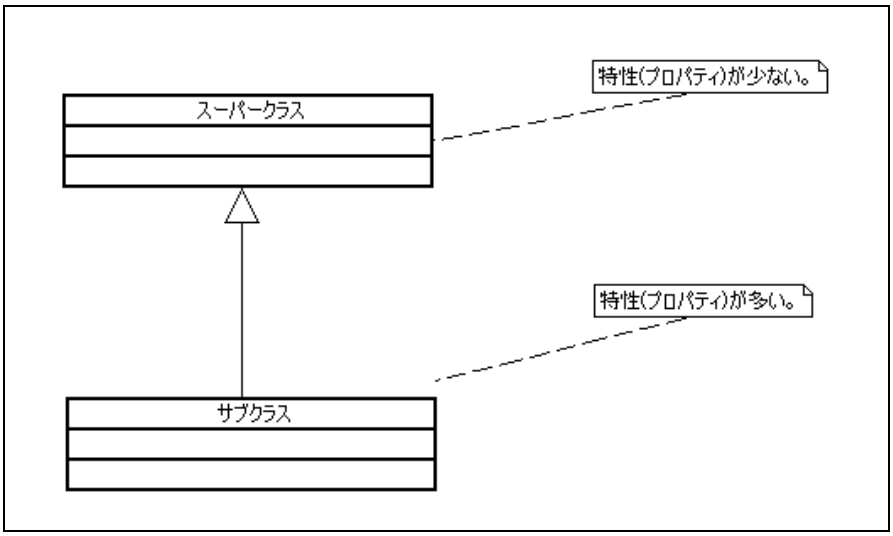
ここまでは、図形の長方形と正方形の継承関係を図形の「種類」に注目して考えてきましたが、オブジェクト指向では、継承関係を考える時にクラス間の特性(プロパティ)に注目して検討することも多くあります。

今度は図形の「種類」ではなく、長方形と正方形の持つ「特性(プロパティ)」に注目して検討して継承関係を考えてみます。

【図5-4】は特性(プロパティ)に注目した場合の継承関係の特徴について表現しています。

特性(プロパティ)に注目した場合は、スーパークラスはサブクラスと共通の特性(プロパティ)を持ち、サブクラスの方は、自分の固有の特性(プロパティ)を追加することで特性(プロパティ)が多くなります。

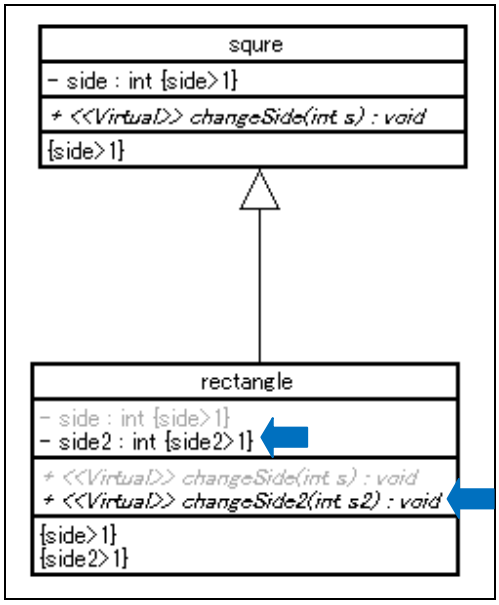
スーパークラスはサブクラスの継承関係においてこれは自然なことです。ただし、この場合は原則２つのクラス間に「is-a」関係／「a-kind-of」関係が成立しません。



【図5-4】

クラス「rectangle(長方形)」とクラス「square(正方形)」の特性(プロパティ)に重点を置き継承関係を表現すると、【図5-5】のようになります。

この図から分かる様に特性(プロパティ)に注目した継承関係の場合とでは、長方形と正方形の継承関係の位置が逆転します。



【図5-5】

【図5-5】ではクラス「rectangle(長方形)」がクラス「square(正方形)」から「属性と属性の不変条件」「操作」「クラス不変条件」を継承しています【図5-3】とは逆です。

クラス「square(正方形)」は辺の長さを表現する属性は１つで済むので、クラス「rectangle(長方形)」で属性「side2」を追加してタイプ(型)を「拡張(extension)」しています。

長方形は「横(Length)」と「縦(Height)」の辺をもつので、クラス「rectangle(長方形)」の属性名が「length」と「height」ならより自然だったでしょう。

操作を見てみると、長方形は縦と横の辺の長さが異なることために、クラス「rectangle(長方形)」が継承した操作「changeSide(int len):void」に加えて、操作「changeSide2(int len):void」追加してタイプ(型)を「拡張(extension)」しています（タイプ(型)の「拡張(extension)」については第4回のコラムを参照してください）。

クラス「rectangle(長方形)」操作の名称についても、長方形は「横(Length)」と「縦(Height)」の辺の長さを変更する操作名にしたいところですから、属性と全く同様の事が言えます。

【図5-5】の継承関係は属性名と操作名にやや不自然さが存在しますが、【図5-3】よりもかなり単純な印象を受けます。

なぜ２つの継承関係が考えられるのか？

クラス「rectangle(長方形)」、クラス「square(正方形)」の継承関係を考える時に、図形の「種類」あるいは図形の「特性(プロパティ)」の２つの視点が存在し、それにより全く逆の継承関係が考えられることが分かりました。

このように今回２つの継承関係が考えられる理由は、クラスには「種類」と「特性(プロパティ)」の全く異なる性質(character)が同居しているからです。

異なる性質(character)とは、簡単に言ってしまえば；

- **「種類」＝タイプ(型)**
- **「特性(プロパティ)」＝機能**

です。

つまり、継承関係を「種類」に注目して考えるか、あるいは「特性(プロパティ)」に注目して考えるかという問題の本質的な意味は；

- **継承関係を「タイプ(型)」に注目して考える**
- **継承関係を「機能」に注目して考える**

ということになります。

クラスの「タイプ(型)」と「機能」の性質を併せ持つとは、光が波動と粒子の性質を持つのと同じくらい、重要なことです。

「継承パラドクス」

クラス「rectangle(長方形)」、クラス「square(正方形)」の継承を考える時に、２つの継承関係が考えられ、スーパークラスとサブクラスの関係が逆になる継承の現象を**「継承パラドックス(inheritance paradox)」**[文献5-2]と呼んでいます。

上記で

- 「種類」＝タイプ(型)
- 「特性(プロパティ)」＝機能

と書きましたが、より的確な表現をすれば【図5-3】と【図5-5】の２つの継承の考え方が存在するのは、クラスは「抽象データータイプ(型)(abstract data type)」であるので、「モジュール(機能)」と「タイプ(型)」の両方の性質が備えているからです。

クラスを「タイプ(型)」に焦点をあてて**概念の包含関係**を強く意識して継承を考えるか、あるいはクラスの**データ構造やコード**に焦点をあて「モジュール」を強く意識して継承を考えかにより、スーパークラスとサブクラスの関係が逆になる継承の現象が**「継承パラドックス(inheritance paradox)」**です。

実は数学的な概念(正三角形と二等辺三角形、円と楕円など)間の継承関係は、この「継承パラドックス(inheritance paradox)」が存在します。

言語設計者のBertrand Meyerは【図5-3】のタイプの継承を**「制約継承」**、【図5-5】のタイプの継承を**「拡張継承」**と分類し名前をつけています(第2回コラム参照、[文献5-2]も参照)。

- 「制約継承」とはサブクラスで**制約を追加する**継承関係：
  - タイプ(型)から見れば、サブクラスで追加された制約を除けばスーパークラスとサブクラスタイプのタイプ(型)は**同型**です(第4回コラム参照)
  - 逆に言えば、サブクラスで**制約を追加する継承**になります
  - サブクラスは継承した操作を再定義(オーバーライド)しているので、スーパークラスのタイプ(型)をサブクラスで**「特殊化(specialization)」**していることになります(第4回コラム参照)
- 「拡張継承」とは特性(プロパティ)を追加し拡張を行う継承関係：
  - タイプ(型)から見ればサブクラスで特性(プロアパティ)が追加されているのでサブクラスのタイプ(型)は**「拡張(extension)」**されています(第4回コラム参照)
  - クラスをモジュールの視点からみても属性と操作がサブクラスで追加されているので機能の拡張がされています

さて、「継承パラドックス(inheritance paradox)」を紹介しましたが、実は「継承パラドックス」それ自体が悪い事ではありません。

クラス「rectangle(長方形)」、クラス「square(正方形)」の継承関係において、【図5-4】と【図5-5】を示しましたが、継承関係として、このどちらが正解であり、もう片方が不正解ということは自動的に決まりません。

「継承パラドックス(inheritance paradox)」の存在は、継承関係を考える時に

- **「私達に何が目的なのか？」**
- **「何を基準にするのか？」**

を明確にすることを私達に示していると言えます【表5-7】。

実は、継承関係は色々な種類が存在します。そして、どの継承関係の種類を利用するかを検討する上で、「継承関係を理由する目的は？」「継承関係によりどのような利益を得たいか」を明確にすることが大切です。

「継承パラドックス(inheritance paradox)」が示すこと
● 何が「目的」で２つのクラス間の継承関係を定義するのか？
● 何を「基準」に正しい・効果的な継承関係であると判断できるのか？

【表5-7】

Bertrand Meyerは、継承の効果的な使い方を[文献5-2]の中で「継承グループ」「継承カテゴリー」として分類しています。そして各「継承カテゴリー」について明確な定義と基準を記述しています【表5-8】。

【表5-8】にある継承関係では、クラス間に「is-a」関係／「a-kind-of」関係が存在しないもが多数存在します。

彼は継承関係を使う時には、「継承グループ」「継承カテゴリー」を正確に理解し、必ず分類されているどの継承関係のカテゴリーに含まれるのかを明確にした上で利用することを勧めています。

Bertrand Meyerの継承グループとカテゴリーの分類	
モデル継承 (model inheritance)	部分型継承(subtype inheritance.)
	ビュー継承(view inheritance)
	制約継承(restriction inheritance)
	拡張継承(extension inheritance)
ソフトウエア継承 (software inheritance)	具体化継承(reification inheritance)
	構造継承(structure inheritance)
	実装継承(implementation inheritance)
	定数継承(constant inheritance)
	マシン継承(machine inheritance)
バリエーション継承 (variation inheritance)	ファンクショナルバリエーション継承(functional variation inheritance)
	タイプ(型)バリエーション継承(type variation inheritance)
	無効化継承(uneffecting inheritance)

【表5-8([文献5-2]より)】

『科学的継承関係』の考え方が必要

Bertrand Meyerの「継承グループ」「継承カテゴリー」でも分かる通りに、継承関係の種類(カテゴリー)は多く存在します。

継承関係を用いる目的や状況に応じて正確に継承を使い分けるスキルを身に着ける必要が出来てきます。

【表5-8】に掲載した「継承グループ」「継承カテゴリー」については、特に重要なものについては、技術コラムの中で解説します。そして、その解説の中で重要な継承関係に関係する原理や定理などを合わせて解説してきます。

これにより「is-a」関係／「a-kind-of」関係のような曖昧な基準ではなく、『科学的継承関係』の考え方を示していきたいと思います。

まとめ＆次回

今回は長方形と正方形を例にとり**「継承パラドックス(inheritance paradox)」**を紹介しました。

「継承パラドックス(inheritance paradox)」が存在する理由として、クラスが「抽象データータイプ(型)(abstract data type)」であり「モジュール」と「タイプ(型)」の両方の性質を持つことを解説しました。

今回は、クラスが「抽象データータイプ(型)(abstract data type)」であり「モジュール」と「タイプ(型)」の２つの性質を持つ事については簡単にふれただけでしたが、第7回のコラムでクラスの「抽象データータイプ(型)(abstract data type)」、「モジュール」および「タイプ(型)」について少し詳しく解説します。

次回は長方形と正方形の例をより詳細に解説していきます。今回は属性と操作についてのみに着目しましたが、次回はたの特性(プロパティ)にも着目して科学的に継承関係を解説していきます。

参考文献

- 文献[5-1] [Clemens Szyperski 2002] Component Software (2nd): Beyond Object-Oriented Programming
- 文献[5-2] [Bertrand Meyer 1997] Object-Oriented Software Construction(邦訳「オブジェクト指向入門 第2版-原則・コンセプト」「オブジェクト指向入門 第2版-方法論・実践」)
- 文献[5-3] [Ian Joyner 1999] Objects Unencapsulated: Java、 Eiffel and C++??(邦訳「オブジェクト指向言語のはなし」)
- 文献[5-4] [Marshall P. Cline,Greg Lomow,Mike Girou1998]C++ FAQs (2nd Edition) (邦訳「C++FAQ C++プログラミングをきわめるためのQ&A集」)

- 文献[5-5] [Kayshav Dattatri 1997] C++: Effective Object-Oriented Software Construction: Concepts, Practices, Industrial Strategies and Practices(邦訳「C++実践オブジェクト指向プログラミング」)
- 文献[5-6] [渡辺 慧1986] 認知科学選書8 知るということ 認知学序説
- 文献[5-7] [渡辺 慧1978] 認知とパタン