第 1 回 オブジェクト指向ゼミ (講習)

ゼミ担当者　：　渋谷　翔吾，島田　将成，松本　哲明
指導院生　：　西村　悟，西岡　雅史
開催日　：　2007 年 5 月 22 日

1　はじめに

近年，オブジェクト指向技術が幅広く使われるようになってきた．オブジェクト指向とは，システムを互いに交信を行なうオブジェクトの集合として捉える概念のことであり，アプリケーションの適用される実世界における対象を抽象化したものである．また，オブジェクト指向プログラミングでは，もの，対象，オブジェクトが主役であり，命令（処理）はオブジェクトに付属している．そして，オブジェクトにメッセージを送り，オブジェクトがメッセージに反応してメソッドを実行するという形になる．これにより，オブジェクト指向プログラミングは，クラスというプログラミングユニットとして実現できる．

オブジェクト指向プログラミングに対して，手続き型プログラミングがある．手続き型プログラミングは，記述された命令を逐次的に実行し，処理の結果に応じて変数の内容を変化させていくプログラミング言語のことである．よって，全体の流れを考えながらプログラムを組まなくてはならない．一方，オブジェクト指向プログラミングの利点としては，プログラムを小さい部品に分解することによって，分担して開発ができる点が挙げられる．また，プログラムの修正や追加がオブジェクト単位で可能となるため保守性が向上し，この他にも，後述するクラスの再利用などが挙げられる．

2　オブジェクト指向のコンセプト

2.1　オブジェクト

オブジェクトとは，ソフトウェアが扱おうとしている現実世界に存在する物理的あるいは抽象的な実体である．オブジェクト指向では，関連する属性（データ）の集合と，それに対する振る舞い（メソッド）をまとめて「オブジェクト」とし，それを組み合わせることでソフトウェアを構築する．[1)]

属性（データ）とは，オブジェクトの固有の姿，形，性質などの状態を表現した値である．Fig. 1 にオブジェクトの例を示す．

Fig. 1 の A 君というオブジェクトの例では，「名前」，「性別」がデータである．振る舞い（メソッド）とは，各

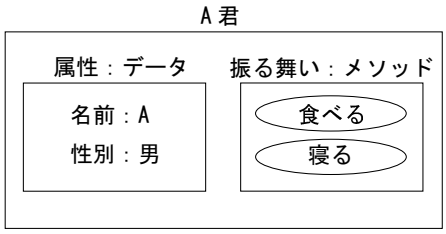

Fig. 1　オブジェクト（出典：自作）

オブジェクトが持つ固有の振る舞いである．Fig. 1 の例では，振る舞いとして，「食べる」，「寝る」がある．

2.2　クラス

クラスとは，オブジェクト作成の基盤となる型枠（テンプレート）であり，同様な状態・振る舞いを持つオブジェクトの共通部分を集め，1 つの型を構成する役割をもつ．つまり，オブジェクトはクラスのインスタンス（型枠に中身を入れ，実体化したもの）である．[2)]

Fig. 2 に示すように，たい焼きはたい焼きの鋳型に中身を詰めて焼き上げる．できたたい焼きはどれも同じ形をしており，材料があれば 1 つの鋳型からいくつでも作成できる．たい焼きの鋳型がクラスであり，中身を詰めて焼く作業がインスタンス生成，焼き上がった個々のたい焼きがインスタンスである．

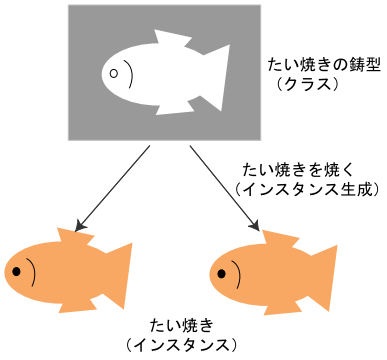

Fig. 2　クラスとインスタンス（出典：自作）

クラスを用いることのメリットは，多くのオブジェクトを 1 つのクラスに対して，簡単な修正で表現すること

が可能になり，労力が削減できることである．Fig. 3 に示すように，α ロボット，β ロボットといったそれぞれのオブジェクトを一から作成するよりも，ロボットクラスを用いてそこから α ロボット，β ロボットに固有の部分の変更を加えるといった方法を用いれば，手間が省ける．つまり，クラスを何度も再利用することができる．

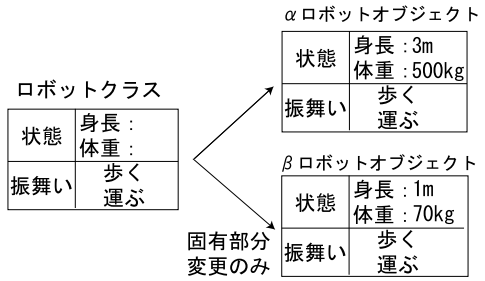

Fig. 3　クラスとオブジェクト（出典：自作）

クラスのメンバには，下記のようにフィールド（変数），コンストラクタ，メソッドの 3 種類がある．

```
public class クラス名 {
メンバ変数
コンストラクタ
メソッド
}
```

変数はクラスの状態を表す．メソッドは下記のように記述され，クラスによって定義された動作（振る舞い）を構成する．これにより，処理の対象だけが異なる場合に似たようなコードを書かなくて済むようになる．

```
戻り型　メソッド名（引数型　引数名）{
    メソッド本体
}
```

コンストラクタはクラスの新しいインスタンスの状態を初期化する特殊なメソッドである．なお，コンストラクタ名はクラス名と同じ名称にする．

```
戻り型　コンストラクタ名（引数型　引数名）{
    コンストラクタ本体
}
```

2.3　インターフェース

次に，インターフェースについて説明する．インターフェースとは，オブジェクト間のメッセージ通信の基礎になるものである．オブジェクトが提供するメソッドは，提供されたインターフェースを介して送信されるメッセージによって呼び出される．通常，インターフェースはメソッドだけを持ち，変数を持たない．ユーザーが変数へのアクセスを必要とする場合は，その変数を返すメソッドを作成して呼び出し，変数へアクセスする．

インターフェースは，オブジェクト指向におけるカプセル化の基本であり，堅牢なクラス作成のために不可欠である．カプセル化，インターフェースのメリットに関しては，後節で述べる．また，Fig. 4 に示すように，ユーザーは実装を直接参照せず，インターフェースを介して対話を行い，実装の具体的な内容は隠蔽されているので知ることはない．ユーザーにとっては，実装の内容は重要ではなく，正しい入出力が行われれば途中経過は気にしないという考えである．[3)]

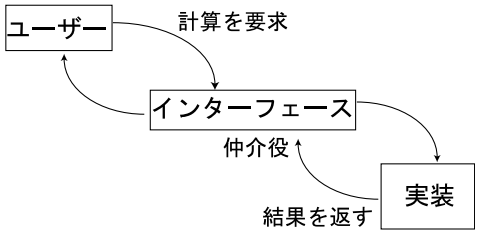

Fig. 4　インターフェースと実装（出典：自作）

3　継承

本章では，継承について説明する．継承とは，既に定義した別のクラスから，変数やメソッドを引き継ぐことである．Fig. 5 を用いて説明する．

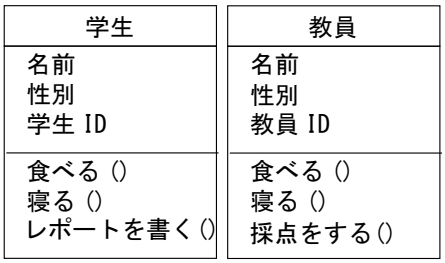

Fig. 5　クラス（出典：自作）

上記のクラスには，共通した属性やメソッドがある．このような場合，オブジェクト指向では，この 2 つのクラスに共通した抽象クラスを作成することによって，以下の Fig. 6 のようにまとめることができる．これを汎化，または抽象化という．汎化することによって，プログラムが冗長になることを防ぐことができる．

抽象化して作成した人間クラスをスーパークラスといい，学生クラス，および教員クラスは人間クラスのサブクラスという．サブクラスはスーパークラスの属性やメソッドを受け継ぐ．したがって，Fig. 6 の学生クラスと教員クラスには人間クラスで定義した属性やメソッドは定義しないで用いることができる．スーパークラスとサ

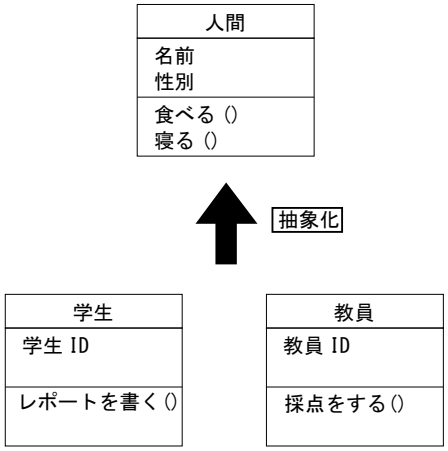

Fig. 6　抽象化（出典：自作）

ブクラスの関係は Fig. 7 のように表す．人間クラスと学生クラスの関係，人間クラスと教員クラスの関係をそれぞれ Is-a 関係という．

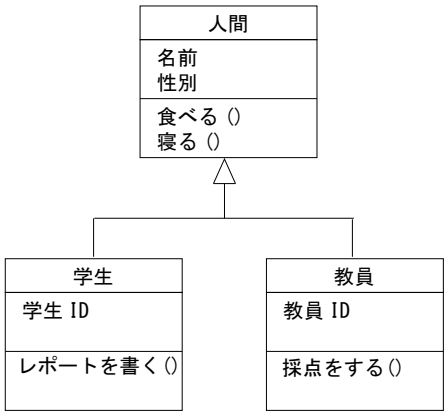

Fig. 7　継承（出典：自作）

プログラム中では，継承は以下のように用いられる．

```
class クラス名 extends スーパークラス名 {
        …
}
```

サブクラスにはスーパークラスから継承した属性やメソッド以外に，独自の変数やメソッドを定義することができる．これを特化という．

人間クラスを継承することによって，学生クラス，教員クラスの他にも，会社員クラスなどを定義することができる．例えば，人間クラスを継承した会社員クラスを新たに作成する場合は以下のように記述する．

```
class 会社員クラス extends 人間クラス {
        会社名 … 新たな属性

        働く () … 新たな振る舞い
}
```

上述したように，この会社員クラスの内容は，人間クラスにはない内容を定義するだけでよい．人間クラスで定義された「名前」,「性別」の属性，また「食べる」,「寝る」の振る舞いは継承されているからである．つまり，サブクラスには,「スーパークラスとのクラスの差分」だけを定義すればよい．上記の会社員クラスでは,属性として「会社名」，振る舞いとして「働く」を新たに定義している．

クラスの継承関係は，クラス間の階層を定義するものであるが，継承関係を考える時に,「継承」と「包含」関係とを混同しないように注意する必要がある.「会社員が人間を継承する」ということは,「会社員 is a 人間」ということがいえなければならない．同じ階層関係でも,「県」と「市」というのは，包含関係にあるが，継承関係にはなりえない.「県は市ではない」ということからも明らかである．[4)]

この継承の利点は，クラスの再利用性が高まることにより，コーディングの負担を軽減させることができることである．また，プログラム全体をオブジェクトの持つ共通性に着目して体系的に整理することができるため，クラス間の属性を明確に記述でき，プログラムの可読性を向上させることができる．

4　カプセル化

Fig. 8 に示すように，カプセル化とは，オブジェクト内部のデータや，オブジェクトの振る舞いを隠す仕組みを指す．メソッドや変数にアクセスできる対象を特定の範囲内に限定し，クラス間の依存関係が極小化されるようにする．これにより，部品の独立性を高め，外部からのアクセス制御を行い，機密性を高めることが可能になる．

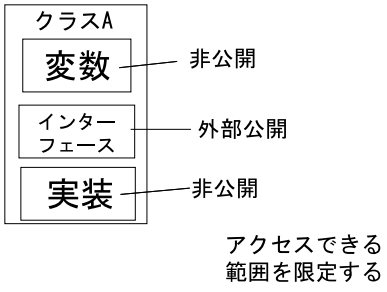

Fig. 8　カプセル化の概念（出典：自作）

メリットとしては，デバッグと保守，変更が容易になることに加え，処理の仕組みが分からなくても，入出力の関係が分かっていれば，そのクラスやメソッドを利用できることが挙げられる．

Fig. 9 に示すように，顧客情報を扱うシステムを構築する場合，従来のプログラミングでは”どのデータベースにデータを格納しているか”などを理解してプログラミングする必要があった．よって，顧客情報の格納先が変更になった場合，全体的にプログラムを変更する必要があった．しかし，オブジェクト指向のカプセル化を用いたプログラミングでは，格納作業とその他の作業が独立し，アクセス範囲が限定されているため，格納先が変わっても全体のプログラムを変更する必要がない．格納作業に関わる部分だけを変更すれば良いので，手間が省ける．

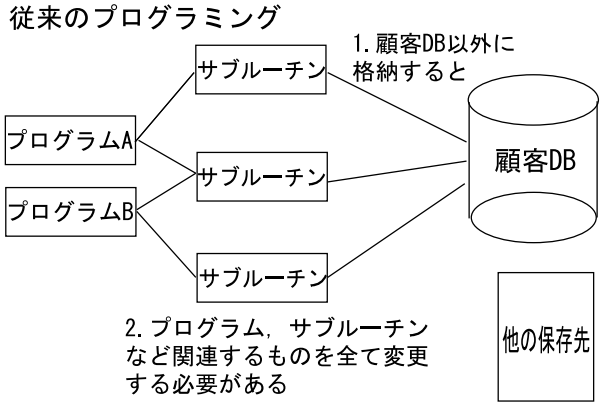

Fig. 9　カプセル化のメリット（出典：参考文献 3) より参照）

5　ポリモーフィズム

ポリモーフィズムは一般的に多様性と訳される．本章では，ポリモーフィズムとは何かを説明し，利用するメリットを述べる．

Fig. 10 に示すように,「商品」クラスをスーパークラスとする 3 つのクラス「パン」「机」「本」があるとする．

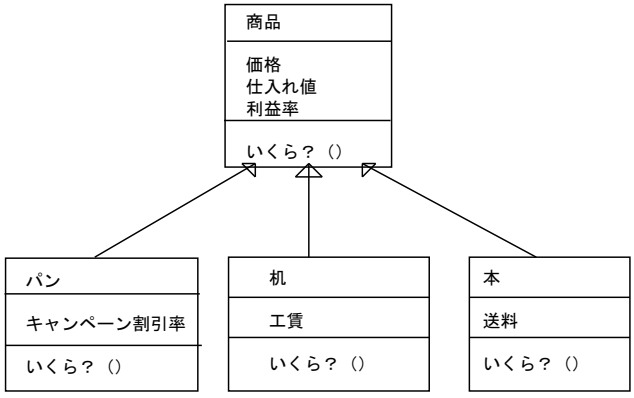

Fig. 10　ポリモーフィズム（出典：自作）

各サブクラスは,「商品」クラスで定義された「いくら？()」メソッドを継承している．ユーザーが任意のサブクラスの価格を問い合わせるとき，共通の「いくら？()」メソッドで価格を取得することができる．このようにクラスを設計しておくと，たとえ新たな商品が追加されても，ユーザは以前と同じメソッドで新たな商品の価格を問い合わせることができる．

変化に強いというオブジェクト指向の強みは，この機能により実現されている．このようなクラス同士の関係をポリモーフィズムという．2)

- パン…仕入れ値×（1＋利益率）×キャンペーン割引
- 机　…仕入れ値×（1＋利益率）＋工賃
- 本　…仕入れ値×（1＋利益率）＋送料

また，上記のようにサブクラスごとに価格の算出方法が異なっていても，ユーザの問い合わせ方法は影響を受けず,「いくら？()」メソッドで問い合わせ可能である．同様に，各サブクラスの算出方法が変更されても問い合わせ方法に影響することがない．これは，先に説明したカプセル化のメリットである．

ポリモーフィズムは，先に述べた継承とカプセル化を同時に利用したものと言える．継承により拡張性を高め，カプセル化によって保守性を高めることで，ポリモーフィズムは変化に強い設計を実現する．

6　コンポジット

前章までで説明した「オブジェクト」,「クラス」,「継承」,「カプセル化」,「ポリモーフィズム」といったオブジェクト指向の主要な概念の他に,「コンポジット」という概念がある．コンポジットは，関連と集約というクラスの関係性の総称であり，正確にはコンポジットパターンと呼ばれるデザインパターンの一つであるが，次節以降では関連と集約の概念のみを述べる．

6.1　関連

Fig. 11 に示すように，パソコンにプリンタ，スキャナ，デジタルカメラが接続されている．それぞれはオブジェクト指向のクラスを比喩的に表す．ユーザーは，パソコンを利用することで，プリンタ，スキャナ，デジタルカメラなどを組み合わせて利用することができる．このようなクラスの関係を関連と呼び．各種サービスを連携させて，一つのサービスを実現する際に使用する．[3)]

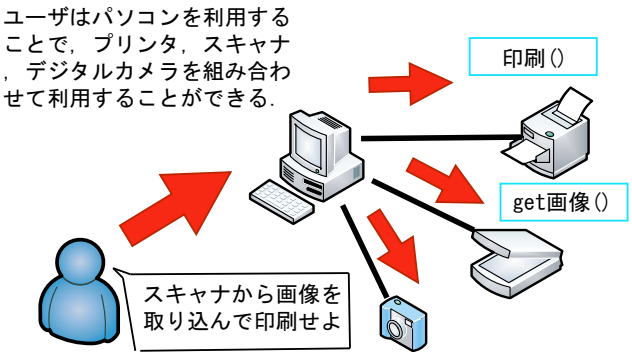

Fig. 11　関連（出典：自作）

6.2　集約

Fig. 12 に示すように，パソコンはハードディスク，メモリ，CPU，マザーボード，グラフィックボードなどの部品から構成されている．各部品はオブジェクト指向のクラスを比喩的に表す．パソコンは新たな部品を追加することで機能を追加することができる．また，各部品が故障しても，その部品を直すことで元の機能を実現することができる．このような関係を集約と呼ぶ．大規模なシステムを設計する際に，機能ごとにクラスを分割することでソースコードの保守が容易になり，また新たなクラスを作成することでシステムの拡張を容易に行うことができる．また，クラスを分割することで，システムが正しく動作しなかった際に問題の特定が容易である．

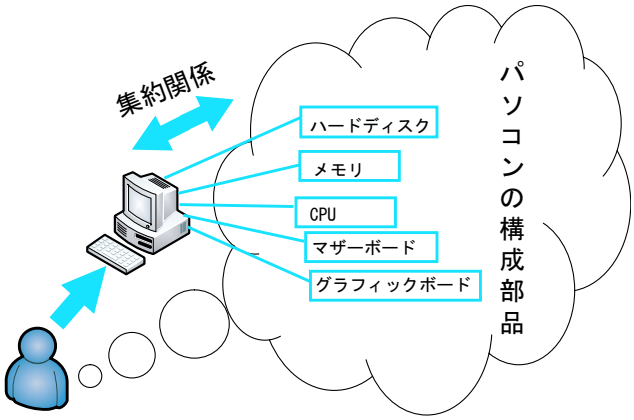

Fig. 12　集約（出典：自作）

参考文献

1)　ジョゼフ・オニール，独習 Java，翔泳社，2005 年

2)　Matt Weisteld，Java と UML で学ぶオブジェクト指向の考え方，翔泳社，2002 年

3)　牛尾剛，オブジェクト脳のつくり方，翔泳社，2003 年

4)　芳賀博英，Java によるプログラミング，森北出版，2003 年