オブジェクト指向プログラミング (OOP)

第 1.0 版

作　成　者	中村広二
作　成　日	2019 年 10 月 3 日
最終更新日	2019 年 10 月 3 日

オブジェクト指向とは？

オブジェクトと呼ばれる機能の部品でソフトウェアを構成させるもの（集まりをみつけモノとして扱う）であり、一般的に

・カプセル化 【accessor(アクセサ) ／ class（クラス） 】
→プロパティ（属性）とメソッド（手続き）を一つのオブジェクトにまとめ、
その内容を隠蔽すること。

・インヘリタンス 【extends（エクステンズ)】
→オブジェクトが他のオブジェクトの特性を引き継ぐこと。「継承」

・ポリモフィズム
【抽象クラス abstract（アブストラクト）
／ interface（インターフェース）と implements（インプリメンツ)】
→「抽象クラス」や「インターフェース」などを利用してメソッドの呼び出し方法を共通化し、
さらに「オーバーライド」させることで同じメソッドを呼び出しても、
実際のインスタンス毎にその挙動を変化させようとすること。「多様性」

などの機能や特徴を活用したプログラミング技法のことをいい、それらを文法的に提供している開発言語のことをオブジェクト指向言語という。

〈主なオブジェクト指向言語の変遷〉

1962 年 Simula（シミュラ）
・・・オブジェクトとクラスを導入した最初の言語。
1972 年 Smalltalk（スモールトーク）
・・・オブジェクト間の相互作用にメッセージという機構を導入した最初の言語。
1979 年 C++
・・・C 言語のオブジェクト指向拡張言語。
1983 年 Objective-C（オブジェクティブ シー）
・・・C 言語のコードと Smalltalk 型のオブジェクトシステムを混在させた C 言語の拡張言語。
iOS アプリの開発で利用する。
1990 年 Python（パイソン）
・・・最初のオブジェクト指向のスクリプト言語。
1995 年 Java（ジャバ）
・・・仮想マシン（Java 仮想マシン）上で動作することによる高い可搬性、
スレッドの標準サポートと充実したライブラリ群で知られているオブジェクト指向言語。
Android アプリの開発で利用する。
2000 年 C# （シーシャープ）
・・・Java とよく似た作りで Java と同じく仮想マシン(.NET Framework)上で動作する言語。
2002 年 Visual Basic.NET（ビジュアルベーシックドットネット）
・・・既存の Visual Basic(6.0 以前)を大きく改訂してオブジェクト指向に対応した言語。
C#同様、.NET Framework 上で動く。

Java は必ずしもオブジェクト指向ではない。

Java は必ずしもオブジェクト指向ではない。コードの書き方によってオブジェクト指向となる。
オブジェクト指向を実現するソースを書くことができるように Java は最初から構成されている。
Java を十分に活用できるようにソースを書こう。
以下のプログラムソースコードを例に挙げて説明する。

A. オブジェクト指向でプログラミングされているソースの例(OOP の例)

B. オブジェクト指向でプログラミングされていないソースの例(非 OOP の例)

上記 A と B の 2 つのサンプルプログラムは同じ機能を持っている。
機能は、A と B とも

① 社員一人の「社員 ID」「社員名」「性別」「今日は出勤している？していない？」という情報を画面に表示する。
② 前提として社員は”ABCDEF”という社員 ID を持っている。

A. オブジェクト指向でプログラミングされているソース

```
// オブジェクト指向で作られた部品を使うクラス
Class TestA {

   public static void main() {

            //社員 ID を定義
            String shainId = “ABCDEF”;

            //社員クラスから社員オブジェクトを生成（インスタンスの生成）
            ClassShainA shain = new ClassShainA(shainId);

            // 画面上に社員 ID を表示
            System.out.println(shain.shainId);

            // 画面上に社員名を表示
            System.out.println(shain.shainName);

            // 画面上に社員性別を表示
            System.out.println(shain.shainSex);

            // 画面上に「今日は出勤しているのか？」を表示
            System.out.println(shain.isShukkin());
   }
}
```

```
// 「オブジェクト指向で作られている」部品クラス
Class ClassShainA {

   //社員 ID
   public String shainId = null;

   //社員名
   public String shainName = null;

   //社員性別
   public String shainSex = null;

   // コンストラクタ
   public ClassShainA(String argShainId) {
            shainID = argShainId;
            shainName = DB から社員名を所得する処理（引数は shainId）;
            shainSex = DB から社員性別を所得する処理（引数は shainId）;
   }

   //メソッド「明日は出勤しているのか？」
   //戻り値  true：出勤している  ／  false：出勤していない
   public boolean isShukkin() {

            // 本日の日付を取得
            Date dateToday = new Date();

            // 「今日は出勤しているのか？」を取得
            boolean flgShukkin = DB から出勤状況を取得する処理（引数は shainId と dateToday）;

            // 結果を戻り値として返却
            rerurn flgShukkin;
   }
}
```

オブジェクト指向プログラミングで記述されているソース。クラス「TestA」を手動で起動すると、クラス「TestA」からクラス「ClassShainA」の処理が呼び出している。TestA の 10 行目でオブジェクト指向プログラミングらしく「new」という言葉を使って「社員オブジェクトの作成」を行っている。続いて社員オブジェクトが持つ情報へアクセスを行って画面にデータを表示している。

B. オブジェクト指向でプログラミングされていないソース

```
// オブジェクト指向で作られていない部品を使うクラス
Class TestB {

   public static void main() {

            //社員 ID を定義
            String shainId = “ABCDEF”;


            // 画面上に社員 ID を表示
            System.out.println(shainId);

            // 画面上に社員名を表示
            System.out.println(ClassShainB.getShainName(shainId));

            // 画面上に社員性別を表示
            System.out.println(ClassShainB.getShainSex(shainId));

            // 画面上に「今日は出勤しているのか？」を表示
            System.out.println(ClassShainB.isShukkin(shainId));
   }
}
```

```
// 「オブジェクト指向で作られていない」部品クラス
Class ClassShainB {


   //社員名
   public String getShainName(String argShainId) {
            return DB から社員名を取得する処理(引数は argShainID)
   }

   //社員性別
   public String getShainSex(String argShainId) {
            return DB から社員性別を取得する処理(引数は argShainID)
   }


   //メソッド「明日は出勤しているのか？」
   //戻り値  true : 出勤している  ／  false : 出勤していない
   public boolean isShukkin(引数は argShainID) {

            // 本日の日付を取得
            Date dateToday = new Date();

            // 「今日は出勤しているのか？」を取得
            boolean flgShukkin = DB から出勤状況を取得する処理（引数は shainId と dateToday）;

            // 結果を戻り値として返却
            rerurn flgShukkin;
   }
}
```

オブジェクト指向プログラミングで記述されていないソース。クラス「TestB」を手動で起動すると、クラス「TestB」からクラス「ClassShainB」の処理が呼び出される。このソースでは上記 TestA のような「new」という言葉を使っていません。続いて社員オブジェクトが持つ情報へアクセスを行って画面にデータを表示しています。

カプセル化

カプセル化する目的は、「自分の機能を使ってくれる他のプログラム」に対しての「配慮」が「安全なシステム」を作ることにある。

1．OOP におけるカプセル化は、分割した処理を「その処理を利用してくれる他のプログラム」に対して「公開する」OR「隠ぺいする」という手法を自身のプログラムに対して盛り込む事である

2．これは他者に対して「余計な処理を見せて惑わせない」「処理の意図しない使われ方をさせない」という配慮である

3．この「配慮」がオブジェクト(≒プログラム)同士の信頼関係を築きあげ、安全なシステムを作る土台となる

２つのソースを比較する。
① 適切なカプセル化が行われているソース
② 適切なカプセル化が行われていないソース

二つは同じ処理とする。処理の内容としては、

・二つの商品クラスというものがある(商品 AAA と商品 BBB)
・これらの商品クラスはコンストラクタとして商品コードを受け取って、
DB に格納されている商品情報を自分自身のオブジェクト内部に保持する
・自身が持っている各商品情報を他者からの求めに応じて出力する

という処理をおこなうプログラム。

① 適切なカプセル化が行われているソース

```
// 適切なカプセル化が行われているプログラム
public class 商品AAA {

   private String 商品コード;
   private String 商品名称;
   private double 消費税率;
   private double 商品金額税抜;
   private double 商品金額税込;

   // コンストラクタ
   public 商品AAA(String arg商品コード) {
            商品コード = arg商品コード;
            商品名称 = DBからの商品名称取得処理(arg商品コード);
            消費税率 = DBからの消費税率取得処理();
            商品金額税抜 = DBからの商品金額取得処理(arg商品コード);
            商品金額税込 =商品金額税抜＋（商品金額税抜*消費税率）;
   }

   public String get商品コード() {
            return 商品コード;
   }

   public String get商品名称() {
            return 商品名称;
   }

   public String get商品金額税抜() {
            return 商品金額税抜;
   }

   public String get商品金額税込() {
            return 商品金額税込;
   }
}
```

②　適切なカプセル化が行われていないソース

```
// 適切なカプセル化が行われていないプログラム
public class 商品BBB {

   public String 商品コード;
   public String 商品名称;
   public double 消費税率;
   public double 商品金額税抜;
   public double 商品金額税込;

   // コンストラクタ
   public 商品AAA(String arg商品コード) {
            商品コード = arg商品コード;
            商品名称 = DBからの商品名称取得処理(arg商品コード);
            消費税率 = DBからの消費税率取得処理();
            商品金額税抜 = DBからの商品金額取得処理(arg商品コード);
            商品金額税込 =商品金額税抜＋（商品金額税抜*消費税率）;
   }
}
```

継承(インヘリタンス)

「親クラス super（継承される側)」には「共通部分」を、「子クラス this（継承する側)」には「差分」を書く。

継承機能を利用することによってコードが再利用される例として、例３つのクラスを用意した。
例題によってコードを再利用する方法について理解する。

[例　業務仕様]

営業社員一人の

社員 ID
社員名
性別
今日は出勤している？していない？
その営業社員が持っている顧客の数

という情報を画面に表示する。

[クラスの説明]

・クラス「Test」: OOP で作られたクラスを呼び出す(利用する)クラス。

・クラス「ClassEigyoShain」:「営業社員」クラス。
OOP で作られたクラス。下記「ClassShain」を継承して作られたクラス。
今回の継承機能を説明する上では「子クラス＝サブクラス」となるクラス。

・クラス「ClassShain」:「社員」クラス。
OOP で作られたクラス。上記「ClassEigyoShain」の継承元となるクラス。
今回の継承機能を説明する上では「親クラス＝スーパークラス」となるクラス。

[継承の考え方]

今回の継承機能の説明においてもっとも注目していただきたいのは、
上記「営業社員クラス：ClassEigyoShain」。このクラスは、
上記「社員クラス」を継承して(元にして)作成いる。
これは「現実的な事象として、営業社員という職業は、社員という職業を内包している」という考え方に基づいた設計になっている。

上記のプログラムが継承機能を用いる目的は、「営業社員クラス：ClassEigyoShain]のプログラムコード量を削減する」事にある。
「営業社員という職業が社員という職業を内包するのであれば、内包される部分、すなわち上記「社員クラス」のコードは再利用したい」という考え方がある。

継承の例

```
// オブジェクト指向で作られた部品を使うクラス
Class TestA {

   public static void main() {

            //社員 ID を定義
            String shainId = “ABCDEF”;

            //営業社員クラスから営業社員オブジェクトを生成（インスタンスの生成）
            ClassEigyoShain eigyoShain= new ClassEigyoShain(shainId);

            // 画面上に社員 ID を表示
            System.out.println(eigyoShain.shainId);

            // 画面上に社員名を表示
            System.out.println(eigyoShain.shainName);

            // 画面上に社員性別を表示
            System.out.println(eigyoShain.shainSex);

            // 画面上に「今日は出勤しているのか？」を表示
            System.out.println(eigyoShain.isShukkin());

            // 画面上に「その営業社員がもっている顧客の数」を表示
            System.out.println(eigyoShain.getKokyakuCnt());

   }
}
```

```
// 「オブジェクト指向で作られている」部品クラス(継承における子クラス＝サブクラス)
Class ClassEigyoShain extends ClassShain {


   // コンストラクタ
   public ClassEigyoShain (String argShainId) {
            super(argShainId);
   }

   //メソッド「その営業社員がもっている顧客の数」
   //戻り値　その営業社員が持っている顧客の数
   public int getKokyakuCnt() {

            // 「その営業社員がもっている顧客の数」を取得
            rerurn DBから顧客の数を取得する処理(引数はsuper.shainId);
   }
}
```

```
// 「オブジェクト指向で作られている」部品クラス(継承における親クラス＝スーパークラス)
Class ClassShain {

   //社員ID
   public String shainId = null;

   //社員名
   public String shainName = null;

   //社員性別
   public String shainSex = null;

   // コンストラクタ
   public ClassShain(String argShainId) {
            shainID = argShainId;
            shainName = DBから社員名を所得する処理（引数はshainId）;
            shainSex = DBから社員性別を所得する処理（引数はshainId）;
   }

   //メソッド「明日は出勤しているのか？」
   //戻り値　true：出勤している　／　false：出勤していない
   public boolean isShukkin() {

            // 本日の日付を取得
            Date dateToday = new Date();

            // 「今日は出勤しているのか？」を取得
            boolean flgShukkin = DBから出勤状況を取得する処理（引数はshainIdとdateToday）;

            // 結果を戻り値として返却
            rerurn flgShukkin;
   }
}
```

多様性(ポリモフィズム)

interface（インターフェース）と implements（インプリメンツ）

〈注意〉
・インターフェースを理解する上で、とりあえず「継承」の事は忘れる。
・「インターフェースと継承との違い」を考えない。インターフェースの理解の邪魔になります。

interface（インターフェース）と implements（インプリメンツ）について説明する。
Interface は、メソッドを実装するクラスを別に持つ、表紙のような役割をする。
機能は決まっているが、処理が決まっていない時に使用する。

```
// インターフェース
interface SampleInterfaceAAA {
   void doSomething()
}
```

```
// インターフェースを実装したクラス
public class SampleClassAAA implements SampleInterfaceAAA  {

   public void doSomething() {
            System.out.println(“こんにちは”);
   }
}
```

```
// インターフェースを実装クラスを使用する実行クラス
public class SampleRun  {

   public static void main(String[] args) {
            SampleInterfaceAAA objAAA = new SampleClassAAA();
            objAAA. doSomething();
   }
}
```

〈END〉