Javaから見たオブジェクト指向入門

オブジェクト指向AtoZセミナー

(株)豆蔵　井上 樹

趣旨

- Javaを通してオブジェクト指向に対する理解をより深め、次のステップへの足がかりとなるようにする

対象

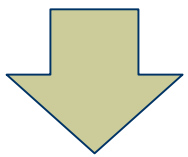
- オブジェクト指向をこれから始めたい方
- Javaをもっと使いこなしたい方
- 必要な知識
  - Javaの言語仕様
  - UML

Agenda

- この先何が必要か
- オブジェクト指向最初の一歩
- Javaに隠されたフレームワーク
- まとめ

この先何が必要か

- Javaを使いこなしたい
- いいソフトウエアを作りたい



オブジェクト指向の知識が必須

- オブジェクト指向はいいソフトウエアを作るための技術のひとつ
- Javaにはオブジェクト指向の技術が詰まっている

どんなオブジェクト指向の知識が必要か

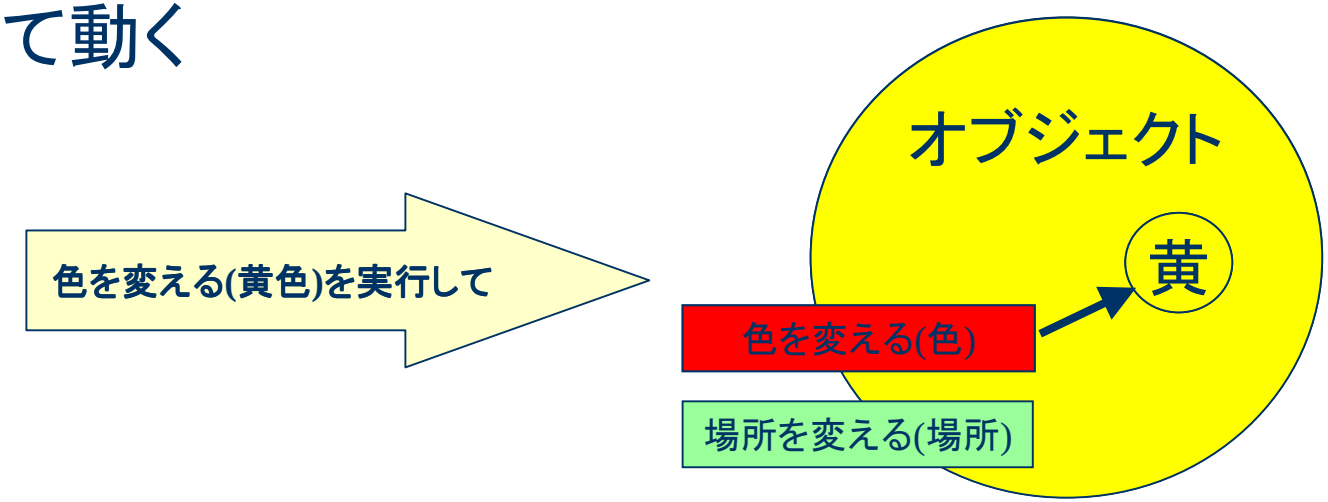
- オブジェクト指向
- フレームワーク
- パターン
- コンポーネント
- 分散オブジェクト
- etc....

オブジェクト指向の基本概念

- オブジェクト
- クラス
- クラスとインスタンス
- オブジェクト指向で作られたソフトウエア
- フレームワーク

オブジェクト

- 自分専用の属性と操作を持つ
- 他のオブジェクトや外界からのメッセージに反応して動く



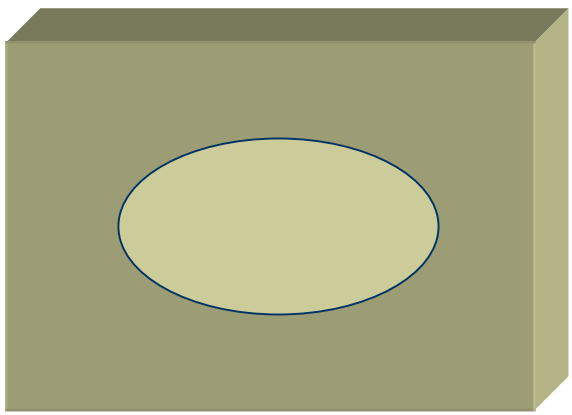
クラス

- オブジェクトの設計図
  - 属性とメソッドの定義が書いてあるもの

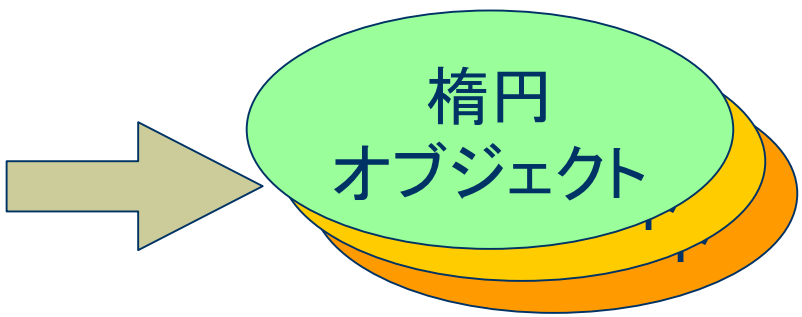
```
クラス 楕円 {
      // 属性
      色        塗りつぶす色;
      幅と高さ  サイズ;
      .....
      // メソッド
      色を変える(色 指定色) {
          塗りつぶす色 ＝ 指定色;
          塗りつぶす();
      }
      .....
}
```

クラスとインスタンス

- インスタンス　＝　クラスから作られたオブジェクト
- クラスに書かれた仕様に基づいてオブジェクトを生成することをインスタンシエーションという

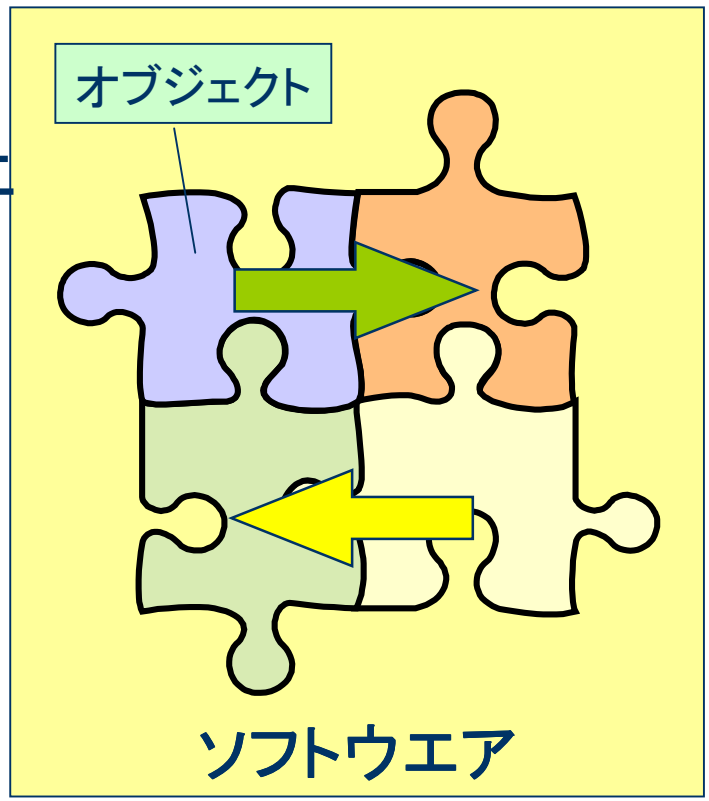


楕円クラス



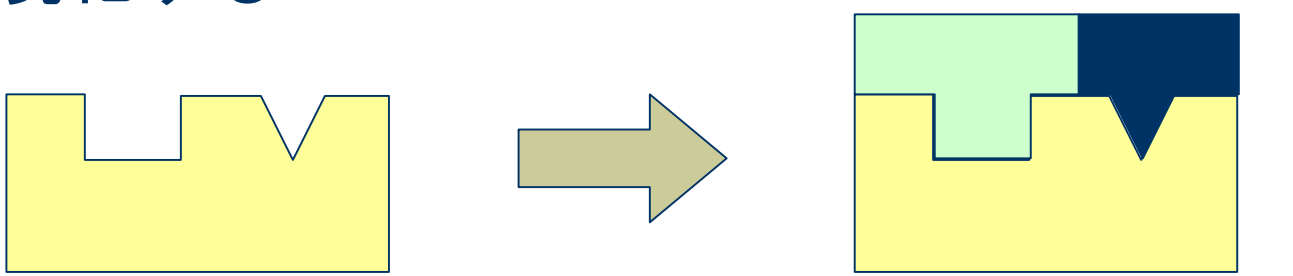
オブジェクト指向で作られたソフトウエア

- オブジェクト指向で作られたソフトウエアは「オブジェクト」という単位で構成されている
- この「オブジェクト」がお互いにメッセージをやり取りすることでソフトウエアが実現される



フレームワーク

- オブジェクト指向でソフトウエアを作る際の様々なノウハウ(部品の組合わせ方や、業務フロー)を再利用可能な形で提供したもの
- 半完成品なので、利用時には利用方法に合わせて実現化する



フレームワーク

フレームワークを
使った完成品

オブジェクト指向最初の一歩

最初の一歩

オブジェクトを作る！

Java = OOP ?

- Javaでは自分でオブジェクト(クラス)を作らなくても、ソフトウエアの作成は可能

```
class AddressBook {
    int maxAddress = 100;
    int currentIndex = 0;
    String name[] = new String[maxAddress];
    String address[] = new String[maxAddress];
    String tel[] = new String[maxAddress];
    ...
    public void add(String val1, String val2,
                                        String val3) {
        name[currentIndex] = val1;
        address[currentIndex] = val2;
        tel[currentIndex] = val3;
        currentIndex++;
    }
    ...
}
```

オブジェクトを作るとは？

- 「オブジェクトを作る」とは、ソフトウエアを構成するクラスを自身で定義し、ソフトウエアを組み上げていくこと

```
import java.util.*;
class AddressBook {
    Vector addresses = new Vector();
    ...
    public void add(Person aPerson) {
        addresses.add(aPerson);
    }
    ...
}

class Person {
    String name;
    String address;
    String tel;
    ...
}
```

オブジェクトを作るメリット

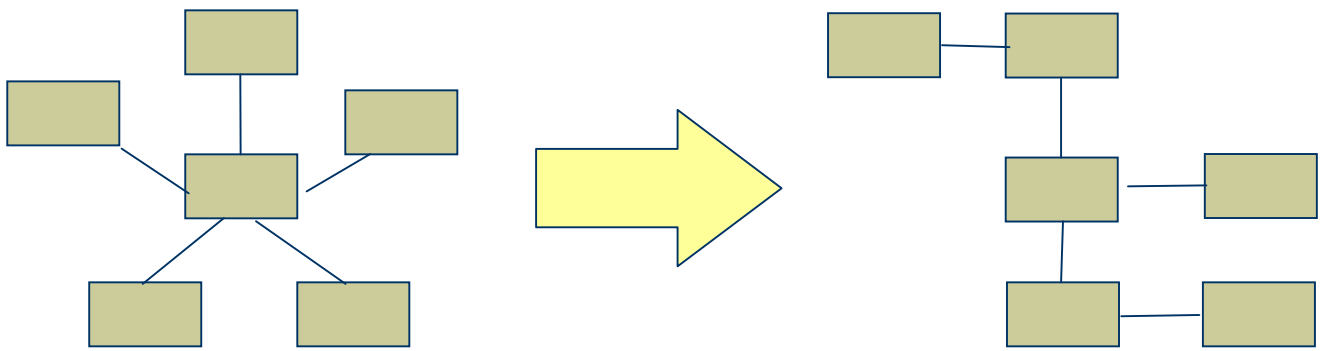
- ソフトウエアの構造が理解しやすくなる
  - 「オブジェクト」という、より抽象度の高い単位でソフトウエアを見ることができる
- オブジェクト指向の技術を利用しやすくなる
  - 分散、永続化、再利用

オブジェクトの作り方

- ソフトウエアをオブジェクトの集合として捉える
  - 機能とデータの集合からオブジェクトの集合へ
- それぞれのオブジェクトに役割を与え、オブジェクト同士の協力関係を考える
- 余談：このために「オブジェクト指向設計」が存在する

オブジェクトを作るときの注意点

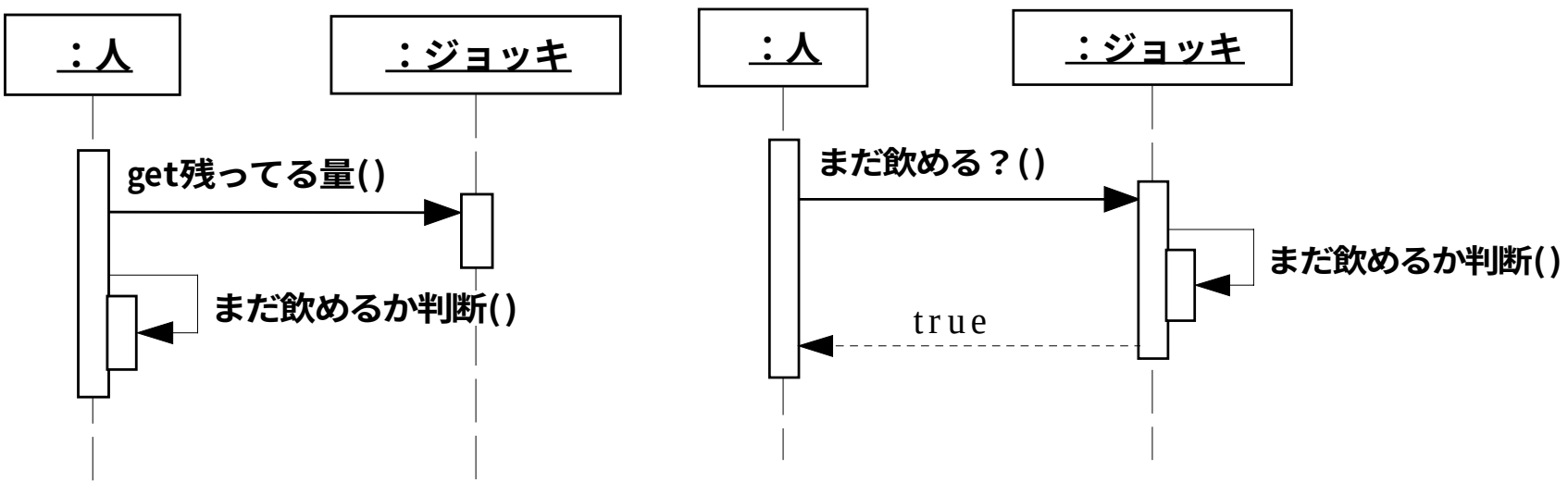
- オブジェクトに沢山の役割を持たせない
  - オブジェクトの役割は1つ！
- オブジェクトが協力し合うことが大事



1つのオブジェクトが頑張る形

オブジェクトが協力しあうような役割分担

役割分担の例



ジョッキ以外にもお皿や鍋があったとすると…

- 左のモデルは、全ての判断を人が行わないといけない
- 右のモデルは、オブジェクトに聞いて周るだけでいい

次の一歩

オブジェクトを作らない！

オブジェクトを作らないとは？

- 「車輪の再発明」の回避
  - プログラマはなんでも自分で作ってしまう(作りたがる)傾向がある
- 再発明したものは
  - 既存のものよりも質が落ちることが多い
  - 既存のものがしてきたことと同じ失敗を繰り返す
  - テストによる信頼性確認が必要

作ってしまいがちなクラス

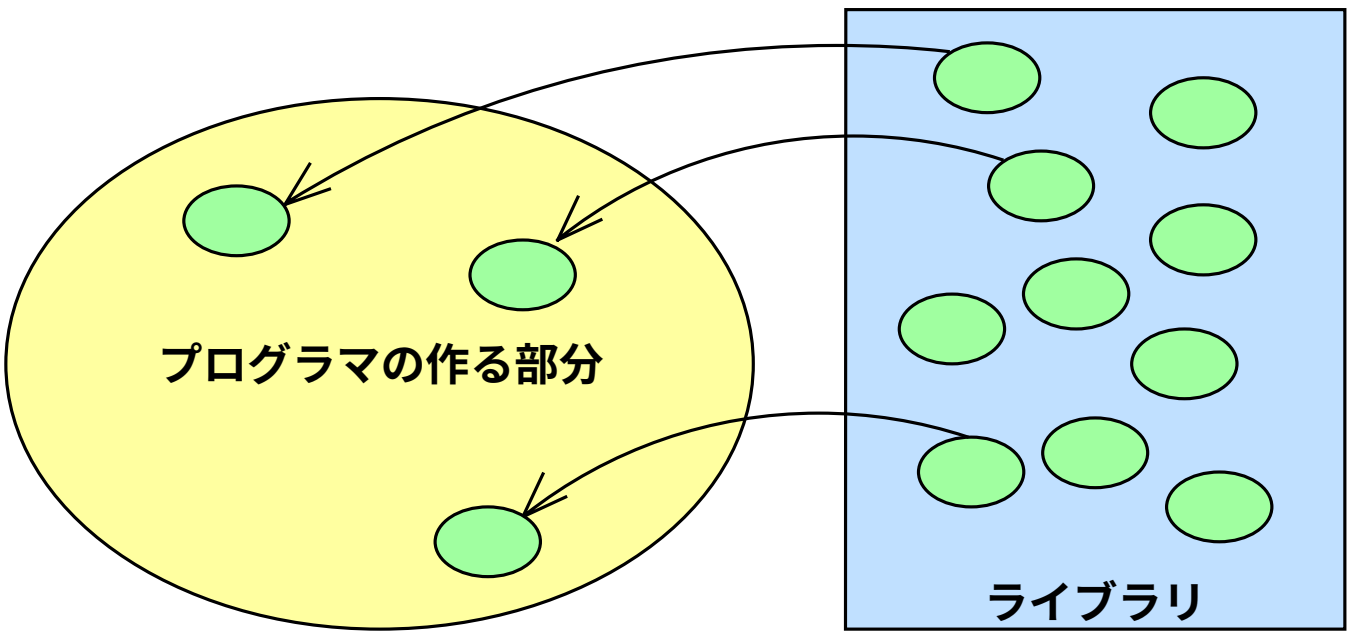
- オブジェクトの集合(コンテナクラス)
  - 配列で実装しがち
- ソート
- 日付、カレンダー
  - 日付とその計算方法に関するクラス
- フォーマッタ
  - 表示形式を整えるためのクラス

再利用という視点

- Javaではクラスライブラリで再利用可能な部品を提供している
  - コンテナクラス　Vector, Hashtable
  - ソート　Arrays.sort()
  - 日付・カレンダー　Date, Calendar
  - フォーマッタ　DateFormat, MessageFormat
- 「既に有るものは作らない」という考え方が大切
- 近年、様々な粒度の再利用部品が提供されることで、プログラミングスタイルが変化しつつある

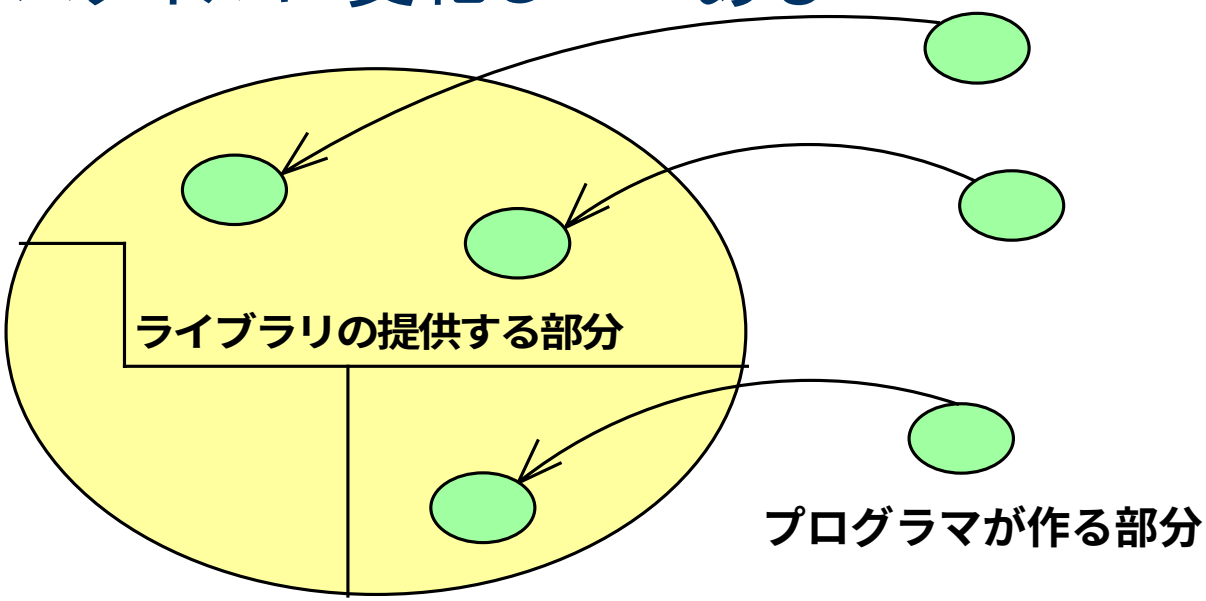
プログラミングスタイルの変化 I

- 従来は大部分のコードをプログラマが作成し、小さな再利用部品を利用していた



プログラミングスタイルの変化 II

- プログラマはライブラリから必要な部品を探し、それらを組合わせ、足りない部分を作るという開発スタイルに変化しつつある



オブジェクト指向最初の一歩：まとめ

- オブジェクトを作ろう！
  - これから作るものをオブジェクトの集合として考える
  - オブジェクトを作ることで、オブジェクト指向のメリットを享受
- オブジェクトは作らないようにしよう！
  - 再利用の視点

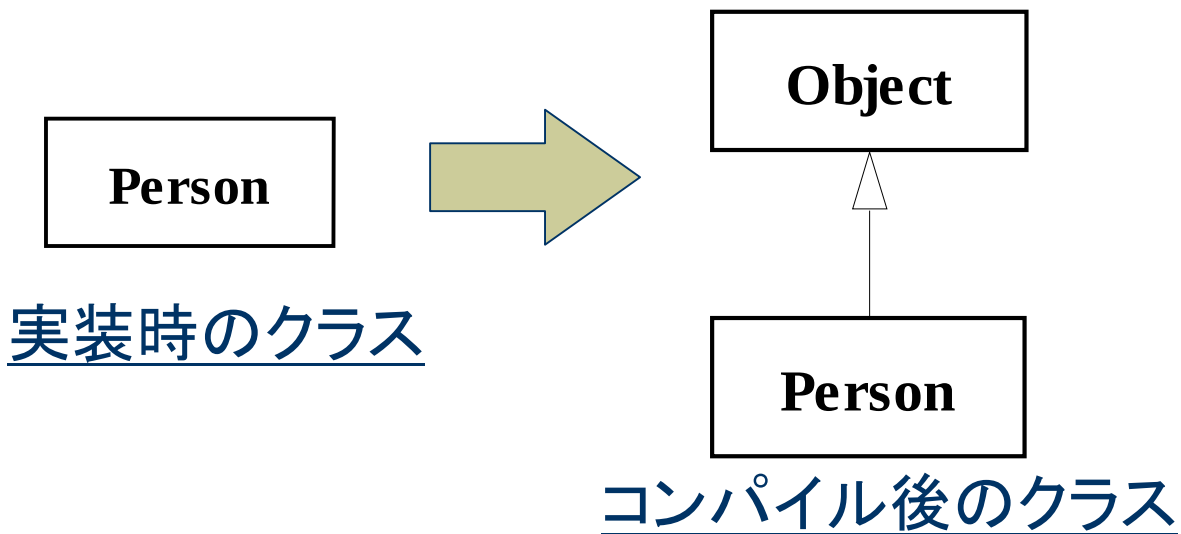
Javaに隠されたフレームワーク

Javaをより活用するためには

- Javaの提供するクラスライブラリとそこで提供される仕組みについて理解する必要がある
- 自身の作ったオブジェクトがクラスライブラリ側からどのように利用されるかを知る

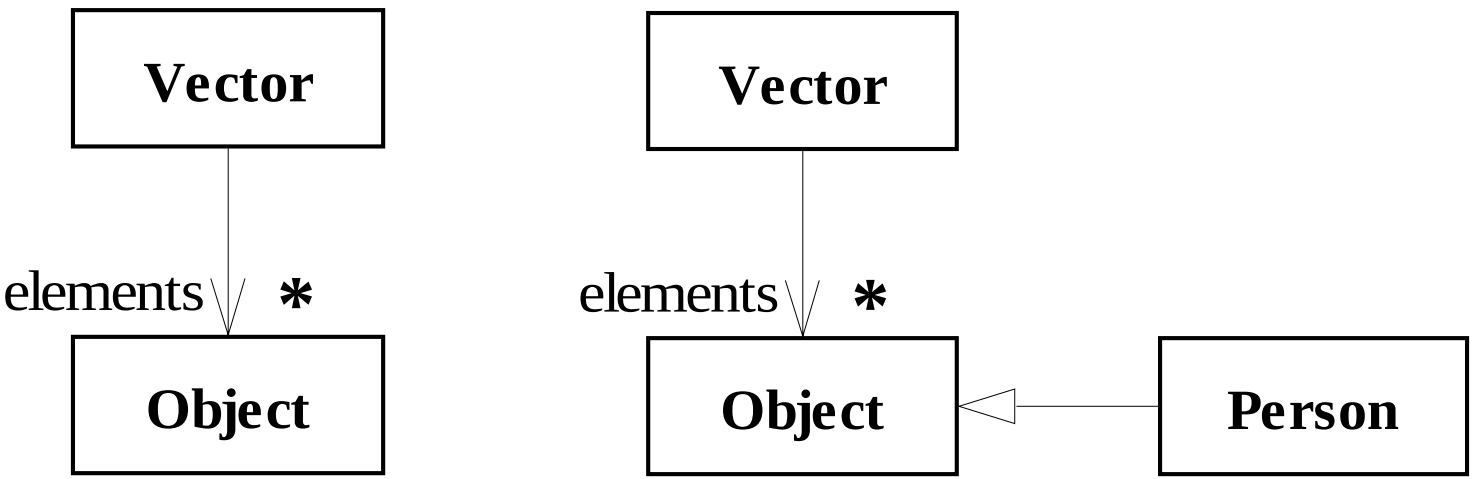
Javaのオブジェクト

- Javaでは、全てのオブジェクトがObjectクラスを継承したクラスとなる
- 自分で作成したオブジェクトも例外ではない



Objectクラスの継承は何を意味するか？

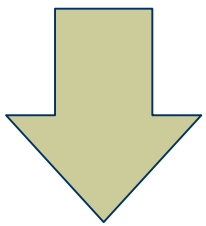
- Javaクラスライブラリでは全てのオブジェクトに適用可能な仕掛け(フレームワーク)が提供されている
- Objectクラスを継承しているということはこの仕組みに取り込まれているということになる



クラスライブラリの構造　　利用時の構造

ライブラリ側から見ると

- どのようなクラスから利用されるか分からない



- 「Objectクラス」という形でクラスライブラリを利用するクラスを抽象化して扱っている

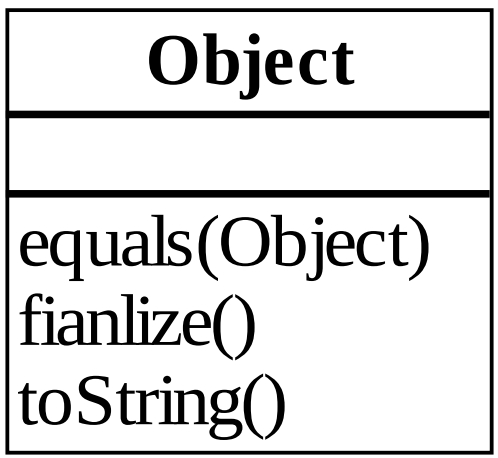
クラスライブラリの修得

- Javaクラスライブラリの持つフレームワークを理解すれば、よりJavaクラスライブラリを有効に活用することができる

- 余談：Javaクラスライブラリを知ること
  → Java開発者の設計意図を知ること

Javaに隠されたフレームワーク

- どんなフレームワークに取り込まれているかはObjectクラス(java.lang.Object)を見るとわかる



Objectクラスに用意されている、Javaクラスライブラリのフレームワークで利用される代表的なメソッド

定義：ホットスポットとフローズンスポット

- フローズンスポット
  - フレームワーク内の変更されない部分
- ホットスポット
  - フレームワーク内の変更可能な部分
  - フレームワークに柔軟性を与える
  - Objectクラスに用意されているtoString()やequals()はホットスポット
  - HotSpotとは別もの

Javaに隠されたフレームワークの紹介

- toString()
  文字列化が必要なフレームワークで使用されるメソッド
- equals()
  オブジェクトの比較が必要なフレームワークで使用されるメソッド

toString()

- オブジェクトを文字列化したいところで使われる
- System.out.println(anObject)等

```
public static String valueOf(Object obj) {
        return (obj == null) ? "null" : obj.toString();
}
```

toString()を設定すると…

- 以下のテストプログラムを考える

```
class Test {
    public static void main(String argv[]) {
        Person aPerson = new Person(“井上 樹”,
                                    “横浜市”, "045-***-****");
        System.out.println(aPerson);
    }
}
```

toString()設定前

- 以下のPersonクラスでテストプログラムを実行する

```
class Person {
    String name;
    String address;
    String tel;

    public Person(String nam, String add,
                                  String phone) {
        name    = nam;
        address = add;
        tel     = phone;
    }
}
```

実行結果

```
> java Test
Person@60deb39c
```

toString()設定後

- 以下のPersonクラスでテストプログラムを実行する

```
class Person {
    // 以下のメソッドを追加
    public String toString() {
        return name + " " + address + "在住 tel:" + tel;
    }
}
```

実行結果

```
> java Test
井上 樹 横浜市在住 tel:045-***-****
```

equals()

- オブジェクトが同じかどうか比較するところで使われる
- VectorのindexOf()等

```
public synchronized int indexOf(Object elem, int index) {
        if (elem == null) {
            ...
        } else {
            for (int i = index ; i < elementCount ; i++)
                    if (elem.equals(elementData[i]))
                        return i;
        }
        return -1;
}
```

オブジェクトが「同じ」とは？

- 質問
  - 私の時計.equals(彼の時計) を実行したら、trueが返ってきました。なにが「同じ」だったのでしょう？

「同じ」とはの答え

- 例
  - 時間、メーカー、色、形、インスタンス
- 「同じ」の概念は場面によって異なる
- それを定義するのがequals()
- デフォルトでは同一インスタンスかどうかを比較

equals()を設定すると…

```
class Test {
    public static void main(String argv[]) {
        Vector addresses = new Vector();
        Person aPerson = new Person("井上 樹", "横浜市", "045-***-****");
        addresses.add(aPerson);

        Person query = new Person("井上 樹", "", "045-***-****");
        int resultIndex = addresses.indexOf(query);

        System.out.println("検索結果:");
        if (resultIndex >= 0) {
            Person resultPerson = (Person)addresses.elementAt(resultIndex);
            System.out.println(resultPerson);
        } else {
            System.out.println("見つかりませんでした");
        }
    }
}
```

equals()設定前

- 先ほどの toString() 設定後のPersonクラスでテストプログラムを実行する

実行結果

```
> java Test
検索結果:
見つかりませんでした
```

equals()設定後

- 以下のPersonクラスでテストプログラムを実行する

```
class Person {
    public String getName() { return name; }
    public String getAddress() { return address; }
    public String getTel() { return tel; }
    public boolean equals(Object obj) {
        Person person = (Person)obj;
        if (name.equals(person.getName()) &&
              tel.equals(person.getTel())) {
            return true;
        } else {
            return false;
        }
    }
}
```

実行結果

```
> java Test
検索結果:
井上 樹 横浜市在住 tel:045-***-****
```

toString()やequals()を定義する意味

- toString()やequals()を再定義することで、Javaの提供するフレームワークに自身の作ったオブジェクトを適用することができる
- 新規コードを書く量を減らすことができる
  - 例：equals()を定義すれば、検索ルーチンを書かなくても良い

Javaに隠されたフレームワーク:まとめ I

- Javaでオブジェクトを作ると、自動的にJavaの提供するフレームワークへの適用が可能になる
- フレームワークから利用されるメソッド(toString()やequals())を再定義することで、フレームワークをより活用できるようになる

Javaに隠されたフレームワーク：まとめ Ⅱ

- Objectクラス以外にもフレームワークを利用するためのクラスは沢山隠されている
- クラスライブラリを知ることが、Java活用の第一歩
- クラスライブラリ修得のコツ
  - 1つのクラスに注目するのではなく、関連しあう複数のクラスで何が実現されるかに注目
  - ホットスポットとフローズンスポットを見極める

まとめ

オブジェクトを作ろう！

- オブジェクトを作ることで初めて、オブジェクト指向を元にした様々な技術を利用することができる
- ただ、既に存在するオブジェクトは再利用すること

フレームワークを利用しよう！

- Javaのクラスライブラリでは様々なフレーワークを提供している
- それらのフレームワークを利用することがJava･オブジェクト指向修得への道

これからのステップ

- 分析からオブジェクト指向を使えば、もっと有効にオブジェクト指向を活用できる
- フレームワークを作ってみよう
  - クラスライブラリに無いものは自分で作り、作ったものを再利用可能なようにまた、クラスライブラリに追加していく
  - 部品が増えてくれば、開発者はより本質的な部分に注力できる
  - 自分のノウハウをフレームワークとして蓄積