

1. ガイダンス & オブジェクト指向 (とJava) 入門

その1



1.1 オブジェクト指向って何？

オブジェクト指向とは …

「継承、隠蔽(カプセル化)、多態性(ポリモーフィズム)という概念を用い、目標となる対象(**Object**)を中心としてシステムを開発しようとする考え方」

文章を読んでも意味を捉えるのは難しい(そもそも、継承、隠蔽、多態性って何だ?)



字面をみても良く分からないので、まずはオブジェクト指向(プログラミング)の例を示すことにする

ちなみに …

「オブジェクト指向の考え方に基づいてプログラムを効率的に作成する機能をサポートしているプログラミング言語」を、オブジェクト指向プログラミング言語(OOPL)と呼ぶ(例: C++、Java、SmallTalk等)



OOPLでなくてもOOPは実現できる(実現するのが“難しいだけ”)

オブジェクト指向プログラミングの例



問題 1: 多変量データ (n 個の変数が存在する総数 N のデータ) から, 様々な確率を計算すること (計算機上で **Java** を使って実現する)

プログラムに必要な条件:

- 多変量データを保持
- 変数の数や, 各変数の取りうる値 (状態数) を保持
- 周辺確率や同時確率, 条件付確率の計算
- 計算不能な状況に陥った際の対応 (存在しない変数や存在しない値が入力された場合の対応, 条件付確率が計算不能な時の対応など)

プログラムが持つべき(持っていると思われ) **情報**

多変量データ(確率変数の実現値) : **rv_**

各確率変数の取りうる状態数 : **numOfStates_**

多変量データに含まれる確率変数の数 : **numOfVariables_**

データ数 : **numOfData_**



これらのデータをすべてひとまとめに
したもの ... **クラス**

```
class Probability{  
    public int[][] rv_  
    public int[]   numOfStates_  
    public int     numOfVariables_  
    public int     numOfData;  
    :  
}
```

ここまでは、C 言語の「構造体」と似ている
が、
これだけでは終わらないのがオブジェクト指向
:
ひとまとめにされるのはデータだけではない！

プログラムが有すべき機能

※ 引数を持つ場合もあるが省略

データを取り込む(初期化)	: Probability()
$P(X_n = x)$ を計算	: getProbability()
$P(X_1 = x_1, X_2 = x_2, \dots)$ を計算	: getJointProbability()
条件付確率を計算	: getConditionalProbability()
確率変数の数を返す	: getNumOfVariables()
⋮	⋮

全ての機能はメソッド(C言語でいう関数)として準備する

+

データとともに、クラスの中にまとめられる(全てのメソッドがクラスに所属する)

初期化専用のメソッド・・・ **クラス名と同じメソッド** : コンストラクタと呼ばれる

あるデータに対応する確率を計算する情報・機能一式(=**オブジェクト**)は、**Probability**クラスから生成される！

(複数データに対応する確率を計算したければ、オブジェクトを複数生成すれば良い)

プログラミング例

```
Class Simulation{
```

```
    public static void main(String[] args){ // C言語のmain()と同じ  
        int[][] data; // Javaにおける2次元配列の宣言法の一つ  
        double prob;
```

```
        Probability P; // Probability クラスのオブジェクト P を宣言
```

オブジェクトの生成(new で始まるメソッドはコンストラクタ。コンストラクタでは、オブジェクトの持つ変数(rv_, numOfStates_ など)の初期化が実行される)

```
        P = new Probability();
```



```
        // 多変量データに含まれる変数の数を表示(Cでいうprintf()と同様)  
        System.out.println( P.getNumOfVariables() );
```

```
        prob = P.getProbability(3,0); // P( $X_3=0$ ) の値をprob へ代入
```

```
        ⋮
```

```
    }
```

```
}
```

先ほどのプログラムで、

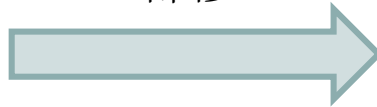
確率変数の数(numOfVariables_の値)を獲得するため、メソッド
(P.getNumOfVariables())を用いた : しかし、実は今のままでは・・・

メソッドを使わずとも、“ **P.numOfVariables_;**”とすることで値を獲得できてしまう

便利な面もある一方、勝手に P.numOfVariables_ の内容が書き換えられてしまうことも・・・

```
class Probability{  
  public int [][] rv_  
  public int numOfStates_  
  :  
}
```

Probability の
class 定義を
一部修正



```
class Probability{  
  private int [][] rv_  
  private int numOfStates_  
  :  
}
```

頭を「public」から「private」に変えることで、Probability クラスの外部からの
変数アクセスを禁止できる ... **カプセル化** (オブジェクト指向の必須要素)

どうしても外部から変更したければ、クラス内に
“_changeNumOfVariables(変更後の値)” 等といった関数を作れば可能
(なければ、外部からの変更は絶対にできない)

オブジェクト指向をもとにしたプログラム拡張

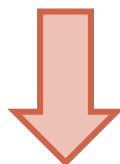


問題 2: 当該データに対応するモデルを作ったり、モデルの情報量規準を計算できるようにする

つまり・・・

Probability クラスで作成した確率計算のメソッド等を活用して、さらなる機能（ベイジアンネットの構造学習や確率推論、構造に対応するMDLやAICなどを計算できるようにする）

これまで作成したプログラム（というかクラス）をベースにして、新たな機能を追加したい（ただし、ベースとなるプログラムそのものには極力手を加えない）



クラスの継承

```
class Probability{
```

```
    protected int [][] rv_;
```

```
    protected int  numOfStates_;
```

```
        ⋮
```

クラス内の
変数宣言部

```
    public probability(){
```

```
        コンストラクタの処理内容
```

```
    }
```

```
        ⋮
```

コンストラクタ
の宣言部

```
    public int  _getNumOfVariables(){
```

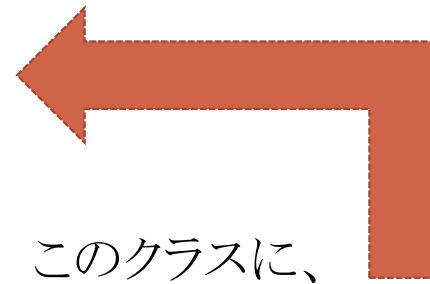
```
        メソッドの処理内容
```

```
    }
```

```
        ⋮
```

```
    }
```

メソッドの
宣言部



このクラスに、

- 各変数間の相互情報量計算メソッド
- データを用いたBN構造学習メソッド
- BNの構造に対するMDL計算メソッド

等などを追加したい

ただ、今後のさらなる拡張の可能性を考慮して、
このクラスはそのままにしておきたい・・・



「クラスの継承」
を活用

クラスの継承(オブジェクト指向の必須要素)

```
class BN extends Probability { // Probability クラスの内容を継承
```

```
    protected String[] nameOfNode_;  
    protected int    numOfLinks_;  
    protected int    numOfParameters_;
```

クラス内の
変数宣言部

```
public BN(){
```

```
    super(); // Probability クラスのコンストラクタ呼出し  
    必要に応じてBN クラスに特有の処理を追加
```

```
    :  
}
```

コンストラクタ宣言部
super() を呼ぶ時は、
メソッドの最初を書く

```
public _setNameOfNode(int index, String name){
```

```
    // この(this)オブジェクトのindex番目のノードの名前
```

```
    // (nameOfRobot_) に nameを代入
```

```
    this.nameOfNode_ = name;
```

```
}
```

```
public 情報量規準計算用メソッド(){  
    メソッドの処理内容
```

```
    :  
}
```

```
}
```

BnRobotクラスに
特有のメソッド宣言部
("**this.**"は、「このクラスの」という意味。なくてもいい)

BN クラスのオブジェクトを作成した場合、

- BN クラスの変数やメソッドを利用できる
- Probability クラスの「public もしくは protected が頭につく」変数やメソッドを利用できる

例)

```
BN bn1 = new BN();
```

```
int h;
```

```
bn1._setNameOfNode(o, X1);    // BN クラスのメソッド  
h = bn1.getNumOfVariables();  // Probability クラスのメソッド
```



【小休止】

北越の研究テーマについて



後期・・・ゼミ2もあるので、簡単に研究内容を紹介しておきます
ベイジアンネット (BN)・・・北越の研究テーマの一つ

研究内容1：BN，ビッグデータ周辺に基づく様々なこと

テーマの一例(&現在興味のあること)

- いろいろな現象の表現と利用
(例: 高齢者の行動履歴、オープンデータの作成や活用)
- いろいろな現象の予測に利用
(例: 高齢者の体調予測、利用者が好む話題を予想し提供する、
利用者の習熟に応じて取組の難易度や負荷を調整する、
学生が間違いやすい作業について予め注意を促す)
- 理論的な研究
(例: BNの構造学習法や確率推論法の提案・改良)

北越が研究しているテーマの一つ



研究内容2：学習と適応に関する研究(BN研究も一部含む)

テーマの一例(&現在興味のあること)

- エージェント(ロボット)との対戦型ゲームにもとづく介護予防システム
(システムへ親しみを感じ、楽しみながら長期間利用できるシステム)
- 研究活動支援システム
(研究室ごとのノウハウを蓄積、学生の状況に応じて適切なノウハウを提示し、円滑な研究進捗を促進)
- 独居高齢者や障害者が生活しやすい環境づくりを目指す研究
(高齢者の生活状況から現在の体調や、今後起こり得る危険を予測、
障害者が自立して生活可能な環境づくりをIT技術を活用して実現)
- 理論的な研究
(例:ヒューマンエージェントインタラクション、
マルチエージェントシステム、ナレッジマネジメント等など)

多態性(ポリモーフィズム)とは



最後に残った、オブジェクト指向の必須要素・・・

多態性: 使用者が(見かけ上)同じメソッドを呼び出しても、実際にはその引数や呼び出した先のオブジェクトの違いによって、異なる処理を実行可能とすること

そのメリットは？

例) 日本円 100円を外貨に両替した時、いくらになるかを返すメソッド
`exchange(int yen)` を考える

ユーザは毎回`exchange`を呼び出すが、`exchange`が`Usa` クラスのメソッドなら米ドルに、`China` クラスのメソッドなら元に、`Mexico` クラスのメソッドならペソに換算される(呼び出されるメソッドはあくまで“`exchange`”)


多態性の実現方法はいくつかある・・・ 詳細はいずれ

実現方法の一例を[こちら](#)に示します(興味のある人は読んでおいて下さい)

オブジェクト指向(と私の研究テーマ)については簡単な紹介終了

⋮

Java の件はどうなった？

- 既にごくごく簡単ながらソースコードも紹介済み  基本的な記述方法はC言語と類似

なぜ**Java**でコーディングするのか(C++やその他の言語ではないのか)？

理由1) 教える人間(私)が **Java** しか知らないから(C++ は挫折しました)



オブジェクト指向という意味では、C++ よりも **Java** の方が厳密
言い換えれば、C++ では不十分

- C++ : C の拡張(オブジェクト指向でないプログラミングが「書けてしまう」)
- Java : オブジェクト指向でないプログラムは書けない

理由2) 記述方法が C言語 に近いので、入りやすい
if 文やfor 文など、多くの記述はCと同様



一部の表記(main メソッドの宣言など)や関数は異なるので、
“混乱しないように”注意しなければならない

理由番外編) ポインタを意識せずにコーディングできる

Java にはポインタがない、というのは嘘(らしい)
あるかないかはおいておいて、**Java** の(我々を書く)ソースコード中
にポインタが出てくることはない

その他、**Java** の特徴をまとめてみると・・・

Java のその他の特徴

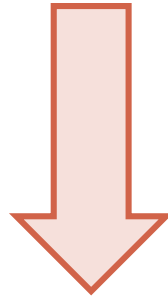


- **Java**仮想マシン(**JVM**)を利用することで、一度コンパイルしたコードはどのマシンでも、どんな**OS**でも動く(と言われている) : **Write Once, Run Anywhere**
- 純粹なるオブジェクト指向プログラミング言語
- メモリに気を遣う必要がない(メモリの確保や解放など)
- 企業の情報システムや携帯端末上で動くシステム(例:iアプリ)開発にも利用されている(**Java EE: Enterprise Edition, Java ME: Micro Edition**)

このような特徴を有する**Java** で実現できるオブジェクト指向プログラミングは・・・

- 大規模なシステム、拡張性のあるシステムの開発に適している
- 寿命の長いシステムの開発に適している(一度作ると拡張のたび、異なるマシンや**OS**へ移植するたびにコードを組みなおす必要がないから)

では、とりあえず実際にコーディングしてみましょう
(と言いながら、実はここからが今日の課題・・・)



つづきはこっち