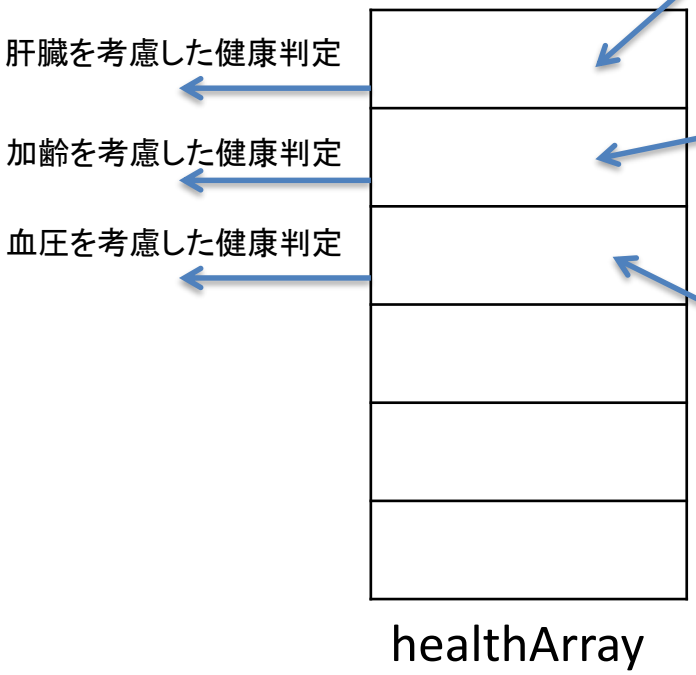
仮想関数と polymorphism

仮想関数とは？

- 基本クラスでは実体の無いメンバ関数（純粋仮想関数）
  - 例：virtual void draw (svg* svgObj) = 0; // Circle.h で宣言
- 派生クラスで実体を記述
  - 例：virtual void draw (svg* svgObj); // ColorCircle.h で宣言
  - void draw (svg* svgObj) {…} // ColorCircle.cpp で実装
- 異なる派生クラスの実体を，共通する基本クラスの実体集合として一括処理するのに用いる
  - 基本クラスのポインタ変数に派生クラスを代入できる
  - 基本クラスの仮想関数を呼び出すと，実際に代入されている派生関数の仮想関数が呼び出される

仮想関数利用のシナリオ

```
Health **healthArray;
healthArray = new Health* [6];
healthArray[0] = new Liver...
healthArray[1] = new Ageing ...
healthArray[2] = new Blood ...
...
healthArray[n]->isHealthy();
```




```
class Liver: public Health {
  int GPT, GOT; // 各種血液酵素の値
public:
  virtual bool isHealthy ();//仮想関数
};
```

```
class Ageing : public Health {
  int age; // 年齢
public:
  virtual bool isHealthy ();//仮想関数
};
```

```
class Blood : public Health {
  int pressure; // 血圧
public:
  virtual bool isHealthy ();//仮想関数
};
```

仮想関数の利用例

```
Health **healthArray;
healthArray = new Health* [6];
healthArray[0] = new Liver (“taro”, 1.65, 67, 20, 25); //GPT=20,GOT=25
healthArray[1] = new Ageing (“jiro”, 1.75, 80, 21); // age=21
healthArray[2] = new Blood (“kana”, 155, 50, 110); // pressure=110
...
 for (int i = 0; i < 6; i++) {
    if (healthArray[i]->isHealthy ())
        cout << healthArray[i]->getName () << “ is healthy !“;
    else
        cout << healthArray[i]->getName () << “ is NOT healthy !“;
}
```

異なるデータ形式（class）と判定方式（仮想関数 isHealthy）に対しても，同一の配列の並びで計算できる！

仮想関数を使用しない場合

- 派生クラスを識別するための変数を基本クラスに導入
  - 例：std::string type; // Health.h 内で宣言（文字列）
- 実体毎にクラスを判定して，キャストしてから共通名の関数を呼び出す
  - 例：if (healthArray[i]->type == "Liver") {

    ((Liver *) healthArray[i])->isHealthy ();
- どちらが記述量が少なく，拡張性に富むか？
  - 派生クラスが多数（例えば100種類）になった場合を想像する

仮想関数を使用しない実装例

```
Health **healthArray;
healthArray = new Health* [10];
healthArray[0] = new Liver (“taro”, 1.65, 67, 20, 25);
...
for (int i = 0; i < 10; i++) {
    bool hantei;
    if (healthArray[i]->type == "Liver")
        hantei = ((Liver*)healthArray[i])->isHealthy();
    else if (healthArray->type == "Ageing")
        hantei = ((Ageing*)healthArray[i])->isHealthy();
...
    if (hantei)
        cout << healthArray[i]->getName () << “ is healthy !“;
    ...
```

新たな派生クラスを作成する度に条件分岐の追加が必要なコード部分

拡張性に乏しい

仮想関数の変則的な呼び出し法

- 仮想関数で宣言されたメンバ関数は、派生クラスで宣言された関数の内容が実行される

→　これを、その基本クラスのメンバ関数を意図的に呼び出す方法が存在する。

その方法は、仮想関数を、

基本クラス名::メンバ関数名(...);

の形式で呼び出す。

仮想関数の変則的な呼び出し法

```
class Circle {
    virtual void draw () { cout << "Circle !"; }
...
class ColorCircle : public Circle {
    virtual void draw () { cout << "Color !"; }
```

```
ColorCircle *cc = new ColorCircle (1,2,3);
cc->draw(); // Color ! が出力される
cc->Circle::draw(); // Circle ! が出力される
```

まとめ

- 仮想関数で実現される、Polymorphism とは？
  - 多様性，多相性，多形性，多態性等の和訳
  - 要するに，共通の表現（基本クラス）で，多様な実体（複数の派生クラス）を操作できる事
- virtual というキーワードで仮想関数を特定する
  - Java では指定不要（デフォルトで仮想関数扱いとなる）
- 複数の異なる派生クラスを同様に扱う際に便利
  - 派生クラスを自動的に識別　→　条件分岐命令が不要
- 基本クラスでは，仮想関数の実装は不要
  - ただし、実装しても問題無い
    → 実体が基本クラスの際に呼び出される