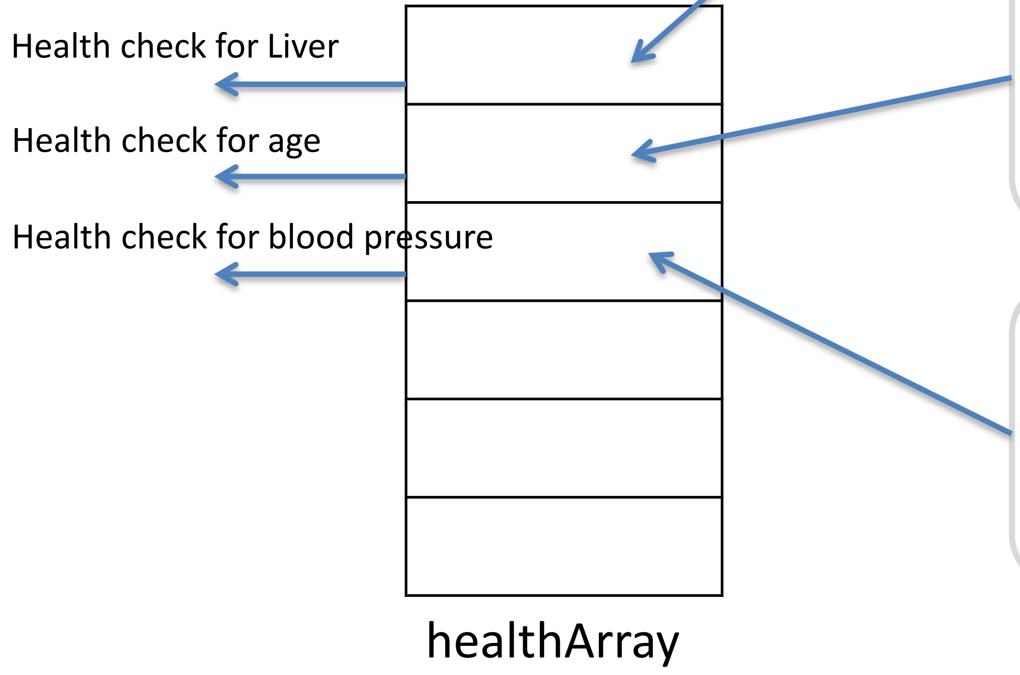
# Polymorphism with virtual function

# Virtual function

- Member function of **non** entity in a base class
  - virtual void draw (svg* svgObj)  = 0; // pure virtual function
  - virtual void draw (svg* svgObj); // virtual function
  - void draw (svg* svgObj) {…}// ordinary member function
- Unified process for different derived classes by using a common base class
  - Can assign a pointer of derived class to a pointer of a base class
  - Virtual function of the base class automatically calls the virtual function of the derived class

# How to use virtual function

```
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 ();
};
```

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

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

## Sample code

```
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 !“;
}
```

Can process a single array including different data classes and different virtual functions [ isHealthy() ]

# Code without virtual function

- Introduce a parameter for discriminating derived classes
  - std::string type; // define in Health.h
- Call commonly-named function with cast operator, after conditional branching with class names
  - if (healthArray[i]->type == "Liver") {

        ((Liver *) healthArray[i])->isHealthy ();
- Which representation is more compact and extensible?
  - for example, against hundreds of derived classes

# Code without virtual function

```
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 !“;
   …
```

Additional statement is required each time when new derived class is defined

NOT extensible!

# Irregular call of virtual function

- Virtual function executes the code implemented in a derived class of the same virtual function
-  →　Virtual function of base class can be executed by explicitly calling the name of base class using prefix as follows:

```
NameOfBaseClass::NameOfVirtualFunction(…);
```

## Irregular call of virtual function

```
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(); // print Color !
cc−>Circle::draw(); // print Circle !
```

# Summary

- Polymorphism with virtual function
  - In short, operate various instances of a common based class in a unified representation of function
- Add the keyword of "virtual" to normal functions
  - In Java, this keyword is unnecessary (default setting)
- Useful in managing different derived classes
  - Automatically discriminate derived classes

  → No conditional branch is required
- Implementation is omitted in a base class
  - However, virtual function for the base class is implementable

    → it is called when the instance is defined as a base class