プログラミング言語論
第5回

オブジェクト指向プログラミング

継承

今週の内容

- クラス（残り）
- 継承（inheritance ）

C++の機能と詳しいクラスの説明

- コンストラクタ・デストラクタ
- new / delete

コンストラクタ・デストラクタ（1/4）

- コンストラクタとデストラクタ
- オブジェクト生成時，消滅時に呼ばれるメソッドで，前者はコンストラクタ，後者はデストラクタ.
  - コンストラクタはクラス名と同じ名前でメソッド
  - デストラクタはそれに～を先頭につけたメソッド
  - 両者とも戻り値はなく，両者共にpublicに置く（後述）
- 初期化の設定，メモリの確保をコンストラクタで行い，メモリの解放をデストラクタで行う
  - これがあるだけで使用者はメモリの確保，開放の動作をしなくてよくなるため非常に楽に

コンストラクタ・デストラクタ（2/4）


```
class TestClass
{
public:
//これがコンストラクタ．戻り値は無い(戻り値型の指定がない).
        TestClass(){ printf("コンストラクタが呼ばれました¥n");}

//これがデストラクタ．戻り値は無い．　先頭に～を忘れずに.
        ~TestClass(){ printf("デストラクタが呼ばれました¥n");}

//これはただのメソッド
        void method(){ printf("methodが呼ばれました¥n");}
};
```

コンストラクタ・デストラクタ（3/4）


```
int main()
{
 TestClass a;
 a.method();
 {
   TestClass b;
   {
    TestClass c;
   }
 }
}
```

実行結果

```
コンストラクタが呼ばれました
methodが呼ばれました
コンストラクタが呼ばれました
コンストラクタが呼ばれました
デストラクタが呼ばれました
デストラクタが呼ばれました
デストラクタが呼ばれました
```

コンストラクタ・デストラクタ（4/4）


```
int main()
{
 TestClass a;
 a.method();
 {
   TestClass b;
   {
    TestClass c;
   }
 }
}
```

実行結果

```
コンストラクタが呼ばれました
methodが呼ばれました
コンストラクタが呼ばれました
コンストラクタが呼ばれました
デストラクタが呼ばれました
デストラクタが呼ばれました
デストラクタが呼ばれました
```

```
cに関するデストラクタ
bに関するデストラクタ
aに関するデストラクタ
```

スコープから外れたとき，その
インスタンスは寿命を終える．
その瞬間にデストラクタが呼ばれる．

スコープとインスタンスの寿命

TestClassのインスタンス “c” のスコープ

```
{
    TestClass c;
    ....



}
```

インスタンス生成時に
コンストラクタが呼ばれる

処理がこのブロックをでた瞬間
(cのスコープ外) このインスタンス
には２度とアクセスできなくなる.
このときデストラクタが呼ばれる.

コンストラクタとデストラクタの仕事

コンストラクタ（前準備）

- メンバ変数の初期化
- メモリ領域・計算機資源の確保（new）

デストラクタ（あと片付け）

- メモリ領域の開放
- 計算機資源の開放（例：ファイルのクローズ）

原則としてnew, delete はコンストラクタ・デストラクタにおいてのみ、使う

引数付きコンストラクタ

初期化用にコンストラクタに
引数をつけて宣言

```
class TestClass2
{
private:
        int a;
        int b;
public:
        //コンストラクタによる初期化．名前の衝突防止に_（アンダーバーを付与））
        TestClass2(int a_, int b_) {
                a=a_;
                b=b_;
        }
        void print() {
          printf("%d %d ¥n", a, b);
        }
};

int main(){
        TestClass2 ctest(1,2);
        ctest.print();
}
```

newとdelete：メモリ確保と開放

- C++ では，malloc, freeの代わりに確保・開放関数 new, deleteを使う
- デストラクタで開放すれば「開放忘れなし！」

```
int* a = new int;
delete a;
```

整数の確保と開放

```
int* a = new int[10];
delete[] a;
```

整数の配列の確保と開放

```
TestClass* a = new TestClass;
delete a;
```

クラスの確保と開放

newはコンストラクタの中だけ、deleteはデストラクタのなかだけで利用することが望ましい。（リソースへのアクセスの局所化）

例：動的1次元配列クラス


```
#include <stdio.h>
class IntArray{
private:
  int size;
  int* array;
public:
  IntArray(int _size){
    size = _size;
   array = new int[size];
    for (int i = 0; i < size; i++) array[i] = 0;
    }
 ~IntArray(){
    delete[] array;
    printf("destructor called¥n");
    }
void print_array(){
    for (int i = 0; i < size; i++)
printf("%d: %d¥n",i, array[i]);
    }
};

int main(){
  IntArray a(4);
  a.print_array();
}
```

デストラクタのメリット

- C++では計算機リソースの確保(メモリを含む）はコンストラクタで，開放はデストラクタで行う

- インスタンスが消滅するときにデストラクタが**「自動実行」**されるため，わざわざ開放のためのコードを書く必要がなく，開放忘れが抑制.

- **「計算機資源へのアクセスの局所化」**に役立つ

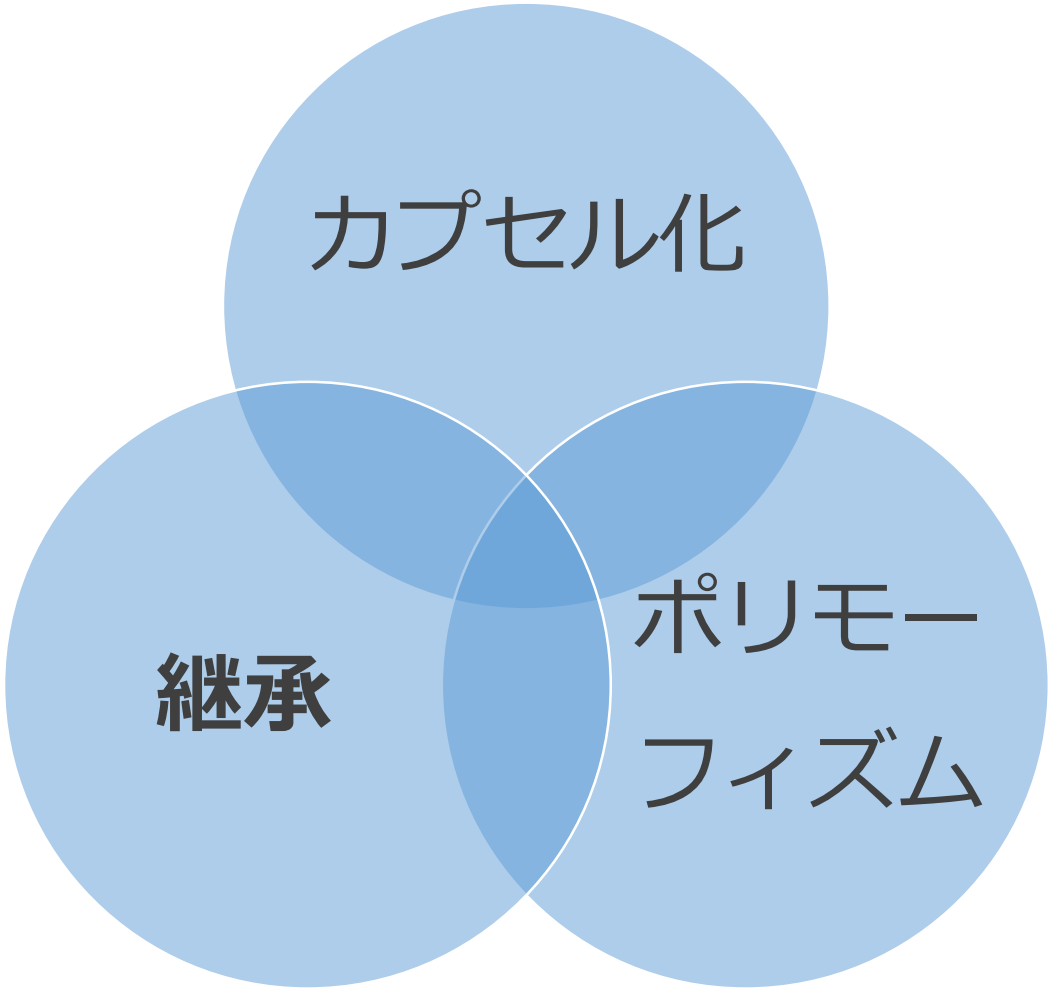
計算機資源へのアクセスの局所化

- 例えば，メモリリークを起こす３万行のソースコードからなるプログラムをデバッグしているとする．もし，局所化のルールが守られているなら，デバッグ対象はプログラムで利用したクラス群のデストラクタだけを確認すればよい（メモリ開放失敗の疑い）．
- 守られていないなら，どのタイミングで開放を忘れたのか，開放が失敗したのかを，**３万行**の中から見つける必要．
- メモリリークは，誰もアクセスできないスコープに開放されていないメモリが残ってしまうことに起因．
- この話は，メモリだけではなく，他の計算機リソース（例えば，ファイルなど）でも同じ．

ものぐさの言い分

- デストラクタ書くの面倒だよね．．．複雑なデータ構造を開放するときなんか特に．．．
- Javaを使うとメモリ開放をユーザ側は忘れても良い．スコープから外れた変数のメモリ領域は「**ガベージコレクタ**」が勝手に開放．
- でもアプリによってはガベージコレクタが使えないこともある（リアルタイム処理など）
- C++で楽をするには，**スマートポインタ**を使うと良い
  - 意欲ある人は自分で調べてみるとC++で少し幸せになれる

オブジェクト指向を構成する3大要素



※昔は，抽象データ型，派生，仮想関数の３要素

継承とは

- あるクラスの変数やメソッドを引き継いで，新たなクラスを作ること

- 継承先では，基底クラスで定義されていたメンバ関数やメソッドを**一切書く**必要が無く，**追加機能の実装だけに専念**できるため，簡潔に記述が出来る

- 抽象クラスの場合、継承を使って派生クラスを作り，そのクラスに具体的な実装をおこなう.

用語集：継承する・されるクラスの呼び方　18

- 継承元のクラス：
  - 基本クラス，基底クラス，ベースクラス，親クラス，スーパークラス
- 継承先のクラス：
  - 派生クラス，子クラス，サブクラス

C++による継承


```
#include <stdio.h>
class Super
{
public:
void methodSuper()
{
printf("super¥n");
}
};

class Sub :public Super
{
public:
void methodSub()
{
printf("sub¥n");
}
};
```

クラス宣言の後に:publicと書いて継承元のクラスを指定するだけ.

※private, protectedでも良いが意味が違う．詳細は割愛

```
int main()
{
Super a;
Sub b;

b.methodSub();
b.methodSuper();

return 0;
}
```

継承したクラスは親クラスのメソッドも子クラスのメソッドも使える.

継承の例

基底クラス

継承

派生クラス


```
class Sub :public Super
{
public:
void methodSub()
{
printf("sub¥n");
}
};
```

```
class Super
{
public:
void methodSuper()
{
printf("super¥n");
}
};
```

派生クラスの定義においては
「差分」のみを書けばよい

継承の例


```
#include <stdio.h>
class Super
{
public:
void methodSuper()
{
printf("super¥n");
}
};

class Sub :public Super
{
public:
void methodSub()
{
printf("sub¥n");
}
};
```

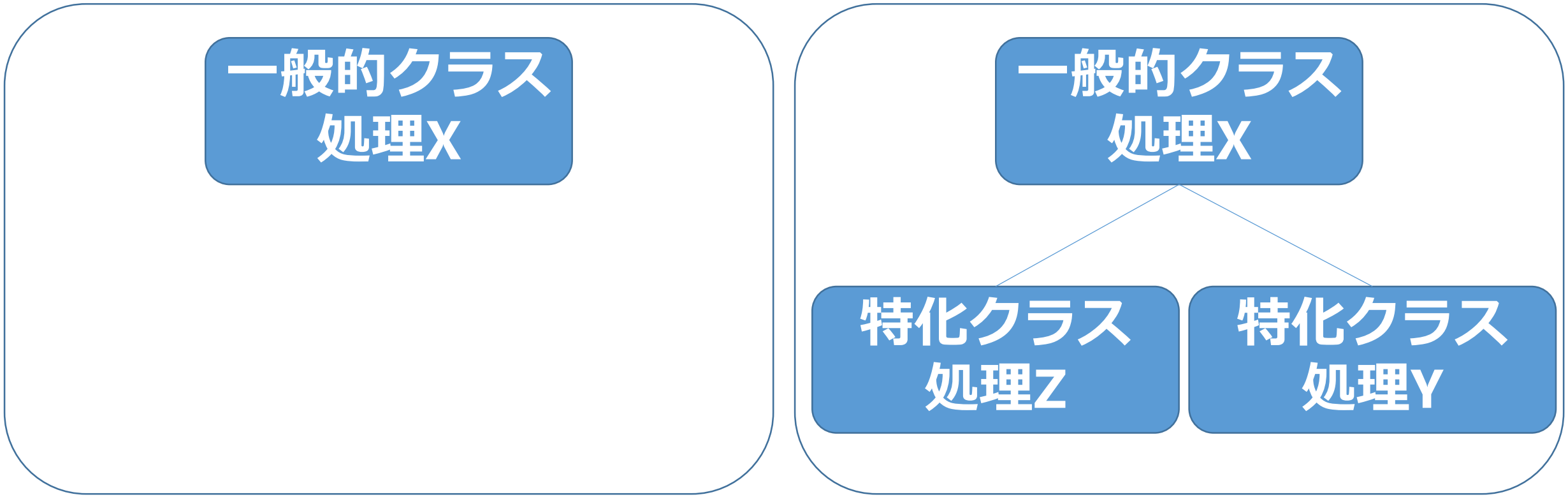
```
class Sub2 :public Super
{
public:
void methodSub()
{
printf("sub¥n");
}
};

int main(){
Super a;
Sub b;
Sub2 c;
b.methodSub();
b.methodSuper();
c.methodSub();
c.methodSuper();
return 0;
}
```

複数のクラスを継承して作っても親クラスのメソッドを子クラスのメソッドが使える.

継承の使い方：一般化・特殊化

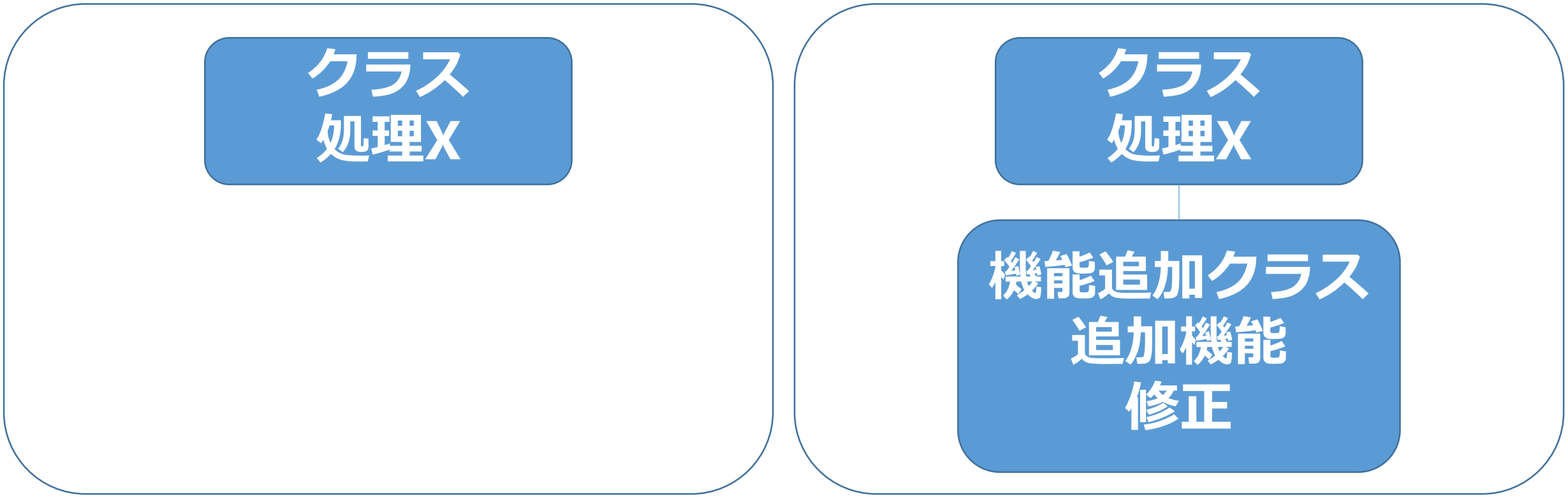
親クラスで一般的な機能を定義し，
子クラスで目的特化の機能を定義する



継承の使い方：差分プログラミング

既存のクラスを拡張するために継承して
追加部分を書く方法

変更・追加の他コードへの影響を抑える



差分プログラミングの恩恵（１）

- 似た機能を持つクラスを作りたいときにどうする？
- コピペでしょ！クラスAのコードをコピーして、作りたいクラスBにペタッと．あとはBに必要な機能を追加してっと．できあがり！
- 上のアプローチは，同じようなコードがプログラム中に繰り返し現れて，それも少しづづ異なっているという**酷い状況**.
- クラスAのメソッドに，もしバグが含まれていたら，コピペ先のコードへもそのバグが波及．エンドレスのデバッグへ.
- コピペは撲滅！BはAの派生クラスとして書くべし.
  - 可能な限りコピペを抑制する機能を使ってコードを書く
  - 高速なコードを書くために，仕方なくコピペを使うこともある

差分プログラムの恩恵（２）

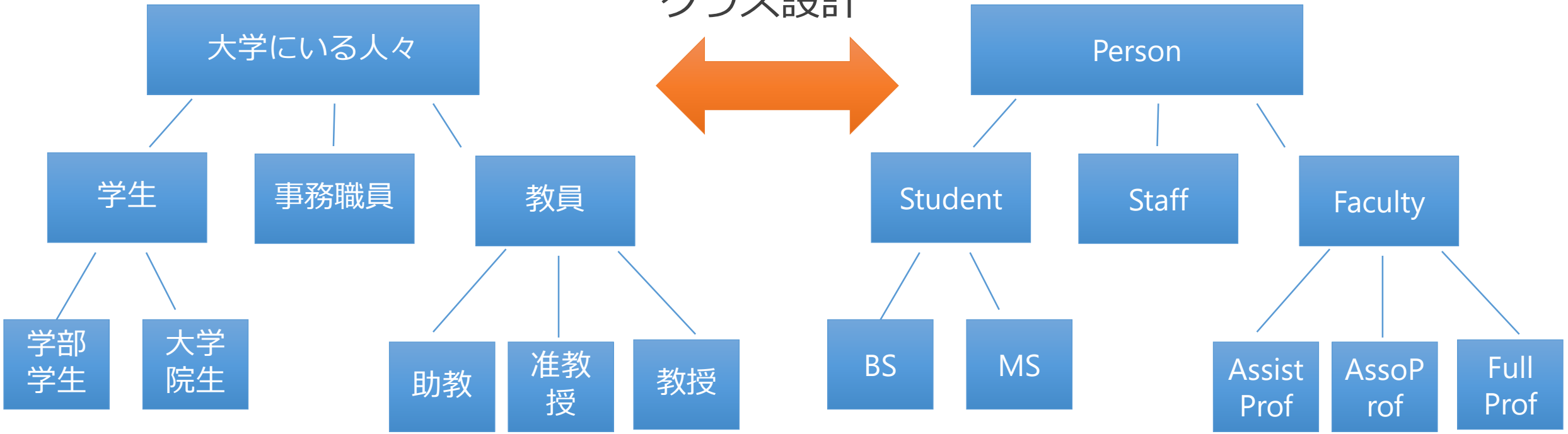
- このライブラリに含まれているクラスにちょっと機能を追加できるといいんだけどな．．．APIドキュメントはあるけどライブラリのソースコードは無いしな．．．
- 問題なし．そのクラスの派生クラスをユーザ側で書けばよい．好きなようにカスタマイズできる

大学に属する人のデータベースを作りたい　26

現実のカテゴリ間の包含関係

対応するようにクラス設計

クラスの継承関係



継承時のアクセス制御

- privateなメソッド，メンバ変数は外部から直接アクセスできなかったのと同様に，Public継承では派生クラスから基底クラスのprivate メンバにアクセスできない
  - publicメンバ，メソッドにはアクセス出来る
- Public継承以外に、protected継承、private継承があるが基本はpublic 継承を利用すると考えておけばよい.
- 基底クラスのprotected メンバ，メソッドは派生クラスからアクセスできる．可能な限りprotected メンバ・メソッドよりもprivateを使うべき

小テスト：継承 （次ページコード参照） 28

1. Superクラスの変数a, b, cにmain関数からアクセスしようとしたときに何が起こるか観察せよ.
2. Subクラスのshowから変数a, bにアクセスが出来ていることを確認せよ．クラス定義内のprotected 宣言の意味を述べよ.
3. Subクラスのshowから変数cにアクセスができないことを確認せよ.
4. Protected 宣言は便利だが，なるべく使わないほうが良いと言われることも多い．その理由を考察せよ.

前ページ（小テスト）のためのプログラム　29

```
#include <stdio.h>
class Super
{
protected:
  int b;
private:
  int c;
public:
  int a;
  Super()
    {
     a = 1;
     b = 2;
     c = 3;
    }
};
class Sub: public Super
{
public:
  void show(void)
    {
     printf("%d¥n",a);
     printf("%d¥n",b);
    }
};
int main()
{
  Super x;
  Sub y;
  y.show();
}
```

レポート5-1：コンストラクタ・デストラクタ30

席の数，机の数，プロジェクタの数，黒板の数をメンバ変数に持つクラスを作り，コンストラクタで初期化せよ

- まず，何もしないコンストラクタをまず作り，メンバ変数の値を表示せよ.
- 次に，席，机，プロジェクタ，黒板の数を変更できるコンストラクタを**オーバーロード**（引数の型が異なれば複数のコンストラクタを定義できる）し，同様に表示せよ
  - ヒント：class Kyoshitsuに対して，下記コンストラクタ を作成
    - Kyoshitsu(int zaseki, int tsukue, int projector, int kokuban)

オーバーロード

- 日本語で書けば関数の多重定義：引数が違えば同じ名前で違う挙動をする関数がいくつも書ける
- 例えばfunc(int a)とfunc(short a)で全く違う動作を定義可能（ただし，そのようなトリッキーなコードはバグの要因になるのでオーバーロード時は似たような操作を書くこと．）
  - 良くない例
    - int func(char a){return a;}
    - int func(short a){return a*a;}
    - int func(int a){return a*a*a*a;}

レポート5-2：継承

12ページの動的１次元配列のクラスIntArrayから継承して（簡単のため12ページのコードでprivateになっているところをprotectedに変えること），新しいクラスを作成せよ.

ただし，新しいクラスは，配列の各要素に数値を設定する公開メソットsetと，配列内の数値の合計を返す公開メソッドsumを持つものとする.