C++ 008 アクセス指定子

【メンバ変数／関数のアクセス指定子】

```
class sample {
public:
    メンバ変数・関数の記述
protected:
    メンバ変数・関数の記述
private:
    メンバ変数・関数の記述
};
```

アクセス指定子はクラスのメンバが他クラス、他関数から使用できるのか、できないのかの制限をかけるものです。
メンバ変数・関数には下記の 3 種類のアクセス指定子があります。

アクセス指定子	同クラス	派生クラス	どこからでも
public	○	○	○
protected	○	○	×
private	○	×	×

○ 使用できる、× 使用できない

アクセス指定子を省略した場合、自動的にクラスは「private」、構造体は「public」になります。

```
class sample {
private:                    //以下のメンバは private
    int var;
public:                     //以下のメンバは public
    void func();
};

void sample::func() {
    var = 10;               //privat メンバは自クラスのみで使用できる
}

int main(){
    sample instance;        //クラス外で sample のインスタンス化
    instance.var = 10;      //privat メンバは使用できない。この行はエラーになる
    instance.func();        //public メンバは使用できる
```

【カプセル化】

クラスを使用する場合、使用できる外部化（public）されたメンバの使い方さえ分かればいいです。使用できない内部化（private）されているメンバを理解する必要はありません。

アクセス指定子を使用して、外部化、内部化を明確にした作りを「カプセル化」と称します。

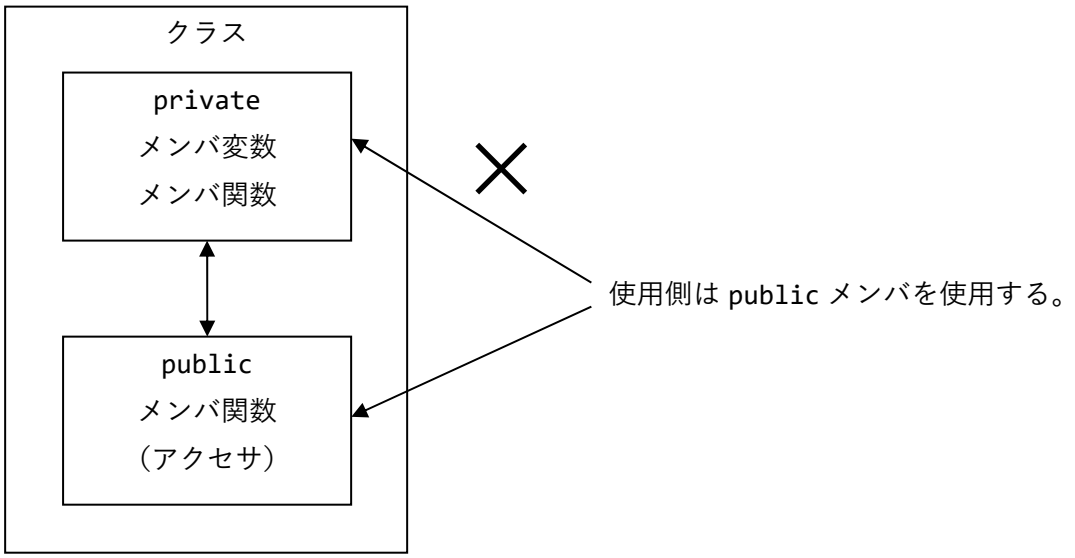
＜メンバ変数の隠蔽＞

カプセル化ではメンバ変数はアクセス指定子を「private」にして隠蔽し、アクセサメソッドを介して値の入出力を行います。

メンバ変数に値を出し入れするメンバ関数を「アクセサ」と称し、値を代入するメンバ関数を「セッター」、値を取得するメンバ関数を「ゲッター」と称します。

「そのメンバ変数は 0～100 までしか代入できない」場合は点数のセッターに処理を、「メンバ変数の型は数値型だけど、戻り値が文字列型で返したい」場合はゲッターに処理を記述します。メンバ変数を読み取り専用にしたいならゲッターのみ定義すればいいです。

メンバ変数を public にすると、直接値を代入、参照できてしまい、この手の処理が組み込めなくなります。



＜カプセル化のメリット＞

- 重要なメンバ変数、メンバ関数を隠蔽できる。
- クラスの使い方を簡潔できる。
- インターフェース（抽象クラス）が適用しやすい。
- プログラムの修正箇所を対象化しやすい。

C 言語スタイルの構造体を使用する場合、メンバ変数のみ使用する事が多いので、アクセス指定子は「public」で宣言することが多いです。

＜例文＞

```
//テストの点数を記録するクラス
class record {
private:

    //メンバ変数は隠蔽
    int language;       //国語の点数
    int mathematics;    //数学の点数
    int english;        //英語の点数

public:

    record();                   //デフォルトコンストラクタ
    record(int, int, int);      //初期化用コンストラクタ

    //必要に応じてアクセッサを作る
    void set_language(int);     //国語の点数に代入するセッター
    int get_language();         //国語の点数を取得するゲッター
    int get_total();            //3 教科の合計点を取得するゲッター
};

record::record() {
    language = 0;
    mathematics = 0;
    english = 0;
}

record::record (int language, int mathematics, int english) {
    this->language = language;
    this->mathematics = mathematics;
    this->english = english;
}

void record::set_language(int language) {

    //入力値をチェックする
    if (language < 0) { language = 0; }
    if (language > 100) { language = 100; }

    this->language = language;
}

int record::get_language(){
    return language;
}

int record::get_total() {
    return language + mathematics + english;
}
```

【friend】

アクセス指定子が「private」なメンバは自クラス内でしか使用できません。

```
class classA {
private:
    int var1;    //このメンバは classA しか使用できない
public:
    int var2;    //このメンバはどのクラスも使用できる
};

class classB {
public:
    void func()；
};

void classB::func() {
    classA instance;
     instance .var1 = 10;    //var1 は「private」なのでエラー
     instance .var2 = 10;
}
```

＜friend クラス＞

「friend クラス」に指定されたクラスは、アクセス指定子に関係なくメンバを使用できます。

```
class classB;           //前方宣言。classA が classB を使用するために必要

class classA {
friend class classB;    //classB を friend クラスに指定
private:
    int var;            //このメンバは private メンバ
};

class classB {
public:
    void func();
};

void classB::func() {
    classA instance;
    instance.var = 10;      //classB は friend クラスなので使用可能
}
```

friend クラスを記述しているクラスを継承しても、派生クラス側にその効果はありません。

＜friend 関数＞

「friend 関数」に指定された関数は、アクセス指定子に関係なくメンバをできます。

```
class classA {
private:
    int var;              //このメンバは private メンバ
};

int main() {
    classA instance;
    instance.var = 10;    //エラーになる
```

```
class classA {
private:
    int var;              //このメンバは private メンバ
public:
    friend int main();    //メイン関数を friend 関数に指定
};

int main(){
    classA instance;
    instance.var = 10;    //メイン関数は friend 関数なので使用可能
```

「friend 関数」は「メンバ関数」ではないのに注意。friend 関数は通常の関数を friend 化するものであり、記述上は classA のメンバ関数に見えるが、メンバ関数ではないので

classA instance;

instance.main();

なんて記述はできません。