

[戻る](#)

- [継承](#)
 - [新しいクラスを作る](#)
 - [派生クラスの宣言](#)
- [メンバへのアクセスの制限 その2](#)
 - [派生クラス 基本クラス](#)
 - [外部 派生クラス 基本クラス](#)
- [オーバーライド](#)
 - [同じ名前に関数を定義](#)
 - [どっちが呼ばれる？](#)

継承

新しいクラスを作る

車、と一言で言ってもさまざまな車種があります
例えばバスなら、走るコースを決めたり表示する機能なども必要です
しかし車であることに変わり無く、ガソリンやナンバーが必要なのは共通です
それなら、最初にしたCarクラスに新しい機能を追加するだけでバスを作れます
このように、既に作成したクラスを使って、新しいクラスを作成することを

クラスを派生する(extends)

と云い、新しく作成したクラスが、既存のクラスのメンバを受け継ぐことを**継承**と言います

派生クラスの宣言

派生クラスは次のように宣言します

```
class 派生クラス名 : アクセス指定子 基本クラス名
{
    追加するメンバの宣言
};
```

例として、先のCarクラスを使ってバスのクラスBusを作成します

```
class Bus : public Car
{
    private:
        int course
    public:
        Bus();
        void setCourse();
};

Bus::Bus()
{
    course = 1;
}

void Bus::setCourse()
{
    int c;

    scanf("%d",&c);
    printf("%d番線を走行します\n",&c);

    course = c;
}
```

Carクラスを継承しているので、Busクラスではnumやgasなどの変数を書き直す必要がありません
また、クラスを継承しているので、Carのコンストラクタも自動的に実行されます
あくまで、Carにない箇所だけを記述すればいいわけです

メンバへのアクセスの制限 その2

派生クラス 基本クラス

派生クラスから基本クラスのメンバにアクセスする場合、基本クラスのpublicメンバなら問題ありません
ですが、privateメンバは基本クラスはもちろん、派生クラスからもアクセス出来ません
アクセスが必要な場合は、基本クラスのアクセス指定子をprotectedにします

```
class Car
{
    protected:
        int num;
        double gas;
    public:
        Car();
        . . .
}
```

protectedメンバにすることで、派生クラスの内部からならアクセス出来ます

外部 派生クラス 基本クラス

ここまで来るとさすがに説明がシンドイんですが、まずは下のソースを見てください

```
class Bus : public Car
{
    . . .
}
```

基本クラスの前に付いているアクセス指定子によってアクセス制限が変わります
まとめると、こんな感じです

基本クラスでのアクセス指定	継承の仕方	派生クラスからの利用	外部からの利用
public	public		
protected	public		×
private	public	×	×
public	protected		× (派生クラスのprotectedメンバになる)
protected	protected		×
private	protected	×	×
public	private		× (派生クラスのprivateメンバになる)
protected	private		×
private	private	×	×

正直ややこしいですが、オブジェクト指向を使うには避けては通れない道です

オーバーライド

同じ名前で関数を定義

派生クラスで新しくメンバ関数を書く時に

基本クラスと全く同じ関数名・引数の数・型を持つメンバ関数を定義できる

ようになっています

```
void Car::show()
{
    . . .
}

void Bus::show()
{
    . . .
}
```

スコープ解決演算子を用いることで、どのクラスの関数かを明確にします

どっちが呼ばれる？

さて、次のソースを見てください

```
Bus bus1;

bus1.show();
```

さて、先の関数を定義した後では、show関数はどちらのクラスから呼ばれるでしょうか？

全く同じ関数の場合、派生クラスのメンバ関数が優先的に呼ばれます

このように、派生クラスで定義したメンバ関数が、基本クラスのメンバに代わって機能することを**オーバーライド**と言います