

プログラミングII

<http://bit.ly/Prog3i>

継承 (1)

後期 第6週

2019/11/6

本日は・・・

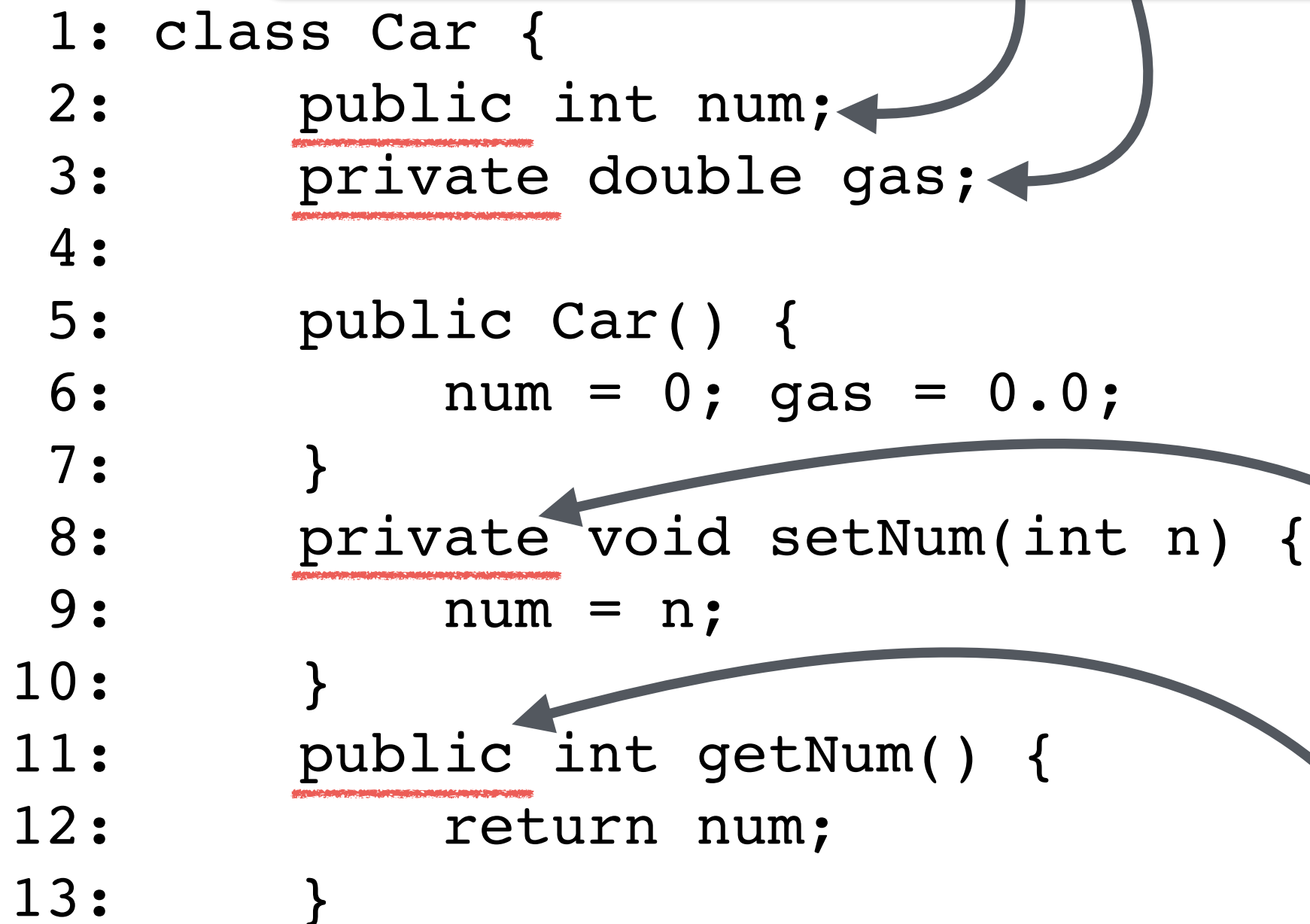
- ▶ 可視性によるメンバへのアクセス制御
- ▶ 継承の基礎

本日は・・・

- ▶ 可視性によるメンバへのアクセス制御
- ▶ 継承の基礎

【Point 1】メンバの可視性に**public**を指定すると、クラスの外部から参照することができるようになる。(35行目)
が、**フィールドは基本的にprivateにする。**


```
1: class Car {  
2:     public int num;  
3:     private double gas;  
4:  
5:     public Car() {  
6:         num = 0; gas = 0.0;  
7:     }  
8:     private void setNum(int n) {  
9:         num = n;  
10:    }  
11:    public int getNum() {  
12:        return num;  
13:    }
```



【Point 2】**private**を指定すると、参照できるのはクラスの内部のみで、クラスの外部からは参照することができなくなる。(36行目) が、**メソッドは基本的にpublicにする。**


【Point 3】フィールドに値を代入するメソッドは「セッタ(setter)」と呼ばれ、「set + フィールド名」と名前を付ける習慣がある。
この例では、このセッタを使うことで、フィールドgasには指定した範囲内の値の代入が保証されることになる。

```
14:      public void setGas(double g) {  
15:          if(g > 0 && g < 1000 ) {  
16:              gas = g;  
17:          } else {  
18:              System.out.println("Out of range.");  
19:          }  
20:      }
```



【Point 4】フィールドの値を取得するメソッドは「**ゲッタ** (getter)」と呼ばれ、「**get + フィールド名**」と名前を付ける習慣がある。

```
21:    public double getGas() {  
22:        return gas;  
23:    }  
24:    public void show() {  
25:        System.out.println(" (num)" + num  
                                + " (gas)" + gas);  
26:    }  
27: }  
28:
```




```
29: class Pd8car1 {
30:     public static void main(String[] args) {
31:         Car c1 = new Car(); c1.show();
32:         c1.setGas(500); c1.show();
33:         c1.setGas(-100); c1.show();
34:
35:         c1.num = 1234;
36:         //c1.setNum(5678); //Error
37:         c1.show();
38:     }
39: }
```

【Point 5】フィールドと、それを使って処理するメソッドを1つのクラスにまとめることを「**カプセル化**」と呼び、フィールドをprivateとし、それにアクセスするメソッド(**アクセッサ** (accessor) = セッターとゲッターの総称)を用意することを「**情報隠蔽** (データ隠蔽)」と呼ぶ。

【Point 6】public, privateを省略すると、「**package**」を指定したことになる。packageは**同じパッケージ内** (同じフォルダ内) のクラスからのみアクセス可能な可視性となる。

【課題の準備】

演習室で作業する前に、以下のコマンドを
入れるだけで準備が完了する

```
$ mygitclone 「自分のGitHubユーザ名」  
$ cd prog3i-ユーザ名  
$ ./myconf
```

※本体をシャットダウンするまでは、
上記「mygitclone」と「myconf」の設定は有効です

【課題の準備】

以下の流れで、課題のプログラムを作るためのフォルダを準備しましょう。

1. 端末を起動して、以下のコマンドを実行して後期第6週のフォルダを作る
\$ cd prog3i-ユーザ名 (←既に移動しているなら不要)
\$ mkdir week206
\$ cd week206

【練習6-1】

サンプルプログラム「2_06_Car.java」を
コンパイルして、実行結果を確認しましょう。

【課題6-1】

サンプルプログラムの36行目のコメントを外してコンパイルし、**どのようなエラーが出力されるのか**を確認してください。

そして、36行目のエラーの原因となっているメソッドsetNumを**public**となるようにプログラムを修正し、動作を確認してください。

（その際、35行目は必要なくなる。）

【課題の提出】

以下の流れで、作ったCプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A
```

```
$ git commit -m “課題6-1提出”
```

```
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する


[https://github.com/nit-ibaraki-prog3i/prog3i-\(ユーザ名\)](https://github.com/nit-ibaraki-prog3i/prog3i-(ユーザ名))

本日は・・・

- ▶ 可視性によるメンバへのアクセス制御
- ▶ 継承の基礎

【Point 9】 スーパークラスの**private**メンバは、サブクラスからでもアクセスできない。（privateは、サブクラスも「外部のクラス」として扱う。） **protected**メンバは、サブクラスからはアクセスできるようになる（その他のクラスは「外部のクラス」となる。）

```
1: class Car {  
2:     protected int num;  
3:     private double gas;  
4:  
5:     public Car() {  
6:         num = 0; gas = 0.0;  
7:     }  
8:     public void setNum(int n) {  
9:         num = n;  
10:    }  
11:    public int getNum() {  
12:        return num;  
13:    }
```




The diagram consists of two curved arrows originating from a common point on the right side of the slide. One arrow points to the `protected int num;` line (line 2), and the other points to the `private double gas;` line (line 3). This illustrates that both protected and private members of a superclass are accessible from a subclass, which is a key concept in Java's access control.


```
14:      public void setGas(double g) {
15:          if(g >0  && g < 1000 ) {
16:              gas = g;
17:          } else {
18:              System.out.println("setGas: Out of
range." );
19:          }
20:      }
21:      public double getGas() {
22:          return gas;
23:      }
24:      public void show() {
25:          System.out.println("show: (num) "
+ num + " (gas) " + gas);
26:      }
27:  }
28:
```

【Point 7】 「**extends**」 の後続くクラスを継承する。継承は、**クラスを拡張する**ために利用する。この場合、クラスCarを継承して、クラスRacingCarを定義している。

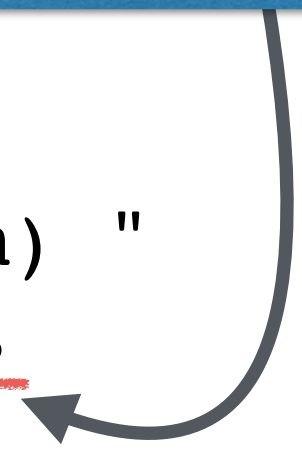
- Car → **スーパークラス** (親クラス)
- RacingCar → **サブクラス** (子クラス)
- 追加されるフィールド course
- 追加されるメソッド getCourse, setCourse, show



```
29: class RacingCar extends Car {
30:     private int course;
31:     public int getCourse() {
32:         return course;
33:     }
34:     public void setCourse(int c) {
35:         course = c;
36:     }
```

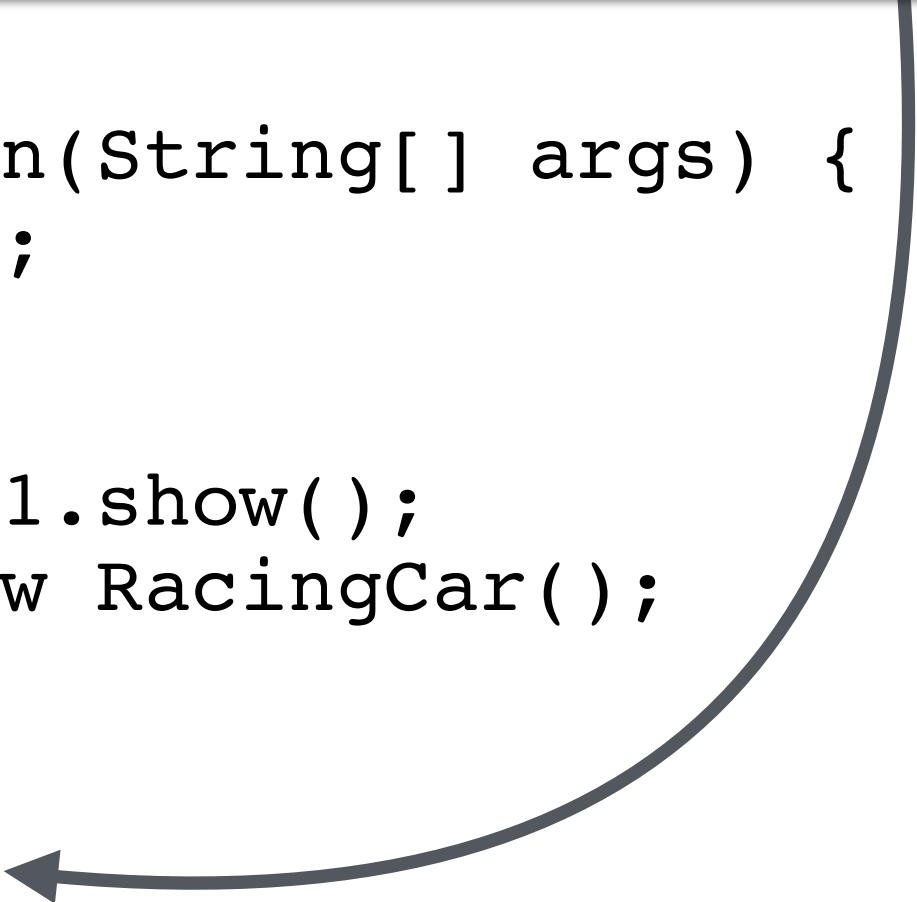
【Point 10】 スーパークラスのprotectedメンバ（ここではnum）にはサブクラスからはアクセスできるが、privateメンバ（ここではgas）にはアクセスできない。（エラーとなるためコメントにしている。）

```
37:      public void show2() {  
38:          System.out.println("show2: (num) "  
          + num // + " (gas) " + gas  
39:          + " (course) " + course);  
40:      }  
41: }  
42:
```

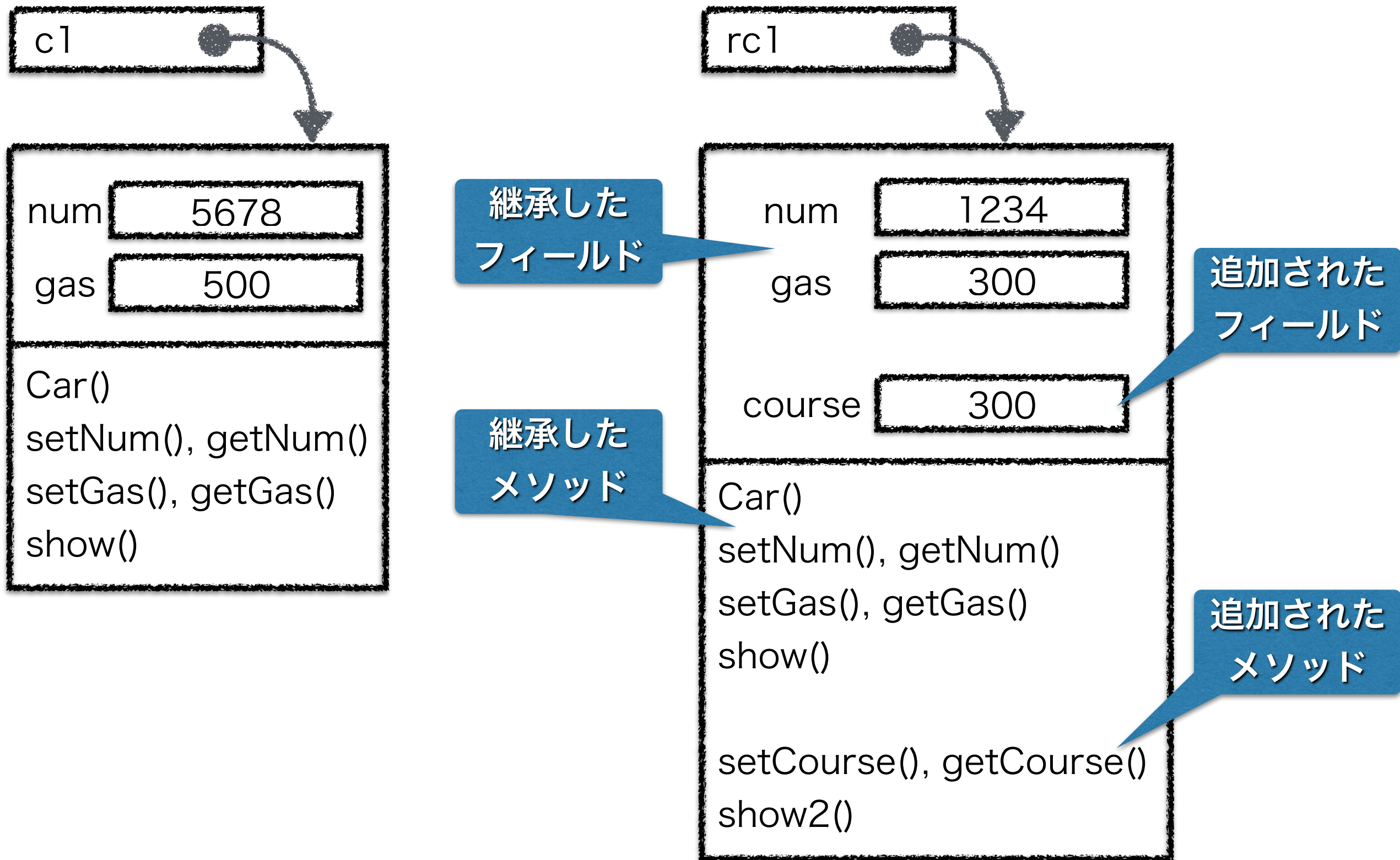


【Point 8】 継承したクラスは、**スーパークラスのメソッド**を呼び出すことができ（ここでは setNum, setGas, showを呼び出している）、さらに、**追加したメソッド**も呼び出せる。（ここではsetCourse, show2を呼び出している）

```
43: class Pd9car1 {
44:     public static void main(String[] args) {
45:         Car c1 = new Car();
46:         c1.setGas(500);
47:         c1.setGas(-100);
48:         c1.setNum(5678); c1.show();
49:         RacingCar rc1 = new RacingCar();
50:         rc1.setCourse(7);
51:         rc1.setNum(1234);
52:         rc1.setGas(-200);
53:         rc1.setGas(300);
54:         rc1.show();
55:         rc1.show2();
56:     }
57: }
```



インスタンスの様子



【練習6-2】

サンプルプログラム「2_06_Car2.java」を
コンパイルして、実行結果を確認しましょう。

【課題6-2】

サンプルプログラムの38行目のコメントを外してコンパイルし、**どのようなエラーが出力されるのか**確認してください。

そして、このエラーの原因となっているフィールド `gas` を **protected** となるようにプログラムを修正し、動作を確認してください。

【課題の提出】

以下の流れで、作ったCプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A
```

```
$ git commit -m “課題6-2提出”
```

```
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

[https://github.com/nit-ibaraki-prog3i/prog3i-\(ユーザ名\)](https://github.com/nit-ibaraki-prog3i/prog3i-(ユーザ名))

【課題6-3】

次のような整数を扱うクラスBaseを考えます。
(「2_06_Base.java」から入手可能)

```
class Base {  
    protected int num;  
    public void setNum(int n) {  
        num = n;  
    }  
    public int getNum() {  
        return num;  
    }  
    public void showNum() {  
        System.out.printf("num: %d\n", num);  
    }  
}
```

(課題は次のスライドに続きます)

【課題6-3】

クラスBaseを継承して、次のようなメソッドが追加されたクラスEvenを作成して、動作を確認してください。

```
public void setEven(int n)
```

```
//引数nが偶数の場合、nをフィールドnumに代入する
```

(mainを持つクラスとその実行結果は、
課題6-4とまとめてあります。)

【課題の提出】

以下の流れで、作ったCプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A
```

```
$ git commit -m "課題6-3提出"
```

```
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

[https://github.com/nit-ibaraki-prog3i/prog3i-\(ユーザ名\)](https://github.com/nit-ibaraki-prog3i/prog3i-(ユーザ名))

【課題6-4】

クラスBaseを継承して、次のようなメソッドが追加されたクラスOddを作成して、動作を確認してください。

```
public void setOdd(int n)
```

```
//引数nが奇数の場合、nをフィールドnumに代入する
```


【課題6-4】

このクラスはファイル「2_06_Main.java」に含まれている

```
class Pd6Base {
    public static void main(String[] args) {
        Base b1 = new Base();
        //Baseはフィールドnumに自由に値を代入できる
        b1.setNum(12); b1.showNum();
        b1.setNum(5); b1.showNum();

        Even e1 = new Even();
        //Evenはnumに偶数のみ代入できる
        e1.setEven(12); e1.showNum();
        e1.setEven(5); e1.showNum();

        Odd o1 = new Odd();
        //Oddはnumに偶数のみ代入できる
        o1.setOdd(12); o1.showNum();
        o1.setOdd(5); o1.showNum();
    }
}
```

【課題6-4】

[実行結果]

| | |
|-------------------|----------------|
| num: 12 | (← b1に対する処理) |
| num: 5 | |
| setEven: 値を代入します | (← e1に対する処理) |
| num: 12 | |
| setEven: 値を代入しません | |
| num: 12 | |
| setOdd: 値を代入しません | (← o1に対する処理) |
| num: 0 | |
| setOdd: 値を代入します | |
| num: 5 | |

【課題の提出】

以下の流れで、作ったCプログラムをGitHubにプッシュして、Webサイトで確認してみましょう。

1. 端末内で、以下のコマンドで課題を提出

```
$ git add -A
```

```
$ git commit -m "課題6-4提出"
```

```
$ git push origin master
```

2. 自分のリポジトリを開いて、提出したファイルがプッシュされているか確認する

[https://github.com/nit-ibaraki-prog3i/prog3i-\(ユーザ名\)](https://github.com/nit-ibaraki-prog3i/prog3i-(ユーザ名))

小テストについて

小テストの注意点

- 他人の力は借りずに、自分だけでプログラムを作成する。（つまり定期試験と同様）
- プログラムの提出はGitHubを使用する。

小テストについて

小テスト中に参照できるもの

- 教科書, 参考書, 配付資料
- 自分のホームディレクトリ（ホームフォルダ）以下に保存されているファイル
- 小テストでは紙媒体のものは参照可能
- 上記以外の情報を参照することは不正行為とする
例：USBで接続された機器に保存されているファイルの参照
Webブラウザ、ネットワークを介した情報の参照
自分のPCを使用する、など

小テストの追試について

11月14日のプログラミングII試験後に、小テストの追試験を実施します。未受験の小テストがある人、前回提出がうまくできなかった人は必ず受けて下さい。

未受験分の小テストは0点として評価します。

試験範囲

▶ 第1週～第6週

▶ クラス

▶ 継承

定期試験の実施について

試験中に使用できるもの

- 筆記用具

(メモ用紙が必要な人には試験中に配布する)

- 演習室のコンピューター台

(一つの机に一人の配置で、座る場所はどこでもよい)

定期試験の実施について

試験中に参照できるもの

- 自分のホームディレクトリ（ホームフォルダ）以下に保存されているファイル
（定期試験では紙媒体のものは参照不可）
- 授業の資料や自分のGitHubリポジトリなどは事前にダウンロードまたはコピーしておく
- 上記以外の情報を参照することは不正行為とする
例：USBで接続された機器に保存されているファイルの参照
Webブラウザ、ネットワークを介した情報の参照
自分のPCを使用する、など

ネットワークの遮断について

- 試験開始5分後に演習室外へのネットワーク接続を切断する
- 試験開始60分後にネットワーク接続を戻す
- それ以降は、GitHubへの提出のためのコマンドに限り
ネットワーク利用が可能（それ以外は不正行為とする）

講義資料のダウンロードについて

演習室で作業する前に、以下のコマンドを入
れると講義資料のリポジトリがダウンロードされる

```
$ mygitclone-p2
```

ダウンロードが完了すると、
ホーム以下に作られた「lecture」フォルダの中に
資料などが保存されています

※本体をシャットダウンするまではPCに残ります