

プログラミングII

<http://bit.ly/Prog3i>

クラス (2)

～コンストラクタ, メソッドの多重定義～

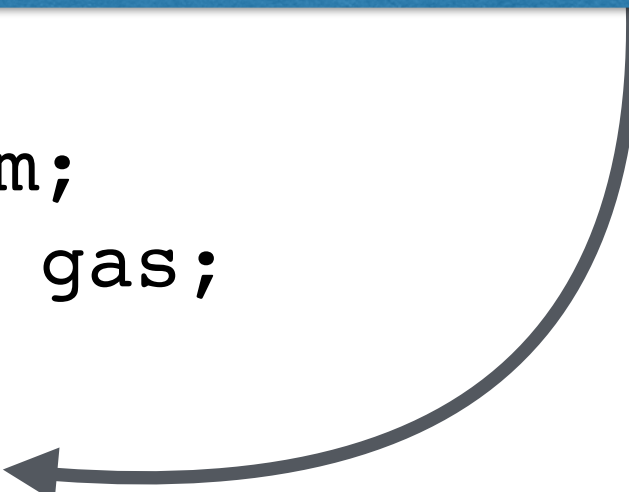
後期 第3週

2019/10/11

【Point 1】 **コンストラクタ**はインスタンス作成時に呼び出されるメソッド

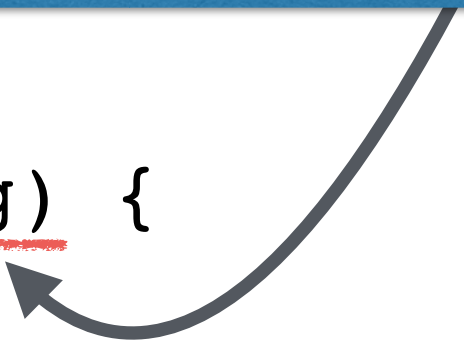
- ・ クラス名と同じ
- ・ 戻り値がない

```
1: class Car {  
2:     private int num;  
3:     private double gas;  
4:  
5:     public Car() {  
6:         num = 0;  
7:         gas = 0.0;  
8:         System.out.println("車を作成しました。");  
9:     }
```



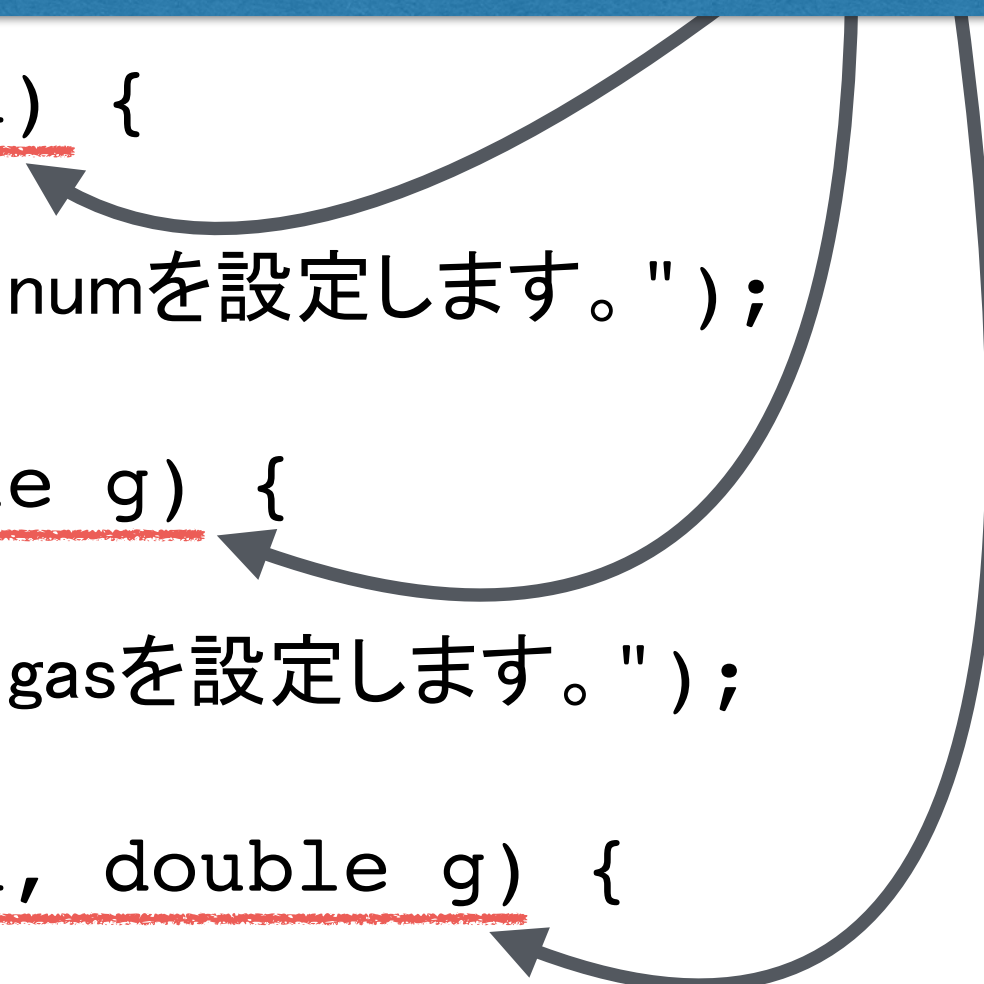
【Point 4】コンストラクタもオーバーロード
することができる

```
10: public Car(int n, double g) {  
11:     num = n;  
12:     gas = g;  
13:     System.out.println("車を作成しました。(引数付き)");  
14: }  
15: public void show() {  
16:     System.out.println("num: " + num);  
17:     System.out.println("gas: " + gas);  
18: }
```

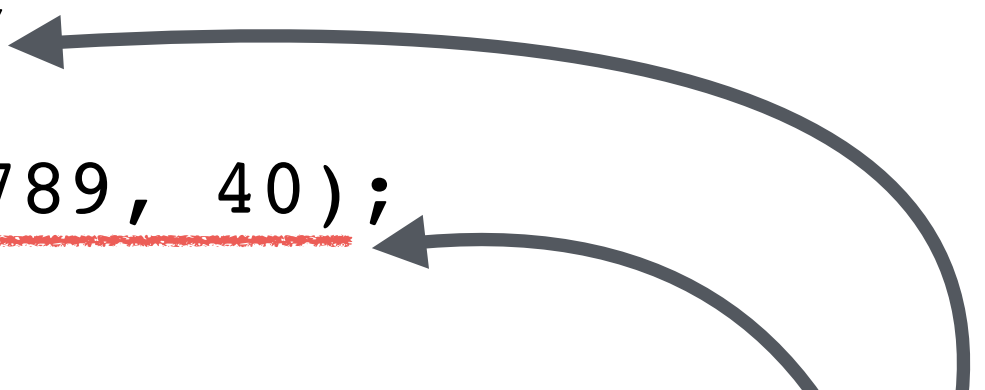


【Point 3】これらは全て同じ名前のメソッドだが、
「メソッドの引数の型、個数（メソッドのシグネ
チャ）が異なる」ため複数定義可能（「メソッドの
オーバーロード」と呼ぶ）

```
19: public void setCar(int n) {  
20:     num = n;  
21:     System.out.println("numを設定します。");  
22: }  
23: public void setCar(double g) {  
24:     gas = g;  
25:     System.out.println("gasを設定します。");  
26: }  
27: public void setCar(int n, double g) {  
28:     num = n;  
29:     gas = g;  
30:     System.out.println("numとgasを両方設定します。");  
31: }  
32: }  
33:
```

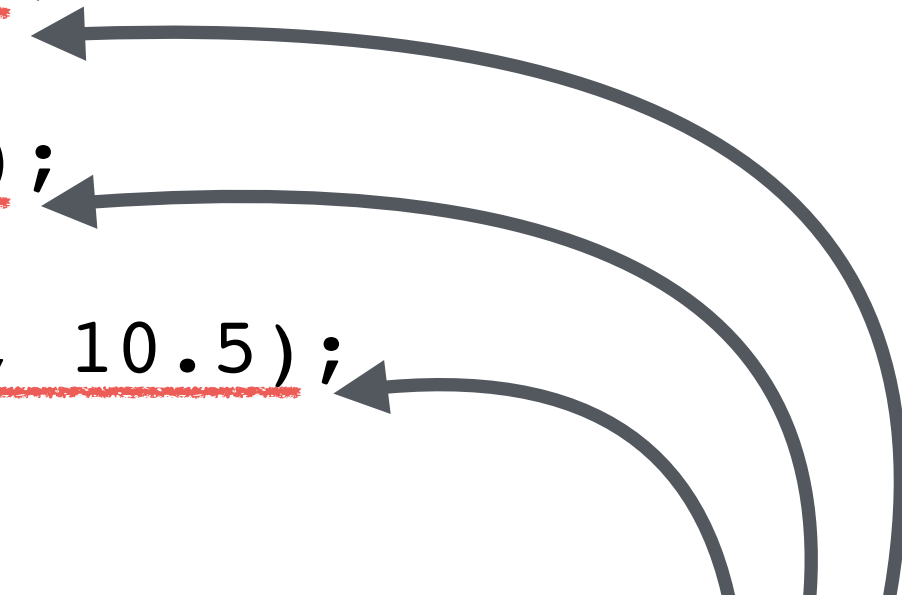


```
34: class Pd3car1 {
35:     public static void main(String[] args) {
36:         Car car1, car2;
37:         car1 = new Car();
38:         car1.show();
39:         car2 = new Car(6789, 40);
40:         car2.show();
41:
```

Two curved arrows originate from the blue text box at the bottom right. One arrow points to the 'new' keyword in line 37, and the other points to the 'new' keyword in line 39. Both lines of code are underlined in red.

【Point 2】 **new**を使ってインスタンスを作成する時に、**対応したコンストラクタ**が呼び出される


```
42:      car1.setCar(1234);
43:      car1.show();
44:      car2.setCar(30.5);
45:      car2.show();
46:      car1.setCar(4567, 10.5);
47:      car1.show();
48:  }
49: }
```



【Point 5】 オーバーロードされたメソッドを呼び出すと、実引数の型と個数に対応したメソッドが自動的に選択され実行される

「1つの（名前の）メソッドで、複数の処理を使い分けられる」ことを多態性（ポリモフィズム）と呼ぶ

参考Webサイト

▶ コンストラクタについて

<https://www.javadrive.jp/start/constructor/>

▶ オーバーロードについて

<https://www.javadrive.jp/start/method/index8.html>

【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3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 week203
\$ cd week203

【練習3-1】

サンプルプログラム「2_03_Car.java」を
コンパイルして、実行結果を確認しましょう。

【課題3-1】

サンプルプログラムにあるメソッドsetCarとコンストラクタを参考に、クラスCarに、次のようなメソッドsetCarとコンストラクタを追加して、オーバーロードの動作を確認してください。

```
public void setCar()  
    //フィールドnumに2222、gasに45を代入する  
  
public Car(double g)  
    //フィールドnumに1111を代入する  
    //仮引数gの値をフィールドgasに代入する、  
    //ただし、gが負の場合は正の値に変えて代入する
```

【課題3-1】

このクラスはファイル「2_03_Main.java」に含まれている

[mainを持ったクラス]

```
class Pd3car2 {  
    public static void main(String[] args) {  
        //コンストラクタの動作確認  
  
        Car car1, car2;  
        car1 = new Car(10.5);  
        car1.show();  
        car2 = new Car(-15.0);  
        car2.show();  
        //引数なしsetCarの動作確認  
  
        car1.setCar();  
        car1.show();  
    }  
}
```

【課題3-1】

[実行結果]

num: 1111

gas: 10.5 (←コンストラクタに正の値を指定した結果)

num: 1111

gas: 15.0 (←コンストラクタに負の値を指定した結果)

num: 2222 (←引数なしでsetCarを呼び出した結果)

gas: 45.0

【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3-1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))

【課題3-2】

課題2-2（第2週）で作成したクラスCalcに以下のコンストラクタを追加して、オーバーロードの動作を確認してください。

```
public Calc()  
    //フィールドxが5、フィールドyが7となるインスタンスを生成する  
  
public Calc(int num1, int num2)  
    //フィールドxがnum1、yがnum2となるインスタンスを生成する
```

【課題3-2】

このクラスはファイル「2_03_Main.java」に含まれている

```
class Pd03calc {  
    public static void main(String[] args) {  
        Calc c3, c4;  
        c3 = new Calc();  
        c3.show();  
        c4 = new Calc(100, 200);  
        c4.show();  
    }  
}
```

[実行結果]

```
x: 5, y: 7  
x: 100, y: 200
```

【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3-2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3-3】

課題3-2で作成したクラスCalcに以下のメソッドを追加して、オーバーロードの動作を確認してください。

```
public void setX()  
    //フィールドxに0を代入する
```

```
public void setY()  
    //フィールドyに0を代入する
```

【課題3-3】

課題3-2のメソッドmainの続きに以下を追加する

```
c3.setX();  
c4.setY();  
c3.show();  
c4.show();
```

[実行結果]

```
x: 0, y: 7  
x: 100, y: 0
```

【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3-3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3-4】

次のような「人」を表すクラスPersonを作り、mainでこのクラスの動作を確認してください。

- ☐ フィールドとして、String型のname（名前）とint型のage（年齢）を持つ。
- ☐ コンストラクタ（引数付き）の処理は、「引数で名前と年齢を指定して、それぞれフィールドnameとageに代入する」となる。
- ☐ コンストラクタ（引数なし）をオーバーロードする。この処理は、「フィールドnameには“anonymous”、ageには20を代入する」となる。
- ☐ 次のようなメソッドspeakを持つ

```
public void speak()  
    // 「My name is “自分の名前” . I am “自分の年齢” years old. 」  
    // のように自分の情報を出力する
```

【課題3-4】

このクラスはファイル「2_03_Main.java」に含まれている

```
class Pd03person {  
    public static void main(String[] args) {  
        Person p1, p2;  
        p1 = new Person("Steve", 56);  
        p2 = new Person();  
        p1.speak(); p2.speak();  
    }  
}
```

[実行結果]

My name is Steve. I am 56 years old.

My name is anonymous. I am 20 years old.

【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3-4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