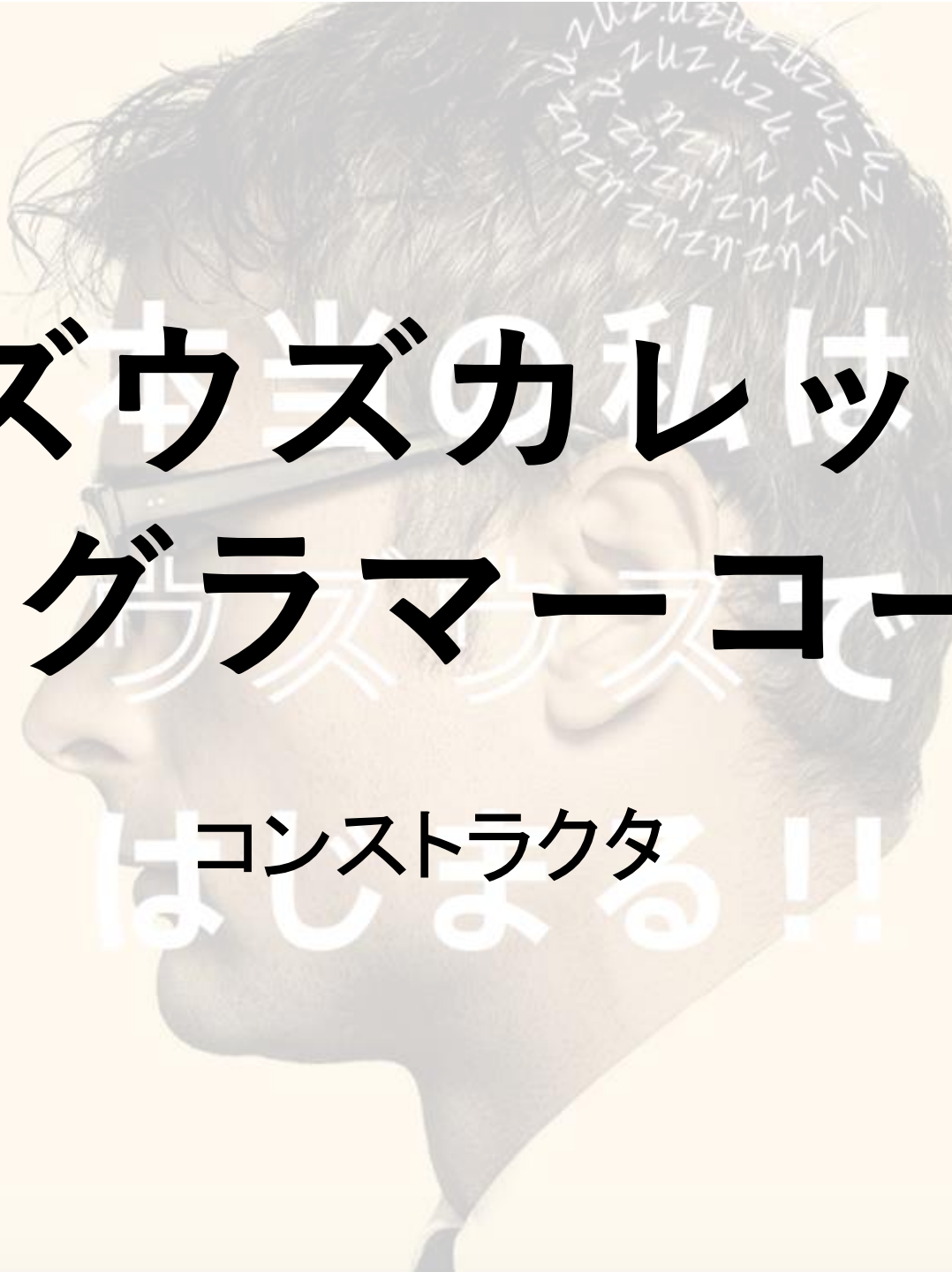
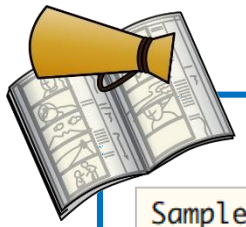


ウズウズカレッジ
プログラマーコース

コンストラクタ

～インスタンスは1つのクラスから複数生成できる～

復習



Sample2_01_1_driveクラス

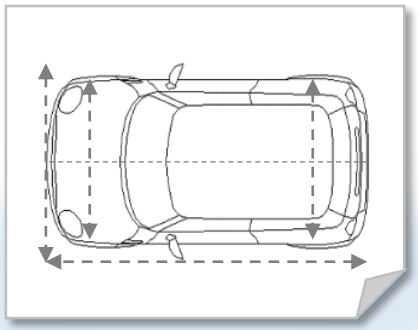
```
Sample2_01_1_car mocoCar  = new Sample2_01_1_car();
Sample2_01_1_car chocoCar = new Sample2_01_1_car();

mocoCar.accelerator();
```

～フィールドの初期値～

機能
- 加速する
- 減速する
- ライトが光る
- クラクションが鳴る

属性
- 車種名　：クーパー
- オーナー　：モコ
- 塗装色　：赤
- 現在のスピード　：100



クラス

インスタンス化

オーナー

インスタンス名：mocoCar

機能
- 加速する
- 減速する
- ライトが光る
- クラクションが鳴る

属性
- 車種名　：クーパー
- オーナー　：モコ
- 塗装色　：赤
- 現在のスピード　：100

インスタンス化

オーナー

インスタンス名：chocoCar

機能
- 加速する
- 減速する
- ライトが光る
- クラクションが鳴る

属性
- 車種名　：クーパー
- オーナー　：モコ
- 塗装色　：赤
- 現在のスピード　：100

ふふふ…
インスタンス化されるたびに
ぼくの車が増えていく…！

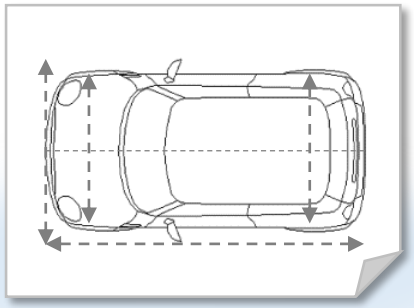
～属性のない実体？？～

機能

・加速する
・減速する
・ライトが光る
・クラクションが鳴る

属性

・車種名　　：？
・オーナー　　：？
・塗装色　　：？
・現在のスピード　：？



Sample2_02_1_carクラス

インスタンス化

？

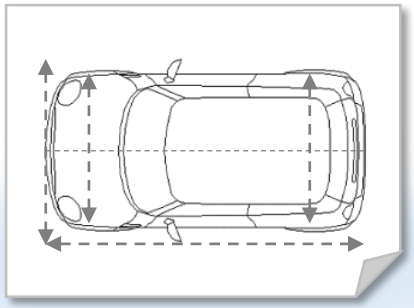
～属性のない実体？？～

機能

・加速する
・減速する
・ライトが光る
・クラクションが鳴る

属性

・車種名　　：？
・オーナー　　：？
・塗装色　　：？
・現在のスピード　：？



Sample2_02_1_carクラス

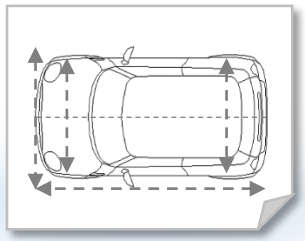
インスタンス化

車という概念

～フィールドの初期値～

機能
・加速する
・減速する
・ライトが光る
・クラクションが鳴る

属性
・車種名　：？
・オーナー　：？
・塗装色　：？
・現在のスピード　：？



車の共通機能&共通情報（属性）
を定義したクラス

インスタンス化

インスタンス化

車という概念

インスタンス名：car1

機能
・加速する
・減速する
・ライトが光る
・クラクションが鳴る

属性
・車種名　：？
・オーナー　：？
・塗装色　：？
・現在のスピード　：？

インスタンス化後に
フィールドに
具体的な値を代入

オーナー

属性
・車種名　：クーパー
・オーナー　：モコ
・塗装色　：赤
・現在のスピード　：100

車という概念

インスタンス名：car2

機能
・加速する
・減速する
・ライトが光る
・クラクションが鳴る

属性
・車種名　：？
・オーナー　：？
・塗装色　：？
・現在のスピード　：？

インスタンス化後に
フィールドに
具体的な値を代入

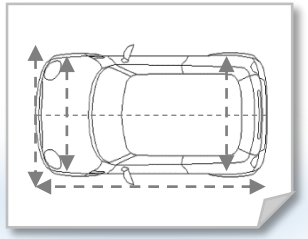
オーナー

属性
・車種名　：トラック
・オーナー　：ポチ
・塗装色　：白
・現在のスピード　：60

～コンストラクタ～

機能
・加速する
・減速する
・ライトが光る
・クラクションが鳴る

属性
・車種名　：？
・オーナー　：？
・塗装色　：？
・現在のスピード　：？



車の共通機能&共通情報（属性）
を定義したクラス

インスタンス化

車という概念

インスタンス名：car1

機能
・加速する
・減速する
・ライトが光る
・クラクションが鳴る

属性
・車種名　：？
・オーナー　：？
・塗装色　：？
・現在のスピード　：？

インスタンス化後に
フィールドに
具体的な値を代入

属性
・車種名　：クーパー
・オーナー　：モコ
・塗装色　：赤
・現在のスピード　：100

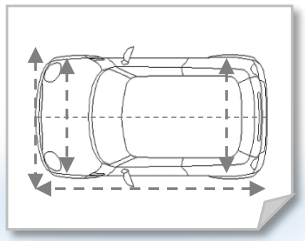
実体と言える？

属性あっての実体！

～コンストラクタ～

機能
・加速する
・減速する
・ライトが光る
・クラクションが鳴る

属性
・車種名　：？
・オーナー　：？
・塗装色　：？
・現在のスピード　：？



車の共通機能&共通情報（属性）
を定義したクラス

理想のインスタンス化

車という概念

インスタンス名：car1

機能
・加速する
・減速する
・ライトが光る
・クラクションが鳴る

属性
・車種名　：？
・オーナー　：？
・塗装色　：？
・現在のスピード　：？

インスタンス化後に
フィールドに
具体的な値を代入

属性
・車種名　：クーパー
・オーナー　：モコ
・塗装色　：赤
・現在のスピード　：100

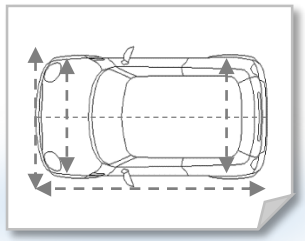
まとめてやりたい！

コンストラクタが必要

～コンストラクタ～

機能
・加速する
・減速する
・ライトが光る
・クラクションが鳴る

属性
・車種名　：？
・オーナー　：？
・塗装色　：？
・現在のスピード　：？



車の共通機能&共通情報（属性）
を定義したクラス

理想のインスタンス化

車という概念

インスタンス名：car1

機能
・加速する
・減速する
・ライトが光る
・クラクションが鳴る

属性
・車種名　：？
・オーナー　：？
・塗装色　：？
・現在のスピード　：？

インスタンス化後に
フィールドに
具体的な値を代入

属性
・車種名　：クーパー
・オーナー　：モコ
・塗装色　：赤
・現在のスピード　：100

コンストラクタ

インスタンス化されると同時に自動で動き出す特殊なメソッド。
フィールドに実体を定義するうえで必要な初期値を代入する役割を担う。

▼Sample2_02_1_driveクラス

```
class Sample2_02_1_drive {
    public static void main (String[] args) {

        //シナリオ①：車を場に登場させる（Sample2_02_1_carクラスのインスタンス化）
        System.out.println("▼シナリオ①：車を場に登場させる------------------") ;

        Sample2_02_1_car chocoCar = new Sample2_02_1_car();
        Sample2_02_1_car mocoCar  = new Sample2_02_1_car( "クーペ"   , "モコ" , "RED"   , 100 , true );
        Sample2_02_1_car pochiCar = new Sample2_02_1_car( "トラック" , "ポチ" , "WHITE" , 80  , true );

    }
```

▼Sample2_02_1_carクラス

```
//コンストラクタ①（引数なし）
Sample2_02_1_car(){
    System.out.println("□□▼コンストラクタ①（引数なし）------------------------") ;
    carModel = "未登録" ;  //車種名
    owner    = "未登録" ;  //オーナー
    color    = "未登録" ;  //塗装色
    speed    = 0        ;  //現在の速度
    right    = false    ;  //ライト（true:点灯/false:消灯）
    System.out.println("□□▲----------------------------------------------------") ;
}

//コンストラクタ②（引数あり）
Sample2_02_1_car(String cm , String on , String cl , int sp , boolean rt ){
    System.out.println("□□▼コンストラクタ②（引数あり）------------------------") ;
    carModel = cm ;        //車種名
    owner    = on ;        //オーナー
    color    = cl ;        //塗装色
    speed    = sp ;        //現在の速度
    right    = rt ;        //ライト（true:点灯/false:消灯）
    System.out.println("□□▲----------------------------------------------------") ;
}
```

≪コンストラクタとインスタンス化≫

□属性に具体的な値がないと実体として扱うことができないため、
インスタンス化が起こった際に自動でフィールドを初期化する特殊なメソッドを定義することが可能であり、これをコンストラクタと言います。

□コンストラクタには以下のルール・特徴があります。

- **コンストラクタの定義において戻り値の情報は何も記述しない**
  ※voidすら不要
- **コンストラクタ名はクラス名と同じ**
  ※メソッド名のルールを外れ、大文字からの命名となります
- **インスタンス生成時、必ず最初に呼び出される**

□実はインスタンス化の右辺はコンストラクタを表しており、右辺の( )内にフィールドの初期値として設定したい値を入れておくと、コンストラクタに値渡しされてフィールドの初期化が行われます。

クラス名 インスタンス名 = new クラス名(引数) ;

※コンストラクタを表しています

□コンストラクタはメソッドの一種なので、様々な引数パターンに応じてオーバーロードして定義することが可能です。

フィールドの参照では
thisを使おう！

コンストラクタの呼び出しは
コンストラクタの先頭で！

```
class Sample2_02_2_car {
    //---フィールド（クラス直下で定義された変数）----------------
    String  carModel ;  //車種名
    String  owner    ;  //オーナー
    String  color    ;  //塗装色
    int     speed    ;  //現在の速度
    boolean right    ;  //ライト（true:点灯/false:消灯）
    //---コンストラクタ-------------------------------------------------
    //コンストラクタ①（引数なし）
    Sample2_02_2_car(){
        System.out.println("□□▼コンストラクタ①（引数なし）-------------------------") ;
        carModel = "未登録" ;  //車種名
        owner    = "未登録" ;  //オーナー
        color    = "未登録" ;  //塗装色
        speed    = 0        ;  //現在の速度
        right    = false    ;  //ライト（true:点灯/false:消灯）
        System.out.println("□□▲-------------------------------------------------") ;
    }
    //コンストラクタ②（引数あり）
    Sample2_02_2_car(String carModel , String owner , String color , int speed , boolean right ){
        this();                  //コンストラクタ①（引数なし）の起動
        System.out.println("□□▼コンストラクタ②（引数あり）-------------------------") ;
        this.carModel = carModel;  //車種名
        owner    = owner ;         //オーナー
        color    = color ;         //塗装色
        speed    = speed ;         //現在の速度
        right    = right ;         //ライト（true:点灯/false:消灯）
        System.out.println("□□▲-------------------------------------------------") ;
    }
```

≪this/コンストラクタの呼び出し≫

□「**this**」は「**このインスタンス**」という意味を表します。
具体的には以下の意味を持ちます。

this()　　：　コンストラクタ
this.変数名　：　フィールド変数

□**コンストラクタからコンストラクタを呼ぶ際は、呼び出し元の
コンストラクタの先頭で呼ばなければならない**点にご注意ください。
これはコンストラクタがコンストラクタを呼んでいる構造になっている場合、一番深いコンストラクタから順に実行されねばならないという仕様によるものです。

□特に**メソッド内のローカル変数とフィールド変数が同じ名称の
場合、ローカル変数が優遇される**という仕様から、フィールド変数を明示する必要があるときは必ずthisをつける癖をつけましょう。

～コンストラクタは一番深いものから動く！～

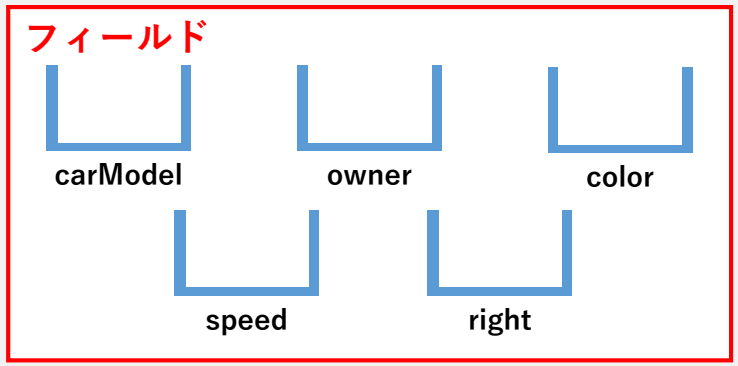
▼Sample2_02_2_driveクラス

```
Sample2_02_2_car mocoCar  = new Sample2_02_2_car( "クーペ"   , "モコ" , "RED"   , 100 , true );
```

▼Sample2_02_2_carクラス

```
//コンストラクタ②（引数あり）
Sample2_02_2_car(String carModel , String owner , String color , int speed , boolean right ){
    this();                    //コンストラクタ①（引数なし）の起動
    System.out.println("□□▼コンストラクタ②（引数あり）------------------------") ;
    this.carModel = carModel ;  //車種名
    owner         = owner    ;  //オーナー
    color         = color    ;  //塗装色
    speed         = speed    ;  //現在の速度
    right         = right    ;  //ライト（true:点灯/false:消灯）
    System.out.println("□□▲------------------------------------------------------") ;
}
```

```
//コンストラクタ①（引数なし）
Sample2_02_2_car(){
    System.out.println("□□▼コンストラクタ①（引数なし）------------------------") ;
    carModel = "未登録" ;  //車種名
    owner    = "未登録" ;  //オーナー
    color    = "未登録" ;  //塗装色
    speed    = 0        ;  //現在の速度
    right    = false    ;  //ライト（true:点灯/false:消灯）
    System.out.println("□□▲------------------------------------------------------") ;
}
```



～コンストラクタは一番深いものから動く！～

▼Sample2_02_2_driveクラス

```
new Sample2_02_2_car( "クーペ"   , "モコ" , "RED"   , 100 , true )
```

▼Sample2_02_2_carクラス

```
//コンストラクタ②（引数あり）
Sample2_02_2_car(String carModel , String owner , String color , int speed , boolean right ){
    this();                    //コンストラクタ①（引数なし）の起動
    System.out.println("▼コンストラクタ②（引数あり）------------------------") ;
    this.carModel = carModel ;  //車種名
    owner         = owner    ;  //オーナー
    color         = color    ;  //塗装色
    speed         = speed    ;  //現在の速度
    right         = right    ;  //ライト（true:点灯/false:消灯）
    System.out.println("▲----------------------------------------------------") ;
}
```

```
//コンストラクタ①（引数なし）
Sample2_02_2_car(){
    System.out.println("▼コンストラクタ①（引数なし）------------------
    carModel = "未登録" ;  //車種名
    owner    = "未登録" ;  //オーナー
    color    = "未登録" ;  //塗装色
    speed    = 0        ;  //現在の速度
    right    = false    ;  //ライト（true:点灯/false:消灯）
    System.out.println("▲----------------------------------------------------") ;
}
```

具体的な処理

～コンストラクタは一番深いものから動く！～

▼Sample2_02_2_driveクラス

```
new Sample2_02_2_car( "クーペ"   , "モコ" , "RED"   , 100 , true )
```

▼Sample2_02_2_carクラス

```
//コンストラクタ②（引数あり）
Sample2_02_2_car(String carModel , String owner , String color , int speed , boolean right ){
    this();                    //コンストラクタ①（引数なし）の起動
    System.out.println("▼コンストラクタ②（引数あり）----------");
    this.carModel = carModel ;  //車種名
    owner         = owner    ;  //オーナー
    color         = color    ;  //塗装色
    speed         = speed    ;  //現在の速度
    right         = right    ;  //ライト（true:点灯/false:消灯）
    System.out.println("▲----------") ;
}
```

具体的な処理

```
//コンストラクタ①（引数なし）
Sample2_02_2_car(){
    System.out.println("▼コンストラクタ①（引数なし）----------");
    carModel = "未登録" ;  //車種名
    owner    = "未登録" ;  //オーナー
    color    = "未登録" ;  //塗装色
    speed    = 0        ;  //現在の速度
    right    = false    ;  //ライト（true:点灯/false:消灯）
    System.out.println("▲----------") ;
}
```

具体的な処理

～コンストラクタは一番深いものから動く！～

▼Sample2_02_2_driveクラス

```
new Sample2_02_2_car( "クーペ"   , "モコ" , "RED"   , 100 , true )
```

コンストラクタの呼び出しは
コンストラクタの先頭で！

▼Sample2_02_2_carクラス

```
//コンストラクタ②（引数あり）
Sample2_02_2_car(String carModel , String owner , String color , int speed , boolean right ){
    System.out.println("□□▼コンストラクタ②（引数あり）----------");
    this();                    //コンストラクタ①（引数なし）の起動
    this.carModel = carModel ;  //車種名
    owner         = owner    ;  //オーナー
    color         = color    ;  //塗装色
    speed         = speed    ;  //現在の速度
    right         = right    ;  //ライト（true:点灯/false:消灯）
    System.out.println("□□▲----------------------------------------------------") ;
}
```

具体的な処理

```
//コンストラクタ①（引数なし）
Sample2_02_2_car(){
    System.out.println("□□▼コンストラクタ①（引数なし）----------");
    carModel = "未登録" ;  //車種名
    owner    = "未登録" ;  //オーナー
    color    = "未登録" ;  //塗装色
    speed    = 0        ;  //現在の速度
    right    = false    ;  //ライト（true:点灯/false:消灯）
    System.out.println("□□▲----------------------------------------------------") ;
}
```

コンパイルエラー！

コンストラクタの呼び出し処理
よりも前に自身のコンストラクタの
具体的な処理を行うことは許されない

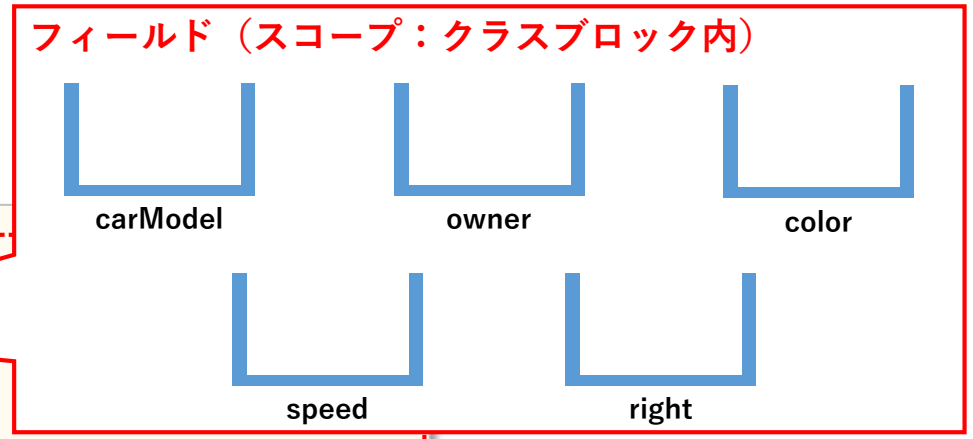
～フィールドの変数名とローカル変数名がかぶった場合～

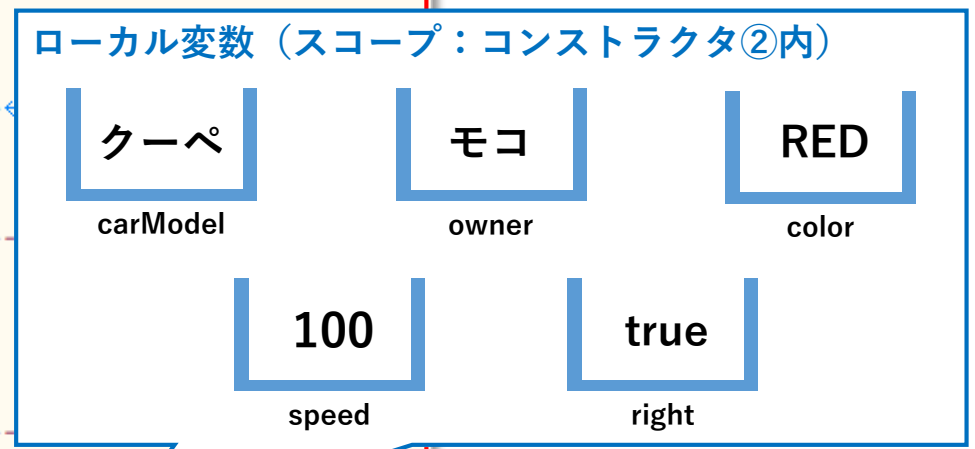
▼Sample2_02_2_carクラス

```
class Sample2_02_2_car {
    //---フィールド（クラス直下で定義された変数）----------
    String  carModel ;  //車種名
    String  owner    ;  //オーナー
    String  color    ;  //塗装色
    int     speed    ;  //現在の速度
    boolean right    ;  //ライト（true:点灯/false:消灯）

    //---コンストラクタ----------
    //コンストラクタ①（引数なし）
    Sample2_02_2_car(){
        System.out.println("□□▼コンストラクタ①（引数なし）----------");
        carModel = "未登録" ;  //車種名
        owner    = "未登録" ;  //オーナー
        color    = "未登録" ;  //塗装色
        speed    = 0        ;  //現在の速度
        right    = false    ;  //ライト（true:点灯/false:消灯）
        System.out.println("□□▲----------");
    }

    //コンストラクタ②（引数あり）
    Sample2_02_2_car(String carModel , String owner , String color , int speed , boolean right ){
        this();                 //コンストラクタ①（引数なし）の起動
        System.out.println("□□▼コンストラクタ②（引数あり）----------") ;
        this.carModel = carModel ;  //車種名
        owner         = owner    ;  //オーナー
        color         = color    ;  //塗装色
        speed         = speed    ;  //現在の速度
        right         = right    ;  //ライト（true:点灯/false:消灯）
        System.out.println("□□▲----------") ;
    }
```





～フィールドの変数名とローカル変数名がかぶった場合～

▼Sample2_02_2_carクラス

```
class Sample2_02_2_car {
    //---フィールド（クラス直下で定義された変数）----------
    String  carModel ;  //車種名
    String  owner    ;  //オーナー
    String  color    ;  //塗装色
    int     speed    ;  //現在の速度
    boolean right    ;  //ライト（true:点灯/false:消灯）

    //---コンストラクタ----------
    //コンストラクタ①（引数なし）
    Sample2_02_2_car(){
        System.out.println("□□▼コンストラクタ①（引数なし）----------");
        carModel = "未登録" ;  //車種名
        owner    = "未登録" ;  //オーナー
        color    = "未登録" ;  //塗装色
        speed    = 0        ;  //現在の速度
        right    = false    ;  //ライト（true:点灯/false:消灯）
        System.out.println("□□▲----------");
    }

    //コンストラクタ②（引数あり）
    Sample2_02_2_car(String carModel , String owner , String color , int speed , boolean right ){
        this();                 //コンストラクタ①（引数なし）の起動
        System.out.println("□□▼コンストラクタ②（引数あり）----------") ;
        this.carModel = carModel ;  //車種名
        owner         = owner    ;  //オーナー
        color         = color    ;  //塗装色
        speed         = speed    ;  //現在の速度
        right         = right    ;  //ライト（true:点灯/false:消灯）
        System.out.println("□□▲----------") ;
    }
```

フィールド（スコープ：クラスブロック内）

carModel	owner	color
未登録	未登録	未登録

speed	right
0	false

ローカル変数（スコープ：コンストラクタ②内）

carModel	owner	color
クーペ	モコ	RED

speed	right
100	true

～フィールドの変数名とローカル変数名がかぶった場合～

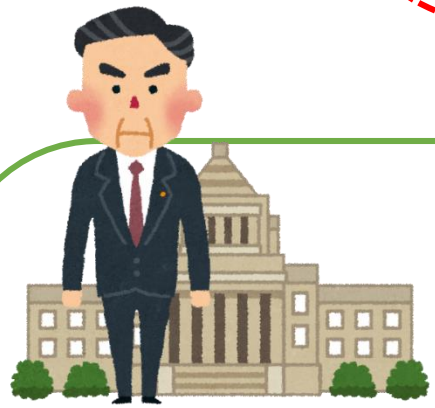
▼Sample2_02_2_carクラス

```
class Sample2_02_2_car {
    //---フィールド（クラス直下で定義された変数）----------------
    String  carModel ;  //車種名
    String  owner    ;  //オーナー
    String  color    ;  //塗装色
    int     speed    ;  //現在の速度
    boolean right    ;  //ライト（true:点灯/false:消灯）

    //---コンストラクタ-------------------------------------------------
    //コンストラクタ①（引数なし）
    Sample2_02_2_car(){
        System.out.println("□□▼コンストラクタ①（引数なし）------------------------") ;
        carModel = "未登録" ;  //車種名
        owner    = "未登録" ;  //オーナー
        color    = "未登録" ;  //塗装色
        speed    = 0        ;  //現在の速度
        right    = false    ;  //ライト（true:点灯/false:消灯）
        System.out.println("□□▲-------------------------------------------------") ;
    }

    //コンストラクタ②（引数あり）
    Sample2_02_2_car(String carModel , String owner , String color , int speed , boolean right ){
        this();                    //コンストラクタ①（引数なし）の起動
        System.out.println("□□▼コンストラクタ②（引数あり）------------------------") ;
        this.carModel = carModel ;  //車種名
        owner         = owner    ;  //オーナー
        color         = color    ;  //塗装色
        speed         = speed    ;  //現在の速度
        right         = right    ;  //ライト（true:点灯/false:消灯）
        System.out.println("□□▲-------------------------------------------------") ;
    }
```

同名の変数が存在する場合、
よりローカルなものが優先される。



アベさん

クラスブロック

メソッドブロック
（ローカル）

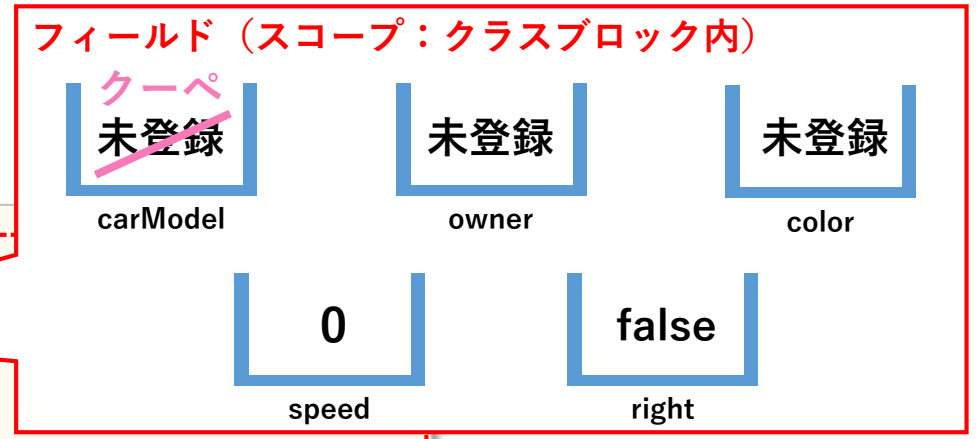
アベさん

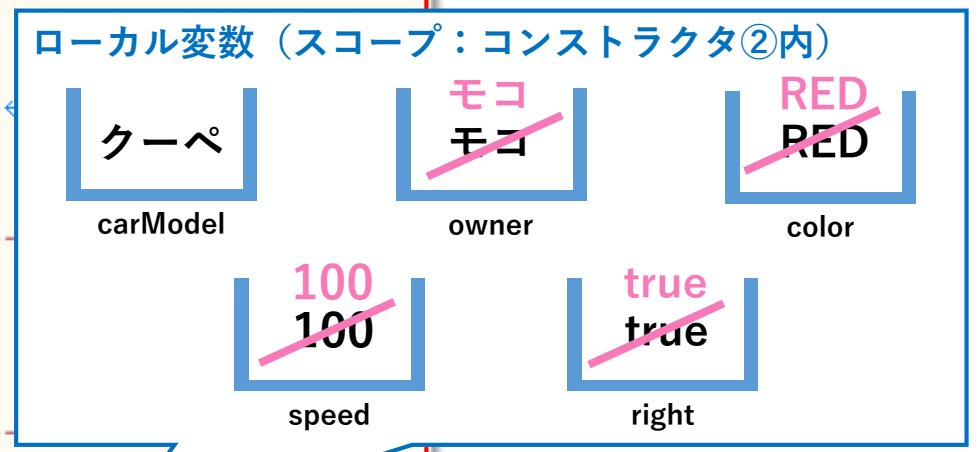
アベさん！

～フィールドの変数名とローカル変数名がかぶった場合～

▼Sample2_02_2_carクラス

```
class Sample2_02_2_car {
    
    //---フィールド（クラス直下で定義された変数）----------
    
    String  carModel ;  //車種名
    String  owner    ;  //オーナー
    String  color    ;  //塗装色
    int     speed    ;  //現在の速度
    boolean right    ;  //ライト（true:点灯/false:消灯）
    
    //---コンストラクタ----------------------------------------
    
    //コンストラクタ①（引数なし）
    Sample2_02_2_car(){
        System.out.println("□□▼コンストラクタ①（引数なし）----------
        carModel = "未登録" ;  //車種名
        owner    = "未登録" ;  //オーナー
        color    = "未登録" ;  //塗装色
        speed    = 0        ;  //現在の速度
        right    = false    ;  //ライト（true:点灯/false:消灯）
        System.out.println("□□▲----------------------------------
    }
    
    //コンストラクタ②（引数あり）
    Sample2_02_2_car(String carModel , String owner , String color , int speed , boolean right ){
        this();                 //コンストラクタ①（引数なし）の起動
        System.out.println("□□▼コンストラクタ②（引数あり）------------------------") ;
        this.carModel = carModel ;  //車種名
        owner         = owner    ;  //オーナー
        color         = color    ;  //塗装色
        speed         = speed    ;  //現在の速度
        right         = right    ;  //ライト（true:点灯/false:消灯）
        System.out.println("□□▲----------------------------------------------") ;
    }
```



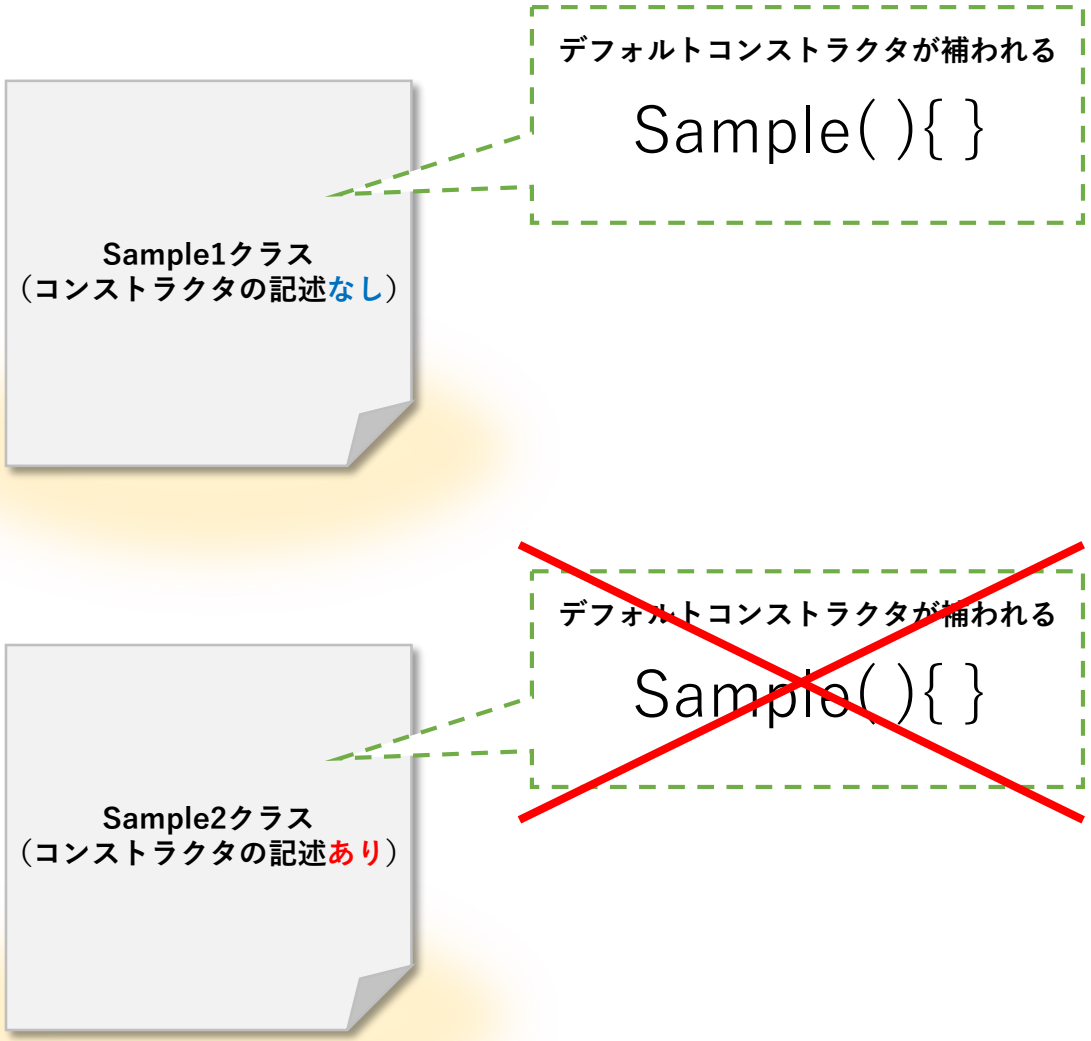


～フィールドの変数名とローカル変数名がかぶった場合～

フィールドと名前のかぶる変数を
極力定義しないように心がけましょう

```
//コンストラクタ②（引数あり）
Sample2_02_2_car(String cm , String on , String cl , int sp , boolean rt ){
    System.out.println("▼コンストラクタ②（引数あり）------------------------") ;
    this();                    //コンストラクタ①（引数なし）の起動
    this.carModel = cm    ;    //車種名
    this.owner    = on    ;    //オーナー
    this.color    = cl    ;    //塗装色
    this.speed    = sp    ;    //現在の速度
    this.right    = rt    ;    //ライト（true:点灯/false:消灯）
    System.out.println("▲------------------------------------------------") ;
}
```

フィールドの参照では
thisを使おう！

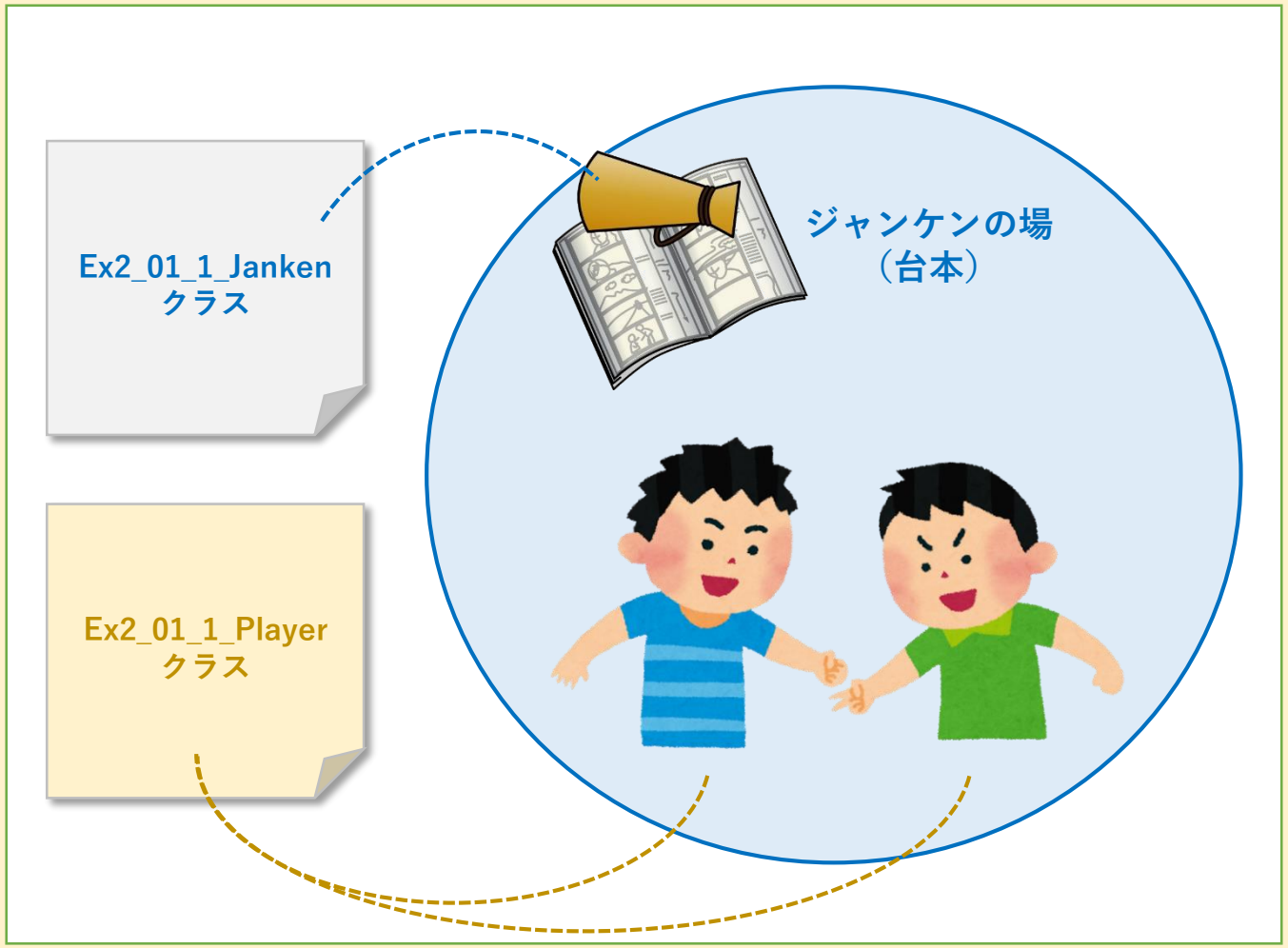


≪デフォルトコンストラクタ≫

□**コンストラクタはインスタンス化の際には必ず実行されます。**
もしもコンストラクタがソースコード内で定義されていない場合、**裏で引数なし/処理なしのコンストラクタが補われて実行され、これをデフォルトコンストラクタと言います。**

□デフォルトコンストラクタは**ソースコード内に1つでもコンストラクタが定義されていた場合、補われない**ことにご注意ください。

＜演習：Ex2_02_1＞
右の結果が出るジャンケンのプログラムを作成しましょう。




```
C:¥WorkSpace>java Ex2_02_1_Janken モコ ポチ
じゃんけん・・・ぽん！！！！！
モコさんの手：グー
ポチさんの手：グー
結果は・・・
あいこ！勝負つかず！

C:¥WorkSpace>java Ex2_02_1_Janken モコ ポチ
じゃんけん・・・ぽん！！！！！
モコさんの手：パー
ポチさんの手：チョキ
結果は・・・
ポチさんの勝利！

C:¥WorkSpace>java Ex2_02_1_Janken モコ ポチ
じゃんけん・・・ぽん！！！！！
モコさんの手：チョキ
ポチさんの手：パー
結果は・・・
モコさんの勝利！
```

＜演習：Ex2_02_1＞

▼Ex2_02_1_Playerクラス

・**フィールド ※初期化はしないこと！**
- name（String型）　　…プレイヤー名
- handStatus（String型）　…ジャンケンの手

・**コンストラクタ**

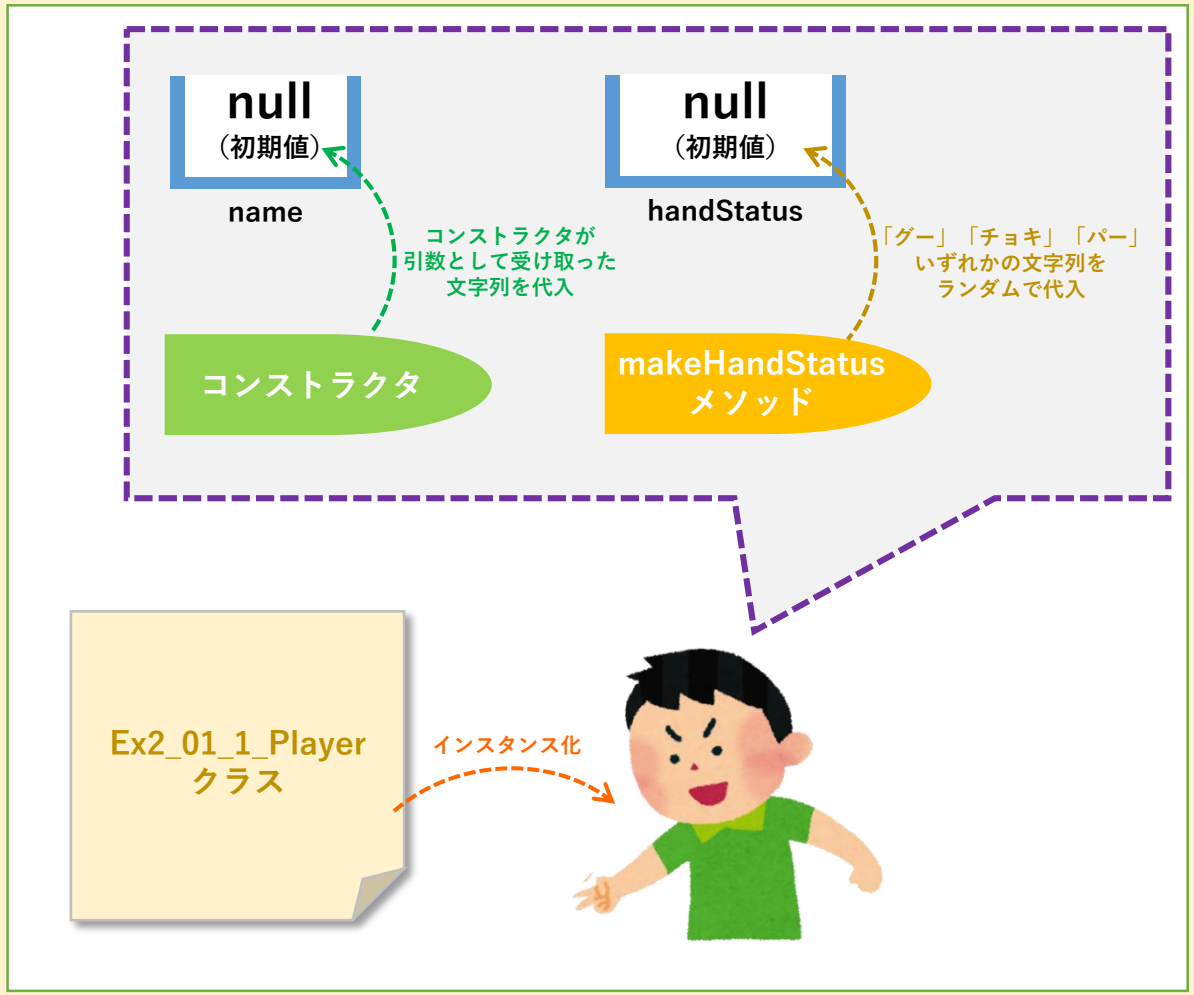
<引数>
String型の文字列を1つ受け取る。

<処理>
引数で受け取った文字列をnameに設定。

・**メソッド**
- makeHandStatusメソッド

<引数 / 戻り値>
なし

<処理>
handStatusに「グー」「チョキ」「パー」のいずれかを
ランダムに設定。



＜演習：Ex2_02_1＞

▼Ex2_02_1_Jankenクラス

・メソッド
- mainメソッド

<シナリオ①>
コマンドライン引数より2つの文字列を受け取る。
（ジャンケンを行う2名のプレイヤーの名前として利用）

<シナリオ②>
2名のプレイヤーを場に登場させる。（インスタンス化する）
インスタンス化の際はコマンドライン引数で受け取った文字列をそれぞれ
コンストラクタに渡す。

<シナリオ③>
それぞれのプレイヤーに手を握らせる。
※握らせる前に「じゃんけん・・・ぽん！！！！！」というメッセージを画面に表示。
※それぞれのプレイヤーがどんな手を出したか、それぞれ
「（nameの値） さんの手：（ handStatus の値））」の形で画面に表示。

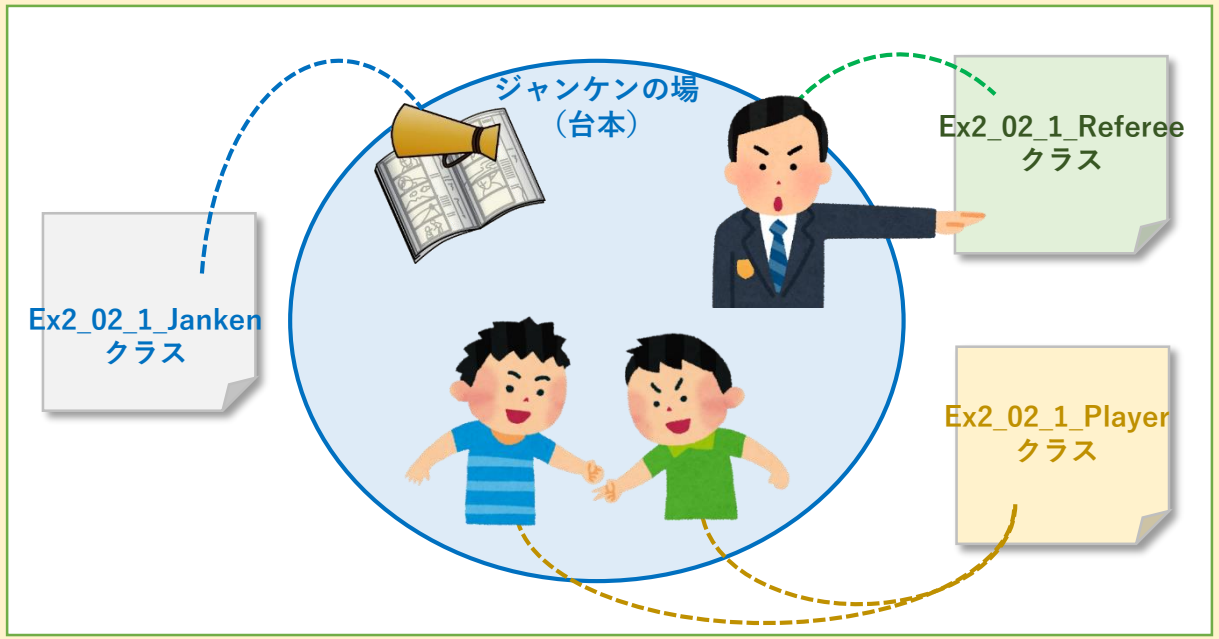
<シナリオ④>
勝敗の結果を画面に表示する。
※はじめに「結果は・・・」というメッセージを画面に表示。
※あいこだった場合は「あいこ！勝負つかず！」というメッセージを画面に表示。
※あいこでない場合は「（nameの値）さんの勝利！」というメッセージを画面に表示。

```
C:¥WorkSpace>java Ex2_02_1_Janken モコ ポチ
じゃんけん・・・ぽん！！！！！
モコさんの手：グー
ポチさんの手：グー
結果は・・・
あいこ！勝負つかず！

C:¥WorkSpace>java Ex2_02_1_Janken モコ ポチ
じゃんけん・・・ぽん！！！！！
モコさんの手：パー
ポチさんの手：チョキ
結果は・・・
ポチさんの勝利！

C:¥WorkSpace>java Ex2_02_1_Janken モコ ポチ
じゃんけん・・・ぽん！！！！！
モコさんの手：チョキ
ポチさんの手：パー
結果は・・・
モコさんの勝利！
```

＜演習：Ex2_02_2＞
Ex2_02_1に審判を追加し、右の結果が得られるようなプログラムを作成しましょう。




```
C:¥WorkSpace>java Ex2_02_2_Janken モコ ポチ 審判チョコ
審判チョコ「じゃんけん・・・ぽん！！！！！」
審判チョコ「モコさんの手はチョキでした！」
審判チョコ「ポチさんの手はチョキでした！」
審判チョコ「結果は・・・」
審判チョコ「あいこ！勝負つかず！」

C:¥WorkSpace>java Ex2_02_2_Janken モコ ポチ 審判チョコ
審判チョコ「じゃんけん・・・ぽん！！！！！」
審判チョコ「モコさんの手はチョキでした！」
審判チョコ「ポチさんの手はパーでした！」
審判チョコ「結果は・・・」
審判チョコ「モコさんの勝利！」

C:¥WorkSpace>java Ex2_02_2_Janken モコ ポチ 審判チョコ
審判チョコ「じゃんけん・・・ぽん！！！！！」
審判チョコ「モコさんの手はパーでした！」
審判チョコ「ポチさんの手はチョキでした！」
審判チョコ「結果は・・・」
審判チョコ「ポチさんの勝利！」
```

＜演習：Ex2_02_2＞
Ex2_02_1に審判を追加し、右の結果が得られるようなプログラムを作成しましょう。

▼Ex2_02_2_Playerクラス
基本的な内容はEx2_02_1_Playerクラスに同じ。

▼Ex2_02_2_Jankenクラス
- mainメソッド

<シナリオ①>
コマンドライン引数より3つの文字列を受け取る。
（ジャンケンを行う2名のプレイヤー及び審判の名前として利用）

<シナリオ②>
2名のプレイヤー及び審判を場に登場させる。（インスタンス化する）
インスタンス化の際はコマンドライン引数で受け取った文字列をそれぞれ
コンストラクタに渡す。

<シナリオ③>
それぞれのプレイヤーに手を握らせる。
※握らせる前に「じゃんけん・・・ぽん！！！！！」と審判が言う。
※それぞれのプレイヤーがどんな手を出したか、それぞれについて
「（nameの値） さんの手は（ handStatus の値）でした！」と審判が言う。

<シナリオ④>
審判が勝敗を判定し、結果を言う。
※はじめに「結果は・・・」と審判が言う。
※あいこだった場合は「あいこ！勝負つかず！」と審判が言う。
※あいこでない場合は「（nameの値）さんの勝利！」と審判が言う。

```
C:¥WorkSpace>java Ex2_02_2_Janken モコ ポチ 審判チョコ
審判チョコ「じゃんけん・・・ぽん！！！！！」
審判チョコ「モコさんの手はチョキでした！」
審判チョコ「ポチさんの手はチョキでした！」
審判チョコ「結果は・・・」
審判チョコ「あいこ！勝負つかず！」

C:¥WorkSpace>java Ex2_02_2_Janken モコ ポチ 審判チョコ
審判チョコ「じゃんけん・・・ぽん！！！！！」
審判チョコ「モコさんの手はチョキでした！」
審判チョコ「ポチさんの手はパーでした！」
審判チョコ「結果は・・・」
審判チョコ「モコさんの勝利！」

C:¥WorkSpace>java Ex2_02_2_Janken モコ ポチ 審判チョコ
審判チョコ「じゃんけん・・・ぽん！！！！！」
審判チョコ「モコさんの手はパーでした！」
審判チョコ「ポチさんの手はチョキでした！」
審判チョコ「結果は・・・」
審判チョコ「ポチさんの勝利！」
```

＜演習：Ex2_02_2＞
Ex2_02_1に審判を追加し、右の結果が得られるようなプログラムを作成しましょう。

▼Ex2_02_2_Refereeクラス

- **フィールド ※初期化はしないこと！**
  - name（String型）　…審判の名前
- **コンストラクタ**

  <引数>
  String型の文字列を1つ受け取る。

  <処理>
  引数で受け取った文字列をnameに設定。
- **メソッド**
  どんなメソッド（審判の機能）があればよいか、
  自分で考えて書いてみましょう！
  ポイントは「実際なら・・・」と想像することです！

実際のジャンケンを想像し、
うまく役割分担させた定義をしよう！