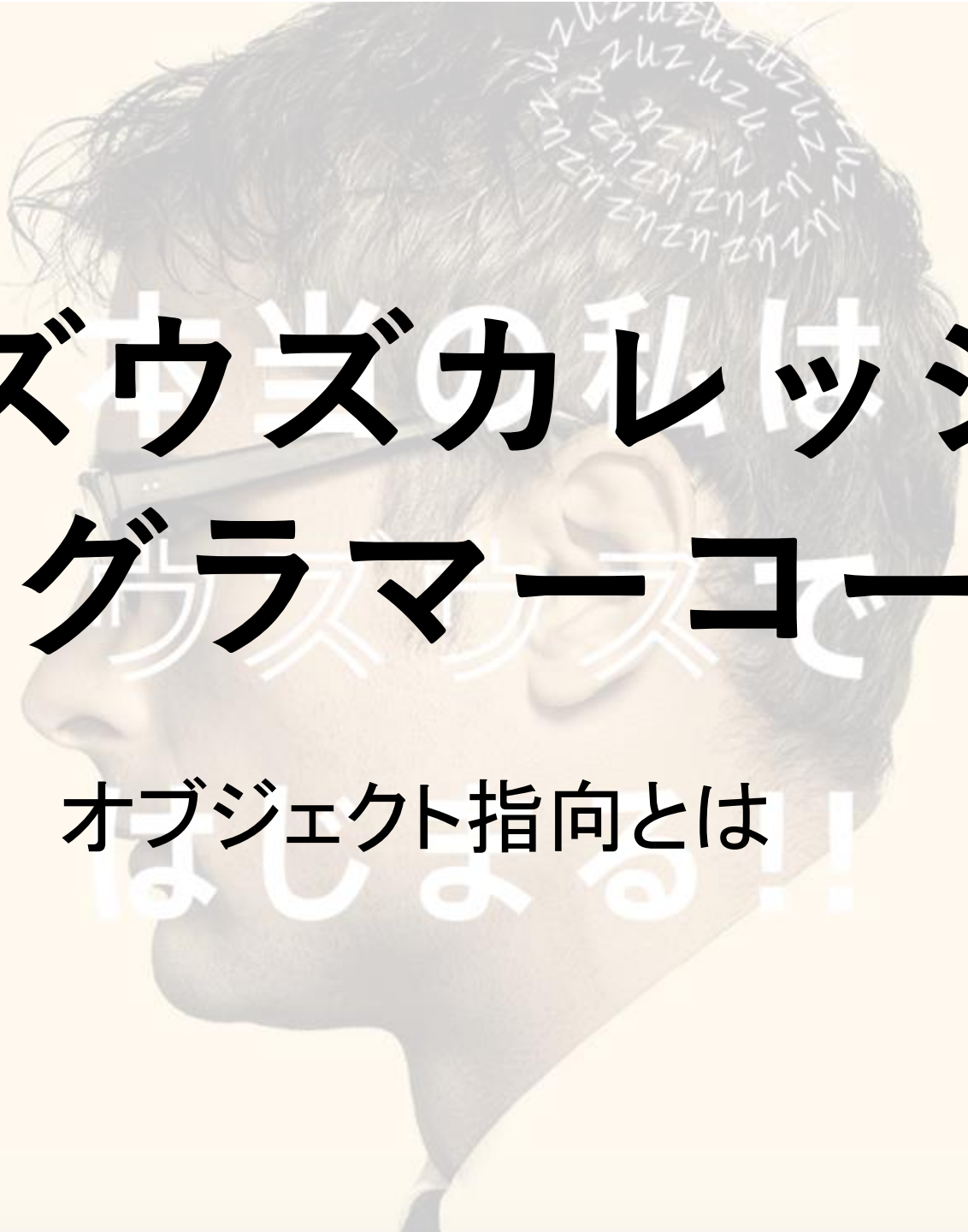


ウズウズカレッジ
プログラマーコース

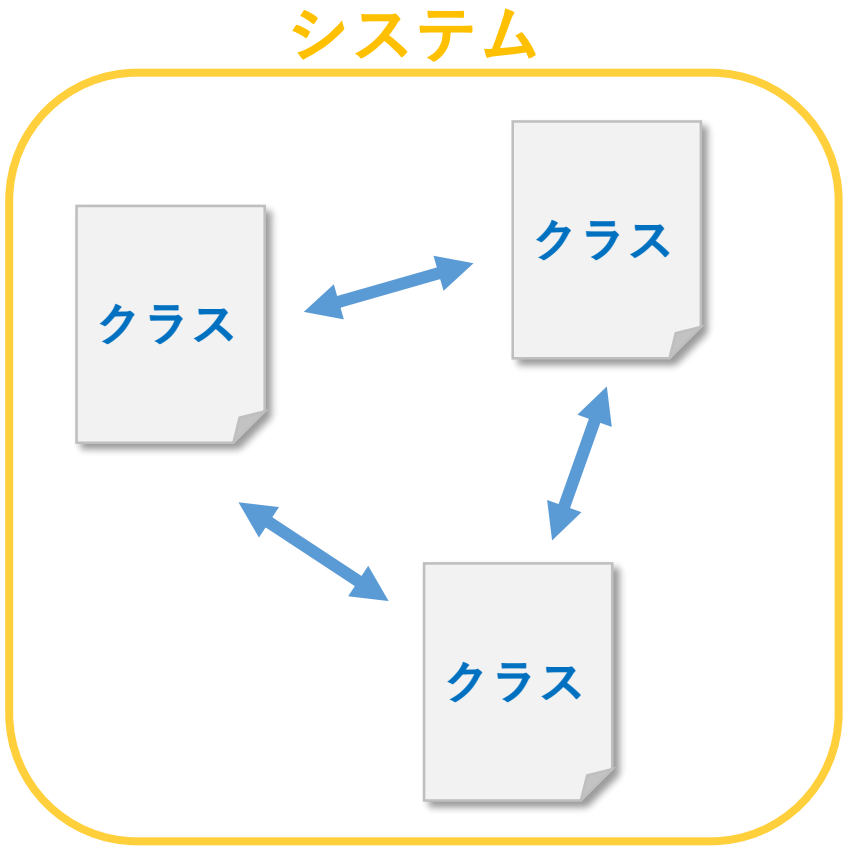
オブジェクト指向とは

～システムとは？～

システム【 system 】

システムとは、個々の要素が相互に影響しあいながら、全体として機能するまとまりや仕組みのこと。
ITの分野では、個々の電子部品や機器で構成され、全体として何らかの情報処理機能を持つ装置のことや、ハードウェアやソフトウェア、ネットワークなどの要素を組み合わせ、全体として何らかの機能を発揮するひとまとまりの仕組み(情報システム、ITシステム)のことを指す。

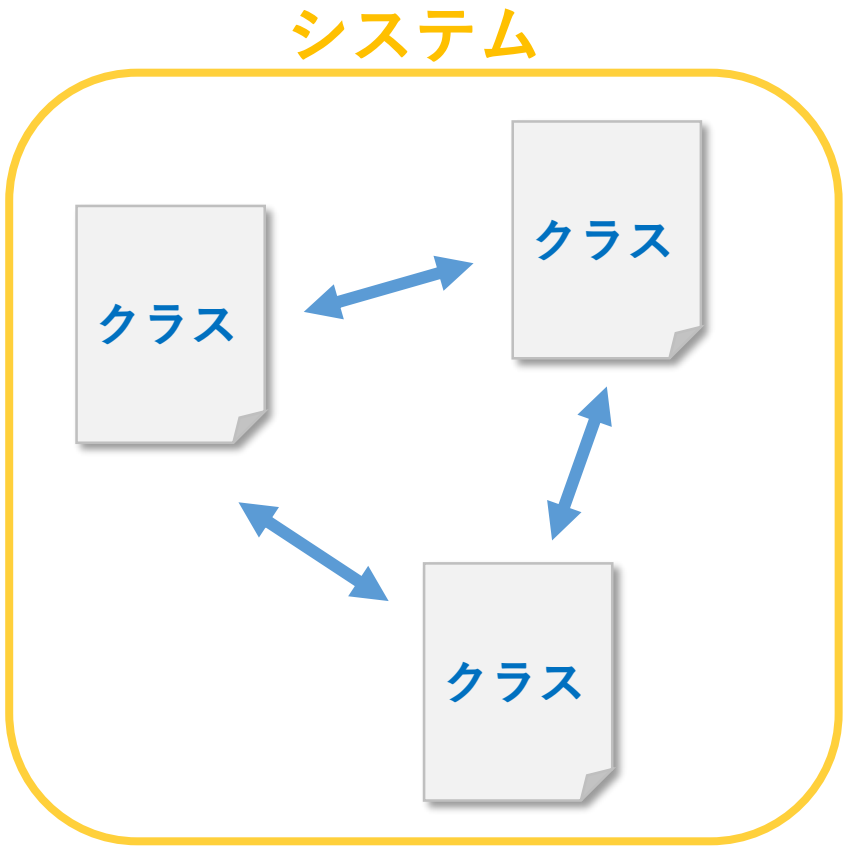
― IT用語辞典より抜粋



～どのようにクラス分けする？～

超長文
ソースコード

なぜ分ける？



～なぜクラスやメソッドに分けるんだろう？～

メリット

・クラスごとに役割分担して作れる！
　大規模システムだって複数人で一気に！

・短くまとまったソースコードは読みやすい
　&わかりやすい&テストが楽！

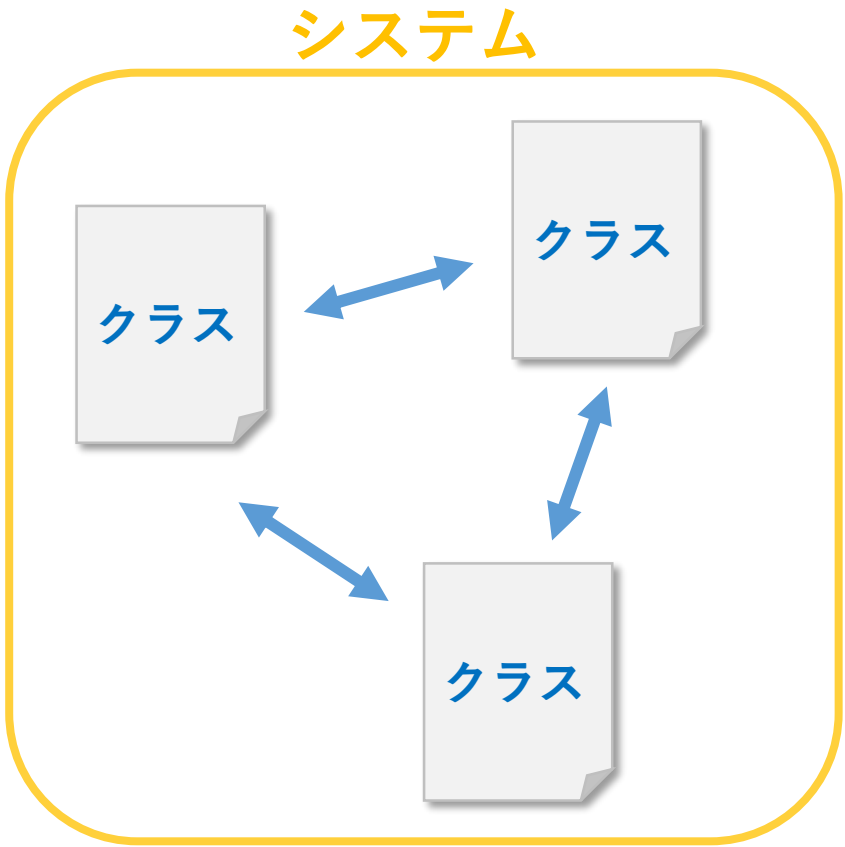
・一度書いたソースコードを様々なシーンで何度でも
　使いまわすことが可能！（再利用性）

・使うクラスだけをインスタンス化することでメモリ
　節約&処理スピードUP！

・どのプログラムがどこに書かれているか、誰がいつ
　どのような背景で書いたかなど、情報の管理や特定
　がしやすい！

・変更が必要なクラスやメソッドのみを修正すれば
　よい！（メンテナンス性）

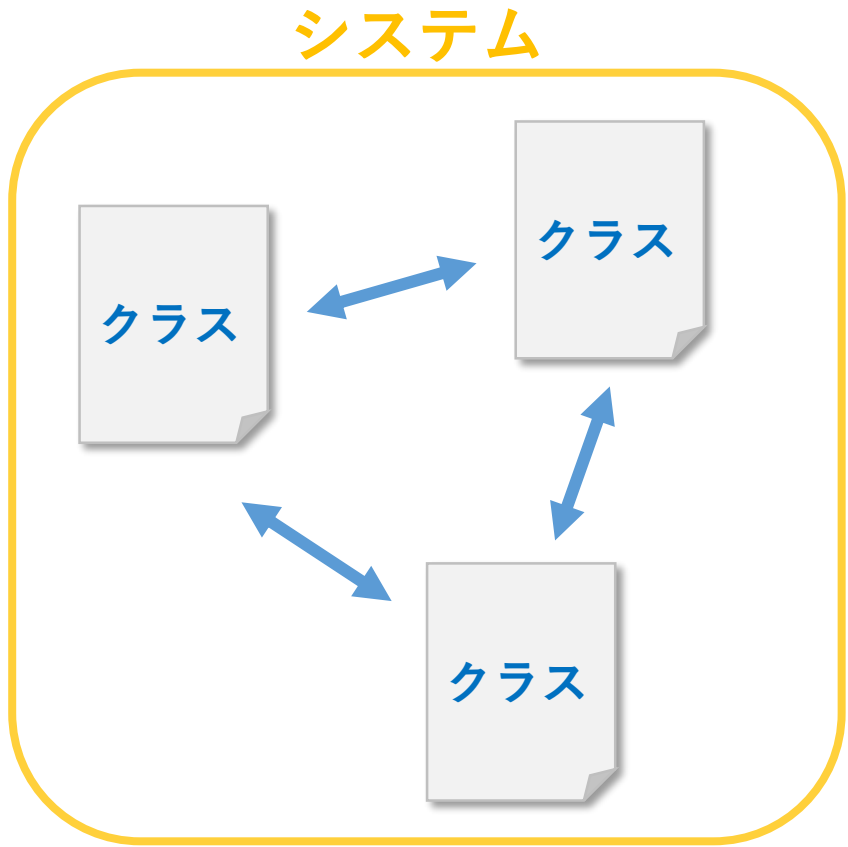
：



～どのようにクラス分けする？～

超長文
ソースコード

では、
どのように分ける？



～どのようにクラス分けする？～

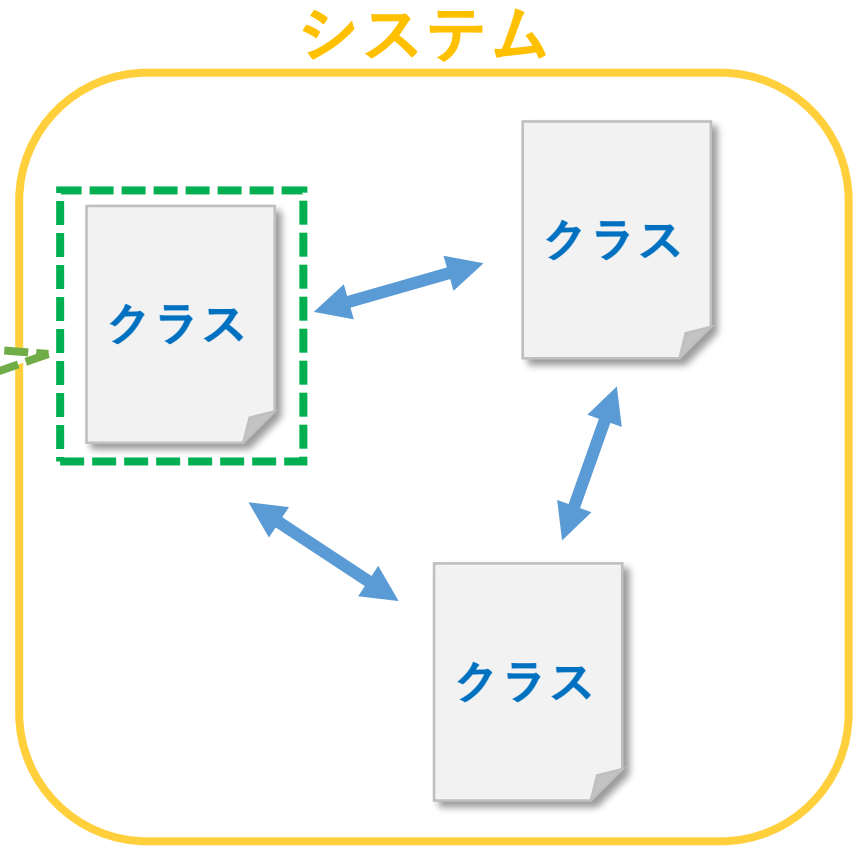
1つ1つのクラスを、役割を持った

モノ（object）

として扱おう！

||

オブジェクト指向



～どのようにクラス分けする？～

パソコン
クラス

役割を持った
モノごとに分ける

モノとモノに関連性を
持たせることでシステムを構築

システム

本体
クラス

モニター
クラス

キーボード
クラス

マウス
クラス

～モノにはどういった情報がある？～

Carクラス

～モノにはどういった情報がある？～

・加速する

・減速する

・ライトが光る

・クラクションが鳴る

⋮

Carクラス

・車種名

・オーナー

・塗装色

・現在のスピード

⋮

～モノにはどういった情報がある？～

機能

何ができるか

- 加速する
- 減速する
- ライトが光る
- クラクションが鳴る

⋮

Carクラス

属性

どんなものか

- 車種名
- オーナー
- 塗装色
- 現在のスピード

⋮

～モノにはどういった情報がある？～

機能

何ができるか

- **加速する**
  - acceleratorメソッド
- **減速する**
  - brakeメソッド
- **ライトが光る**
  - rightOnメソッド
- **クラクションが鳴る**
  - klaxonメソッド

⋮

Carクラス

属性

どんなものか

- **車種名**
  - carModel（String型）
- **オーナー**
  - owner（String型）
- **塗装色**
  - color（String型）
- **現在のスピード**
  - speed（int型）

⋮

メソッド

フィールド（変数）

～クラスはモノの設計図～

クラスブロック直下で
宣言された変数

＝

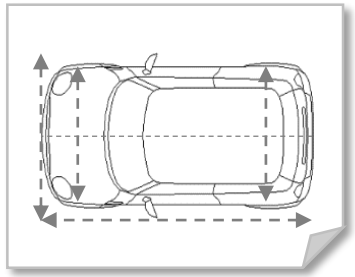
『フィールド』
そのクラス（モノ）の
情報を表します

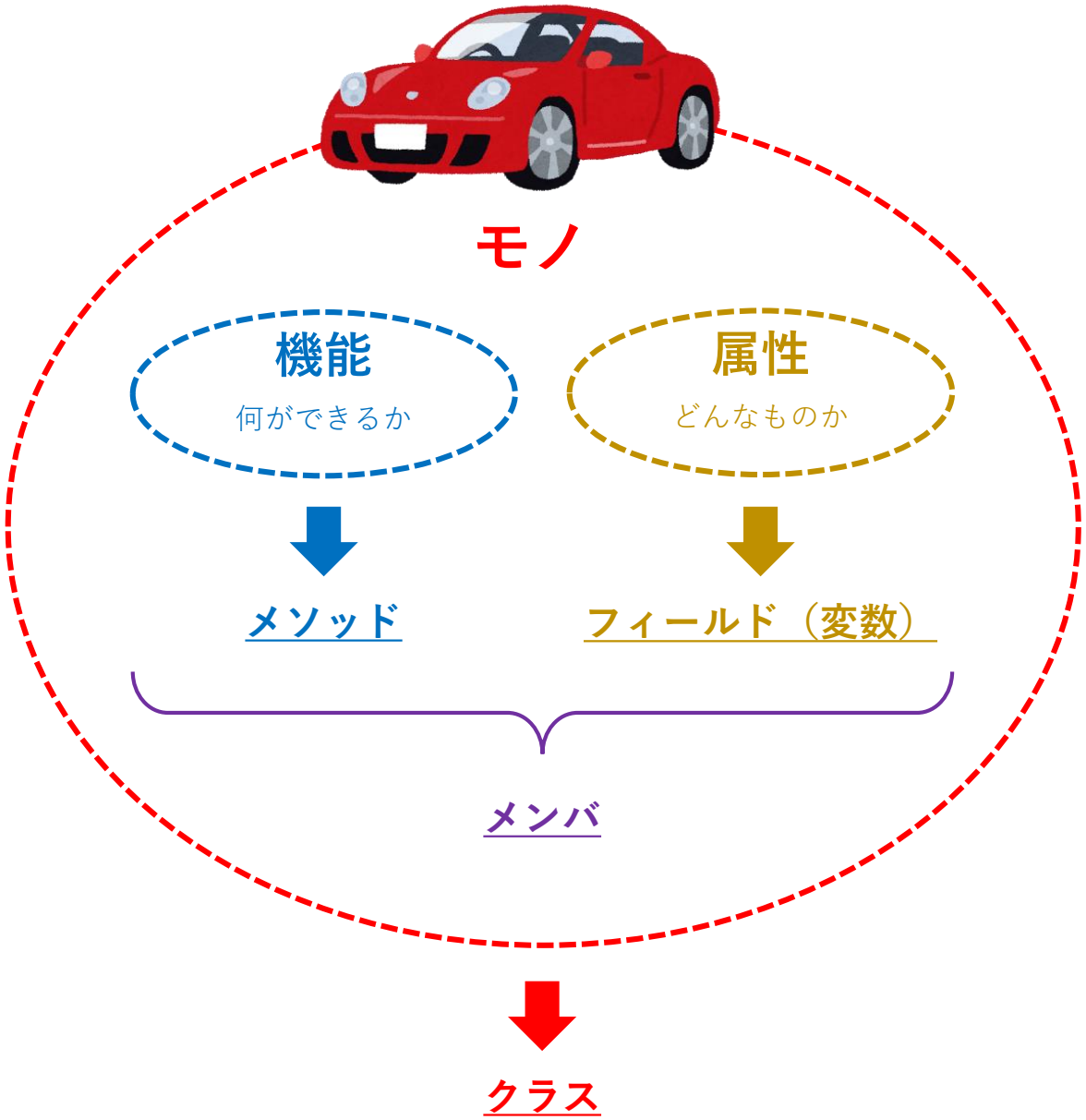
```
class Sample2_01_1_car {
    //---フィールド（クラス直下で定義された変数）----------------
    String  carModel = "クーペ" ;  //車種名
    String  owner    = "モコ"   ;  //オーナー
    String  color    = "RED"    ;  //塗装色
    int     speed    = 0        ;  //現在の速度
    boolean right    = false    ;  //ライト（true:点灯/false:消灯）

    //---メソッド-----------------------------------------------

    //acceleratorメソッド（現在の速度を1km/h上げる）
    void accelerator(){
        speed++ ;
        System.out.println("（少し速くなった）") ;
    }

    //brakeメソッド（現在の速度を1km/h下げる）
    void brake(){
        speed-- ;
        System.out.println("（少し遅くなった）") ;
    }

    //rightOnメソッド（ライトを点灯させる）
    void rightOn(){
        right = true ;
        System.out.println("（周囲が明るくなった）") ;
    }

    //rightOffメソッド（ライトを消灯する）
    void rightOff(){
        right = false ;
        System.out.println("（周囲が暗くなった）") ;
    }
    //klaxonメソッド（クラクションを鳴らす）
    void klaxon(){
        System.out.println("「プップ～～～～～～～～～～ッ」") ;
    }
}
```

属性
どんなものか

機能
何ができるか

モノの設計図





≪オブジェクト指向とは≫

□**オブジェクト指向は『ある役割を持ったモノ』ごとにクラスを分割し、モノとモノとの関係性を定義していくことでシステムを作り上げようとするシステム構成の考え方**のことです。

モノ（クラス）ごとに分業して一気に開発を進めることができたり、情報の管理やメンテンナンス性に優れているという点から、大規模なシステム開発で特に効果を発揮します。

Javaは「オブジェクト指向言語」と呼ばれ、オブジェクト指向でシステムを作るための様々な機能を提供しています。

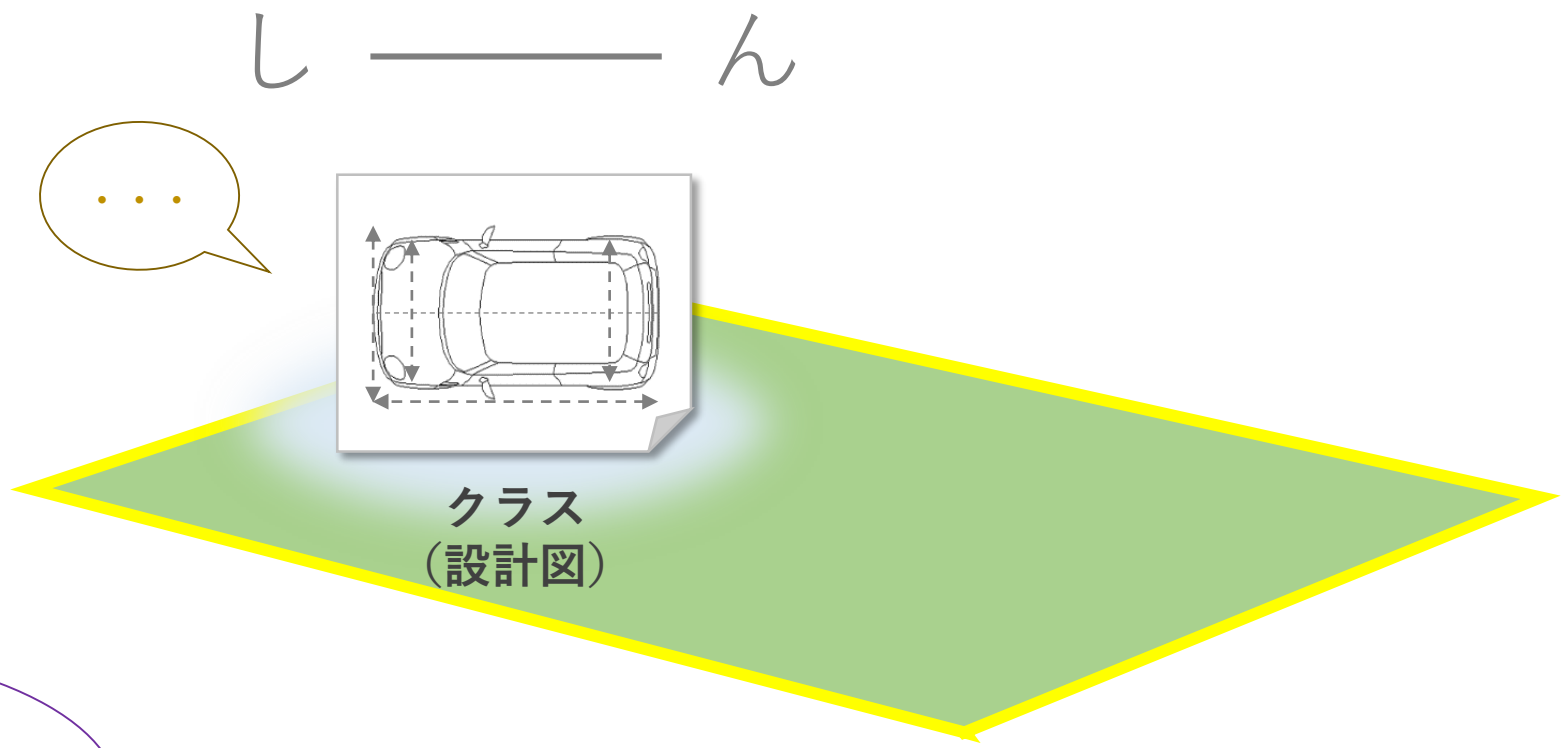
□モノを表現するためには2つの情報**『属性（どのようなものか）』**
『機能（何ができるか）』が必要になります。

□これまでクラスブロック直下にはメソッドしか定義してきませんでしたが、実は変数も定義することが可能です。
このクラスブロック直下で定義された変数のことを**フィールド**と言い、**そのクラス（＝モノ全体の設計図）の『属性』を表します。**

また、**メソッドはそのクラスの『機能』を表します。**

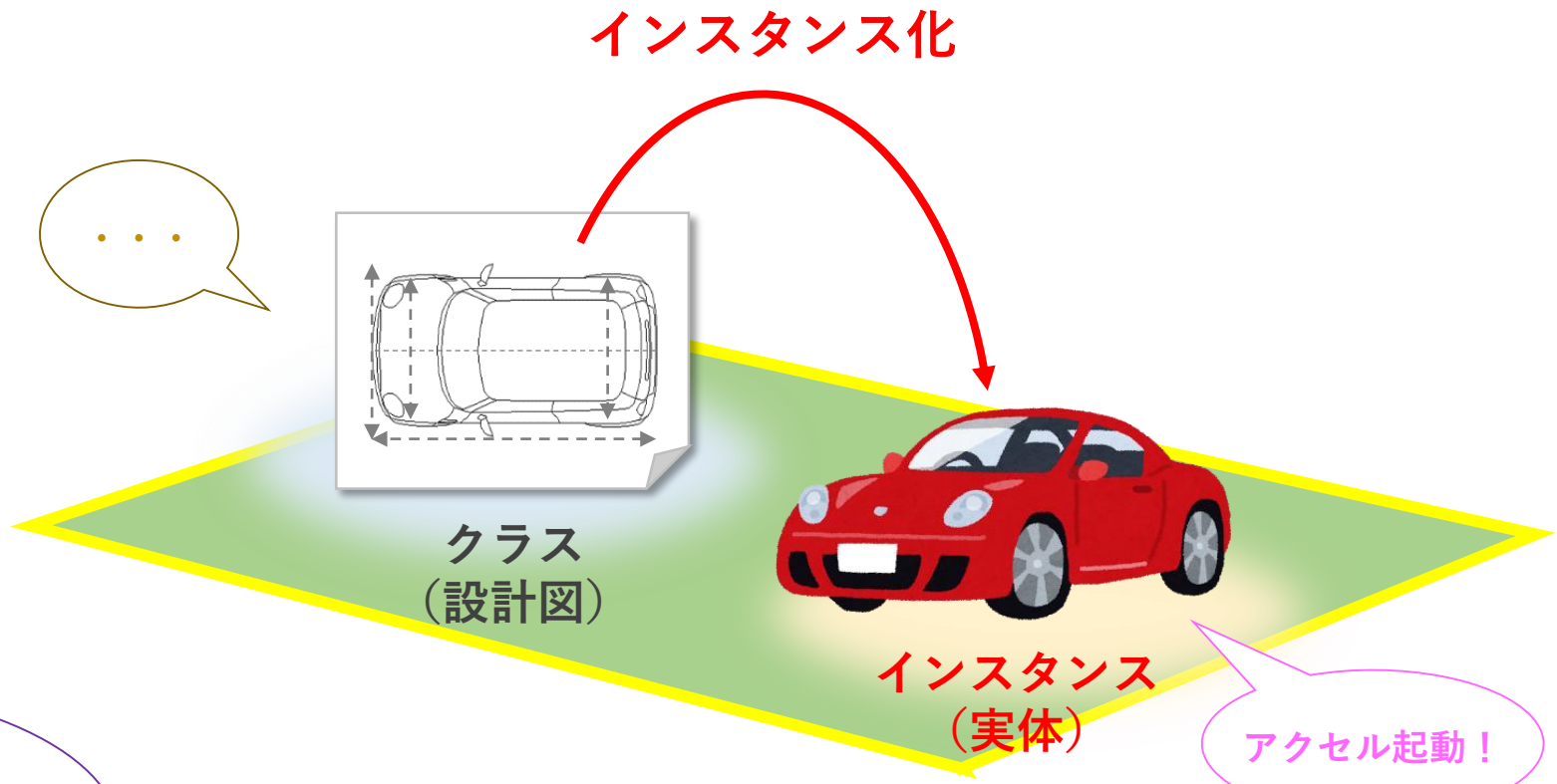
□メソッドとフィールドを合わせて**メンバ**と言います。

～実際にプログラムとして動作するのは設計図を元に作り出された”実体”～



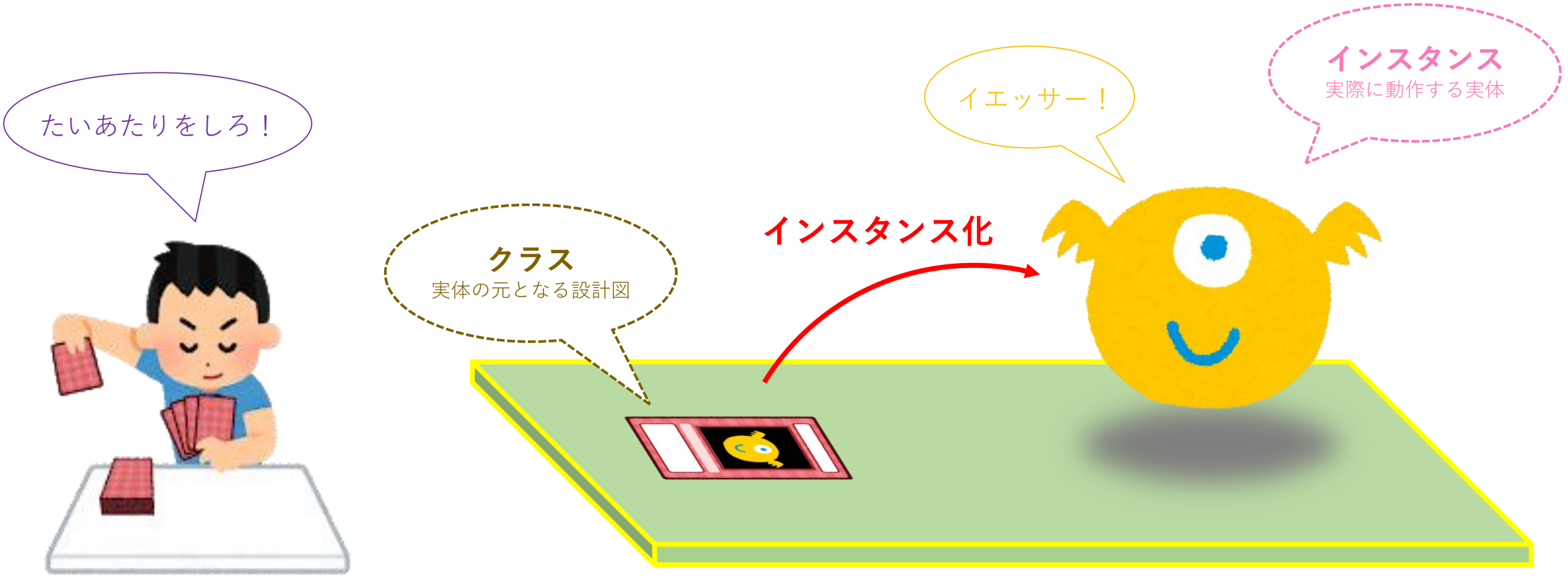
速度を上げてください！

～実際にプログラムとして動作するのは設計図を元に作り出された”実体”～



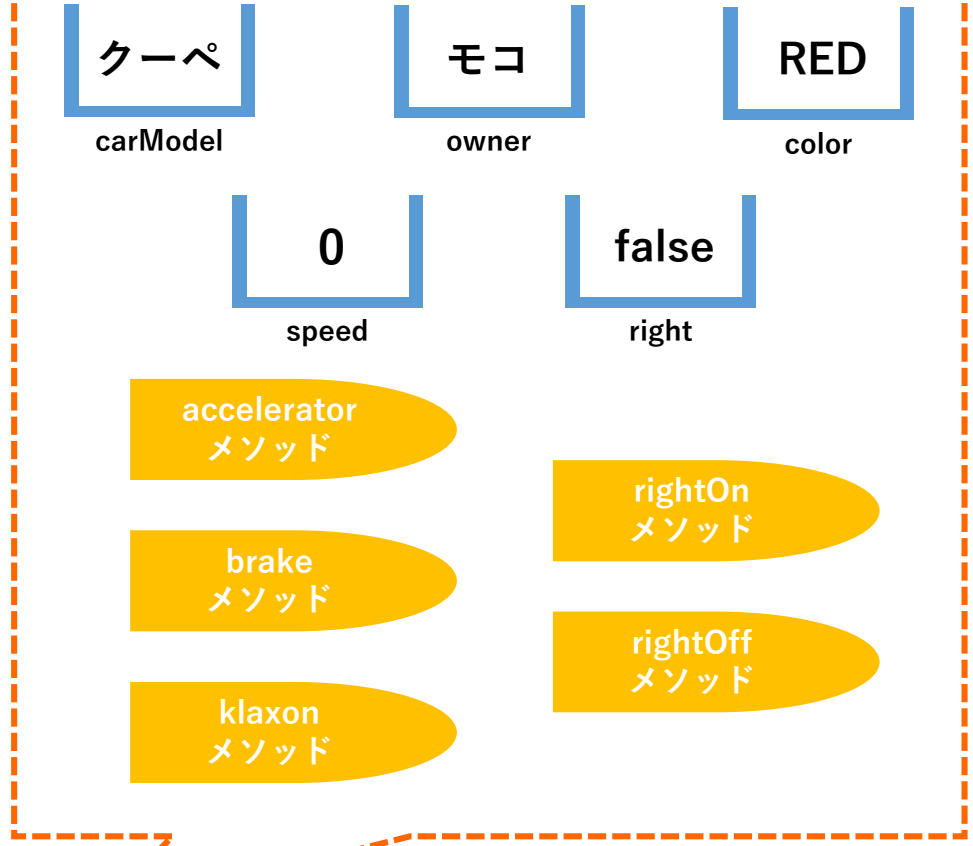
速度を上げてください！

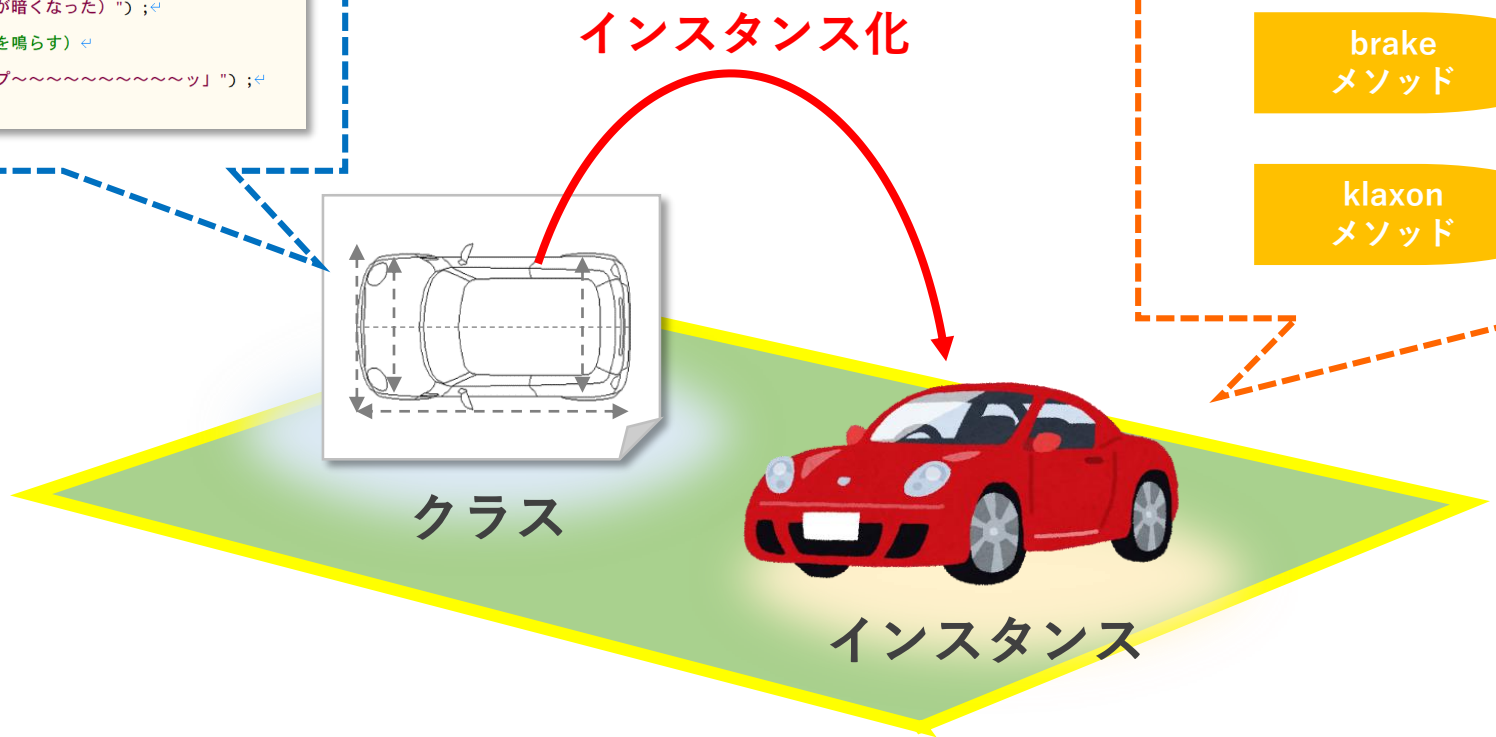
～実際にプログラムとして動作するのは設計図を元に作り出された”実体”～



～インスタンスのイメージ～

```
class Sample2_01_1_car {
    //---フィールド（クラス直下で定義された変数）----------------
    String  carModel = "クーペ" ;  //車種名
    String  owner    = "モコ"   ;  //オーナー
    String  color    = "RED"    ;  //塗装色
    int     speed    = 0        ;  //現在の速度
    boolean right    = false    ;  //ライト（true:点灯/false:消灯）

    //---メソッド-----------------------------------------------
    //acceleratorメソッド（現在の速度を1km/h上げる）
    void accelerator(){
        speed++ ;
        System.out.println("（少し速くなった）") ;
    }
    //brakeメソッド（現在の速度を1km/h下げる）
    void brake(){
        speed-- ;
        System.out.println("（少し遅くなった）") ;
    }
    //rightOnメソッド（ライトを点灯させる）
    void rightOn(){
        right = true ;
        System.out.println("（周囲が明るくなった）") ;
    }
    //rightOffメソッド（ライトを消灯する）
    void rightOff(){
        right = false ;
        System.out.println("（周囲が暗くなった）") ;
    }
    //klaxonメソッド（クラクションを鳴らす）
    void klaxon(){
        System.out.println("「プップ～～～～～～～～～～ッ」") ;
    }
}
```



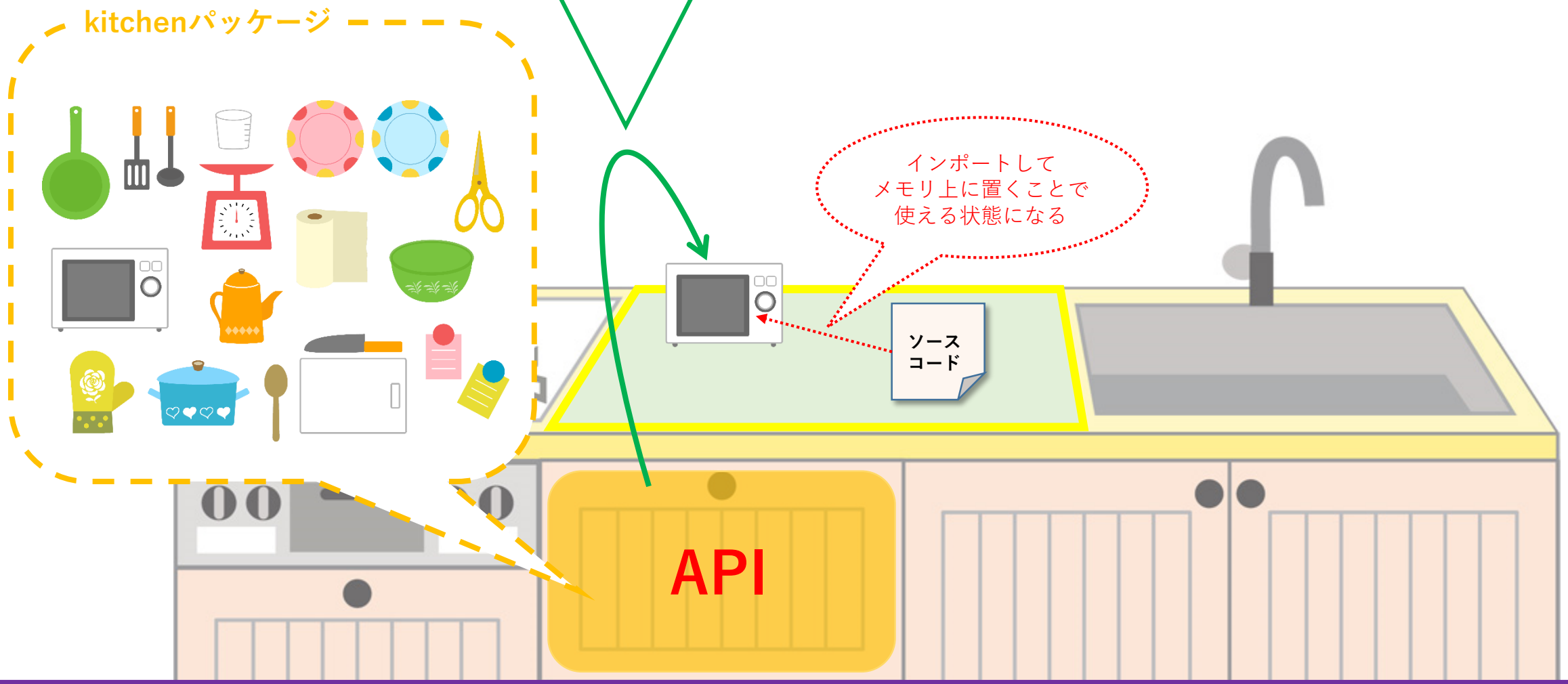


～インポートのイメージ～

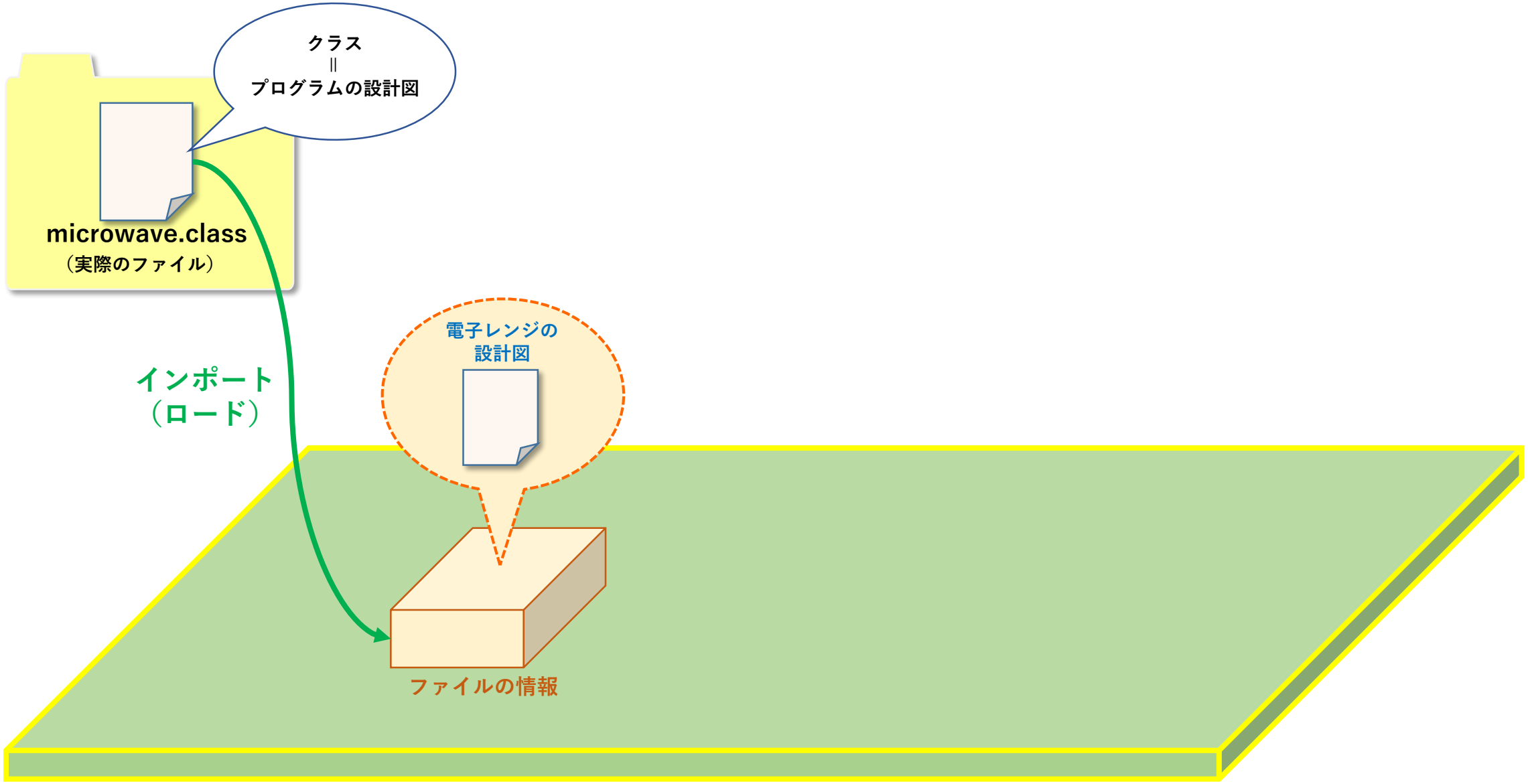
復習

≪インポート≫

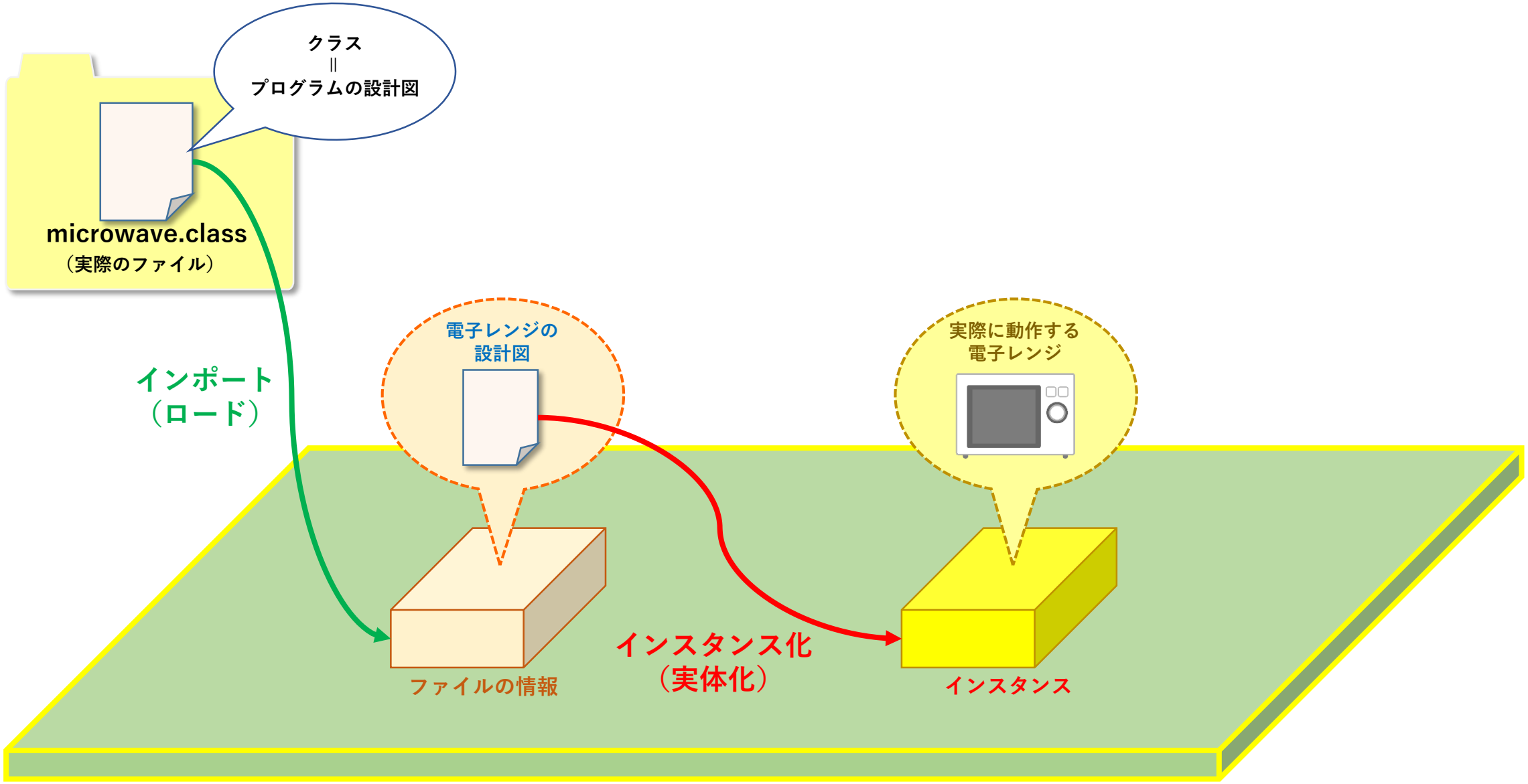
import kitchen. microwave ;



～インポートしてメモリ上に置かれるのは「設計図」～



～インポートしてメモリ上に置かれるのは「設計図」～



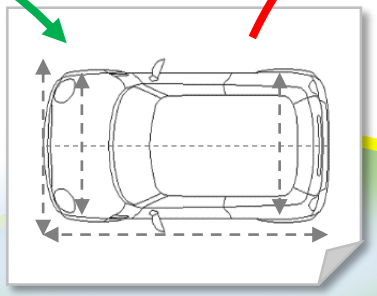
～モノだけではシステムは動かない！～

インポート（ロード）

インスタンス化

実際のファイル

・・・



クラス
（設計図）

インスタンス
（実体）

アクセル起動！

mainメソッドを持つ
クラスなど

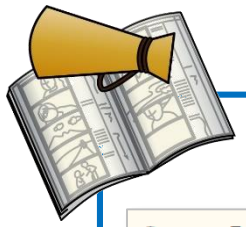
台本
プログラムの処理手順を定義

まずはクラスファイルを
インポート！

インスタンス化を
実行！

インスタンスの
速度を上げるメソッドを起動！

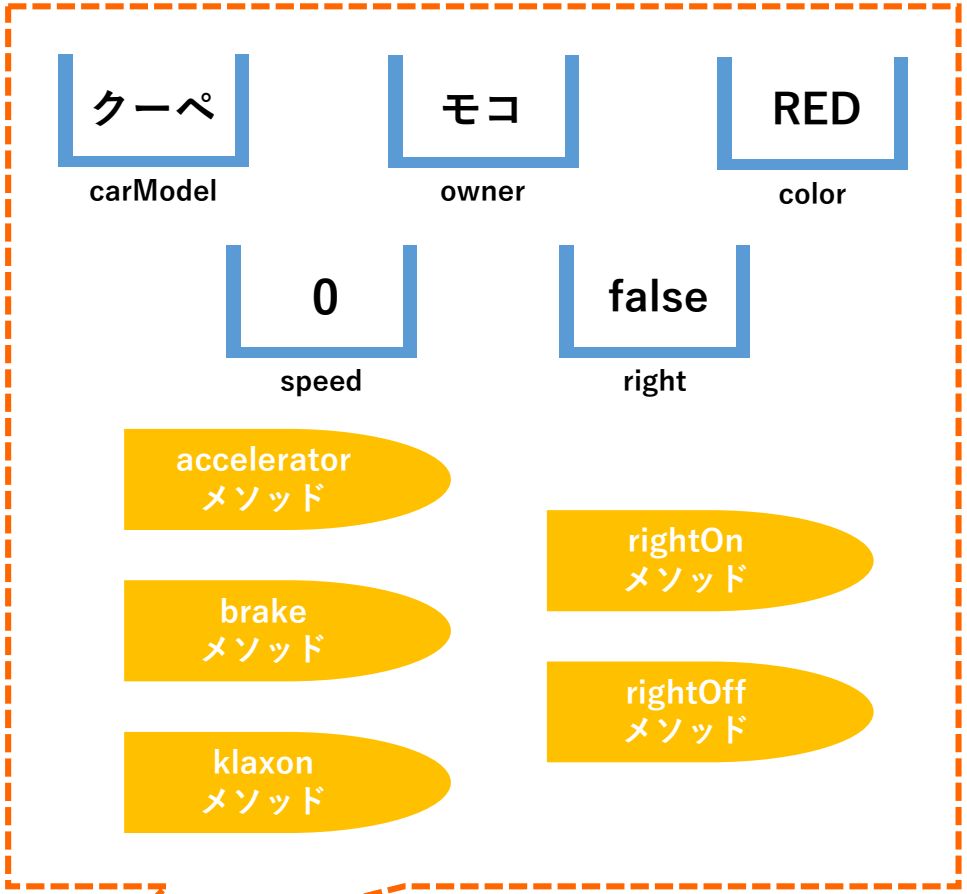
～インスタンスの活用（インスタンス化）～

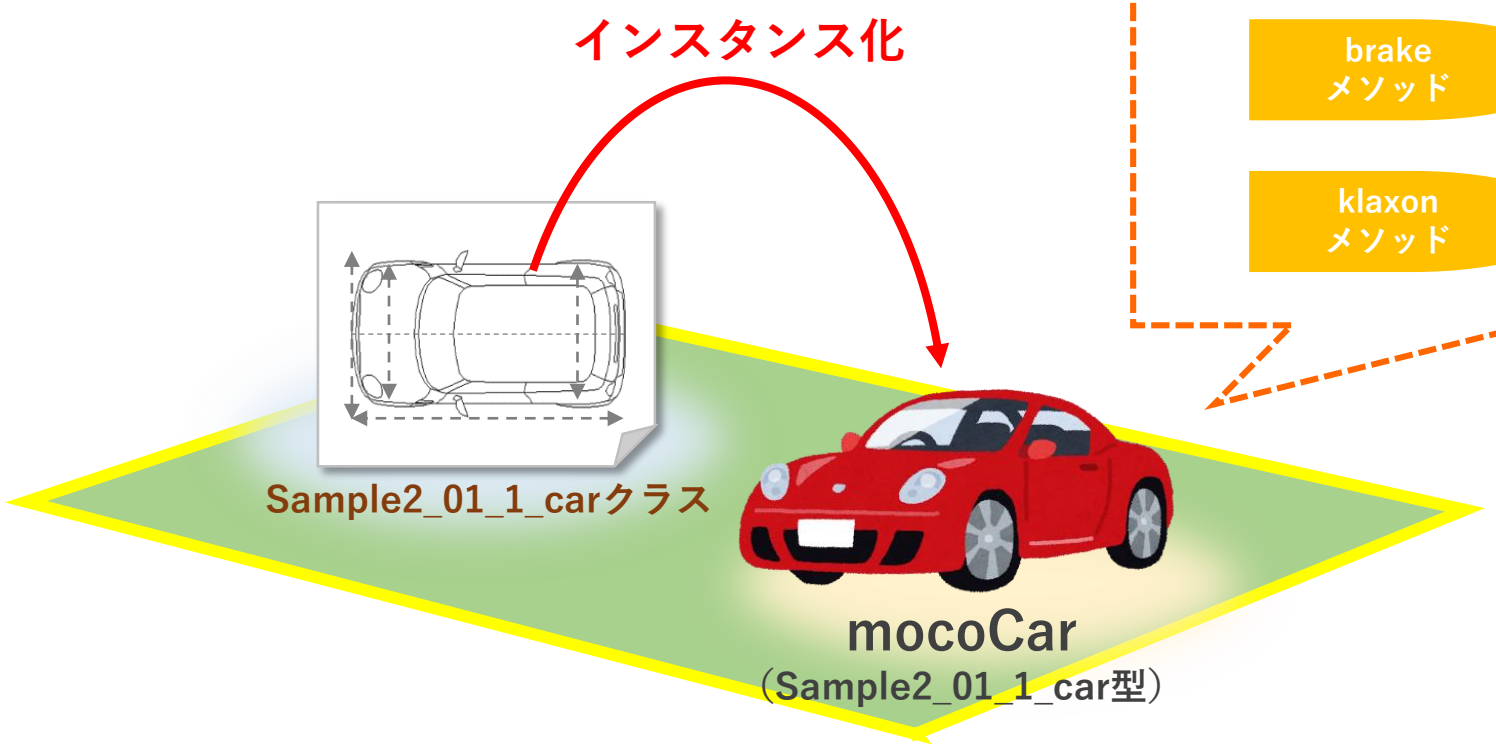


Sample2_01_1_driveクラス

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

型（クラス名）　変数名（インスタンス名）　Sample2_01_1_carクラスのインスタンス化





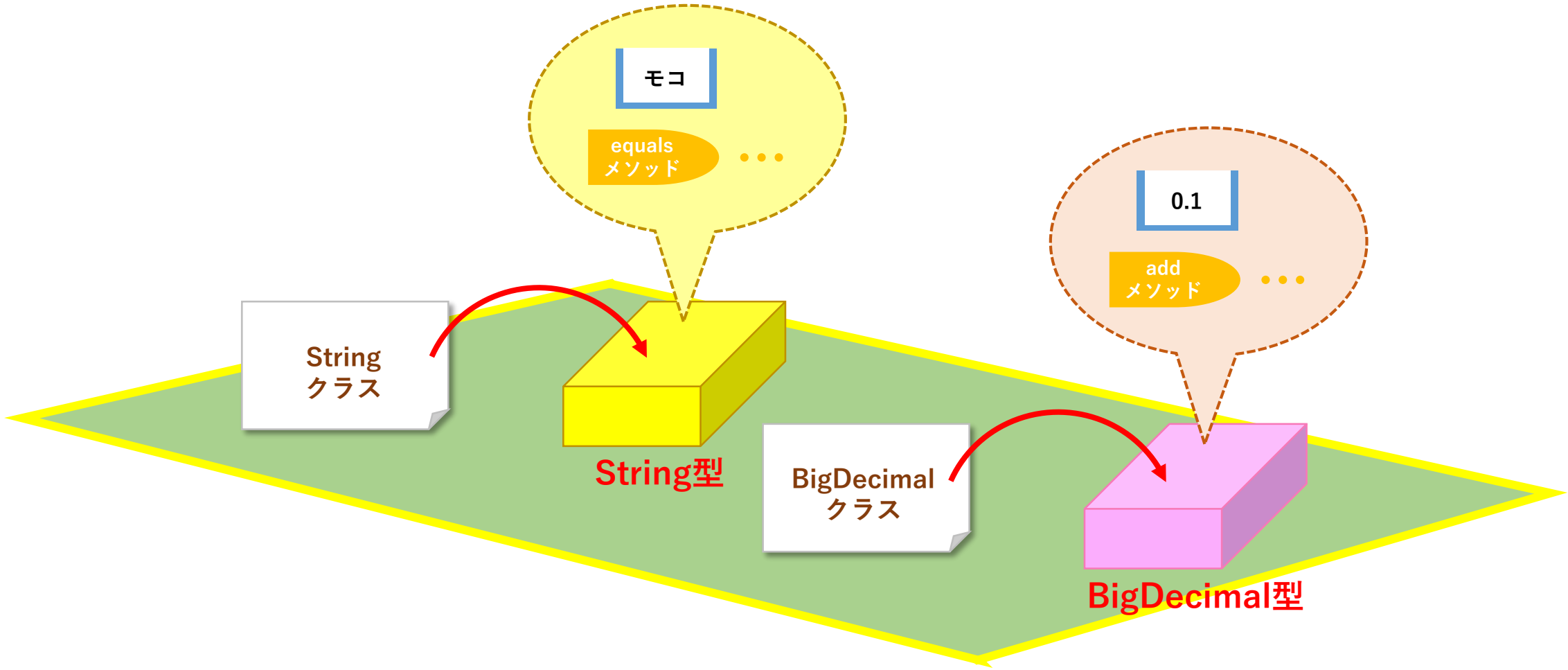
～BigDecimalクラスの活用～

復習

メモリ上
（2進数の世界）


```
BigDecimal b1      = new BigDecimal( 1.5 ) ;
BigDecimal b2      = new BigDecimal( -6 ) ;
BigDecimal b3      = new BigDecimal( "0.1" ) ;
```

～インスタンスはクラス名を型として扱う～



～インスタンスの実際の姿～

Sample2_01_1_driveクラス

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

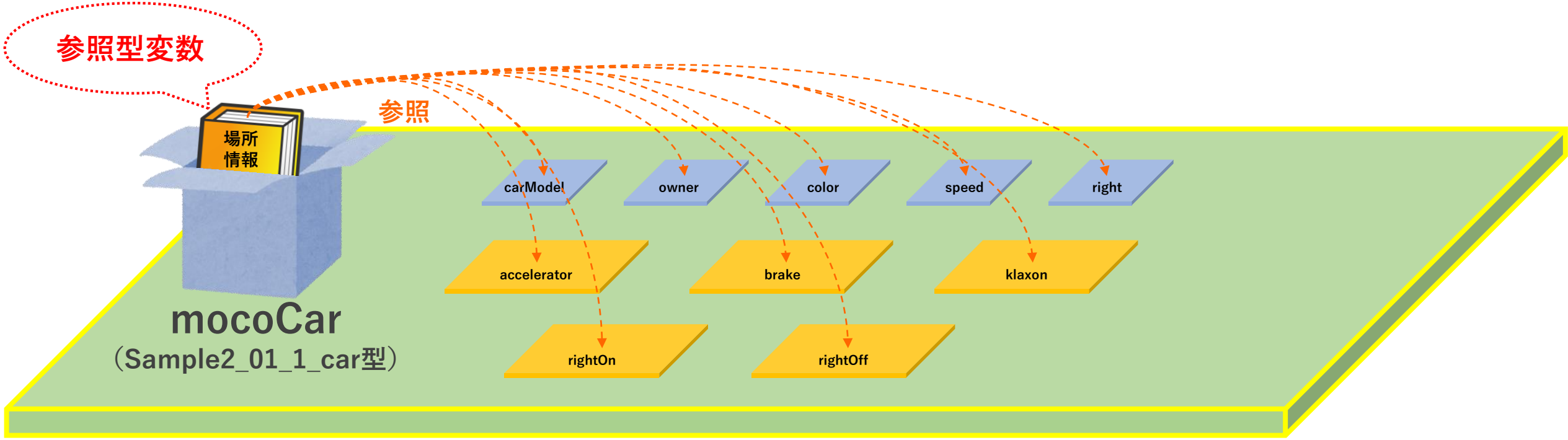
型（クラス名）　変数名（インスタンス名）　Sample2_01_1_carクラスのインスタンス化

～インスタンスの実際の姿～

Sample2_01_1_driveクラス

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

型（クラス名）　変数名（インスタンス名）　Sample2_01_1_carクラスのインスタンス化



～インスタンスの実際の姿～

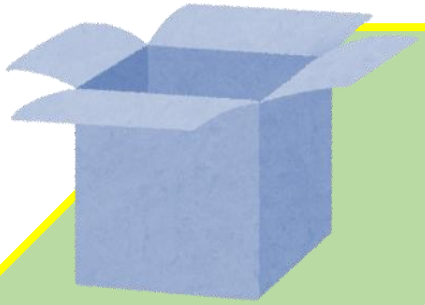
Sample2_01_1_driveクラス

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

型
（クラス名）

変数名
（インスタンス名）

参照型変数



mocoCar
（Sample2_01_1_car型）

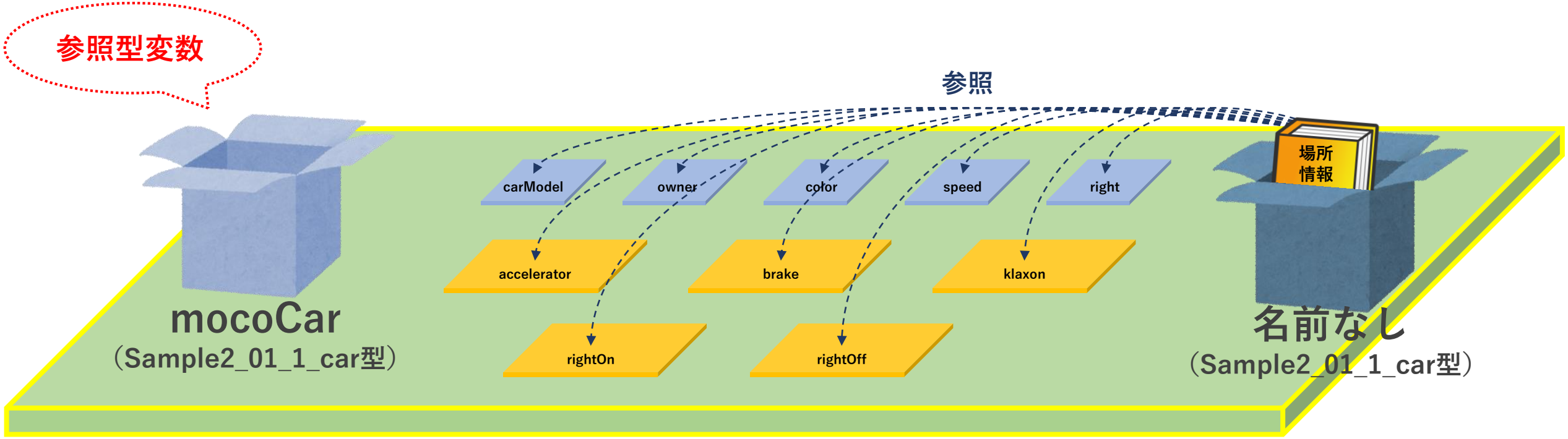
～インスタンスの実際の姿～

Sample2_01_1_driveクラス

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

型（クラス名）　変数名（インスタンス名）　Sample2_01_1_carクラスのインスタンス化

参照型変数

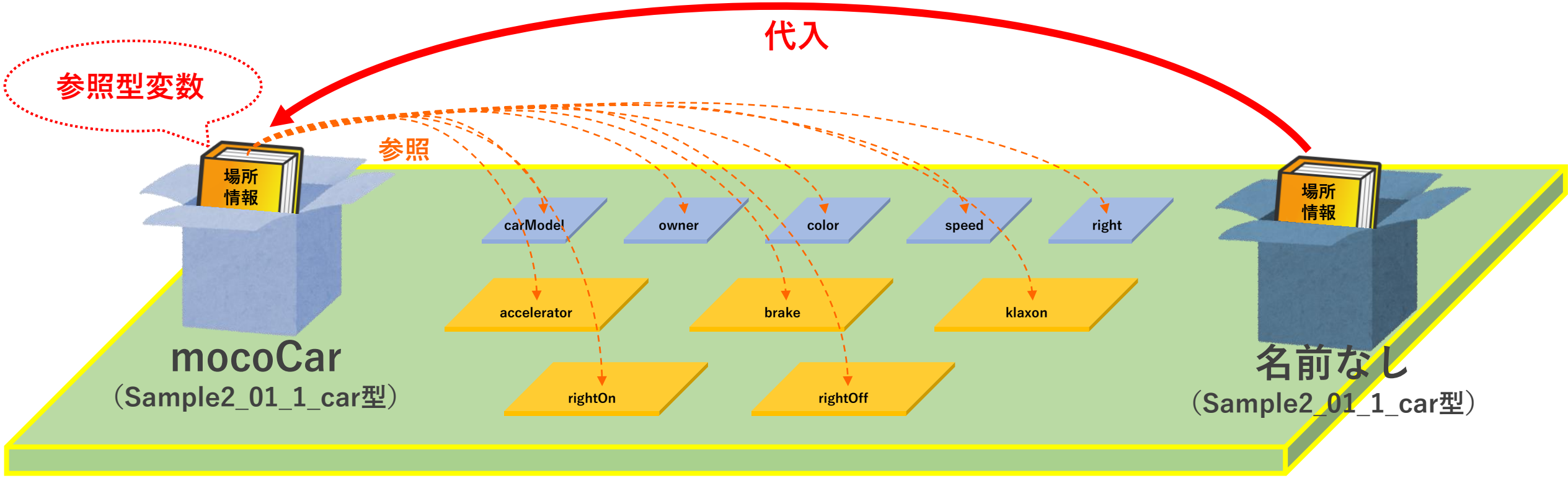


～インスタンスの実際の姿～

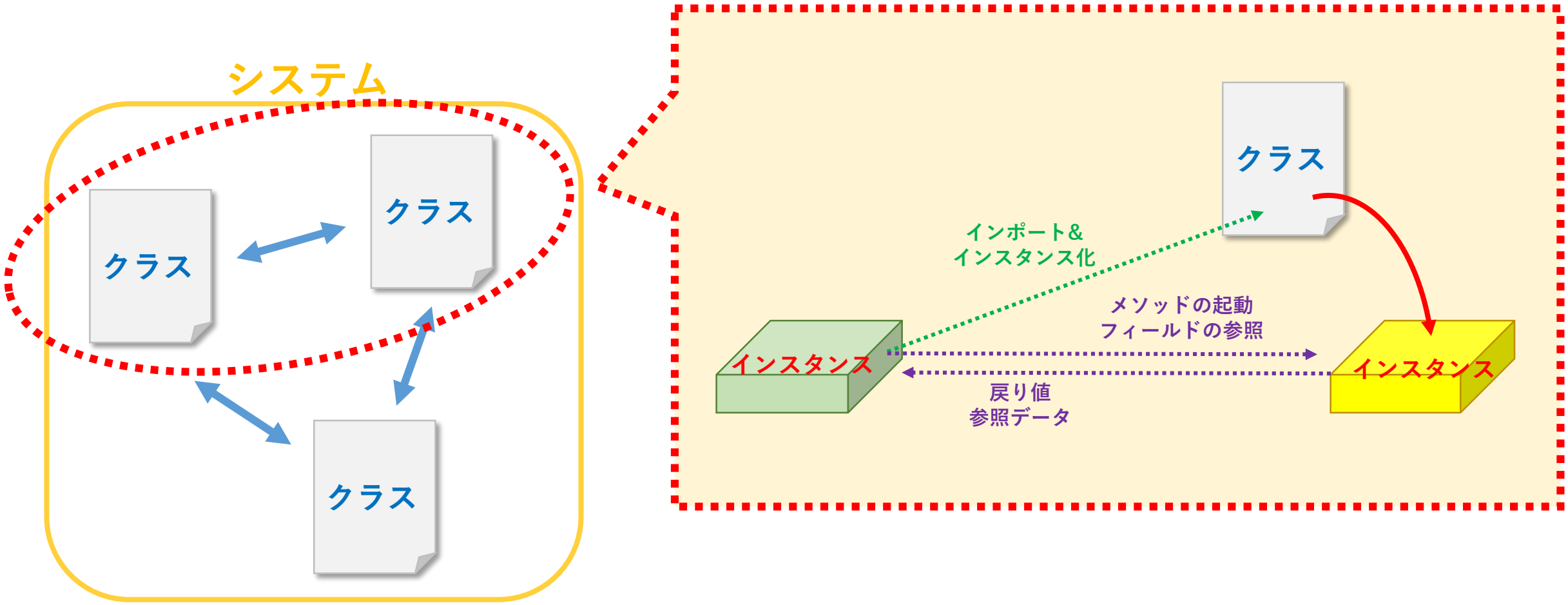
Sample2_01_1_driveクラス

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

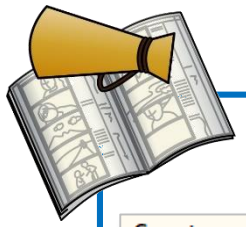
型（クラス名）　変数名（インスタンス名）　Sample2_01_1_carクラスのインスタンス化



～インスタンスの活用～


システム
クラス
クラス
クラス
クラス
インポート&
インスタンス化
メソッドの起動
フィールドの参照
インスタンス
インスタンス
戻り値
参照データ

～インスタンスの活用（変数の扱い）～



Sample2_01_1_driveクラス

```
System.out.println("mocoCar.carModel : " + mocoCar.carModel );
System.out.println("mocoCar.owner    : " + mocoCar.owner    );
System.out.println("mocoCar.color    : " + mocoCar.color    );
System.out.println("mocoCar.speed    : " + mocoCar.speed    );
System.out.println("mocoCar.right    : " + mocoCar.right    );
```

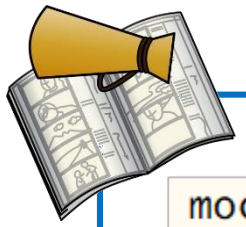
『インスタンス名 . 変数名』で
インスタンスが保有する変数を参照できる

クーペ → carModel
モコ → owner
RED → color
0 → speed
false → right

accelerator メソッド
brake メソッド
klaxon メソッド
rightOn メソッド
rightOff メソッド

mocoCar
（Sample2_01_1_car型）

～インスタンスの活用（メソッドの扱い）～

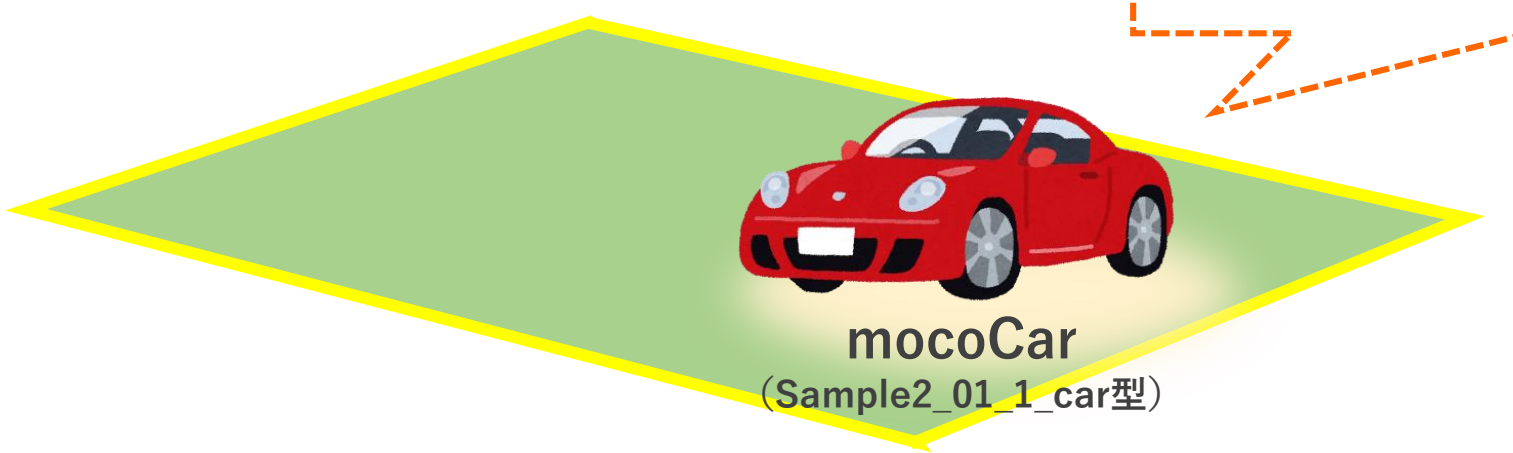


Sample2_01_1_driveクラス

```
mocoCar.accelerator();
```

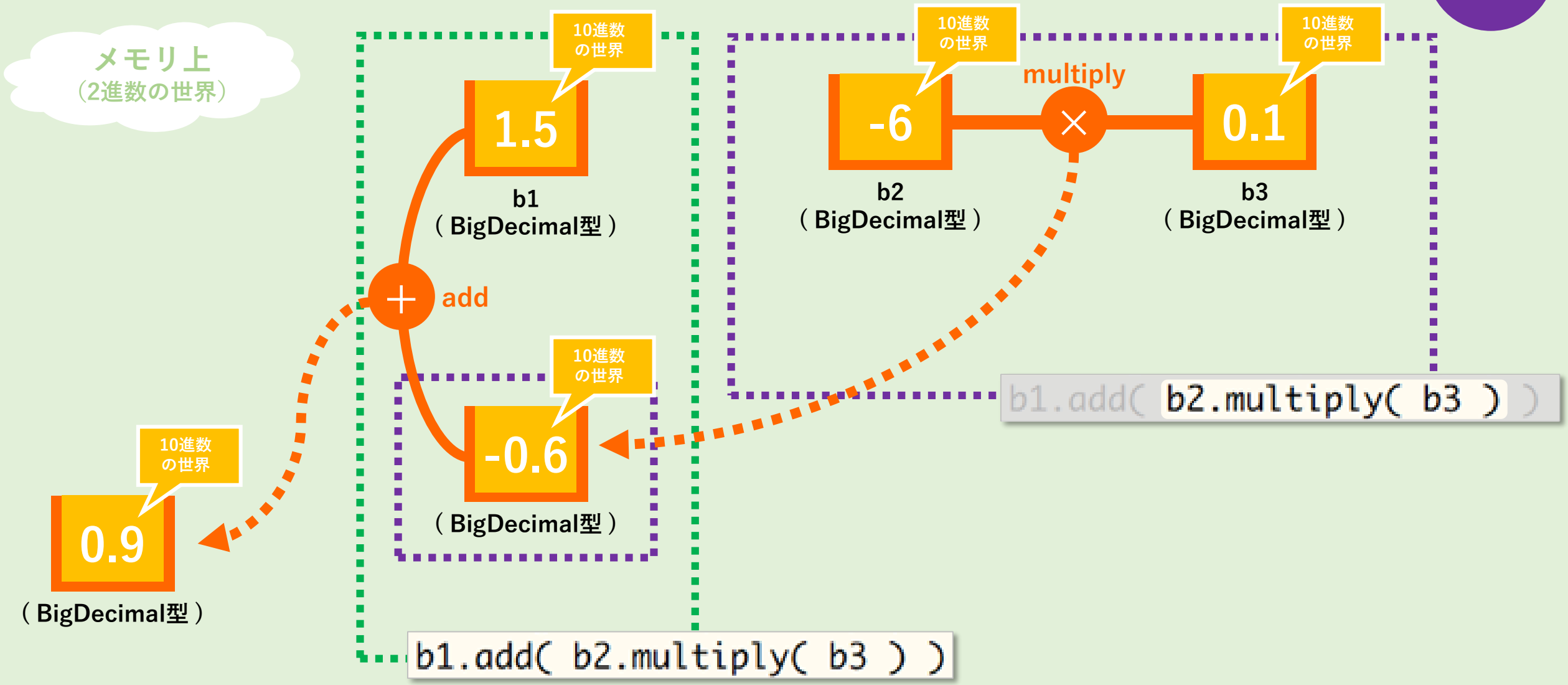
『インスタンス名．メソッド名』で
インスタンスが保有するメソッドを
起動することができる

```
void accelerator(){
    speed++ ;
    System.out.println("（少し速くなった）") ;
}
```



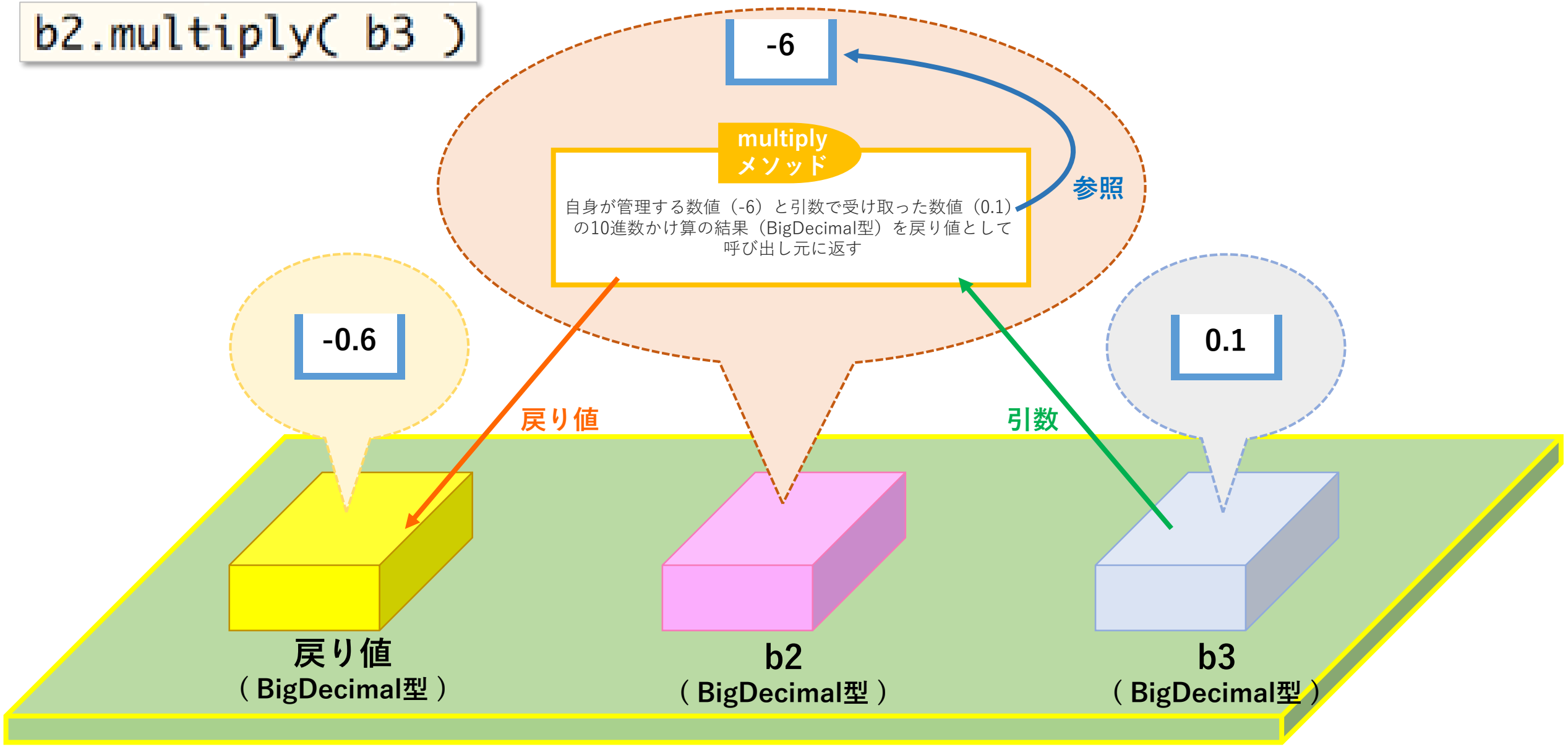
～BigDecimalクラスの活用～

復習

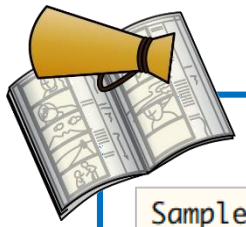


～インスタンスの活用（メソッドの扱い）～

```
b2.multiply( b3 )
```



～インスタンスは1つのクラスから複数生成できる～



Sample2_01_1_driveクラス

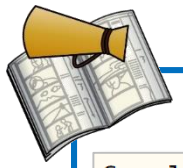
```
Sample2_01_1_car mocoCar  = new Sample2_01_1_car();
Sample2_01_1_car chocoCar = new Sample2_01_1_car();

mocoCar.accelerator();
```

1
0
speed
accelerator
メソッド
…

0
speed
accelerator
メソッド
…

mocoCar
（Sample2_01_1_car型）

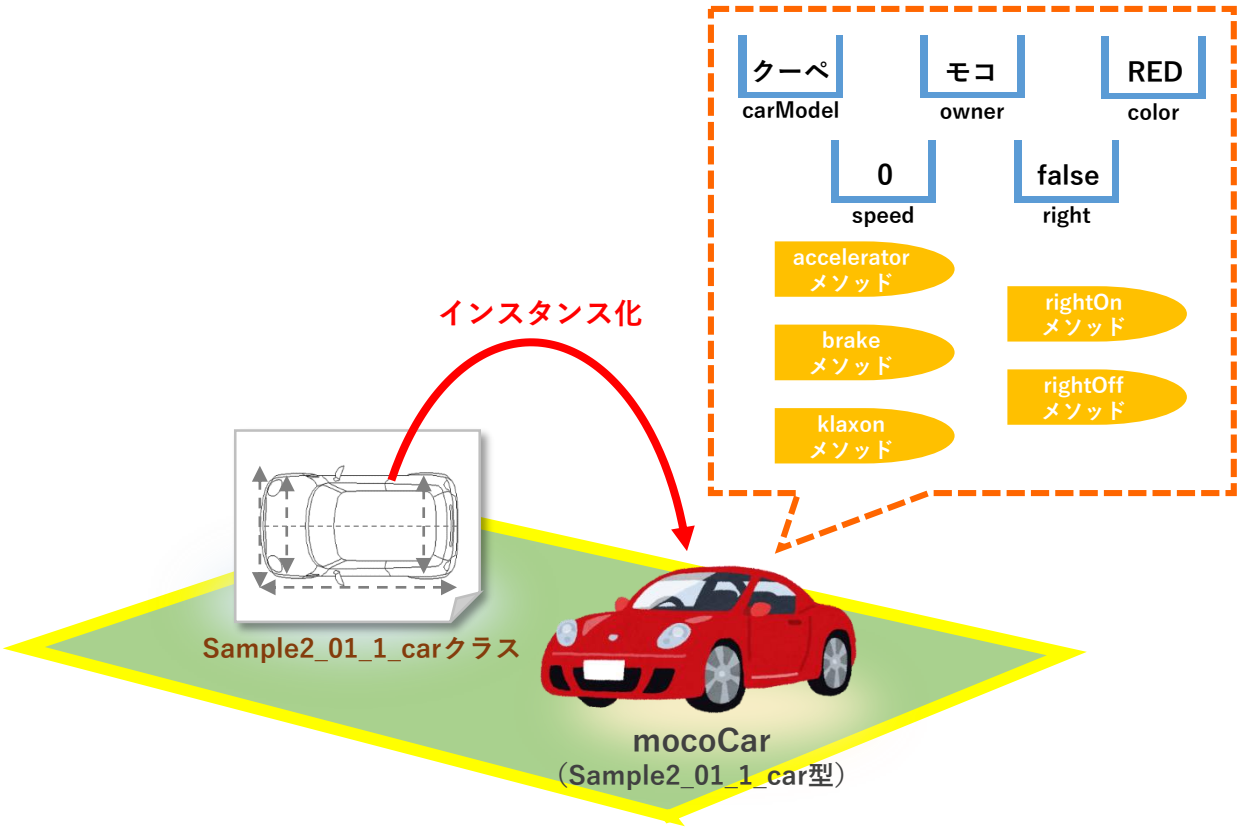
chocoCar
（Sample2_01_1_car型）



Sample2_01_1_driveクラス

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

型（クラス名）　変数名（インスタンス名）　Sample2_01_1_carクラスのインスタンス化



≪インスタンス≫

□**クラスはあくまでプログラムの設計図であるため、クラスそれ自身では命令を受け付けたり動作することはできません。**
インポートは設計図の情報をメモリ上に置くだけの処理なので、これだけではまだそのクラスの機能（メソッド）を使える状態にはなっていないことになります。

外部クラスの属性（フィールド）や機能（メソッド）を利用するためには、**そのクラス（設計図）を元に生成した、命令を受け付け実際に動作することのできる実体が必要**になります。
この実体のことを**インスタンス**と言い、クラスを元にインスタンスを生成することを**インスタンス化**と言います。

□インスタンス化は以下のように記述します。

クラス名 インスタンス名 = new クラス名() ;

※厳密な情報ではないのですが、現時点ではここまでの認識を持ちましょう。

□インスタンスの実体はフィールドやメソッドのメモリ上の場所情報を管理する参照型変数であり、クラス名を型として扱います。

□以下のように記述すると生成したインスタンスのフィールドを参照したりメソッドを実行したりすることができます。

インスタンス名 . フィールド変数名
インスタンス名 . メソッド名（引数）

□1つのクラスからいくらでもインスタンス化はは可能であり、インスタンス名さえ違っていればそれらは完全に独立したモノとして扱うことができます。