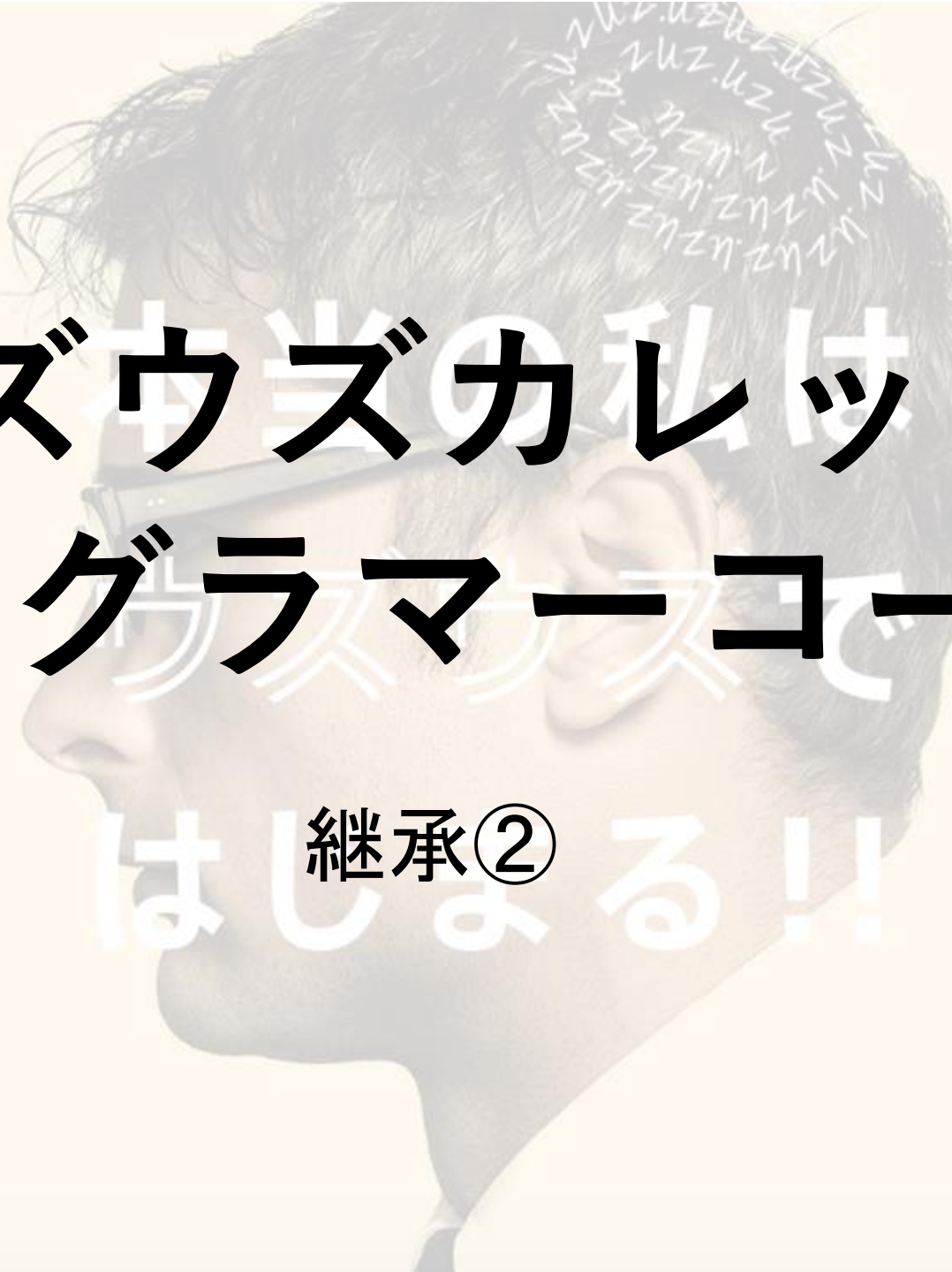


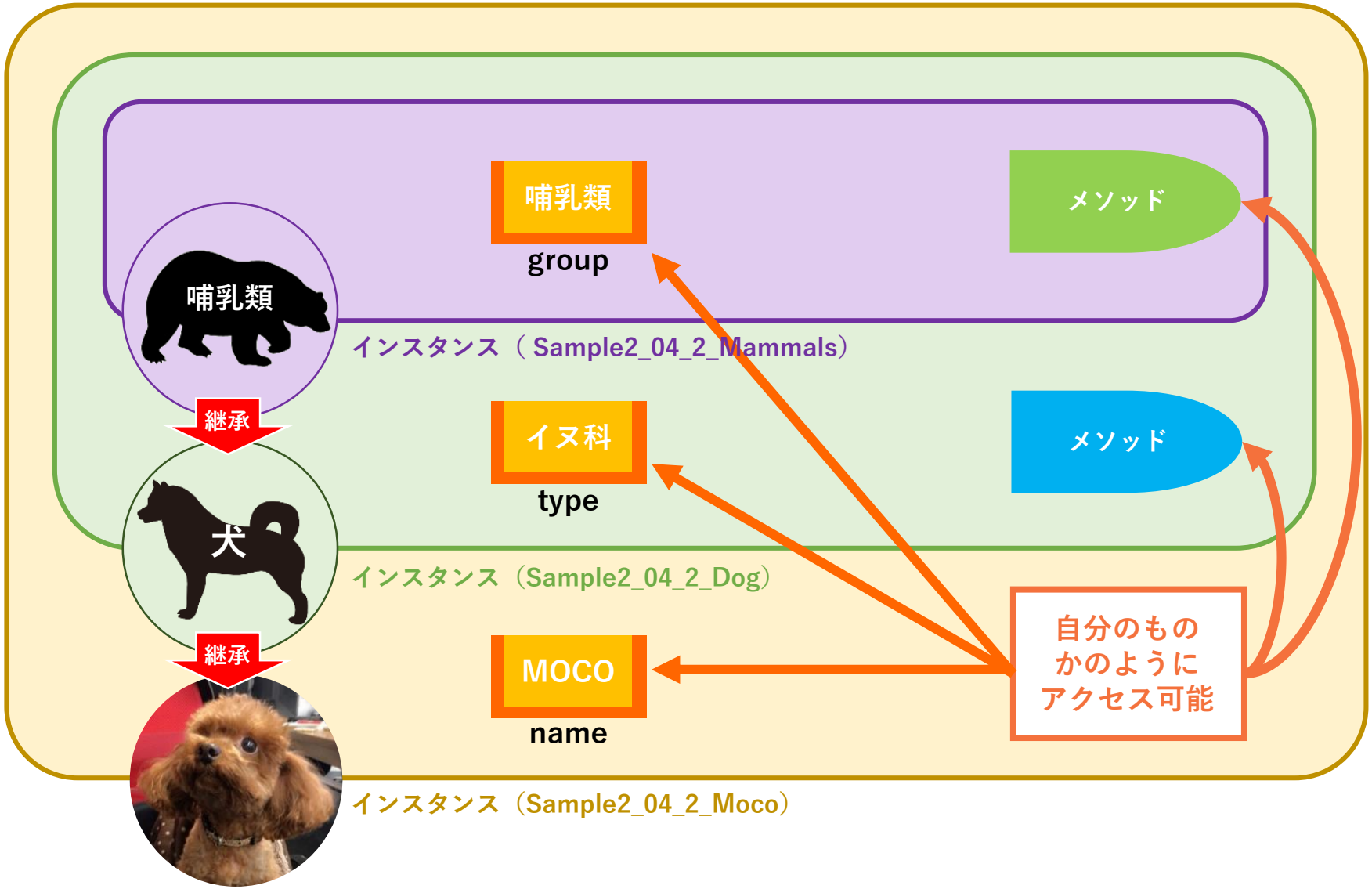
ウズウズカレッジ
プログラマーコース

継承②

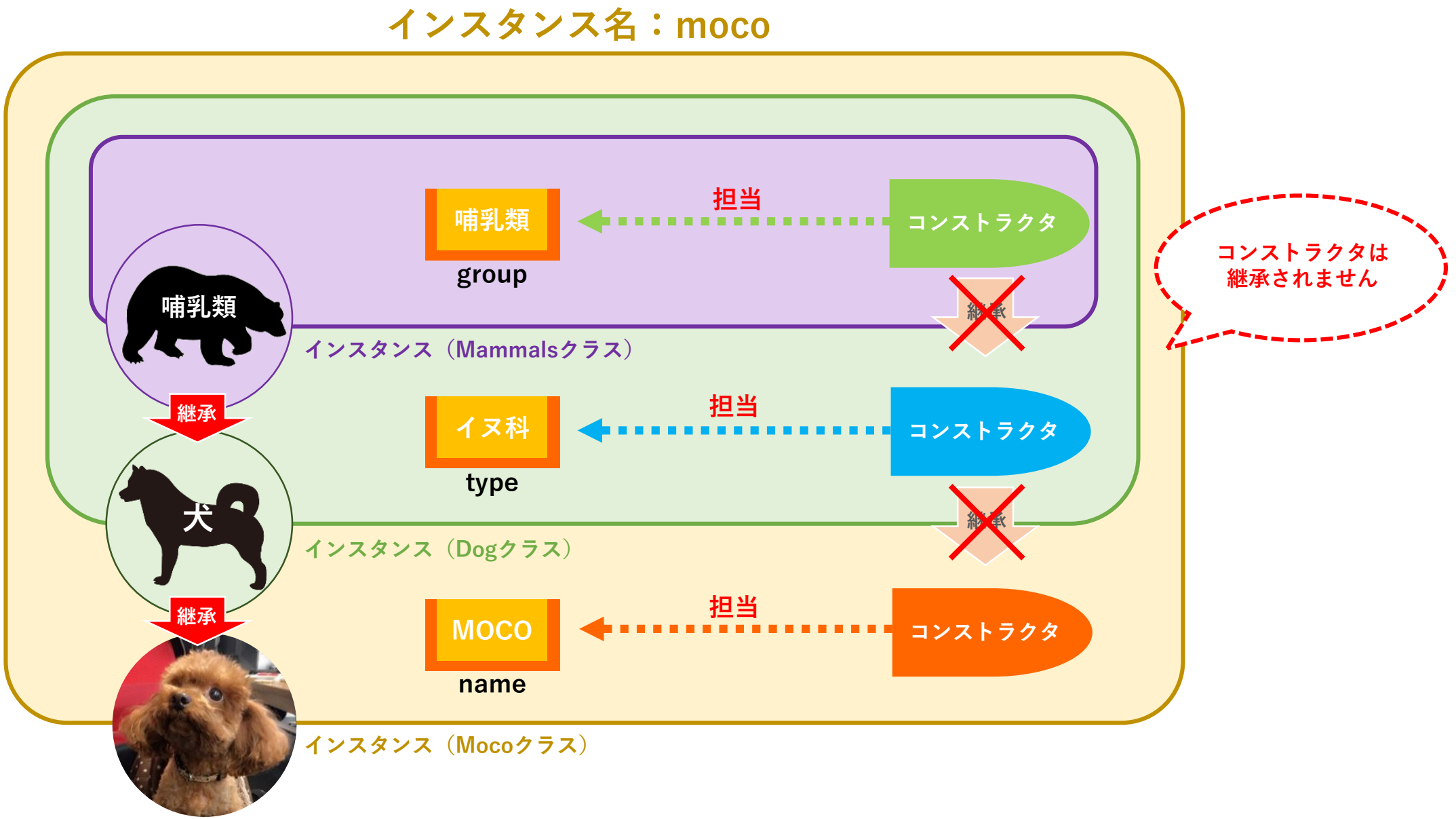
～継承関係にあるオブジェクトのイメージ～

復習

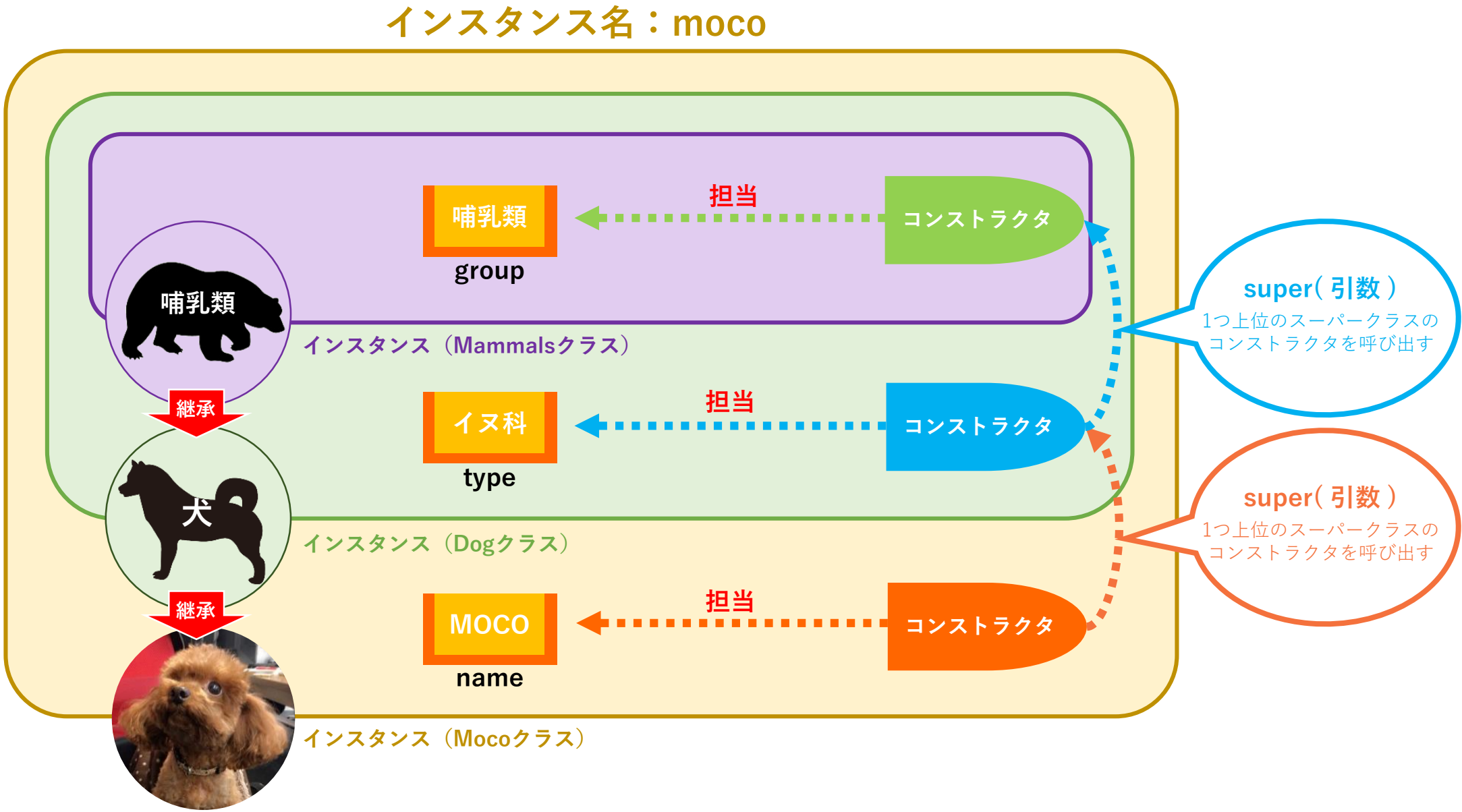
インスタンス名：moco



～継承におけるコンストラクタの動き～



～継承におけるコンストラクタの動き～



～コンストラクタは一番深いものから動く！～

復習

▼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

フィールド

～コンストラクタは一番深いものから動く！～

復習

▼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！～

復習

▼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("□□▲----------------------------------------------------") ;
}
```

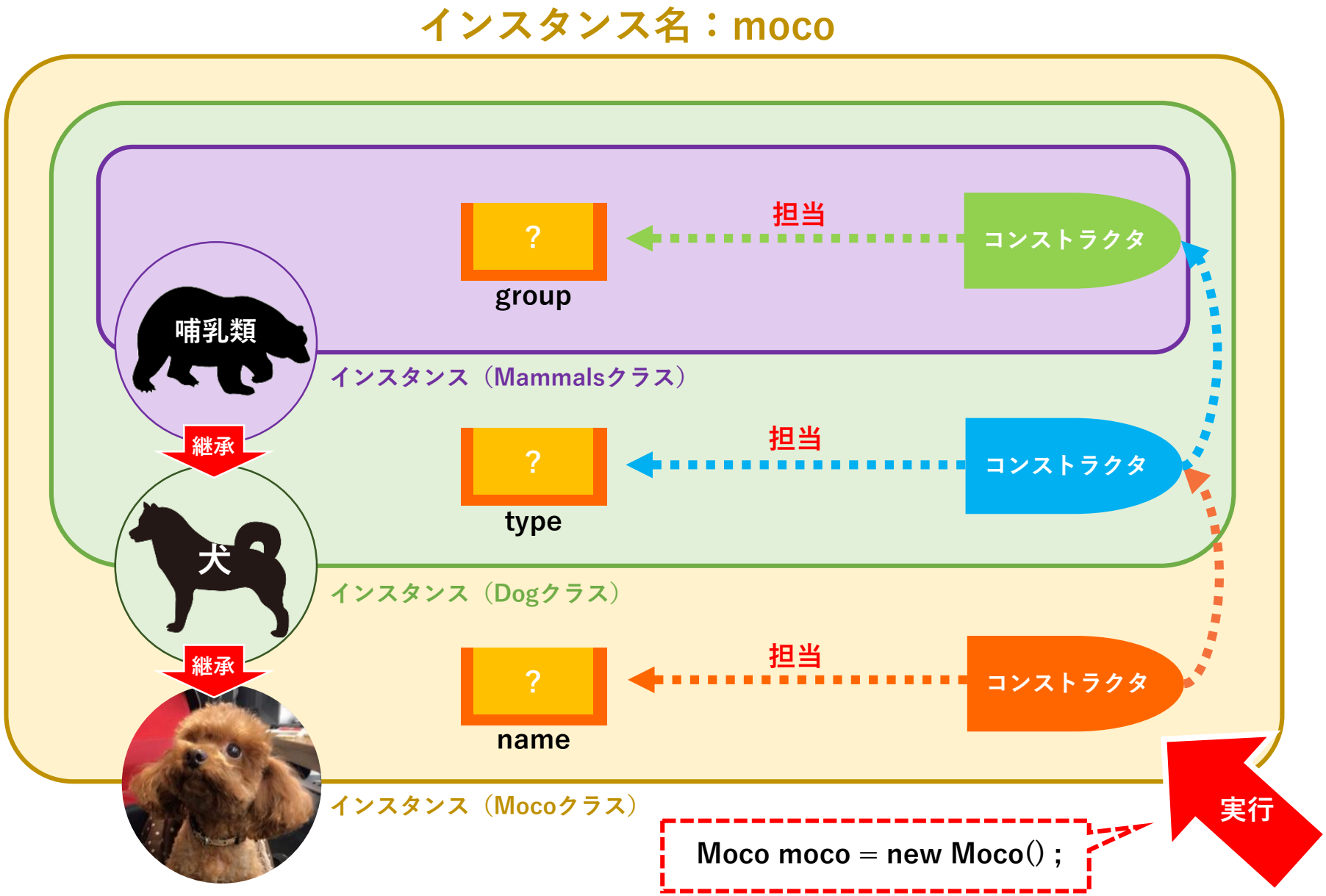
具体的な処理

```
//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("□□▲----------------------------------------------------") ;
}
```

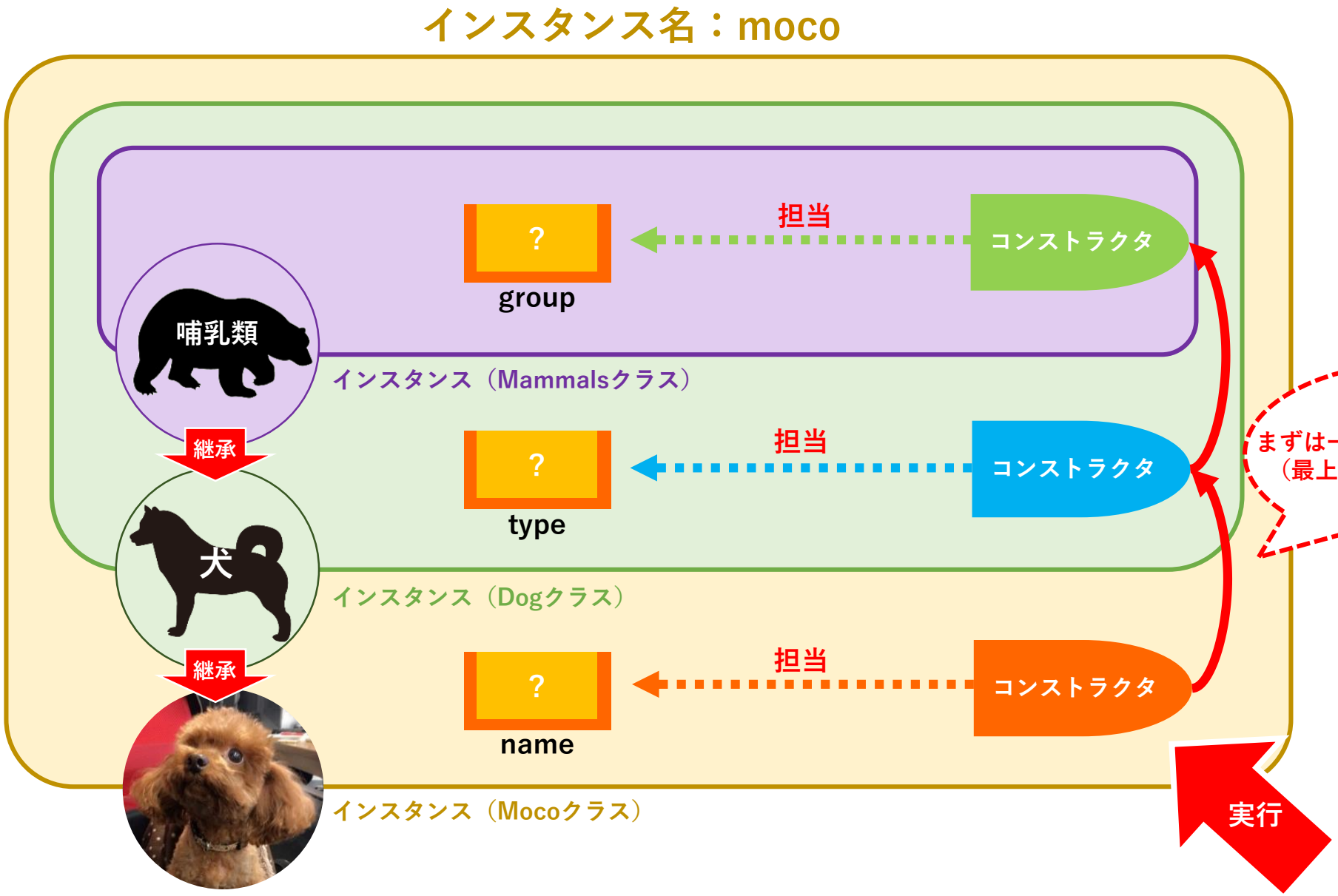
コンパイルエラー！

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

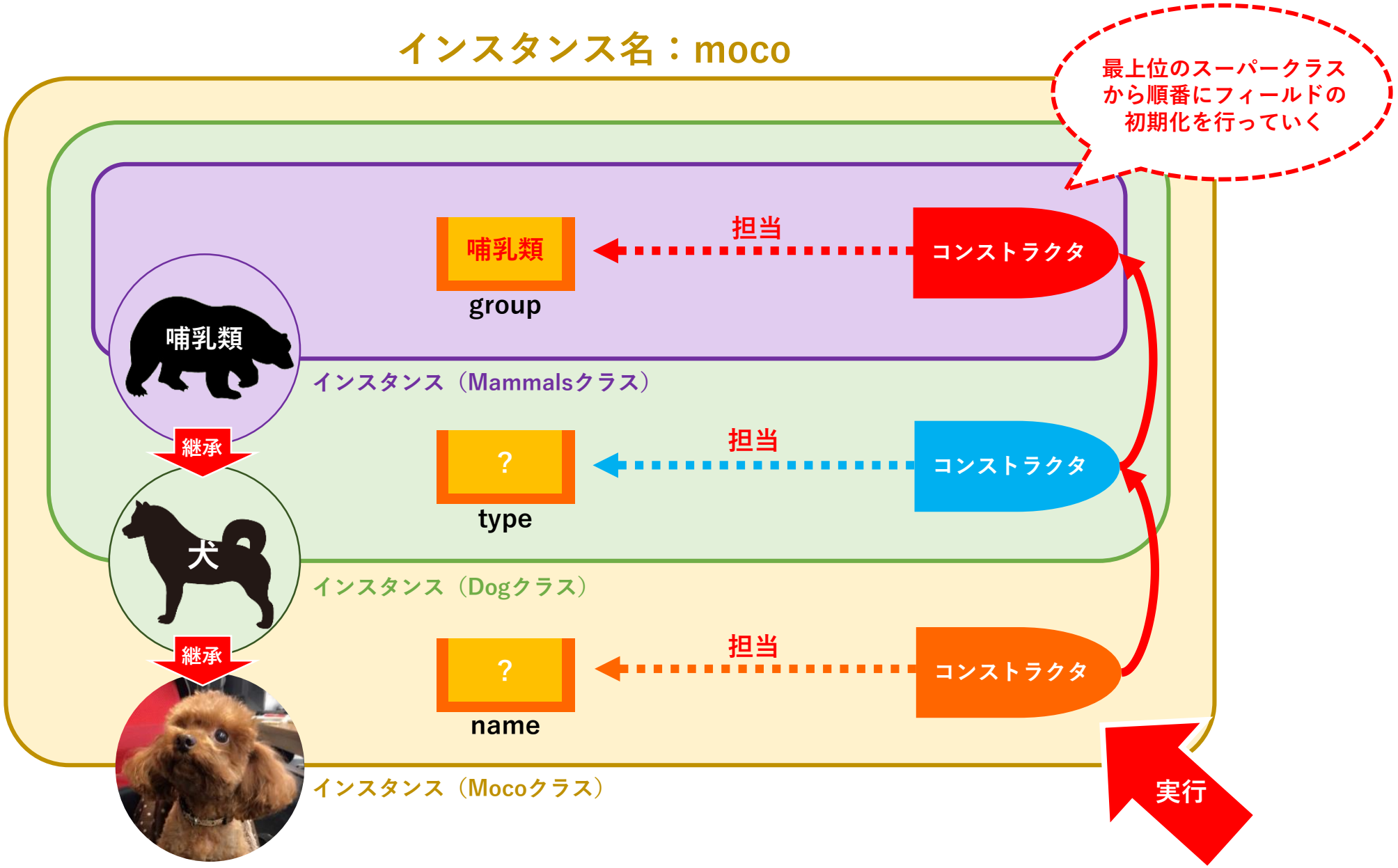
～継承におけるコンストラクタの動き～



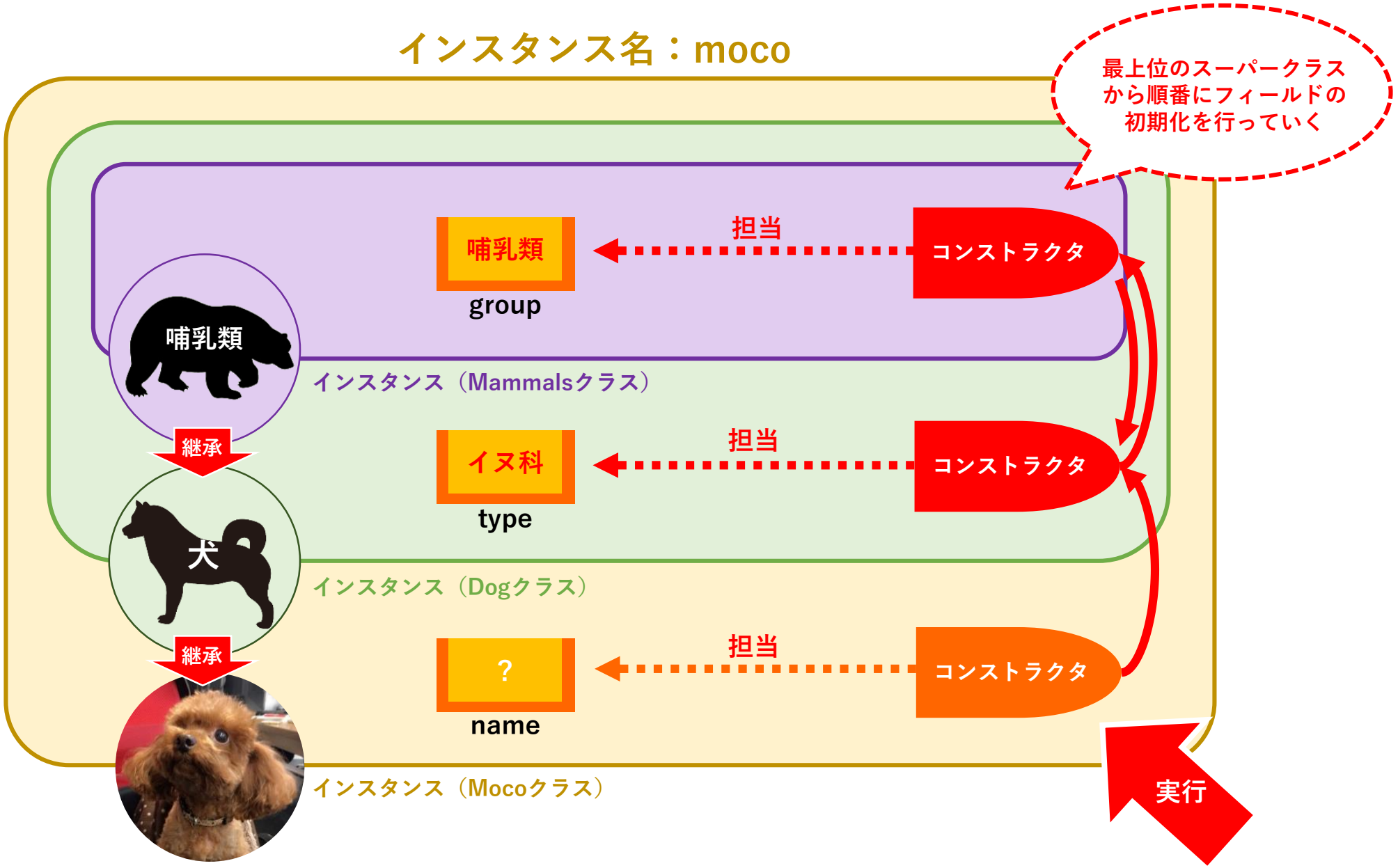
～継承におけるコンストラクタの動き～



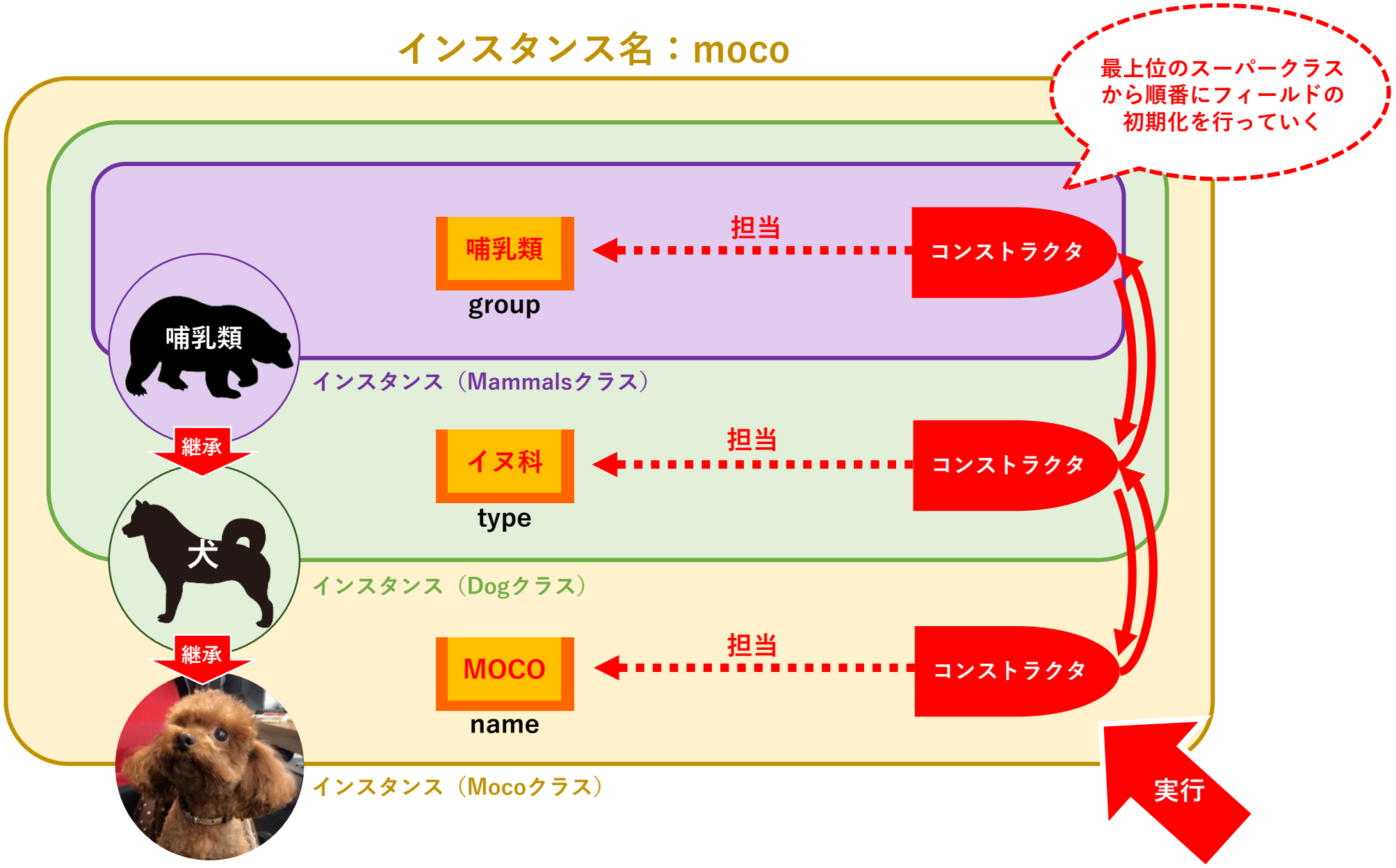
～継承におけるコンストラクタの動き～



～継承におけるコンストラクタの動き～



～継承におけるコンストラクタの動き～

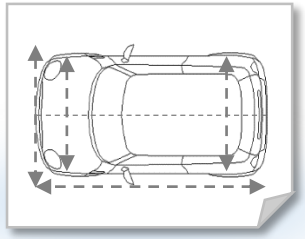


～コンストラクタ～

復習

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　：？

実体と言える？

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

属性あっての実体！

～継承におけるコンストラクタの動き～

サブクラス

JapaneseGalクラス
（日本人のギャル）

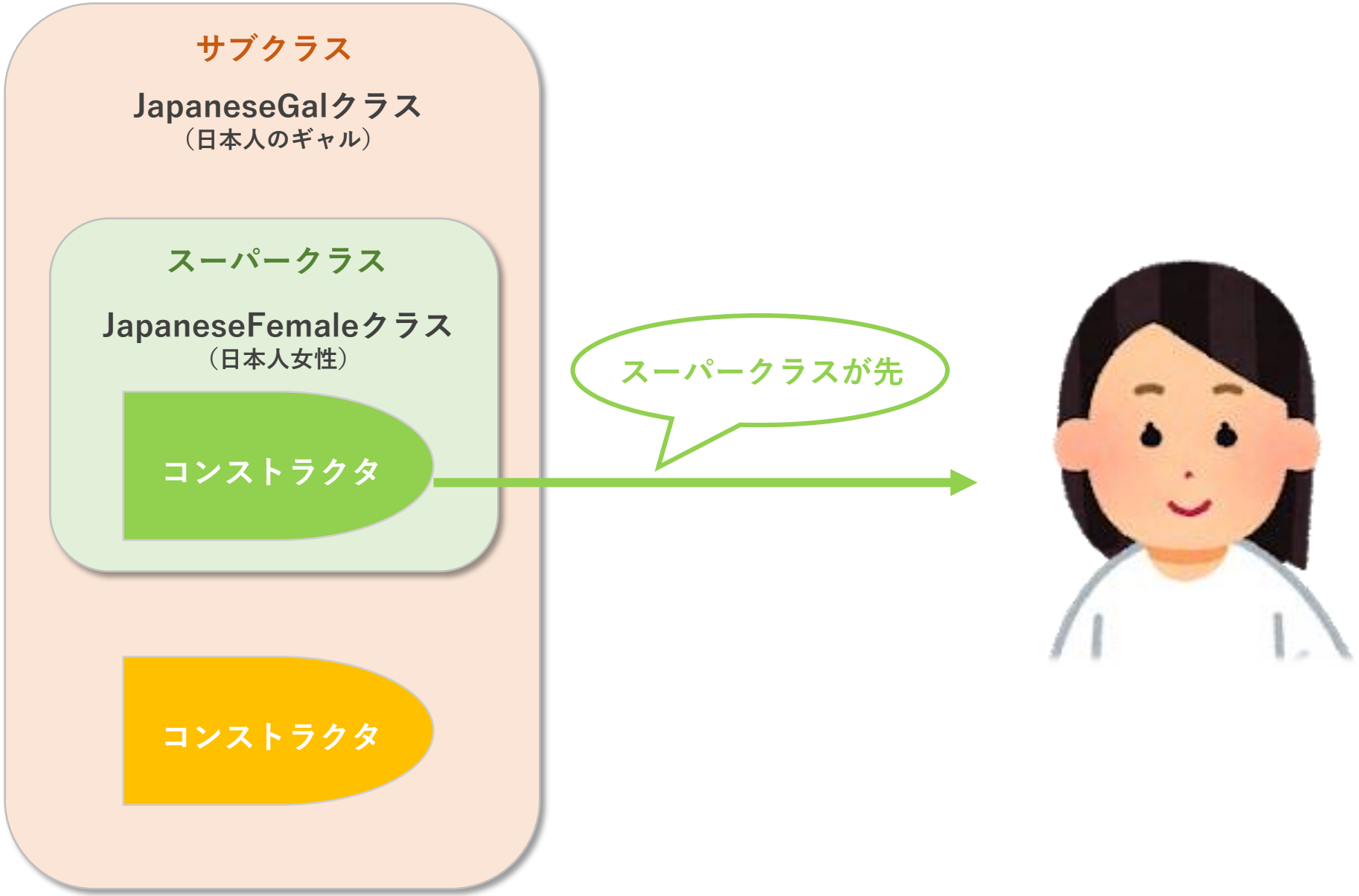
スーパークラス

JapaneseFemaleクラス
（日本人女性）

コンストラクタ

コンストラクタ

～継承におけるコンストラクタの動き～



～継承におけるコンストラクタの動き～

サブクラス

JapaneseGalクラス
（日本人のギャル）

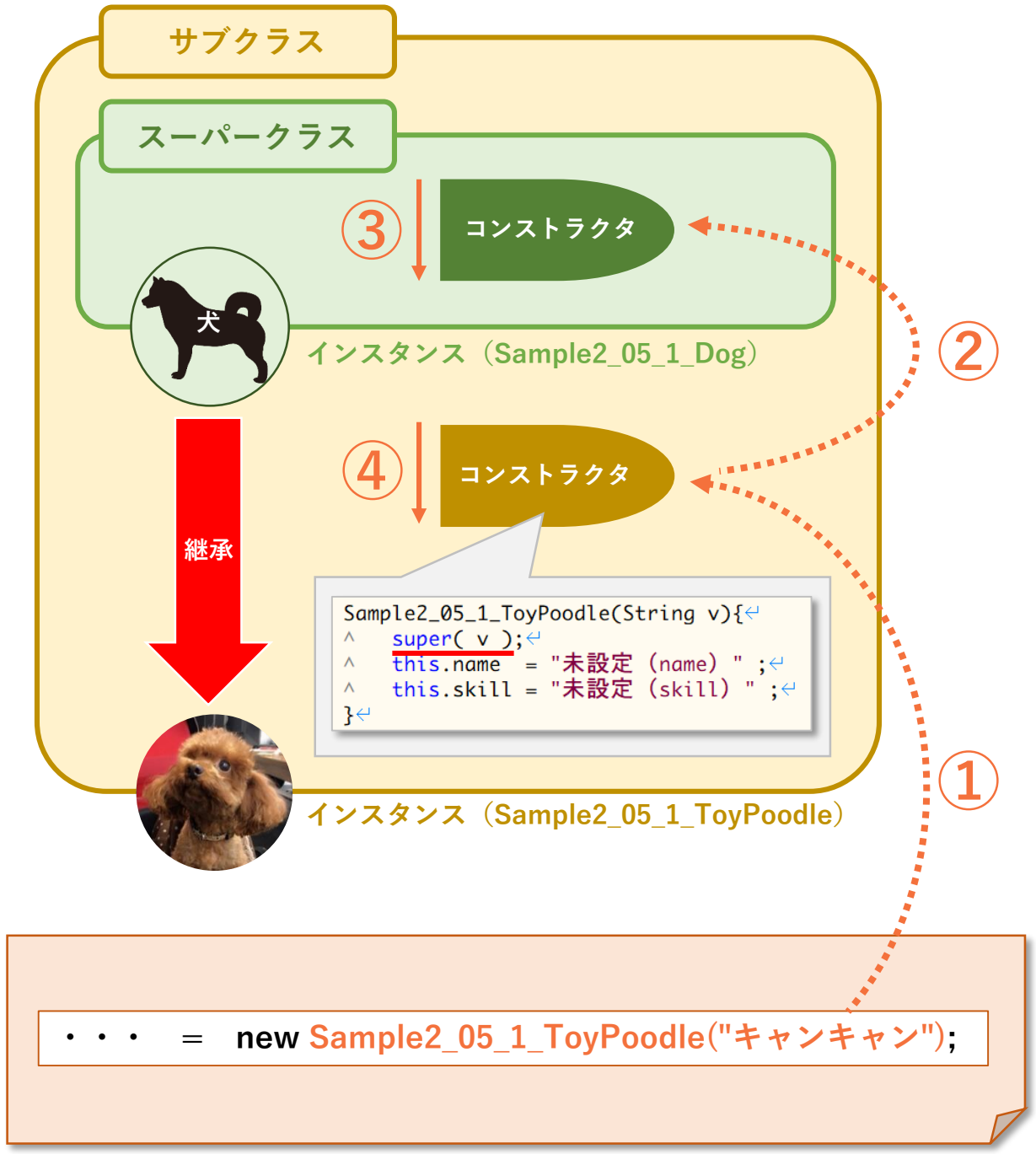
スーパークラス

JapaneseFemaleクラス
（日本人女性）

コンストラクタ

コンストラクタ

サブクラスが後



≪継承関係を持つクラスのコンストラクタの動き≫

□**コンストラクタはそのクラス固有の特殊なメソッドであるため継承はされません。**
インスタンス化の際に**フィールドに対して初期値を設定する担当はあくまでそのクラスのコンストラクタ**であり、スーパークラスのフィールドを初期化したければそのクラスのコンストラクタを呼び出す必要があります。

□**super(引数)**を用いることでスーパークラスのコンストラクタをサブクラス内で明示的に呼び出すことが可能です。

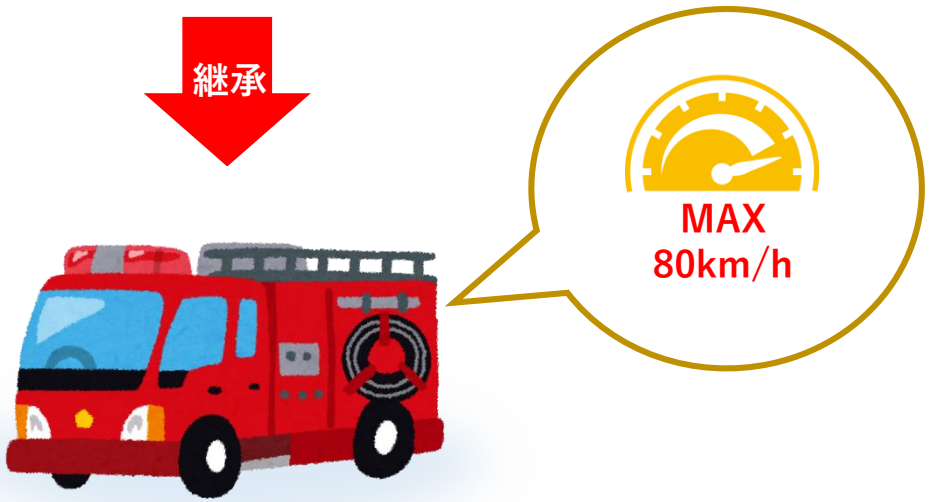
□継承関係を持つクラスをインスタンス化する際、**コンストラクタは必ず上位クラスのものから順に呼び出されねばなりません。**
この動作を実現させるため、インスタンス化されたクラス（サブクラス）のコンストラクタは**その先頭でスーパークラスのコンストラクタの呼び出し処理を必ず行わなければなりません。**

□コンストラクタの中で**他のコンストラクタを呼び出す際はその先頭で呼び出さなければいけない**という仕様であるため、**1つのコンストラクタ内でsuperやthisを2つ以上使用できない**点にご注意ください。

□サブクラスのコンストラクタでスーパークラスの呼び出し処理を記述しなかった場合、処理の先頭で**暗黙的に『 super(); 』（引数なしのスーパークラスのコンストラクタの呼び出し）が実行されます。**

スーパークラス

継承



サブクラス

≪オーバーライド≫

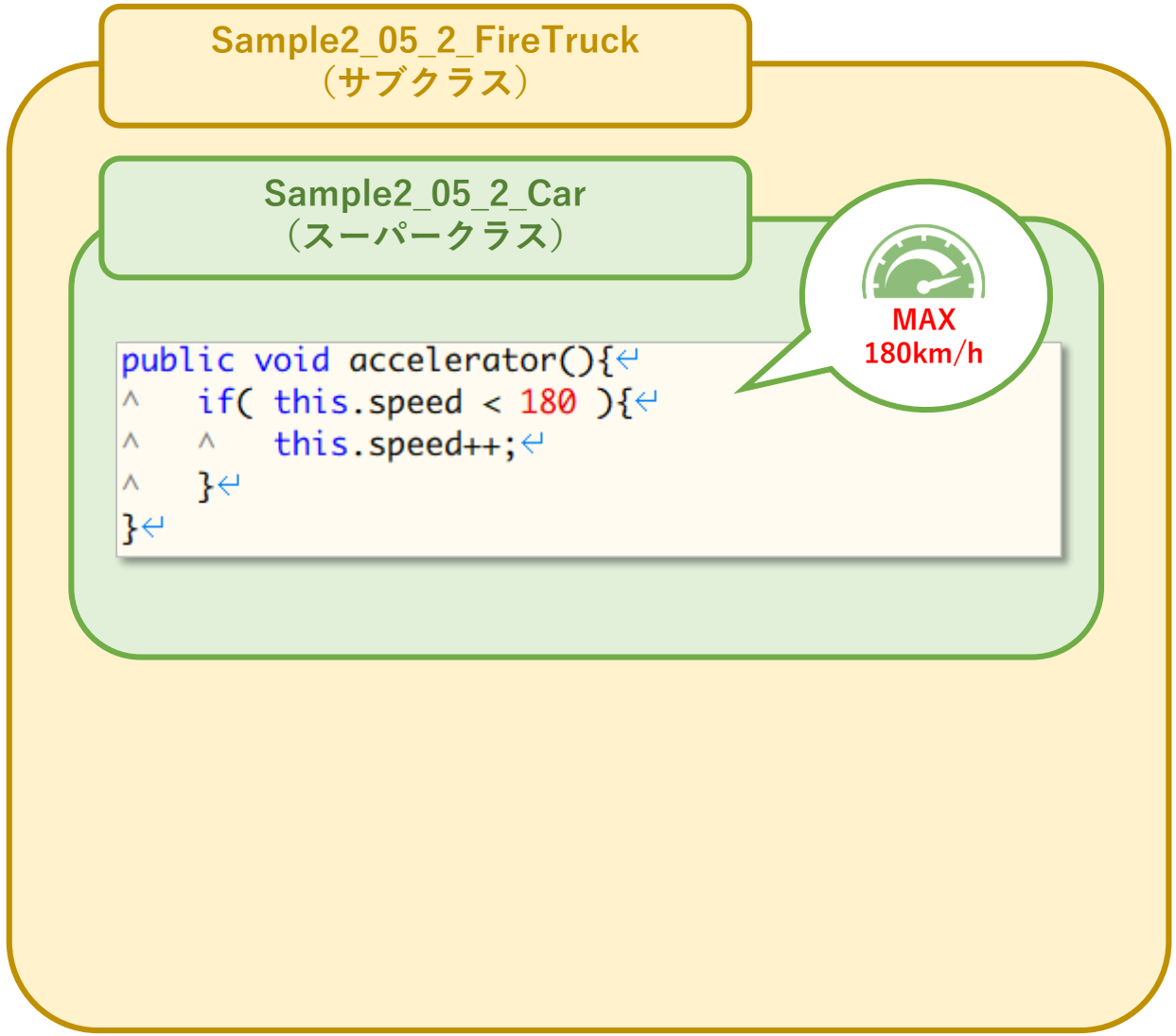
□スーパークラスのメンバをサブクラス側で同じ名前で再定義することを**オーバーライド**と言います。
メソッドをオーバーライドする際のルールは以下のとおりです。

- **オーバーライドするスーパークラスのメソッドとメソッド名、戻り値の型、引数の数および型が同じである**
- **スーパークラスのメソッドとアクセス制御が同じか緩い**

□オーバーライドをするとスーパークラスとサブクラスで同じ名前のメンバが存在することになります。
この場合**サブクラス側で定義されたものが優先される**という仕様であるため、スーパークラス側のメンバは外部から見えなくなります。
（この作用のことを隠蔽と言います。）

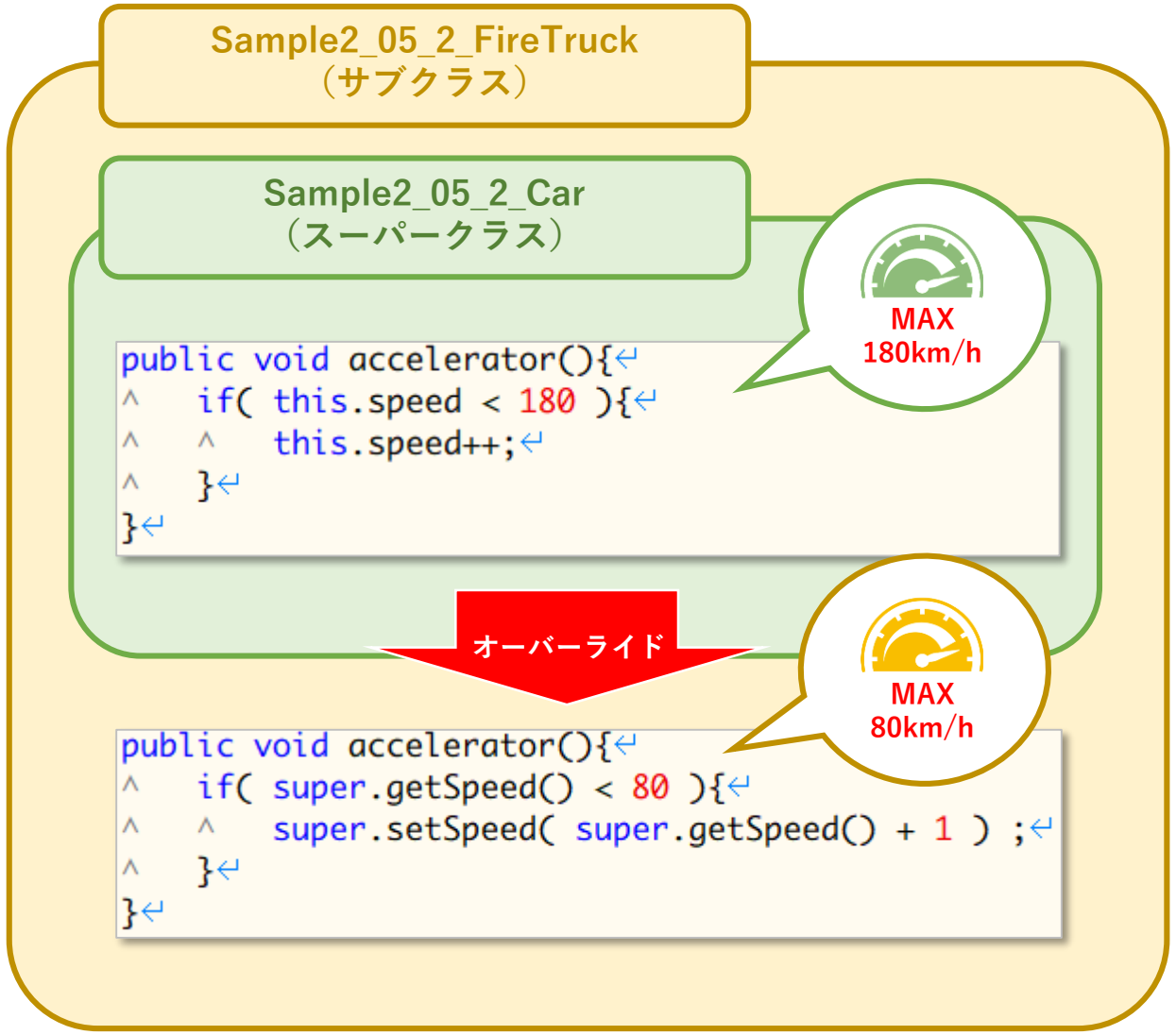
□**スーパークラスのメソッドにfinal修飾子をつけるとオーバーライド不可にすることが可能**です。
同様にクラスにfinal修飾子を用いることで**クラス自体を継承不可**にすることが可能になります。
なお、final修飾子つきのフィールド（定数）はオーバーライド可能です。

～オーバーライドのしくみ～



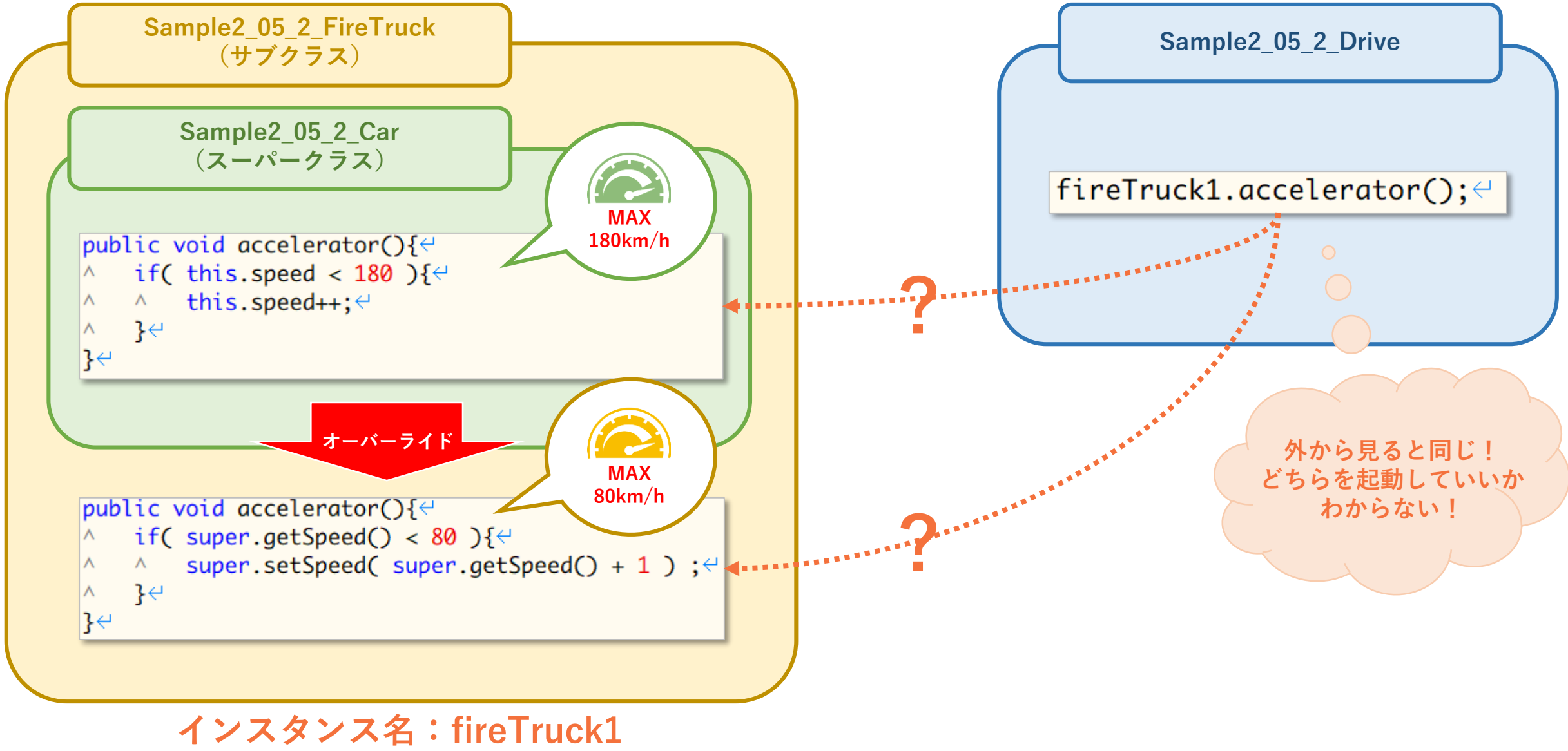
インスタンス名：fireTruck1

～オーバーライドのしくみ～

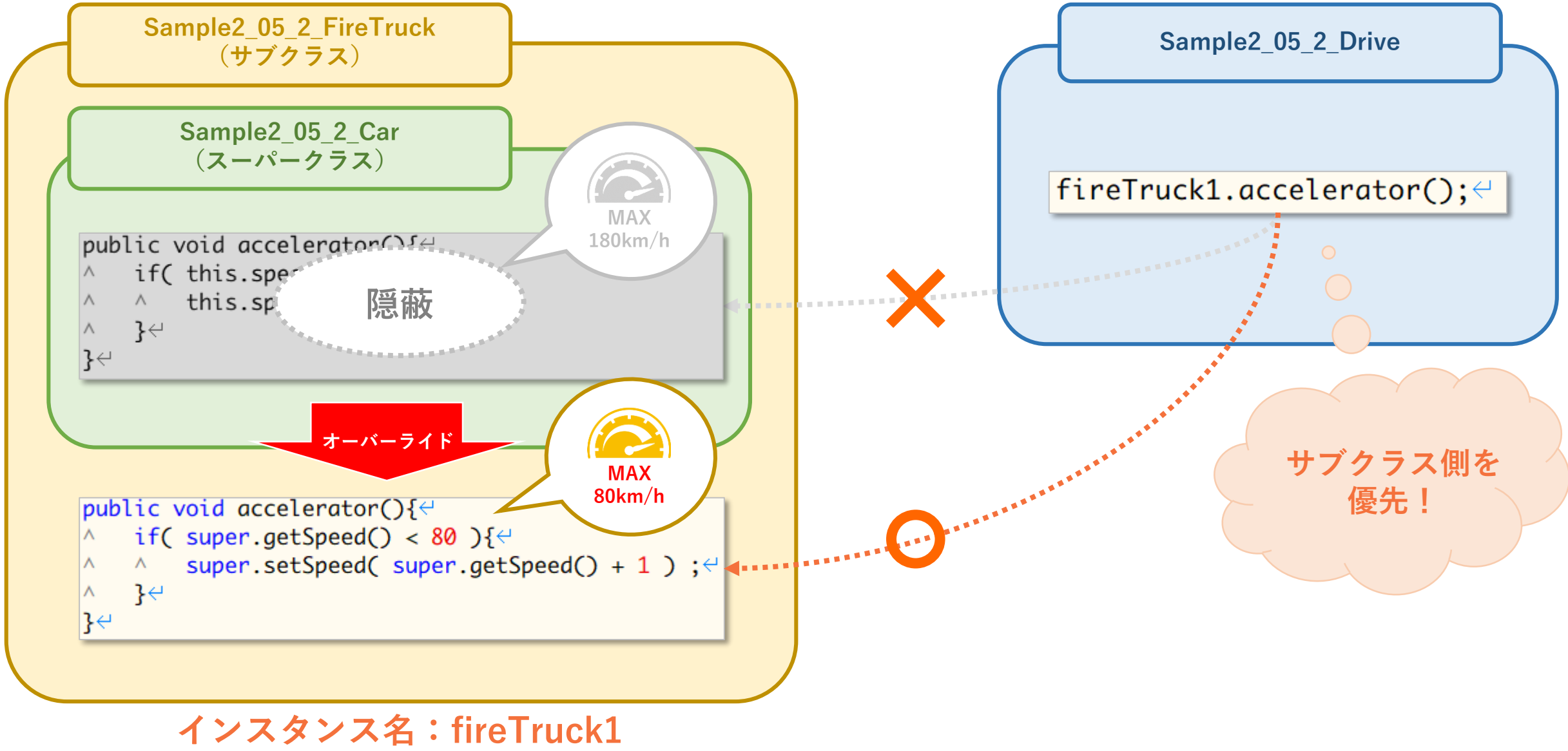


インスタンス名：fireTruck1

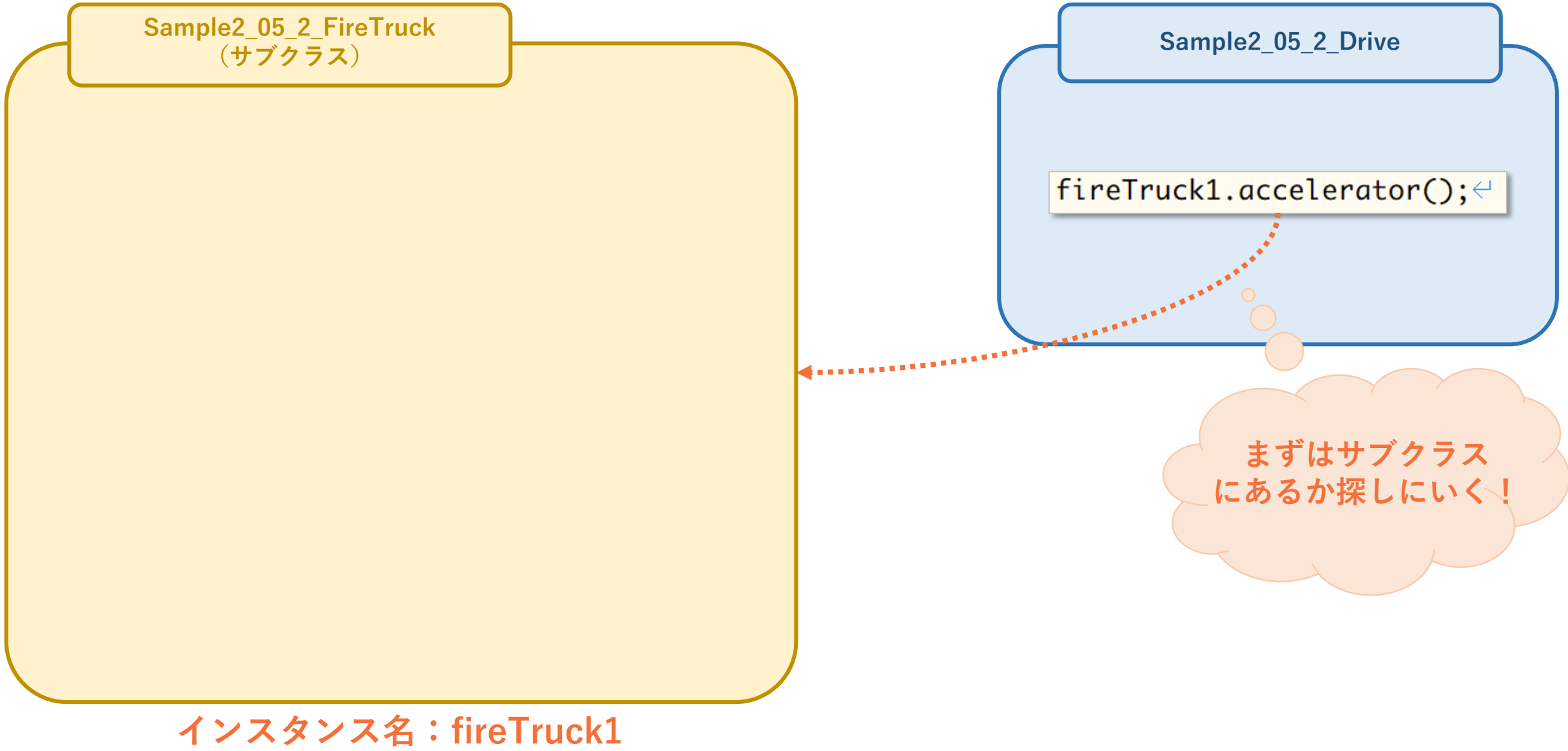
～オーバーライドのしくみ～



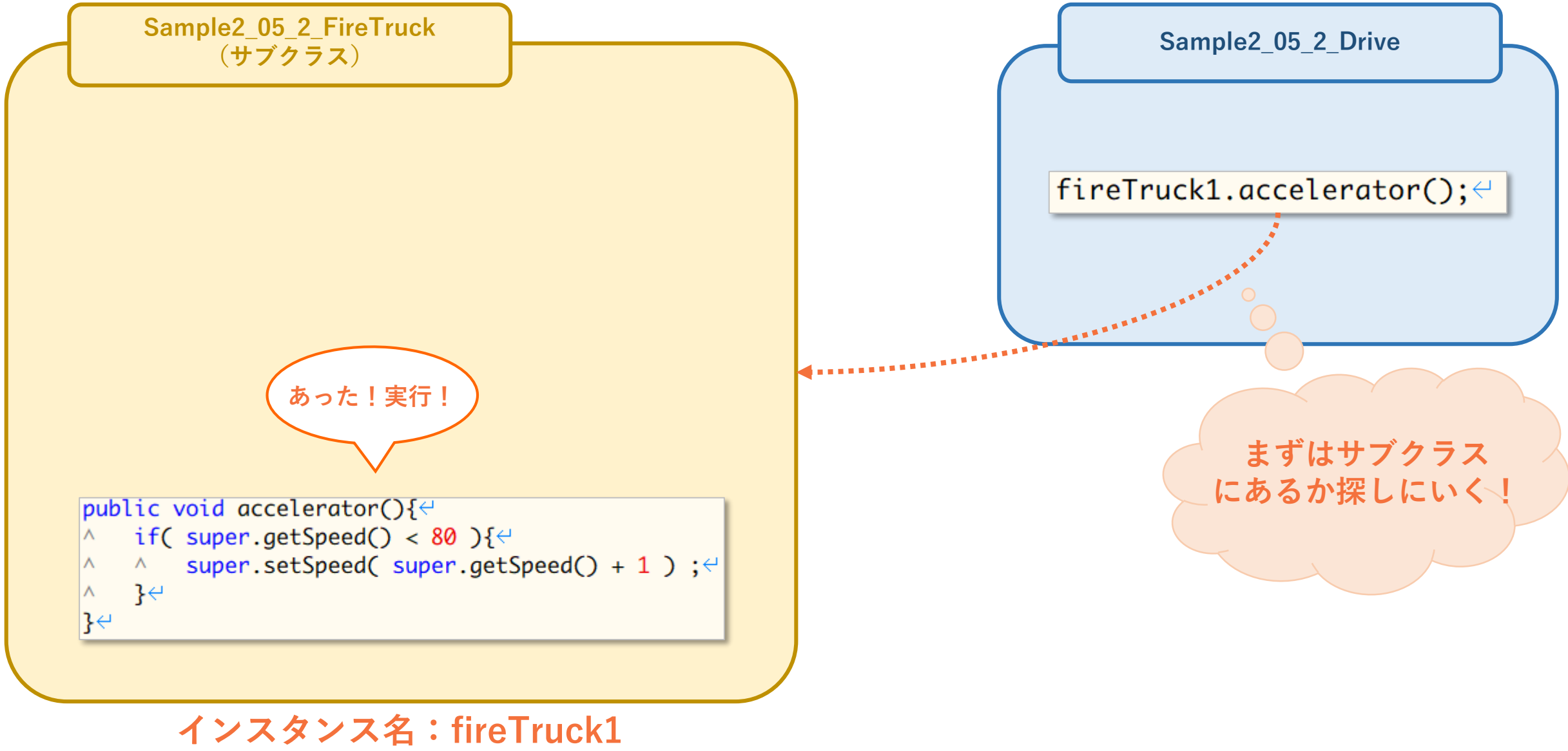
～オーバーライドのしくみ～



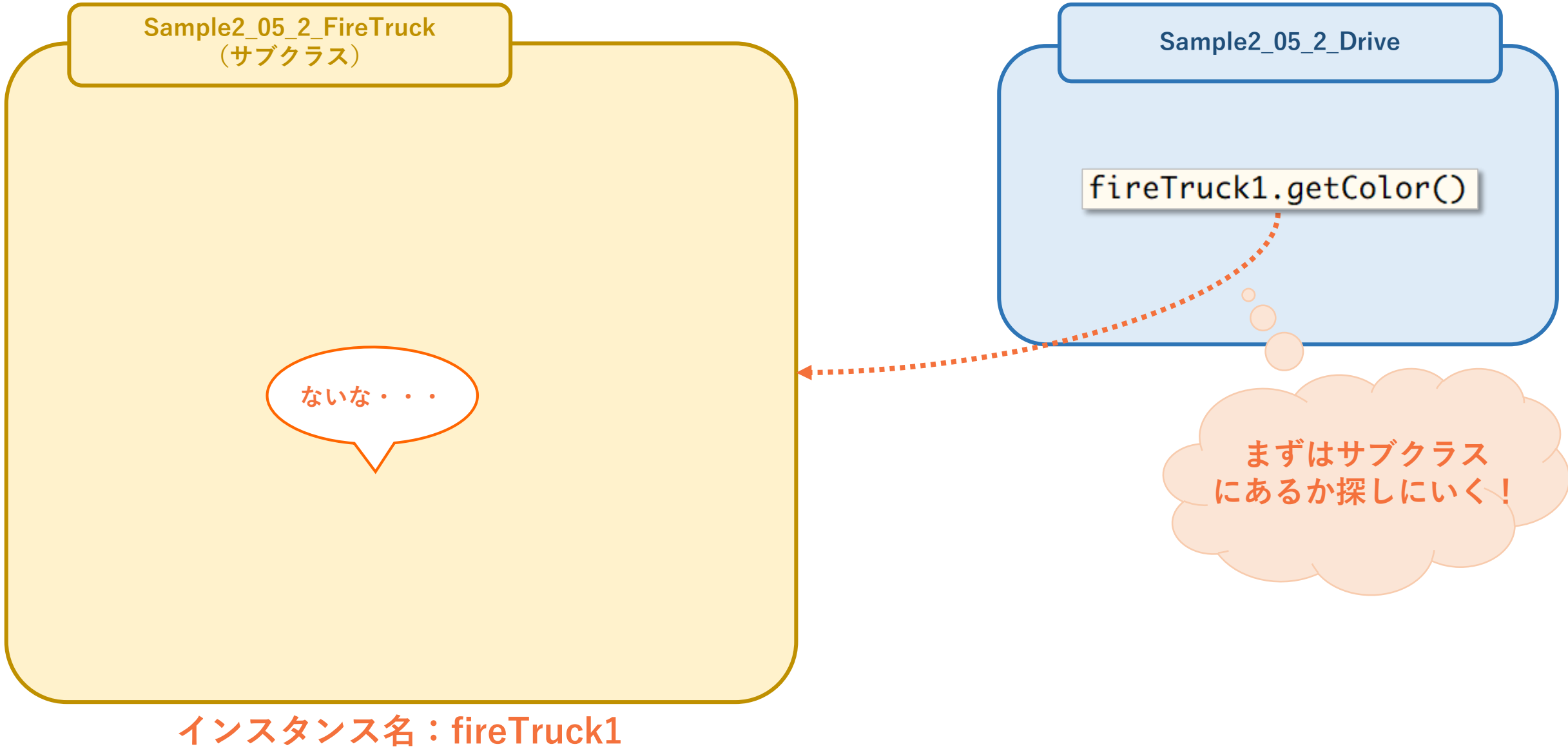
～継承関係を持つクラスのメンバの呼び出しのイメージ～



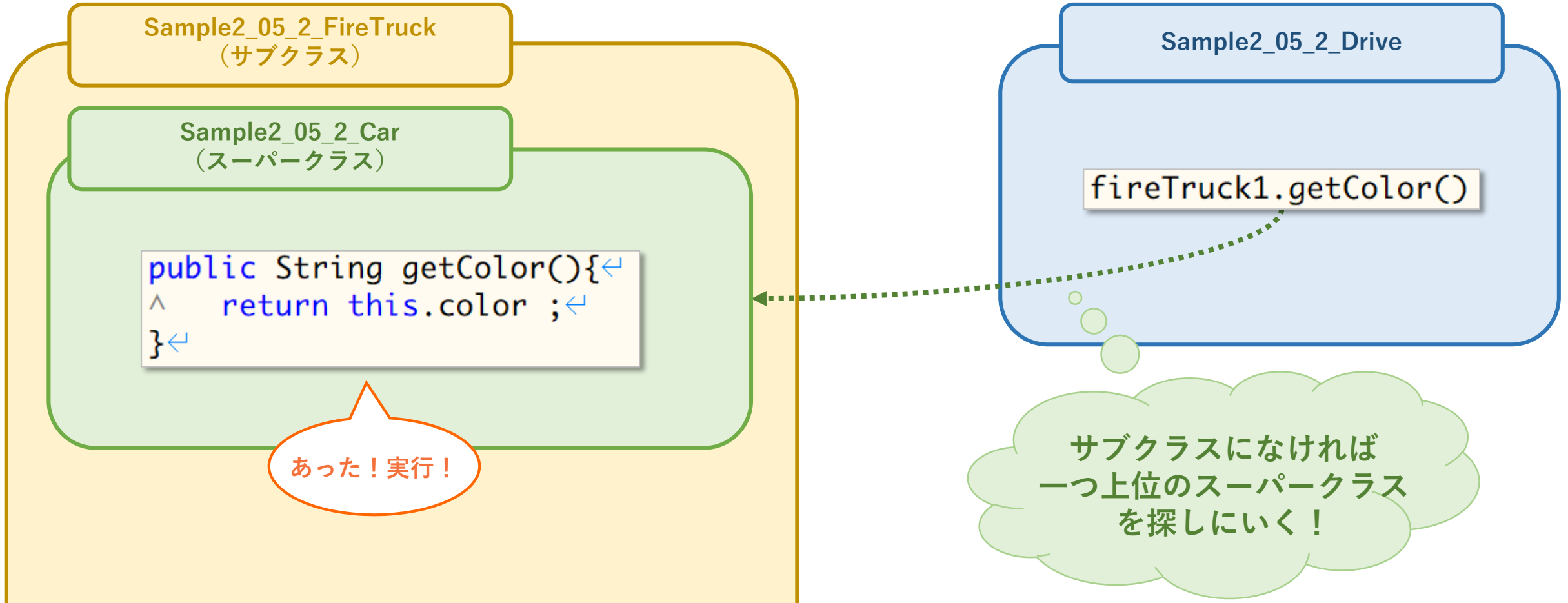
～継承関係を持つクラスのメンバの呼び出しのイメージ～

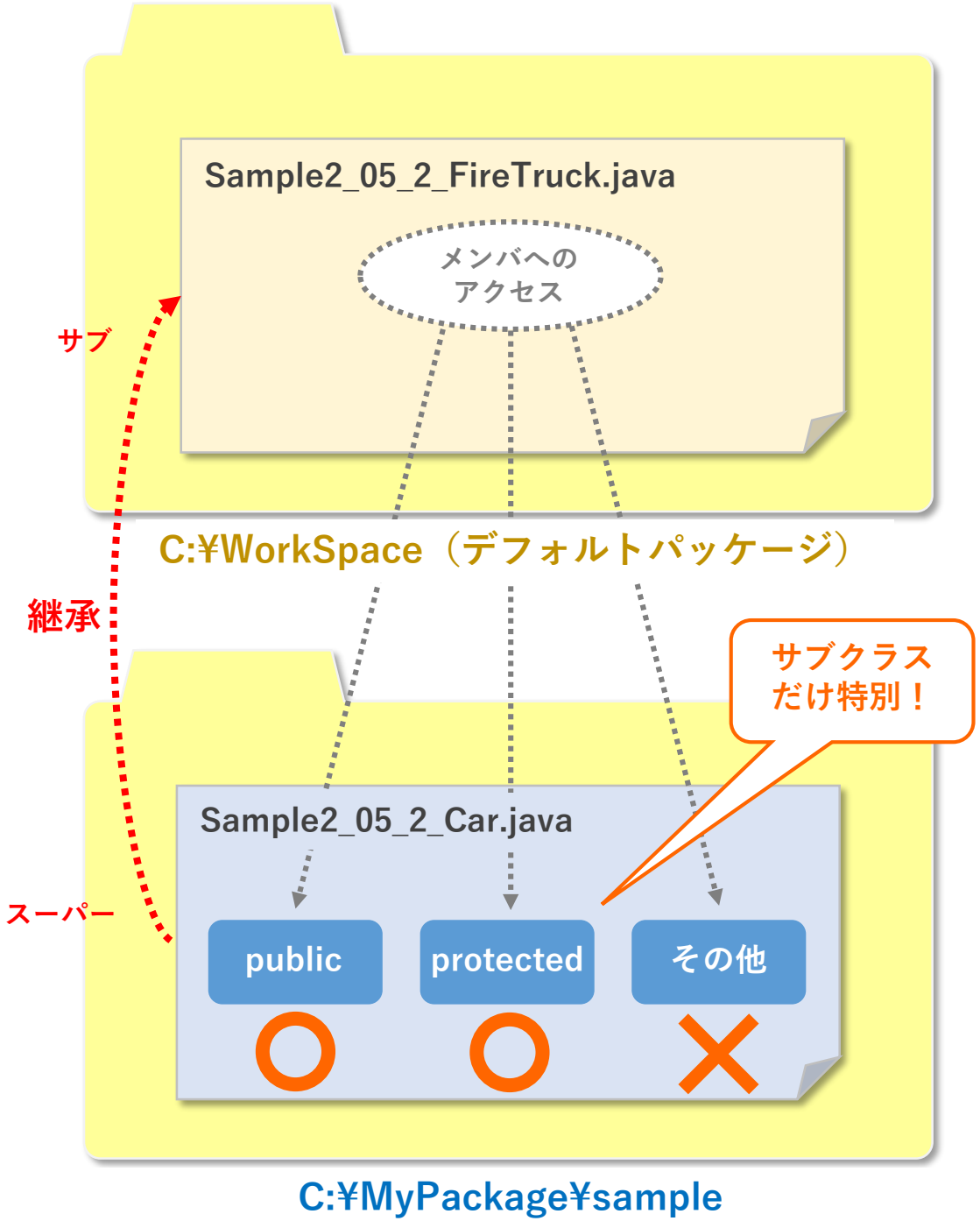


～継承関係を持つクラスのメンバの呼び出しのイメージ～



～継承関係を持つクラスのメンバの呼び出しのイメージ～





≪protected≫

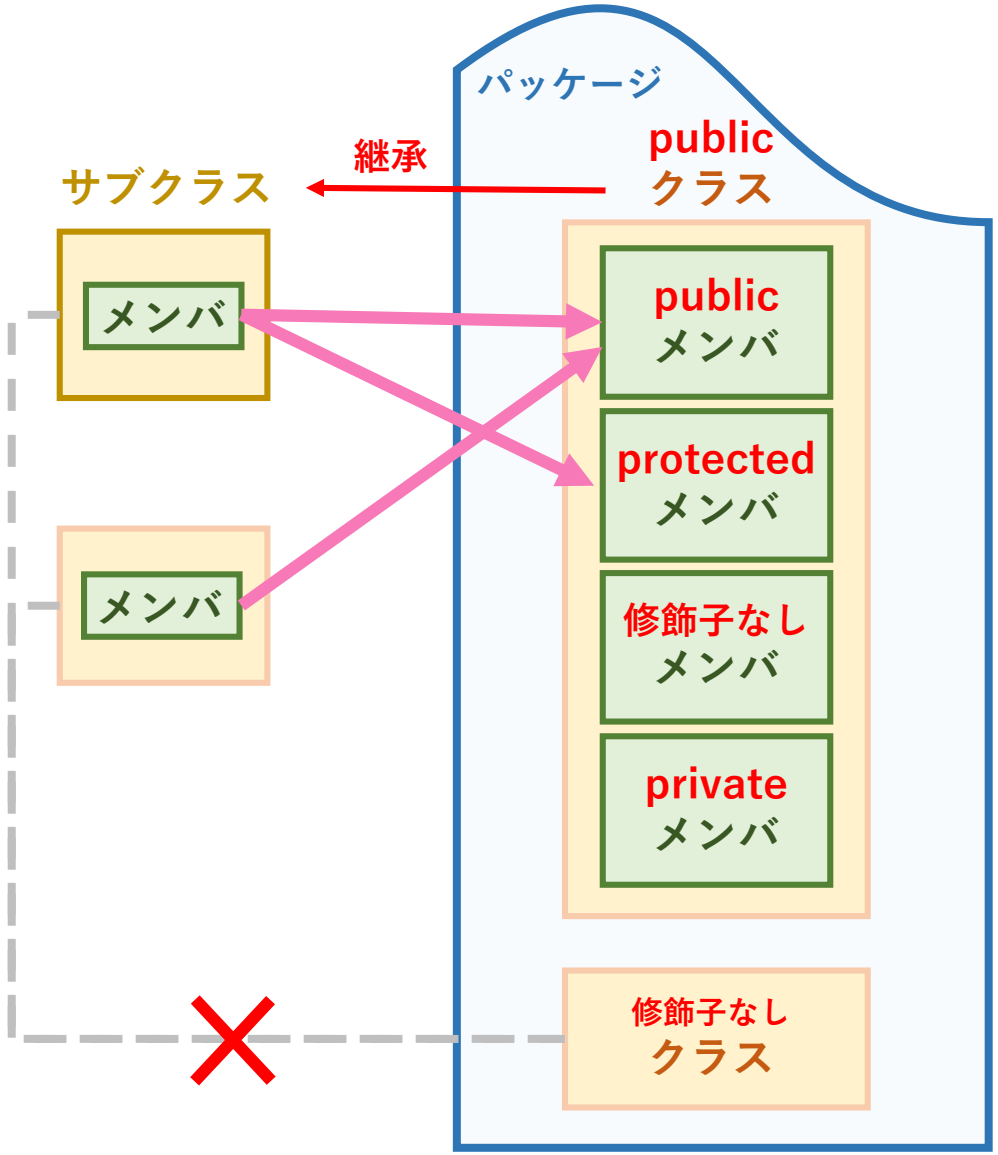
□protectedはアクセス修飾子の一種です。
パッケージを跨いだクラスにアクセスする場合、基本的にpublic修飾子のついたメンバにしかアクセスができませんが、
サブクラスからスーパークラスへのアクセスの場合は、protected修飾子のついたメンバへのアクセスが可能になります。

□privateなフィールドへのアクセスは例えサブクラスであっても通常getter/setterを用いなければなりませんが、protectedをつけておけばサブクラスのみを特別扱いするアクセス制御が可能になります。
特にオーバーライドと組み合わせると効果的です。

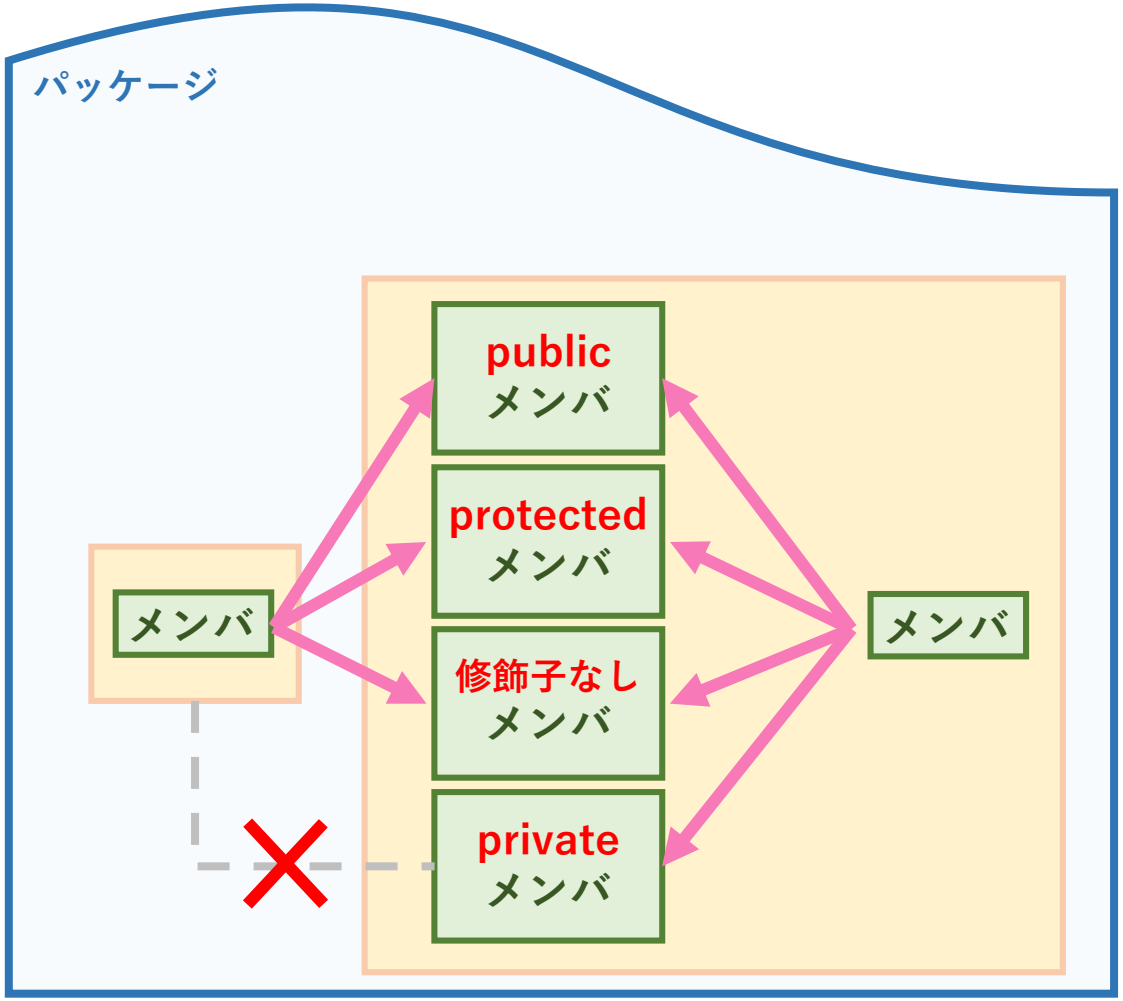
～Javaにおけるアクセス制御～

復習

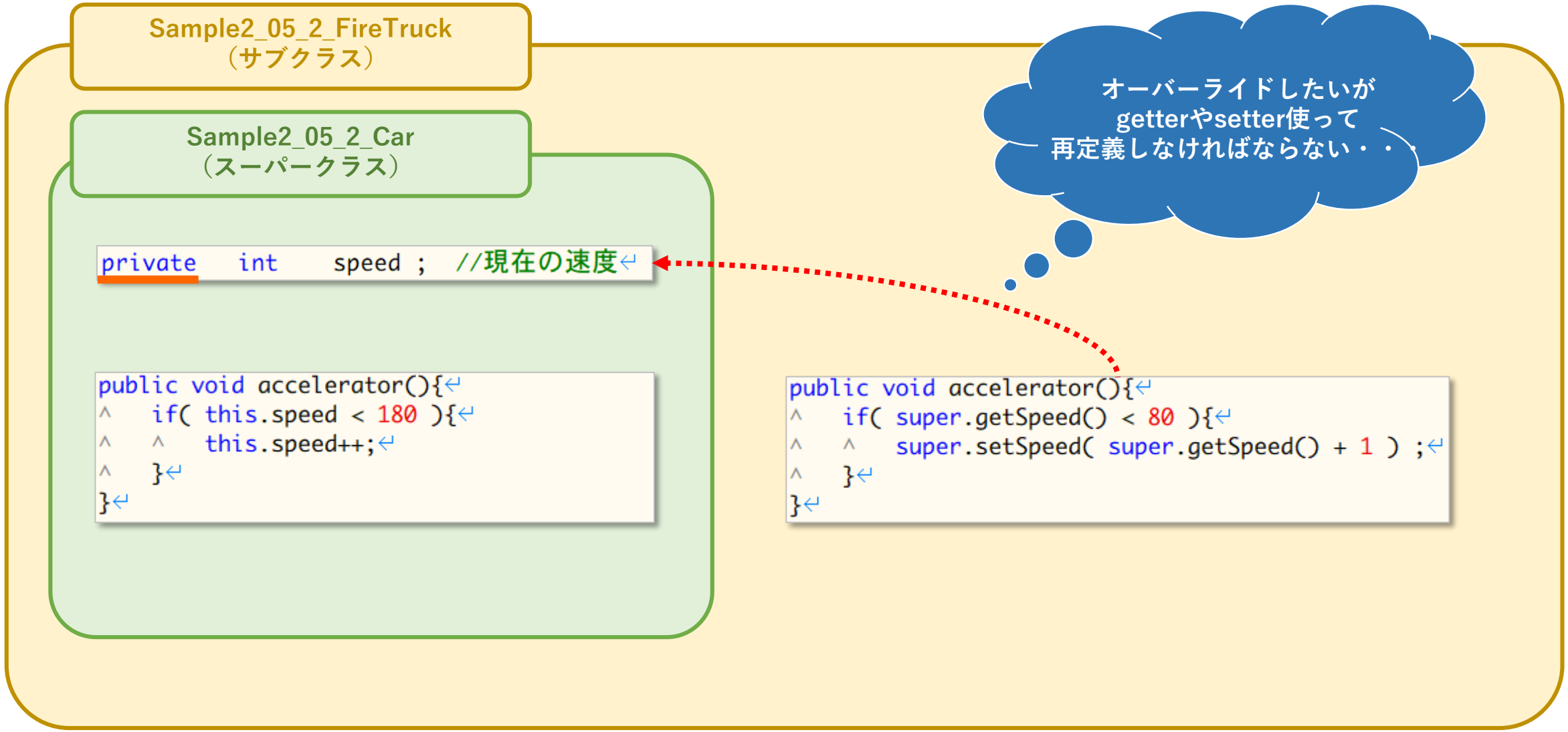
パッケージ外からのアクセスルート



同パッケージ内におけるアクセスルート



～privateへのアクセスはたとえサブクラスでも大変！～



インスタンス名：fireTruck1

～privateへのアクセスはたとえサブクラスでも大変！～

Sample2_05_2_FireTruck
（サブクラス）

Sample2_05_2_Car
（スーパークラス）

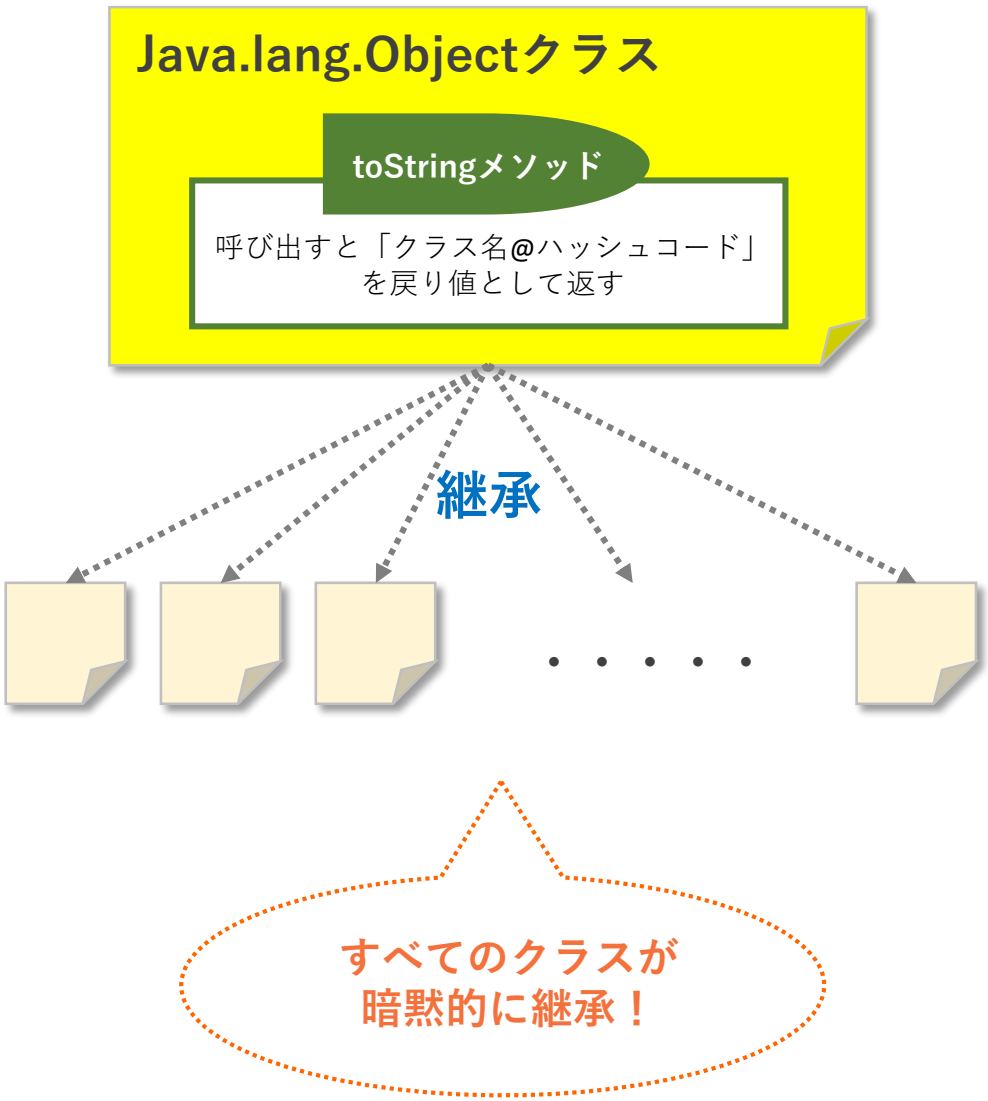
```
protected String color ;  //塗装色
```

```
public void paint( String cl ){
    if( cl.equals( "WHITE" ) || cl.equals( "BLACK" )|| cl.equals( "RED" ) ){
        this.color = cl ;
    }
}
```

自クラスのフィールドかのように
扱えるためオーバーライドも
らくちん！

```
public void paint( String cl ){
    if( cl.equals( "RED" ) ){
        super.color = cl ;
    }
}
```

インスタンス名：fireTruck1



≪toStringメソッド≫

□実は**Javaの全クラスはJava.lang.Objectというクラスを暗黙的に継承**しており、このObjectクラスで定義されているメソッドはどんなインスタンスでも使用可能な状態になっています。

□**toStringメソッド**はObjectクラスで定義されているメソッドの一つで、呼び出すと**インスタンスを文字列として表現した結果を返す**機能を持ちます。

□Objectクラスで定義されているtoStringメソッドは「クラス名@ハッシュコード（インスタンスの場所情報）」というあまり役に立たない文字列を戻り値として返します。

toStringメソッドをオーバーライドし、そのクラス固有の情報を戻り値として返すよう再定義することで初めて「インスタンスを文字列として表現した結果を返す」という役割を果たせます。
（toStringメソッドは公式にもすべてのクラスでオーバーライドすることが推奨されています。）

□System.out.printlnメソッドにインスタンスを引数として渡すと内部でtoStringメソッドを呼び出した戻り値を画面に表示します。**疑似プリミティブ型の変数（厳密にはインスタンス）をprintlnメソッドに引数として渡すと適切な値を表示してくれるのはこのため**です。

～インスタンスは何とも表現しがたいもの～

引数としてprintlnメソッドに値渡し

System.out.println(moco) ;

何を表示していいか
わからないよ…

printlnメソッド

インスタンス
（moco）

～オリジナルのtoStringは役立たずメソッド！？～

▼実際のソースコード（Java.lang.Objectクラス）

```
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

インスタンスのクラス名を取得

ハッシュコードを取得

～オリジナルのtoStringは役立たずメソッド！？～

▼各クラスでオーバーライドして使用する！

```
public String toString() {
    return returnする文字列は各クラスでオーバーライドして設定する！;
}
```

オーバーライドすれば
インスタンスごとに
好きな文字列を設定できるね！

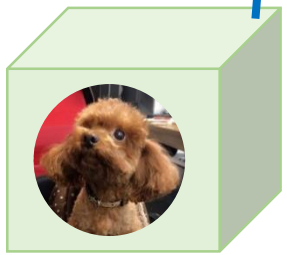
～インスタンスを受け取ったときのprintlnの動作～

引数としてprintlnメソッドに値渡し

System.out.println(moco) ;

toStringメソッドをオーバーライド

```
public String toString(){
   return “モコです” ;
}
```



インスタンス
（moco）

printlnメソッド

≪ 引数がObject型（インスタンス）の場合 ≫

インスタンス名.toString()の戻り値を画面に出力

～インスタンスを受け取ったときのprintlnの動作～

引数としてprintlnメソッドに値渡し

System.out.println(moco) ;

疑似プリミティブ型

printlnメソッド

まるでプリミティブ型を
扱っているかのように見える

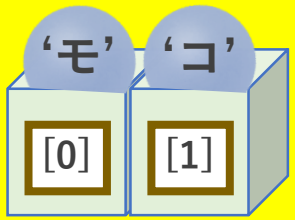
“モコです”

インスタンス
（moco）

モコです

～インスタンスを受け取ったときのprintlnの動作～

```
public String toString(){
   return 配列の要素を繋げて作成した文字列 ;
}
```



char型配列

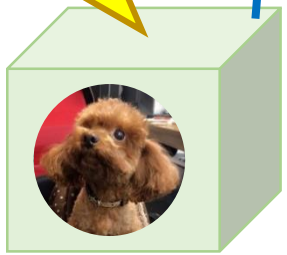
引数としてprintlnメソッドに値渡し

```
System.out.println( moco ) ;
```

printlnメソッド

≪ 引数がObject型（インスタンス）の場合 ≫

インスタンス名.toString()の戻り値を画面に出力



Stiring型のインスタンス
（moco）

～toStringメソッドの使い方～

Sample2_05_3_GoldenRetriever

Java.lang.Object（最上位のスーパークラス）

```
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

インスタンスのクラス名を取得

ハッシュコードを取得

インスタンス名：pochi

Sample2_05_3_ToyPoodle

Java.lang.Object（最上位のスーパークラス）

```
public String toString() {
    return getClass().get          toHexString(hashCode());
}
```

隠蔽

インスタンスのクラス名を取得

ハッシュコードを取得

オーバーライド

```
//toStringメソッド（オーバーライド）
public String toString(){
    return "ToyPoodle ( name:" + this.name + " / " + "skill:" + this.skill + " )";
}
```

インスタンス名：moco