

プログラミング基礎

第7回

オブジェクトの初期化

複数の同名メソッドの定義

本日の内容

- コンストラクタ(=オブジェクトの初期化)
- オーバーロード(=複数の同名メソッドの定義)
- this

復習

Vector2Dクラスを振り返る.

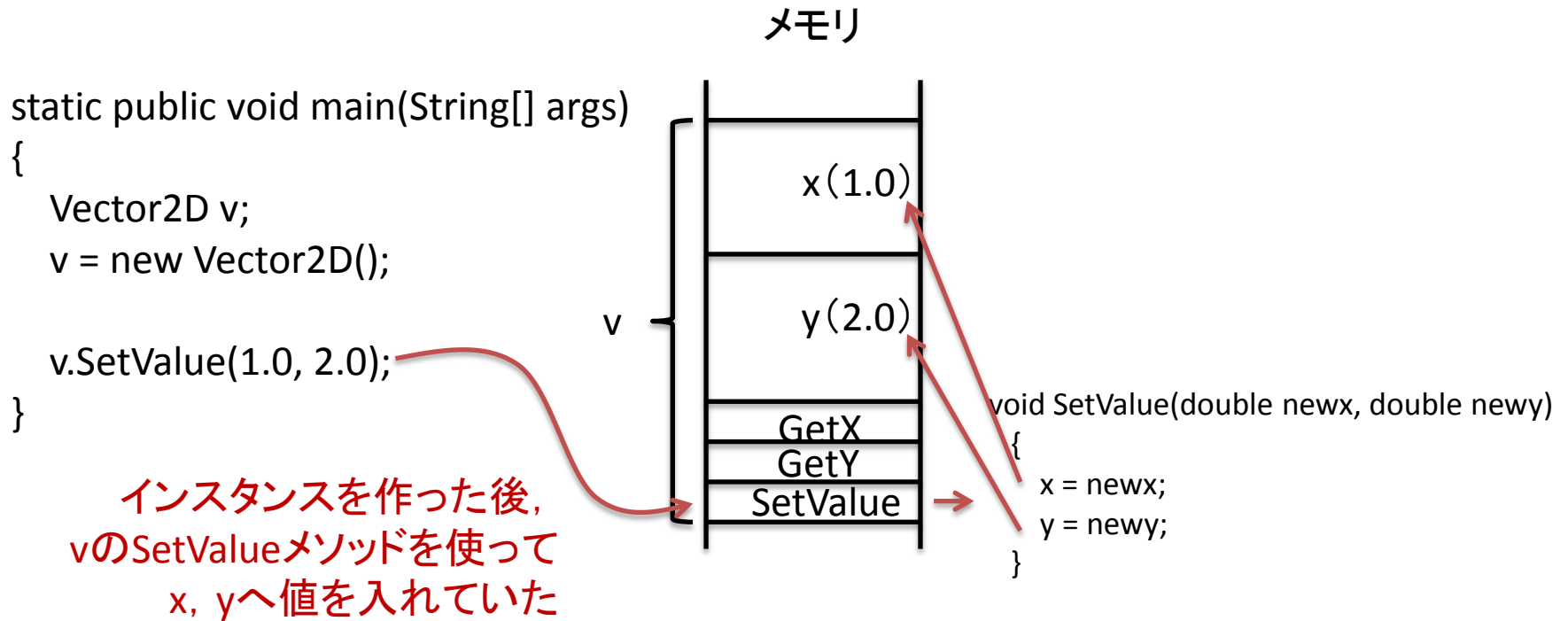
クラス

```
class Vector2D
{
    属性 { double x;
          double y;
          double getX()
          {
              return x;
          }
          double getY()
          {
              return y;
          }
          void setValue(double xx, double yy)
          {
              x = xx;
              y = yy;
          }
    }
}
```

メソッド

復習

Vector2Dクラスを振り返る.



コンストラクタ

Vector2Dクラスのインスタンスを作るのと同時に,
x, yに値を入りたい(初期化したい)



コンストラクタの利用

コンストラクタ

- オブジェクト(=インスタンス)
作成時に自動的に実行される特殊なメソッド

コンストラクタ

- 書き方 (基本はメソッドと同じ)

メソッドの名前は必ずクラス名とする

通常の方法とは
異なり、戻り値の型を
書かないことに注意

クラス名 (引数1の型 引数名1, 引数2の型 引数名2, ...)

```
{  
    //処理内容を記述  
}
```

[Vector2Dクラスでのコンストラクタ例]

```
Vector2D(double xx, double yy)  
{  
    x = xx;  
    y = yy;  
}
```

コンストラクタ

- 使い方

```
static public void main(String[] args)
{
    Vector2D v;
    v = new Vector2D(1.0, 2.0);
}
```

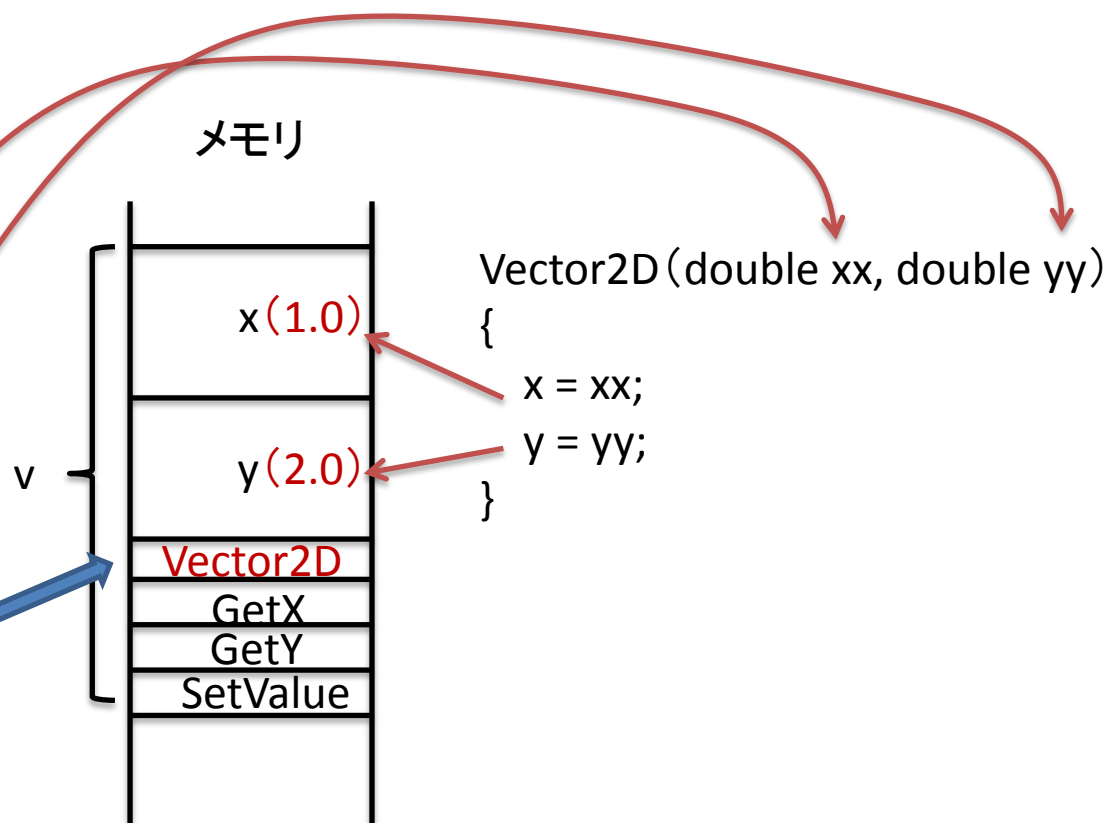

コンストラクタ

• 使い方

```
Vector2D v;  
v = new Vector2D(1.0, 2.0);
```

①newでインスタンスができる

②指定されている引数で、
今作ったインスタンスの
コンストラクタが自動的に
実行される



演習

- コンストラクタを使って, $x=1.0$, $y=2.0$ の
Vector2Dクラスインスタンスを作るプログラム
を作成せよ
プログラム名 : VectorConstructor

デフォルトコンストラクタ

- プログラマがクラスにコンストラクタを1個も書いていない場合には...

```
class Vector2D
{
    double x;
    double y;
}
```

引数なし, 何もしないコンストラクタが勝手に作られている

⇒デフォルトコンストラクタと呼ばれる

⇒実はこれがあつたから `v = new Vector2D();` と今まで書けた

```
class Vector2D
{
    double x;
    double y;
```



```
    Vector2D()
    {
    }
}
```

デフォルトコンストラクタ

- プログラマがクラスにコンストラクタを書くと...

```
class Vector2D
```

```
{  
    double x;  
    double y;
```

```
    Vector2D()  
    {
```

```
    }  
}
```

```
    Vector2D(double xx, double yy)  
    {
```

```
        x = xx;
```

```
        y = yy;
```

```
    }  
}
```

デフォルトコンストラクタ
は作られなくなる



```
class Vector2D
```

```
{  
    double x;  
    double y;
```

```
    Vector2D(double xx, double yy)  
    {
```

```
        x = xx;
```

```
        y = yy;
```

```
    }  
}
```

デフォルトコンストラクタがなくなったので

`v = new Vector2D(1.0, 2.0);` //OK

`v = new Vector2D();` //NG

となるので注意

デフォルトコンストラクタ


```
v = new Vector2D(1.0, 2.0);  
v = new Vector2D();
```

} 両方の書き方を使えるようにしたいのであれば、
プログラマ自身でデフォルトコンストラクタを書いておく

```
class Vector2D  
{  
    double x;  
    double y;
```

```
    Vector2D()  
    {  
    }  
}
```

```
    Vector2D(double xx, double yy)  
    {  
        x = xx;  
        y = yy;  
    }  
}
```



オーバーロード(多重定義)

```
class Vector2D
{
    double x;
    double y;
```

```
    Vector2D()
    {
    }
```

```
    Vector2D(double xx, double yy)
    {
        x = xx;
        y = yy;
    }
}
```

よく見ると同じ名前の
メソッドが2つある

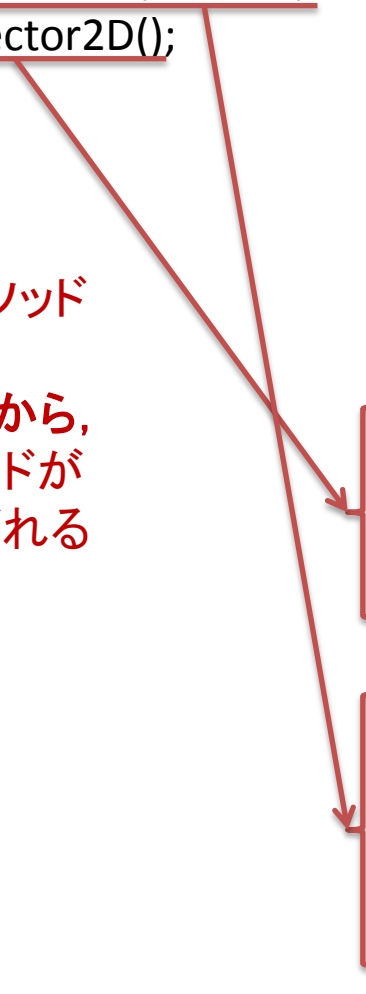
⇒ (メソッドの)オーバーロード
(多重定義)と呼ばれる

オーバーロード(多重定義)

```
v = new Vector2D(1.0, 2.0);  
v = new Vector2D();
```

クラスに
同じ名前のメソッド
が存在しても,
引数の型, 数から,
対応したメソッドが
自動的に呼ばれる

```
class Vector2D  
{  
    double x;  
    double y;  
  
    Vector2D()  
    {  
    }  
  
    Vector2D(double xx, double yy)  
    {  
        x = xx;  
        y = yy;  
    }  
}
```



The diagram illustrates the process of method resolution for the two constructor calls in the code block above. Red arrows originate from the underlined constructor calls and point to the corresponding method definitions in the class. The first arrow points from `new Vector2D(1.0, 2.0);` to the `Vector2D(double xx, double yy)` constructor. The second arrow points from `new Vector2D();` to the `Vector2D()` constructor. This visualizes how the compiler automatically selects the correct method based on the number and types of arguments provided.

オーバーロード(多重定義)

Web資料での例:

// 引数をすべて省略:

//座標(0,0) 半径1の円とする
a.setCoordinates(); **引数がない**

// 座標情報のみ与える:

//座標(5,4) 半径1の円とする
b.setCoordinates(5, 4); **int型の引数2つ**

// 座標情報と半径を与える:

//座標(2,3) 半径2の円とする
c.setCoordinates(2, 3, 2); **int型の引数3つ**

```
class Circle {  
    int x, y;  
    int radius;
```

```
void setCoordinate() {  
    x = 0;  
    y = 0;  
    radius = 1;  
}
```

```
void setCoordinate(int a, int b) {  
    x = a;  
    y = b;  
    radius = 1;  
}
```

```
void setCoordinate(int a, int b, int r) {  
    x = a;  
    y = b;  
    radius = r;  
}
```

```
}
```


演習

以下の実行結果が得られるように、オーバーロードを使用して
Vector2D.setValueメソッドを作成せよ

実行結果:

v=(1.0, 2.0)

v=(3.0, 3.0)

v=(4.0, 5.0)

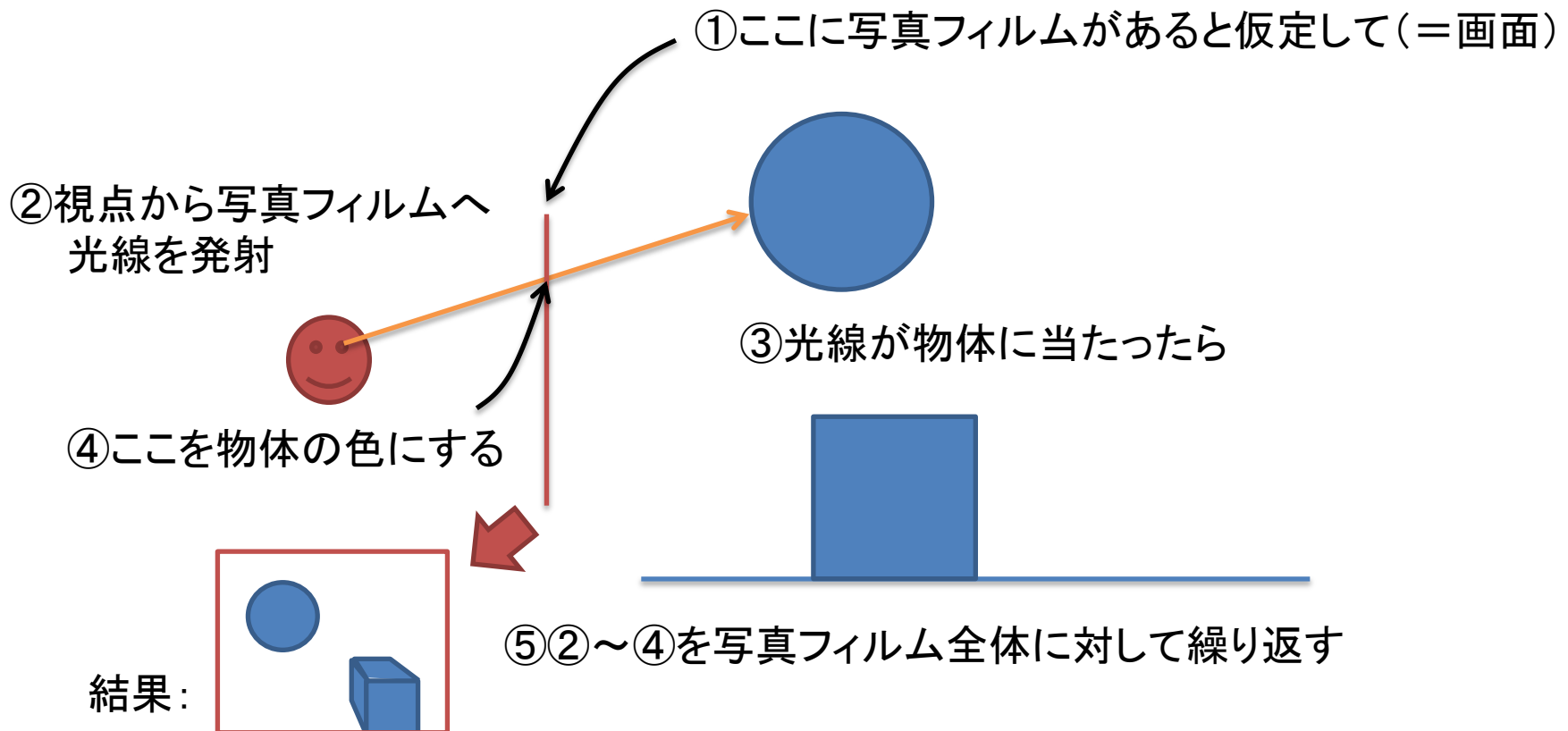
```
public class VectorOverload
{
    static public void main(String[] args)
    {
        Vector2D v;
        v = new v(1.0, 2.0);
        System.out.println("v=(" + v.getX() + ", " + v.getY() + ")");

        v.setValue(3.0); //setValueメソッドで引数が1個の場合には,
                        //x, yをその引数の値とする

        System.out.println("v=(" + v.getX() + ", " + v.getY() + ")");
        v.setValue(4.0, 5.0);
        System.out.println("v=(" + v.getX() + ", " + v.getY() + ")");
    }
}
```

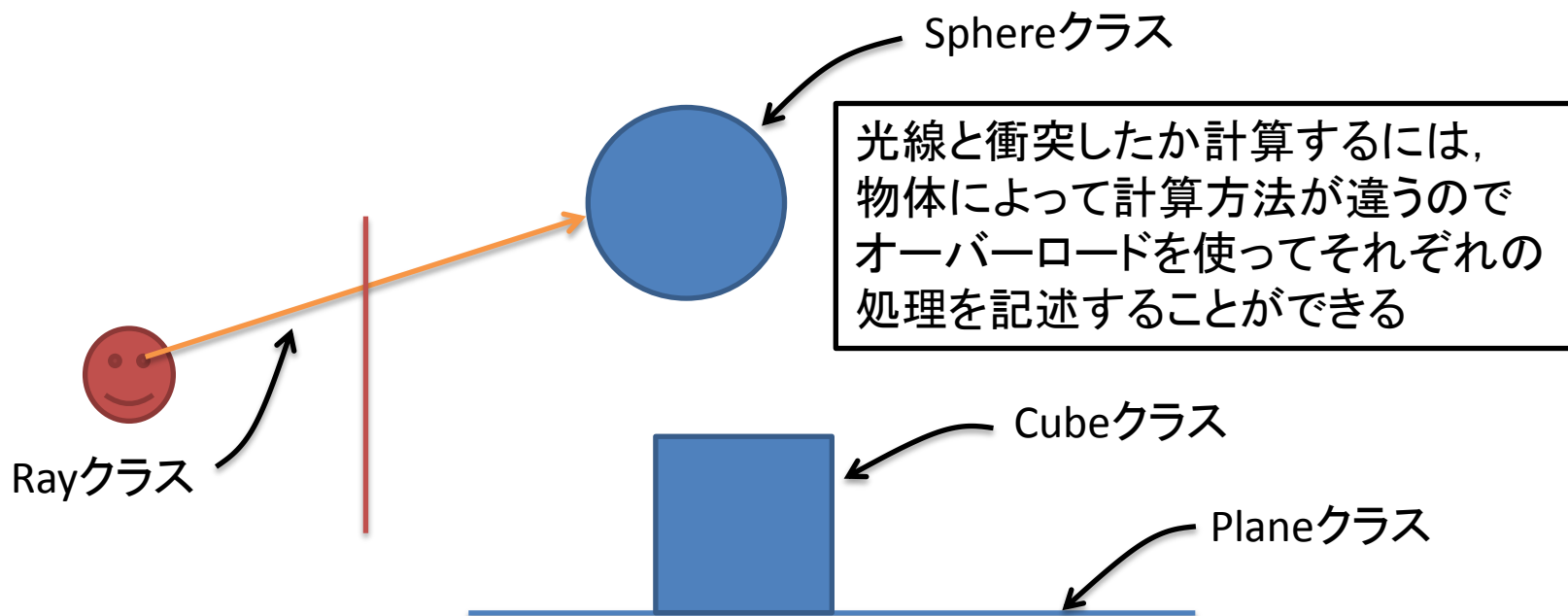
オーバーロードの使いどころ

- グラフィックス関連だと次のような使い方も
レイトレーシング: 3DCGを作るための一手法



オーバーロードの使いどころ

- グラフィックス関連だと次のような使い方もある
レイトレーシング: 3DCGを作るための一手法



オーバーロードの使いどころ

オーバーロードを使わない場合

```
class Ray
{
    Vector3D Intersect(物体 o) //戻り値: 衝突した位置
    {
        if (o.Type == Plane)
        {
            ... //光線と平面の衝突計算
        }
        else if (o.Type == Box)
        {
            ... //光線と四角の衝突計算
        }
        else if (o.Type == Sphere)
        {
            ... //光線と球の平面の衝突計算
        }
    }
}
```

オーバーロードの使いどころ

オーバーロードを使った場合

```
class Ray
{
    Vector3D Intersect(Plane p) //戻り値: 衝突した位置
    {
        ... //光線と平面の衝突計算
    }

    Vector3D Intersect(Box b) //戻り値: 衝突した位置
    {
        ... //光線と四角の衝突計算
    }

    Vector3D Intersect(Sphere s) //戻り値: 衝突した位置
    {
        ... //光線と球の衝突計算
    }
}
```

オーバーロードの使いどころ

オーバーロード実例:

XNA 4.0でのRayクラス(C#においては本当はRay構造体と呼ぶ)
でのIntersectsメソッド説明書より

The screenshot shows the MSDN documentation page for the `Ray.Intersects` method. The page title is "Ray.Intersects Method". Below the title, it says "Other Versions" and "Checks whether the Ray intersects a specified plane or bounding volume." The "Overload List" section contains a table with 8 overloads. The left sidebar shows the navigation tree with "Intersects Method" selected. The bottom left shows "Community Content" with a link to "Add code samples and tips to enhance this topic."

Ray.Intersects ...

検索: [検索ボタン] ▼ 次を検索 ▲ 前を検索 ⊕ ハイライト

Home Library Learn Downloads Support Community Sign in | 日本 - 日本語 | [設定アイコン] [印刷アイコン]

Search MSDN with Bing [検索ボタン]

- MSDN Library
- Development Tools and Languages
- XNA Game Studio
- XNA Game Studio 4.0 Refresh
- XNA Framework Class Library Reference
- Microsoft.Xna.Framework
- Ray Structure
- Ray Methods
 - Intersects Method**
 - Intersects Method (BoundingBox)
 - Intersects Method (BoundingBox, Nullable<Single>)
 - Intersects Method (BoundingFrustum)
 - Intersects Method (BoundingSphere)
 - Intersects Method (BoundingSphere, Nullable<Single>)
 - Intersects Method (Plane)
 - Intersects Method (Plane, Nullable<Single>)

Community Content

[ユーザーアイコン] Add code samples and tips to enhance this topic.

More...

Ray.Intersects Method

Other Versions ▼

Checks whether the Ray intersects a specified plane or bounding volume.

Overload List

Name	Description
Ray.Intersects (BoundingBox)	Checks whether the Ray intersects a specified BoundingBox.
Ray.Intersects (BoundingBox, Nullable<Single>)	Checks whether the current Ray intersects a BoundingBox.
Ray.Intersects (BoundingFrustum)	Checks whether the Ray intersects a specified BoundingFrustum.
Ray.Intersects (BoundingSphere)	Checks whether the Ray intersects a specified BoundingSphere.
Ray.Intersects (BoundingSphere, Nullable<Single>)	Checks whether the current Ray intersects a BoundingSphere.
Ray.Intersects (Plane)	Determines whether this Ray intersects a specified Plane.
Ray.Intersects (Plane, Nullable<Single>)	Determines whether this Ray intersects a specified Plane.

See Also

References

<http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.ray.intersects.aspx>

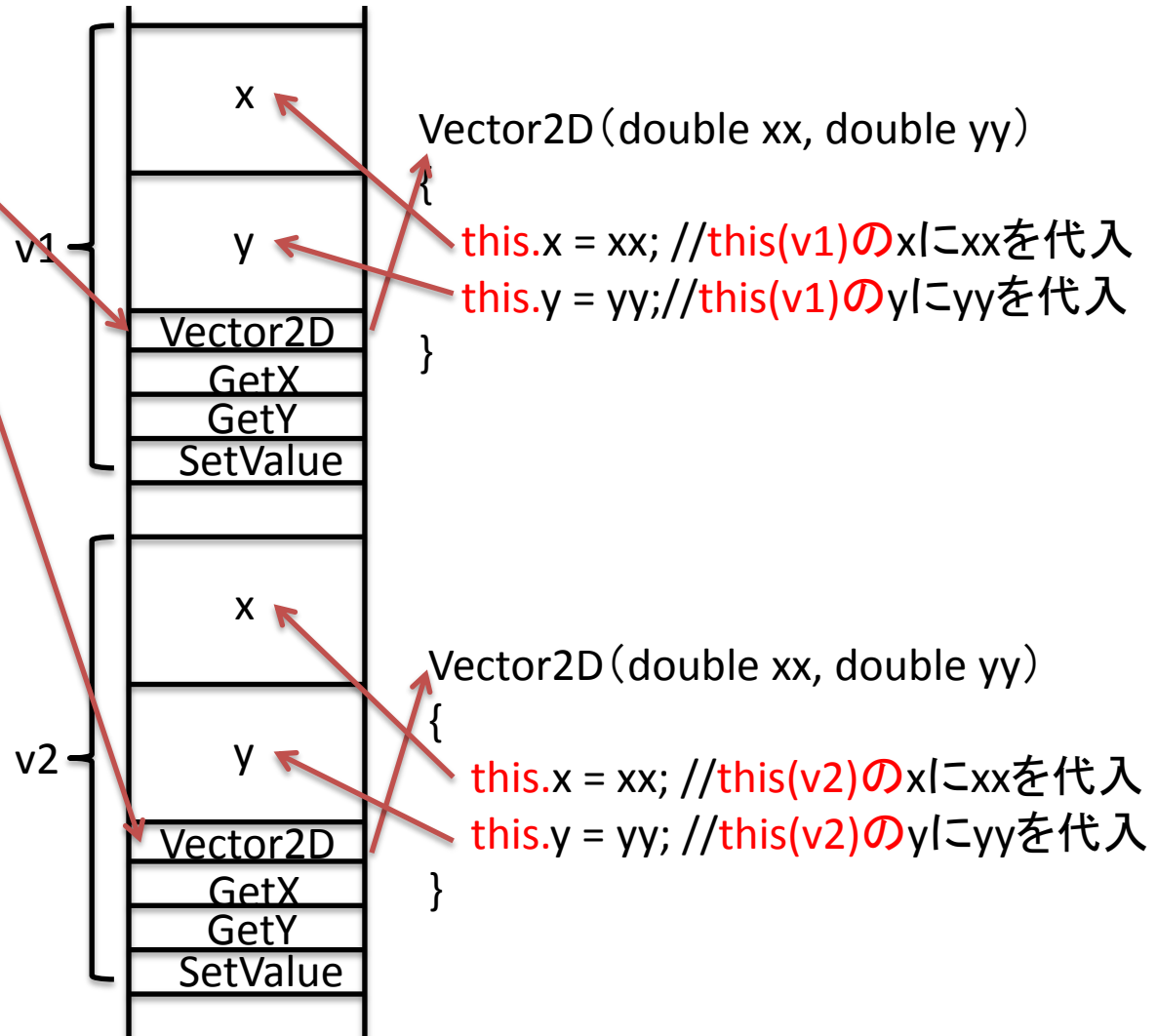
this

クラス内でthisと書くと、それは自身(インスタンス)を表す

```
Vector2D v1, v2;
```

```
v1 = new Vector2D(1.0, 2.0);
```

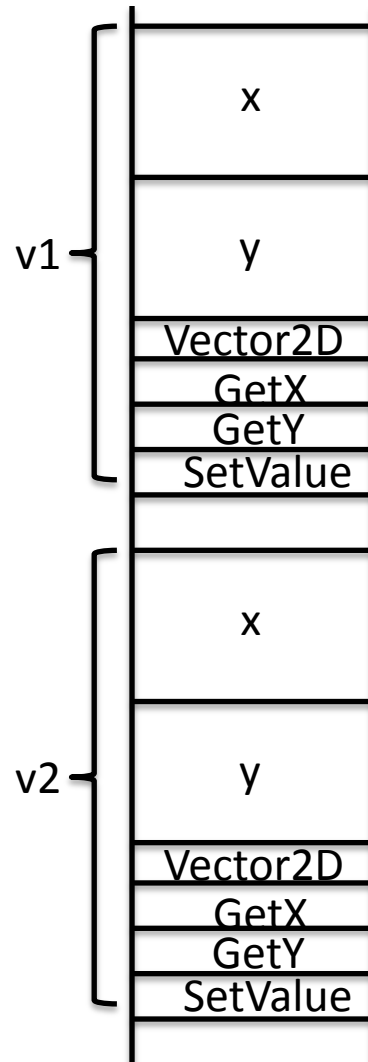
```
v2 = new Vector2D(3.0, 4.0);
```



this

クラス内でthisと書くと、それは自身(インスタンス)を表す

```
Vector2D v1, v2;  
v1 = new Vector2D(1.0, 2.0);  
v2 = new Vector2D(3.0, 4.0);
```



```
Vector2D(double x, double y)  
{  
    this.x = x; //this(v1)のxにxを代入  
    this.y = y; //this(v1)のyにyを代入  
}
```

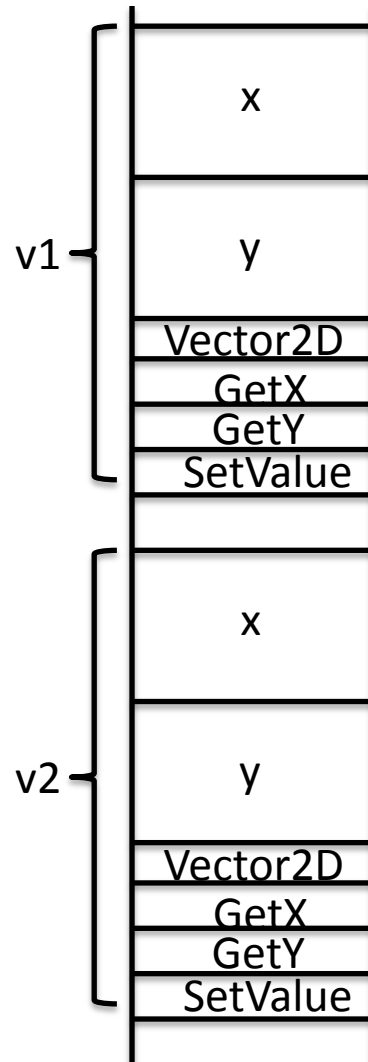
引数名と、属性名を同じにしても
thisを用いればどちらのx, yが
区別できるため、このような
書き方が可能に

```
Vector2D(double x, double y)  
{  
    this.x = x; //this(v2)のxにxを代入  
    this.y = y; //this(v2)のyにyを代入  
}
```


this

クラス内でthisと書くと、それは自身(インスタンス)を表す

```
Vector2D v1, v2;  
v1 = new Vector2D(1.0, 2.0);  
v2 = new Vector2D(3.0, 4.0);
```



```
Vector2D(double x, double y)  
{  
    x = x;  
    y = y;  
}
```

属性のx, yか,
引数のx, yか
区別できないのでこれはNG

```
Vector2D(double x, double y)  
{  
    x = x;  
    y = y;  
}
```