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

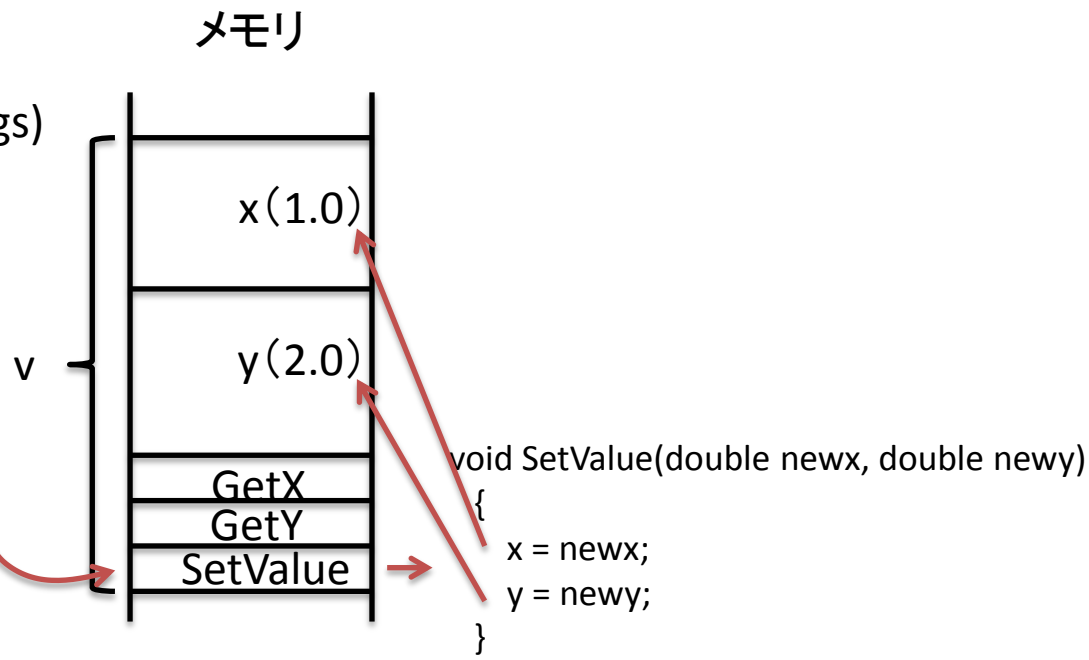
属性

メソッド

復習

Vector2Dクラスを振り返る.

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

    v.SetValue(1.0, 2.0);
}
```



インスタンスを作った後，
vのSetValueメソッドを使って
x，yへ値を入れていた

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に値を入れたい（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

```
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
}
```

通常のメソッドとは
異なり，戻り値の型を
書かないことに注意

[Vector2Dクラスでのコンストラクタ例]

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

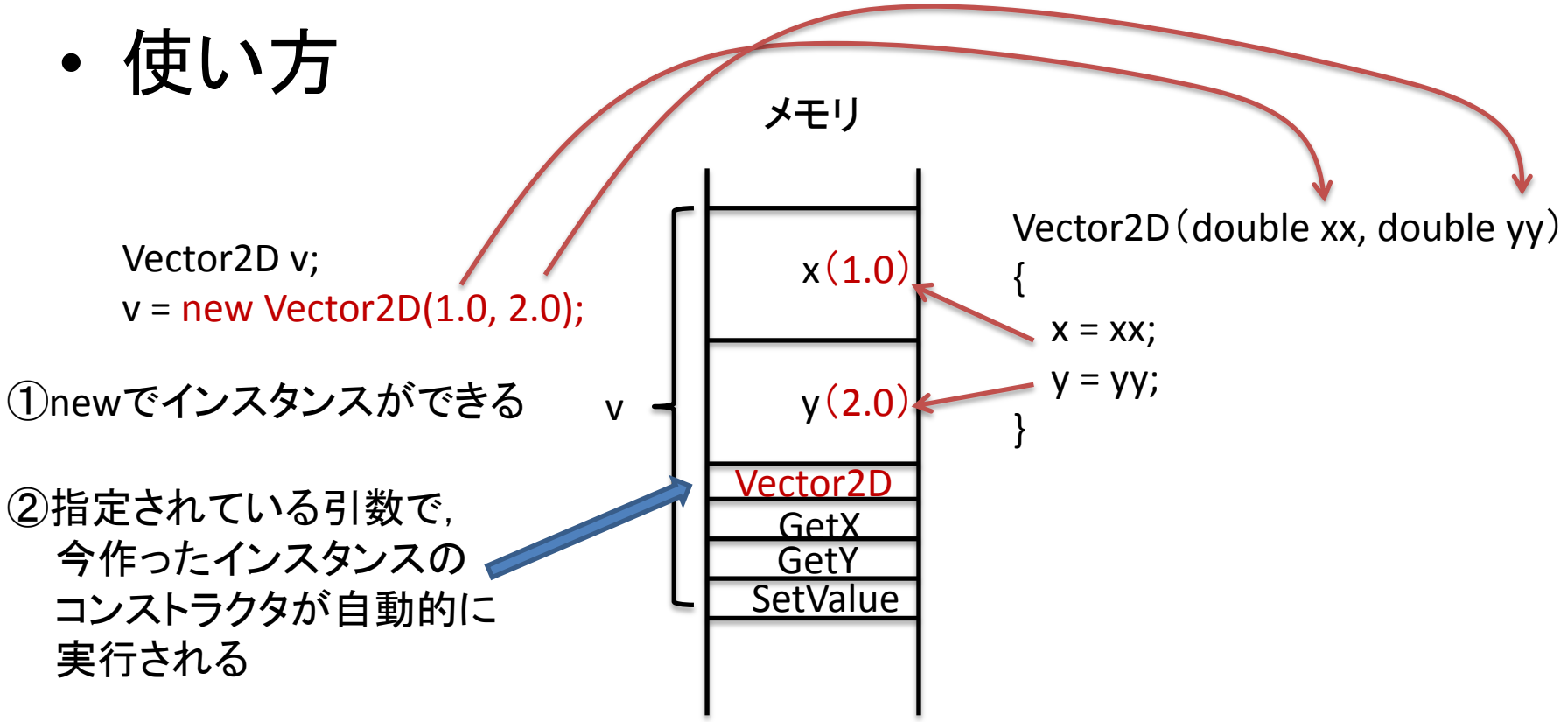
コンストラクタ

- 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


メモリ
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
v
x（1.0）
y（2.0）
Vector2D
GetX
GetY
SetValue
Vector2D（double xx, double yy）
{
x = xx;
y = yy;
}

演習

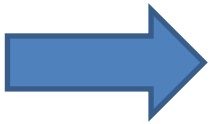
- コンストラクタを使って，x=1.0，y=2.0のVector2Dクラスインスタンスを作るプログラムを作成せよ
  プログラム名：VectorConstructor

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



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

   Vector2D()
   {

   }
}
```

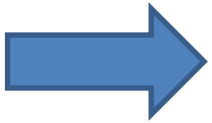
⇒デフォルトコンストラクタ
　と呼ばれる
⇒実はこれがあったから v = new Vector2D();と今まで書けた

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



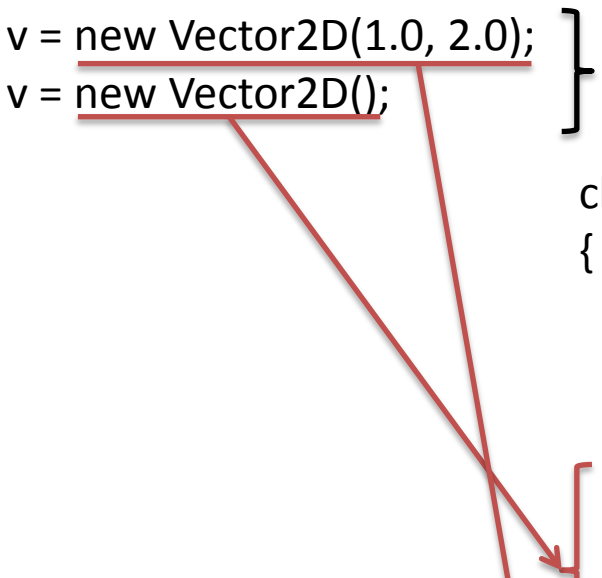
```
class Vector2D
{
   double x;
   double y;

   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

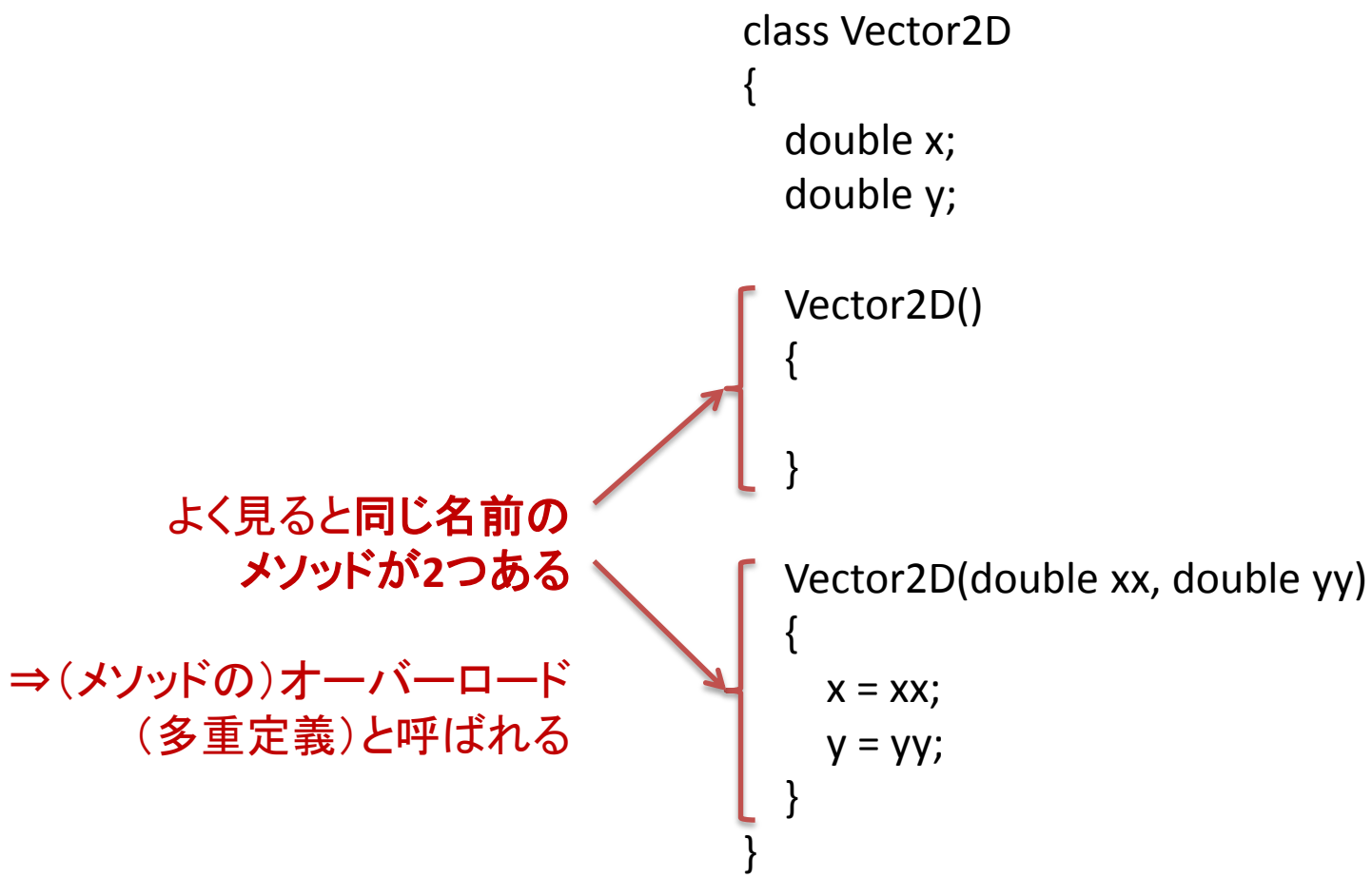
```
v = new Vector2D(1.0, 2.0);
v = new Vector2D();
```

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）



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;
    }
}
```

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();

// 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);

// 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);
```

引数がない

int型の引数2つ

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

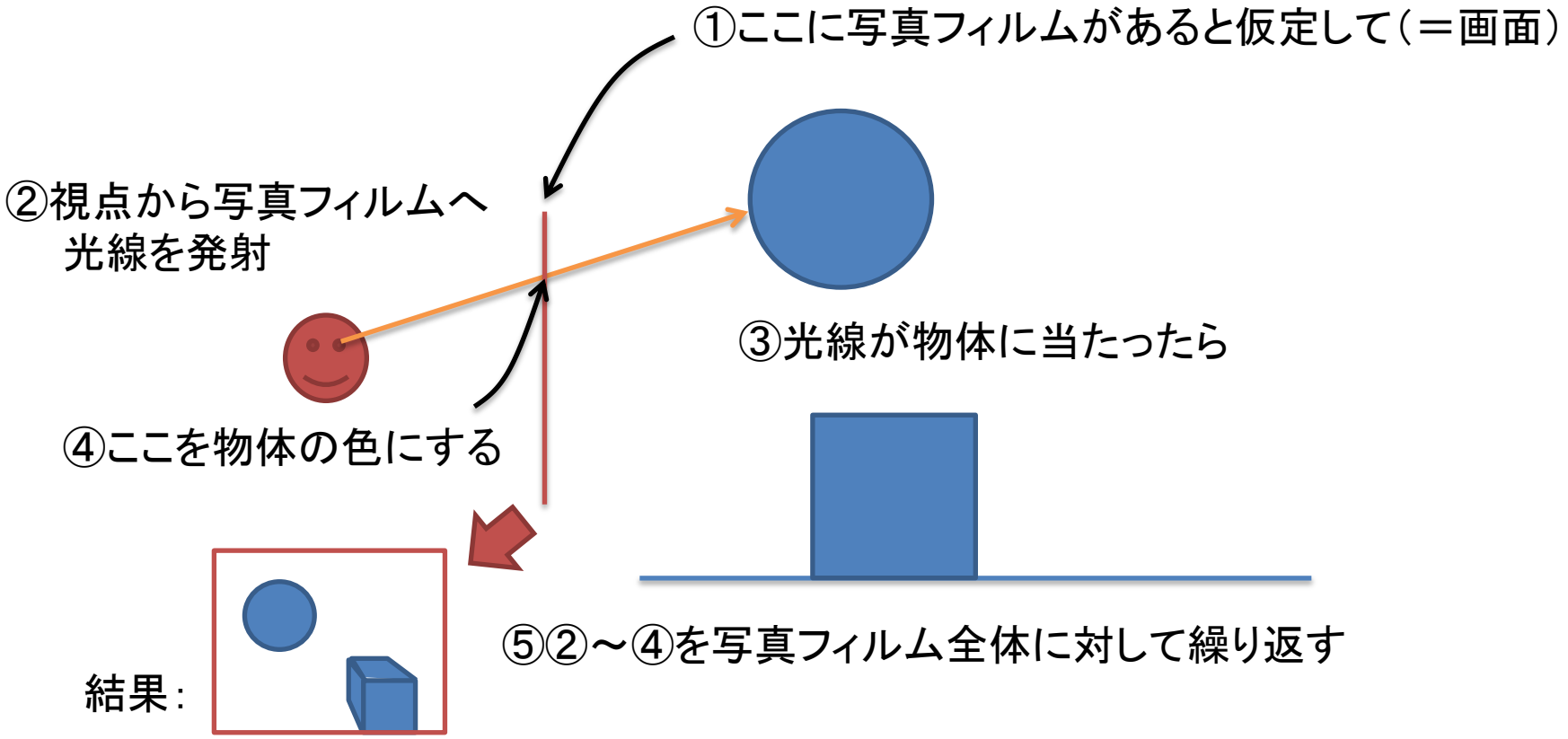
以下の実行結果が得られるように，オーバーロードを使用して
Vector2D.setValueメソッドを作成せよ

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() + ")");
    }
}
```

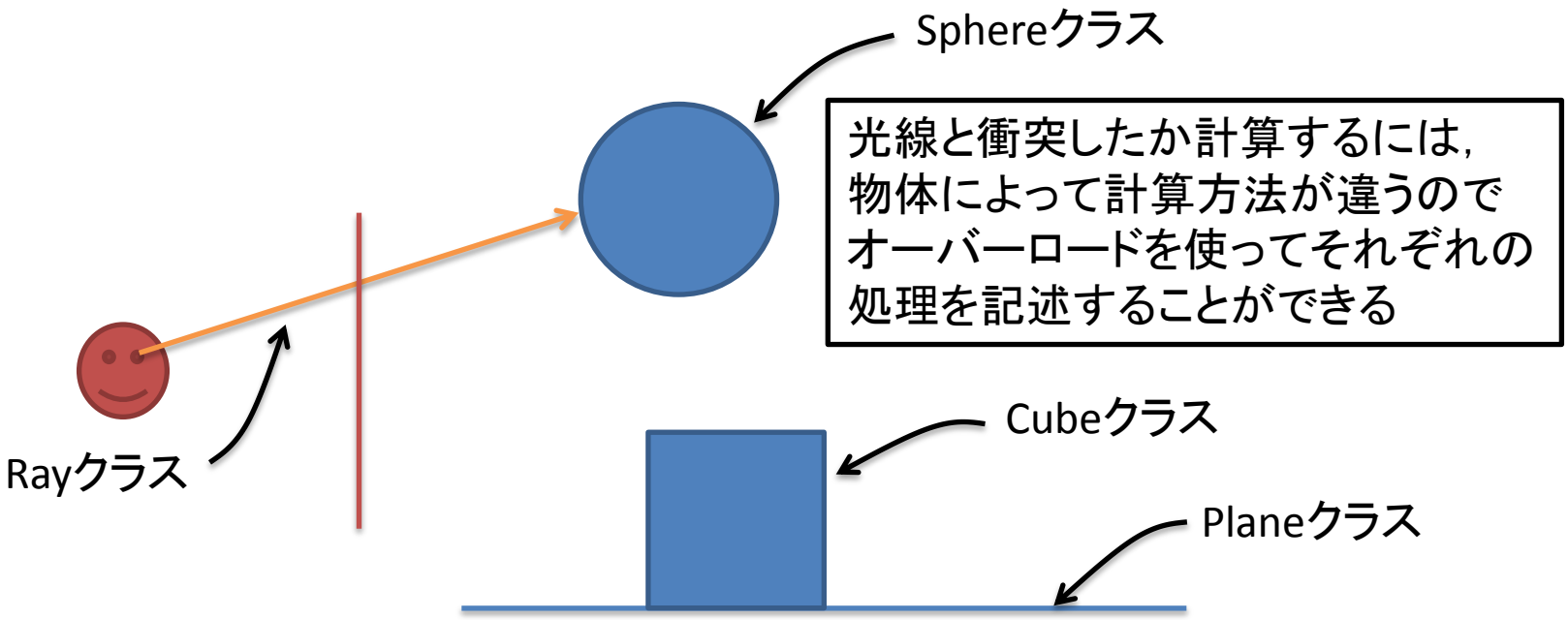
オーバーロードの使いどころ

- 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
  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
      }
   }
}
```

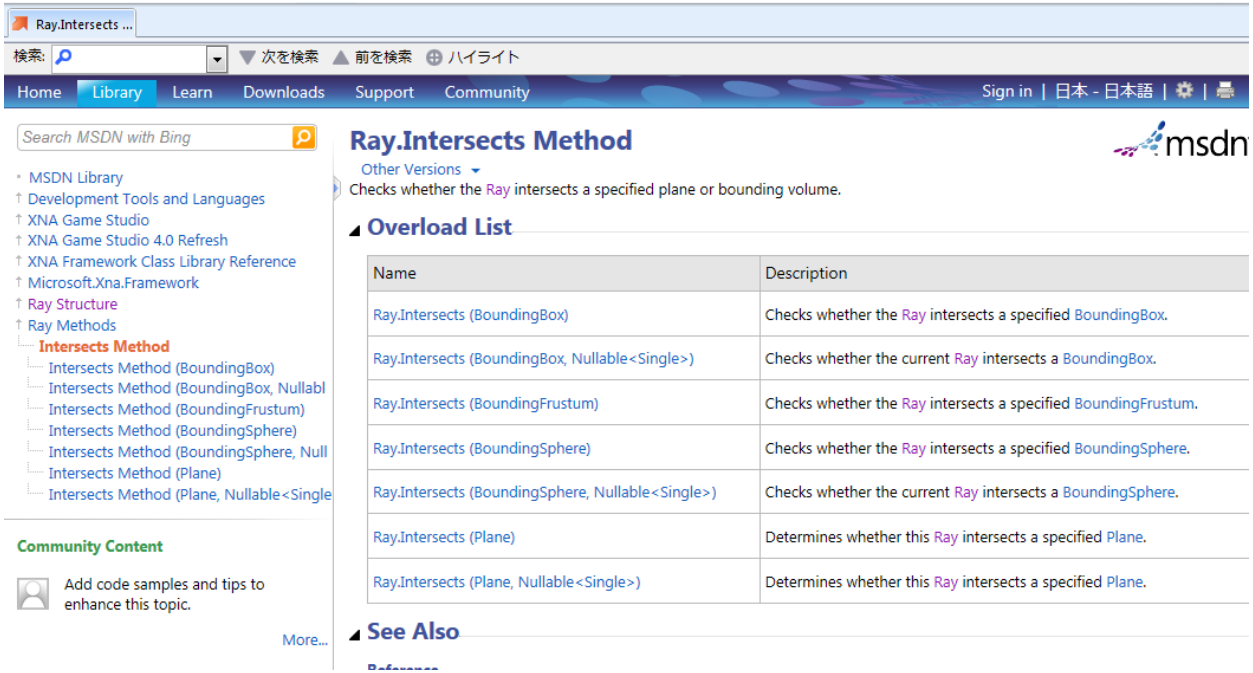
オーバーロードの使いどころ

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

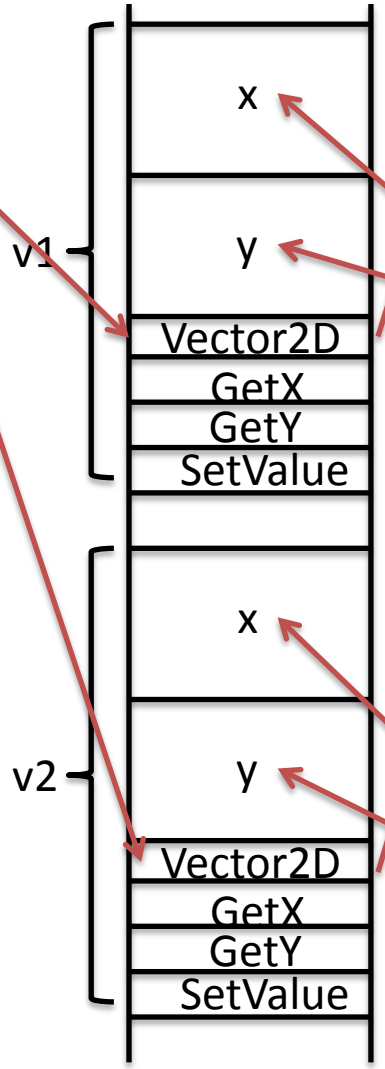


http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.ray.intersects.aspx

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);
```



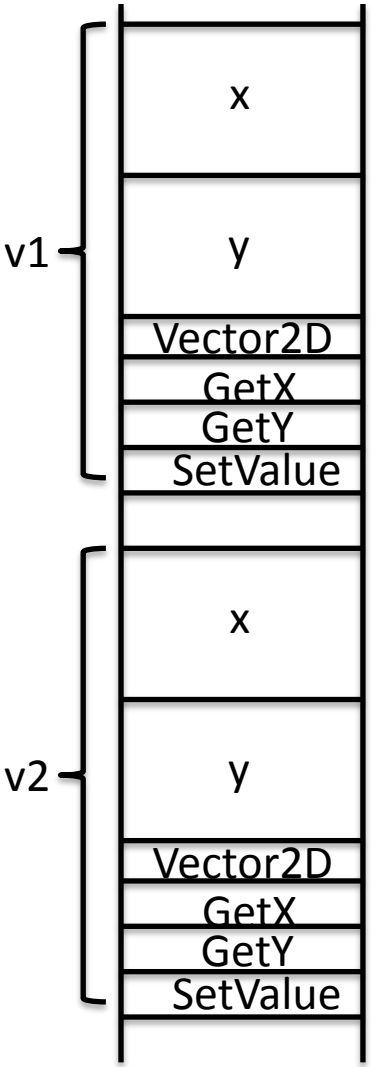

```
Vector2D（double xx, double yy）
{
    this.x = xx; //this(v1)のxにxxを代入
    this.y = yy;//this(v1)のyにyyを代入
}
```

```
Vector2D（double xx, double yy）
{
    this.x = xx; //this(v2)のxにxxを代入
    this.y = yy; //this(v2)のyにyy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）
{
   this.x = x; //this(v1)のxにxを代入
   this.y = y;//this(v1)のyにyを代入
}
```

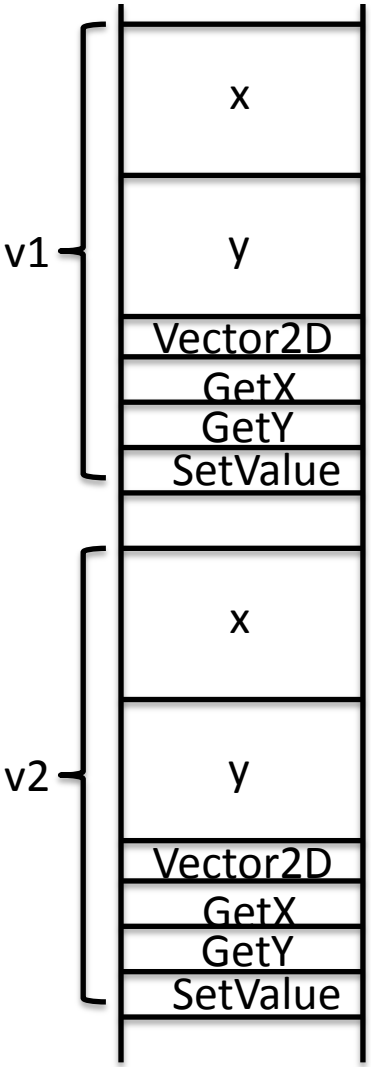
引数名と，属性名を同じにしても
thisを用いればどちらのx，yか
区別できるため，このような
書き方が可能に

```
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;
}
```