オブジェクト指向論(Q)

オブジェクト指向概論(B1)
オブジェクト指向(K1)

第8回講義資料
(プログラミング#2 OOP2)
2023/5/29

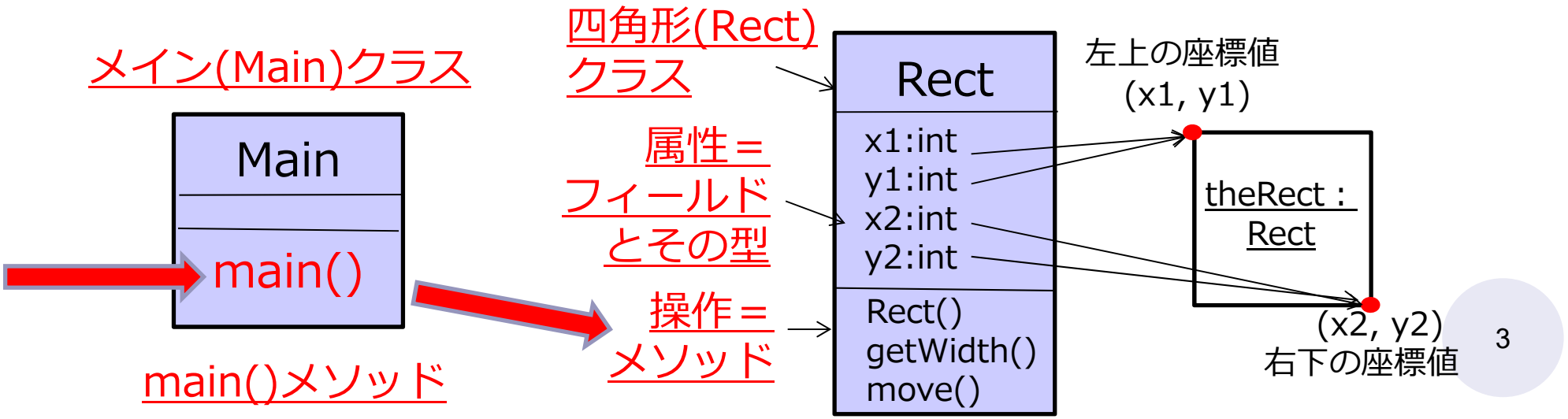
來村 徳信

前回と今回の講義の流れ

- オブジェクト指向プログラミング(2)：基本
  - Javaの文法の基礎
    - 変数（型）の宣言と利用，条件分岐等
  - 前回 クラスの利用：メソッドの呼び出し
    - コンストラクタの利用
      - インスタンス生成とクラス型変数
    - インスタンスメソッドの呼び出し
  - 今回 クラスの定義
    - フィールドの定義
      - ローカル変数の違い
    - メソッドの定義
      - コンストラクタの定義
      - インスタンスメソッドの定義
      - インスタンスフィールドの参照と代入
      - 自分のインスタンスメソッドの呼び出し

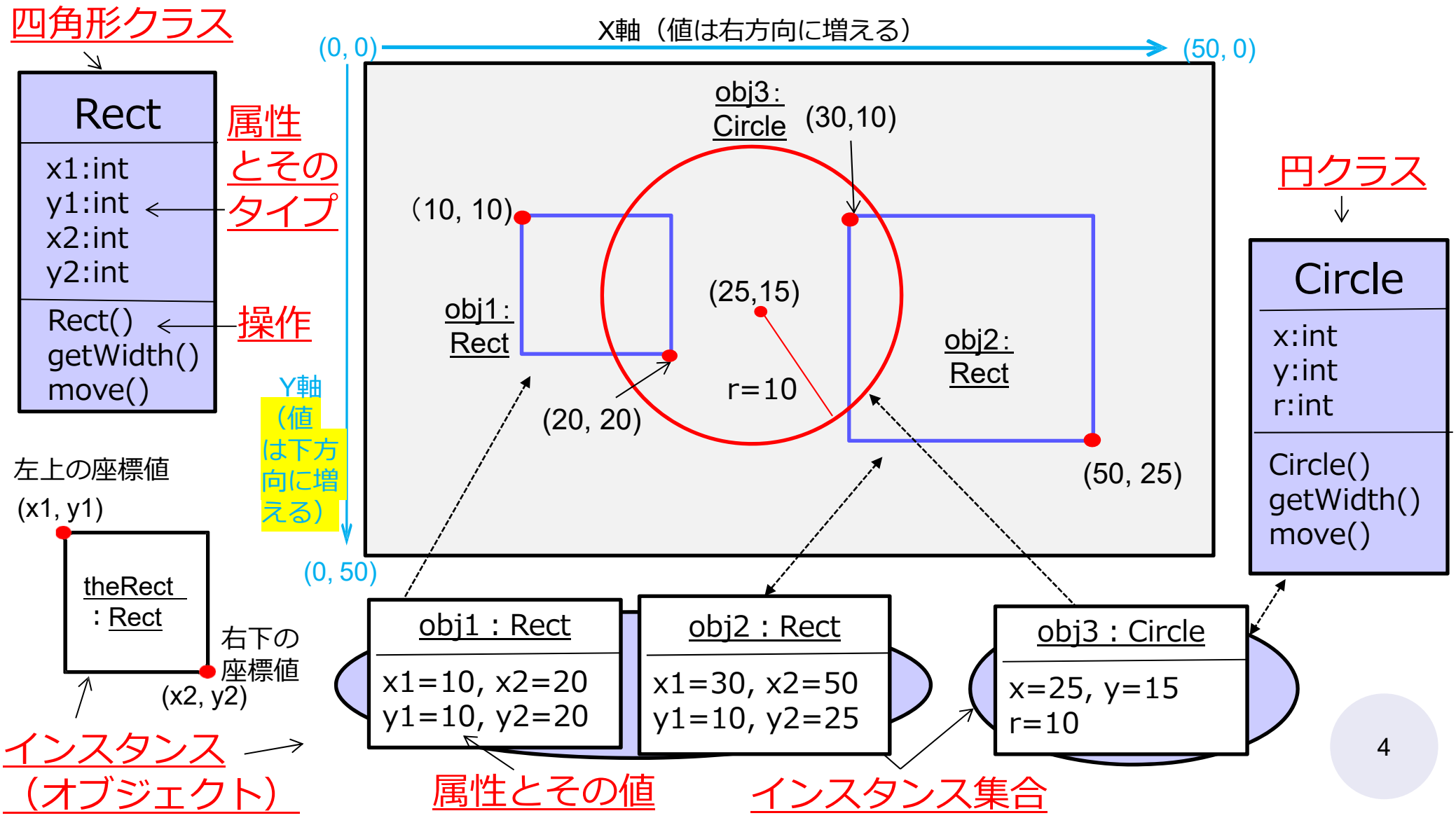
本日の講義の進め方

- 四角形(Rect)クラスのプログラムを例とする.
- 前回，Rectクラスを「利用する」プログラムを書いた
  - Javaでは main() というメソッドが最初に実行される.
    - Mainクラスの main メソッドから，Rect クラスのインスタンスを作ったり，メソッドを呼び出したりして，「利用する」方法を理解した.
- 今回，Rectクラスの「定義の仕方」を学ぶ
  - すこし拡張してみよう
  - 他のクラスを定義してみよう



例題におけるクラスとインスタンス例

- クラス：四角形（Rect），円（Circle）
- インスタンス：obj1:Rect, obj2:Rect, obj3:Circle



本日のプログラム

OOP2-A
Main.Java
Rect.java

- OOP2-A
  - コンパイルは「**javac *.java**」と入力する.
    - Main.java と Rect.java の両方をコンパイルする.
  - Main.java ファイル： Main クラス
    - 実行するときは「**java Main**」と入力する.
      - Main クラス内の main メソッドが最初に実行される.
    - 前回，Rectクラスのメソッドを呼び出した(OOP1-A)
    - 前回のプログラム＋今回の拡張用のプログラム(OOP2-A)
  - Rect.java ファイル：Rect クラスの定義
    - 四角形を表すクラス．拡張してみよう

編集後には再度コンパイルする

```
public class Main {
    public static void main(String[] args){
        System.out.println("Hello World! by OOP");
        ...
        }
}
```

コンソール（ターミナル）に文字列を出力するメソッド(Java 標準).
System.out とか意味が分からないと思いますが，今は気にせず，
そのまま使いましょう.

再掲

クラスの定義と仕様（メソッド）

- 例：四角形(Rect)クラスのメソッドの仕様
  - コンストラクタ（後述）の仕様：
    - Rect(int x1, int y1, int x2, int y2)
    - 左上の座標(x1,y1), 右下の座標(x2,y2)を表す，4つの正の整数を引数にして，コンストラクタを呼び出すと，新しいインスタンスが作られる（後述）．
  - インスタンスメソッドの仕様の表現例
    - int getWidth()
      - そのインスタンスのx軸方向の幅（正の整数）を返す

メソッドの戻り値の型　メソッド名　引数とその型（この例では引数なし）

    - void move(int dx, int dy)
      - そのインスタンスをx軸方向にdx, y軸方向にdy だけ移動させる

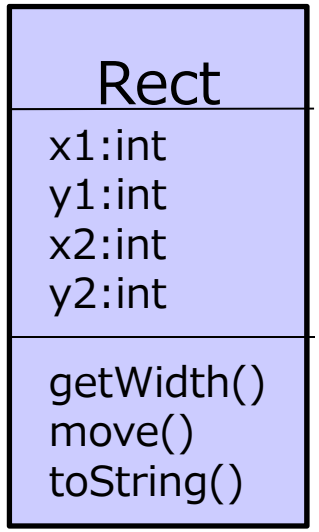
メソッドの戻り値が「ない」ことを表す　メソッド名はmove　引数は整数2つ．仮引数は dx, dy という名前

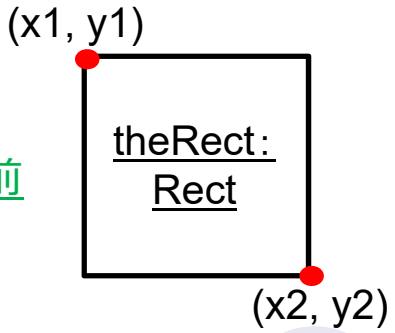
    - String toString()

引数なし

      - そのインスタンスの座標値を表す，“R (x1, y1)-(x2, y2)” のような文字列を返す．例: x1=10,y1=15,x2=20,y2=25であれば，“R (10, 15)-(20, 25)” を返す．

メソッドの戻り値は文字列型



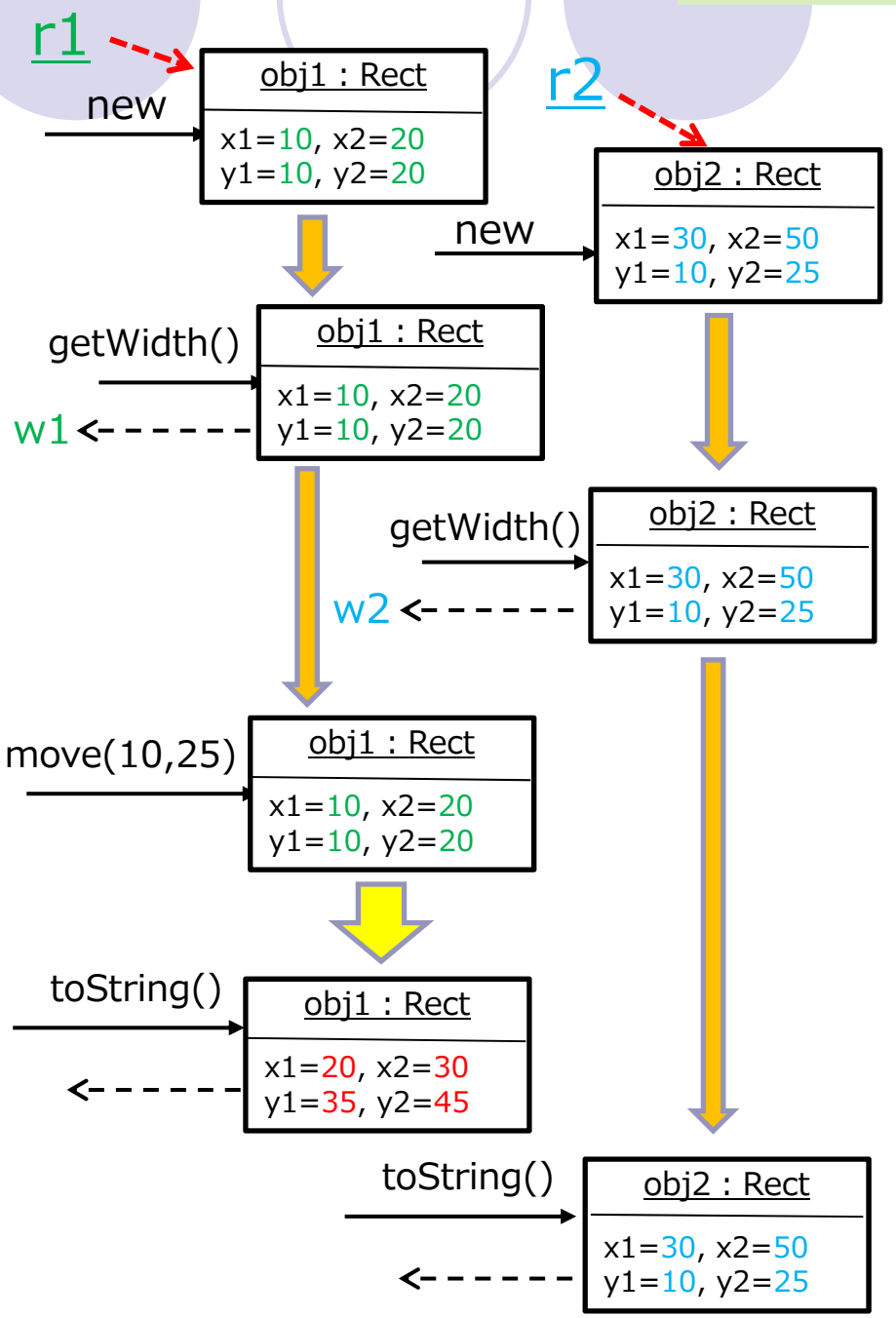


復習

OOP1/2-A
Main.Java

前回のプログラム（流れ）

- (1) 2つのRectインスタンス r1, r2 を作る
  - Rect r1 は (10,10)-(20,20)
  - Rect r2 は (30,10)-(50,25)
- (2) r1とr2の幅を求める
  - getWidth()を r1, r2 に向けて呼び出す
- (3) r1 を (10,25) 動かす
  - r1 に向けて move(10,25) を呼び出す
- (4) 座標値を表示する
  - r1,r2 に toString()を呼び出す



復習

超重要

OOP1/2-A
Main.Java

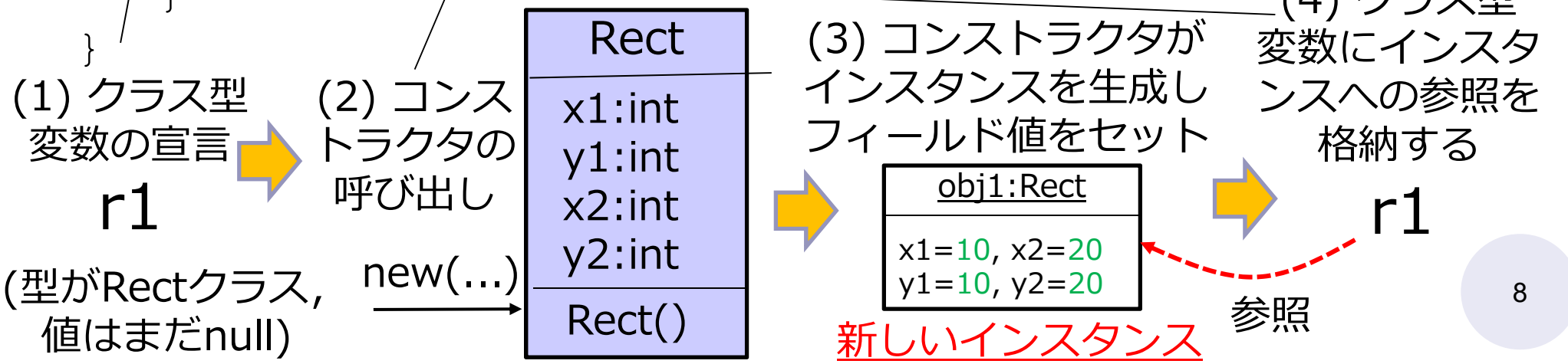
インスタンス生成とクラス型変数

- コンストラクタを キーワード **new** を付けて呼び出すと，インスタンスを１つ生成できる
  - 別のクラスのメソッドから呼び出す．
  - 戻り値は生成されたインスタンスへの参照
  - 型がそのクラスである変数（クラス型変数）に戻り値を格納する
    - 変数の「型」は生成するインスタンスのクラス(例：Rectクラス）．
    - クラス型変数はインスタンスへの参照を値として保持する．

pythonとほぼ同じ．違いは，
(1) new が必要，
(2) 変数の宣言が必要

```
public class Main {
  public static void main(String[] args) {
    Rect r1;
    r1 = new Rect(10,10,20,20);
    }
}
```

クラス型変数
Rectクラス型の r1 という変数



復習

超重要

OOP1/2-A
Main.Java

インスタンスメソッドの呼び出し

※コンストラクタやクラスメソッド以外

- 「クラス型変数」.「メソッド名」(引数…)
  - ドットでつなげる（半角のピリオド）
  - メソッド名がメッセージにあたる．後ろの(　)内が実引数．
  - メッセージの「宛先」は，クラス型変数の値が「参照」しているインスタンス．メソッド呼び出しには必ず「クラス型変数」が必要．

Pythonとほぼ同じ．
引数部に違いがある

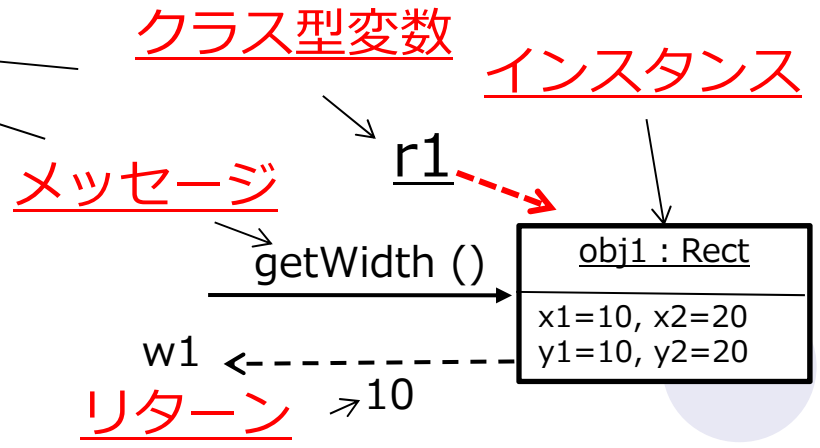
```
public class Main {
  public static void main(String[] args) {
    Rect r1 = new Rect(10,10,20,20);
    Rect r2 = new Rect(30,10,50,25);
    int w1 = r1.getWidth();
  }
}
```

クラス型変数 *r1* が値として保持する参照が指し示すインスタンスにメッセージ *getWidth* を送る

＝

インスタンス *r1*に メッセージ *getWidth* を送る

インスタンス *r1* の メソッド*getWidth* を呼び出す



復習

メソッド呼び出しの引数

- クラス型変数.メソッド名(実引数1, 実引数2,...)
  - 実引数に，データ値(例:10）や変数，式などを書く．
    - 実引数は，メソッドの仕様通り，定義側の仮引数の「数」と「型」（例:2個の整数(int)型）に，順番に一致していなければならない．仮引数名は関係ない．
    - 型名（例:int）や仮引数名（例:dx）は呼び出し側には書かない（書けない）．
    - VS Code ユーザ必見 → VS Code では「dx:」等が表示されるが実際には挿入されていない（できない）
      - 設定で表示機能をOFFにすることを強く推奨．java資料の p.17 を参照．
  - 実引数と仮引数の対応は，順番で決まる
  - 実変数の値が，メソッド側の仮引数に代入される
  - メソッド前のクラス型変数(例:r2)の値も this に代入され，同じインスタンスを指す(と考えてよい)

Pythonの「位置引数」に対応

Pythonの「キーワード引数」例: (dx=10, dy=25)はない

メソッドの呼び出し側

```
r2.move(20,10);
```

実引数1　実引数2

```
r2.move(x,y);
```

x,yはint型. x==20,y==10のとき

メソッドの定義側

順番に対応する

```
public class Rect {
    public void move(int dx, int dy) {
this=r1 こう代入される  dx=20; dy=10;
        this.x1 += dx; this.x2 += dx;
        this.y1 += dy; this.y2 += dy;
    }}
```

仮引数1　仮引数2

実際のプログラムには書かないこと！

OOP1/2-A
Main.Java

前回のプログラム

- (1) 2つのRectインスタンス r1, r2 を作る
- (2) r1とr2の幅を求める
- (3) r1 を (10,25) 動かす
- (4) 座標値を表示する

```
public class Main {
   public static void main(…){
     Rect r1 = new Rect(10,10,20,20);
     Rect r2 = new Rect(30,10,50,25);
     int w1 = r1.getWidth();
     System.out.println("width of r1 = " + w1);
     r2.getWidth();
     System.out.println("width of r2 = " + w2);
     r1.move(10,25);
     System.out.println("r1 = " + r1.toString());
     System.out.println("r2 = " + r2.toString());
     }
   }
```

実行結果

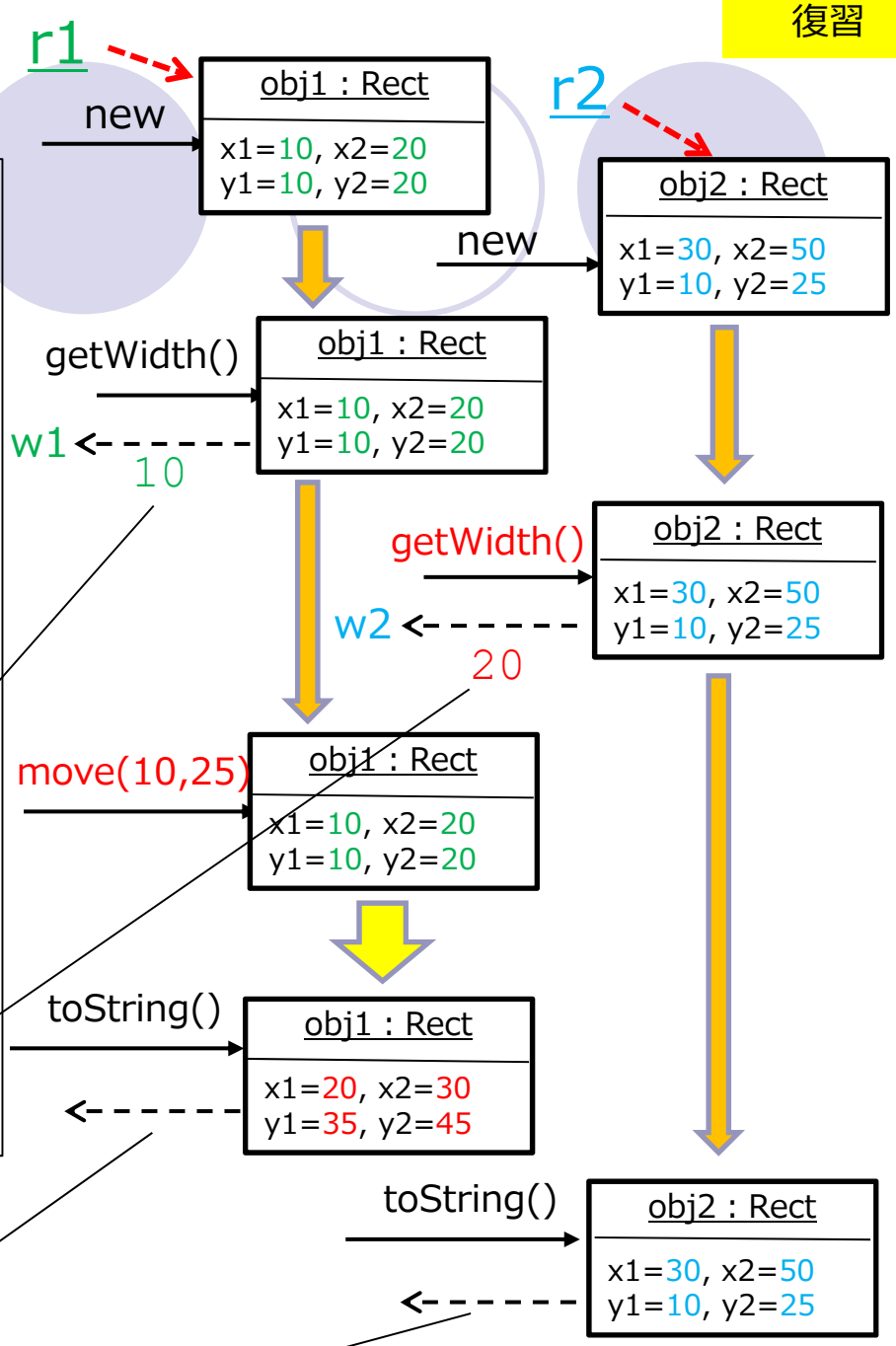
```
Rect r1 = new Rect(10,10,20,20);
Rect r2 = new Rect(30,10,50,25);

int w1 = r1.getWidth();
System.out.println("width of r1 = "
  + w1);
int w2 = r2.getWidth();
System.out.println("width of r2 = "
 + w2);

r1.move(10,25);
System.out.println("r1 = " 
 + r1.toString());
System.out.println("r2 = " 
 + r2.toString());
```

r1 を右下方向に(10,25)だけ移動させる.

```
width of r1 = 10
width of r2 = 20
r1 = R (20, 35)-(30, 45)
r2 = R (30, 10)-(50, 25)
```



前回と今回の講義の流れ（再掲1）

- オブジェクト指向プログラミング(2)：基本
  - Javaの文法の基礎
    - 変数（型）の宣言と利用，条件分岐等
  - 前回 ⇨ クラスの利用：メソッドの呼び出し
    - コンストラクタの利用
      - インスタンス生成とクラス型変数
    - インスタンスメソッドの呼び出し
  - 今回 ➡ クラスの定義
    - ➡ フィールドの定義
      - ローカル変数との違い
    - メソッドの定義
      - コンストラクタの定義
      - インスタンスメソッドの定義
      - インスタンスフィールドの参照と代入
      - 自分のインスタンスメソッドの呼び出し

Rectクラスの定義例(OOP1/2-A)

OOP1/2-A
Main.Java

```
public class Rect {
   private int x1, y1, x2, y2;

   public Rect(int x1,int y1,int x2,int y2) {
      this.x1 = x1; this.x2 = x2;
      this.y1 = y1; this.y2 = y2;
      }
   public int getWidth() {
      return (this.x2 - this.x1)
      }
   public void move(int dx,int dy) {
      this.x1 += dx; this.x2 += dx;
      this.y1 += dy; this.y2 += dy;
      }
   @Override
   public String toString() {
       return ("R (" + this.x1 + ", "
       + this.y1 + ")-(" + this.x2 + ", "
       + this.y2 + ")");
       }
```

フィールド定義

コンストラクタ

左上(x1,y1),右下(x2,y2)のインスタンスを生成する

自分の幅を返す

インスタンスメソッド

右下方向に(dx, dy)だけ移動させる

座標値を表す文字列を返す（表示用）

基礎的文法：クラス定義（概要）

- Javaプログラムは「クラスの定義」(宣言)である
  - クラス名＝ .java ファイル名（本講義では）
    - Rectクラスは Rect.java で定義されている．中身をみてみよう
  - **class ＜クラス名＞ { }** で宣言する．
    - フィールド（≒変数）とメソッド（≒関数）の定義からなる．
    - フィールド定義（宣言）はメソッドの外側で行う．
      - メソッドの内側は「ローカル変数」になる（後述）
      - 慣習として，クラス名宣言の直後（最初のメソッド定義の前）に書く．
      - 先週のMain クラスにはフィールドが無かっただけ．

Javaプログラム

クラス定義

```
public class Rect {
  private int x1, y1, x2, y2;
```

フィールド定義（今から述べる）

メソッド定義

```
  public Rect(int x1,int y1,int x2,int y2) {
    this.x1 = x1; this.y1 = y1;
    this.x2 = x2; this.y2 = y2;
  }
}
```

フィールドの定義の仕方

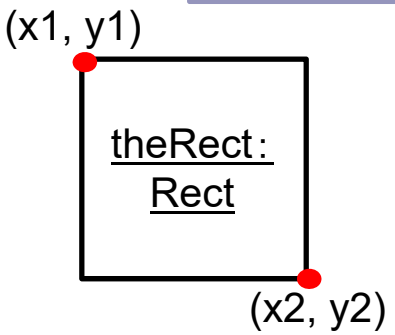
- クラス宣言のすぐ下(メソッドの外)で定義する.
- (インスタンス)フィールド（インスタンス変数）
  - 各インスタンスが固有な値を保持できる変数.
  - 例：四角形(Rect)クラスのフィールド定義

```
public class Rect {
  private int x1;
  private int y1;
  private int x2, y2;
```

修飾子　型　フィールド名

同じ型ならば1行で複数を，で区切って宣言してもよい.

Pythonと同じ概念だが，宣言が必要



    - 整数(int)型のインスタンス変数 x1, y1, x2, y2.
    - 四角形の左上の座標(x1,y1) と右下の座標(x2,y2)を保持する.
  - 修飾子：（後述）
    - **private**：クラス外から参照／変更できない<推奨>
    - **public**：クラス外から参照／変更できる
  - 型（タイプ）：int などの基本データ型またはクラス（後述）

クラス内でのフィールドへのアクセス

- フィールド名でアクセスする
  - または **this.** <フィールド名>
    - this は自分自身(インスタンス)への参照を表す（後述）
    - this は省略可能な場合もあるが，常に付けることを強く推奨
      - フィールド変数であることが分かる.
      - ローカル変数（次スライド）と区別できる
    - 「.」は半角のピリオド（ドット）

Pythonの self に対応するが，Javaでは明示的な宣言はない

C 言語の構造体のメンバー変数へのアクセスと同じ表記

```
public class Rect {
  private int x1, y1, x2, y2;

  メソッドの定義部分 {
    this.x1 = xa;
    int leftx = this.x1
    int w = this.x2 - this.x1
    }
  }
```

フィールド宣言

フィールドへの代入

フィールドの値の取り出し

（後述）

ローカル変数（次スライド）

超重要

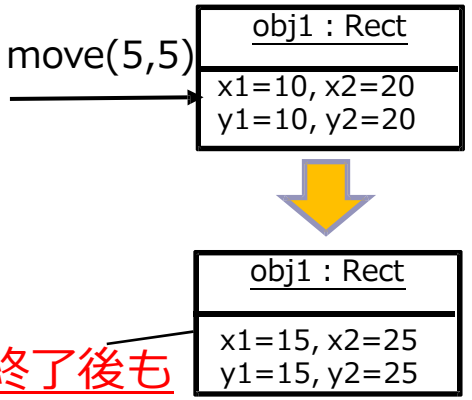
フィールドとローカル変数の違い

- (インスタンス)フィールド
  - メソッド定義の外側（クラス名宣言の直下）で定義される.
  - インスタンスごとに，メソッドから出ても値が保持される.
  - this. を付けて参照する（省略可能だが強く推奨）
- ローカル変数（メソッドの仮引数も同じ）
  - メソッド内で宣言される変数．場所は任意のブロック.
  - 変数の値はメソッドから出るとなくなる．外から参照不可.

```
public class Rect {
   private int x1, y1, x2, y2;

   public void move(int dx, int dy) {
      int delta = dx+dy;
      this.x1 += dx; this.x2 += dx;
      this.y1 += dy; this.y2 += dy;
      }
   }
```

フィールド



move()の終了後も
x1,y1などの値は保持される

ローカル変数である delta の値は破棄される. 仮引数のdx,dyも同じ

前回と今回の講義の流れ（再掲2）

- オブジェクト指向プログラミング(2)：基本
  - Javaの文法の基礎
    - 変数（型）の宣言と利用，条件分岐等
  - クラスの利用：メソッドの呼び出し
    - コンストラクタの利用
      - インスタンス生成とクラス型変数
    - インスタンスメソッドの呼び出し
  - クラスの定義 

    - フィールドの定義
      - ローカル変数の違い
    - メソッドの定義 
      - コンストラクタの定義
      - インスタンスメソッドの定義
      - インスタンスフィールドの参照と代入
      - 自分のインスタンスメソッドの呼び出し

メソッドの定義

- メソッド＝そのインスタンスへのメッセージに対して反応して，行う操作／動作／処理
  - 操作というよりオブジェクトの振る舞い
  - 修飾子：
    - **public**：異なるクラスから呼び出しできる ＜こちらが主＞
    - **private**：そのクラスのメソッド内でしか呼び出しできない

クラス

メソッド

```
public class Rect {
        private int x1, y1, x2, y2;

    public int getWidth() {
            int w = this.internalwidth();
            return (w);
            }
    private int internalwidth() {
            return (this.x2-this.x1);
            }
}
```

pythonのdefは不要だが，修飾子と戻り値の型の宣言が必要

コンストラクタの定義

Pythonには明示的なコンストラクタの定義はないが，初期化メソッド __init__ がすることに似ている．

- 新しいインスタンスを作る特別なメソッド
  - メソッド名が クラス 名と同じ public メソッド
  - 新しいインスタンス（オブジェクト）を１つ，生成する
  - 一般的に，必須なフィールドの値を引数にとり，生成直後に自分自身(thisで参照）にセットする．
    - 四角形の場合，例えば，左上と右下の座標値
  - オブジェクトへの参照(reference)を返す（暗黙的）

```
public class Rect {
  private int x1, y1, x2, y2;

  public Rect(int x1, int y1, int x2, int y2) {
    this.x1 = x1; this.y1 = y1;
    this.x2 = x2; this.y2 = y2;
    }
  }
```

Rectクラスのコンストラクタ

インスタンス生成とクラス型変数

再復習

OOP1/2-A
Main.Java

- コンストラクタを キーワード **new** を付けて呼び出すと，インスタンスを１つ生成できる
  - 別のクラスのメソッドから呼び出す．
  - 戻り値は生成されたインスタンスへの参照
  - 型がそのクラスである変数（クラス型変数）に戻り値を格納する
    - 変数の「型」は生成するインスタンスのクラス(例：Rectクラス）．
    - クラス型変数はインスタンスへの参照を値として保持する．

pythonとほぼ同じ．違いは，
(1) new が必要，
(2) 変数の宣言が必要

```
public class Main {
  public static void main(String[] args) {
    Rect r1;
    r1 = new Rect(10,10,20,20);
    }
  }
```

クラス型変数
Rectクラス型の r1 という変数

(1) クラス型変数の宣言
r1
(型がRectクラス，値はまだnull)

(2) コンストラクタの呼び出し
new(...)

Rect
x1:int
y1:int
x2:int
y2:int
Rect()

(3) コンストラクタがインスタンスを生成しフィールド値をセット

obj1:Rect
x1=10, x2=20
y1=10, y2=20
新しいインスタンス

(4) クラス型変数にインスタンスへの参照を格納する
r1
参照

例：Rectクラスのコンストラクタ

- Rect という(クラス名と同じ)名前のメソッド
  - (0) 4つのint型の実引数値(4つの整数)を渡される(仕様)
    - 順に，仮引数のx1, y1, x2, y2 に実引数値が代入されてくる.
  - (1) Rectクラスのインスタンスを1つ生成する（暗黙的）
  - (2) そのインスタンスフィールドに仮引数の値をセットする.
    - 仮引数とフィールドは別の変数．仮引数はローカル変数であり，メソッドから出たら消えるから，値をフィールドに代入する必要がある.
  - (3) 生成したRectインスタンスへの参照を返す（暗黙的）

```
public class Rect {
  private int x1, y1, x2, y2;
  public Rect(int x1, int y1, int x2, int y2) {

    this.x1 = x1;
    this.y1 = y1;
    this.x2 = x2; this.y2 = y2;

  }
}
```

Rectクラスのコンストラクタ

引数（ローカル変数）

生成されたインスタンスのフィールド

(1) ここで自動的にインスタンスが生成される. this で参照できる.

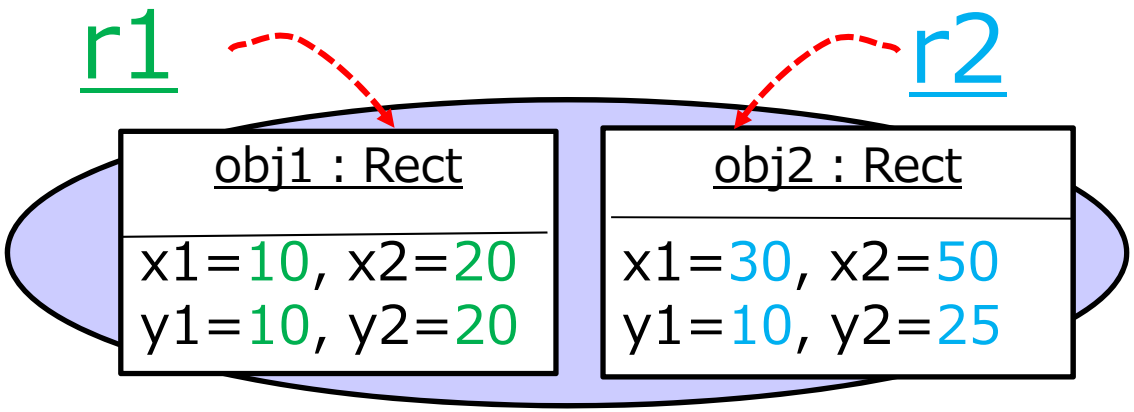
(3) ここで自動的に this が呼び出し元に返される.
（return this とは書かなくてよい．むしろ書けない）

コンストラクタの使い方(2)

OOP1-A
Main.Java

- コンストラクタを複数回，呼び出す
  - インスタンスを複数生成し，
    それらへの参照を複数のクラス型変数(r1, r2)で保持する.
    - r1, r2 は同じ Rect クラス型の，異なる変数
  - 複数のインスタンスが，個別のフィールド変数「値」を
    保持していることに注目!

```
public class Main {
  public static void main(String[] args){
    Rect r1;
    r1 = new Rect(10,10,20,20);
    Rect r2 = new Rect(30,10,50,25);
  }
}
```



再復習

OOP1/2-A
Main.Java

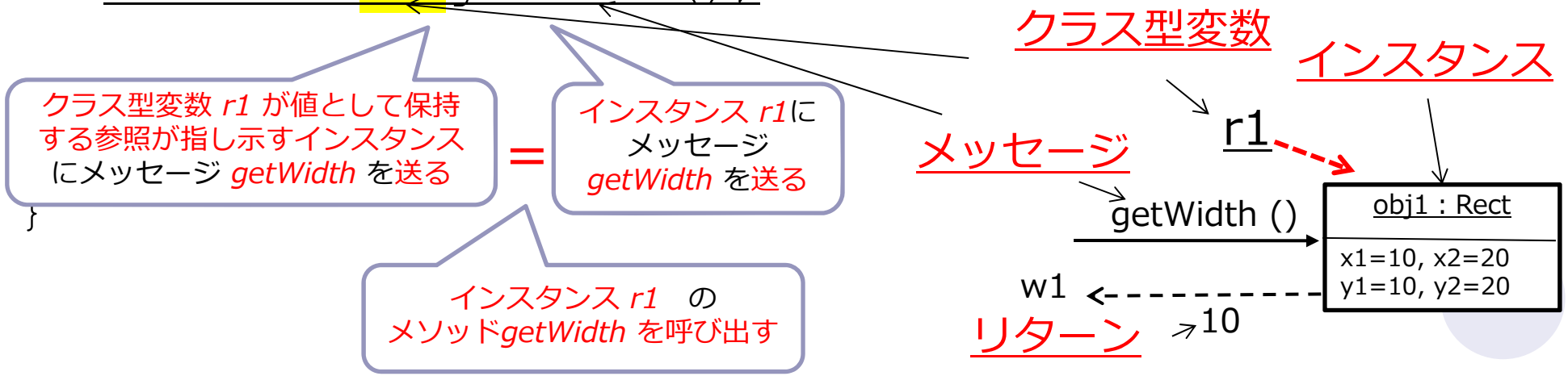
インスタンスメソッドの呼び出し

※コンストラクタやクラスメソッド以外

- 「クラス型変数」.「メソッド名」(引数…)
  - ドットでつなげる（半角のピリオド）
  - メソッド名がメッセージにあたる．後ろの(　)内が実引数．
  - メッセージの「宛先」は，クラス型変数の値が「参照」しているインスタンス．メソッド呼び出しには必ず「クラス型変数」が必要．

Pythonとほぼ同じ．
引数部に違いがある

```
public class Main {
  public static void main(String[] args) {
    Rect r1 = new Rect(10,10,20,20);
    Rect r2 = new Rect(30,10,50,25);
    int w1 = r1.getWidth();
  }
```



メソッドの定義例1：問い合わせ

- Rectクラスの getWidth() メソッド
  - 引数なし．自分自身の(x軸方向の)幅（整数値(int型)）を返す
  - 自分自身のインスタンスのフィールド値 x1, x2 を参照する．
    - インスタンスごとにフィールド値が保持されているから．
    - **this.**＜フィールド名＞で参照．this は自分自身(インスタンス)への参照を表す．引数には現れないことに留意．
    - return 文で値を返す．1つだけ．文末に ; が必要．
      演算の結果を返す場合は ( ) で囲む．
      - return と (の間に半角空白を入れることを推奨．
        メソッド呼び出しと区別するため（なくても動く）

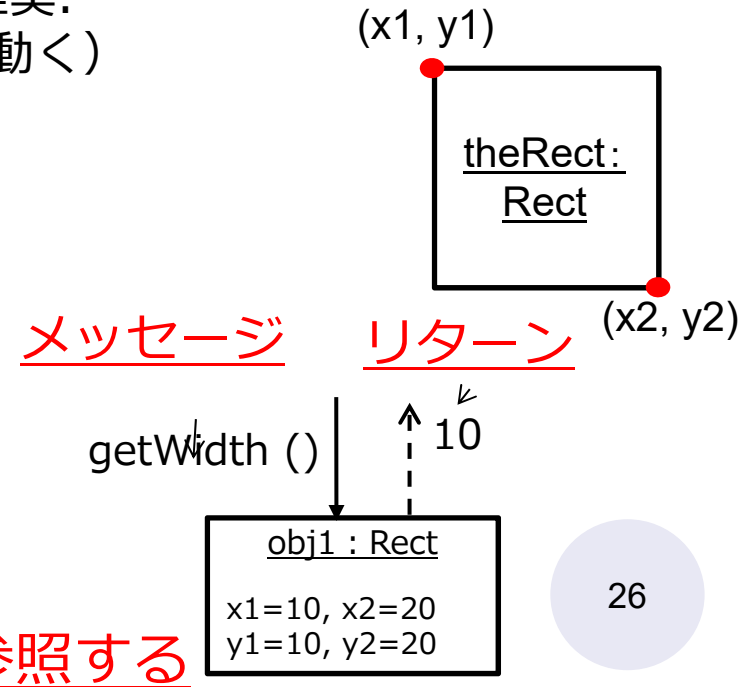
```
public class Rect {
  private int x1, y1, x2, y2;

  public int getWidth() {
    return (this.x2 - this.x1);
    }
}
```

修飾子　戻り値の型　メソッド名

インスタンスメソッド

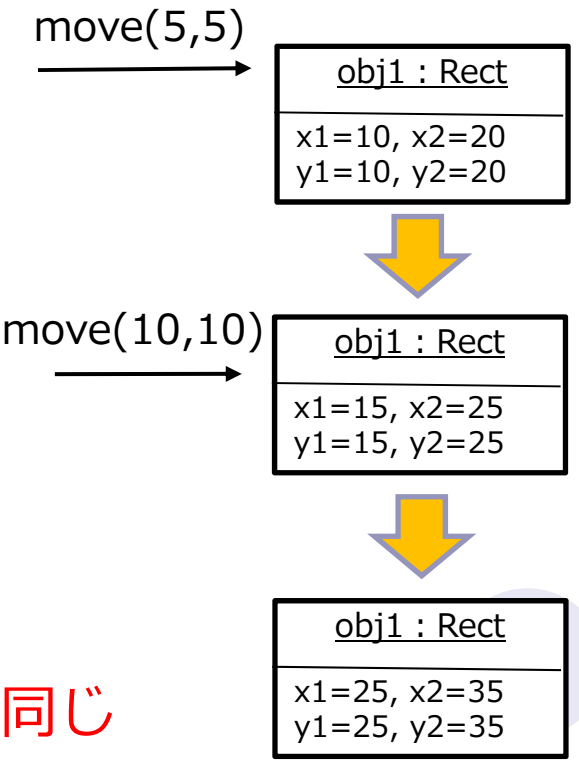
半角スペース　自分のフィールドの値を参照する



OOP2-A
Rect.java

メソッドの定義例２：更新

- インスタンスの状態を変化させるメソッド
  - インスタンスの状態=フィールドの値
  - インスタンスごとに異なるフィールド値を保持できる.
  - インスタンスごとにずっと保持される.
  - メソッド呼び出しにより<u>インスタンスの状態（≒属性値≒フィールド値）を変化させる</u>＝「<u>副作用</u>」を起こす
  - 例：move(int dx, int dy)
    - x方向にdx, y方向に dy だけ移動させる




```
public class Rect {
   private int x1, y1, x2, y2;
   public void move(int dx, int dy) {
     this.x1 += dx; this.x2 += dx;
     this.y1 += dy; this.y2 += dy;
     }
}
```

戻り値がない

仮引数のリスト

```
this.x1 = this.x1 + dx;
```

と同じ

再復習

メソッド呼び出しの引数

- クラス型変数.メソッド名(実引数1, 実引数2,...)
  - 実引数に，データ値(例:10）や変数，式などを書く．
    - 実引数は，メソッドの仕様通り，定義側の仮引数の「数」と「型」（例:2個の整数(int)型）に，順番に一致していなければならない．仮引数名は関係ない．
    - 型名（例:int）や仮引数名（例:dx）は呼び出し側には書かない（書けない）．

VS Code ユーザ必見 → VS Code では「dx:」等が表示されるが実際には挿入されていない（できない）

- 設定で表示機能をOFFにすることを強く推奨．java資料の p.17 を参照．

  - 実引数と仮引数の対応は，順番で決まる
  - 実変数の値が，メソッド側の仮引数に代入される
  - メソッド前のクラス型変数(例:r2)の値も this に代入され，同じインスタンスを指す(と考えてよい)

Pythonの「位置引数」に対応

Pythonの「キーワード引数」例: (dx=10, dy=25)はない

メソッドの呼び出し側

```
r2.move(20,10);
```

実引数1　実引数2

```
r2.move(x,y);
```

x,yはint型. x==20,y==10のとき

メソッドの定義側

順番に対応する

```
public class Rect {
    public void move(int dx, int dy) {
this=r1 こう代入される  dx=20; dy=10;
        this.x1 += dx; this.x2 += dx;
        this.y1 += dy; this.y2 += dy;
    }}
```

仮引数1　仮引数2

実際のプログラムには書かないこと！

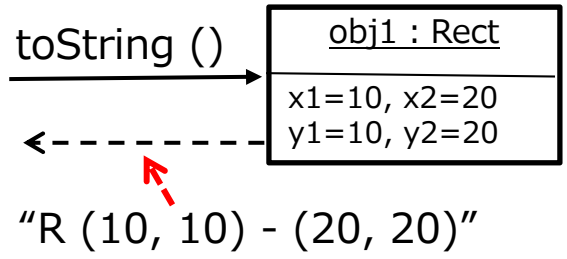
メソッドの定義例３：状態の表示

- インスタンスの状態を文字列に変換するメソッド
  - インスタンスの状態=全フィールドの値
  - フィールドの値を１つの文字列に変換して返す.
    - Java では文字列は String クラス で表せる.
    - toString() という名前のメソッドとして定義する.
  - 戻り値を System.out.println() すれば状態が表示される.

戻り値の型が String クラス型

@Overrideの意味は後述

```
@Override
public String toString() {
    return ("R (" + this.x1 + ", "
    + this.y1 + ")-(" + this.x2 + ", "
    + this.y2 + ")");
    }
```



文字列を連結する演算子

フィールドの値を取り出す（暗黙に文字列に変換される）

Rectクラスの定義例(OOP1/2-A)

OOP1/2-A
Rect.java

```
public class Rect {
   private int x1, y1, x2, y2;

   public Rect(int x1,int y1,int x2,int y2) {
      this.x1 = x1; this.x2 = x2;
      this.y1 = y1; this.y2 = y2;
      }
   public int getWidth() {
      return (this.x2 - this.x1)
      }
   public void move(int dx,int dy) {
      this.x1 += dx; this.x2 += dx;
      this.y1 += dy; this.y2 += dy;
      }
   @Override
   public String toString() {
       return ("R (" + this.x1 + ", "
       + this.y1 + ")-(" + this.x2 + ", "
       + this.y2 + ")");
       }
```

フィールド定義

コンストラクタ

左上(x1,y1),右下(x2,y2)
のインスタンスを生成する

自分の幅を返す

インスタンス
メソッド

右下方向に(dx, dy)
だけ移動させる

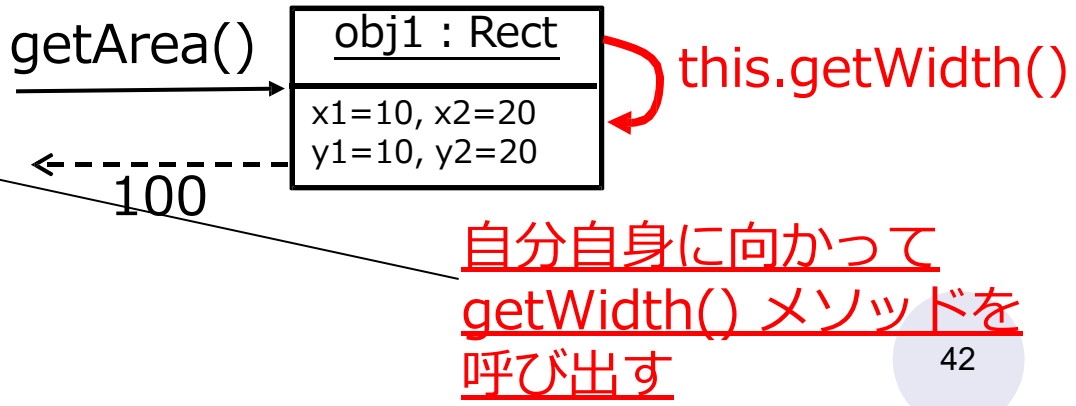
座標値を表す文字列
を返す（表示用）

インスタンスメソッド内でのメソッド呼び出し

- メソッド内で同じクラスの他のメソッドを呼び出す
  - 例：`double` **`getArea()`**：引数なし，自分の面積(double型)を返す
  - 自分(this)に向けて，他のメソッドを呼ぶ
    - this.<メソッド名>(引数)で呼ぶ．例：**`this.getWidth()`**
    - this は省略可能だが、付けることを強く推奨．
      - 自分のクラス（Rectクラス）のメソッドであることが分かるから．
      - 継承がある場合は他クラスで定義されている場合もある（後述）

自分の「面積」(double型の値）を返す

```
public double getArea() {
    int w = this.getWidth();
    ...
    }
```



参考

キャストと代入（Java言語）

- キャスト＝明示的な型変換
  - 「(型名) 変数名」と書くと，変数の値が，もともとの変数の型から指定された型に変換される．（正確には，変数名→右辺値）
- 型がプリミティブ型の場合
  - Javaのプリミティブ型：int (整数：byte, short, int, long), float, double（実数）, boolean（論理値), char（1文字)
  - もともとの型よりも大きい型の変数へ代入するときは，自動で変換されるので，キャストは必要ではないが，キャストした方がベター．
  - 小さい型の変数へ代入するときには，明示的にキャストする必要がある（コンパイルエラーになる）．また，桁あふれに注意．

```
short shortNum=0;
long longNum=1000000;
longNum = shortNum;
shortNum = (short) longNum;
```

- 変数がクラス型の場合もキャストできる（後述）
  - 上位クラス型の変数へはキャストなしに代入できる．有用（後述）
  - 下位クラス型の変数へも代入できるが注意が必要（後述）

今回の講義のまとめ

- オブジェクト指向プログラミング(2)：基本
  - Javaの文法の基礎
    - 変数（型）の宣言と利用，条件分岐等
  - クラスの利用：メソッドの呼び出し
    - コンストラクタの利用
      - インスタンス生成とクラス型変数
    - インスタンスメソッドの呼び出し
  - 今回 → クラスの定義
    - フィールドの定義
      - ローカル変数の違い
    - メソッドの定義
      - コンストラクタの定義
      - インスタンスメソッドの定義
      - インスタンスフィールドの参照と代入
      - 自分のインスタンスメソッドの呼び出し