

# オブジェクト指向論(Q)

第9回講義

(プログラミング#3 OOP3)

2023/6/5

來村 徳信

# 今回の講義のテーマと流れ

- クラス間の継承

- ➡ ○ 上位クラス・下位クラス
  - 下位クラスの定義
    - コンストラクタの定義
    - 上位クラス型変数への代入
  - メソッドの定義と継承
    - (a) メソッドの継承と呼び出し
      - 上位クラスのみでメソッドが定義されている場合
    - (b) メソッドオーバーライド
      - 下位クラスで上位クラスと同名のメソッドが定義されている場合
      - インスタンスの所属クラスによる動的メソッド呼び出し
      - 下位クラスからの上位クラスの名義メソッドの呼び出し
    - (c) 下位クラスのためのメソッド
  - フィールドのアクセス制御
    - private, public, protected の使い分け

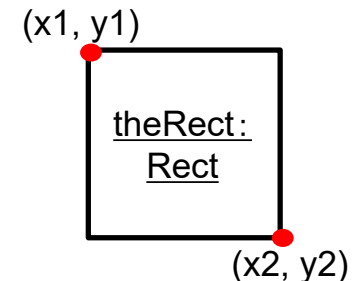
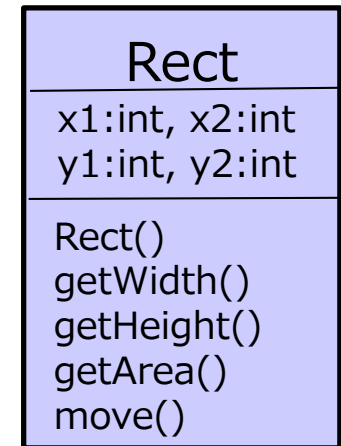
# Rectクラスの仕様

## ● フィールドの仕様

- 整数(int)型の  $x1, x2, y1, y2$ . 左上座標( $x1, y1$ ) と右下の座標( $x2, y2$ )を表す
- すべて正の値で,  $x2 > x1$  かつ  $y2 > y1$  とする

## ● メソッドの仕様

- `Rect(int x1, int y1, int x2, int y2)`
  - コンストラクタ. 左上座標( $x1, y1$ ), 右下座標( $x2, y2$ )の四角形インスタンスを生成し, それへの参照を返す.
- `int getWidth()`
  - インスタンスのX軸方向の幅 (正の整数) を返す
- `int getHeight()`
  - インスタンスのY軸方向の高さ (正の整数) を返す
- `double getArea()`
  - インスタンスの面積 (正の実数) を返す
- `void move(int dx, int dy)`
  - 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.
- `String toString()`
  - 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.



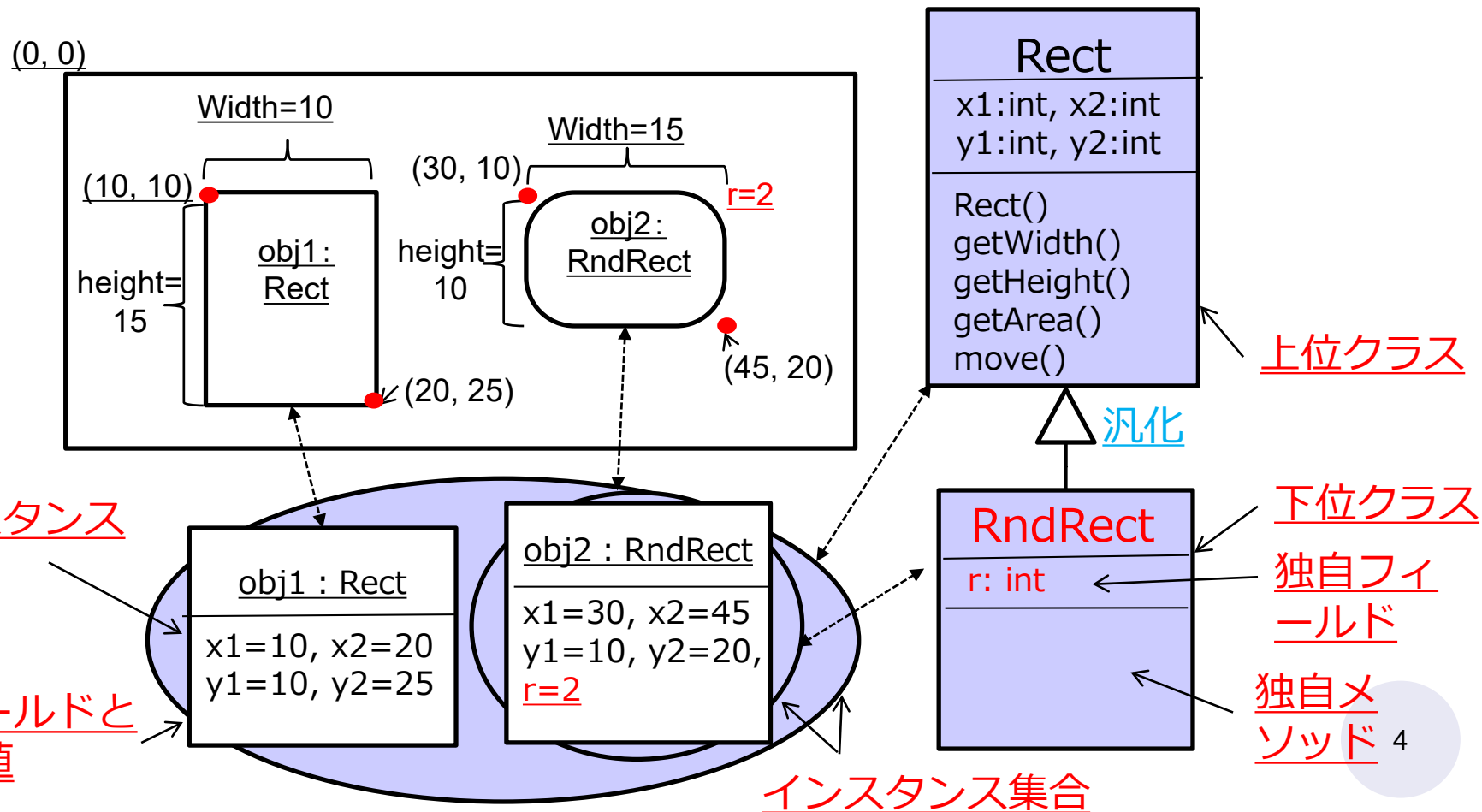
# 上位／下位クラスとインスタンス

- 共通性質を持つ上位クラス

UMLでの用語

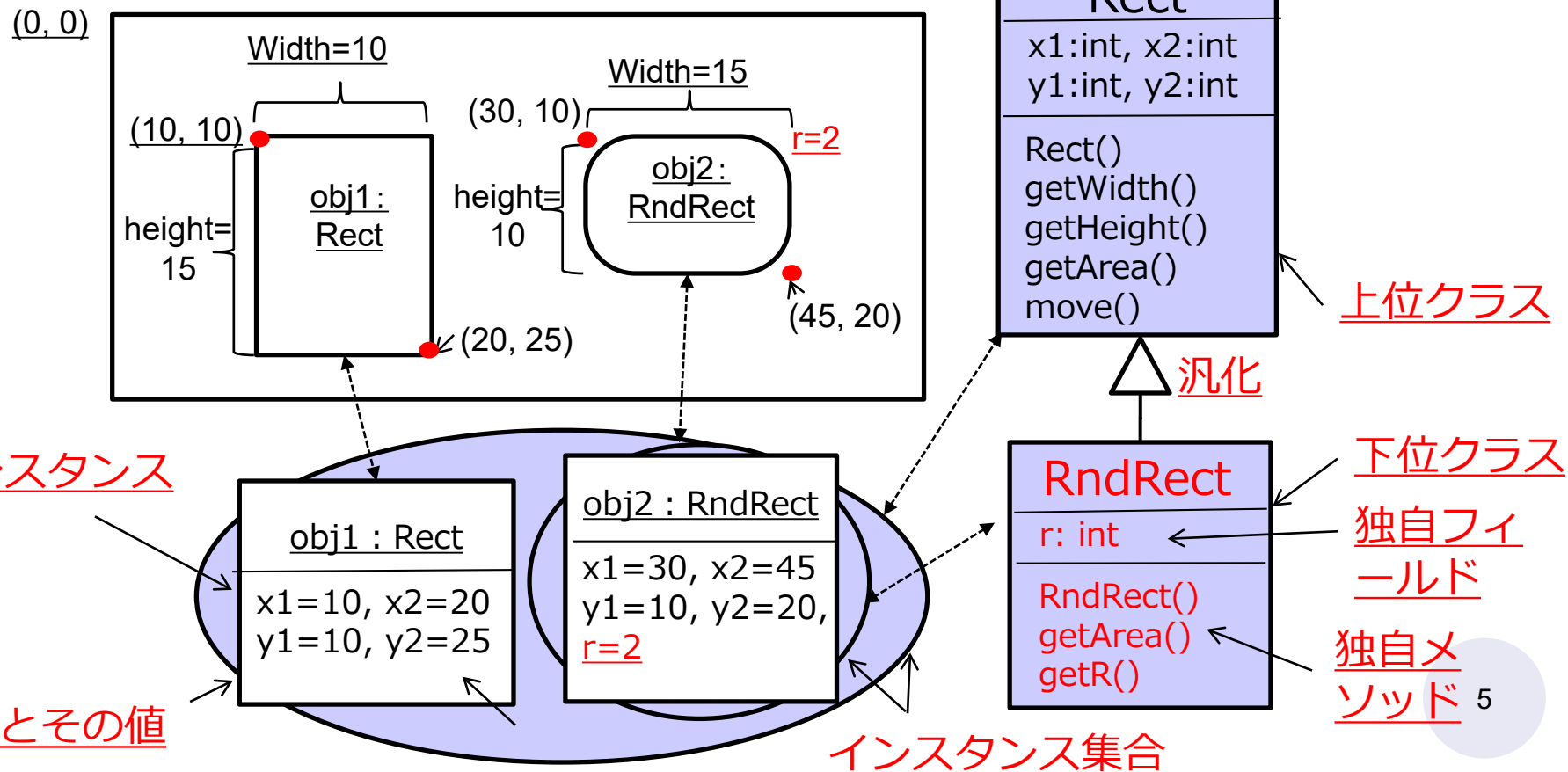
- 下位クラスは上位クラスを継承し、独自の性質を持つ(特化)

- 例：下位クラス：角丸四角形(RndRect)は、四角形(Rect)クラスのフィールドに加えて、角半径(r)フィールドを持つ



# 上位／下位クラスのメソッド

- 上位のものがそのまま使える共通メソッドはどれ？
  - 共通：[ getWidth(), getHeight(), move() ]
  - 非共通：[ getArea() ] + コンストラクタ(RndRect())
  - 独自：[ getR() ]



Pythonでは class 下位クラス名(上位クラス名) と書く

# 下位クラスの定義

- 「class 下位クラス **extends** 上位クラス」 と書く
  - 上位クラスを「**拡張**」して, 下位クラスを定義する
    - 上位クラスは**基底**クラス, 下位クラスは**派生**クラスとも呼ばれる
  - 基本的に, 下位クラスは, 上位クラスの定義内容 (フィールドやメソッド) を「**継承**」する (以下で説明する)
    - アクセスや呼び出しには制限がある (後述する)

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

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

**四角形**

obj1 : Rect

x1=10, x2=20  
y1=10, y2=25

**角丸**  
**四角形**

obj2 : RndRect

x1=30, x2=45  
y1=10, y2=20,  
**r=2**

```
public class RndRect extends Rect {
    private int r;
    ...
}
```

**下位クラスの**  
**独自フィールド**

# コンストラクタの定義

- クラス名と同名のメソッド（前回と同じ）
- 通常、下位クラスのコンストラクタは、上位クラスのコンストラクタを super(...) で呼び出したあと、独自の処理を行う。

```
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 class RndRect extends Rect {  
    private int r;  
  
    public RndRect(int x1,int y1,int x2,int y2,int r) {  
        super(x1,y1,x2,y2); ← 追加された仮引数  
        this.r=r; ← 上位クラスのコンストラクタの呼び出し（特殊構文. この位置）  
    }  
}
```

このクラス固有の処理

# コンストラクタの定義（参考）

- superを明示的に書かなくても自動的に呼ばれる.
- 明示的に `super()` を呼び出すことを推奨.

```
public class Rect {  
    protected int x1, y1, x2, y2;  
  
    public Rect() {} ← 定義されていないとコンパイルエラーになる  
    public Rect(int x1,int y1,int x2,int y2) {  
        this.x1 = x1; this.x2 = x2;  
        this.y1 = y1; this.y2 = y2;  
    }  
}
```

※この場合, protected 指定に  
する必要がある. 理由は後述

```
public class RndRect extends Rect {  
    public int r;  
  
    public RndRect(int x1,int y1,int x2,int y2,int r) {  
        ← ※このタイミングで暗黙的に super() (引数なし) が呼ばれる  
        this.x1 = x1; this.x2 = x2;  
        this.y1 = y1; this.y2 = y2; this.r=r;  
    }  
}
```



# RndRect の定義(第1版 : OOP3-A)

- ほぼコンストラクタの定義のみ

```
Public class RndRect extends Rect {  
    private int r;  
    public RndRect(int x1,int y1,int x2,int y2,int r) {  
        super(x1,y1,x2,y2);  
        this.r=r;  
    }  
    @Override ※@Override については後述  
    public double getArea() {  
        double ca = this.r*this.r*4  
            - (this.r*this.r*Math.PI);  
        return ((double)0);  
    }  
    @Override  
    public String toString() {  
        return ("Rnd" + super.toString() + ", r=" + this.r);  
    }  
}
```

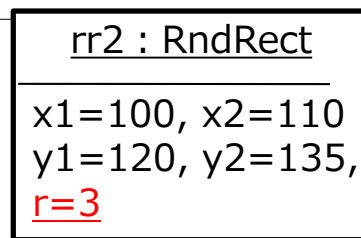
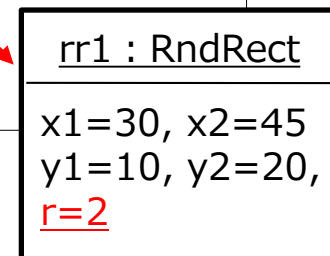
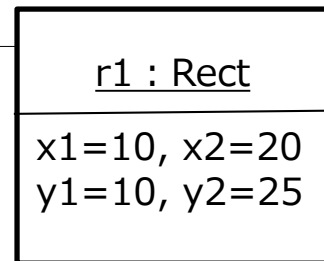
※コンパイルエラーを防ぐために  
0を返している. 後で, 変数 ca を  
使って, ちゃんと定義する.

※super. については後述

# コンストラクタの呼び出し

- クラス型変数への参照の代入（前回と同じ）
  - メソッドの呼び出し方やフィールドへのアクセス方法も同じ
    - RndRect型のクラス型変数にインスタンスへの参照を格納する.
  - 上位クラスを型とするクラス型変数へも代入可能 — **NEW**
    - 下位クラスのインスタンスは、常に上位クラスのインスタンスでもあるから.

```
public class Main {  
    public static void main(String[] args) {  
        Rect r1 = new Rect(10,10,20,25);  
        RndRect rr1 = new RndRect(30,10,45,20,2);  
        Rect rr2 = new RndRect(100,120,110,135,3);  
    }  
}
```



Rect型でも可.  
オブジェクト指向の効用の一つ.  
※ただし制限あり（後述）

# 今回の講義のテーマと流れ(再掲1)

- クラス間の継承

- 上位クラス・下位クラス

- 下位クラスの定義

- コンストラクタの定義

- 上位クラス型変数への代入

- メソッドの定義と継承



- (a) メソッドの継承と呼び出し

- 上位クラスのみでメソッドが定義されている場合

- (b) メソッドオーバーライド

- 下位クラスで上位クラスと同名のメソッドが定義されている場合

- インスタンスの所属クラスによる動的メソッド呼び出し

- 下位クラスからの上位クラスの名義メソッドの呼び出し

- (c) 下位クラスのためのメソッド

- フィールドのアクセス制御

- private, public, protected の使い分け

# メソッドの継承と呼び出し(a)

- (a) 同名のメソッドが下位クラスで定義されていない場合
  - 上位クラスのメソッドが下位クラスに継承される.
  - (1) 下位クラスのインスタンスへメソッドを呼び出すと、実際には、上位クラスのメソッドが呼び出される。
  - 同じ処理は一度だけ記述すればよい(OOPの最大の効果)

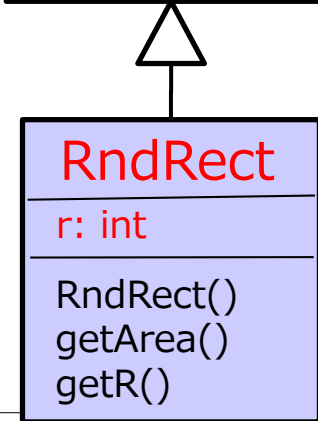
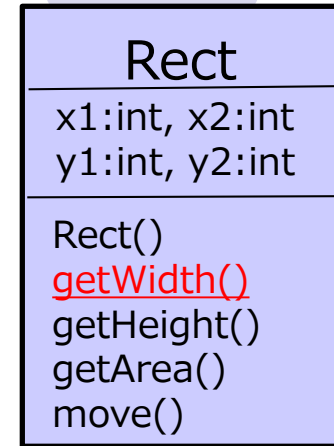
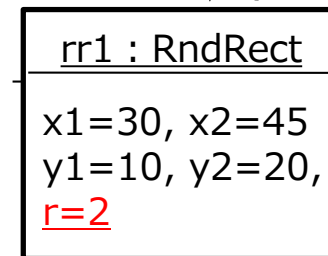
```
public class Rect{  
    private int x1, y1, x2, y2;  
    public int getWidth() {  
        return (this.x2-this.x1);  
    }  
}
```

```
public class RndRect extends Rect {  
    private int r;  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        RndRect rr1 = new RndRect(30,10,45,20,2);  
        int rr1w = rr1.getWidth();  
    }  
}
```

継承メソッド (RectクラスのgetWidthメソッド) が実行される)

getWidth() | ↑ 15  
rr1



# メソッドの継承と呼び出し(a)

- (a) 同名のメソッドが下位クラスで定義されていない場合
  - 上位クラスのメソッドが下位クラスに継承される。
  - (1) 下位クラスのインスタンスへメソッドを呼び出すと、実際には、上位クラスのメソッドが呼び出される。
  - Rect型変数に代入していても、同様

```
public class Rect{  
    private int x1, y1, x2, y2;  
    public int getWidth() {  
        return (this.x2-this.x1);  
    }  
}
```

```
public class RndRect extends Rect {  
    private int r;  
}
```

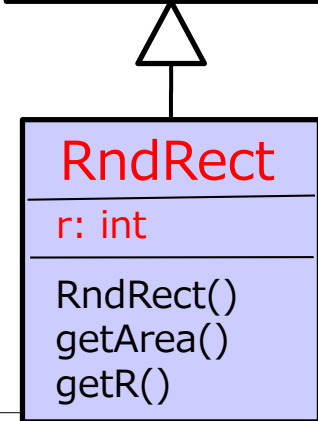
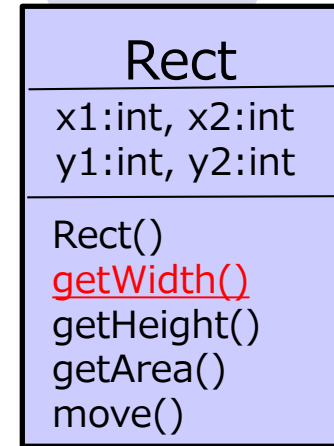
```
public class Main {  
    public static void main(String[] args) {  
        Rect rr2 = new RndRect(100,120, 150,140, 5);  
        int rr2w = rr2.getWidth();  
    }  
}
```

継承メソッド (RectクラスのgetWidthメソッド) が実行される)

getWidth()  
rr1

15

rr1 : RndRect  
x1=30, x2=45  
y1=10, y2=20,  
r=2



# メソッドの継承と呼び出し(a)

- (a) 同名のメソッドが下位クラスで定義されていない場合
  - (2) 下位クラスの定義内でも, 上位クラスのメソッドを, 自分のクラスのメソッドのように呼びだせる.
  - 参照されるフィールドの値はあくまで RndRectインスタンスのもの

```
public class Rect{  
    private int x1, y1, x2, y2;  
    public int getWidth() {  
        return (this.x2-this.x1);  
    }  
}
```

rr2

rr2 : RndRect

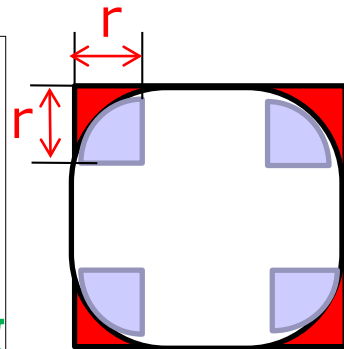
x1=100, x2=110  
y1=120, y2=135,  
r=3

getArea() の第1版

```
public class RndRect extends Rect {  
    private int r;  
    public double getArea() {  
        double ca = this.r*this.r*4  
        - (this.r*this.r*Math.PI);  
        return ( (  
            - ca ) );  
    }  
}
```

赤の角丸の部分  
の面積

円周率を表す定数



自分の四角形としての面積を求める。継承メソッドの呼び出し<sup>14</sup>  
(自分に向けて getWidth, getHeightメソッドを呼び出す)

# クラスメソッドとクラスフィールド

- 定数を表すクラスフィールド

- 例 : double **Math.PI**
- クラス Math のフィールド(static). インスタンスと無関係.
- 全て大文字で定数(final)であることを表す.

- プリミティブ型用のクラスメソッド

- 例 : double **Math.sqrt**(double d)
  - 引数の実数 d の平方根を返す関数

```
double d = 9.0;  
double d2 = Math.sqrt(d) ;  
System.out.println("Sqrt of " + d + " = " + d2);
```

- 実数(double)型はプリミティブ型. クラスではない.
  - メソッドを定義しようにも, クラスがないので, Math という特殊なクラスのクラスメソッドとして定義している.
  - 呼び出す処理対象のインスタンスがないので,  
先頭が大文字のクラス名を. の前に付けて呼ぶ.

# 今回の講義のテーマと流れ(再掲2)

- クラス間の継承

- 上位クラス・下位クラス

- 下位クラスの定義

- コンストラクタの定義

- 上位クラス型変数への代入

- メソッドの定義と継承

- (a) メソッドの継承と呼び出し

- 上位クラスのみでメソッドが定義されている場合

- (b) メソッドオーバーライド

- 下位クラスで上位クラスと同名のメソッドが定義されている場合

- インスタンスの所属クラスによる動的メソッド呼び出し

- 下位クラスからの上位クラスの名義メソッドの呼び出し

- (c) 下位クラスのためのメソッド

- フィールドのアクセス制御

- private, public, protected の使い分け



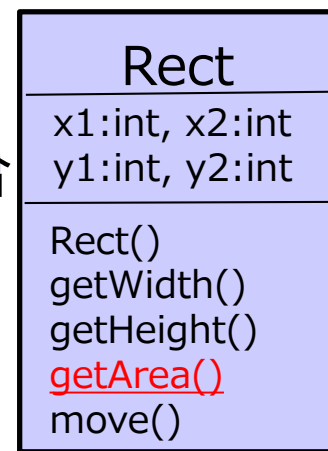
# メソッドの継承と呼び出し(b)

○ (b)同名のメソッドが下位クラスで定義されている場合

- 「メソッド **オーバーライド**」と呼ばれる
- メソッド名, 引数, 戻り値の全てが一致する場合

```
public class Rect {  
    public double getArea() {  
        return ((double)this.getWidth() *  
            this.getHeight());  
    }  
}
```

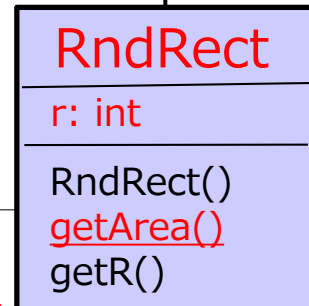
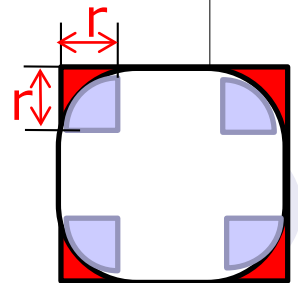
四角形の面積を  
求めるメソッド



```
public class RndRect extends Rect {  
    private int r;  
    @Override public double getArea() {  
        double ca = this.r*this.r*4  
            - (this.r*this.r*Math.PI);  
        return ((this.getWidth()*this.getHeight()  
            - ca);  
    }  
}
```

上位クラスのメソッドをオーバーライドしている  
ことを表すアノテーション. コンパイラがチェック  
してくれるので, 強く推奨. 角丸四角形の面積を  
求めるメソッド第1版

赤の角丸の部分  
の面積



# メソッドの継承と呼び出し(b)

- (b) 同名のメソッドが下位クラスで定義されている場合：
  - 呼び出されると、そのインスタンスが「直接, 所属するクラス」によって異なるメソッドが実行される。
    - 例：面積が求められる、という意味を共有している。
  - 呼び出す側はどのメソッドが呼ばれるか気にしなくてよい。
    - 「情報の隠蔽化」, 「カプセル化」

```
public class Main {  
    public static void main(String[] args) {
```

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

```
        RndRect rr1 = new RndRect(30, 10, 45, 20, 2);
```

```
        double r1a = r1.getArea();;
```

```
        double rr1a = rr1.getArea();;
```

```
    }}
```

Rectクラスのメソッドが呼ばれる

RndRectクラスのメソッドが呼ばれる

# メソッドの継承と呼び出し(b)

- (b)同名のメソッドが下位クラスで定義されている場合：
  - 呼び出されると、そのインスタンスが「直接、所属するクラス」によって異なるメソッドが実行される。
    - 例：面積が求められる、という意味を共有している。
  - 呼び出す側はどのメソッドが呼ばれるか気にしなくてよい。
  - ➡ ○上位クラスであるRect型変数に代入していても、正しくRndRect クラスの getArea() が呼ばれる。

```
public class Main {  
    public static void main(String[] args) {
```

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

```
        Rect rr2 = new RndRect(100,120,110,135,3);
```

```
        double r1a = r1.getArea(); ← Rectクラスのメソッドが呼ばれる
```

```
        double rr2a = rr2.getArea();
```

```
    } }
```

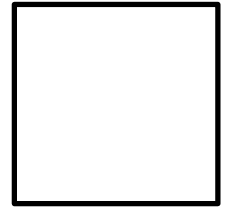
← RndRectクラスのメソッドが呼ばれる

# メソッドの継承と呼び出し(b)

- (b)同名のメソッドが下位クラスで定義されている場合：
  - 下位クラス内のメソッドで、上位クラスの名義メソッドを呼び出す場合は、**super.<メソッド名>**と書く。

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

四角形の面積を  
求めるメソッド



```
    public double getArea() {  
        return ((double) (this.getWidth() * this.getHeight()));  
    }  
}
```

```
public class RndRect extends Rect {  
    private int r;
```

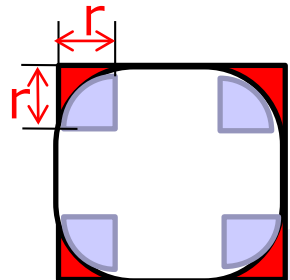
角丸四角形の面積を  
求めるメソッド  
(第2版)

```
    public double getArea() {  
        double ca = this.r * this.r * 4  
            - (this.r * this.r * Math.PI);  
        return (super.getArea() - ca);  
    }  
}
```

赤の角丸の  
部分の面積

四角形の面積から

を引く



# 今回の講義のテーマと流れ(再掲3)

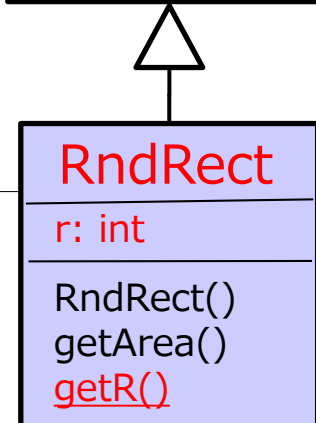
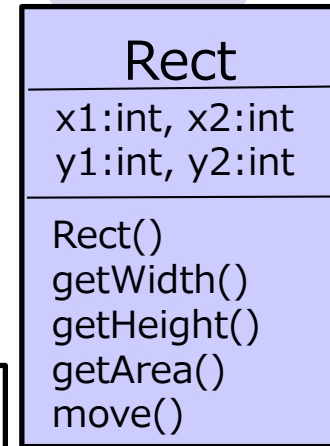
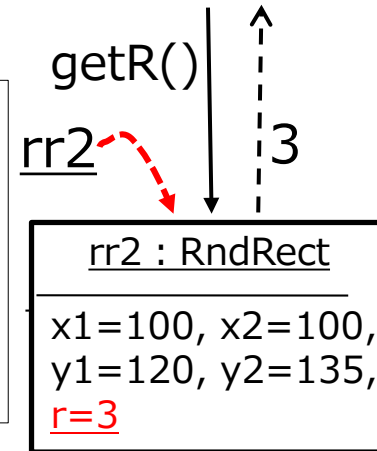
- クラス間の継承

- 上位クラス・下位クラス
- 下位クラスの定義
  - コンストラクタの定義
  - 上位クラス型変数への代入
- メソッドの定義と継承
  - (a) メソッドの継承と呼び出し
    - 上位クラスのみでメソッドが定義されている場合
  - (b) メソッドオーバーライド
    - 下位クラスで上位クラスと同名のメソッドが定義されている場合
    - インスタンスの所属クラスによる動的メソッド呼び出し
    - 下位クラスからの上位クラスの名義メソッドの呼び出し
  - (c) 下位クラスのためのメソッド
- フィールドのアクセス制御
  - private, public, protected の使い分け

# メソッドの継承と呼び出し(c)

- (c) 下位クラスのみでメソッドが定義されている場合
  - RndRect型のクラス型変数で参照される必要がある。
  - Rect型だとコンパイルが通らない。キャストすればOK。
    - しかし危険なので、注意が必要。

```
public class RndRect extends Rect {  
    private int r;  
    public int getR() {  
        return this.r;  
    }  
}
```



```
public class Main {  
    public static void main(String[] args) {  
        RndRect rr1 = new RndRect(30,10,40,20,2);  
        int rr1r = rr1.getR();  
        Rect rr2 = new RndRect(100,120,110,135,3);  
        int rr2r = rr2.getRX;  
        int rr2r = ((RndRect) rr2).getR();  
          
        rr2をRndRect型へ変換 (キャスト)  
    }  
}
```

# 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）

# RndRect の定義例 (OOP3-B)

```
public class RndRect extends Rect {
    private int r;

    public RndRect(int x1,int y1,int x2,int y2,int r) {
        super(x1,y1,x2,y2);
        this.r=r;
    }

    public int getR() {return this.r;}
```

@Override

```
public double getArea() {
    double ca = this.r*this.r*4
        - (this.r*this.r*Math.PI);
    return (super.getArea() - ca);
}
```

@Override

```
public String toString() {
    return ("Rnd" + super.toString() + ", r=" + this.r);
}
```

*String toString() は java.lang.Object (すべてのクラスの最上位クラス) で定義されている. それをオーバーライドすることで, そのクラス固有の内容の文字列を返すことができる.*

*Rectクラスの toString() を呼び出している.  
x1,y1,x2,y2 は同じだから.*

*RndRectクラス  
固有の処理*



# 今回の講義のテーマと流れ(再掲4)

- クラス間の継承

- 上位クラス・下位クラス

- 下位クラスの定義

- コンストラクタの定義

- 上位クラス型変数への代入

- メソッドの定義と継承

- (a) メソッドの継承と呼び出し

- 上位クラスのみでメソッドが定義されている場合

- (b) メソッドオーバーライド

- 下位クラスで上位クラスと同名のメソッドが定義されている場合

- インスタンスの所属クラスによる動的メソッド呼び出し

- 下位クラスからの上位クラスの名義メソッドの呼び出し

- (c) 下位クラスのためのメソッド

- ➡ フィールドのアクセス制御

- private, public, protected の使い分け

# フィールドのアクセス制御：基本

- アクセス修飾子によってアクセスの可否が決まる
  - フィールド宣言時のアクセス修飾子：
    - **private** : クラス外から参照／変更できない
    - **protected** : 下位クラス（と同じパッケージ内）から参照できる
    - **public** : クラス外から参照／変更できる

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

```
public class RndRect {  
    private int r;  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Rect r1 = new Rect(10, 10, 20, 20);  
        RndRect rr1 = new RndRect(30, 10, 45, 20, 2);  
        System.out.println("r1= " + r1.x1 + r1.y1);  
        System.out.println("rr1 r= " + rr1.r);  
    }  
}
```

OK

コンパイルエラー


「rはRndRectでprivateアクセスされます」

# Public フィールドの問題点

- 他クラスから直接フィールドを参照/変更できる
  - Rectクラスの実装方法に，他のクラスが依存してしまう。
    - 四角形を左上座標と幅と高さで表すように変更したら？
- Private 指定することで「情報の隠蔽化」（メソッドも含めて「カプセル化」）ができる

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

アクセス修飾子



変更したら？

```
public class Rect {  
    public int x,y,  
        width,height;  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Rect r1 = new Rect(10,10,20,20);  
        r1.x1 += 5; r1.x2 += 5;  
        r1.y1 += 5; r1.y2 += 5;  
    }  
}
```

← r1.move(5,5)の代わりに直接，  
座標値を変更できてしまう。  
変更したら動かなくなる。

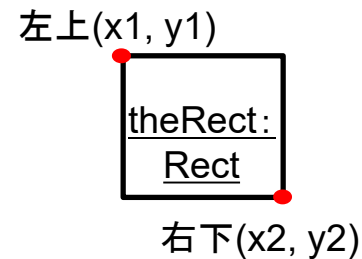
# アクセサ(accessor)

- Private なフィールドに対するアクセスを提供するメソッド.
  - 値を取り出すメソッド(getXXX(),ゲッター)
  - 値をセットするメソッド(setXXX(),セッター)
- 最小限にすることを推奨

```
public class Rect{  
    private int x1, y1, x2, y2;  
  
    public int getX1() {return this.x1;}  
    public int getY1() {return this.y1;}  
  
    public void setX1Y1(int x1, int y1) {  
        this.x1 = x1;  
        this.y1 = y1;  
    }  
}
```

左上座標値のゲッター

左上座標値のセッター



# 上位／下位クラスのフィールドのアクセス制御

- アクセス修飾子によって決まる
  - Public : 全てのクラスから参照可能
    - 非推奨
  - Protected : そのクラスの下位クラス, または同じパッケージ内からのみ参照可能
    - 基本的に, 下位クラスからのアクセスに限定するために用いる. 下位クラスから見れば, Public と同じ
    - パッケージを使わなければ, Public と同じ
      - ※本講義ではパッケージを使わないが, 必要な場合 Protected を使う
  - Private : そのクラス内からのみ参照可能 <基本的に推奨>
    - 下位クラスからも不可.
- いずれの場合もフィールド自体は継承される
  - 直接アクセスできるかどうかだけの違い

# Protectedフィールドへのアクセス(1)

- 同じ名前のフィールドが下位クラスで定義されていない場合（普通の場合）：
  - フィールド名を用いて、直接アクセスできる
    - public/protected 宣言されたフィールドのみ.
    - private は不可（後述）

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

四角形

obj1 : Rect

x1=10, x2=20  
y1=10, y2=25

```
public class RndRect extends Rect {  
    private int r;  
  
    public double getArea() {  
        return ((this.x2 - this.x1) * ... - this.r ...)  
    }  
}
```

上位クラスと同じフィールド名でアクセス  
(値自体は自クラスのインスタンスのもの)

角丸  
四角形

obj2 : RndRect

x1=30, x2=45  
y1=10, y2=20,  
r=2

# Protectedフィールドへのアクセス(2)

- 同じ名前のフィールドが下位クラスで定義されている場合（**非推奨**）：
    - 下位クラスで定義されたフィールドによって、上位クラスの名義フィールドが「覆い隠される」
    - フィールド名またはthis.フィールド名は下位クラスのもの
    - super.フィールド名は上位クラスのフィールドを指す
- ※紛らわしいので**非推奨**

```
public class Rect{  
    protected int x1, y1, x2, y2, r;  
}
```

上位クラス：  
例：画面倍率？

```
public class RndRect extends Rect {  
    private int r;  
    public double getArea() {  
        return (super.r * ((x2-x1) * ...  
        - r * this.r));  
    }  
}
```

下位クラス：  
角丸の半径

下位クラスのフィールド

上位クラスのフィールド

obj2 : RndRect

x1=30, x2=45  
y1=10, y2=20,  
r=2, super.r=0

# Privateフィールドへのアクセス(1)

- Private指定のフィールドはそのクラス内でしか参照できない
  - 下位クラスからも不可.
- 下位クラスから上位クラスのフィールドは直接参照できない
  - コンパイル時にエラーになる. `super` を付けてもダメ
  - フィールド自体は継承されている. 下位クラスのメソッド内から直接アクセスできないだけ.

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

```
public class RndRect extends Rect {  
    private int r;  
  
    public double getArea() {  
        return ((this.x2 - super.x1) * (y2 - y1))  
    }  
}
```

いずれもコンパイル時にエラーになる

obj2 : RndRect

x1=30, x2=45  
y1=10, y2=20,  
r=2



# Privateフィールドへのアクセス(2)

- ではどうするのか(1) : アクセサメソッドの利用
  - 上位クラスのアセセサメソッドを用いて, 値を取り出す／変更する
  - フィールドの値自体は, 自分のクラスのインスタンスが保持していることに注意. 直接アクセスできないだけ.

```
public class Rect{  
    private int x1, y1, x2, y2;  
  
    public int getX1() {  
        return (this.x1);  
    }  
}
```

obj2 : RndRect

x1=30, x2=45  
y1=10, y2=20,  
r=2

```
public class RndRect extends Rect {  
    private int r;  
    public double getArea() {  
        return ((... - this.getX1()) * ...);  
    }  
}
```

アクセサ (ゲッター) の利用

# Privateフィールドへのアクセス(3)

- ではどうするのか(2)：適切なメソッドの利用
  - 一般的には、フィールドに直接アクセスしなくてもよいように、上位クラスで適切なメソッドが定義されているべき

```
public class Rect{  
    private int x1, y1, x2, y2;  
  
    public int getWidth() {  
        return (this.x2-this.x1);  
    }  
}
```

[前述の `getArea\(\)` の第1版](#)

```
public class RndRect extends Rect {  
    private int r;  
    public double getArea() {  
        return (this.getWidth() * this.getHeight() -...)  
    }  
}
```

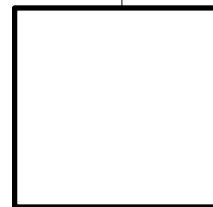
直接, `this.x1` などのフィールドを参照するのではなく, 幅と高さを得るメソッドを利用

# Privateフィールドへのアクセス(4)

- ではどうするのか(3) : メソッドオーバーライド
  - メソッドオーバーライドされた上位クラスの名義メソッドを呼び、下位クラスに固有の処理を足す。

```
public class Rect{  
    private int x1, y1, x2, y2;  
  
    public double getArea() {  
        return (this.getWidth()*this.getHeight());  
    }  
}
```

四角形の面積を  
求めるメソッド



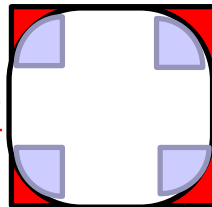
前述の getArea() の第2版

```
public class RndRect extends Rect {  
    private int r;  
    public double getArea() {  
        double ca = this.r*this.r*4  
            - (this.r*this.r*Math.PI);  
        return (super.getArea() - ca);  
    }  
}
```

角丸四角形の面積を  
求めるメソッド

上位クラス（四角形）の  
面積を得るメソッドを利用

赤の角丸の部分  
の面積を引く



# 今回の講義のまとめ

- クラス間の継承

- 上位クラス・下位クラス
- 下位クラスの定義
  - コンストラクタの定義
  - 上位クラス型変数への代入
- メソッドの定義と継承
  - (a) メソッドの継承と呼び出し
    - 上位クラスのみでメソッドが定義されている場合
  - (b) メソッドオーバーライド
    - 下位クラスで上位クラスと同名のメソッドが定義されている場合
    - インスタンスの所属クラスによる動的メソッド呼び出し
    - 下位クラスからの上位クラスの名義メソッドの呼び出し
  - (c) 下位クラスのためのメソッド
- フィールドのアクセス制御
  - private, public, protected の使い分け