# オブジェクト指向プログラミング

第5,6回：継承

知識科学教育研究センター
金井 秀明

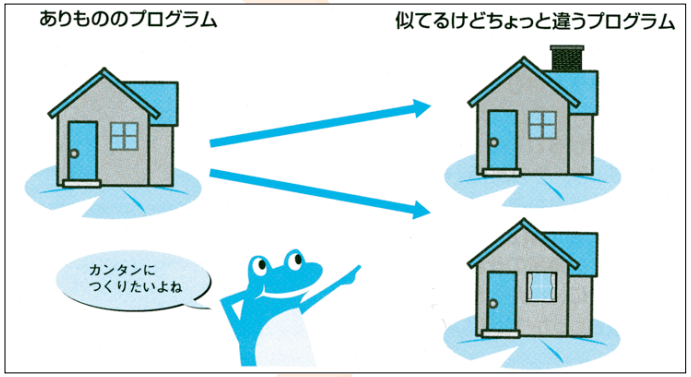


# 継承



## 継承(inheritance)#1

- 「継承」とは，"似てるけどチョット違う"クラス（プログラム）をより簡単に作成する仕組み




出典：立山秀利「Javaのオブジェクト指向がゼッタイにわかる本」秀和システム

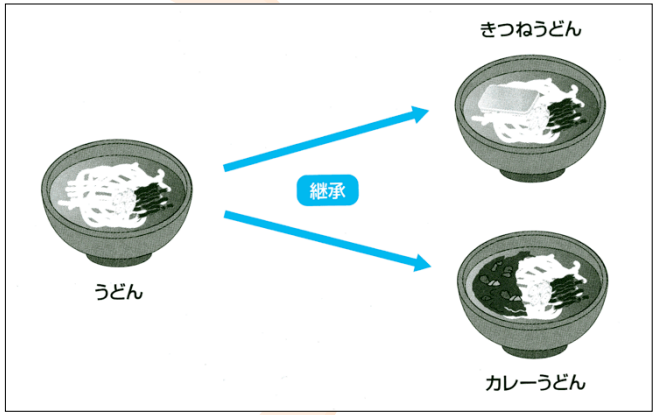


## 継承(inheritance)#2

- 継承は，あるクラスの機能を，他のクラスが引き継ぐ仕組み．
  - あるクラスのメンバ（フィールドやメソッドなど）を別のクラスが引き継ぐことができる．
  - 同じ部分の記述を省略して，違う部分だけを記述することができる．

# ○ 例：継承

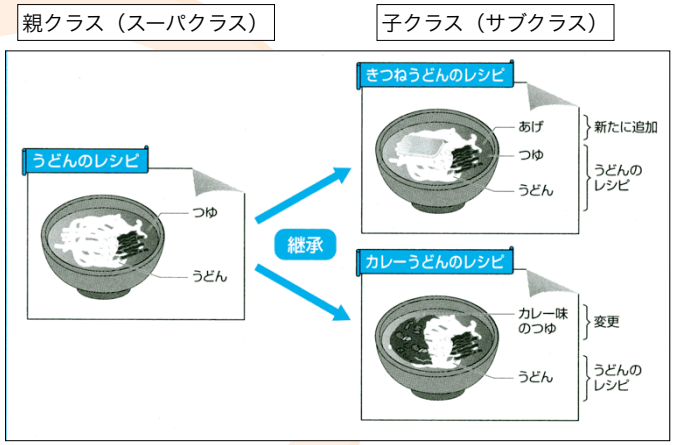
- 例




出典：立山秀利「Javaのオブジェクト指向がゼッタイにわかる本」秀和システム



# ○ 例：継承




出典：立山秀利「Javaのオブジェクト指向がゼッタイにわかる本」秀和システム

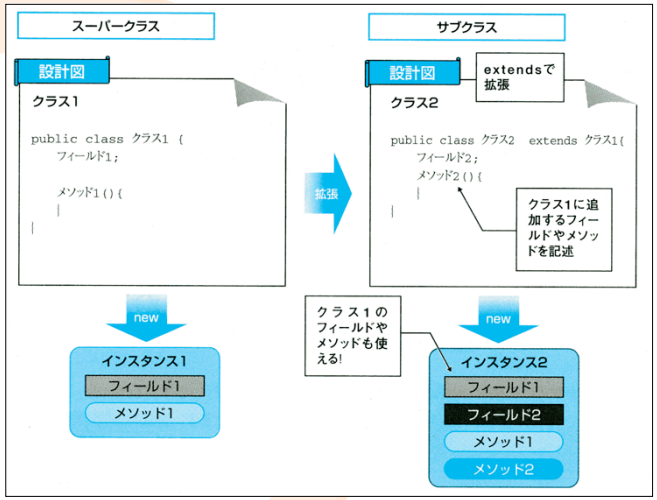


# ○ クラスの継承

```
class <サブクラス名> extends <スーパークラス名>
{
        型名  フィールド名1;
        . . .;
        戻り値型名  メソッド名(引数リスト) {
            . . .;
        }
        . . .;
}
```



# ○ 継承の様子




出典：立山秀利「Javaのオブジェクト指向がゼッタイにわかる本」秀和システム

## サブクラスのオブジェクトの作成

- いままでのオブジェクトの作成と同じ.
- オブジェクトを作成し，その変数を扱えるようにする.

```
クラス名 変数名= new クラス名
```

```
Human2 h = new Human2();
```



## 継承の例#1

スーパークラス「うどん」

```
class Udon {
        String men="太めん";
        void cooked(){System.out.println(men
+"をゆでる");}}}
```

サブクラス「きつねうどん」

継承される

```
class KitsuneUdon extends Udon {
        String item="油揚げ";
        String men="太めん"
        void cooked(){...}
}
```



## 継承の例#2-1

```
class Ape {
  String favorite=”バナナ”;
  void sleep() {System.out.println(”木の上でスヤスヤ”);
  }}
```

```
class Human extends Ape { }
class HumanMain{
  public static void main(String[] args) {
    Human h = new Human();
    System.out.println(h.favorite);
    h.sleep();
    }}
```



## 継承の例#2-2

```
class Human2 extends Ape {
  String item = ”パソコン”;
  void talk() {System.out.println(”おはよう”);
 }

class Human2Main{
  public static void main(String[] args) {
    Human2 h = new Human2();
    System.out.println(h.favorite); h.sleep();
    System.out.println(h.item); h.talk();
    }
}
```

# 継承の例#2-3

```
class Human3 extends Ape {
  void eatAndSleep() {
    System.out.println(favorite+"食べます");
    sleep();
    }
 }
class Human3Main{
  public static void main(String[] args) {
    Human3 h = new Human3();
    h.eatAndSleep();
    }
}
```

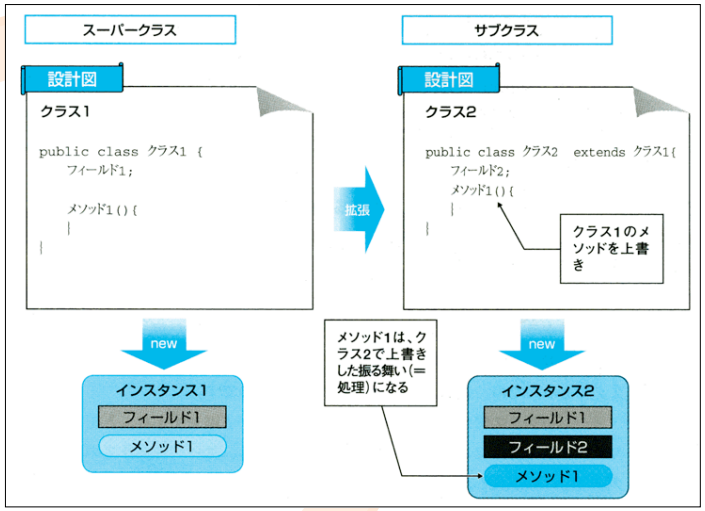


# メッソドのオーバーライド

- スーパークラスで定義されたメソッドと「同じメソッド名・引数の数・型」をもつメソッドをサブクラスで再定義すること.
- サブクラスのオブジェクでメソッドを呼び出すと，サブクラスのメソッドが呼び出される.



# オーバライドの様子




出典：立山秀利「Javaのオブジェクト指向がゼッタイにわかる本」秀和システム



# オーバライドの例#1

スーパークラス「うどん」

```
class Udon {
      String men="太めん";
      void cooked()
        {System.out.println(men+"をゆでる");}
}
```

サブクラス「きつねうどん」

```
class KitsuneUdon extends Udon {
 String item= "油揚げ";
 void cooked(){System.out.println(men+"を
 ゆでて, " +item+ " をのせる");}
}
```

オーバライド

# オーバライドの例#2

```
class Human4 extends Ape {
  void sleep() {
    System.out.println(”布団の上でふかふか”);
    }
 }
class Human4Main{
  public static void main(String[] args) {
    Human4 h = new Human4();
    h.sleep();
    }
}
```



# 練習１

- 会員クラス「Member」を定義せよ
  - フィールド：名前（name, String型）
  - フィールド：会員番号（no, int型）
  - フィールド：年齢（age, int型）
  - メソッド：print()：name,no,ageの情報を画面に出力する



# 練習２

- 練習１で定義したクラス「Member」を継承して，新たにクラス「SpecialMember」を作成せよ
  - フィールド：特典（privilege, String型）



# 練習３

- 練習２で定義したクラス「SpecialMember」にメソッドprint()をオーバライドせよ
  - メソッド：print()：name, no, ageおよびprivilegeの情報を画面に出力する

# 継承の禁止

- クラスやメソッドにfinal修飾子をつけると，継承を禁止できる．

```
final class ApeX{ }
class HumanX extends ApeX{}
```

```
class ApeX{
  final void sleep() {}
}
class HumanX extends ApeX{
  void sleep() {}
}
```



# finalキーワード

- finalでクラスを修飾すると
  - そのクラスを継承できない
- finalでメソッドを修飾すると
  - そのサブクラスからオーバーライドできない．
- finalでフィールドを修飾すると
  - その値を変更できない



# super

- スーパクラスのメンバに明示的にアクセスする方法

```
super.<フィールド名>
```

```
super.<メソッド名>
```

- スーパクラスのコンストラクタを呼び出す方法

```
super.(<引数>)
```



# super.メソッド名の例

```
class Ape {
  String favorite=”バナナ”;
  void sleep() {System.out.println(”木の上でスヤスヤ”);
  }}
```

```
class Human5 extends Ape {
  void sleep() {
    System.out.println(”ハンモックつきで”);
    super.sleep();
 }}
class Human5Main{
  public static void main(String[] args) {
    Human5 h = new Human5(); h.sleep();
}}
```

# this

- 自クラスのメンバに明示的にアクセスする方法

this.<フィールド名>　this.<メソッド名>

- 自クラスのコンストラクタを呼び出す方法

this.(<引数>)



# サブクラスのオブジェクトの作成

- 最初に，スーパクラスのコンストラクタが自動的に呼び出される．
- 指定しない：
  - 引数なしのコンストラクタが呼び出される．
- 指定：
  - super(引数リスト）より，呼び出すコンストラクタを指定できる．
  - コンストラクタ内の先頭に記述すること．



# コンストラクタの呼び出し#1

```
class Udon {
  Udon(){ ....}
  Udon(String m){men=m}
}

class KitsuneUdon extends Udon{
  KitsuneUdon() {}
  KitsuneUdon(String i) {item=i;}
  KitsuneUdon(String m, String i)
  {men = m; item=i;}
}
```

何も指定がないときには,スーパークラスの引数なしのコンストラクタ(=super())が最初に呼ばれる



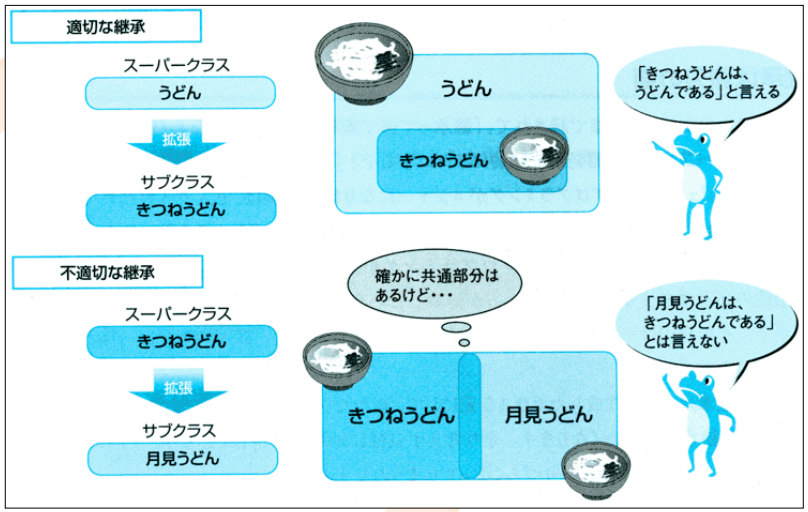
# コンストラクタの呼び出し#2

```
class Udon {
  Udon(){ ....}
  Udon(String m){men=m}
}

class KitsuneUdon extends Udon{
  Kitsuneudon() {}
  KitsuneUdon(String i) {item=i;}
  KitsuneUdon(String m, String i)
  {super(m);
   item=i;}
}
```

super()を使うと,目的のコンストラクタが呼べる

# 適切な継承:委譲




出典：立山秀利「Javaのオブジェクト指向がゼッタイにわかる本」秀和システム



# 練習4

- 練習1で定義したクラス「Member」のコンストラクタを定義せよ.
  - Member(String name, int no, int age)



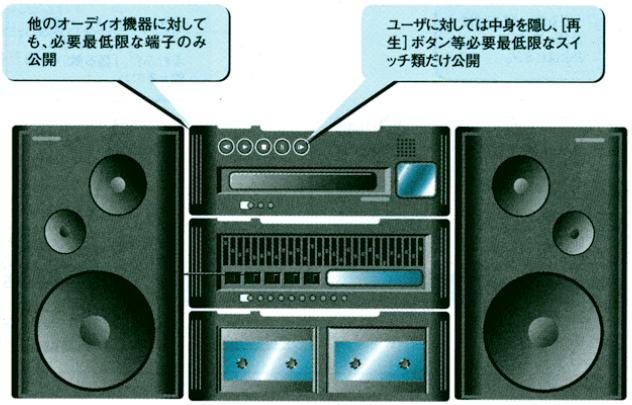
# 練習5

- 練習2で定義したクラス「SpcialMember」のコンストラクタを定義せよ.
  - ただし，SpecialMember(String name, int no, int age , String privilege）を定義する際には，練習４で定義したコンストラクタをsuper(...)で利用せよ.



# カプセル化

# ○ カプセル化#0




出典：立山秀利「Javaのオブジェクト指向がゼッタイにわかる本」秀和システム



# ○ カプセル化#0



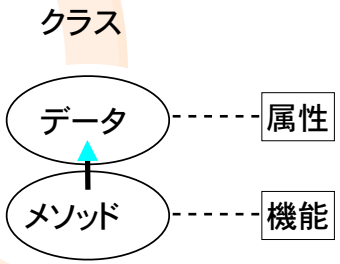


カプセル化



# ○ カプセル化#1

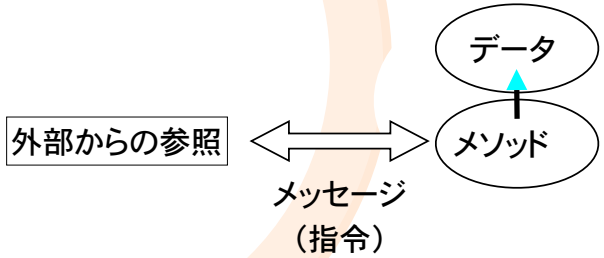
- 属性（データ，フィールド）と機能（その操作：メソッド）を一塊として，外部からのアクセスによる誤りがないように保護すること



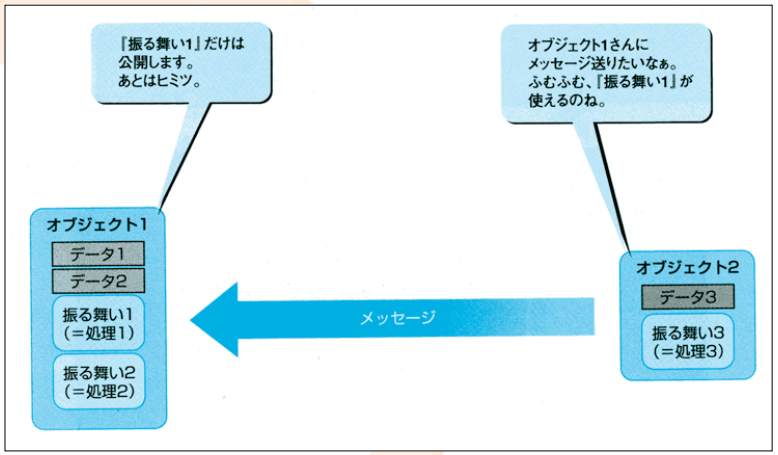



# ○ カプセル化#2

- クラス宣言では，極力データを外部から直接参照できないようにし，メソッドを通してデータへのアクセスをするように設計する.



# ○ カプセル化#3




出典：立山秀利「Javaのオブジェクト指向がゼッタイにわかる本」秀和システム



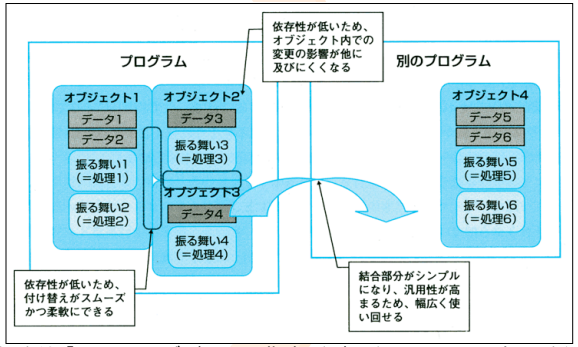
# ○ カプセル化#4

- 他のオブジェクトに「使っていいよ」と公開するフィールドとメソッドを最小限に抑える.
- 公開するなら，操作できる範囲を最小限に抑える.
- 他のオブジェクトはメソッドの中身をしらなくとも実行できる.



# ○ カプセル化の利点

- オブジェクト内部の仕様の変更が外のプログラム対して影響が減る. （依存性が低下）
- 再利用が容易になる. （汎用性が高まる）

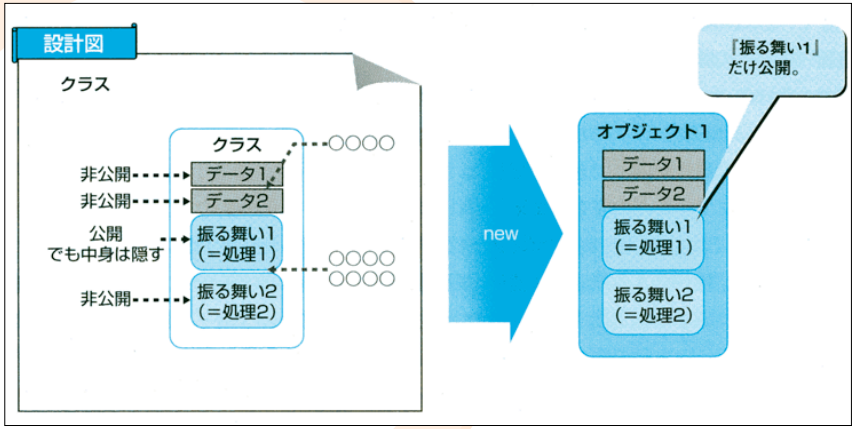



出典：立山秀利「Javaのオブジェクト指向がゼッタイにわかる本」秀和システム



# ○ カプセル化の指定

- カプセル化は，クラス単位で指定する.




出典：立山秀利「Javaのオブジェクト指向がゼッタイにわかる本」秀和システム

# ○ カプセル化の原則

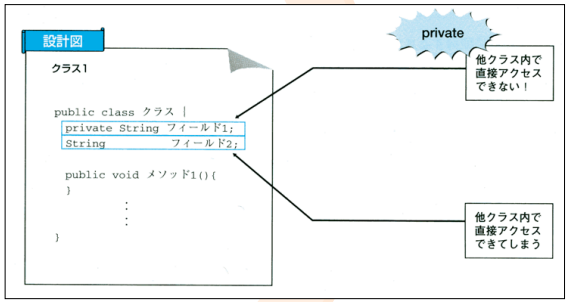
- フィールドは，原則，他のオブジェクトに直接アクセスさせない．
- 必要に応じて，間接的にフィールドにアクセスする手段を用意する．
- メソッドは必要に応じて，他のオブジェクトから隠す．



# ○ カプセル化の方法#1

- フィールドは，原則，他のオブジェクトに直接アクセスさせない．

```
private　型名 フィールド名;
```




出典：立山秀利「Javaのオブジェクト指向がゼッタイにわかる本」秀和システム



# ○ カプセル化の方法#2

- 必要に応じて，間接的にフィールドにアクセスする手段を用意する．
  - ○ フィールド値を取得するメソッド（Getterメソッド）

```
public 戻り値の型 メソッド名()
{　return フォールド名; }
```

```
public int getNum()
{ return num;　}
```

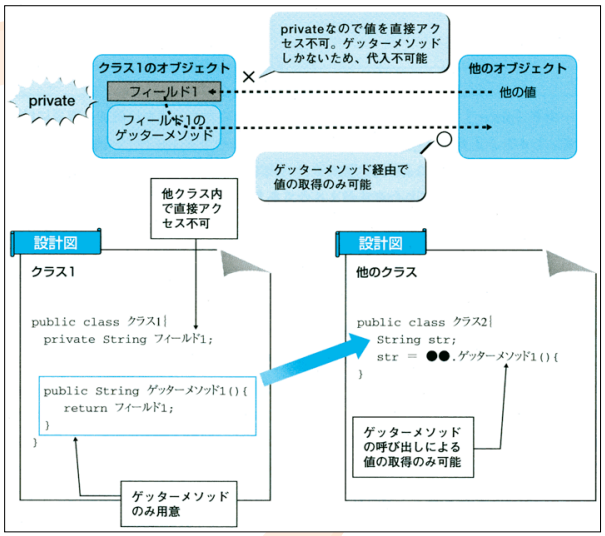
  - ○ フィールド値を設定するメソッド（Setterメソッド）

```
public 戻り値の型 メソッド名(引数)
{　フォールド名　= 引数;
}
```

```
public void
　setNum(int n){
num　= n; }
```

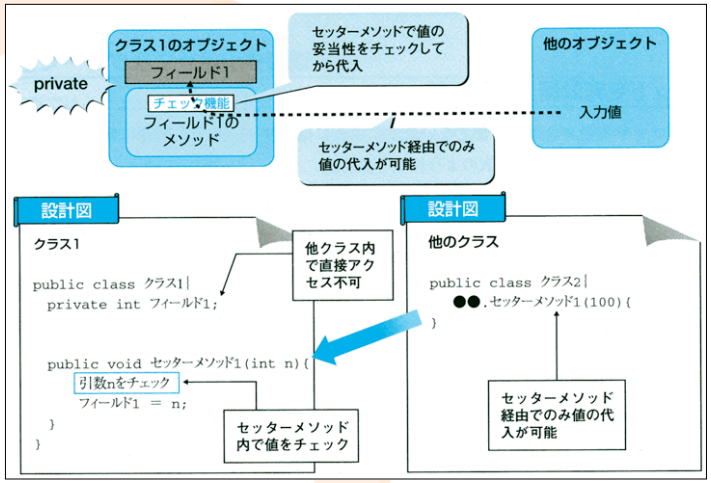


# ○ Getterメソッド




出典：立山秀利「Javaのオブジェクト指向がゼッタイにわかる本」秀和システム

# Setterメソッド




出典：立山秀利「Javaのオブジェクト指向がゼッタイにわかる本」秀和システム



# カプセル化の方法#3

- メソッドは必要に応じて，他のオブジェクトから隠す．

```
private　戻り値の型 メソッド名(引数1,引数2,...){
    処理文;
    ...
    ...
    return 戻り値;
}
```



# アクセス制限

- private：クラスの外からアクセスできない
- public：クラスの外からアクセスできる

| | |
|---|---|
| フィールド | privateを付加 |
| メソッド | publicを付加 |



# フィールドへのアクセス

- private：
  - 同一クラス内からのアクセスのみ可能
- protected：
  - 同一クラス内，そのサブクラスからのアクセスが可能

# 練習６

- 練習１で定義した「Member」のフィールド**name,no,age**を**private**で宣言した場合，SpcialMemberのオブジェクトを生成したときエラーがでます．SpecialMemberの定義を直せ．



# オブジェクトの参照



# クラス型変数#1

- 変数は「文字列や数値を保存するための箱」
  - これは，intやdoubleのような基本型変数の場合のこと．
- 一方，クラス型変数は，「オブジェクトの参照」を保存
  - 参照とは，オブジェクトがどこにあるかを表す場所情報のこと



# クラス型変数#2

- 変数の場合は異なるものをさしている．
- r1とr2は同じオブジェクトをさしている．
  - r1のフィールドの値を変更すると，rect2の対応するフィールドの値も変更される．

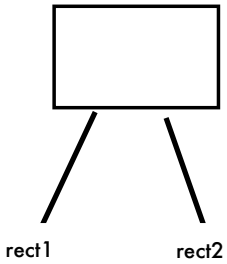
```
int i = 10;
int ii;
ii = i;
```

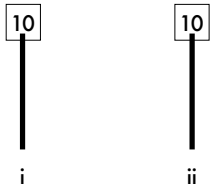
```
Rectangle r0 = new Rectangle();
Rectangle r1;
r1 = r0;
```

# 例： クラス型変数

```
Rectangle r1 = new Rectangle();
Rectangle r2;
r2 = r1;
```

```
int i = 10;
int ii;
ii = i;
```






# 例：クラス型変数

```
class Rectangle {
      int width;
      int height;
}

class K227_07_Sample1 {
  public static void main(String args[]) {
    Rectangle r1 = new Rectangle(); Rectangle r2;
    r2 = r1;

    r1.width = 10; r1.height = 20;
    System.out.println("r1: "+r1.width+":"+r1.height);
    System.out.println("r2: "+r2.width+":"+r2.height);

    r2.width = 20; r2.height = 50;
    System.out.println("r1: "+r1.width+":"+r1.height);
    System.out.println("r2: "+r2.width+":"+r2.height);
  }
}
```



# オブジェクトの配列

- 複数のオブジェクトをまとめて扱うためにつかう.
  - 配列を準備
  - オブジェクトを作成し，その配列の要素に代入

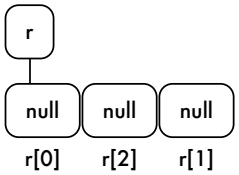
```
int test[]= new int[5];
test[0]=80;
test[1]=90;
...
```

```
Rectangle r[]=new
Rectangle[3]
r[0]= new Rectangle();
r[1]= new Rectangle();
...
```

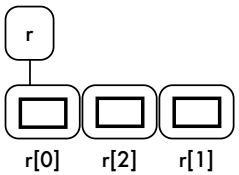


# 例

```
Rectangle r[]=new Rectangle[3]
```




```
Rectangle r[]=new Rectangle[3]
for(int i=0; i < r.length; i++) {
    r[i] = new Rectangle();
}
```



# 継承のときの参照

```
class Car{
      // Carのクラス定義

}
class Person{
      // Personのクラス定義

}
class Student extends Person{
      // Studentのクラス定義

}

//
Person p = new Person();Student s = new Student();
Car c = new Car();

Person p = new Car();Car c = new Person(); //NG
Person P = new Student(); // OK
```



# 例

```
class Person{
      // Personのクラス定義

}
class Student extends Person{
      // Studentのクラス定義

}

//
Person persons[] = new Person[2];

persons[0] = new Person();
persons[1] = new Student();
```