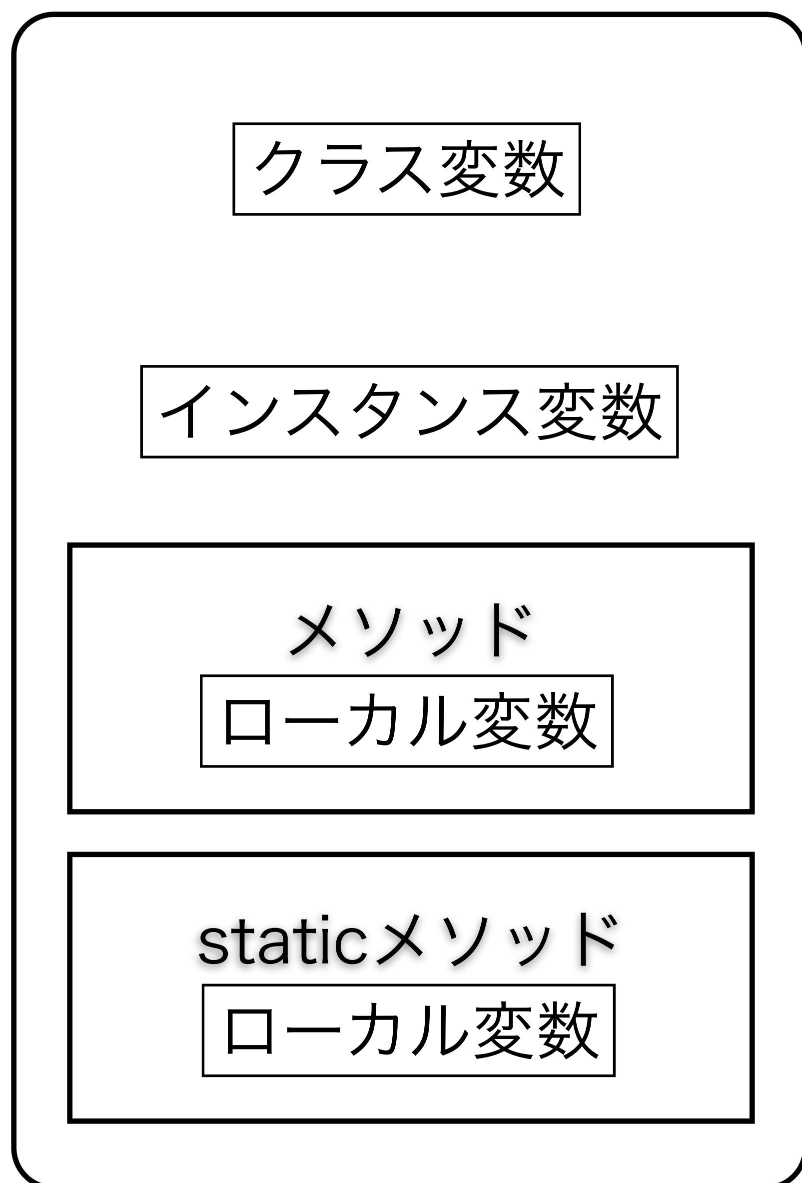
変数のスコープ



クラス変数
すべてのメソッドの外側でstaticで定義
staticメソッドとインスタンスのすべてのメソッドからアクセスできる

インスタンス変数
すべてのメソッドの外側で定義
インスタンス内の各メソッドからアクセスできる

ローカル変数
メソッドの内側で定義
そのメソッドのみ有効

```
public class Hensu {
static int stakazu = 10;
int inkazu = 40;
void keisan() {
    int keikazu = 30;
    System.out.println(keikazu);
    System.out.println(inkazu);
}
public static void main(String[] args) {
    // TODO 自動生成されたメソッド・スタブ
    Hensu hen = new Hensu();
    int mainkazu = 20;
    System.out.println(stakazu);
    System.out.println(mainkazu);
    hen.keisan();
    //System.out.print(keikazu);エラー
    //System.out.println(inkazu);エラー
}
}
```

カプセル化

setter, getter,を用いてアクセス制御（他とのやりとり）をおこなう。

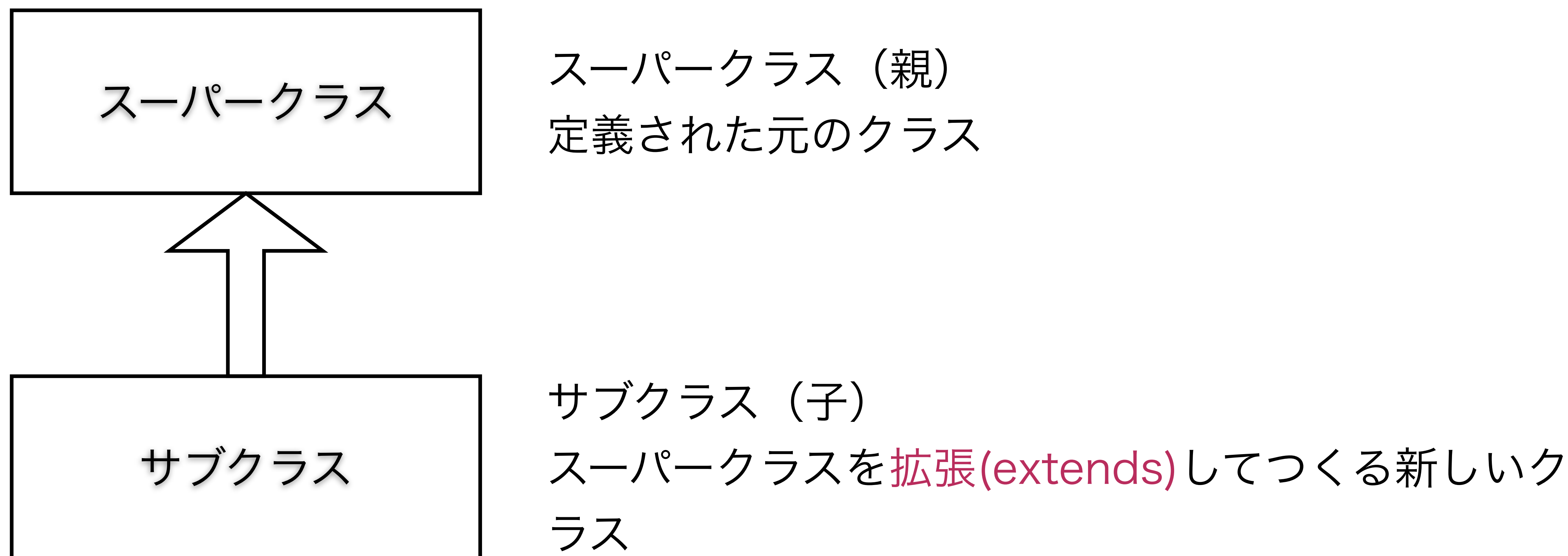
setter
変数に値を設定するメソッド

getter
変数から値を取り出すメソッド

データと処理を一体化する。独立性が高まる、ソフトウェアの部品化ができるなどのメリットがある。

```java
public class GetSet {
	int a;
	void setKazu(int a){
		this.a = a;
		System.out.println("セッターの"+a);
	}
	int getKazu(){
		return a;
	}
	public static void main(String[] args) {
		// TODO 自動生成されたメソッド・スタブ
		int b;
		GetSet ge = new GetSet();
		ge.setKazu(10);
		b=ge.getKazu();
		System.out.println("メインの"+b);
	}
}
```

継承（インヘリタンス）



スーパークラス（親）
定義された元のクラス

サブクラス（子）
スーパークラスを拡張(extends)してつくる新しいクラス

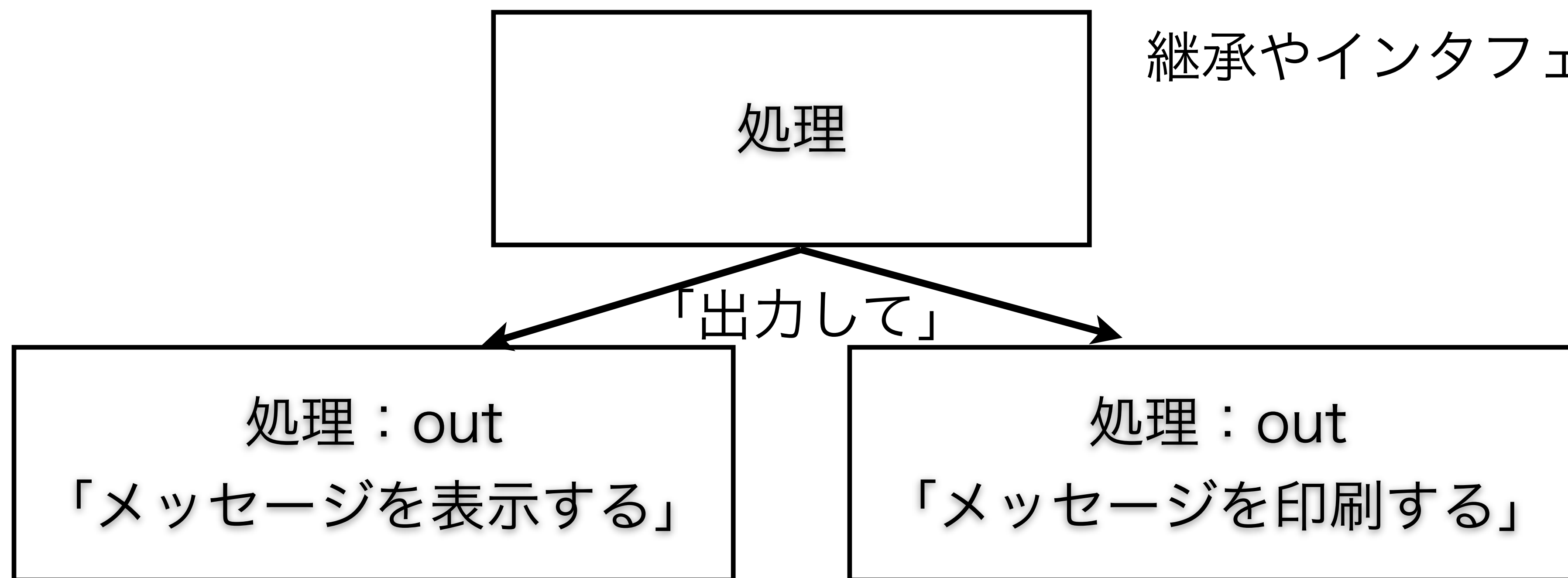
　サブクラスから生成されたインスタンス内部には、サブクラスで定義したデータと処理のみならず、スーパークラスで定義したデータと処理も含まれる。

```java
public class Keisyou1_1 {
  public static void main(String[] args) {
    // TODO 自動生成されたメソッド・スタブ
    Keisyou1_3 kei1_3 = new Keisyou1_3();
    kei1_3.hyouji();
    kei1_3.hyoujioya();
    System.out.println("Keisyou1_1からの表示です");
  }
}

public class Keisyou1_2 {
  public void hyoujioya(){
    System.out.println("Keisyou1_2からの表示です");
  }
}

public class Keisyou1_3 extends Keisyou1_2{
  public void hyouji(){
    System.out.println("Keisyou1_3からの表示です");
  }
}
```

多態性（ポリモーフィズム）



継承やインタフェースを用いておこなう。

各インスタンスに同じ名前のメソッドを呼び出しても、インスタンスによって動作が異なる。重複するメソッド名を利用でき、見通しがよくなる。

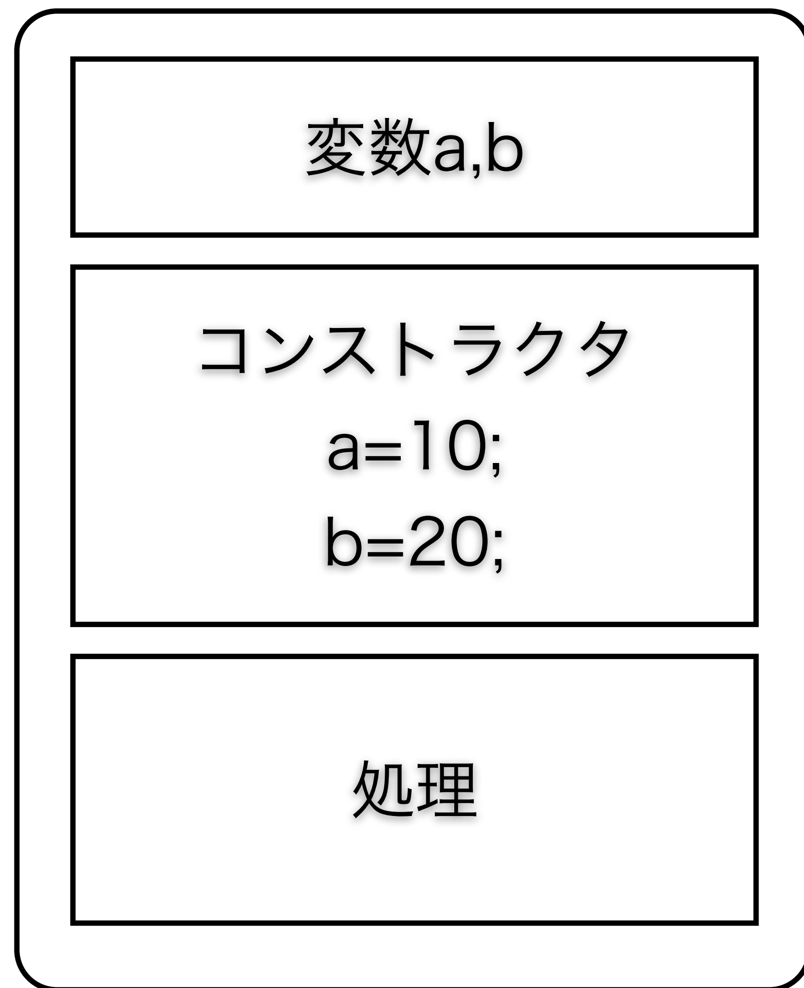
```java
    class Display {
    public void out(String message) {
        System.out.println("ディスプレイに「" + message + "」と表示します");
    }
}

class Printer {
    public void out(String message) {
        System.out.println("「" + message + "」という文字列を印刷します");
    }
}

public class PolymorPhism {
    public static void main(String[] args) {
        String message = "今日は雪です";
        Display display = new Display();
        Printer printer = new Printer();
        display.out(message);
        printer.out(message);
    }
}
```

コンストラクタ

クラス



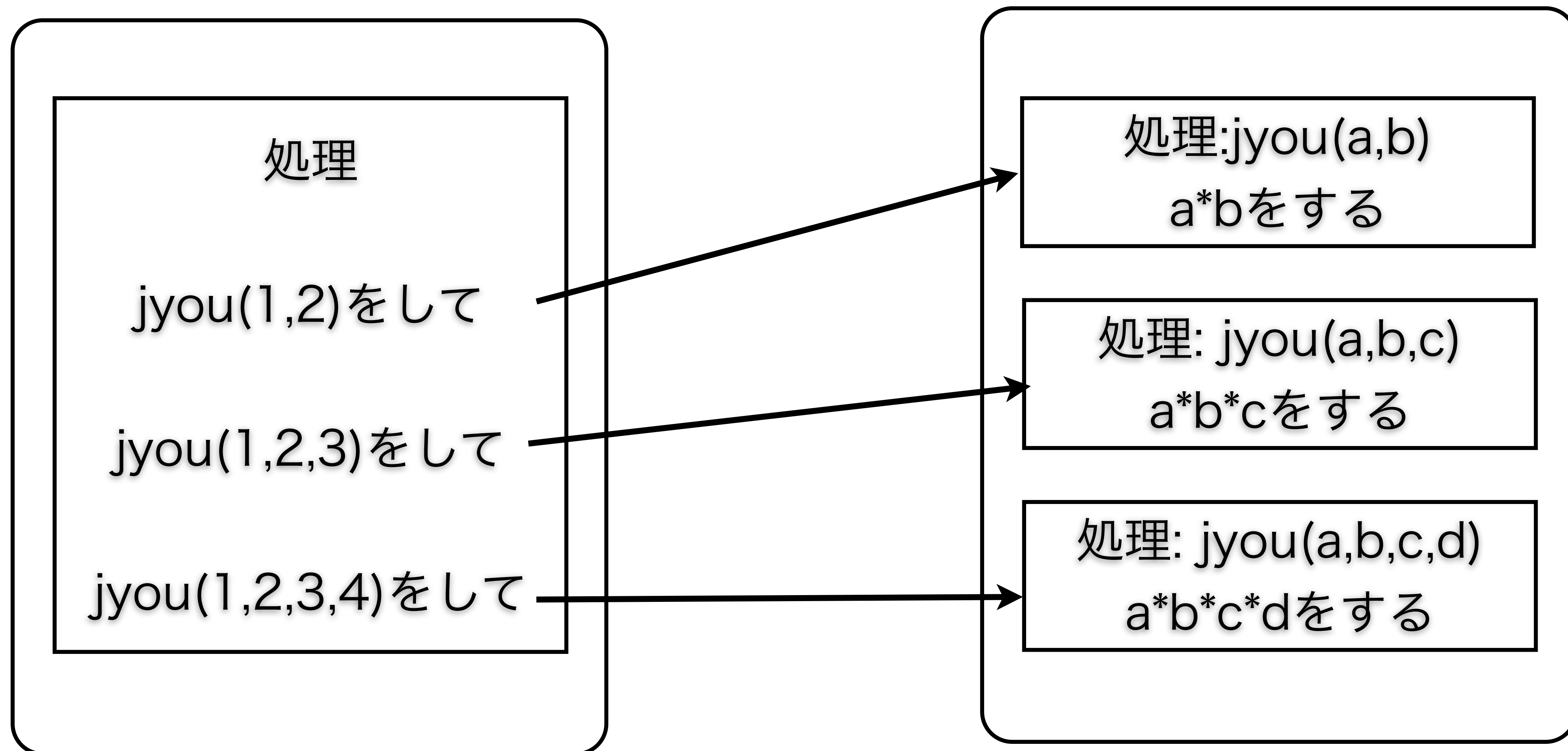
インスタンス変数

コンストラクタ
コンストラクタ名はクラス名と同じにする

インスタンス生成時にインスタンス変数を初期化できる。記述されていないときは自動的に追加される。

```java
public class Constracta {
 int a,b;
 Constracta() {
   a = 10;
   b = 20;
 }
 public void hyouji(){
   System.out.println("a+bは"+(a+b));
 }
  public static void main(String[] args) {
   // TODO 自動生成されたメソッド・スタブ
   Constracta cons = new Constracta();
   cons.hyouji();
 }
}
```

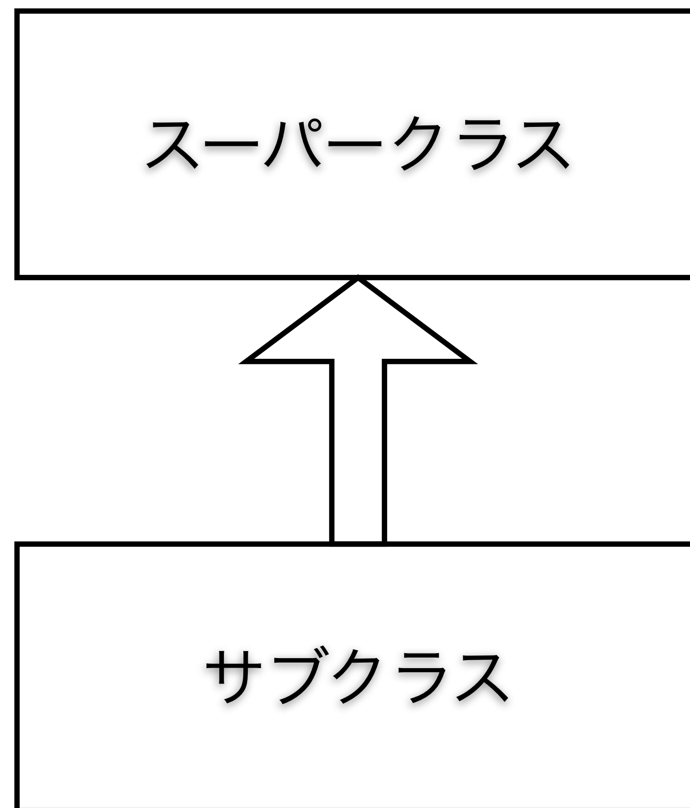
オーバーロード



引数の数や型が異なれば、同じ名前のメソッドを定義できる。

```java
public class OverLoad1_1 {
 public static void main(String[] args) {
   OverLoad1_2 ove1_2 = new OverLoad1_2();
   int kekka;
   kekka = ove1_2.jyou(1,2);                    //オーバーロード
   System.out.println("２つのかけ算"+kekka);
   kekka = ove1_2.jyou(1,2,3);
   System.out.println("３つのかけ算"+kekka);
   kekka = ove1_2.jyou(1,2,3,4);
   System.out.println("４つのかけ算"+kekka);
 }
}
public class OverLoad1_2 {
 int kotae;
 public int jyou(int s1,int s2){
   return kotae = s1 * s2;
 }
 public int jyou(int s1,int s2,int s3){
   return kotae = s1 * s2 * s3;
 }
 public int jyou(int s1,int s2,int s3,int s4){
   return kotae = s1 * s2 * s3 * s4;
 }
}
```

オーバーライド

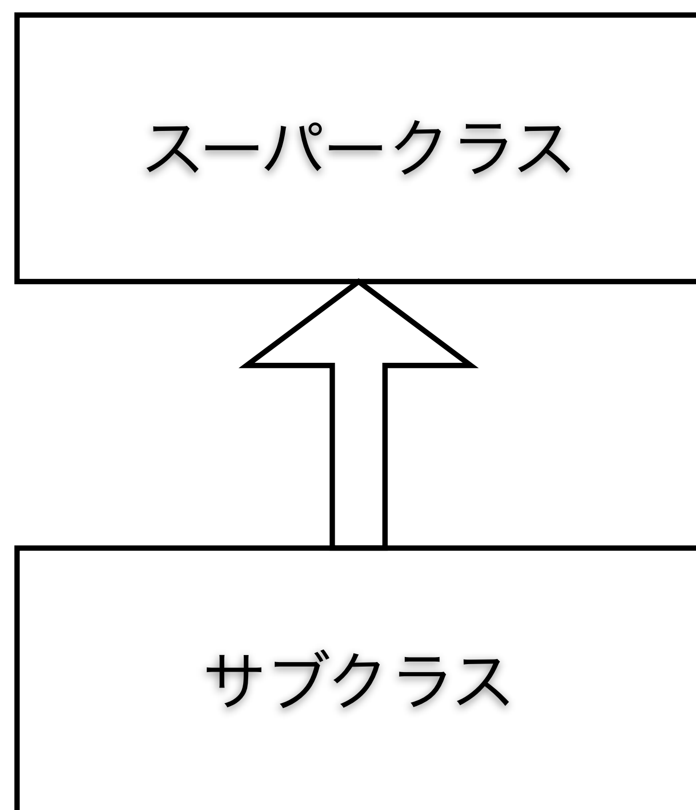


スーパークラスのhello()がサブクラスのhello()に上書きされた

スーパークラスが定義したものサブクラスで再定義する。

```java
public class OverRide{
    public static void main(String[] args) {
        OverRidesub ovrsub = new OverRidesub();
        ovrsub.hello();
        ovrsub.night();
    }
}
class OverRidesuper{
    void hello(){
        System.out.println("おはよう");
    }
    void night(){
        System.out.println("こんばんは");
    }
}
class OverRidesub extends OverRidesuper{
    void hello(){ //再定義
        System.out.println("おはようございます");
        super.hello();
       //hello(); これを書くと「おはようございます」と「おはよう」の無限ループ
    }
}
```

抽象クラス



抽象クラス

クラス名
abstract classで定義

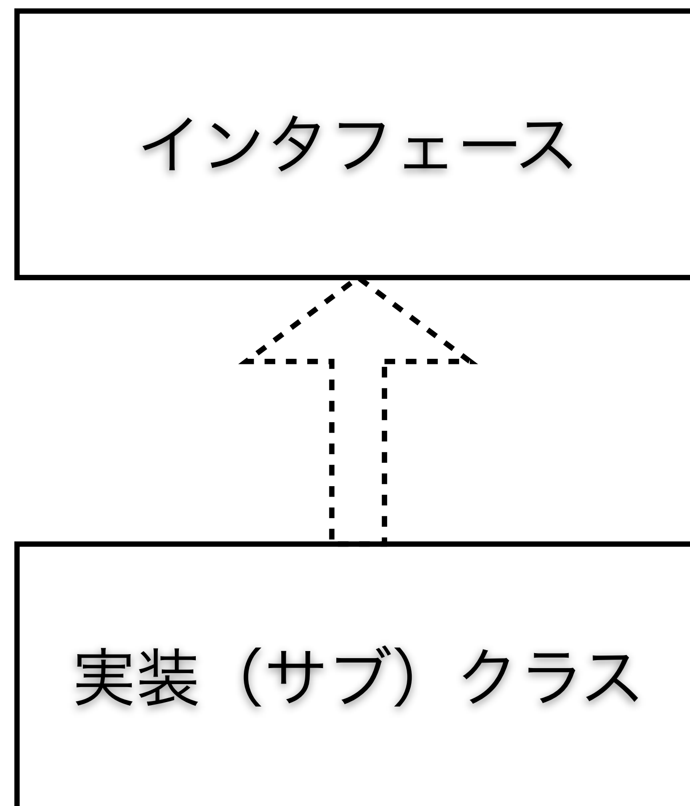
メソッド名
abstractで定義

中身のない抽象クラスのhyouji(kotae)がサブクラスのhyouji(kotae)に上書きされた

具体的な処理内容がない（中身が未定でコーディングされていない）抽象メソッドを持つクラス

```
abstract class Keisan {       //抽象クラス
 abstract void hyouji(int kotae);   //抽象メソッド
 int n,m;
 void setsuuti(int n,int m){ //セッター
   this.n=n;
   this.m=m;
 }
 int getsuutin(){            //ゲッター
   return n;
 }
 int getsuutim(){
   return m;
 }
}
class Tasizan extends Keisan{
 void hyouji(int kotae){
   System.out.println(kotae);
 }
}
class Abstract1 {
 public static void main(String[] args) {
 Tasizan tasi = new Tasizan();
 tasi.setsuuti(5,6);
 int kotae = tasi.getsuutin()+tasi.getsuutim();
 tasi.hyouji(kotae);
 }
}
```

インタフェース



インタフェース

インタフェース
interfaceで定義

implementsで定義

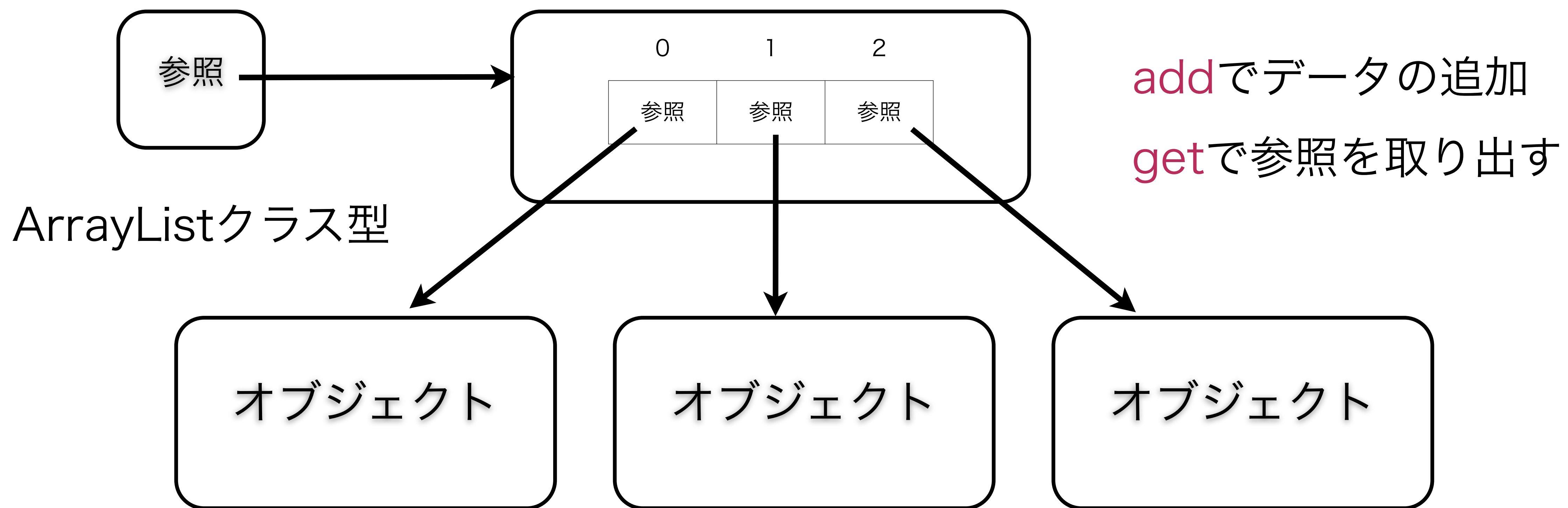
中身のないインタフェースのkei()が実装クラスのkei()に上書きされた

　インタフェースはインスタンスが持つ機能の利用方法だけを定義してある。実装（サブ）クラスにより実装する。インタフェースは複数同時に実装できる。

```java
interface asanoSHR{
   void asaSHR();
}
interface kaerinoSHR{
   void kaeriSHR();
}

class aisatsu implements asanoSHR,kaerinoSHR{
   public void asaSHR(){
      System.out.println("おはよう");
   }
   public void kaeriSHR(){
      System.out.println("さようなら");
   }
}
public class Interface {
   public static void main(String[] args) {
      // TODO 自動生成されたメソッド・スタブ
      aisatsu ai = new aisatsu();
      ai.asaSHR();
      ai.kaeriSHR();
   }
}
```

ArrayList



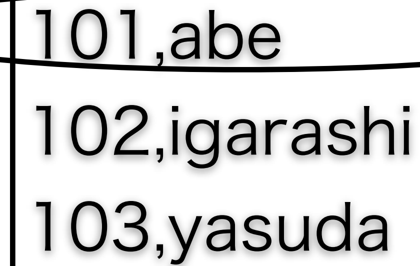
ArrayListは配列のようにオブジェクトを格納する。

```
Scanner sc = new Scanner(System.in);
String innamae;
String naiyou;
int n;
ArrayList<String> list = new ArrayList<String>();

public void nyuryoku() {
   System.out.println("名前を入力してください：終わりはend");
   innamae = sc.next();
   while(innamae.equals("end") != true){
      list.add(innamae);
      System.out.println("名前を入力してください：終わりはend");
      innamae = sc.next();
   }
}
public void hyouji(){
   for (n=0;n<list.size();n++){
      String naiyou = (String) list.get(n);
      System.out.println(naiyou);
   }

}
```

StringTokenizer



101,abe
102,igarashi
103,yasuda

0	1	2	3	4	5	6
1	0	1	,	a	b	e

StringTokenizer(line,",")
,で区切り取り出す

code

101

name

abe

CSVファイルから１行読み込み、カンマ区切りで取り出す

```java
   import java.io.*;
import java.util.*;
public class Tokenizer {
 public static void main(String[] args) throws
                                NumberFormatException, IOException {
    // TODO 自動生成されたメソッド・スタブ
    File csv = new File("meibo.csv");
    BufferedReader br = new BufferedReader(new FileReader(csv));
    String line = "";
    int code;
    String name;
    while((line = br.readLine()) != null){
      StringTokenizer str = new StringTokenizer(line,",");
      code = Integer.parseInt(str.nextToken());
      name = str.nextToken();
      System.out.println(code+name);
    }
  }
}
```

データベースアクセス

処理	方法
ドライバ読み込み	class.forNameで読み込み
DBMS接続を開く	DriverManager.getConnectionで接続
SQL送信路を開く	createStatement()で開く
SQL文送信	executeQuery(select〜)で select文送信と結果受信
実行結果受信	
実行結果から値を取り出す	get〜で結果を取り出し
実行結果（表）を閉じる	closeで閉じる
SQL送信路を閉じる	closeで閉じる
DBMS接続を閉じる	closeで閉じる

```
  try {
      //JDBCドライバのロード
      Class.forName("com.mysql.jdbc.Driver");
      con = DriverManager.getConnection(
           "jdbc:mysql://localhost:8889/jikken","root","root");
      //データベースに接続
      stmt = con.createStatement();
      rs = stmt.executeQuery("select * from meibo");
      while(rs.next() == true){
         String gBan = rs.getString("ban");
         String gNam = rs.getString("name");
         System.out.println(gBan+gNam);
                                 (途中省略)

try{
      if(rs != null){
         rs.close();
      }
      if(stmt != null){
         stmt.close();
      }
      if(con != null){
         con.close();
      }
    }
```