オブジェクト指向プログラム Java I

オブジェクト指向

- Javaプログラムは１つ以上のクラスで構成される
- Javaプログラムはすべてクラス
- クラスの構成
  - フィールド（データ、属性）
  - メソッド（コード、動作、機能、処理）
- クラスはオブジェクトを作成する雛形
- オブジェクト指向とは
  - ものごとを表現するすべてのものをオブジェクト単位で考えること
  - オブジェクトには、属性（データ）と機能（処理）が含まれる
- 例えば、自動車をオブジェクトと考えると
  - パーツという要素・データから構成される自動車は、個々の要素を変更し、バス、トラック、乗用車となる
  - 動かす、止まるといった機能がメソッドとなる
  - 基となるクラスは自動車の設計図
  - 基となる抽象的なクラスから、実態となるオブジェクトを作成する

オブジェクト指向プログラム

■>クラスとオブジェクト

```
class CircleC {                          // クラス
    double pai = 3.14;                   // データ
    double circle(int r) {               // メソッド
        return pai * r * r;
    }
}
class Demo {                             // クラス
    public static void main(String args[]) {   // メソッド
        double area;;
        CircleC obj;                     // オブジェクト
        obj  = new CircleC();
        area = obj.circle(10);
        System.out.println("area = " + area);
    }
}
```

※実行結果

```
area = 314.0
```

・クラスCircleC内のcircleメソッドを利用するために、クラスCircleCのオブジェクトを生成しなければならない
・オブジェクトの生成をインスタンス化といい、次のように行う

```
CircleC obj;         //クラスCircleCのオブジェクト変数objを宣言
obj  = new CircleC(); //クラスCircleCのオブジェクトを生成
```

・１行で記述すると

```
CircleC obj = new CircleC();
```

・クラス内のメソッドを使うためには、そのクラスで定義したオブジェクトのどのメソッドであるのかピリオド(.)を付けて指定する

```
area = obj.circle(10);
       |   |    |
       |   |    引数
       |   メソッド
       オブジェクト
```

クラス

- クラスは、データとメソッド(動作・機能)を定義したもので、オブジェクトの雛形となるもの

```
[修飾子] class クラス名 [extends スーパークラス名]
                        [implements インタフェース名] {
  データ定義部
  メソッド定義部
}
```

- 修飾子
  - public：パッケージ外からもこのクラスを参照できるクラス
  - abstract：実装されていない抽象クラス
  - final：スーパークラスとしてサブクラスを作れないクラス

メソッド

- メソッドは動作や処理内容（機能）を定義したもの

```
[修飾子] メソッドの型 メソッド名([引数]) {
  処理内容
}
```

- 例えば、２つの値ｘ，ｙの合計を計算して表示するメソッドは次のようになる

```
void add (int x, int y) {
  int z;
  z = x + y;
  System.out.println("x + y = " + z);
}
```

- void>は、このメソッド自身が値を返さないものであるということ
- メソッドが値を返す場合は、メソッドの型と値を返す命令を記述する

```
int add (int x, int y) {  // メソッドの型がint
  int z;
  z = x + y;
  return z;           // 値 z を返す命令 return
}
```

オブジェクト

- オブジェクトの生成と初期化は次のように行う

```
クラス名 オブジェクト変数 = new クラス名();
```

- 例えば、オブジェクトを生成して、表示するプログラムは次のようになる

```
class ObjSample {
  public static void main(String args[]) {
    Display obj = new Display();   // クラスDisplayのオブジェクトobj生成
    obj.Disp("Hello Java");        // オブジェクトobjのメソッドDispを呼ぶ
  }
}
class Display {                    // クラスDisplayの定義
  void Disp(String message) {      // メソッドDispの定義
    System.out.println(message);
  }
}
```

・プログラム内にクラスを複数定義することは可能である。コンパイルすることによって、プログラム内の各クラス毎に～.classファイルが作成される。ここでは、ObjSample.classとDisplay.classが作成される。実行する場合は、main()が含まれるObjSample.classを指定する。

・四則演算を各計算毎にクラス化して計算プログラム例

```
public class Comp4 {                    // クラスComp4を定義
  int add(int x, int y) {               // メソッドadd(int x,int y)を定義
    return x + y;                       // x + y を返す
  }
  int substract(int x, int y) {         // メソッドsubstractを定義
    return x - y;                       // x - y を返す
  }
  int multiply(int x, int y) {          // メソッドmultiplyを定義
    return x * y;                       // x * y を返す
  }
  int divide(int x, int y) {            // メソッドdivideを定義
    return x / y;                       // x / y を返す
  }
}

class Keisan {
  public static void main(String args[]) {
    int a, b, wa, sa, seki, sho;
    Comp4 enzan = new Comp4();          // クラスComp4のオブジェクトenzan作成
    a = 20;
    b = 10;
    wa   = enzan.add(a, b);             // オブジェクトenzanのadd(a,b)を実行
    sa   = enzan.substract(a, b);       // 同上のsubstract(a,b)を実行
    seki = enzan.multiply(a, b);        // 同上のmultiplu(a,b)を実行
    sho  = enzan.divide(a, b);          // 同上のdivide(a,b)を実行
    System.out.println(a + " + " + b + " = " + wa);
    System.out.println(a + " - " + b + " = " + sa);
    System.out.println(a + " * " + b + " = " + seki);
    System.out.println(a + " / " + b + " = " + sho);
  }
}
```

※実行結果

```
20 + 10 = 30
20 - 10 = 10
20 * 10 = 200
20 / 10 = 2
```

ローカル変数とクラス変数

- **ローカル変数>=>メソッド内部で宣言した変数**
- **クラス変数>=>メソッドの外部で宣言した変数**
- **Ｃ言語の変数のスコープを参照せよ**

```
class Demo {
  int a;
```

```
  void Demo1() {
    int x;
    …
  }
  vod Demo2() {
    int x;
    …
  }
}
```

・変数>a>はクラス変数
・メソッドDemo１()とDemo2()内のxはローカル変数であるが、メモリ領域は別

・ １からｎまでを表示するプログラム例

```
classs Display {
  int n                                  // クラス変数
  void Disp() {
    int i;                               // ローカル変数
    for (i = 1; i <= n; i++)
      System.out.print(I + " ");
  }

class VarSample {
  public static void main(String args[]) {
    Display obj = new Display();              // クラスDisplayのオブジェクトobj生成
    obj.n = 10;                              // オブジェクトobjの変数nに10を代入
    obj.Disp();                              // オブジェクトobjのDisp()実行
  }
}
```

※実行結果
1 2 3 4 5 6 7 8 9 10

static変数

・同一クラスのオブジェクトを複数作成した場合、それぞれのオブジェクトで参照する変数の値は異なる

・異なるオブジェクト間の変数合計プログラム例

```
class Sum {
  int total;
  Sum() {                            // コンストラクタ(クラス名と同名）
    total = 0;                       // 0 クリア
  }
  void add(int x) {
    total += x;
  }
}

class SumSample {
  public static void main(String args[]) {
    Sum obj1 = new Sum();              // クラスSumのオブジェクトobj1生成
    Sum obj2 = new Sum();              // クラスSumのオブジェクトobj2生成
    obj1.add(10);
    obj1.add(20);
    obj2.add(30);
    obj2.add(40);
    System.out.println("obj1.total = " + obj1.total);
    System.out.println("obj2.total = " + obj2.total);
  }
}
```

※実行結果
obj1.total = 30

```
obj2.total = 70
```

・オブジェクトobj1の変数totalとオブジェクトobj2の変数totalは異なる。メモリ領域が異なる（変数のスコープ）

・staticを用いて値を共有するプログラム例

```
class Sum {
  static int total;                  // 値を共有させる
  Sum() {                            // コンストラクタ(クラス名と同名）
    total = 0;                       // 0 クリア
  }
  void add(int x) {
    total += x;
  }
}

class SumSample {
  public static void main(String args[]) {
    Sum obj1 = new Sum();            // クラスSumのオブジェクトobj1生成
    Sum obj2 = new Sum();            // クラスSumのオブジェクトobj2生成
    obj1.add(10);
    obj1.add(20);
    obj2.add(30);
    obj2.add(40);
    System.out.println("obj1.total = " + obj1.total);
    System.out.println("obj2.total = " + obj2.total);
  }
}
```

※実行結果

```
obj1.total = 100
obj2.total = 100
```

・合計の表示は、オブジェクト変数obj1かobj2のtotal、またはクラス変数Sumのtotalで表すことができる

```
System.out.println("sum = " + Sum.total);
```

コンストラクタ

・クラス名と同じ名前のメソッドをコンストラクタという。
・コンストラクタはオブジェクトの初期化処理に利用できる

・コンストラクタを用いた合計プログラム例

```
class Sum {
  int total;
  Sum() {                            // コンストラクタ 引数無し
    total = 0;
  }
  Sum(int x) {                       // コンストラクタ 引数 x
    total = x;
  }
  void add(int x) {
    total += x;
  }
}

class ConstrcSample {
  public static void main(String args[]) {
    Sum obj1 = new Sum();            // クラスSumのオブジェクトobj1生成
    obj1.add(10);
    obj1.add(20);
```

```
    System.out.println("sum1 = " + obj1.total);
    Sum obj2 = new Sum(100);          // クラスSumのオブジェクトobj2生成
    obj2.add(10);
    obj2.add(20);
    System.out.println("sum2 = " + obj2.total);
  }
}
```

※実行結果
sum1 = 30
sum2 = 130

・コンストラクタに引数を用いたり、オーバーロード（同じ名前で引数が異なる定義）することができるが、返し値をもつことはできない。

staticイニシャライズ

・staticイニシャライズは、クラスが呼ばれた（ロードされた）ときに自動的に実行される
・コンストラクタよりも早く処理したい場合や、一度だけ処理を行いたい場合などに用いる

・staticイニシャライズを行ったプログラム例

```
class Disp {
  static {                        // static イニシャライズ
    System.out.println("initialize");
  }
  Disp() {                        // コンストラクタ
    System.out.println("construct");
  }
}

class StaticSample {
  public static void main(String args[]) {
    Disp obj1 = new Disp();          // オブジェクト生成時にコンストラクタを処理
    Disp obj2 = new Disp();          // オブジェクト生成時にコンストラクタを処理
  }
}
```

※実行結果
initialize　※ Dispクラスのロード時に処理
construct　　※ Disp.obj1 = new Disp(); が実行されるときに処理
construct　　※ Disp.obj2 = new Disp(); が実行されるときに処理

メソッドのオーバーロード

・メソッドのオーバーロードとは、クラス内に同じ名前のメソッドを定義すること

・メソッドのオーバーロードを行ったプログラム例

```
class Display {
  void Disp() {                     // メソッド Disp 引数なし
    System.out.println("Nothing");
  }
  void Disp(int x) {                // メソッド Disp 引数 x
    System.out.println(x);
  }
  void Disp(int x, int y) {         // メソッド Disp 引数 x, y
    System.out.println(x + y);
  }
}

class OvldSample {
```

```
  public static void main(String args[]) {
    Display obj = new Display();    // クラスDisplayのオブジェクトobj生成
    obj.Disp();                     // オブジェクトobjのDisp()を実行
    obj.Disp(1);                    // オブジェクトobjのDisp(1)を実行
    obj.Disp(1, 2);                 // オブジェクトobjのDisp(1,2)を実行
  }
}
```

※実行結果

```
Noting
1
3
```

・メソッドのオーバーロードを行う場合は、引数の数や型が異なっていなければならない

アクセス制御

・変数を宣言するときの修飾子の中には、アクセス制御するものがある

- **public：全てのオブジェクトの全てのメソッドからアクセス可能**
- **private：定義されたクラス内のメソッドからだけアクセス可能**
- **protected：定義されたクラス内のメソッド、そのクラスから派生したサブクラス、同一パッケージ内のクラスからアクセス可能**

・privateを用いて、変数の参照を不可にするプログラム例

```
class keisan {
  atatic int tanka;
  private float rate;                  // private変数rateの宣言
  keisan() {                           // コンストラクタ
     tanka = 1000;
     rate  = 0.05f;
  }
  init keisan(int kazu) {
    int kingaku;
    kingaku = (int)(tanka * kazu * (1.0 + rate));
    return kingaku;
  }
}

class PrvSample {
  public static void main(String args[]) {
    keisan obj1 = new keisan();
    int kazu = 3;
    int kingaku = obj1.keisan(kazu);
    System.out.println("tanka   = " + obj1.tanka);
    System.out.println("kazu    = " + kazu);
    System.out.println("kingaku = " + kingaku);
//  System.out.println("rate = " + obj1.rate);  //private変数のため参照不可
  }
}
```

※実行結果

```
tanka   = 1000
kazu    = 3
kingaku = 3150
```

TANIGUCHI,Yuji

Last>modified:>2003/11/17(Mon)>12:11