プログラミングII

黒瀬　浩

kurose@neptune.kanazawa-it.ac.jp

OH: 講義の前後，Eメール問合せ，月3限21-405

居室　67・121

Javaプログラミングの注意

ファイル拡張子は　.java にする
クラス名はファイル名と同じにする
クラス名は大文字で始める
変数，メソッドは小文字で始める
クラス内にはmain()メソッドが必要 (importされる場合は異なる)
実行したクラスのmain()メソッドから動く
文の行末には；が必要

クラス定義，メインメソッド定義は決まり文句なので使い回す

エラーは
コンパイルエラー（文法的なもの）と
ランタイムエラー（実行時にこれ以上動作できなくなって強制終了される）
がある．エラーが出たらソースコードを修正してやり直す

Java復習

ファイル名は主クラス名と合わせる
ソースの拡張子は　.java
コンパイル　javac クラス名.java↵　エラーがなければ実行　java クラス名↵
クラスの中にデータ(フィールド)や関数(メソッド)を登録する

```
public class Hello { // クラス
    public static void main(String[] args){ // メソッド
        System.out.println("Hello");      // メソッドの中身
    }
}
```

変数は型宣言が必要　`int a;`
文末は ; が必要
主な型 `byte, short, int, long, double, float, boolean, char`
関数も型宣言が必要　値を戻さなければ `void`型
文字（シングルクォート）と文字列（ダブルクォート）は別物
演算は容量が大きい型に合わせられる　文字列+数値　は文字列の結合
型変換はキャストを使う　　`double a = (double)1;`

制御構造　if, switch case

```
if( 条件式 ){ 真の場合 }
if( 条件式 ){ 真の場合 } else { 偽の場合 }
if( 条件式1 ){ 式1が真の場合 }
else if( 条件式2 ){ 式2が真の場合 }
...
else if( 条件式n ){ 式nが真の場合 }
else { いずれも偽の場合 }
```

```
switch (式){
        case 値1:
                        式の結果が値1の時に実行する文
                        break;
        . . .
        default:
                        いずれでもないときに実行する文
}
```

制御構造　繰り返し

```
for (ループ実行前に行う文; 判定条件式;  ループに戻る時に行う文){
        ループ内で実行する文
}

while(判定条件){
        ループ内で行う文
}

do {
        ループ内で行う文
} while(判定条件);
```

continue　　ループの最初に戻る(continueの後ろを飛ばす)
break　　(一番内側の)ループを脱出する

簡単なJavaプログラム

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

クラスHello内に 戻り値なしmain()関数がある(定義されている)
main関数内に System.out.println()を呼んでいる

java Hello↵ を実行すると　Helloクラスのmain()を呼び出す
そのため，ファイル名とクラス名は合わせておく必要がある

コンパイル　　javac Hello.java↵
実行　　java Hello

メソッド(クラス内の関数 javaの関数は全てメソッド)

```
public class Ex1 {
    public static void countdown(){
        System.out.println("カウントを出します");
        for(int i = 5; i>=0; i--){   // iが5,4,3,2,1,0となる
            System.out.println(i);
        }
    }
    public static void main(String[] args){
        countdown();     // メソッド呼び出し
    }
}
```

このクラスには，coundtDown(), main()のメソッドがある

通常　クラス.メソッド（引数）で呼ぶが，自分のクラスのメソッドを呼ぶ場合，メソッド(引数) で呼べる（関数と同じ）

メソッドは戻り値の型を指定する（戻り値がない場合は void)

メソッドの引数と戻り値

関数と同様に引数を渡せる

引数は，型を意識する必要がある

```
public static void method1(int i){   定義 }
```

method1を呼び出すには整数型の定数，変数，式が必要

関数と同様に戻り値が返せる

戻り値は，型を意識する必要がある

```
public static int method2(){ 定義; return 整数値 }
```

method1の結果は整数型の変数で受ける必要がある

演習　以下のメソッドを作り動作を確認せよ

method1	整数型の値の2倍を返す
method2	整数型の値2つを指定すると合計を返す
method3	実数型の値2つを指定すると大きい方を返す
method4	文字列と数を指定すると文字列を繰り返して返す

前ページの解答例

```
public class MethodEx1 {
    public static int method1(int x){   // このメソッドは1行でかけますがブロックで↵した方が良いでしょう
        return x*2;
    }
    public static int method2(int x, int y){
        return x+y;
    }
    public static double method3(double x, double y){
        return x>y ? x : y;             // 3項演算子を使ったがif文でも可
    }
    public static String method4(String x, int n){
        String r = x;                   // この例はn<=0に対応していません
        for(int i=2; i<=n; i++){
            r += x;
        }
        return r;
    }
    public static void main(String[] args){
       int a1 = 3, a2 = 4;
       double b1 = 1.2, b2 = 2.1;
       String c1 = "abc";
       System.out.println( "method1("+a1+")="+method1(a1) );
       System.out.println( "method2("+a1+","+a2+")="+method2(a1,a2) );
       System.out.println( "method3("+b1+","+b2+")="+method3(b1,b2) );
       System.out.println( "method4("+c1+","+a2+")="+method4(c1,a2) );
   }
}
```

実行結果

```
method1(3)=6
method2(3,4)=7
method3(1.2,2.1)=2.1
method4(abc,4)=abcabc
```

method4で繰り返しが0回以下だったら空文字列を返すように変更せよ

出力がどうなるか確認せよ

出力がわかりにくいのでメッセージを出すように変更せよ
　　method4は文字列を返すので簡単

method1でメッセージを返そうとすると何が困るか

メソッドのオーバーロード(overload)

先の2倍する関数　method1に実数を渡したい
しかし　method1の戻り値はint型で double ではない!

例にもう一個method1を追加してみる

```
int    method1(int    x){ return x*2; }
double method1(double x){ return x*2; }  // 追加
```

mainにも呼び出し側を追加

```
System.out.println( "method1("+a1+")="+method1(a1) );
System.out.println( "method1("+b1+")="+method1(b1) );
```

結果

```
method1(3)=6
method1(1.2)=2.4
```

略
動いた！

処理を変えてみる

```
int    method1(int    x){ return x*2; }
double method1(double x){ return x*3; }  // 変更
```

結果

```
method1(3)=6
method1(1.2)=3.5999999999999996
```

1.2*3 が 3.6 にならないのは浮動小数点演算のため

同じmethod1を呼んでも動くメソッドが違う！

数値+数値 ⇒ 加算
文字列+文字列 ⇒ 連結　となるのは同じ演算子でも処理が異なる

それと同じことがメソッドでもできる　（メソッドのオーバーロード機能)

メソッドのオーバーロードができる条件

メソッド名が同じ
引数の数が異なるか
引数の型の並びが異なる

引数の型，数，並びが同じ
だと同じメソッドを複数登録できない（戻り値の型が違っても）

同じ処理をやるのに型が違うだけで処理を多重登録しなければならないのか
はい(今はそうです)

数値2つを加算

```
int    method2(   int x,    int y){ return x+y; }
double method2(double x, double y){ return x+y; }
```

byte, short, long, floatなど引数型が異なれば多重に登録する
精度に注意すればキャスト(型変換)で回避できるかもしれない

レポート4 べき乗　5点

Javaにはべき乗の演算子がない(pythonでは**)

べき乗を行うメソッドpow0()をつくれ

引数が1つなら　pow0(3) ⇨　10^3=1000　　(10^nを行う)

引数が2つなら　pow0(2,3) ⇨ 2^3=8　　(n^mを行う)

なお赤字の値はキーボードから入力する（P116参照)

引数が2つの場合の後に1つの場合のメソッドを呼び，それぞれの結果を表示する

べき乗は値(2とか10)をn回乗算すれば良い

for文で繰り返しを行える

初期値を0にしてしまうと何回乗算しても結果は0となるので注意

チャレンジ1　n^mにおいてm=0やm<0の場合にも対応せよ

チャレンジ2　実数と整数を入力し実数の整数乗を行うpow0()も考えよ

（参考：　実数と実数のべき乗はmath.pow()がある）