情報処理概論
(Basic Theory of Information Processing)

第１０回：メソッド

概要

- メソッドとは
- メソッドの宣言
- メソッドの応用

13　メソッド

13.1　メソッドの定義

メソッドの定義の基本(コンストラクタ以外)

戻り値の型 メソッド名() {
　文
　 ....
}

戻り値の型 メソッド名(引数の型 仮引数) {
　文
　 ....
}

戻り値の型 メソッド名(引数の型 仮引数, 引数の型 仮引数, ・・・, 引数の型 仮引数) {

　文
　 ....
}

- メソッド名は識別子.
- 引数の型には，基本データ型(int，double，charなど)，参照型(配列型，クラス型)を指定できる.
- 戻り値の型には，基本データ型，参照型，void(戻り値なし)を指定できる.
- メソッドの定義の中で，戻り値はreturn文を使って指定する.

```
return x;
```

- return文を実行すると，メソッドの処理を終了してメソッド呼び出したプログラムに戻る.
- 戻り値がない場合は,

```
return;
```

だけとなるが，省略することもできる。

- return文がない場合は，メソッドを定義するプログラムの最後の文を実行後に，メソッド呼び出したプログラムに戻る.
- 戻り値の型の前に修飾子(public，protected，private，static)が置かれることがある。

メソッド定義の例：素数かどうか判定するメソッド

- 名前：chkPrime
  　名前が意味するものがメソッドと明示するために，引数を省略してchkPrime()と書くことが多い。
- 引数：int型のprime：素数かどうかチェックする数
- 戻り値：boolean型：primeが素数ならばtrue

```
// 素数かどうか判定するメソッド
public static boolean chkPrime(int prime) {
  int divider, nDivided;
  nDivided = 0;
  for (divider = 1 ; divider <= prime ; divider = divider + 1) {
    if (prime % divider == 0) {
      nDivided = nDivided + 1;
    }
  }
  if (nDivided == 2) return true;
  return false;
}
```

chkPrime()を使った素数を1000個表示するプログラム

```
public class PrimeMethod {
  public static void main(String args[]){
    int prime;
    for (prime = 2 ; prime <= 1000 ; prime = prime + 1) {
      if (chkPrime(prime)) {
         System.out.println(prime + "は, 素数です。");
      }
    }
    prime = prime + 2;
  }

  // 素数かどうか判定するメソッド
  public static boolean chkPrime(int prime) {
    // chkPrime()の中身は省略
  }
}
```

13.2　値渡し，参照渡し

変数の型の種類

- **値型**：変数に，**値そのもの**が格納されている。
  基本データ型
- **参照型**：変数に，あらわすべき値を**格納している位置**が格納されている。
  配列型，クラス型

メソッド (関数) に引数の値を引き渡すときの方法

- **値渡し**：引数の値を渡す。(Java，C 言語，C++言語)
- **参照渡し**：引数の参照情報を渡す。(Fortran，C++言語)

　参照型変数の場合は，変数の値が参照情報であるため，値渡しであっても値への参照情報が渡される。

例：

```
class Ref {
  public static void main(String args[]) {
    int a, b[] = new int[10];
    a = 2; b[2] = 3;
    System.out.println("Before : a = " + a + " b[2] = " + b[2]);
    func(a, b);
    System.out.println("After  : a = " + a + " b[2] = " + b[2]);
  }
  static void func(int c, int d[]) {
    System.out.println("Before : c = " + c + " d[2] = " + d[2]);
    c = 5; d[2] = 8;
    System.out.println("After  : c = " + c + " d[2] = " + d[2]);
  }
}
```

結果：

```
Before : a = 2 b[2] = 3
Before : c = 2 d[2] = 3
After  : c = 5 d[2] = 8
After  : a = 2 b[2] = 8
```

13.3 プログラムを作ってみよう

- 2つのint型の引数をとり，その最大公約数を戻り値とするメソッド
  `int GCD(int l, int m)`。

- 2つのint型の引数をとり，その最小公倍数を戻り値とするメソッド
  `int LCM(int l, int m)`。
  このメソッドを作成するときに，GCD()を利用すること。
- 1つのdouble型の引数xと1つのint型の引数Nをとり，N次で近似したxの指数関数を戻り値とするメソッド`double expN(double x, int N)`。

- 1つのdouble型の引数xと1つのint型の引数Nをとり，N次で近似したxの正弦関数を戻り値とするメソッド`double sinN(double x, int N)`。

- 1つのint型配列の引数dataをとり，それを選択ソートによって昇順に並び替える，戻り値なしのメソッド`void selectionSort(int[] data)`。

注 現在はオブジェクトを作成せずにメソッドを使っているので，修飾子staticを付ける。

13.4 再帰プログラミング

- メソッドからメソッドを呼び出し，その呼ばれたメソッドからメソッドを呼び出していく連鎖の中で，連鎖の上の方にもあるメソッドが，連鎖の下の方のメソッドから呼び出されること。
- ローカル変数は，同じメソッドでも，呼び出されるごとに新たに領域が確保される.
- 参照型の変数から参照されるオブジェクや配列に関しては，新しい領域が必要な場合は，newを使って領域を確保しなければいけない.
- 自己再帰：メソッドが直接自分自身のメソッドが呼び出すこと。

```
class Recursive {
  static public void main(String[] args) {
    System.out.println("1+2+...+10 = " + sumUp(10));
  }
  static int sumUp(int num) {
    if (num == 0) return 0;
    return sumUp(num - 1) + num;
  }
}
```

何をするプログラムか考えてみよう.

```
class Recursive2 {
  static public void main(String[] args) {
    System.out.println(pow(3));
  }
  static int pow(int num) {
    if (num == 0) return 1;
    return pow(num - 1) + pow(num - 1);
  }
}
```

```
class Recursive3 {
  static public void main(String[] args) {
    System.out.println(mul(4, 3));
  }
  static int mul(int x, int y) {
    if (x == 0) return 0;
    return mul(x - 1, y) + y;
  }
}
```

13.5　グループワーク

- **アクティブラーニング** (講義ばかりでは学生が勉強しないから？)
- グループでプログラムを作成する。何を作るかはグループの自由である。
- グループ分けは教員側で指定する。
- 最終日に実演と共にプレゼンを行う (プレゼン資料は提出する)。
- グループワークの評価は，試験とレポートの点に乗算 $(0.9 \sim 1.1)$ することによって。個人の成績の反映する。

13.6　今日のまとめ

- メソッド
- メソッドの応用
- 再帰プログラミング
- グループワーク