情報処理技法
(Javaプログラミング)1

第9回
同じ種類の情報をたくさん扱いたい場合は?

人間科学科コミュニケーション専攻
白銀 純子

第9回の内容

- 同じ種類の情報をたくさん扱う方法

前回の復習問題の解答

- 下記のプログラムで、どこでコンパイルエラーが出るか、またその理由は何かを答えなさい。
  - 変数iには、if文の前の処理(「...略...」の部分)で、何らかの値が入っていることとする。
  - 変数numとresultは、if文の前の処理では、何も値が入らないこととする。

```
int i, num, result;
...略...
if (i >= 0) {
  num = 0;
} else if (i < 0) {
  num = 10;
}
result = num * 100;
```

if文で、「else」ではなく「else if」としているため
➢「i>=0」でなく「i<0」でもない
場合が存在すると解釈される

繰り返し文(復習)

繰り返し(ループ)って?(p. 177)

- ある条件を満たしている間、同じ処理を何回もする場合に使う文
  - for文
  - while文
  - do～while文

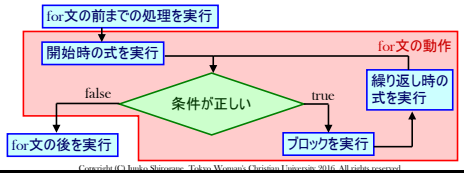
例えば処理を実行した回数を数え、指定された回数に達しているかどうか、など

for文(p. 178)

- 主に、処理を繰り返す回数が決まっている場合に利用
  - 配列(この後で)を扱う場合に使うことが多い

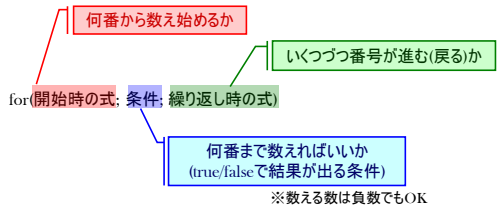
for文の書き方と動作(p. 178)

```
for(開始時の式; 条件; 繰り返し時の式) {
    文1;
    文2;
    .....
}
```
ブロック



for文の一般的な使い方(p. 178)

- 1, 2, 3, ...や、50, 49, 48, ...のように、何かの数を数える場合に利用
  - 繰り返す回数が決まっている場合に利用



for文の使い方例(1)(p. 179)

- 0から49まで順に数えたい場合

```
for(i = 0; i < 50; i = i + 1)
```

- 0から100までの数のうち、偶数のみを扱う場合

```
for(i = 0; i < 100; i = i + 2)
```

- 49から0まで順に数えたい場合

```
for(i = 49; i > 0; i = i - 1)
```

※「i」はint型の変数
(i, jは、ループ文で数を数えるための変数としてよく使われる)

for文の使い方例(2)(p. 179)

- 1から50までの数を足し合わせた数を求める

```
result = 0;
for(i = 1; i <= 50; i = i + 1) {
    result = result + i;
}
```

「代入の不思議」の復習:「result=result+i」は、この式の前までの「result」の値に「i」を足し、「result」の値を新しくする

for文に入る前: resultの値は0になる
iの値が1の時: resultの値は0 + 1になる
iの値が2の時: resultの値は0 + 1 + 2になる
iの値が3の時: resultの値は0 + 1 + 2 + 3になる
.....
iの値が50の時: resultの値は0 + 1 + 2 + ... + 50になる
iの値が51の時: ループを終わる

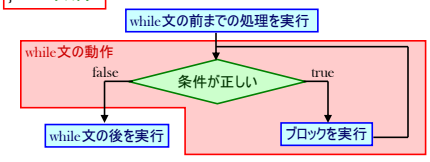
while文(p. 183)

- 主に、処理を繰り返す回数が決まっていない場合に利用

while文の書き方と動作(p. 183)

```
while(条件) {
    文1;
    文2;
    .....
}
```
ブロック

条件の書き方はif文とまったく同じ(true/falseで結果が出る条件)



while文の使い方例(1)(p. 184)

- 1から50までの数を足し合わせた数を求める

```
i = 1;
result = 0;
while (i <= 50) {
  result = result + i;
  i = i + 1;
}
```

while文に入る前: resultの値は0になる
iの値が1の時: resultの値は0 + 1になる
iの値が2の時: resultの値は0 + 1 + 2になる
iの値が3の時: resultの値は0 + 1 + 2 + 3になる
.....
iの値が50の時: resultの値は0 + 1 + 2 + ... + 50になる
iの値が51の時: ループを終わる

※この処理は、for文の例をwhile文に直したもの

同じ種類の情報をたくさん扱う

同じ種類の情報って?(p. 196)

- 例えば、高校の生徒の成績

1クラス50人の成績(英, 数, 国, 理, 社の得点)を管理するプログラム

- 英語: 約50人分
- 数学: 約50人分
- 国語: 約50人分
- 理科: 約50人分
- 社会: 約50人分

の得点が存在する

それぞれの教科について、約50人分ずつの変数が必要

そんなにたくさん変数を宣言する?

```
int english1, english2, english3, ..., english50;
int math1, math2, math3, ..., math50;
int language1, language2, language3, ..., language50;
int science1, science2, science3, ..., science50;
int society1, society2, society3, ..., society50;
```

面倒&ややこしい

扱う情報はたくさんあってもプログラムでは...

- 各生徒の点数を比較して順位付けをする
- 各生徒の合計点と平均点を計算する
- 赤点かどうか判定する

それぞれの情報の扱い方(処理のしかた)は同じ

「配列」または「ArrayList」を使う

配列

「配列」って?(p. 196)

- データ型が同じで、処理方法も同じ変数をたくさん扱うときに利用する変数
  - たくさんの変数を1度にまとめて宣言する方法

➡ 変数に番号をつけて扱う

例えば...英語の成績を表す変数を「english」
出席番号1番の生徒の成績: english[1]
出席番号2番の生徒の成績: english[2]
....
出席番号50番の生徒の成績: english[50]

➡ 宣言する変数は「english」だけ

配列の宣言のしかた(p. 197)

1. 変数の名前(配列変数と呼ぶ)を決める
2. 扱う値の個数を決める
3. 配列を宣言する

書き方その1

```
データ型 変数名[];
変数名 = new データ型[個数];
```

「[ ]」で、配列を宣言するという意味

扱う値の個数を決める

書き方その2

```
データ型[ ] 変数名 = new データ型[個数];
```

書き方その3

```
データ型 変数名[ ] = new データ型[個数];
```

※書き方その1, 2, 3のどれで宣言してもかまわない

配列の宣言のしかた(例)(p. 197)

英語の成績を表す変数(50個)

```
int english[];
english = new int[50];
```

または

```
int[] english = new int[50];
```

50個の変数を宣言

生徒の名前を表す変数(100個)

```
String name[];
name = new String[100];
```

または

```
String[] name = new String[100];
```

100個の変数を宣言

配列の使い方(p. 198)

「[ ]」の中に値の番号を書き、宣言した変数の後ろにつけて使う

- 番号を付けた1つ1つの変数を、配列の「要素」と呼ぶ
  - 配列の要素が、通常の変数に相当

「添え字」と呼ぶ
※番号は0から数える

例えば...英語の成績を表す配列変数「english」の要素に点数を代入

```
出席番号1番の生徒の成績: english[0] = 80;
出席番号2番の生徒の成績: english[1] = 50;
.... 略 ....
出席番号50番の生徒の成績: english[49] = 75;
```


添え字(p. 198)

0から数えて、(宣言した数-1)まで使うことができる

つまり、50個宣言をしたら、
「[ ]」の中の番号は0～49
50個

➡ -1, -2, ...や、50, 51, ...という数は使えない

配列の初期化(1)(p. 199)

最初に宣言をした後、要素に1つ1つ値を代入していく方法

最初に変数名と個数を宣言し、1つ1つ値を代入

```
int english[] = new int[50];
english[0] = 80;
english[1] = 75;
....
english[49] = 50
```

※初期化: 変数を宣言した後、初めて値を代入すること

配列の初期化(2)(p. 199)

宣言と同時に要素に値を代入する方法

```
int english[] = {80, 75, ..., 50};
```

「,」(コンマ)で値を区切って、左側の値から順に0番、1番、2番...に代入される
※この場合、配列の個数は、値の個数になる(配列の個数の宣言は不要)

※初期化: 変数を宣言した後、初めて値を代入すること

配列の使い方例(1)(p . 201)

- 複数の配列で、要素の添え字が同じものを、1まとまりとして扱う

出席番号1番の生徒の得点
– 添え字が0番の要素

language[0]
math[0] english[0]
science[0] society[0]

出席番号2番の生徒の得点
– 添え字が1番の要素

language[1]
math[1] english[1]
science[1] society[1]

配列の使い方例(2)(p . 200)

- 配列変数に配列変数を代入する
  - 個数の違うものに代入すると、代入される側の配列の個数が、代入する側の個数に変わる

例えば...

```
int abc[ ] = new int[10];
int def[ ] = new int[5];
```

abc[0]～abc[9]の値をdefに代入する

```
def = abc;
```

この場合、defの個数は10個になる

def[0]～def[4]の値をabcに代入する

```
abc = def;
```

この場合、abcの個数は5個になる

配列の使い方例(3)(p . 201)

- 配列の長さ(値が代入されている変数の個数)を求める

「*length*」という機能を使う
「配列変数.length」という書き方

例えば...配列変数「english」の長さを求める場合

```
int englishLength = english.length;
```

英語の試験を受けた人数:englishLength

ループ文との組み合わせでの利用

ループ文との組み合わせ

- 配列は、複数の同じデータ型のデータを扱うもの
  - 複数のデータに対して同じ処理を行う
- 配列は、添え字という番号をつけて管理
  - 添え字は、0, 1, 2, ...というように0番から順になっている

ループ文と組み合わせて、1つ1つの配列の要素を処理することが多い
- ループ文の処理として、i番目の要素に対する処理を定義しておく
  - ✓ iは、ループ文で数を数えるための変数
- ループ文で、添え字を0から順に(または最後から順に)数える

for文との組み合わせ

- 50人の生徒の英語の平均点を求める場合

```
sum = 0;
for(i = 0; i < 50; i = i + 1) {
  sum = sum + english[i];
}
average = sum / 50;
```

配列のi番目の要素に対する処理

iの値が0の時: sumの値は0 + english[0]になる
iの値が1の時: sumの値は0 + english[0] + english[1]になる
iの値が2の時: sumの値は0 + english[0] + english[1] + english[2]になる
.....
iの値が49の時: sumの値は0 + english[0] + english[1] + ... + english[49]になる
iの値が50の時: ループを終わって、次の処理(平均を計算)をする

配列におけるループ文の考え方(3)

Ex. 配列score(要素数: 100個)に入っている得点の平均を計算

```
int sum = score[0] + score[1] + score[2];
int average = sum / 3;
```

```
sum = score[0];

sum = score[0] + score[1];

sum = score[0] + score[1] + score[2];
```

足し算を1回分ずつ書くと...

配列におけるループ文の考え方(5)

Ex. 配列score(要素数: 100個)に入っている得点の平均を計算

```
int sum
(1) sum = score[0];
(2) sum = score[0] + score[1];
(3) sum = score[0] + score[1] + score[2];
int average = sum / 3;
```

- (2)の段階で変数sumに(1)の処理結果が入っている
- (3)の段階で変数sumに(2)の処理結果が入っている

↓

- (2)は「sum = sum + score[1]」と書ける
- (3)は「sum = sum + score[2]」と書ける

↓

「変数 = 変数 + 配列の要素」の形でループ文にすることが可能

配列におけるループ文の考え方(5)

Ex. 標準入力で数を100個入力し、その平均を計算するプログラム

```
int sum

sum = score[0];

sum = score[0] + score[1];

sum = score[0] + score[1] + score[2];

int average = sum / 3;
```

100回繰り返すループ文に書き直すと...

```
int num, sum, i;
String str;

for (i = 0; i < 100; i = i + 1) {
  sum = sum + score[i];
}
int average = sum / 100;
```


while文の使い方例(1)

50人の生徒の英語の平均点を求める場合

```
i = 0;
sum = 0;
while (i < 50) {
  sum = sum + english[i];
  i = i + 1;
}
average = sum / 50;
```

配列のi番目の要素に対する処理

iの値が0の時: sumの値は0 + english[0]になる
iの値が1の時: sumの値は0 + english[0] + english[1]になる
iの値が2の時: sumの値は0 + english[0] + english[1] + english[2]になる
.....
iの値が49の時: sumの値は0 + english[0] + english[1] + ... + english[49]になる
iの値が50の時: ループを終わって、次の処理(平均を計算)をする

※この処理は、for文の例をwhile文に直したもの

多次元配列

多次元配列(p . 203)

1人の生徒が持つ情報: 英語・数学・国語・理科・社会の得点

```
int english[] = new int[50];
int math[] = new int[50];
int language[] = new int[50];
int science[] = new science[50];
int society[] = new society[50];
```

のように1科目ずつ変数を用意する??

↓

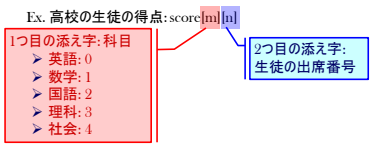
種類が多くなると扱いが面倒!!

科目は違っても、データの内容は「得点」
- データ型はすべて同じ(int型)
- 扱い方はすべて同じ(順位付けをする, 平均を計算する, etc.)

多次元配列

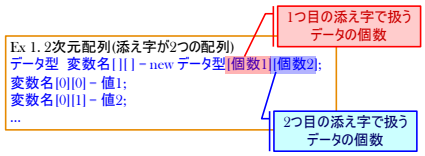
多次元配列の利用(p . 203)

- 多次元配列: 添え字を複数持つ配列
  - 1つ目の添え字でxx、2つ目の添え字でyy、3つ目…を意味する、というように、添え字の意味を決めておく



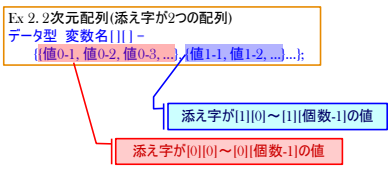
宣言と初期化(1)(p . 203)

- 配列変数の名前を決める
- 扱う要素の種類(添え字の種類)と値の個数を決める
- 配列を宣言と初期化をする



宣言と初期化(2)(p . 203)

- 配列変数の名前を決める
- 扱う要素の種類(添え字の種類)と値の個数を決める
- 配列を宣言と初期化をする



宣言と初期化(3)(p . 203)

- Ex. 5教科の得点を、50人分扱う場合
  - 1つ目の添え字が教科、2つ目の添え字が1人1人の生徒

```
int score[][] = new int[5][50];
score[0][0] = 80;
score[0][1] = 73;
…
```

- Ex. 10のクラスの生徒の英語の得点
  - 1つ目の添え字がクラス、2つ目の添え字が1人1人の生徒

```
int english[][] = new int[10][50];
english[0][0] = 95;
english[0][1] = 98;
…
```


多次元配列の参照(p . 205)

- 50人の生徒1人1人の5教科の得点の平均を計算

```
int score[][] = new int[5][50];
int average[] = new int[50];
int i, j, sum;
……
for(i = 0; i < 50; i = i + 1) {
    sum = 0;
    for (j = 0; j < 5; j = j + 1) {
        sum = sum + score[j][i];
    }
    average[i] = sum / 5;
}
```

1つ目の添え字(生徒を表す添え字)を数える

2つ目の添え字(教科0を表す添え字)を数える

エラー(配列)

実行時のエラー(1)(p . 208)

- Exception in thread "main"
  java.lang.ArrayIndexOutOfBoundsException
  - 配列を使っている場合に、プログラムを実行したときに起こることがあるエラー
  - 配列の添え字で使ってはならないものを使ったときのエラー
    - 配列は50個しか宣言していない(0～49番までしか使えないのに50番目の配列を使おうとした場合など
    - 配列の添え字にマイナスの数を使おうとした場合など

配列の添え字を何番まで使おうとしているか
よく確認しよう!

実行時のエラー(2)(p . 208)

```
int num[ ] = {1, 2, 3, 4, 5};
int i, sum = 0;
for (i = 0; i <= 5; i = i + 1) {
sum = sum + num[i];
}
```

配列は5個(添え字は0～4を利用可能)

for文は0～5の6回分繰り返し
(繰り返し条件が「<=」のため)

iの値が0の時: sumの値は0 + num[0]になる
iの値が1の時: sumの値は0 + num[0] + num[1]になる
iの値が2の時: sumの値は0 + num[0] + num[1] + num[2]になる
iの値が3の時: sumの値は0 + num[0] + num[1] + num[2] + num[3]になる
iの値が4の時: sumの値は0 + num[0] + num[1] + num[2] + num[3] + num[4]になる
iの値が5の時: sumの値は0 + num[0] + num[1] + num[2] + num[3] + num[4] + num[5]
を計算しようとする

配列の添え字は「4」まで使って良い
- 「5」という添え字は使ってはならない
- エラー

実行時のエラー(3)(p . 208)

- チェック方法
  - iの値が0のとき, iの値が1のとき, ...と、1つずつ数えていき、添え字がそれぞれのときでどうなっているかをチェック
    - 添え字が、利用可能な範囲を超えていないか? をチェック
  - for文の「繰り返すかどうか調べる式」のところが、「<=」で良いか、「<」が良いかをチェック

```
int num[ ] = {1, 2, 3, 4, 5};
int i, sum = 0;
for (i = 0; i <= 5; i = i + 1) {
sum = sum + num[i];
}
```

この例では、「<=」を「<」に直すと、エラーがなくなる
- ➢「<=」だと、添え字が5まで数えるため
- ➢「<」だと、添え字は4まで

ArrayList

「ArrayList」って?

- データ型が同じで、処理方法も同じ値をたくさん扱うときに利用する機能(クラス)

➡ 複数の値を入れる箱のようなもの

例えば...英語の成績

出席番号1番の生徒の成績
出席番号2番の生徒の成績
....
出席番号50番の生徒の成績

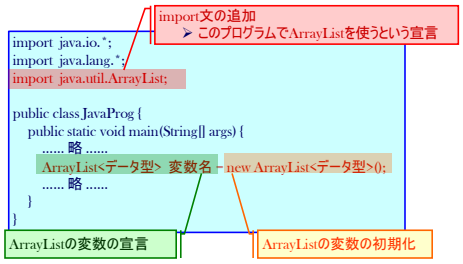
入れる

ArrayList

ArrayListの準備(1)

1. Javaファイルの冒頭に「import java.util.ArrayList;」を追加
2. ArrayList型の変数を宣言
   - 何のデータ型の値を登録するかもあわせて宣言
   - ArrayListに入れるデータは、すべて同じデータ型である必要
   - 配列のような、登録する値の個数の宣言は不要
3. 2.の変数を初期化

ArrayListの準備(2)



ArrayListの準備(3)

- 「<データ型>」の部分は、ArrayListに登録する値のデータ型
  - String型の値を登録する場合はそのままデータ型を書く
    ArrayList<String> 変数名 = new ArrayList<String>();
  - int・float・double型の値を登録する場合は、データ型の表記が異なる
    - int型の場合: Integer
    - float型の場合: Float
    - double型の場合: Double

※これまでに授業で出てきたデータ型以外のデータ型の値も登録可能

ArrayListの準備(4)

- String型のデータを登録する場合

```
ArrayList<String> 変数名 = new ArrayList<String>();
```

- int型のデータを登録する場合

```
ArrayList<Integer> 変数名 = new ArrayList<Integer>();
```

- float型のデータを登録する場合

```
ArrayList<Float> 変数名 = new ArrayList<Float>();
```

- double型のデータを登録する場合

```
ArrayList<Double> 変数名 = new ArrayList<Double>();
```


ArrayListに値を登録(1)

- 「ArrayListの変数名.add(登録する値)」で登録
  - String型の値の登録例

```
list.add(str);
```

```
list.add("abc");
```

  - int型の値の登録例

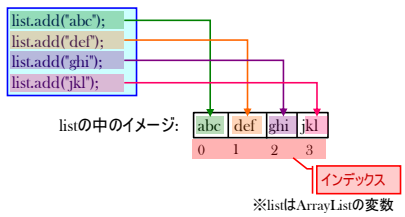
```
list.add(num);
```

```
list.add(10);
```

※listはArrayListの変数, strはString型の変数, numはint型の変数

ArrayListに値を登録(2)

- 値は、登録された順番に、ArrayListの中で並べられる
  - 番号(インデックス)が0から順番につけられる



ArrayListの値を取り出し(1)

- インデックスを指定して取り出す方法
  - 1つ1つの値を個別に取り出し可能
- for文で取り出す方法
  - ArrayListに登録されている値を順番にすべて取り出し

ArrayListの値を取り出し(2)

- インデックスを指定して取り出し
  - getメソッドを利用

ArrayListの変数.get(インデックス)

- 引数: ArrayListに登録されている値のインデックス(int型)
- 戻り値: 登録されている値のデータ型

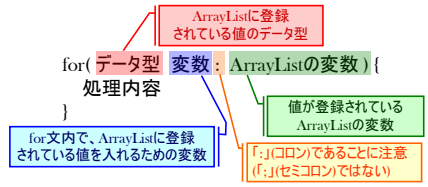
例:

```
ArrayList<String> nameList = new ArrayList<String>();
... 略 ...
String name = nameList.get(3);
```

- インデックス「3」の値を取り出し(最初から4つ目の値)
- ➡ String型の値を登録するようにArrayListを宣言しているので、取り出した値もString型

ArrayListの値を取り出し(3)

- for文で取り出し(1)
  - ArrayList専用のfor文の書き方を利用
  - for文を1回繰り返すごとに、ArrayListの0番のインデックスから順に値を取り出し



ArrayListの値を取り出し(3)

- for文で取り出し(2)
  - ArrayList専用のfor文の書き方を利用

```
ArrayList<String> nameList = new ArrayList<String>();
... 略 ...
int i = 1;
for (String name: nameList) {
    System.out.println(i + "番目の人の名前:" + name);
    i = i + 1;
}
```

変数の宣言と利用

ArrayListに登録されている値が、for文が1回繰り返される毎に、最初から順に1つずつ取り出されて変数nameに入る

プログラム例(ArrayList)

```
list.add("abc");
list.add("def");
list.add("ghi");
list.add("jkl");

for (String str: list) {
    System.out.println("文字列: " + str);
}
```

listの中のイメージ: abc def ghi jkl

実行結果の出力

文字列: abc
文字列: def
文字列: ghi
文字列: jkl

for文の1回目の実行
for文の2回目の実行
for文の3回目の実行
for文の4回目の実行

※listはArrayListの変数

登録されている値の個数

- ArrayListに登録されている値の個数を調べる
  - sizeメソッドを利用

ArrayListの変数.size()

- 引数: なし
- 戻り値: 登録されている値の個数(int型)

例:

```
ArrayList<String> nameList = new ArrayList<String>();
... 略 ...
int count = nameList.size();
```

➡ 変数countに、nameListに登録されている値の個数が入る

ちなみに...

```
int i;
String name;
for (i = 0; i < list.size(); i = i + 1) {
    name = list.get(i);
    処理内容
}
```

同じ処理

```
for (String name: list) {
    処理内容
}
```

※「list」はArrayListの変数

エラー(ArrayList)

実行時のエラー(ArrayList)(1)

- Exception in thread "main" java.lang. IndexOutOfBoundsException
  - ArrayListを使っている場合に、プログラムを実行したときに起こることがあるエラー
  - ArrayListのインデックスで存在しないものを指定した時に起こるエラー
    - 値は50個しか登録していない(インデックスは0～49番までしか使えない)のに50番目のインデックスを使おうとした場合など
    - インデックスにマイナスの数を使おうとした場合など

インデックスを何番まで使おうとしているか
よく確認しよう!

実行時のエラー(ArrayList)(2)

```
ArrayList<Integer> numList = new ArrayList<Integer>();
numList.add(1);
numList.add(2);
numList.add(3);

int i, sum = 0;
for (i = 0; i <= 3; i = i + 1) {
sum = sum + numList.get(i);
}
```

ArrayListに登録している値は3個
(インデックスは0～2を利用可能)

for文は0～3の4回分繰り返し
(繰り返し条件が「<=」のため)

iの値が0の時: sumの値は0 + (numListの0)
iの値が1の時: sumの値は0 + (numListの0) + (numListの1)
iの値が2の時: sumの値は0 + (numListの0) + (numListの1) + (numListの2)
iの値が3の時: sumの値は0 + (numListの0) + (numListの1) + (numListの2) + (numListの3)
を計算しようとする

インデックスは「2」まで使って良い
－「3」というインデックスは使ってはならない
－エラー

※この例は、ArrayList専用のfor文の書き方をすれば、エラーは出ない

配列とArrayListとの違い

- 値の個数
  - 配列は、宣言した時に、扱うことができる値の個数が決められている
    - 宣言した個数を超えて、値を登録することはできない
  - ArrayListは、登録できる値の個数に決まりはない
    - 値の個数を気にする必要がない
- 多次元
  - 配列は、多次元配列で複数種類のデータを1つの変数で扱うことができる
  - ArrayListは、多次元配列のような使い方はできない
    - ArrayListの中にArrayListを登録するような形になる

やってみよう!

- 教科書p. 210の例題01-07をやってみよう

- 追加
  - 5人の生徒の英語の点数の平均を求めるプログラム(for文, ArrayList)
  - 以下処理をするプログラム(ArrayList):
    1. 月曜日から日曜日までに、0から6までの番号をつけて、曜日の名前をArrayListに代入する
    2. 標準入力から数を1つ入力し、その数を7で割った余りを求める
    3. その余りに対応する曜日を標準出力に出力する