Java演習
第5回

2024/5/15

横山大作

講義前・後の質問

- 横山まで
- dyokoyama@meiji.ac.jp

提出課題3(再掲)

- ひな型ソース2つ(FilterTest.java, Pos.java)
- Posクラスのオブジェクトとして表現された座標のリストがある
  - x,yの2次元の要素を保持する
  - ArrayList<Pos>型で作られている
- リストを引数で受け取り、ある基準を満たすPosオブジェクトだけのリストを作って返すような関数filter()を作りたい
  - FilterTest.filter(ArrayList<Pos>list, int th)
  - Posのx,yの両方の要素がともにth以下のものを残す
  - もとのリストはそのまま
    - つまり、filterの中で新たにリストを作って返す
- FilterTest.filter()を完成させよ

解答例

```
public class FilterTest {
    static ArrayList<Pos> filter(ArrayList<Pos> l, int th) {
        ArrayList<Pos> ret = new ArrayList<Pos>();
        for (int i = 0; i < l.size(); i++) {
            Pos p = l.get(i);
            if (p.x <= th && p.y <= th) {
                ret.add(p);
            }
        }
        return ret;
    }
...
```

- メソッド内でnewしたリストを返せばよい
- for文は拡張for文にしても良いですね

この資料の内容

- 文法の基本のおさらいと細かい内容の理解
  - 教科書1,2,3章に相当する部分
    - 式と文
    - 変数
    - 型
    - 演算子
    - 制御構造

- 提出課題4

今日の資料

- 範囲が広いので、「教科書を眺めるべきポイント」を書いてある感じです
- この資料で「知らないキーワードだ」と思う個所は、適宜ググるか教科書を読んで勉強してください
  - 多分忘れているんだと思います

- 言語そのものに関する割と細かいところの説明です
  - 解像度を上げるのに役立つでしょう

値・式・文

- プログラムを構成する代表的な要素
- たいていの言語にはこれらがある

値

- プログラムが動くときには、メモリ上に何か数字ができないとだめですよね
  - 何をするにしても「記号（番号）」が必要
- メモリ空間にできる数字を「値」と呼ぶ

- 値の作り方は限定されている
  - 直書き（即値、リテラル）　3　　-5.2　　“Hello”
  - 変数（値が入っている）
  - **式**
    - 演算　　5 + 2　　1 << 7
    - 関数の適用結果　　sqrt(2)

式(expression)

- 値を作る道具
- 評価されると式全体がある値に書き換わるようなもの

- 6 + 7　→　13　　に書き換わる
- sqrt(9.0) → 3.0　に書き換わる
- i < 3　→　true (iの値によるが)　に書き換わる

文(statement)

- 「やること」を示したもの
  - この式の値を求めなさい、次にその値をここに覚えておきなさい…
- 順番を意識した概念
- Javaの場合、;で終わるのが文の例
  - i = 0;
  - j = compute(i);
  - i = i + j;
- 式の値を求める、制御構造に従って仕事を実行する、などが文の例
  - i = 0;
  - if (j < n) { x = 3; }
- やること、を並べていくプログラミングでは文を並べる
  - 手続き型言語(procedural)

```
if ( i > compute(j) ) {
   i = i + 1;
   j = 0;
} else {
   i = 0;
}
```

```
if ( i > compute(j) ) {
   i = i + 1;
   j = 0;
} else {
   i = 0;
}
```

式

文

```
if ( i > compute(j) ) {
   i = i + 1;
   j = 0;
} else {
   i = 0;
}
```

雑にまとめると

- 式
  - 値になるもの
  - 変数に代入できるもの
- 文
  - やること
  - 値はない

処理系の気持ちを少しだけ

- プログラムを実行するとは、何をやろうとしているのか？

- 文を順番に実行しようとする
- 文の実行のとき、必要なところに式が出てきたら、その値を求める
  - 式を値に置き換える
  - 式が入れ子になっていることもある

```
if (f(i) > f(g(j))) {
   i = i + 1; j = 0;
} else {
   i = 0;
}
```

式と関数

- 関数＝式のある部分を部品として定義したもの
- 関数を呼び出す　＝　関数の値を求める
  　＝　関数の値に書き換える
  - returnは「呼び出したところでreturnの値に置き換える」

関数の呼び出し

- 呼ばれる->仕事が終わったら返ってくる

- 何度も呼ばれると積み重なる
  - スタック　と呼ばれる

こんなイメージでも良いかも

- 関数はその関数内で使う変数とかを用意した引き出し
- 関数を呼び出すと、新しい引き出しが作業用に作られて上に出てくる
- 関数が終わると引き出しの中身が捨てられて閉じられる、古い引き出しがさっき開いた状態で現れる

参考

- 「やること」を並べていくのがプログラミング、というのは手続き型言語というパラダイムでの考え方
- そうでないプログラミングの仕方もある

- 要するにこの値が求まればよいのだ、という考え方もできる
  - 関数型言語などに見られる
  - 文が必要なくなる、式だけでプログラムができている
    - 実はRubyも式だけしかない
  - 何をどういう順番で計算するか、はコンピュータ任せ
  - ある意味、エクセルの表もこの考え方

手続き型言語と関数

- 関数　と　手続き　を分けてある言語もある
  - function と procedure
  - 値を返すものと返さないもの
- JavaやCなどは区別しないことにした
- 値が欲しいだけでなく、「呼び出されて何かの仕事をしてほしい」ものも関数として書いてある
  - 状態をprintしてほしいとか、回数をカウントしてほしいとか
  - 「副作用」と呼ぶ
- 副作用がある関数だと、「どのように呼ばれるか」をプログラマが意識する必要がある

```
while (f(i) != 1) {
   if (f(i) % 2 == 0) {
      i = f(i) / 2;
   } else {
      i = f(i) * 3 + 1;
   }
}
```

- Javaだとf(i)は何回も呼ばれますね
  - 数学的にはこう書いても正しいかもしれない
  - プログラムだとちょっと無駄に見える？
- f(i)に副作用がなく、コンパイラが非常に賢ければf(i)の呼び出し回数は少なくできるかもしれない

構造の復習(p.045)

- クラス
  - メソッド
    - 文、文、文…

- という構造だったことを思い出そう
- だんだん複雑になると
  - クラスの外にメソッド
  - メソッドの外に文
  - とかを書きがち

参考：他の言語では

- Python: やること（文）が順番に並んでいる
  - ソースの頭から順番に実行が進んでいく
- Java（やC）: 部品（宣言、実装）が順番に並んでいる
  - ソースコードは部品を並べるだけ
  - 部品を全部並べ終わると、main()を呼び出す
- 構造が違うことに注意（適当に書かない）

```
a = 3

def f(x):
   return x + 5

b = f(a)
```

```
class Test {
    int a;
    int f(int x) {
        return x + 5;
    }
    void g() {
        int b = f(a);
    }
}
```

3種類の文(p.047)

- この教科書では3種類の文に分けている
  - 変数宣言の文
  - 計算の文
  - 命令実行の文

- この分類でわかりやすいようならそれで考えてみてください
- 命令実行の文、はどちらかというと「式が単体で;を付けて文になっている」ものだけど。
  - 式文　と呼ばれるもの
    - 式の値は求まるんだけど、それをどこにも使っていないもの
  - 計算の文と命令実行の文は分けなくていいんじゃないかな…

時々見る例

```
public class Test {
    int sum;
    int countup(int i) {
        return sum + i;
    }
    public static void main(String[] args) {
        Test t = new Test();
        for (int i = 1; i <= 10; i++) {
            t.countup(i);
        }
        System.out.println(t.sum);
    }
}
```

- 1から10まで足しこみたい
- 「何も起きません！」（0と出力される）

時々見る例

```
public class Test {
    int sum;
    int countup(int i) {
        return sum + i;
    }
    public static void main(String[] args) {
        Test t = new Test();
        for (int i = 1; i <= 10; i++) {
            t.countup(i);
        }
        System.out.println(t.sum);
    }
}
```

これは　式
(が1つだけで文
になったもの)

- 式は値を作るもの
- 値を作っただけ（どこかに取っておくとかがない）

こうするのは？

```
public class Test {
    int sum;
    int countup(int i) {
        sum = sum + i;
        return sum;
    }
    public static void main(String[] args) {
        Test t = new Test();
        for (int i = 1; i <= 10; i++) {
            t.countup(i);
        }
        System.out.println(t.sum);
    }
}
```

これは　式
(が1つだけで文
になったもの)

- 式の中でsumを書き換える文を実行している
- 式に「副作用」がある

変数 (1.3章, p.048)

- 宣言して使う
- 型と名前が大事
  - この型が付いたデータをこの名前で呼ぶ

- 「ふせん」のようなもの、という説明を思い出そう

名前の話 (p.49,50)

- クラス名、変数名、メソッド名など: 識別子 (Identifier)
- 使える文字は限られている
- 先頭が数字では始められない
- Javaのキーワードは使えない

- 名前の付け方は慣習として２通り
  - キャメルケース: minPriceInRange
  - スネークケース: min_price_in_range

  どちらかに統一を
    - 定数は大文字のスネークケース MIN_PRICE　にすることもある

スコープ

- 変数の「使える」範囲
  - 宣言したところから使えなくなるところまで

- メソッド内のローカル変数は宣言からメソッドの最後まで
- 別のメソッドが呼ばれると変数はいったん見えなくなる

スコープも関数の呼び出しと同じ

変数はなるべく短命に

- スコープはなるべく小さくしよう
  - 使う直前で宣言するようにしよう
  - 同じ変数を何回も使いまわさないようにしよう

  - for (int i = 0; ...; i++) {...} を使おう
  - while (i < n) { ...; i++; } と「ココロ」が少し違う
    - for文の方が便利な場面も多いけど
  - ニュアンスにこだわれると「わかってる」感じになる
- ifやforなどのブロックを利用して短命にしよう
  - もちろん関数に分けてもよい
- 単なる{}でくくることもできる
  - やや「最後の手段」的ではあるが

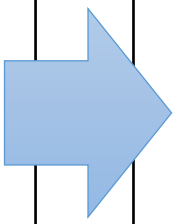
なるべく短命にする例

```
void method() {
    int a;
    int b;

    a = 10;
    if (a > 0) {
        a = a + 1;
        b = 0;
        // ...

    }

    // これ以降bは使わない…
}
```



```
void method() {



    int a = 10;
    if (a > 0) {
        a = a + 1;
        int b = 0;
        // ...

    }

    // これ以降bは使わない…
}
```

- 右の例でif(){}の外側でbを使おうとするとコンパイルできないことにも注意

型 (p.51)

- 基本データ型（プリミティブ型）8種類
  - boolean
  - char, byte, short, int, long
    - charはUnicode文字で2バイトあることにちょっとだけ注意
  - float, double
- このページの中では、String型は基本データ型ではない（参照型）

式と演算子（第2章）

- 文が式だったら、式の値を求めることをやっていく
- 式の値は決まった優先度で順番に評価される(p.68)

- 値を作れるのは
  - リテラル
  - 演算子
  - 関数呼び出し
  - 変数（値が入るもの）
- なので、それを組み合わせたものが式

リテラル(p.64)

- ソースコード中に書かれる「値」
- 変数「名」ではなく、値そのものを書いたときの呼び方
- int a = 1;　// aは変数、1はリテラル
- null, true, falseなどもリテラル

整数リテラル(p.64,65)

```
int a = 12;
long b = 12345678901L;
int c = 020;
int d = 0x2b;
int e = 0b11;
```

- 書き方見ておきましょう
- long型を示すには後ろにL
- 8進リテラルに注意
  - 先頭を0 (ゼロ)から始めると8進数だと思われる
- 2進リテラルは頭に0b (ゼロビー)
- 大きい数字を読みやすく書く(Java7以降)
  - カンマ区切りの代わりにアンダーバー入れてよい
  - long x = 123_456_789L;

文字リテラル

- char
- シングルクオートで囲むのが基本

- ¥n (LF, Line Feed)
- ¥r (CR, Carriage Return)

- 改行がらみはとにかくややこしい
    - C言語ではプログラム中での表現とファイル上の表現を変換したりする
    - Javaは基本的には変換しない
  - まぁ試行錯誤で

エスケープシーケンス

- ’とか”とかの文字はどう表す？
- 特別な文字を付けたら良い: エスケープシーケンス

```
char a = '¥'';
char b = '¥¥';
String c = "¥¥¥"¥n";
```

- ¥を付けると特別扱い
- ¥自身も特別扱いが必要になるので注意

- データ圧縮とかでもよく出てくる考え方

- ¥　と　＼（バックスラッシュ）　は同じなので環境に合わせて読み替えて

テキストブロック

- 改行を含む複数行の文字列をソースに書きやすくするための仕組み
- """ を両端に付けて囲む
- p.68

評価の仕組み(p.70)

- 優先度に合わせて計算していく
  - 普通の数学と同じだと思ってよい
  - 迷ったら（）を付けておこう

演算子(p.74)

- 型によってやることが変わる
  - 例：String型、int型での”+”
    - intの+は足し算、Stringの+は文字列連結

- 代入演算子
  - 代入して、その値を返すような演算子
  - a = 3; は「代入文」だと思ってもいいし、「代入演算子が入った式が単体で置いてある」と思ってもいい
    - 実は言語によって立場が違っていたりする（代入文だと扱って、式の中に書けない言語がある。BASICとか）
    - 代入演算子は副作用がある式

参考

- a = b = 5;
- 代入文の時　(BASICなど)
  - a = (b = 5);　// a =の右は文が来れないので、式だ
    - b = 5 は比較演算だ（BASICなら比較演算がある）
    - a にはbooleanが入る
  - (pythonだとまた別の扱いでa=5にしている)
- 代入式の時　(Javaはこの立場)
  - a = (b = 5);　→ b = 5で5を返して、それをa=5とする
  - while ((c = getchar()) != null) の書き方しようとすると式じゃないとだめ

- pythonに代入式 := が入ったことで有名
  - セイウチ演算子

前置と後置 (p.78)

- ++と--を前に置くか後に置くか

  「特別な理由がない限り、単独で使うように心がけましょう」(p.78)
  「インクリメントの前と後の挙動の違いくらいはJavaプログラマのたしなみとして知っておいていいと思いますが、人が見て「間違いやすい」と思うようなコードは書かないようにしましょう」(基礎からのJava p. 61)

異なるデータ型の演算(p.79)

- 異なるデータ型の数値演算は「なるべく良さそうな方に合わせる」
  - なるべく大きいほうに、小数のほうに合わせる

- おかしくなりそうならコンパイルエラーが教えてくれる
  - エラーは怖くない、エラーは友達

キャスト(p.83)

- 意図せずデータが失われる可能性がある時は明示的な指示が必要
  - 大きい範囲の整数を小さい範囲の整数に入れるとき
  - 小数から整数に入れるとき

  など
- コンパイラが気を付けてくれる
  - ミスじゃないですか？わかっているのならそのように書いてくださいね、という気の使い方
- キャストと呼ぶ
  - c言語にもあった、見た目も同じ

キャストの例

```
int num = 1;
long bignum = 10;

bignum = num; // int ⇒ long は大丈夫

num = bignum; // これはコンパイルエラー
```

```
int num = 1;
long bignum = 10;

num = (int)bignum; // キャストすればOK
int x = (int)1.2; // 小数から整数もキャスト必要（切り捨て）
```

- 無理な型もあるので注意
  - booleanにintはキャストしても入らない　とか

命令実行の文(p.88)

- Javaが用意してくれている関数を呼び出す文、というニュアンスで説明してるみたい
- 要するに関数呼び出しする文
- 関数って式の評価の時にも呼ばれるので、分ける必要あるのかな…
  - と、この教科書の説明にもやもやする

制御構造: 条件分岐 (第3章, p.101)

- if文、if-else文の書き方は大丈夫ですね？
  - 文; 文; 文… の中にif (式) { 文 } という文を書いてよい、ということ
  - p. 117 複雑な場合も見ておきましょう
- switch文(p. 118)というのもありますね
  - case で値と比較
  - defaultがある
  - 新しい書き方ができた
    - 古い書き方はc言語の書き方と同じ
    - 古い書き方だと、breakを書かないとそのまま後ろのcaseに突入することに注意

制御構造: 繰り返し (p. 123)

- for,while文は大丈夫ですね？
- Do-while文もありましたね(p. 124)
- breakとcontinueもありましたね (p. 131)
  - break,continueとfor,whileの組み合わせの動作を確認しておきましょう
    - 特にwhile+continueでループ制御変数の変更忘れが起きがち

```
while (c > 0) {
  if (x % 2 == 0) {
    continue;
  }
  // 何かやって
  // いろいろやって
  // もっとやって
  c--;
}
```

文字列の比較 (p. 111)

- 文字列を比較するときにちょっとだけ注意
- if (s == “test”) { … } みたいな比較は「原則ダメ」
- 理由は次週に説明します
- 2つの文字列を比較するときは「原則、equals() メソッドを使う」ようにしましょう
- if (s.equals(“test”)) { … }

- ちなみに、switch文も大丈夫(equals()を内部で使っている)

関係演算子(p.108)、論理演算子(p.112)

- 主にboolean型を扱うための演算
  - >, >=, ==, != …
  - &&, ||, ^, !

- 実は&&と& があり、動きが少し異なる(p.114)
  - &&は結果が決まったところで評価を打ち切る
  - &は両方を必ず評価して論理値を出す
  - ||と|も同様
  - まぁどうでもいいです

日本語の論理演算はあいまい

「コーヒーか紅茶が付きます」

- コーヒー or 紅茶　ではない
- コーヒー xor 紅茶　です

「牛か豚のひき肉を準備してください」（牛や豚）

- 牛 or 豚で良さそう（合い挽きがあり得る）

提出課題4

- Posクラスのオブジェクトのリストがある
- リスト内のすべての座標の要素をa倍するようなメソッドを作りたい
  - (3,6)という座標を3倍したら(9, 18)
- ただし、2通り
  - もとのリストはそのままで新たなリストを作って返す
    - dup_mult()
  - もとのリストの要素をa倍に書き換える
    - modi_mult()

提出課題4(続き)

- MultTest.javaのdup_mult()とmodi_mult()を完成させよう
- Pos.javaは与えられているのでそのまま使う
  - PosArray.javaと同じディレクトリに入れておけばよい（eclipseでクラスをnewして、上書きしてもよい）
- mainは、dup_mult()と、modi_mult()の結果を表示する
  - 最後に、もう一度さっきのdup_mult()の結果を表示する。どうなっているべきだろうか？

提出物

- 提出物はMultTest.java
  - 先頭に「組番号、名前」と、出力された文字列をコメントで記入
  - package javalec4;
- 答えが正しいことをきちんと確認しよう
- ∠切は5/21(火) 17:00

ヒント

- Posクラスにはコンストラクタありますね。
  使ってみよう。
- String toString()というメソッドは、Javaでは
  「わかりやすい表示用の文字列に変換するメソッド」として扱われることが決まっています。
  PosArray.javaの中では、これを(暗黙のうちに)
  使って表示が行われていますね。

今日のまとめ

- 文法の基本のおさらいと細かい内容の理解
- 教科書1,2,3章に相当する部分
  - 式と文
    - 違いを自信もって言えるように
  - 変数
  - 型
    - キャスト
  - 演算子
  - 制御構造