# 情報処理技法
# (Javaプログラミング)1

第3回
コンピュータが情報を扱うには?(変数, データ型, 代入)(続き),
語句や文章を扱いたいときは?

人間科学科コミュニケーション専攻
白銀 純子



## 第3回の内容

- プログラムで扱うデータのおはなし(続き)



## 前回の復習問題の解答

- プログラムの記述・コンパイル・実行は、それぞれどのようなソフトウェアを使って行うか、また、コマンドを使う場合にはどのようなコマンドを使うかを解答しなさい。
  - 授業で説明したソフトウェアでなくても、記述・コンパイル・実行ができるソフトウェアであればかまいません。

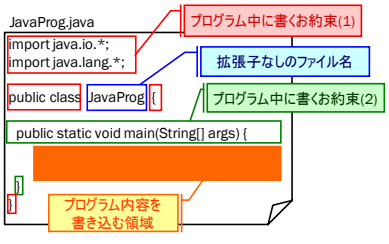
解答例
記述には、JeditやEmacsなどのテキストエディタを使う。コンパイルは、
javac Javaファイル名(拡張子つき)
というコマンドで行い、実行は
java Javaファイル名(拡張子なし)
で行う



## 前回の復習

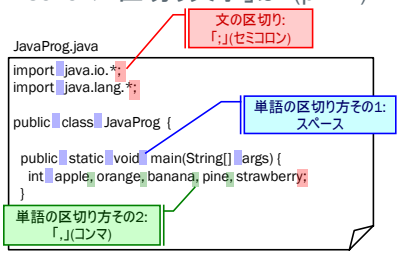


## プログラムの「カタチ」は?(p. 38)






## Javaの「区切り文字」は?(p. 44)



## プログラミングでのエラー(p. 49)

- プログラム作成時に、エラーでうまくいかないことも多い
  - コンパイル時に表示されるエラー: コンパイルエラー
    - スペルミスをした
    - カッコを開き忘れ・閉じ忘れた
    - 必要な場所に必要な命令を書いていなかった, etc.

    プログラム中の文法間違い、という意味のエラー
  - コンパイル後、実行時のエラー: 例外
    - 数を0で割ろうとした
    - 使ってはならない番号を使おうとした(配列など), etc.

    プログラムに文法間違いはないが、何らかのミスでそれ以上実行できない、という意味のエラー



## コンパイルエラーの基本形(p. 49)

基本的なコンパイルエラーのメッセージの形

*XXX.java*:*n*:メッセージ
プログラム中の文
^

*XXX.java*: コンパイルしたファイル名
*n*: エラーが見つかった行数(「n行目にエラーがある」という意味)
^:「プログラム中の文」の中のあやしい部分(間違っていそうな部分)



## コンパイルエラーへの対処の基本(p. 49)

- コンパイルエラーには一番上から順に対処すること
  - コンパイルエラーがたくさん出てきたときは、多くの場合、上の方に出ているメッセージがより適切な意味
  - 1つのまちがいが影響していろいろな部分のメッセージを出すことも
    - 例えば、宣言していない変数を5箇所で使っていたら、5つエラーメッセージが出てくる
- 「メッセージ」の部分をよく読み、エラーの意味を理解すること
- Jeditで、エラーが出た行番号のところをよく見て、ミスを探すこと



## プログラムで扱うもの~データ型~(p. 56)

- 整数 → プログラムでの表現: int
- 小数 → プログラムでの表現: float または double
- 文字 → プログラムでの表現: char

プログラムでのデータは、どの系統になるか決めておく必要あり

「データ型」と呼ぶ



## データの「種類」は?(p. 57)

- 1つ1つのデータにそれぞれ名前をつける

例えば...

「変数」と呼ぶ

購入するりんごの数(int): apple
量った牛肉の分量(float): meat
自分が働いている陳列棚のエリア(char): area

系統(データ型): int
種類(変数名): apple

系統(データ型): float
種類(変数名): meat

系統(データ型): char
種類(変数名): area

「変数」= データを入れるための箱

データは原則として、必ず箱の中に入れて扱う



## 変数の宣言(p. 59)

- 変数を使う(データを入れるなど)前に、変数を準備する必要

変数を「宣言する」という
= それぞれの箱が、「肉・魚系統」か「野菜系統」か 「飲み物系統」かをコンピュータに知らせ、箱を準備する

スペース

準備(宣言)例

```
int apple, orange, banana;
float meat, chicken;
char area, register;
```

変数の系統(データ型)を先頭に書く

「,」で区切って複数の変数を予告(宣言)できる

## 変数の値(p. 62)

- 変数(箱)に値(データ)を入れて扱う
- 「=」で値を決める ― 「代入する」という
  = 箱の中に具体的なデータを入れること
  ※「=」の左側は、必ず変数1つだけ
  - 用意した変数に初めて値を入れること: 初期化

購入したりんごの数が10個だった場合

apple = 10; → 


牛肉の分量を量ったら200.5gだった場合

meat = 200.5; → 




## 数の計算(p. 66)

- 変数(箱)の中には、計算結果や処理結果を入れることもできる
  - 足し算: +
  - 引き算: -
  - かけ算: *
  - 割り算(商): /
  - 割り算(余り): %

例えば…代金計算
(支払い金額: result)

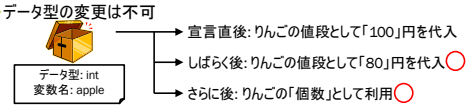
100円のりんごを10個買った場合
apple = 10;
result = apple * 100;
「result」には、「1000」という結果が入る

100円のりんごを10個,
150円のバナナを5個買った場合
apple = 10;
banana = 5;
result = apple * 100 + banana * 150;
「result」には、「1750」という結果が入る



## 変数の使いまわし

- 変数は使いまわし可能
  - 同じ変数の宣言は1度だけで良く、何度も宣言する必要はなし
  - データ型の変更は不可




ただし…すでに値が入っている変数に別の値を代入すると…




## 代入の不思議(p. 67)

「milk」を、店にある牛乳の在庫のパック数と考えると…
トラックが来る前: 在庫のパック数は30

milk = 30;

トラックが牛乳を50パック運んできた
この後の店の在庫数の計算は?

milk = milk + 50;
- milk(左辺): トラックが来た後の在庫数
- milk(右辺): トラックが来る前の在庫数

- 「=」より後の変数は、直前までに代入されていた値
- 「=」より前の変数には、「=」より後の計算結果を代入
  (値が新しいものに更新される)



## 変数宣言の注意(p. 61)

- 同じ名前の変数は、1回しか宣言できない

```
public static void main(String[] args) {
  int abc;
  ..................................
  int abc = 10;
}
```

コンパイルエラーが出る(一度宣言した変数は何回でも使えるので、「int abc = 10;」の「int」は不要)

- 変数には、宣言と同時に値を代入してよい

```
int abc;
abc = 10;
```

```
int abc = 10;
```

同じ意味を表す



## 変数の宣言忘れに注意(p. 61)

- どの変数であっても、宣言していなければ使えない

```
int result;
result = banana + 10;
```

変数「banana」の宣言をしないまま、「banana」のデータを使って計算しようとしている

「シンボルを処理解釈できません」というエラーメッセージ
宣言していない変数はすぐ後に書かれているので、よくメッセージを読んで宣言をすること
※スペルミスの可能性もあるので、要注意

## 代入と参照の注意(p. 67)

- 初期化をしないと、変数を参照できないので注意
  - 箱の中にデータが入っていないので、存在しないデータを使って計算などはできないため

```
int  banana, result;
result = banana + 10;
```

変数「banana」の初期化をしないまま、「banana」の中からデータを取り出して計算しようとしている

「変数banana は初期化されていない可能性があります」というエラーメッセージ

初期化が必要な変数(この変数を初期化すること)
※どのような値を代入すれば良いかはそのときどきでよく考えること




## プログラムの記述とコンパイル・実行

- プログラムの内容
  - JeditまたはEmacsでなどのエディタ記述し、ファイルとして保存
- コンパイル・実行
  - ターミナルで、保存したファイルを指定

エディタとターミナルは、どちらをどのように使うか、きちんと区別しよう!




## コンパイル・実行時の注意(1)

- ターミナルでのカレントフォルダを、Javaファイルを保存しているフォルダに設定すること
  - カレントフォルダ: ターミナルでの、現在の作業フォルダ
  - ターミナルを起動したとき: カレントフォルダはホームフォルダ




## コンパイル・実行時の注意(2)

- カレントフォルダの変更のコマンド:
  - コマンドの入力
    % cd ***ホームフォルダからの相対パス***
  - Ex1. ホームフォルダの中で、「Desktop」→「Java」→「chap」に保存してある場合
    (相対パス: Desktop/Java/chap):
    % cd ***Desktop/Java/chap***
  - Ex2. ホームフォルダの中で、「Download」→「chap」→「chap01」に保存してある場合(相対パス: Download/chap/cha01):
    % cd ***Download/chap/cha01***




## やってみよう!(2)

- 教科書p. 76の例題01-07をやってみよう




## 文字列の扱い

## 文字列とは(p. 80)

- 文字を並べたもの
- 言葉や文章:
  コンピュータにとっては1文字1文字が並んでいるもの
  コンピュータは意味をわかっているわけではない

例えば...blue

人間: 青い「色」と解釈
コンピュータ: 最初に「b」があり、その次に「l」があり、その次に「u」があり、最後に「e」という文字の並びと解釈

人間の考え方も、コンピュータにあわせる



## 文字列の扱い(p. 80)

- 文字列はいろいろな情報を持っている
  - 文字の並び
  - 文字列の長さ(文字の数)
- 文字列にはいろいろな操作ができる
  - *n*番目の文字を取り出す
  - *m*番目の文字から*n*番目の文字までで部分文字列を作る
  - 文字列中の部分文字列を、別の文字列に置き換える
  - etc.

intやfloatなどの
数値とは扱い方が違う!



## データ型(p. 81)

- 文字列のデータ型:String
  最初の「S」は大文字、あとは小文字
- 変数を宣言する方法は、intやfloatなどと同じ
  - String str1, str2;
- 変数でない値を代入するときは、値を「"」で囲む
  - str1 = "abc"; (「abc」は変数でない文字列)
- 変数を代入するときは、「"」は不要
  - str1 = str2; (「str2」はString型の変数)

※変数でない値は、日本語でもOK



## 文字列への値の代入

- 文字列は0文字以上で代入可能
  - 0文字の文字列: String str = "";
  - 1文字の文字列: String str = "a";
  - 2文字の文字列: String str = "ab";
  - ...



## 文字列をつなげて文章を作る(p. 85)

- 2つ以上の文字列をつなげるとき:「+」記号でつなげる

例1: str1の値が「abc」、str2の値が「def」のとき、
str3に、str1とstr2をつなげた「abcdef」を代入したい

str3 = str1 + str2;

例2: str1に「Hello」、str2に「World」が入っているとき、
str3に「Hello, World!」を代入したい

str3 = str1 + ", " + str2 + "!";
(スペース)

※1文字でも、文字列として扱うことができる



## String型のデータ(p. 82)

- 「"」で囲まれた言葉は、コンピュータにとってただの文字列
- 「"」で囲まれていない言葉は、コンピュータにとっては変数

```
String str;
str = "abc";
str = abc;
str = "abc" + def + "ghi";
```

- "abc": ただの文字列なので問題なし
- abc: 変数として扱われるので宣言をしていなければコンパイルエラー
- def: 変数として扱われるので宣言をしていなければコンパイルエラー
- "abc", "ghi": ただの文字列なので問題なし

「"」が必要なときと不要なときをきちんと使い分けよう!

## 文字列のつなげかた(p. 85)

1. できあがりの文字列をイメージする
   金額は1000円です。
2. 変数・単なる文字列ごとに分解する
   金額は 1000 円です。
3. 変数や「"」つきの文字列に置き換える
   "金額は" payment "円です。"
4. 変数・単なる文字列の間に「+」をつける
   "金額は" + payment + "円です。"



## エスケープシーケンス(1)(p. 83)

◆プログラム中で扱うには、いくつか特殊な文字が存在
◆Ex.「Hello, "World"!」というデータを扱いたい場合


「"」の区別がつかない
➢変数でない文字列を囲むための「"」
➢データとしての「"」(「World」を強調するための「"」)



## エスケープシーケンス(2)(p. 83)

◆特殊な文字の区別
◆プログラム中で何らかの処理の一部を表す文字
◆普通に書く
◆単なるデータとしての文字列の一部を表す文字
◆特殊な表記で書く

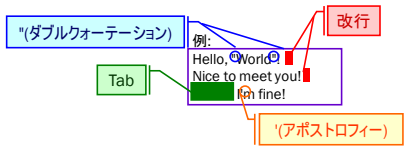
改行, ¥, Tab, ", '

エスケープシーケンス



## エスケープシーケンス(3)(p. 83)

◆改行:「¥n」
◆¥:「¥¥」
◆Tab:「¥t」
◆'(アポストロフィー):「¥'」
◆"(ダブルクォーテーション):「¥"」




String str = "Hello, ¥" World¥" ! ¥nNice to meet you! ¥n¥t I ¥' m fine!"



## 考えてみよう!

◆教科書p. 111の例題01-02をやってみよう



## 文字列に対する操作(p. 87)

◆文字列を扱うために、Javaには様々なメソッドが用意されている

メソッドの形:
String型の変数.メソッド名(引数, 引数, ...)
引数の順番と数、データ型は、それぞれのメソッドで決まっている
(「,」でつなげて書く)

メソッドは、様々な処理をしてその結果を返してくれる
→返してくれた結果(戻り値)を、変数に代入して使う(例えば、
int num=String型の変数.メソッド(...);
のようにして使う)
※戻り値のデータ型はメソッドによって決まっている

## メソッド(p. 87)

- プログラム中で行われる処理の手順をまとめたもの
  - 複数の処理をまとめて、1つの名前を付けたもの
- メソッド名、引数、戻り値(戻り値)から成る
  - メソッド名: メソッドの名前
  - 引数: メソッドに渡す情報
  - 戻り値(戻り値): メソッドから返される処理結果
    - 多くの場合、戻り値を変数に代入して利用する
    - 「変数名 = メソッド」で、変数に戻り値が代入される




## 文字列の文字の数え方(p. 88)

- プログラムでは、文字列の文字は0番目から数える
  - 文字の順番を表す番号を「インデックス」と呼ぶ

例えば..abcdefghij

a: 0番目
b: 1番目
c: 2番目
.....
j: 9番目




## 文字列の長さ(文字数)(p. 88)

- 「length()」というメソッドを使う

```
文字列型の変数.length();
```

int型で結果をもらう

例:

```
int strLength;
String str1="abc";
```

strLength:str1の文字数
str1の長さを求めたいときは?

```
strLength=str1.length();
```




## 部分文字列の最初の出現場所(p. 89)

- あるインデックス以降で、部分文字列が最初に出現する場所
  - 部分文字列: 文字列の一部
- 「indexOf(*str*)」というメソッドを使う

```
文字列型の変数.indexOf(str);
```

- int型で結果をもらう
- 「*str*」は探したい部分文字列 (String型)

例:

```
int index;
String str1="abcdefabcabcab";
```

str1での「abc」が最初に出てくる位置を求めたいときは?
(答え: 0)

```
index=str1.indexOf("abc");
```

※探したい文字列がなかったときは、結果が「-1」になる




## 部分文字列の出現場所(p. 91)

- あるインデックス以降で、部分文字列が最初に出現する場所
- 「indexOf(*str, n*)」というメソッドを使う

```
文字列型の変数.indexOf(str, n);
```

- int型で結果をもらう
- 「*str*」は探したい部分文字列 (String型)
- 「*n*」は、調べ始めるインデックス

例:

```
int index;
String str1="abcdefabcabcab";
```

str1のインデックス1以降で、「abc」が最初に出てくる位置を求めたいときは?
(答え: 6)

```
index=str1.indexOf("abc", 1);
```

※探したい文字列がなかったときは、結果が「-1」になる




## 部分文字列の最後の出現場所(p. 93)

- あるインデックス以降で、部分文字列が最後に出現する場所
- 「lastIndexOf(*str*)」というメソッドを使う

```
文字列型の変数.lastIndexOf(str);
```

- int型で結果をもらう
- 「*str*」は探したい部分文字列(String型)

例:

```
int index;
String str1="abcdefabcabcab";
```

str1での「abc」が最後に出てくる位置を求めたいときは?
(答え: 9)

```
index=str1.lastIndexOf("abc");
```

※探したい文字列がなかったときは、結果が「-1」になる

## 部分文字列の出現場所(p. 95)

◆あるインデックス以前で、部分文字列が最後に出現する場所

◆「lastIndexOf(*str, n*)」というメソッドを使う

文字列型の変数.lastIndexOf(*str, n*);

- int型で結果をもらう
- 「*str*」は探したい部分文字列 (String型)
- 「*n*」は、調べ始めるインデックス

例:

```
int  index;
String  str1="abcdefabcabcab";
```

str1のインデックス8以前で、「abc」が最後に出てくる位置を求めたいときは?
(答え: 6)

➡ index=str1.lastIndexOf("abc", 8);

※探したい文字列がなかったときは、結果が「-1」になる



## 部分文字列(1-1)(p. 97)

◆m番目の文字からn番目の文字までで部分文字列

◆「substring(*m, n+1*)」というメソッドを使う

文字列型の変数.substring(*m, n+1*)

- 「文字列型の変数」: 元の文字列
- String型で結果をもらう

- m: 部分文字列の最初の文字の、元の文字列での位置(int型)
- n: 部分文字列の最後の文字の、元の文字列での位置(int型)

例:「abcdefghi」から、「def」という部分文字列を作りたい

- 部分文字列の最初の文字: d
- 「d」の元の文字列での位置: 3

- 部分文字列の最後の文字: f
- 「f」の元の文字列での位置: 5

➡ mは3, nは5と考える



## 部分文字列(1-2)(p. 97)

◆m番目の文字からn番目の文字までで部分文字列

◆「substring(*m, n+1*)」というメソッドを使う

文字列型の変数.substring(*m, n+1*)

- 「文字列型の変数」: 元の文字列
- String型で結果をもらう

- m: 部分文字列の最初の文字の、元の文字列での位置(int型)
- n: 部分文字列の最後の文字の、元の文字列での位置(int型)

「文字列型の変数.substring(m, n)」とすると...

- 「*m*」番目の文字は、新しい文字列に入る
- 「*n*」番目の文字は、新しい文字列には入らない

➡ *m*番目から*n*番目の文字列を作るときには、substringに「*m*」と「*n+1*」を渡す



## 部分文字列(1-3)(p. 97)

例:

```
String  fullString="abcdefghi";
String  shortString;
```

fullStringの3番目から5番目の部分文字列を求めたいときは?
(答え: def)

➡ shortString=fullString.substring(3, 6);

注意: 文字列は、0番目から数える



## 部分文字列(2-1)(p. 99)

◆m番目から最後の文字列までで部分文字列

◆「substring(*m*)」というメソッドを使う

文字列型の変数.substring(*m*)

- 「文字列型の変数」: 元の文字列
- String型で結果をもらう

m: 部分文字列の最初の文字の、元の文字列での位置(int型)

例:「abcdefghi」から、「e」以降の部分文字列を作りたい

- 部分文字列の最初の文字: e
- 「e」の元の文字列での位置: 4

➡ mは4と考える



## 部分文字列(2-2)(p. 99)

例:

```
String  fullString="abcdefghi";
String  shortString;
```

fullStringの4番目以降の部分文字列を求めたいときは?
(答え: efghi)

➡ shortString=fullString.substring(4);

注意: 文字列は、0番目から数える

## 2つの文字列を比較(p. 104)

◆「equals(*str*)」というメソッドを使う

文字列型の変数.equals(*str*);

- 「str」は等しいか比べたい文字列(String型)
- boolean型で結果をもらう

例:

```
String  str1="abcdef";
String  str2="abcijk";
```

str1とstr2は同じ文字列?
(答え: false)

str1.equals(str2);

※「str2」は変数でなくてもよい
つまり、「str1.equals("abcdef");」という書き方もOK



## 半角アルファベットを小文字化

◆「toLowerCase()」というメソッドを使う

文字列型の変数.toLowerCase();

アルファベットが小文字になった結果をもらう
(もらう結果はString型)

例:

```
String  upper="ABCDEF";
String  lower;
```

upperを小文字にしたい
(答え: abcdef)

lower=upper.toLowerCase();



## 半角アルファベットを大文字化

◆「toUpperCase()」というメソッドを使う

文字列型の変数.toUpperCase();

アルファベットが大文字になった結果をもらう
(もらう結果はString型)

例:

```
String  lower="abcdefghi";
String  upper;
```

lowerを小文字にしたい
(答え: ABCDEFGHI)

upper=lower.toLowerCase();



## よくある使い方

◆indexOf, lastIndexOf, substringを組み合わせて使う

- ◆ある文字で区切られた文字列を分解する場合など

例えば...「,」で区切られた3つの言葉を1つ1つの言葉として取り出す場合
(「apple, pine, banana」を「apple」と「pine」と「banana」に分解)

```
int  m, n;
String first, second, last, original = "apple,pine,banana";
m = original.indexOf(",");
n = original.lastIndexOf(",");
first = original.substring(0, m);
second = original.substring(m + 1, n);
last = original.substring(n + 1);
```



## やってみよう!

◆教科書p. 111の例題03-06をやってみよう