

# 情報処理技法 (Javaプログラミング)1

## 第5回

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5回の内容

- 文字列の扱い方

# 前回の復習問題の解答

- 下記のプログラムを実行し終わったとき、変数resultの値がいくつになっているかを答えなさい。

```
public class Question4th {  
    public static void main(String[] args) {  
        int result = 0, apple = 5, banana = 2;  
  
        result = apple * 100 + banana * 200;  
  
        int pine = 1;  
  
        result = result + pine * 250;  
    }  
}
```

この時点でresultの値は900

結果、 $900 + 250 = 1150$

解答: 1150

# 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

# 文字列をつなげる(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 + "!";`  
                                <sup>スペース</sup>

※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";
```

← ただの文字列なので問題なし

← 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

↑ ただの文字列なので問題なし

↑ 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

「"」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

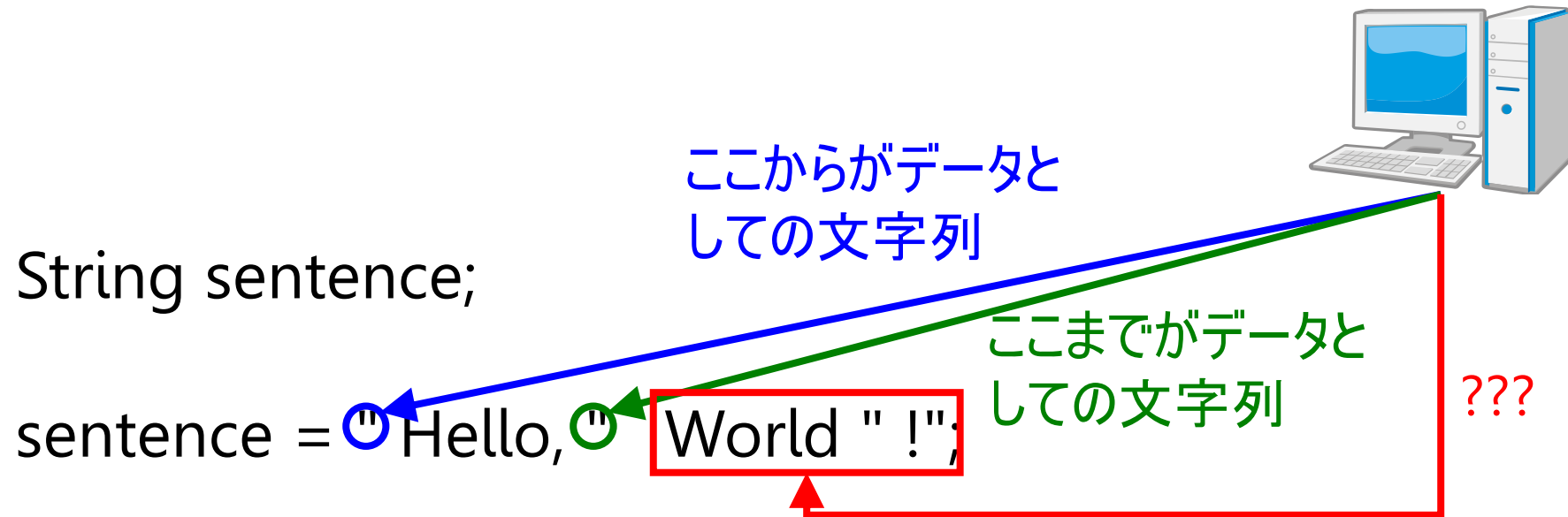
"金額は" payment "円です。"

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

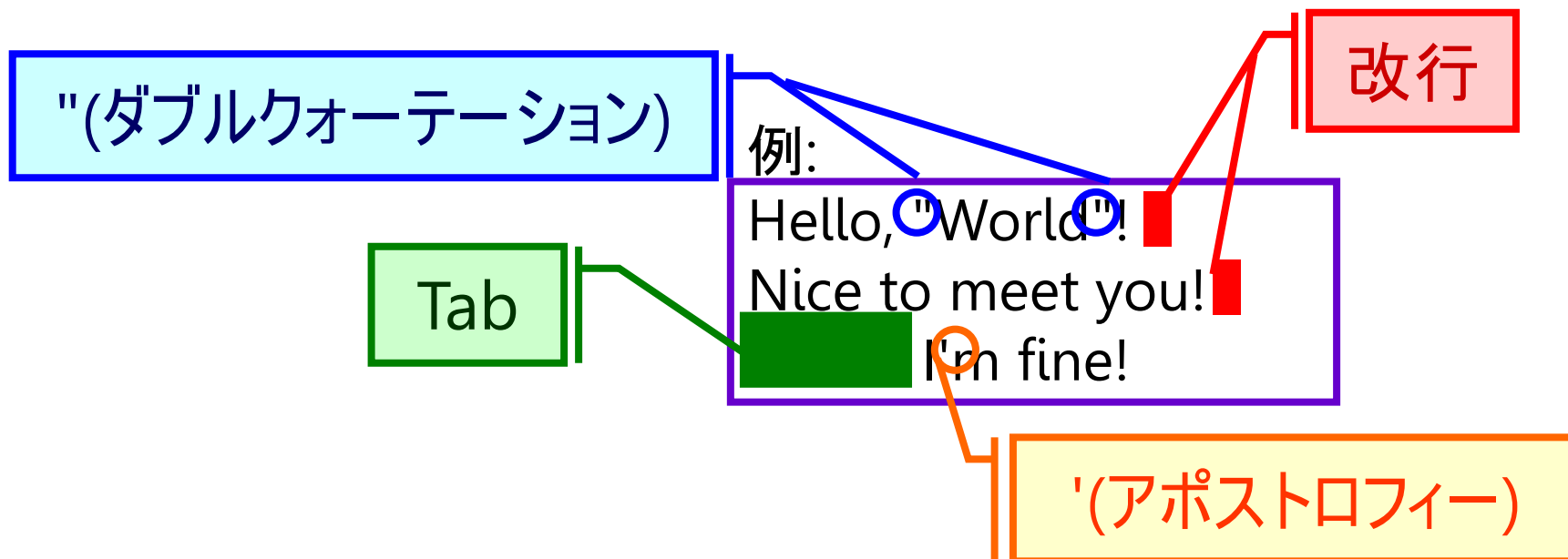
- プログラム中で何らかの処理の一部を表す文字
  - 普通に書く
- 単なるデータとしての文字列の一部を表す文字
  - 特殊な表記で書く

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」
- "(ダブルクォーテーション): 「¥"」
- ¥: 「¥¥」
- '(アポストロフィー): 「¥'」



➡ 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();`

# 最初から $n$ 番目の文字

- 「charAt( $n$ )」というメソッドを使う

文字列型の変数.charAt( $n$ );

- char型で結果をもらう
- 引数「 $n$ 」はint型( $n$ 番目の意味)

例:

```
char letter;  
String str1="abcdef";
```

str1の3番目の文字を求めたいときは?  
(答え: d)



```
letter=str1.charAt(3);
```

注意: 文字列は、0番目から数える

# 部分文字列の最初の出現場所(p. 89)

- 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場所(p. 93)

- 部分文字列: 文字列の一部
- 「`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)
- 「文字列方の変数.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.toUpperCase();

# 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