第19章　クラスライブラリ

目次

クラスライブラリとは

汎用的な機能を持ったプログラムの集まりのこと。
パッケージ化されて提供されており、
機能を1から作成せずに利用できる。

クラスライブラリとは

クラスライブラリ中には異なるクラスで
同名のメソッドが存在する場合がある。
文書中でメソッドを区別する際には
下記の形式でメソッドを表記することが一般的。

```
クラス名#メソッド名()
```

文字列を扱うクラス

Stringクラスは文字列を扱う為に設計されたクラス。

文字列はこのクラスから生成された
オブジェクトとして扱われる。
文字列を操作するメソッドも定義されている。

（メソッド一覧はテキスト参照）

文字を検索

String#indexOf()メソッド

文字列の中から引数で指定された文字列を検索して、
その先頭文字が最初にあらわれる位置(の番号)を返す。

String#substring()メソッド

開始位置と終了位置を指定して文字列を抽出する。
第1引数:切り出す開始位置の番号
第2引数:終了位置の番号
実際に切り出される文字列は
「開始位置」から「終了位置 - 1」までの範囲となる。

【Sample1901 メールアドレスからユーザ名を取り出す】を作成しましょう

Sample1901のポイント

indexOf()メソッドを呼び出し、
その戻り値をsubstring()メソッドの第2引数に渡している。

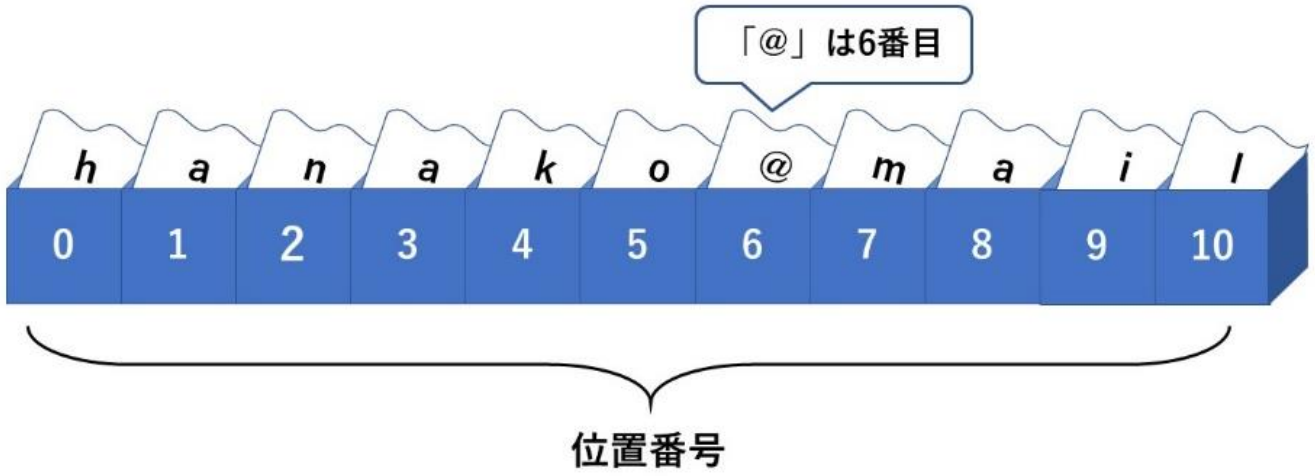
```
String userName =
    mailAddress.substring(0, mailAddress.indexOf("@"));
```

検索対象の文字列が「@」であるため、
indexOf()メソッドの戻り値は6となる。

Sample1901のポイント

substring()メソッドは第一引数番目から第二引数-1番目の値を切り出すため、0番から5番までの文字列が切り出される。

```
String userName =
        mailAddress.substring(0, mailAddress.indexOf("@"));
```



文字列の長さを調べる

文字列の長さ(文字数)を調べる時は、
String#length()メソッドを使用する。

```
// 文字列を返す
int len = str.length();
```

length()メソッドは文字の長さ(バイト数ではなく文字数)を返す。

文字列の変換

String#toUpperCase()メソッド:
　文字列を全て大文字に変換する。
String#toLowerCase()メソッド:
　文字列を全て小文字に変換する。

```
String upperCase = str.toUpperCase();
String lowerCase = str.toLowerCase();
```

※メソッドが作用するのは英字のみ

文字列を連結する

StringBuilderクラスを使用すると、
連結前の文字列をメモリ上に残さないため、
メモリに負荷をかけることなく文字列の連結が行える。

（Stringクラスの文字列を+演算子で連結した場合）
連結前の文字列をメモリ上に残したまま、
新たに連結後の値が作成される。
文字列を何度も連結するような処理の場合、
連結前の文字列がメモリ上に溢れ、負荷をかけてしまう。

【Sample1902 文字を連結する】を作成しましょう

Sample1902のポイント

StringBuilderクラスのappend()メソッドを利用して、
文字列を連結する処理を行える。
文字列を引数に渡すことで、
その文字列を連結することができる。

```
builder.append(str2);
```

StringBuilderクラスのメソッド一覧

メソッド名	機能
`StringBuilder append(String str)`	オブジェクトの後ろに文字列を連結する
`StringBuilder insert(int i,String str)`	指定したインデックス番号に文字列を挿入する
`StringBuilder deleteCharAt(int i)`	指定したインデックス番号の文字を削除する

数値を扱うクラス：ラッパークラス

intやdoubleなどの基本型の情報は、
自身の値を操作する機能などは持っていない。

ラッパークラス：
基本型の値を保存し、かつ操作も行えるようにしたクラス。
基本型の値をラップし（包み）、オブジェクトとして利用できる。

（ラッパークラス一覧はテキスト参照）

Integerクラスを使う

Integerクラスは「int型」に関する様々な機能を提供している。

Integer#parseInt()メソッド:
数値に変換可能な文字列をint型の整数値に変換して、
戻り値として返す。

```
String str = “123”;
int number = Integer.parseInt(str);
```

Mathクラスを使う

Mathクラス：

指数関数、対数関数、平方根、および三角関数といった
基本的な数値処理を実行するための機能をまとめたクラス

（メソッド一覧はテキスト参照）

【Sample1903 大きい値を調べる】を作成しましょう

Sample1903のポイント

キーボードから入力した整数をint型の値に変換し、
どちらの値が大きいかを比較している。

int型への変換にはIntegerクラスの
parseInt()メソッドを使用している。

```
int num1 = Integer.parseInt(str1);
int num2 = Integer.parseInt(str2);
```

Sample1903のポイント

Mathクラスのmax()メソッドを使用して値の比較を行っている。

max()メソッドは、引数の値を比較し、
大きい方を戻り値として返す。

```
int result = Math.max(num1, num2);
```

Sample1903のポイント

- IntegerクラスのparseInt()メソッド
- Mathクラスのmax()メソッド

両方ともstaticメソッドのため、
「クラス名.メソッド名」で記述している。

```
int num1 = Integer.parseInt(str1);
int num2 = Integer.parseInt(str2);
```

```
int result = Math.max(num1, num2);
```

日付を扱うクラス:Dateクラスを使う

Dateクラスは日時を扱うクラス。
Dateオブジェクトを引数なしで作成すると、
今日の日時を取得できる。

```
Date date = new Date();
```

SimpleDateFormatクラスを使う

日付を表示させるにはSimpleDateFormatクラスの
オブジェクトを生成し、コンストラクタに書式化文字列を指定する。

```
SimpleDateFormat sdf =
                    new SimpleDateFormat(“yyyy年MM月dd日”);
```

SimpleDateFormatクラスを使う

書式化文字列は
年（yまたはyyyy）、月（M）、日（d）、時（h）、分（m）、秒（s）。
書式化文字列には任意の文字を表示用に
書き込んでおくことができる。
MMやddのように記述することで月や日の桁を2桁で表示する。

（書式化文字列の一覧はテキスト参照）

【Sample1905 日付を表示する】を作成しましょう

入力チェックを行うクラス：文字列の書式を調べる

書式を調べるには正規表現を活用すると便利。

正規表現：文字列に対して複雑な検索や置換処理を行うなど、
様々な文字列のパターンを一つの形式でまとめて
表現する際に用いる。

入力チェックを行うクラス：文字列の書式を調べる

（「123-1234」の郵便番号を検索したい場合）

```
(″[0-9]{3}-[0-9]{4}″);
```

0~9の数字が3回続いた後に「-（ハイフン）」があり、
その次に0~9の数字が4回続いたら郵便番号として
判断して条件が一致したと判定するという意味。

（正規表現の記号一覧はテキスト参照）

正規表現のパターンを作る

正規表現のパターンオブジェクトを作るには…
①Patternクラスのcompile()メソッドの引数に
　正規表現のパターンを指定する。
②Patternクラスのmatcher()メソッドの引数にマッチさせ、
　文字列を指定してMatcherオブジェクトを作成する。

Matcherオブジェクト:
パターンを使ってターゲットの文字列に対して
様々な操作を行うオブジェクト。

正規表現のパターンを作る

（例）
文字列の中から”111-1111”のような郵便番号を検索したい場合
① Patternクラスのcompile()メソッドの引数に
　正規表現のパターンを指定する。

```
Pattern p = Pattern.compile(″[0-9]{3}-[0-9]{4}″);
```

正規表現のパターンに一致するか調べる

②Matcherクラスのfind()メソッドを使って、
文字列が正規表現のパターンに一致するか調べる。
Matcherクラスのfind()メソッドは、文字列の中に
正規表現のパターンが含まれる場合に”true”を返し、
それ以外の場合には”false”を返す。

```
public static void main(String[] args) {
// ①パターンを指定
Pattern p = Pattern.compile("[0-9]{3}-[0-9]{4}");
// ②パターンにマッチしているか比較
    Matcher m = p.matcher(str);
}
```

Sample1906のポイント

下記の流れで各クラスを生成している。

① 正規表現パターンをコンパイルして
Patternオブジェクトを生成。

② 正規表現を適用したいテキストを引数として渡して、
Matcherオブジェクト生成。

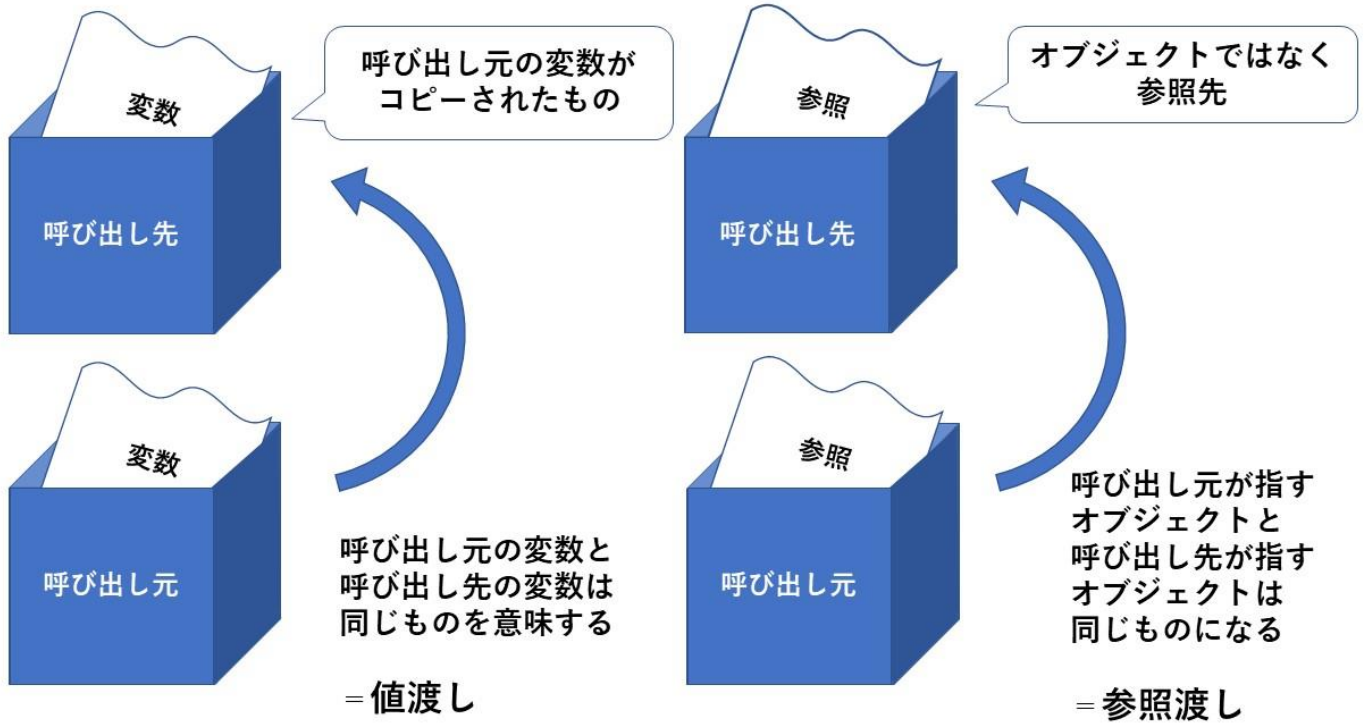
③ Matcherにより、検索・置換・分割などを行う。

クラス型の変数：値渡しと参照渡し

値渡し：
値が渡される呼び出しのこと。
メソッドを呼び出す際の引数に変数を指定した場合、
メソッドに渡されるのは変数ではなく、変数に入っている値が入る。

参照渡し：
参照が渡される呼び出しのこと。
参照型の変数でメソッドに値を渡した場合、引数としてメソッドの
置かれている場所を示す参照が呼び出し先の引数にコピーされる。

クラス型の変数：値渡しと参照渡し



呼び出し元の変数の値を渡すことを値渡しと呼びます。
呼び出し元の変数が指しているオブジェクトの参照を渡すことを参照渡しと呼びます。

クラス型の変数同士の代入

クラス型の変数の値を同じクラス型の変数に代入した場合、
値にはオブジェクトの参照が代入される。

【Sample1907 クラス型の変数に代入】を作成しましょう

Sample1907のポイント

Sample1907クラスでは、Airplane1907クラスの
オブジェクトを参照している変数airplane1の値を、
同じ型の変数airplane2に代入している。

```
Airplane1907 airplane2 = airplane1;
```

Sample1907のポイント

変数airplane2を使用して、
setAirplane()メソッドを呼び出している。
実行結果を見ると、どちらの変数が参照している
オブジェクトも同じ情報であることが分かる。

```
airplane1.showAirplane();
airplane2.showAirplane();
```

```
機体番号：No.2222
定員：50名
機体番号：No.2222
定員：50名
```

Sample1907のポイント

クラス型の変数は参照渡しになっているため、
他の変数に代入した場合、
2つの変数が1つのオブジェクトを参照している状態となる。
※オブジェクト自体が増えるわけではない

nullの仕組みを知る

クラス型の変数にnullという値を代入すると、
その変数は何も参照していない状態となる。
たとえば、下記の変数airplane1は
どのオブジェクトも参照していない状態。

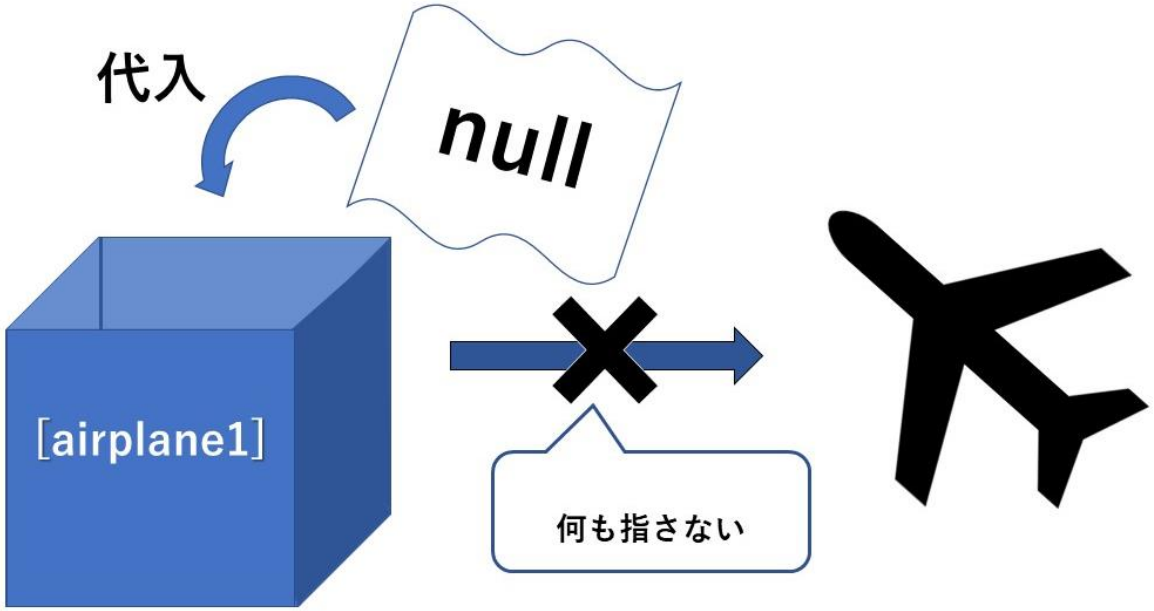
```
airplane1 = null;
```

nullの仕組みを知る

ガーベッジコレクション：
オブジェクトがどの変数からも扱わなくなった場合に、
Javaの判断によりオブジェクトを破棄する仕組み。
（ガーベッジ（Garbage）＝ごみ）

この機能により、メモリ上の不要なデータが占有していた
箇所が解放され、メモリの枯渇を防ぐことができる。

nullの仕組みを知る



参照型の変数にnullを代入すると、
変数がオブジェクトを参照しなくなります 。

章のまとめ

- クラスライブラリのクラスを使うと、
  コードを簡単に作成できます。
- オブジェクトをListで扱うことができます。
- クラス型の変数は参照型の変数になります。
- クラスの型にnullを代入するとその変数は
  オブジェクトを示さなくなります。