23 Java APIプログラミング

23–1 標準ライブラリ，JavaAPI仕様書

標準ライブラリ：

現在最新のJava(J2SE7.0; JDK1.7, Java SE Development Kit 1.7)では、GUI，入出力，ネットワーキング，...等のために合わせて4000 にものぼるクラス(とインタフェース)がJava標準ライブラリとして提供され、必要に応じて自由に利用できる様になっている。

⟶実際の提供情報は、次のWebページ上のAPI仕様書から得られる。

- http://docs.oracle.com/javase/jp/6/api/ ..(J2SE6,日本語)
- http://docs.oracle.com/javase/jp/7/api/ ..(J2SE7,日本語)
- http://docs.oracle.com/javase/7/docs/api/ ..(J2SE7,英語)

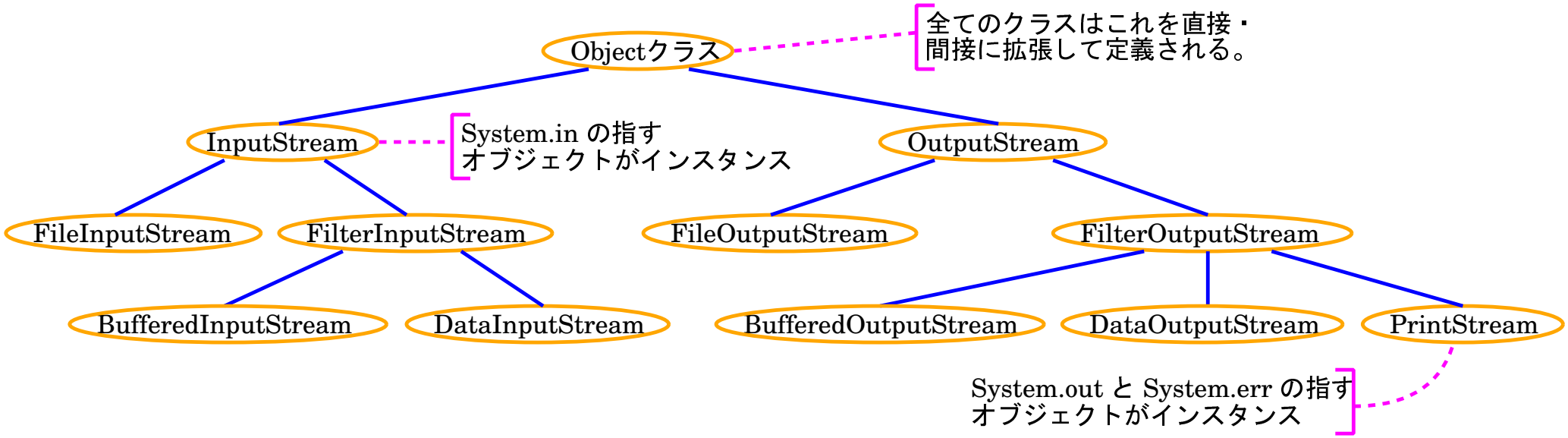
⇨ 標準的に提供されているクラスの利用法については、
まずAPI仕様書を読む。

標準ライブラリの例：

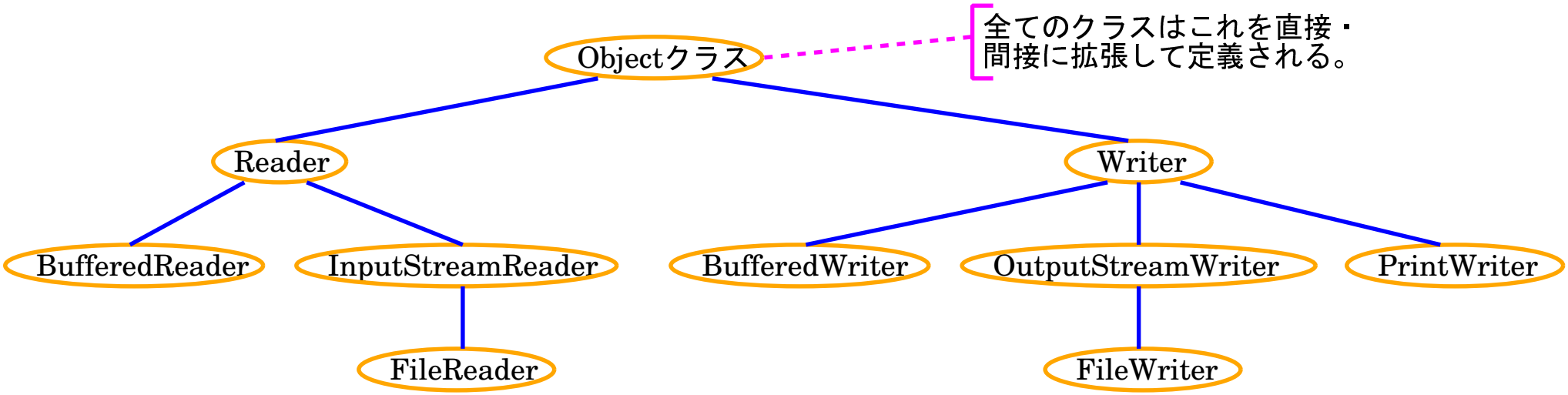
- Stringクラス：（⟶第20.1節）

- ラッパクラス：基本データ型データをカプセル化してオブジェクトとして扱うために用意されたクラス Byte，Short，Integer，Long，Float，Double，Boolean，Character。（⟶第22.8 節）

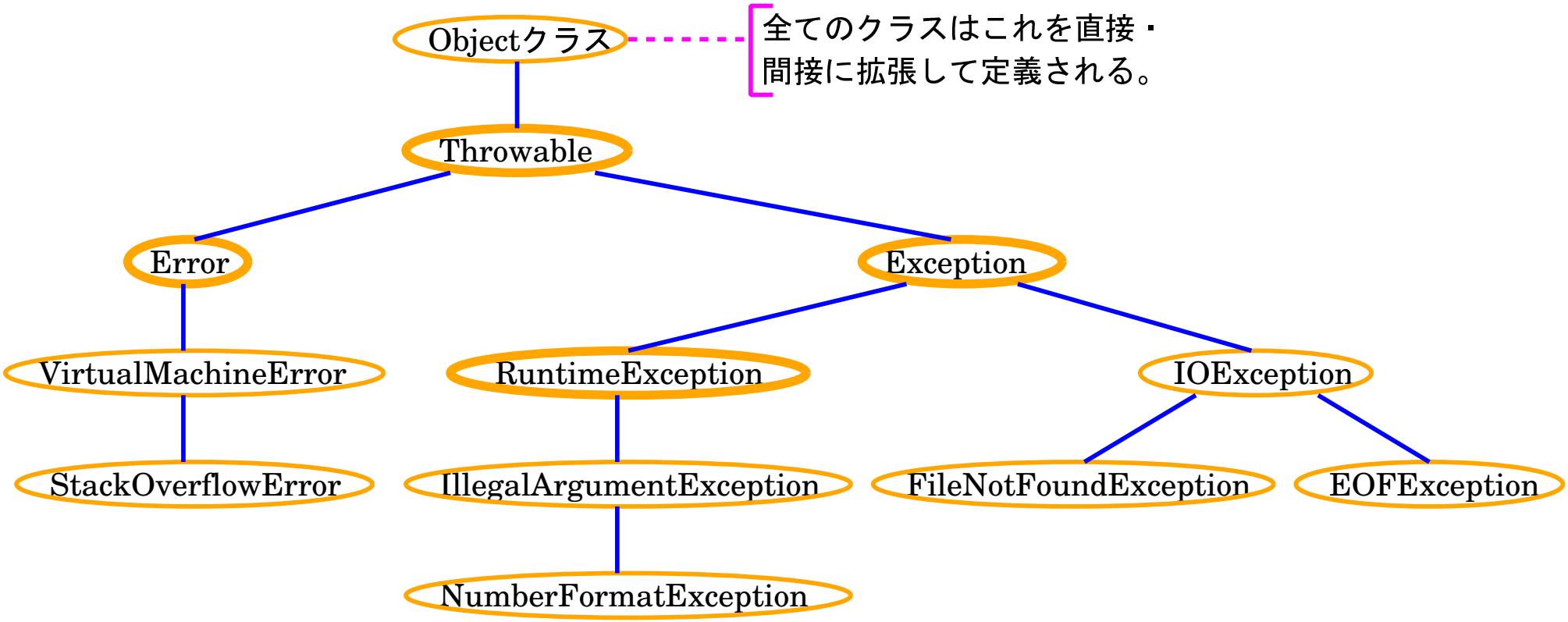
- 入出力用のバイトストリームを扱うためのクラス：java.ioパッケージの中に、入出力用のバイトストリームを扱うためのクラスが多数用意されている。（⟶第23.3節）



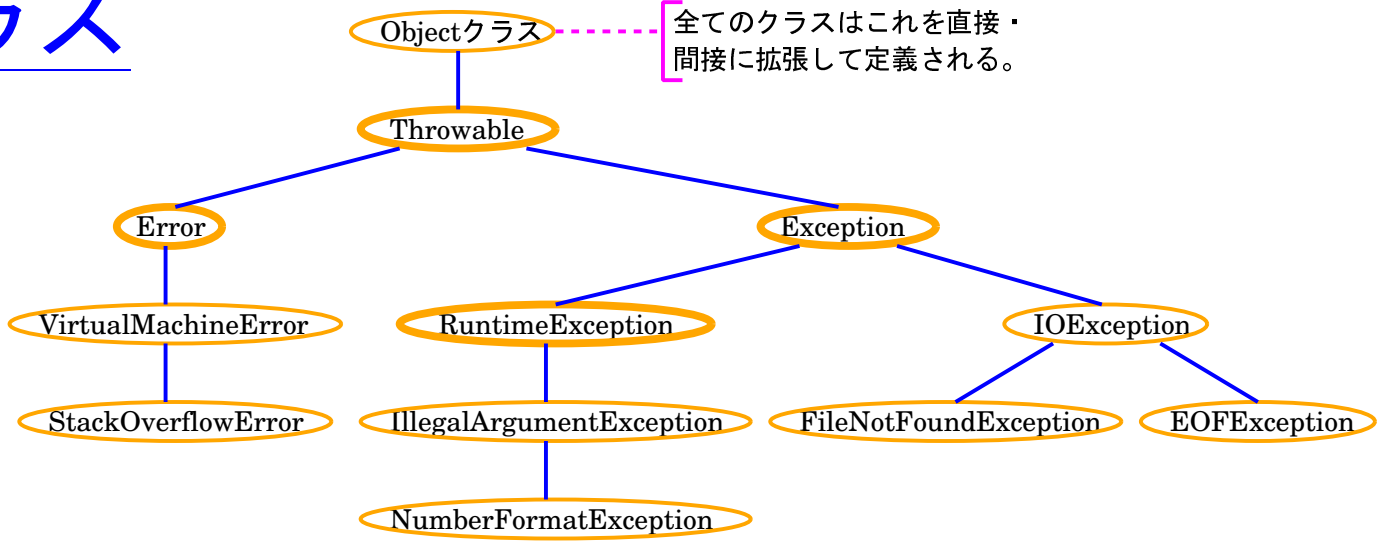
- 入出力用の文字ストリームを扱うためのクラス：java.ioパッケージの中に、入出力用の文字ストリームを扱うためのクラスが多数用意されている。（⟶第23.3節）



- **例外を扱うためのクラス**：コアになる java.lang パッケージの中に例外を扱うための基本となるクラス Throwable, Error, Exception, RuntimeException が用意され、これらを拡張して様々な種類の例外クラスが定義されている。 （⟶第23.2節）



23–2 例外クラス



例外クラスの階層：

- Throwable … 全ての例外の基礎となるクラス
- Error, Exception … Throwableの拡張
- RuntimeException … Exceptionの拡張

このクラス階層によって20.4節で説明した3種類の例外を明確に分類

- Throwable
  - Error … エラー（プログラムの外側の要因で発生）
  - Exception
    - RuntimeException … 実行時例外（プログラムの論理的な誤り等に起因）
    - RuntimeException以外 … チェックされる例外

例23．1（実行時例外発生時の様子）ArrayIndexOutOfBoundsException という実行時例外オブジェクトが発生する時の様子を次に示す。

```
[motoki@x205a]$  cat -n ExampleOfStopByException.java
     1 /**
     2  * 例外発生によって強制終了するプログラムの例
     3  */
     4 public class ExampleOfStopByException {
     5     private static int[] a = {0, 3, 1};
     6
     7     public static void main(String[] args) {
     8         System.out.println("<<処理の開始>>");
     9         int max = getElement(0);
    10         for (int i=1; i<=a.length; ++i) {
    11             System.out.println(
                    "メソッド呼出し getElement(" + i + ");");
    12             int next = getElement(i);
```

```
13              System.out.println(
                   "                                    ---> 成功");
14              if (next > max)
15                  max = next;
16          }
17          System.out.println("max = " + max);
18          System.out.println("<<処理の終了>>");
19      }
20
21      static int getElement(int index) {
22          return a[index];
23      }
24 }
[motoki@x205a]$  javac ExampleOfStopByException.java
[motoki@x205a]$  java ExampleOfStopByException
<<処理の開始>>
メソッド呼出し getElement(1);
```

```
                                    ---> 成功
メソッド呼出し getElement(2);
                                    ---> 成功
メソッド呼出し getElement(3);
Exception in thread "main" java.lang.
                                    ArrayIndexOutOfBoundsException: 3
at ExampleOfStopByException.
                        getElement(ExampleOfStopByException.java:22)
at ExampleOfStopByException.
                              main(ExampleOfStopByException.java:12)
[motoki@x205a]$
```

スタックトレース情報の出力：

Throwableクラスでは次の様なインスタンスメソッドが定義

メソッド名 ··· 説明
void printStackTrace() ··· 例外オブジェクト、および例外発生に至った時点でのスタックトレース情報（i.e. 連鎖的なメソッド呼出し等の情報）を標準エラーストリームに出力する。
StackTraceElement[] getStackTrace() ··· スタックトレース情報にプログラムからアクセスできる様にする。
String toString() ··· 例外オブジェクトの短い記述を返す（Objectクラスからの継承）
..... （他のメソッドについては省略） .....

⇨ これらのメソッドは、全ての例外クラスに（暗黙に）継承される

例23. 2 (printStackTrace()メソッド)

printStackTrace()メソッド等を用いて先の例23.1で示したプログラムに例外処理を加えると、...

発生した例外をプログラム内で処理
$\longrightarrow$ 強制終了は無くなる。

```
[motoki@x205a]$  cat -n ExampleOf_printStackTrace.java
     1 /**
     2  * printStackTrace()メソッドの使用例を示すためのプログラ...
     3  */
     4 public class ExampleOf_printStackTrace {
     5     private static int[] a = {0, 5, 1};
     6
     7     public static void main(String[] args) {
     8         int i=-1000, next=-1000, max=-1000;宣言場所移動
     9         System.out.println("<<処理の開始>>");
    10         try {
```

```
11              max = getElement(0);
12              for (i=1; i<=a.length; ++i) {
13                  System.out.println(
                 "メソッド呼出し getElement(" + i + ");");
14                  next = getElement(i);
15                  System.out.println(
                   "                                  ---> 成功");
16                  if (next > max)
17                      max = next;
18              }
19              System.out.println("max = " + max);
20          } catch (ArrayIndexOutOfBoundsException e) {
21              System.err.println(
                   "                                  ---> 失敗");
22              System.err.println("[例外発生] " + e);
23              System.err.println(
                 "       |--->[この時点の計算状況] i=" + i
```

```
24                                          + ", next=" + next + ",
25                  e.printStackTrace();
26              }
27              System.out.println("<<処理の終了>>");
28          }
29
30          static int getElement(int index) {
31              return a[index];
32          }
33  }
[motoki@x205a]$  javac ExampleOf_printStackTrace.java
[motoki@x205a]$  java ExampleOf_printStackTrace
<<処理の開始>>
メソッド呼出し getElement(1);
                                ---> 成功
メソッド呼出し getElement(2);
                                ---> 成功
```

メソッド呼出し getElement(3);

---> 失敗

[例外発生] java.lang.ArrayIndexOutOfBoundsException: 3

|--->[この時点の計算状況] i=3, next=1, max=5

java.lang.ArrayIndexOutOfBoundsException: 3printStackTrace()の

at ExampleOf_printStackTrace.getElement(

ExampleOf_printStackTrace.java:31)

at ExampleOf_printStackTrace.main(

ExampleOf_printStackTrace.java:14)

<<処理の終了>>

[motoki@x205a]$

例外オブジェクトを使ってエラー処理をすることの利点：

- 本来の処理の流れの明瞭さを損なわずにエラー検出等を行える。

- 記述忘れもなく確実にエラー処理を行うことができる。
  実際、 特別な記述をしなくても 、
  メソッド呼び出しスタック中でエラーの通知が自動的に伝搬される。

  一方、 メソッドの返り値を使ってエラー処理を行う方法
  （e.g.−1が返ってきたらエラーとして処理する方法） の場合は 、

  ◇ メソッド呼出し元が常にメソッドの返り値を意識する必要がある。

  ◇ メソッドの返り値をチェックし忘れることもある。

  ◇ エラーの詳細情報を通知しにくい。

- 例外のクラス階層に応じて色々な粒度で例外をグループ分けして、
  グループ毎にエラー処理を記述できる。

23–3 ほぼ自習 ファイル入出力，ファイル操作

バイトストリームを扱うためのクラス：

入出力用のバイトストリーム
（i.e. データの種類を意識しない1バイト単位のデータ列）
を扱うためのクラスとして、次の様なものが用意されている。

入力用

- java.io.InputStream ··· 入力用のバイトストリームを扱うための Objectの直系の抽象サブクラスで、次の様なメソッドを持つ。
（当然、この抽象クラスの子孫クラスではこれらのメソッドが継承／拡張されることになる。）

メソッド ヘッダー ··· 説明
public abstract int read() throws IOException ··· 入力ストリームから次の1バイトを読み込み、対応する0~ 255の範囲のint値を返す。入力ストリームが終了し次の1バイトがない場合は -1 を返す。
public int read(byte[] b, int off, int len) throws IOException ··· 入力ストリームから最大lenバイトを読み込み、バッファ b[off], b[off+1], b[off+2], ... に順に格納する。そして、実際に読み込まれたバイト数を返す。但し、lenが0の場合は 0 を返し、それ以外の場合は少なくとも1バイトを読み込もうとする。入力ストリームが終了していた場合は -1 を返す。
public int read(byte[] b) throws IOException ··· read(b, 0, b.length) と同じ。
public void close() throws IOException ··· 入力ストリームを閉じる。
.....(他のメソッドについては省略).....

- java.io.FileInputStream ··· ファイルからバイト単位でデータを読み込むためのクラスで、InputStreamの直系のサブクラスになる。次の様なコンストラクタを持つ。

コンストラクタの形 ··· 説明
public FileInputStream(String *name*) throws FileNotFoundException ··· パス名*name*で指定されたファイルからのデータ列を入力用バイトストリームとして扱うためのオブジェクトを生成。
.....（他のコンストラクタについては省略）.....

- java.io.BufferedInputStream ··· 1バイトずつOSに入力依頼を出すと処理効率が悪くなる。これを避けるためのバッファ付き入力用バイトストリームを扱うためのクラスで、InputStreamの子孫クラスになる。バッファ無しの入力用バイトストリームを（コンストラクタの引数として与えて） バッファ付きのものに変換 することによって、このクラスのオブジェクトを生成する。

出力用

- java.io.OutputStream ··· 出力用のバイトストリームを扱うための Objectの直系の抽象サブクラスで、次の様なメソッドを持つ。

メソッド ヘッダー ··· 説明
public abstract void write(int *b*) throws IOException ··· 引数で指定したint値(の下位8ビット)を出力ストリームに書き出す。
public void write(byte[] *b*, int *off*, int *len*) throws IOException ··· *b*[*off*], *b*[*off*+1], *b*[*off*+2], ..., b[*off*+*len*-1] を出力ストリームに書き出す。
public void write(byte[] *b*) throws IOException ··· write(*b*, 0, *b*.length) と同じ。
public void close() throws IOException ··· 出力ストリームを閉じる。
.....(他のメソッドについては省略) .....

- java.io.FileOutputStream ··· ファイルにバイト単位でデータを書き出すためのクラスで、OutputStreamの直系のサブクラスになる。次の様なコンストラクタを持つ。

コンストラクタの形 ··· 説明
public FileOutputStream(String *name*) throws FileNotFoundException ··· パス名 *name* で指定されたファイルへのデータ列を出力用バイトストリームとして扱うためのオブジェクトを生成。
.....（他のコンストラクタについては省略）.....

- java.io.BufferedOutputStream ··· 1バイトずつOSに出力依頼を出すと処理効率が悪くなる。これを避けるためのバッファ付き出力用バイトストリームを扱うためのクラスで、OutputStreamの子孫クラスになる。バッファ無しの出力用バイトストリームを（コンストラクタの引数として与えて） バッファ付きのものに変換 することによって、このクラスのオブジェクトを生成する。

例23．3（ファイルを扱うプログラムの基本構成；1バイトずつ読み読み込

コマンドラインで指定したファイルから1バイトずつ読み込み、それを順にコマンドラインで指定した別ファイルと標準出力に書き出すJavaプログラムを表示し、これをコンパイル・実行している様子を次に示す。

```
[motoki@x205a]$  cat -n CopyAndShowBytesInFile.java
     1 import java.io.FileInputStream;
     2 import java.io.FileOutputStream;
     3 import java.io.FileNotFoundException;
     4 import java.io.IOException;
     5
     6 /**
     7  * コマンドラインで指定したファイルから1バイトずつ読み込...
     8  * コマンドラインで指定した別ファイルと標準出力に書き出す...
     9  */
    10 public class CopyAndShowBytesInFile {
    11     public static void main(String[] args) {
    12         FileInputStream in = null;
```

```
13          FileOutputStream out = null;
14
15          try {
16              in = new FileInputStream(args[0]);
17              out = new FileOutputStream(args[1]);
18              int c;
19              while ((c = in.read()) != -1) {
20                  System.out.printf("%#X(%c) ",
                                              c, (char)c);
21                  out.write(c);
22              }
23              System.out.println();
24          } catch(FileNotFoundException e) {
25              System.err.println(
                        "入力ファイルは存在してますか？" +
26                      "ディレクトリを指定してませんか？");
27              e.printStackTrace();
```

```
28          } catch(IOException e) {
29              System.err.println(
                    "ファイルの読み書き中にエラー発生。");
30              e.printStackTrace();
31          } finally {
32              if (in != null) {
33                  try {
34                      in.close();
35                  } catch(IOException e) {
36                      e.printStackTrace();
37                  }
38              }
39              if (out != null) {
40                  try {
41                      out.close();
42                  } catch(IOException e) {
43                      e.printStackTrace();
```

```
    44                  }
    45              }
    46          }
    47      }
    48 }
[motoki@x205a]$  javac CopyAndShowBytesInFile.java
[motoki@x205a]$  cat ascii.txt
abc
[motoki@x205a]$  java CopyAndShowBytesInFile ascii.txt out
0X61(a) 0X62(b) 0X63(c) 0XA(
)
[motoki@x205a]$  cat out
abc
[motoki@x205a]$  cat utf8.txt
情報
[motoki@x205a]$  java CopyAndShowBytesInFile utf8.txt out
0XE6(^^c3^^a6) 0X83() 0X85() 0XE5(^^c3^^a5) 0XA0() 0XB1(±) 0X
```

修正

)

```
[motoki@x205a]$  cat out
```

情報

```
[motoki@x205a]$  ls abc
ls: abc にアクセスできません: そのようなファイルやディレクトリは
```

ありません

```
[motoki@x205a]$  java CopyAndShowBytesInFile abc out
```

入力ファイルは存在してますか？ディレクトリを指定してませんか？

```
java.io.FileNotFoundException: abc (No such file or directory)
at java.io.FileInputStream.open(Native Method)
at java.io.FileInputStream.<init>(FileInputStream.java:137)
at java.io.FileInputStream.<init>(FileInputStream.java:96)
at CopyAndShowBytesInFile.main(CopyAndShowBytesInFile.java:16)
[motoki@x205a]$
```

文字ストリームを扱うためのクラス：

入出力用の文字ストリーム(i.e.文字単位のデータ列)
を扱うためのクラスとして、次の様なものが用意されている。

入力用

- java.io.Reader ··· 入力用の文字ストリームを扱うためのObjectの直系の抽象サブクラスで、次の様なメソッドを持つ。(当然、この抽象クラスの子孫クラスではこれらのメソッドが継承／拡張されることになる。)

メソッド ヘッダー … 説明
public int read() throws IOException … 入力ストリームから次の 1 文字を読み込み、対応する 0~65535($2^{16}-1$)の範囲のint値を返す。入力ストリームが終了し次の1バイトがない場合は -1 を返す。
public abstract int read(char[] c, int off, int len) throws IOException … 入力ストリームから最大len文字を読み込み、バッファ c[off], c[off+1], c[off+2], … に順に格納する。そして、実際に読み込まれた文字数を返す。但し、lenが0の場合は 0 を返し、それ以外の場合は少なくとも1文字を読み込もうとする。入力ストリームが終了していた場合は -1 を返す。
public int read(char[] c) throws IOException … read(c, 0, c.length) と同じ。
public abstract void close() throws IOException … 入力ストリームを閉じる。
……(他のメソッドについては省略)……

- java.io.FileReader ··· デフォルトの文字エンコーディングに従って構成されたテキストファイルから文字単位でデータを読み込むための簡易クラスで、Readerの子孫クラスになる。次の様なコンストラクタを持つ。

コンストラクタの形 ··· 説明
public FileReader(String *name*) throws FileNotFoundException ··· パス名 *name* で指定されたファイルからのデータ列を入力用文字ストリームとして扱うためのオブジェクトを生成。
..... (他のコンストラクタについては省略) .....

- java.io.InputStreamReader ··· 指定された文字エンコーディングに従って構成されたバイトストリームから文字単位でデータを読み込むためのクラスで、Readerの直系のサブクラスになる。デフォルト以外の文字エンコーディングに従って構成されたテキストファイルから文字単位でデータを読み込むためには、一旦FileInputStreamインスタンスを構成し、これと文字エンコーディング方式を表す文字列(e.g."UTF8", "UTF-16", "ISO-2022-JP", "EUC_JP", "SJIS", "Shift_JIS", "MS932")をコンストラクタの引数として与えて、このクラスのオブジェクトを生成することになる。

- java.io.BufferedReader ··· 1文字ずつOSに入力依頼を出すと処理効率が悪くなる。これを避けるためのバッファ付き入力用文字ストリームを扱うためのクラスで、Readerの直系のサブクラスになる。次の様なコンストラクタ、メソッドを持つ。

コンストラクタ／メソッド ヘッダー ... 説明
public BufferedReader(Reader *in*) ... 引数 *in* で指定された文字入力ストリームを基にバッファ付きの文字入力ストリームのオブジェクトを生成。
public BufferedReader(Reader *in*, int *size*) ... 引数 *in* で指定された文字入力ストリームを基に、大きさ *size* のバッファを備えた文字入力ストリームオブジェクトを生成。
public String readLine() throws IOException ... 入力ストリームから次の 1 行分 (i.e. 次の改行 ('\n'), 復帰 ('\r'), または改行復帰 ("\n\r") まで) の文字列を読み込んで返す。但し、終端文字は含めない。入力ストリームが終了していた場合は null を返す。
.....(他メソッドについては省略).....

出力用

- java.io.Writer ··· 出力用の文字ストリームを扱うためのObjectの直系の抽象サブクラスで、次の様なメソッドを持つ。（当然、この抽象クラスの子孫クラスではこれらのメソッドが継承／拡張されることになる。）

メソッド ヘッダー … 説明
public void write(int *c*) throws IOException … 引数で指定したint値(の下位16ビット)で表される文字を出力ストリームに書き出す。
public abstract void write(char[] *c*, int *off*, int *len*) throws IOException … *c*[*off*], *c*[*off*+1], *c*[*off*+2], ..., c[*off*+*len*−1] を出力ストリームに書き出す。
public void write(char[] *c*) throws IOException … write(*c*, 0, *c*.length) と同じ。
public void write(String *str*) throws IOException … 文字列*str*を出力ストリームに書き出す。
public abstract void close() throws IOException … 出力ストリームを閉じる。
.....(他のメソッドについては省略).....

- java.io.FileWriter ··· デフォルトの文字エンコーディングに従うテキストファイルに文字単位でデータを書き出すための簡易クラスで、Writerの子孫クラスになる。次の様なコンストラクタを持つ。

コンストラクタの形 ··· 説明
public FileWriter(String *name*) throws IOException ··· パス名 *name* で指定されたファイルへのデータ列を出力用文字ストリームとして扱うためのオブジェクトを生成。
..... (他のコンストラクタについては省略) .....

- java.io.OutputStreamWriter ··· 文字単位のデータを指定された文字エンコーディングに従って構成してバイトストリームに書き出すためのクラスで、Writerの直系のサブクラスになる。デフォルト以外の文字エンコーディングに従った文字データを書き出してテキストファイルを構成するためには、一旦FileOutputStreamインスタンスを構成し、これと文字エンコーディング方式を表す文字列(e.g."UTF "UTF-16", "ISO-2022-JP", "EUC_JP", "SJIS", "Shift_JIS", "M をコンストラクタの引数として与えて、このクラスのオブジェクト

を生成することになる。

- java.io.PrintWriter ··· オブジェクトの書式付き表現を出力用の文字ストリームに書き出すためのクラスで、Writerの直系のサブクラスになる。例えば、行単位の出力も行なってテキストファイルを構成したい場合は、一旦FileWriterインスタンスを構成し、これをコンストラクタの引数として与えて、このクラスのオブジェクトを生成することになる。次の様なコンストラクタ、メソッドを持つ。

コンストラクタ／メソッド ヘッダー ··· 説明
public PrintWriter(Writer *out*) ··· 引数 *out* で指定された文字出力ストリームを基に、行の自動フラッシュ無しのPrintWriterオブジェクトを生成。
public PrintWriter(String *name*) throws FileNotFoundExceptior ··· パス名 *name* で指定されたファイルへの文字出力を行うための、行の自動フラッシュ無しのPrintWriterオブジェクトを生成。
public void print(...) ··· (説明省略)
public void println(...) ··· (説明省略)
public PrintWriter printf(String *format*, Object... *args*) ··· (説明省略)
·····(他コンストラクタ／メソッドについては省略)·····

- java.io.BufferedWriter ··· 1文字ずつOSに出力依頼を出すと処理効率が悪くなる。これを避けるためのバッファ付き出力用文字ストリームを扱うためのクラスで、Writerの直系のサブクラスになる。バッファ無しの出力用文字ストリームを（コンストラクタの引数として与えて） バッファ付きのものに変換 することによって、このクラスのオブジェクトを生成する。

例23．4（テキストファイルを扱うプログラムの基本構成；1文字ずつ処理

コマンドラインで指定したファイルから1文字ずつ読み込み、それを順にコマンドラインで指定した別ファイルと標準出力に書き出すJavaプログラムを表示し、これをコンパイル・実行している様子を次に示す。
(例23.3で示したプログラムとの本質的な違いは 、

- FileInputStream の代わりに FileReaderを、
- FileOutputStream の代わりに FileWriterを

用いた点だけである。)

```
[motoki@x205a]$ cat -n CopyAndShowCharsInFile.java
     1 import java.io.FileReader;
     2 import java.io.FileWriter;
     3 import java.io.FileNotFoundException;
     4 import java.io.IOException;
     5
     6 /**
     7  * コマンドラインで指定したファイルから1文字ずつ読み込み...
```

```
 8  * コマンドラインで指定した別ファイルと標準出力に書き出す...
 9  */
10 public class CopyAndShowCharsInFile {
11     public static void main(String[] args) {
12         FileReader reader = null;
13         FileWriter writer = null;
14
15         try {
16             reader = new FileReader(args[0]);
17             writer = new FileWriter(args[1]);
18             int c;
19             while ((c = reader.read()) != -1) {
20                 System.out.printf("%#X(%c) ",
                                              c, (char)c);
21                 writer.write(c);
22             }
23             System.out.println();
```

```
24          } catch(FileNotFoundException e) {
25              System.err.println(
                        "入力ファイルは存在してますか？" +
26                      "ディレクトリを指定してませんか？");
27              e.printStackTrace();
28          } catch(IOException e) {
29              System.err.println(
                      "ファイルの読み書き中にエラー発生。");
30              e.printStackTrace();
31          } finally {
32              if (reader != null) {
33                  try {
34                      reader.close();
35                  } catch(IOException e) {
36                      e.printStackTrace();
37                  }
38              }
```

```
39                 if (writer != null) {
40                     try {
41                         writer.close();
42                     } catch(IOException e) {
43                         e.printStackTrace();
44                     }
45                 }
46             }
47         }
48 }
[motoki@x205a]$  javac CopyAndShowCharsInFile.java
[motoki@x205a]$  cat ascii.txt
abc
[motoki@x205a]$  java CopyAndShowCharsInFile ascii.txt out
0X61(a) 0X62(b) 0X63(c) 0XA(
)
[motoki@x205a]$  cat out
```

```
abc
[motoki@x205a]$  cat utf8.txt
情報
[motoki@x205a]$  java CopyAndShowCharsInFile utf8.txt out
0X60C5(情) 0X5831(報) 0XA(
)
[motoki@x205a]$  cat out
情報
[motoki@x205a]$  file euc.txt
euc.txt: ISO-8859 text
[motoki@x205a]$  nkf -w euc.txt
情報
[motoki@x205a]$  java CopyAndShowCharsInFile euc.txt out
0XFFFD(^^ef^^bf^^bd) 0XFFFD(^^ef^^bf^^bd) 0XFFFD(^^ef^^bf^^bd)
```

文字エンコーディング方式がデフォルト(UTF8)と違うため

```
[motoki@x205a]$  nkf -w out
鐃緒申鐃緒申[motoki@x205a]$
```

ここで、

- 入力ファイルを euc.txt としたプログラム実行

  ... 入力ファイル euc.txt の文字エンコーディング方式がコンピュータのデフォルトのもの(上の実行環境では"UTF8")と違っているため

  デフォルト以外の文字エンコーディング方式を使ってファイル入出力を行いたい場合 は、

  InputStreamReaderを使ってFileInputStreamインスタンスを入力用文字ストリームに変換させたり、OutputStreamWriterを使ってFileoutputStreamインスタンスを出力用文字ストリームに変換させたりする。

  上のプログラムの場合は、

  12~ 13行目...型を InputStreamReader, OutputStreamWriter に

  16~ 17行目...次の様に書き換え

```
reader = new InputStreamReader(
                    new FileInputStream(args[0]), "EUC_JP");
writer = new OutputStreamWriter(
                  new FileOutputStream(args[1]), "EUC_JP");
```

- ファイルへのアクセスはOSへの処理依頼を伴い、通常の計算処理と比べて非常に時間のかかる動作で、1文字ずつの入出力動作を繰り返していたのでは、プログラムの処理効率が非常に低くなる可能性がある。

  ⇨ 文字ストリームをバッファ付き文字ストリームに変換するためのクラス

  BufferedReader, BufferedWriter

  が用意され、その利用が推奨されている。

これを上のプログラムでも利用する場合 は、

12~ 13行目...型をBufferedReader, BufferedWriter に変更

16~ 17行目...次の様に書き換え

```
reader = new BufferedReader(new FileReader(args[0]));
writer = new BufferedWriter(new FileWriter(args[1]));
```

⟶ BufferedReaderインスタンスのreadLine()メソッドを用いて、1行分の文字列をまとめて読み込むことができる様になる。

ファイル操作，ディレクトリ操作のためのクラス：

- java.io.File ··· ファイルやディレクトリのパス名をシステムに依存しない抽象的な形式で保持するためのObjectの直系のサブクラスで、次の様なコンストラクタ、メソッドを持つ。

コンストラクタ／メソッド ヘッダー ··· 説明
public File(String *pathname*) ··· 引数*pathname*で指定されたパス名文字列を内部の抽象的な形式のパス名（抽象パス名）に変換して、それを保持するオブジェクトを生成。*pathname*が空文字列の場合は、空の抽象パス名を持ったオブジェクトが生成される。
public boolean exists() ··· 抽象パス名のファイルまたはディレクトリが実際に存在するかどうかを判定し、その結果を返す。
public boolean isDirectory() ··· 抽象パス名がディレクトリ名として実際に存在するかどうかを判定し、その結果を返す。

public boolean isFile() … 抽象パス名が通常ファイル名として実際に存在するかどうかを判定し、その結果を返す。
public boolean delete() … 抽象パス名が示すファイルまたはディレクトリの削除を試み、それが成功すればtrue、失敗すればfalseを返す。
public boolean delete() … 抽象パス名が示すファイルまたはディレクトリの削除を試み、それが成功すればtrue、失敗すればfalseを返す。
public String[] list() … 抽象パス名が示すディレクトリ内のファイルやディレクトリを表す文字列を要素とする配列を返す。
……（他コンストラクタ／メソッドについては省略）……

例23．5（ディレクトリ内のファイル一覧）

コマンドラインで指定したディレクトリ内のファイルやディレクトリの名前を列挙するJavaプログラムを表示し、これをコンパイル・実行している様子を次に示す。

```
[motoki@x205a]$  cat -n ListFilesInDirectory.java
     1 import java.io.File;
     2
     3 /**
     4  * コマンドラインで指定したディレクトリ内の
     5  * ファイルやディレクトリの名前を列挙するJavaプログラム
     6  */
     7 public class ListFilesInDirectory {
     8     public static void main(String[] args) {
     9         File dir = new File(args[0]);
    10         String[] listOfFiles = dir.list();
    11         for (int i=0; i<listOfFiles.length; ++i) {
    12             System.out.println(listOfFiles[i]);
```

```
    13          }
    14      }
    15 }
[motoki@x205a]$  javac ListFilesInDirectory.java
[motoki@x205a]$  java ListFilesInDirectory .
euc.txt
CopyAndShowBytesInFile.java
ListFilesInDirectory.java
ExampleOf_printStackTrace.java
utf8.txt
ascii.txt
CopyAndShowCharsInFile.java
ListFilesInDirectory.class
CopyAndShowCharsInFileAnyCoding.java
ExampleOfStopByException.java
[motoki@x205a]$  ls
CopyAndShowBytesInFile.java          ExampleOf_printStackTrac
```

```
CopyAndShowCharsInFile.java              ListFilesInDirectory.cla
CopyAndShowCharsInFileAnyCoding.java     ListFilesInDirectory.jav
ExampleOfStopByException.java            ascii.txt
[motoki@x205a]$
```

23–4 ほぼ自習 実行環境とのインタフェース

実行環境とのインタフェースを確保するために、次の様なクラスが用意されている。

- java.lang.System ··· コンピュータシステム全体に関わるユーティリティ機能を提供するための、Objectの直系のサブクラスで、インスタンス化できない。次の様なフィールド、メソッドを持つ。

フィールド／メソッド ヘッダー ··· 説明
public static final InputStream in ··· 標準入力ストリーム
public static final PrintStream out ··· 標準出力ストリーム
public static final PrintStream err ··· 標準エラー出力ストリーム

public static Properties getProperties() 　… 現在のシステムプロパティの情報を取得する。
public static String getenv(String *name*) 　… 指定された環境変数の値を取得する。
public static long currentTimeMillis() 　… 1970年1月1日午前0時からの経過時間をミリ秒単位で返す。
public static long nanoTime() 　… システム内の固定された時刻からの経過時間（ナノ秒）を返す。（ナノ秒の精度は保証されない。）
public static void exit(int *status*) 　… 実行中のJava仮想機械を状態コード*status*（0以外は異常終了を表す）で終了する。
public static void gc() 　… ガベージコレクタを呼び出す。（ガベージコレクションを行う努力をすることをJava仮想機械に要求。）

public static void arraycopy(Object *src*, int *srcPos*, Object *dest*, int *destPos*, int *length*)
··· 配列要素をコピー。(参照情報をコピーするだけの浅いコピー。)
..... (他のメソッドについては省略) .....

- java.lang.Runtime ··· Java仮想機械上でプログラムを実行している時の環境とのインタフェースを提供するためのObjectの直系のサブクラスである。 プログラムの中から新たなインスタンスを生成することはできないが、現在の実行環境をオブジェクト化したRuntimeオブジェクトへの参照はgetRuntimeメソッドにより取得できる。次の様なメソッドを持つ。

メソッド ヘッダー ··· 説明
public static Runtime getRuntime() ··· 現在実行中のJavaプログラムに関連したRuntimeオブジェクト(への参照)を返す。

public void exit(int *status*) ··· 実行中のJava仮想機械を状態コード*status*(0以外は異常終了を表す)で終了する。
public Process exec(String *command*) throws IOException ··· 指定された文字列コマンドを独立したプロセスで実行する。(これでemacsやfirefox等を起動することも可。)
int availableProcessors() ··· Java仮想機械が使用できるプロセッサの数を返す。

long freeMemory() ··· Java仮想機械内のメモリの空き容量(見積もり)をbyte単位で返す。
long totalMemory() ··· Java仮想機械内のメモリの総容量をbyte単位で返す。
long maxMemory() ··· Java仮想機械が使用するメモリの最大容量をbyte単位で返す。
public void gc() ··· ガベージコレクタを呼び出す。(ガベージコレクションを行う努力をすることをJava仮想機械に要求。)
.....(他のメソッドについては省略) .....

- java.lang.Process ··· Runtime.getRuntime().exec()等によって生成された(サブ)プロセスを管理するための、Objectの直系のサブクラスである。(サブプロセスは仮想端末上で実行される訳でないので、場合によってはサブプロセスの標準出力結果を受け取るた

めのストリームを設定する等の処置も必要になる。） 次の様なメソッドを持つ。

メソッド ヘッダー … 説明
public abstract OutputStream getOutputStream() … サブプロセスの標準入力に自プロセスの出力データ列を書き出すためのOutputStreamを取得する。
public abstract InputStream getInputStream() … サブプロセスの標準出力を自プロセスへの入力データ列として読み込むためのInputStreamを取得する。
public abstract InputStream getErrorStream() … サブプロセスの標準エラー出力を自プロセスへの入力データ列として読み込むためのInputStreamを取得する。
public abstract int exitValue() … サブプロセスの終了コードを返す。
public abstract void destroy() … サブプロセスを終了させる。
……(他のメソッドについては省略)……

例23．6（ガベージコレクション）オブジェクトの生成を繰返すのに伴ってヒープ領域の残メモリ量がどう変わっていくかを観察するJavaプログラムを表示し、これをコンパイル・実行している様子を次に示す。

```
[motoki@x205a]$  cat -n ObserveFreeMemory.java
     1 /**
     2  * オブジェクトの生成を繰返すに伴ってヒープ領域の残メモリ
     3  * 量がどう変わっていくかを観察するJavaプログラム
     4  */
     5 public class ObserveFreeMemory {
     6     public static void main(String[] args) {
     7         System.out.printf(
                         "FreeMemory=%9d, totalMemory=%9d, maxMe
     8                 Runtime.getRuntime().freeMemory(),
     9                 Runtime.getRuntime().totalMemory(),
    10                 Runtime.getRuntime().maxMemory());
```

```
11          for (int k=1; k<=30; ++k) {
12              int[] a = new int[100000*k];
13              for (int i=0; i<100000*k; ++i) {
14                  a[i] = i;
15              }
16              System.out.printf(
                    "k=%2d ==> FreeMemory=%9d%n",
17                  k, Runtime.getRuntime().freeMemory());
18          }
19          System.out.printf(
                    "FreeMemory=%9d, totalMemory=%9d, maxMe
20                  Runtime.getRuntime().freeMemory(),
21                  Runtime.getRuntime().totalMemory(),
22                  Runtime.getRuntime().maxMemory());
23      }
24  }
```

メモリ消費

```
[motoki@x205a]$  javac ObserveFreeMemory.java
[motoki@x205a]$  java ObserveFreeMemory
FreeMemory= 55218000, totalMemory= 55508992, maxMemory=8239841
k= 1 ==> FreeMemory= 54526752
k= 2 ==> FreeMemory= 53726736
k= 3 ==> FreeMemory= 52526720
k= 4 ==> FreeMemory= 50926704
k= 5 ==> FreeMemory= 48926688
k= 6 ==> FreeMemory= 46526672
k= 7 ==> FreeMemory= 43726656
k= 8 ==> FreeMemory= 51643936
k= 9 ==> FreeMemory= 48043920
k=10 ==> FreeMemory= 44043904
k=11 ==> FreeMemory= 65251824
k=12 ==> FreeMemory= 60451808
k=13 ==> FreeMemory= 55251792
k=14 ==> FreeMemory= 49651776
```

ガベージコレクション

ガベージコレクション＋メモリ増量

```
k=15 ==> FreeMemory= 43651760
k=16 ==> FreeMemory= 63320696
k=17 ==> FreeMemory= 56520680
k=18 ==> FreeMemory= 49320664
k=19 ==> FreeMemory= 41720648
k=20 ==> FreeMemory= 90777048
k=21 ==> FreeMemory= 82377032
k=22 ==> FreeMemory= 73577016
k=23 ==> FreeMemory= 64377000
k=24 ==> FreeMemory= 54776984
k=25 ==> FreeMemory= 44776968
k=26 ==> FreeMemory= 88436640
k=27 ==> FreeMemory= 77636624
k=28 ==> FreeMemory= 66436608
k=29 ==> FreeMemory= 54836592
k=30 ==> FreeMemory= 42836576
FreeMemory= 42836576, totalMemory= 99155968, maxMemory=8239841
```

ガベージコレクション

ガベージコレクション＋メモリ増量？

ガベージコレクション

```
[motoki@x205a]$
```

ここで、

- プログラム中にガベージコレクションを行なってもらいたい場合 は、その場所に

  `System.gc();` または `Runtime.getRuntime().gc();`

  というメソッド呼び出しを埋め込めば良い。

  ただ、 あまりにも頻繁にガベージコレクションを行うと

  $\longrightarrow$ 処理効率の低下にも繋がる

例23．7（OSのコマンド実行）

プログラムの中からLinuxのlsコマンドを実行するJavaプログラムを表示し、これをコンパイル・実行している様子を次に示す。

```
[motoki@x205a]$  cat -n Exec_lsCommand.java
     1 import java.io.IOException;
     2 import java.io.InputStream;
     3 import java.io.InputStreamReader;
     4 import java.io.BufferedReader;
     5
     6 /**
     7  * プログラムの中からlsコマンドを実行するJavaプログラム
     8  */
     9 public class Exec_lsCommand {
    10     public static void main(String[] args) {
```

```
11          try {
12              Process child =
                        Runtime.getRuntime().exec("ls");
13              BufferedReader in =
14                  new BufferedReader(
                            new InputStreamReader(
15                              child.getInputStream()));
16              String line;
17              while ((line = in.readLine()) != null) {
18                  System.out.println(line);
19              }
20          } catch(IOException e) {
21              e.printStackTrace();
22          }
23      }
24  }
```

```
[motoki@x205a]$  javac Exec_lsCommand.java
[motoki@x205a]$  java Exec_lsCommand
CopyAndShowBytesInFile.java
CopyAndShowCharsInFile.java
CopyAndShowCharsInFileAnyCoding.java
ExampleOfStopByException.java
ExampleOf_printStackTrace.java
Exec_emacsCommand.java
Exec_lsCommand.class
Exec_lsCommand.java
ListFilesInDirectory.java
ObserveFreeMemory.class
ObserveFreeMemory.java
ObserveFreeMemory0.java
ObserveFreeMemoryAndCallGc.java
ascii.txt
euc.txt
```

```
utf8.txt
[motoki@x205a]$  ls
CopyAndShowBytesInFile.java              ListFilesInDirectory.jav
CopyAndShowCharsInFile.java              ObserveFreeMemory.class
CopyAndShowCharsInFileAnyCoding.java     ObserveFreeMemory.java
ExampleOfStopByException.java            ObserveFreeMemory0.java
ExampleOf_printStackTrace.java           ObserveFreeMemoryAndCall
Exec_emacsCommand.java                   ascii.txt
Exec_lsCommand.class                     euc.txt
Exec_lsCommand.java                      utf8.txt
[motoki@x205a]$
```

23–5 ほぼ自習 コレクション・フレームワーク

一般的なアルゴリズムについて議論する中で、

- 複数の要素をまとめて保持するためのデータ構造，
- データ構造内部に保持された要素を操作するアルゴリズム

が色々考え出されている。

これらの代表的なデータ構造／アルゴリズムを使って

- 複数のオブジェクトの集まり（コレクション）を保持し、
- 簡潔で抽象度の高い操作方法

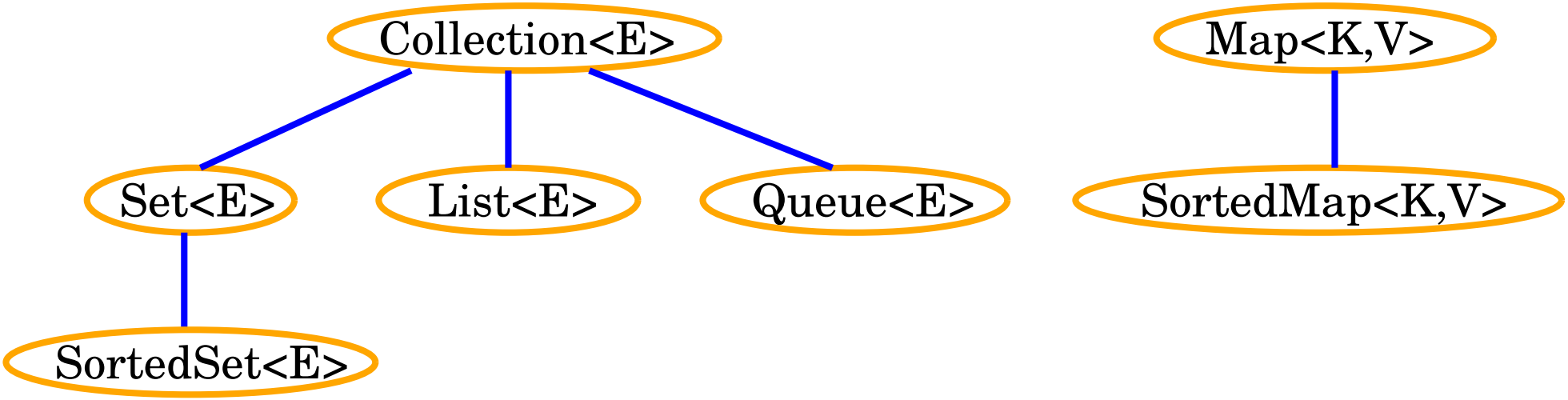
を提供する統合的な枠組み（コレクションフレームワークという）が、Java標準ライブラリの中に用意されている。

コレクションフレームワークの基本構成：

要素のまとめ方(e.g. 保持順序は重要か, 重複要素を許すか)／利用の仕方に応じて、数種類の基本的な「オブジェクトの集まり」を考えることができる。Java標準ライブラリでは、これらの中の幾つかについて、

- 簡潔で抽象度の高い基本操作を提供するためのインタフェースが定義され、
- そのインタフェースを実装したクラスが代表的なデータ構造を使って定義され、さらに
- そのインタフェースを実装したオブジェクトに対して検索や整列化といった有益な処理を多態的に遂行するstaticなユーティリティメソッド群が、(別のクラス, `java.util.Collections`, の中で)代表的なアルゴリズムを使って定義されている。

コレクション関連のインタフェースは次の様な階層関係を持つ。

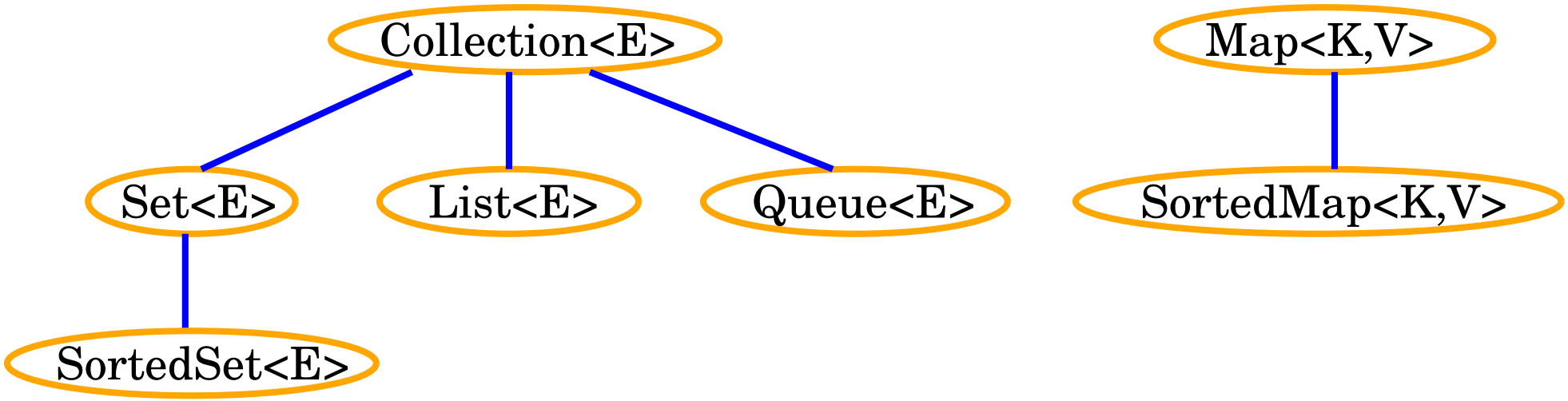


ここで、
各々のインタフェースで想定しているのは
次の様な種類の「オブジェクトの集まり」である。

◇ java.util.Collection<E> ··· 単純な形の要素の集まり。

◇ java.util.Set<E> ··· 数学的な意味での「集合(set)」。重複要素は許さず、保持順序は重要でない。

◇ java.util.SortedSet<E> ··· 数学的な意味での「集合」に、「(要素内のkeyに基いて) 昇順に要素を保持」という制約を追加したもの。



◇ java.util.List<E> ··· 要素を一列に並べたもの。重複要素は許し、並び順も考慮する。

◇ java.util.Queue<E> ··· 処理待ちの要素を入れておく「待ち行列(queu

◇ java.util.Map<K,V> ··· 数学的な意味での「写像のグラフ」、すなわち何らかの写像 f:K→V の対応関係を表す集合 {(x, f(x)) | x∈K}。

◇ java.util.SortedMap<K,V> ··· 数学的な意味での「写像のグラフ」に、「(要素内のキーに基いて)昇順に要素を保持」という制約を追加したもの。

以下、List<E> と Map<K,V> について簡単に紹介する。

インタフェース java.util.List<E> :

- 要素を一列に並べたものを扱うためのCollection<E>の直系のサブインタフェースで、次の様なメソッドを持つ。

メソッド ヘッダー … 説明
public int size() … リスト内の要素の個数を返す。
public boolean isEmpty() … リストが空かどうかを調べてその結果を返す。
public E get(int *index*) … リスト中の*index*番目の要素(への参照)を返す。
public E set(int *index*, E *element*) … リスト中の*index*番目の要素(への参照)を*element*に置き換え、以前にあった要素を返す。 (実装なしの可能性もある。)
……(続く)……

public boolean add(E *e*) … 引数で指定された要素をリストの最後に追加し、(追加は成功するので)trueを返す。　(実装なしの可能性もある。)
public void add(int *index*, E *element*) … リスト中の*index*番目の位置に*element*を挿入する。それ以降の要素の要素番号は1つズレることになる。(実装なしの可能性もある。)
public boolean remove(Object *o*) … 引数で指定された要素がリスト中にあれば、その最初のものをリストから削除してtrueを返す。(実装なしの可能性もある。)
public E remove(int *index*) … リスト中の*index*番目の位置の要素を削除し、削除した要素を返す。それ以降の要素の要素番号は1つズレることになる。(実装なしの可能性もある。)
public void clear() … 全ての要素をリストから削除する。(実装なしの可能性もある。)
……(続く)……

public booean contains(Object o) … 引数で指定した要素がリスト内にあるかどうかを調べてその結果を返す。
public int indexOf(Object o) … 引数で指定した要素がリスト内にあれば最初に現れる位置の番号を返し、なければ -1 を返す。
public int lastIndexOf(Object o) … 引数で指定した要素がリスト内にあれば最後に現れる位置の番号を返し、なければ -1 を返す。
public Iterator<E> iterator() … リスト内の要素を適切な順序で繰り返し処理するためのイテレータを返す。
public ListIterator<E> listIterator(int $index$) … $index$番目の要素から始め、リスト内の要素を適切な順序で繰り返し処理するためのイテレータを返す。
……（続く）……

`public Object[] toArray()` … リスト内の要素（への参照）を適切な順序で格納する配列を返す。
`public List<E> subList(int` *fromIndex*`, int` *toIndex*`)` … リストの *fromIndex*～（*toIndex*−1）番目の要素から成る部分リストを返す。（リストの一部をリストとして扱うということで、部分リストへの操作によって元のリストにも手が加わることになる。）
……（他のメソッドについては省略）……

- このインタフェースを実装したクラスとしては、ArrayList<E>, LinkedList<E>, 等がある。

◇ java.util.ArrayList<E> ··· 配列を用いて実装。
最初にある大きさの配列が内部で確保され、
容量が足りなくなれば 「より大きな配列を確保し直す」という処理が内部で行われることになる。また、リストへの要素の挿入や削除は、内部の配列要素の移動を伴うことになる。

◇ java.util.LinkedList<E> ··· 線形連結リストを用いて実装。

- List<E>インタフェースを実装したオブジェクトに対して適用できる汎用で有益なメソッドが、java.util.Collections というクラスの中で多数定義されている。

メソッド ヘッダー … 説明
public static <T extends Comparable<? super T>> void sort(List<T> *list*) … 要素の自然な順序に基いて、引数で指定されたリスト内の要素を 昇順に並べ替え る。
public static <T> int binarySearch(List<? extends Comparable<? super T>> *list*, T *key*) … リスト*list*内の要素が自然な順序で並んでいるという仮定の下で二分探索 アルゴリズムを用いて、*list*内の*key*という要素を探しその位置番号(index)を返す。リスト*list*中に*key*がない場合は、−(あるとした場合の位置番号)−1 を返す。
public static <T extends Object & Comparable<? super T>> T min(Collection<? extends T> *coll*) … 要素の自然な順序に基いて、引数で指定されたコレクション内で 最も小さな値を持つ要素 を返す。

public static <T extends Object & Comparable<? super T>> T max(Collection<? extends T> *coll*)

… 要素の自然な順序に基いて、引数で指定されたコレクション内で 最も大きな値を持つ要素 を返す。

public static int indexOfSubList(List<?> *source*, List<?> *target*)

… リスト *source* 中で サブリスト *target* が現れる場合、 最初に現れる 区間の先頭の 位置番号 を返す。現れない場合は -1 を返す。

public static int lastIndexOfSubList(List<?> *source*, List<?> *target*)

… リスト *source* 中で サブリスト *target* が現れる場合、 最後に現れる 区間の先頭の 位置番号 を返す。現れない場合は -1 を返す。

public static void shuffle(List<?> *list*)

… 引数で指定されたリスト内の 要素をランダムに入れ替え る。

public static void reverse(List<?> *list*)

… 引数で指定されたリスト内の要素の 並び順を逆に する。

public static void rotate(List<?> *list*, int *distance*) … 引数で指定されたリスト内の全ての要素を後方に向かって *distance* の距離だけ 循環 させる。（最後尾の要素が順にリストの先頭に挿入されることになる。）
public static void swap(List<?> *list*, int *i*, int *j*) … 引数で指定されたリスト内の *i* 番目と *j* 番目の要素を 交換 する。
public static <T> boolean replaceAll(List<T> *list*, T *oldVal*, T *newVal*) … 引数で指定されたリスト内に現れる *oldVal* という要素を全て *newVal* に 置き換え る。実際に置き換えが行われた場合は true を返す。
public static <T> void fill(List<? super T> *list*, T *elem*) … 引数で指定されたリスト内の 各要素を *elem* に 置き換え る。
public static <T> void copy(List<? super T> *dst*, List<? extends T> *src*) … リスト *src* 内の各要素を別のリスト *dst* に コピー する。
……（他のメソッドについては省略）……

例23．8（List<E>インタフェースの利用例）

トランプのカード52枚(シャッフル済)のオブジェクトのクラスと簡単な動作テストから成るJavaプログラムを表示し、これをコンパイル・実行している様子を次に示す。

```
[motoki@x205a]$  cat -n DeckOfCards.java
     1 import java.util.List;
     2 import java.util.ArrayList;
     3 import java.util.Collections;
     4
     5 /**
     6  * 52枚のトランプカードの組を表すオブジェクトのクラス
     7  */
     8 public class DeckOfCards {
     9     //トランプのカード1枚を表すオブジェクトのクラス
    10     public static class Card
                                implements Comparable<Card> {
    11         static final String[] SUIT_NAME = new String[]
    12             {"spades", "hearts", "diamonds", "clubs"};
```

```
13          static final String[] RANK_NAME = new String[]
14              {"ace", "2", "3", "4", "5", "6", "7",
15               "8", "9", "10", "jack", "queen", "king"};
16          private final int suitId;
17          private final int rankId;
18
19          public Card(int suitId, int rankId) {
20              this.suitId = suitId;
21              this.rankId = rankId;
22          }
23
24          @Override
25          public String toString() {
26              return RANK_NAME[rankId] + "_of_"
                                         + SUIT_NAME[suitId];
27          }
28
29          public int getSuitId() {
30              return suitId;
```

```
31          }
32
33          public int getRankId() {
34              return rankId;
35          }
36
37          public int compareTo(Card competitor) {
38              if (this.suitId < competitor.suitId
39                  || (this.suitId == competitor.suitId
40                      && this.rankId
                                    < competitor.rankId))
41                  return -1;
42              else if (this.suitId == competitor.suitId
43                       && this.rankId
                                    == competitor.rankId)
44                  return 0;
45              else
46                  return 1;
47          }
```

```
48      }
49      //-----------------------------------------------
50
51      public List<Card> cardList;
52
53      public DeckOfCards() {
54          cardList = new ArrayList<Card>();
55          for (int suitId=0; suitId<Card.SUIT_NAME.
                                          length; ++suitId) {
56              for (int rankId=0; rankId<Card.RANK_NAME.
                                          length; ++rankId) {
57                  cardList.add(new Card(suitId, rankId));
58              }
59          }
60          Collections.shuffle(cardList);
61      }
62
63      @Override
64      public String toString() {
```

```
    65          return cardList.toString();
    66      }
    67
    68      //--------------------単体での動作テスト用----------
    69      public static void main(String[] args) {
    70          DeckOfCards deck = new DeckOfCards();
    71          System.out.println(deck.cardList.subList(0, 5)
    72          Collections.sort(deck.cardList);
    73          System.out.println(deck.cardList.subList(0, 5)
    74      }
    75 }
[motoki@x205a]$  javac DeckOfCards.java
[motoki@x205a]$  java DeckOfCards
[4_of_clubs, 8_of_hearts, 7_of_spades, queen_of_hearts, 9_of_c
[ace_of_spades, 2_of_spades, 3_of_spades, 4_of_spades, 5_of_sp
[motoki@x205a]$ java DeckOfCards
[jack_of_hearts, 6_of_hearts, 9_of_clubs, 10_of_spades, 5_of_s
[ace_of_spades, 2_of_spades, 3_of_spades, 4_of_spades, 5_of_sp
[motoki@x205a]$
---
```

インタフェース java.util.Map<K,V> :

- 数学的な意味での「写像(map)のグラフ」を扱うためのインタフェースで、次の様なメソッドを持つ。

メソッド ヘッダー ··· 説明
public int size() ··· マップ内の対応付けの個数を返す。
public boolean isEmpty() ··· マップが空かどうかを調べてその結果を返す。
public V get(Object *key*) ··· マップ中で*key*に対応付けられているオブジェクトを返す。
public V put(K *key*, V *value*) ··· マップ中に $key \to value$ という対応付けを組み込む。その際、*key*に対する対応付けが既に存在していた場合は、その対応付けの先の値を*value*に変更し元の値を返す。*key*に対する対応付けが存在しなかった場合は、null を返す。 (実装なしの可能性もある。)
.....(続く) .....

public V remove(Object *key*)

… マップ中に*key*に対する対応付けが存在していた場合は、その対応付けを削除しその対応付けの先の値を返す。*key*に対する対応付けが存在しなかった場合は、null を返す。
（実装なしの可能性もある。）

public void clear()

… 全ての対応付けをマップから削除する。（実装なしの可能性も）

public booean containsKey(Object *key*)

… 引数で指定した*key*に対する対応付けがマップ内にあるかどうかを調べてその結果を返す。

public booean containsValue(Object *value*)

… 引数で指定した*value*をmapping先の値に持つ対応付けがマップ内にあるかどうかを調べてその結果を返す。

public Set<K> keySet()

… マップのキーを要素とするSetインスタンスを返す。（マップの一部をSetインスタンスの要素として扱うということで、Setインスタンスへの操作によって元のマップにも手が加わることになる。）

……（続く）……

public Collection<V> values()

… マップの値部を要素とするCollectionインスタンスを返す。(マップの一部をCollectionインスタンスの要素として扱うということで、Collectionインスタンスへの操作によって元のマップにも手が加わることになる。)

public Set<Map.Entry<K,V>> entrySet()

… マップ内の個々の対応付けを表すMap.Entryオブジェクトを考え、それらを要素とするSetインスタンスを返す。(マップの一部をSetインスタンスの要素として扱うということで、Setインスタンスへの操作によって元のマップにも手が加わることになる。) ここで、Map.Entry<K,V> は入れ子インタフェースで、次の様なメソッドを持つ。

- public K getKey() … 対応付けを表すエントリ内のキーを返す。
- public V getValue() … 対応付けを表すエントリ内の値部を返す。
- public V getKey(V *value*) … 対応付けを表すエントリの値部を *value* に
  設定し古い値を返す。

……(他のメソッドについては省略)……

- このインタフェースを実装したクラスとしては、HashMap<K,V>, TreeMap<K,V>, LinkedHashMap<K,V>, 等がある。

◇ java.util.HashMap<K,V> ··· 「ハッシュテーブル」を用いて実装。（すなわち、キーにhashCode()メソッドを施すことによって得られる（ハッシュ）値を位置データと考え、基本的にはテーブル内のその位置に対応付け要素を配置する、という実装方法。） 要素の追加、削除、検索にほぼ$O(1)$の時間しかかからないので、広く利用されている。

◇ java.util.TreeMap<K,V> ··· 「赤黒木」を用いて実装。

◇ java.util.LinkedHashMap<K,V> ··· HashMap<K,V>の直系のサブクラス。登録された順に要素を記録した双方向連結リストも保持する。

- Map<K,V>インタフェースを実装したオブジェクトに対して適用できる汎用で有益なメソッドは、特にない。
  ただ、keySet()やvalues(), entrySet()メソッドによって作られるSetやCollectionオブジェクトに対しては、java.util.Collectionsクラス中の多数のstaticなメソッドを適用可能である。

例23．9（Map<K,V>インタフェースの利用例）

標準入力に現れる単語の頻度を調べて表示するJavaプログラムを表示し、これをコンパイル・実行している様子を次に示す。

```
[motoki@x205a]$  cat -n ExampleOfUsingMap.java
     1 import java.util.Map;
     2 import java.util.HashMap;
     3 import java.util.Scanner;
     4
     5 /** 標準入力に現れる単語の頻度を調べて表示 */
     6 public class ExampleOfUsingMap {
     7     public static void main(String[] args) {
```

```
 8          Scanner inputScanner = new Scanner(System.in);
 9          Map<String,Integer>
                    map = new HashMap<String,Integer>();
10
11          while (inputScanner.hasNext()) {
12              String word = inputScanner.next();
13              Integer freq = map.get(word);
14              map.put(word, (freq==null) ? 1 : freq+1);
15          }
16          System.out.println(map.size()
                        + " distinct words appeared:");
17          System.out.println(map);
18      }
19 }
[motoki@x205a]$  javac ExampleOfUsingMap.java
[motoki@x205a]$  cat ExampleOfUsingMap.data
```

```
Java is a programming language originally developed by James G
Microsystems and released in 1995 as a core component of Sun M
Java platform.
[motoki@x205a]$  java ExampleOfUsingMap
                                        < ExampleOfUsingMap.data
23 distinct words appeared:
is=1, James=1, a=2, developed=1, at=1, as=1, by=1, in=1,\
 Java=2, and=1, Gosling=1, 1995=1, released=1, of=1,\
 programming=1, originally=1, Microsystems=1, component=1,\
 platform.=1, Sun=2, Microsystem's=1, core=1, language=1
```

(注：長い文字列なので改行して表示)

```
[motoki@x205a]$
```

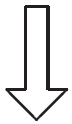
23–6 ほぼ自習 その他の有用なクラス、パッケ

文字列結合の演算子 +

...手軽に使えて便利。

しかし、無造作に使うと非常に非効率な処理になる可能性がある。

例えば、
この演算子を繰り返し使って保持する文字列をどんどん伸ばす場合には、演算子+を使う度に結合対象の文字列のコピー（と場合によってはオブジェクトの削除と生成）が行われる。



java.lang.StringBuilderクラス： ...JDK1.5(J2SE5,2004)以降

- StringBuilderオブジェクトは、Stringオブジェクトと異なり変更が可能で、文字の並びを保持する可変長配列を内部に持つものとして扱われる。

- 次の様なコンストラクタ、メソッドを持つ。

コンストラクタ／メソッド ヘッダー ··· 説明
public StringBuilder() ··· 空の文字の並びを初期値として、容量が16のインスタンスを生成する。
public StringBuilder(int *capacity*) ··· 空の文字の並びを初期値として、容量が*capacity*のインスタンスを生成する。
public StringBuilder(String *str*) ··· *str*を初期の文字の並びとし、その後に16個の文字を追加できる容量を持ったインスタンスを生成する。
·····（続く）·····

`public int length()` … 保持されている文字の並びの長さを返す。
`public int capacity()` … 現在のバッファ容量(保持可能な文字数)を返す。
`public void ensureCapacity(int` *minCapacity*`)` … バッファ容量が*minCapacity*以上になる様にする。具体的には、現在のバッファ容量が*minCapacity*未満の場合は、バッファ容量を$\max\{minCapacity,$ (現在の容量)$\times 2+2\}$ に設定する。
`public StringBuilder append(String` *str*`)` … 引数で指定された文字列*str*を内部で保持している文字の並びの最後尾に追加で繋げ、このオブジェクトへの参照を返す。但し、*str*が`null`の場合は `"null"` という文字の並びを繋げる。
`public StringBuilder append(int` *i*`)` … 引数で指定された`int`型の値の文字列表現を内部で保持している文字の並びの最後尾に追加で繋げ、このオブジェクトへの参照を返す。
……(続く)……

public StringBuilder delete(int *start*, int *end*) … 内部で保持している文字の並び中の、*start*~ *end*−1 番目の文字を削除し、このオブジェクトへの参照を返す。
public StringBuilder insert(int *offset*, String *str*) … 内部で保持している文字の並びの、*offset*番目の文字から始まる位置に文字列*str*を挿入して、このオブジェクトへの参照を返す。但し、*str*がnullの場合は "null" という文字の並びを挿入する。
public StringBuilder replace(int *start*, int *end*, String *str*) … 内部で保持している文字の並び中の、*start*~ *end*−1 番目の文字の部分を文字列*str*で置き換え、このオブジェクトへの参照を返す。
……（続く）……

public int indexOf(String *str*) … 内部で保持している文字の並びの中に文字列 *str* が部分列として現れる場合、最初に現れる区間の先頭の位置番号（index）を返す。現れない場合は -1 を返す。
public StringBuilder reverse() … 内部で保持している文字の並びの並び順を逆にした上で、このオブジェクトへの参照を返す。
public String toString() … 内部で保持している文字の並びと同じ内容のString オブジェクトを返す。
……（他コンストラクタ／メソッドについては省略）……

例23. 10 (String vs. StringBuilder)

長い文字列を徐々に構築する場合に、
(1) 文字列結合の+演算子を繰返す方法と
(2)StringBuilderを使った連結を繰返す方法
の2つが考えられる。これらの方法の処理時間を測って比較するJavaプログラムを表示し、これをコンパイル・実行している様子を次に示す。

```
[motoki@x205a]$  cat -n EffectOfStringBuilder.java
     1 import java.lang.management.ThreadMXBean;
     2 import java.lang.management.ManagementFactory;
     3
     4 /**
     5  * 次の2つの方法の計算時間を比較するJavaプログラム。
     6  * (1) 文字列結合の+演算子を繰返して長い文字列を構築
     7  * (2)StringBuilderを使った連結を繰返して長い文字列を構...
     8  */
     9 public class EffectOfStringBuilder {
    10     public static void main(String[] args) {
```

```
11          ThreadMXBean cpuTimer
                    = ManagementFactory.getThreadMXBean();
12          long timeAtPoint0, timeAtPoint1,
                                timeAtPoint2, timeAtPoint3;
13
14          timeAtPoint0 = cpuTimer.
                                    getCurrentThreadCpuTime();
15          //(1)文字列結合の+演算子を繰返して長い文字列を構...
16          String str0 = "";
17          for (int i=0; i<100000; ++i)
18              str0 += "a";
19
20          timeAtPoint1 = cpuTimer.
                                    getCurrentThreadCpuTime();
```

```
21          //(2.1)StringBuilderを使った連結を繰返して長い文...
22          StringBuilder str1 = new StringBuilder();
23          for (int i=0; i<100000; ++i)
24              str1.append("a");
25
26          timeAtPoint2 = cpuTimer.
                                    getCurrentThreadCpuTime();
27          //(2.2)StringBuilderを使った連結を繰返して長い文...
28          //      (StringBuilderの初期容量を十分大きく指定...
29          StringBuilder str2 = new StringBuilder(100000);
30          for (int i=0; i<100000; ++i)
31              str2.append("a");
32
33          timeAtPoint3 = cpuTimer.
                                    getCurrentThreadCpuTime();
34          //CPU時間計測の結果を出力
35          System.out.printf("(1)  ==>%9.5f秒%n" +
```

```
    36                                  "(2.1)==>%9.5f秒%n" +
    37                                  "(2.2)==>%9.5f秒%n",
    38                              (timeAtPoint1 - timeAtPoint0)*1e-9,
    39                              (timeAtPoint2 - timeAtPoint1)*1e-9,
    40                              (timeAtPoint3 - timeAtPoint2)*1e-9);
    41     }
    42 }
[motoki@x205a]$  javac EffectOfStringBuilder.java
[motoki@x205a]$  java EffectOfStringBuilder
(1)  ==> 21.09000秒
(2.1)==>  0.00000秒
(2.2)==>  0.00000秒
[motoki@x205a]$  java EffectOfStringBuilder
(1)  ==> 21.10000秒
(2.1)==>  0.01000秒
(2.2)==>  0.00000秒
[motoki@x205a]$
```

GUI構築のためのクラス群：

例えば、次の通り。（詳細は、JavaAPI仕様書等で調べて下さい。）

- java.awtパッケージ　… GUIを記述するためのクラスから成る。(Abst
  Window Toolkit)

- java.awt.eventパッケージ　… AWTコンポーネントによって起こる様々な種類のイベント(e.g.GUI上のボタンが押された)を処理するためのインタフェースとクラスを提供する。

- java.awt.fontパッケージ　… AWTのフォント関連のインタフェースとクラスを提供する。

- java.awt.colorパッケージ　… AWTのカラースペースのクラスを提供する。

- javax.swingパッケージ　… GUIコンポーネントを扱うためのクラスから成る。(awtはネイティブのGUIに依存しているが、こちらは全てのシステム上で出来るだけ同じ様に見え振舞う様に書かれている。)

- javax.swing.textパッケージ　··· Swingテキストコンポーネントを処理するクラスとインタフェースを提供する。

- javax.swing.eventパッケージ　··· Swingコンポーネントによって起こる様々な種類のイベント(e.g.GUI上のボタンが押された)を処理するためのインタフェースとクラスを提供する。

例23．11（GUI構築）

修正

GUIウィンドウを開きその中で、

2つの正整数x,yを入力して⌈x/y⌉を出力する、

という作業を行える様にするJavaプログラムを表示し、...

```
[motoki@x205a]$  cat -n ExampleOfGUI.java
     1 import java.awt.*;
     2 import java.awt.event.*;
     3 import javax.swing.*;
     4
     5 /**
     6  * 2つの正整数x,yを入力してx/yの小数部を切り上げて得ら
     7  * れるる整数値を出力、という作業をGUI上で行うJavaプロ...
     8  */
     9 public class ExampleOfGUI extends JPanel {
    10     JLabel label1 = new JLabel("<input>   x = ",
                                       SwingConstants.RIGHT);
    11     JLabel label2 = new JLabel("<input>   y = ",
                                       SwingConstants.RIGHT);
```

```
12      JLabel label3 = new JLabel("==> ceiling(x/y) = ",
                                        SwingConstants.RIGHT);
13      JTextField text1 = new JTextField();
14      JTextField text2 = new JTextField();
15      JTextField text3 = new JTextField();
16      JButton button1 = new JButton("calculate");
17      JButton button2 = new JButton("close");
18
19      public ExampleOfGUI() {
            レイアウトマネージャの指定
20          setLayout(new GridLayout(4, 2, 10, 15));
21          add(label1);
22          add(text1);
23          add(label2);
24          add(text2);
25          add(label3);
26          add(text3);
27          add(button1);
28          add(button2);
```

コンストラクタ

行数,列数,水平間隔,垂直間隔

```
29          button1.addActionListener(new ActionListener() {
30                  public void actionPerformed(ActionEvent event) {
31                      try {
32                          int x = Integer.parseInt(text1.getText());
33                          int y = Integer.parseInt(text2.getText());
34                          int quotient = x/y + (x%y + y - 1)/y;
35                          text3.setText(Integer.toString(quotient));
36                      } catch (NumberFormatException e){
37                          text3.setText("Input Error. ==>
38                      }
39                  }
40              });
```

無名内部クラス

```
41          button2.addActionListener(new ActionListener() {
42                  public void actionPerformed(ActionEvent event) {
43                      System.exit(0);
44                  }
45              });
46      }
47
48      public static void main(String[] args) {
49          JFrame frame = new JFrame();
50          frame.add(new ExampleOfGUI());
51          frame.setSize(300, 200);
52          frame.setDefaultCloseOperation(JFrame.EXIT_ON_C
53          frame.setVisible(true);
54      }
55 }
[motoki@x205a]$  javac ExampleOfGUI.java
[motoki@x205a]$  java ExampleOfGUI&
```

無名内部クラス

```
[1] 29328
[motoki@x205a]$
```

ここで、

- `java ExampleOfGUI&` とコマンド入力すると、...