

ファイル入出力

2018/6/19(火) プログラミングIV 第十回
福井大学 工学研究科 情報・メディア工学専攻
長谷川達人



演習問題の解説

- HP上にアップロードした.
- 点数に**不服がある人**は個別に申し立ててください.
- 前回の解説を今から行います.

質問への回答

- APIとはなんの略？
 - Application Programming Interface
 - すなわちアプリ（App.）の開発（Programming）手続き
 - 公開されている情報は基本全てAPIリファレンスにある.
- APIリファレンスをうまく使うには？
 - ググると大体APIリファレンス見るまでもなく、ブログに手法が公開されていることが多いので、そちらを読めば良い.
 - ただ、より詳細に、より正確に情報を得るときにはAPIリファレンスを読むことが必須であるので、うまく使うというよりは、「いざとなれば使える」なら無理に使わなくても良い.
- APIリファレンスと参考書どっちがいい？
 - 前者は詳細を調べる時、後者は基本を学習する時.

質問への回答

- Date型って結局何？
 - 一つの日時情報（例：20180101 00:00:00）を保持するためのクラスで， Long-Calendar-Stringとの仲介役と思えばよい。
- Mathのメソッドはなぜ毎回「Math.」 とつけるのか。
 - 前回の課題の「StringUtil.」 をつける理由と同じ。 Mathの中身は全てstaticメソッドなのでインスタンス化せず使う。
- Stringのメソッドはなぜ「String.」 とつけないのか。
 - Stringのメソッドはインスタンスメソッドなので， Stringインスタンスにピリオドをつけて呼び出す。（例：str.split(",");）
- SimpleDateFormatのyなどは自分で決められるのか。
 - APIリファレンスの定義に従う。 決めれない。

質問への回答

- `System.currentTimeMillis()`はなぜ1970年～？
 - 1970年～な理由はわからなかったが、`long`値をマイナスにすることで、1970年以前を表現することは可能らしい。
- なぜ`Date`は`java.util`と`java.sql`があるのか。
 - `java.util.Date`はJava1からある超古参APIで、すごい使いづらいといわれている（`year, month`等で初期化できないし、`month`が0からだし）。`java.sql`はDB処理を行うパッケージで、DB内と整合をとるときに使う。
- 他に便利なライブラリある？
 - Java標準ではなく、外部のライブラリに色々便利なライブラリがある。ただ、何かしらの用途に特化していくことになるので、授業ではあまり触れないでおく。

質問への回答

- 標準ライブラリはどのくらいあるのか。更新されるのか。
 - 数えられないくらいたくさんある。
 - Javaのバージョンが上がるごとに更新される。
- APIリファレンスに「推奨されていません」と書かれたものは使うとエラーになるのか。
 - 「非推奨です」とWarningが出るが、一応実行はできる。
 - 非推奨なものはバグやセキュリティ上の問題があることが多いため、なるべく使わないようにする（大体、代替案がある）。
- プログラミングは慣れなのか。
 - 慣れだけではなく、学習にかけた時間や、理解しようとする意志なども重要だと思うが、何とも言えない。

質問への回答

- なるべく早く資料を公開してほしい.
 - いつも遅くてすみません. . . .
 - 資料, 演習問題, 演習の解例, 演習の採点, 質問解答, を作っていると1週間が一瞬で終わってしまうのです. . . .
- パスワード欄, 確認できればミス率下がるのでは?
 - 確かに! 隠す必要ないので修正してみました.
- やりたいことからメソッドを逆引きするには?
 - ググる. (例: Java 文字列 検索)
- ライブラリを使うメリットデメリットは?
 - デメリットはない. メリットは車輪の再開発をしなくても済む点である. 再開発するとバグが混入するリスクがある.

質問への回答

- 期末テストはどのような形式か.
 - この質問回答してるのに何度もくるので, WebClass上に模擬試験を準備した. 参考まで.
- 試験中にAPIリファレンスを見るのはありますか?
 - なし. WebClassの画面のみ見ることができる.
- 自由課題自宅の開発すれば13, 14回休んでもよいか.
 - ダメ (出席確認が一応定められているので).
 - 自宅の開発して, 授業時間中は資料作りとかはOK.
- 自由課題に使う名目でハトヤマ先生描いてもよいか.
 - 好きにしてくれてよい. 長谷川研は落書きで溢れかえっている.

質問への回答

- 自由課題の最低ラインは？
 - コンパイルエラーなしに動いている状態（Hello Worldとコンソール出力できれば1点くらいはあげる）。
- 自由課題はオリジナルでなければだめか。
 - 課題として提出するだけなら、既存のゲーム等を再現してもよい（そのままWebで配信すると著作権アウト）。評価としては既存再現でも全然OK！
- 自由課題，難易度で評価されると困る。
 - それ以外のところでアピール頑張って。
- 発表時間が想像以上に短くて足りるか不安。
 - 80人＋人の入れ替わり時間を考慮すると1分が限界。 . .

本講義の概要

前半		後半	
第1回	基本文法の復習	第9回	標準ライブラリ
第2回	クラス～カプセル化の復習	第10回	ファイル入出力
第3回	抽象クラス, インタフェース	第11回	デバッグ, インポート, 高速化
第4回	ポリモーフィズム	第12回	オブジェクト指向
第5回	GUI 1	第13回	自由開発演習 1
第6回	GUI 2	第14回	自由開発演習 2
第7回	スレッド, 例外処理	第15回	自由開発演習発表会
第8回	ジェネリクス, コレクション	第16回	期末試験

何を開発するか, 少しずつ考えておくこと

目次

めずらしく目次

- ファイルに文字列を書き込む方法
 - `FileWriter`, `BufferedWriter`
- ファイルから文字列を読み込む方法
 - `FileReader`, `BufferedReader`
- 安全動作のための事前準備
 - `File`
- めんどくさい`close()`とおさらばする方法
 - `try-with-resources`文
- `Monster`インスタンスをファイルに読み書きする方法
 - `FileOutputStream`, `ObjectOutputStream`, `Serializable`
 - `FileInputStream`, `ObjectInputStream`

本日の目標

概要

ファイルを用いたシステム開発について学ぶ.

目標

クラス名を暗記するまでしなくてもよいが、
調べて実装できるようになる（意味を理解する）.



なるほど

本日の提出課題

講義パート

課題を意識しながら
講義を聞くと良い。

課題 1

本日の授業を聞いて、
よくわかったと思う内容を2点簡潔に述べよ。

課題 2

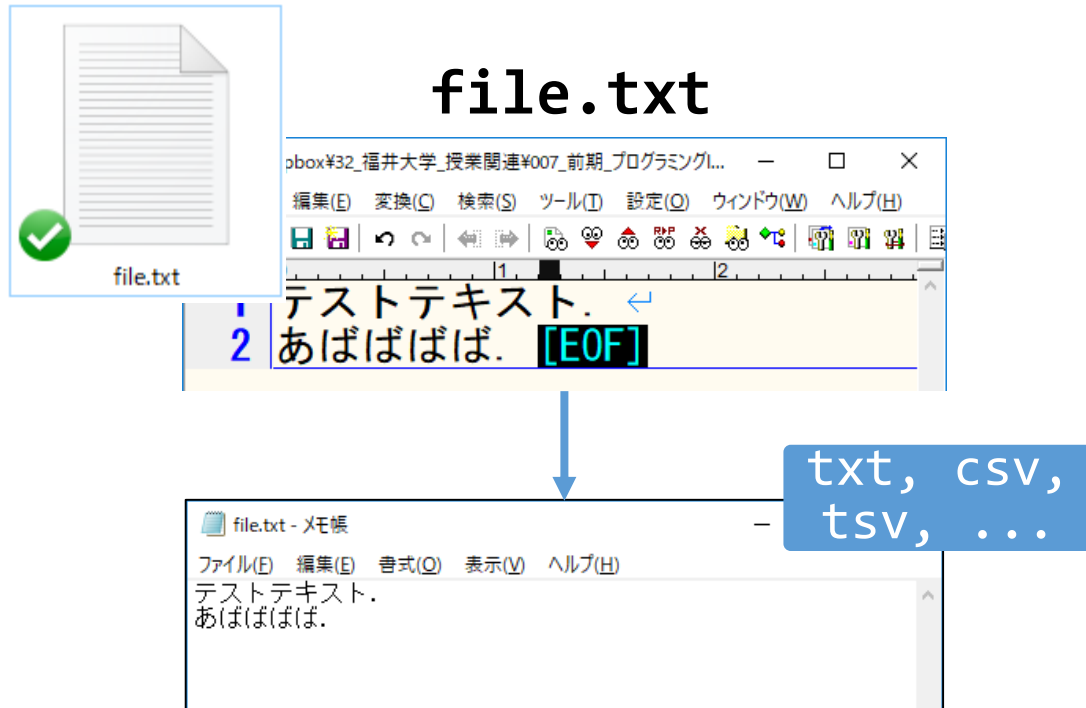
本日の授業を聞いて、
質問事項または**気になった点**を1点以上簡潔に述べよ。

課題 3

感想（あれば）

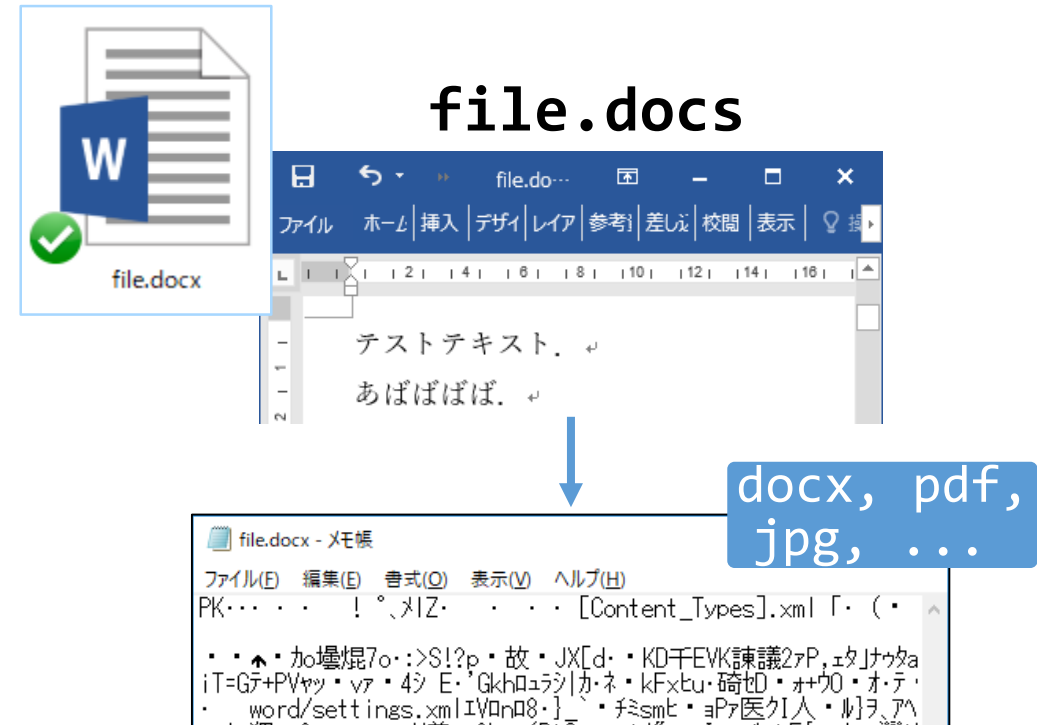
ファイル入出力

ファイルの形式



テキスト形式

文字コードのみから構成される
中身は**人間が理解できる**形式
メモ帳やテキストエディタで編集



バイナリ形式

テキスト形式以外のファイル
中身は基本**人間が理解できない**形式
書式の情報等を保有する(Wordだと)

ファイル入出力

ファイル出力（テキスト形式の出力）

ファイル**出力**には`java.io.PrintWriter`クラスを用いる。

書式

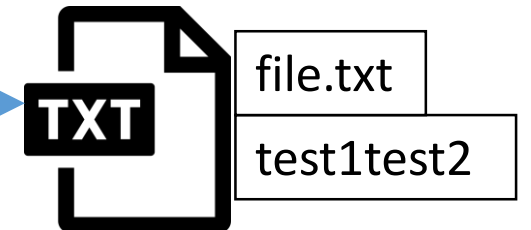
```
PrintWriter fw = new PrintWriter( [出力先ファイルパス] );  
fw.write( [出力する文字列] );  
fw.close();
```

使用例

```
PrintWriter fw = null;  
try {  
    fw = new PrintWriter("file.txt");  
    fw.write("test1");  
    fw.write("test2");  
} catch (IOException e) {}  
finally {  
    if(fw != null){  
        try {  
            fw.close();  
        } catch (IOException e) {}  
    }  
}
```

チェック例外なので
try-catchが必須

close()忘れ防止の為に
finallyに記述



プロジェクトフォルダ
に生成される

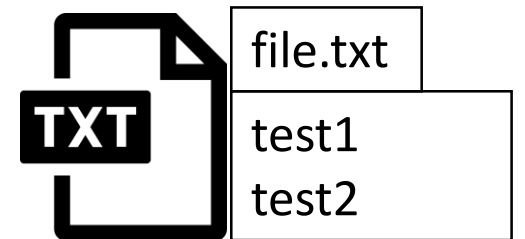
ファイル入出力

ファイル出力（テキスト形式の出力）

次の出力を予想してみよう。

```
FileWriter fw = null;  
try {  
    fw = new FileWriter("file.txt");  
    fw.write("test1¥n");  
    fw.write("test2¥n");  
} catch (IOException e) {}  
finally {  
    if(fw != null){  
        try {  
            fw.close();  
        } catch (IOException e) {}  
    }  
}
```

改行されなかったので
¥n入れてみた。



ファイル入出力

ファイル出力（テキスト形式の出力）

なぜこのような事が起こるのか？

＞テキストの改行コードが複数種類あるため.

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

OSによっても標準で使われる改行コードが違う.

- ¥n (LF) : LINUX系
- ¥r (CR) : MacOS (～v9)
- ¥r¥n (CRLF) : Windows等

ファイル入出力

ファイル出力（テキスト形式の出力）

- 改行コードをbw.newLine()で生成できる.
- バッファに蓄積しまとめてファイル出力できる.

java.io.BufferedWriterクラスを使うとより楽に書ける.

```
FileWriter fw = new FileWriter( [出力先ファイルパス] );
BufferedWriter bw = new BufferedWriter( [FileWriterインスタンス(この場合 fw)] );
bw.write( [出力する文字列] );
bw.newLine();    // 適切な改行コードを自動で生成
bw.close();
```

FileWriter

```
FileWriter fw = new FileWriter("file.txt");
fw.write("出力1¥r¥n");
fw.write("出力2¥r¥n");
fw.write("出力3¥r¥n");
fw.close();
```



BufferedWriter

```
FileWriter fw = new FileWriter("file.txt");
BufferedWriter bw = new BufferedWriter(fw);
bw.write("出力1"); bw.newLine();
bw.write("出力2"); bw.newLine();
bw.write("出力3"); bw.newLine();
bw.close();
```



ファイル入出力

ファイル入力（テキスト形式の入力）

ファイル**入力**には**java.io.FileReader**クラスを用いる。

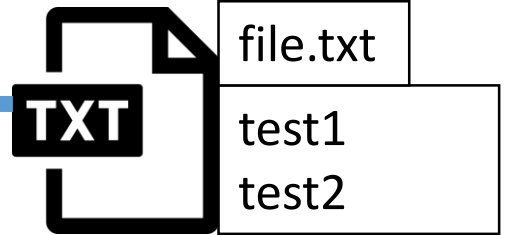
書式

```
FileReader fr = new FileReader( [入力元ファイルパス] );
int ch = fr.read();    // 一文字読み込む
fr.close();
```

使用例

```
FileReader fr = null;
try {
    fr = new FileReader("file.txt");
    int ch;
    while((ch = fr.read()) != -1){
        System.out.print((char)ch);
    }
} catch (IOException e) {
} finally {
    if(fr != null){
        try {
            fr.close();
        } catch (IOException e) {}
    }
}
```

一文字ずつしか
読み込めない



コンソール

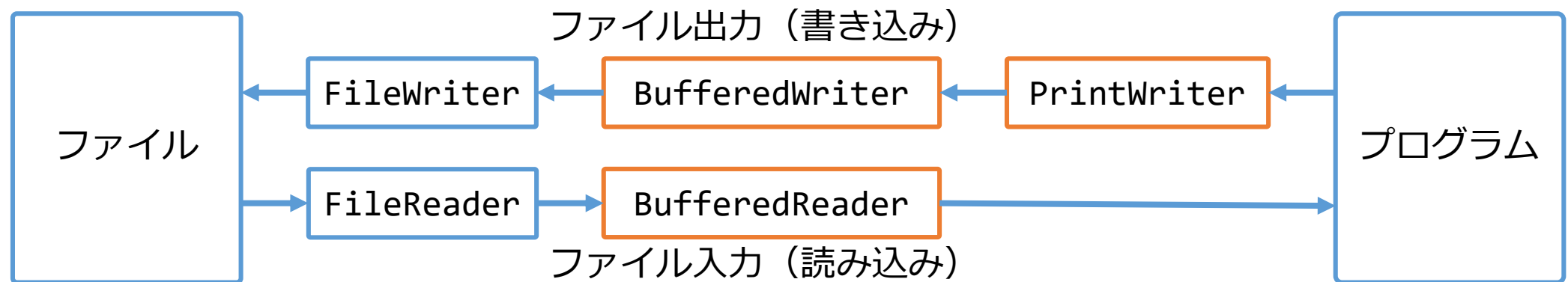
test1
test2

ファイル入出力

ファイル入力（テキスト形式の入力）

java.io.BufferedReaderクラスを使うとより楽に書ける。

```
FileReader fr = new FileReader( [入力元ファイルパス] );  
BufferedReader br = new BufferedReader( [FileReaderインスタンス(この場合 fr)] );  
String str = br.readLine(); // 1行単位で読み込む  
System.out.println(str);  
br.close();
```



必須

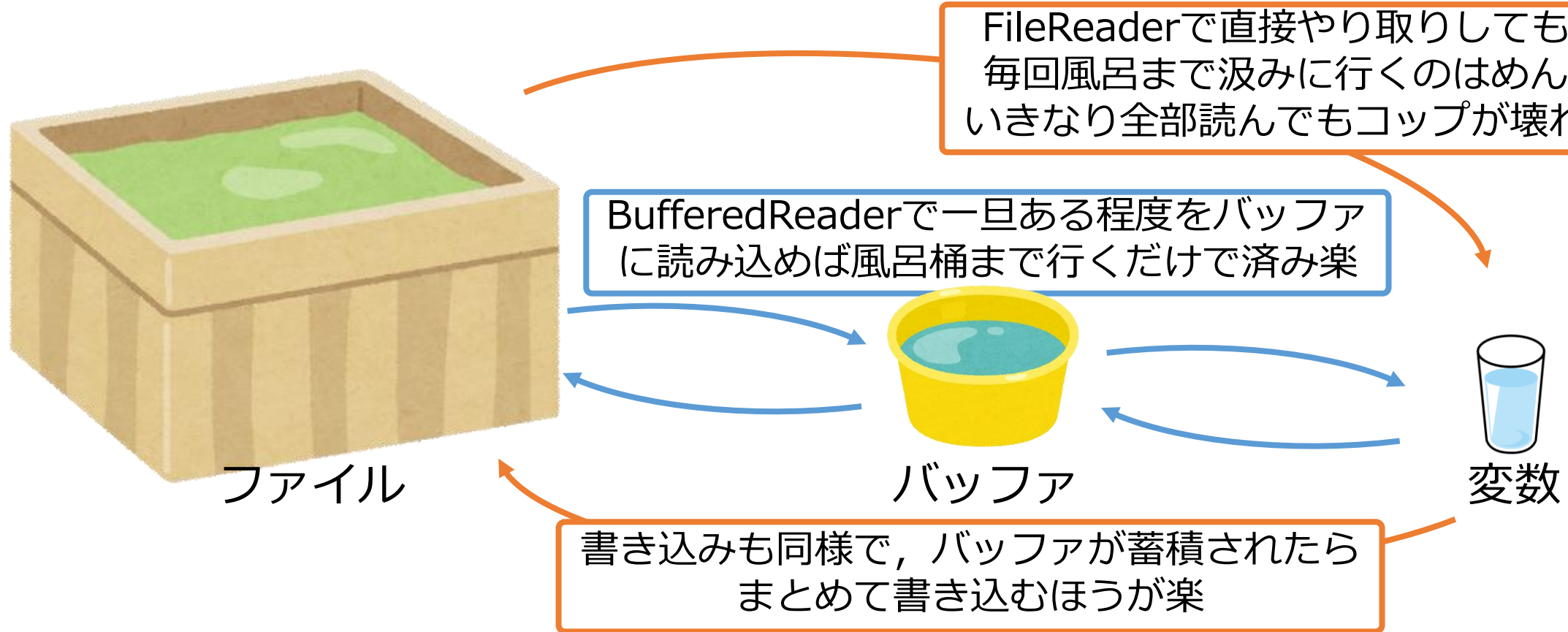
仲介役：あってもなくても良い

ファイル入出力

バッファ



バッファとは、一時的に情報を保存する領域等の総称である。



ファイル入出力

疑問

BufferedWriterやBufferedReaderをclose()したけど、
FileWriterやFileReaderはclose()しなくてよかった
のか？

＞ 前者close()の内部処理でFileWriterや
FileReaderのclose()を呼び出すため問題なし。

```
FileReader fr = new FileReader( [入力元ファイルパス] );  
BufferedReader br = new BufferedReader( [FileReaderインスタンス(この場合 fr)] );  
String str = br.readLine();  
System.out.println(str);  
br.close();
```

これだけでOK

ファイル入出力

練習問題 1 : 書き込んでみよう・読み込んでみよう

1から100までの数字が必要になったのでファイル出力でnumber.txtを出力 + 読み込んで確認する処理を実装しよう.

1. FileWriterを用いてBufferedWriterを使わず出力を実装せよ.
 - まず1からnまでの数字を改行しつつnumber.txtに出力するメソッドwriteNumber(int n)を実装する.
2. FileReaderを用いてBufferedReaderを使わず入力を実装せよ.
 - number.txtを読み込んでコンソール出力するメソッドreadNumber()を実装する (P19あたりを参照する) .
3. 余力のある人は, 2をコンソール出力ではなく, 改行コードで区切ったString配列型で戻り値を返すメソッドに変更せよ.

質問への回答

どうでもいい？ 質問

意識的に使おうとしないと覚えられない。
上半分はせめて当たり前になるべき。

• 便利なショートカットキー教えて。

【知らないで恥ずかしいレベル】

Ctrl+Z : 元に戻す	Ctrl+Y : やり直す (元に戻すを辞める)	
Ctrl+X : 切り取り	Ctrl+C : コピー	Ctrl+V : 貼り付け
Ctrl+A : 全選択	Ctrl+S : 保存	Alt+F4 : Windowを閉じる
Ctrl+P : 印刷	Ctrl+N : 新規作成	Alt+Tab : Window切り替え
Win+E : エクスプローラ表示	Win+D : デスクトップ表示	
Win+R : ファイル名を指定して実行		
"cmd" : コマンドプロンプト, "mspaint" : ペイント, "excel" : エクセル,		
"calc" : 電卓, "notepad" : メモ帳, "control" : コントロールパネル		

【知っておくと良いレベル】

Ctrl+B : 太字	Ctrl+U : 下線,	Ctrl+I : イタリック
Ctrl+F : 検索	Ctrl+R : 置換 (WordとかだとCtrl+H)	
Ctrl+T : 新しいタブ	Ctrl+W : タブを閉じる	
Ctrl+L : URL入力バー	Ctrl+K : Web検索バー	

ファイル入出力

事前チェックとファイル操作

ファイル操作を行う際には例外処理が非常に重要である。

- ＞ そのフォルダ本当にあるの？
- ＞ そのファイル本当にあるの？
- ＞ パーミッション大丈夫？
- ＞ ダメな時どうするの？

例えば数日かかる計算処理を終えて、いざファイル出力しようとした際に例外落ちしたときの絶望を、事前に想定しておく必要がある。

ファイル入出力

事前チェックとファイル操作

読み込み時の事前確認（Fileクラスを仲介して実装可）

- ・ ファイルが存在することの確認
- ・ ファイルが読み込み可能なことの確認

```
File file = new File("file.txt");  
file.exists();           // ファイルの存在確認  
file.isFile();           // ファイルかどうかの確認  
file.isDirectory();      // （一応ディレクトリ版）  
file.canRead();          // 読み込み可能か確認  
file.canWrite();         // 書き込み可能か確認
```

読み込み時は上記チェックでNGの場合、読み込みせずにエラー文を出力して終了する対応を取ることが多い。

読み込み時の事前確認の使用例

```
File file = new File("file.txt");
if(file.exists() && file.isFile() && file.canRead()){
    FileReader fr = null;  BufferedReader br = null;
    try {
        fr = new FileReader(file);
        br = new BufferedReader(fr);
        String str;
        while((str = br.readLine()) != null)
            System.out.println(str);
    } catch (FileNotFoundException e) {
    } catch (IOException e) {
    } finally {
        if(br != null){
            try {
                br.close();
            } catch (IOException e) {}
        }
    }
} else {
    System.out.println("ファイルがないか、書き込めないか、ファイルじゃないか。");
}
```

事前確認

FileからFileReaderへ

BufferedReaderのループ方法

事前に存在確認はしているがnew
FileReader()はFileNotFoundException
のチェック例外なので必要

br.readLine()はIOException
のチェック例外

ファイル入出力

事前チェックとファイル操作

書き込み時の事前確認（Fileクラスを仲介して実装可）

- ・ 対象パス（ディレクトリ）が存在することの確認
- ・ ファイルが存在する場合の対応を検討

チェック方法は前述の通りだが、書き込み時に上記
チェックでNGの場合、エラー文を出力して終了するだけ
だと、書き込み予定だったデータが失われる可能性がある
ため柔軟な対応が必要となる。

ファイル入出力

事前チェックとファイル操作

対象パス（ディレクトリ）の存在確認

```
File file = new File("C:¥¥test¥¥file.txt");
if(!file.getParentFile().exists()){
    // 該当ファイルの親ディレクトリが存在しない場合
    // ディレクトリを作成する.
    file.getParentFile().mkdirs();
}
```

```
File file = new File("file.txt");
if(!file.getParentFile().exists()){
}
```

こういう記述だとパスが不明瞭なのでぬるぽになる

ファイル入出力

事前チェックとファイル操作

ファイルが存在する場合の対応（一例として）

```
public static void write(String name, String contents){
    File file = new File(name);
    if(!file.getParentFile().exists()){
        // 該当ファイルの親ディレクトリが存在しない場合、ディレクトリを作成する.
        file.getParentFile().mkdirs();
    }
    if(file.exists()){ // 既にファイルが存在する場合
        if(file.isDirectory()){
            System.out.println("同名のディレクトリが存在するため別名で保存します. ");
            write(file.getParent()+"¥¥"+System.currentTimeMillis()+".txt", contents);
        }else if(file.isFile()){
            System.out.println("同名のファイルが存在するため別名で保存します. ");
            write(file.getParent()+"¥¥"+System.currentTimeMillis()+".txt", contents);
        }
    }
}
```

以下、通常書き込み処理が続く。
FileWriter fw = new FileWriter(file);

ファイル入出力

事前チェックとファイル操作

疑問：FileNotFoundExceptionをtry-catchできるんだから、わざわざifで検出する必要なくない？

解答：その通り。Fileの説明のために詳細に記述しているが、必ずしも前述の処理をする必要はない。

ただし、FileNotFoundExceptionが発生した際に結局ディレクトリか確認して、親ディレクトリを作成して．．．と処理するのであれば結局はFileを使いこなす必要がある。

ファイル入出力

事前チェックとファイル操作

疑問：例外に対してif文でも事前検出することが可能ではあるが、try-catchとifはどちらを使用すべきか？

解答：諸説ある。「例外はあくまで例外なので、基本的にはcatchされないようにif文を書いておく」という派閥と、「二重チェックする意味がないので例外をキャッチできるならcatchで対応する」という派閥がある。ただしtry-catchの場合は、try内のサブルーチンで発生したExceptionも拾うという点には注意する必要がある。

ファイル入出力

try-with-resources

例外の回でも少し紹介したが、FileWriter等を扱う際にclose()の例外対応を書くことが非常に面倒である。

```
FileReader fr = null;
try {
    fr = new FileReader("file.txt");
    int ch;
    while((ch = fr.read()) != -1){
        System.out.print((char)ch);
    }
} catch (IOException e) {
} finally {
    if(fr != null){
        try {
            fr.close();
        } catch (IOException e) {}
    }
}
```

1. 変数宣言を外に出して

2. close()はfinallyに記述

3. tryの外に宣言したので
nullチェックもして

4. try-catchをネストしつつ
ようやくclose()

ファイル入出力

try-with-resources

JAVA7以降ではこれを簡潔に記述するための
try-with-resources文が使えるようになった。

書式

```
try( [リソースを開く] ){} catch(Exception e){}
```

1. try時にリソースを開く

使用例

```
try(FileReader fr = new FileReader("file.txt")){  
    int ch;  
    while((ch = fr.read())!=-1){  
        System.out.print((char)ch);  
    }  
}catch(IOException e){  
    e.printStackTrace();  
}
```

セミコロンで区切れれば
複数リソースも使用できる。

2. finallyにclose()がなくても
自動でclose()してくれる。

ファイル入出力

練習問題 2 : try-with-resources

練習問題 1 で実装した `writeNumber(int n)` と, `readNumber()` の処理を `try-with-resources` 文を用いて書き直したメソッドを, `writeNumberNew(int n)`, `readNumberNew()` として実装せよ.

ファイル入出力

シリアライゼーション（バイナリ形式の入出力）

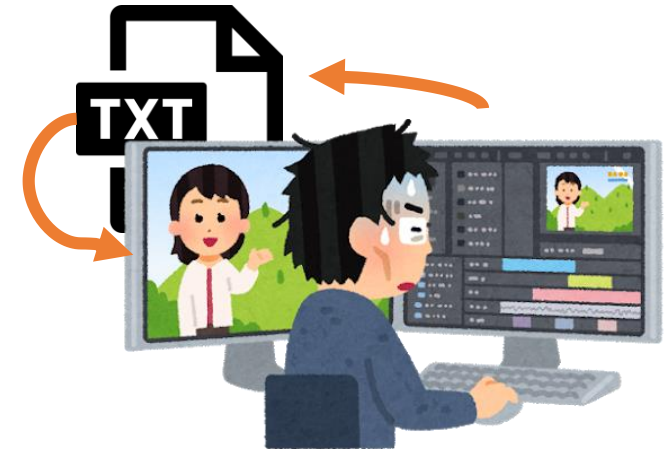
ファイル入出力はどのような時に使うだろうか？



誰かにもらった
大規模データを
処理するとき



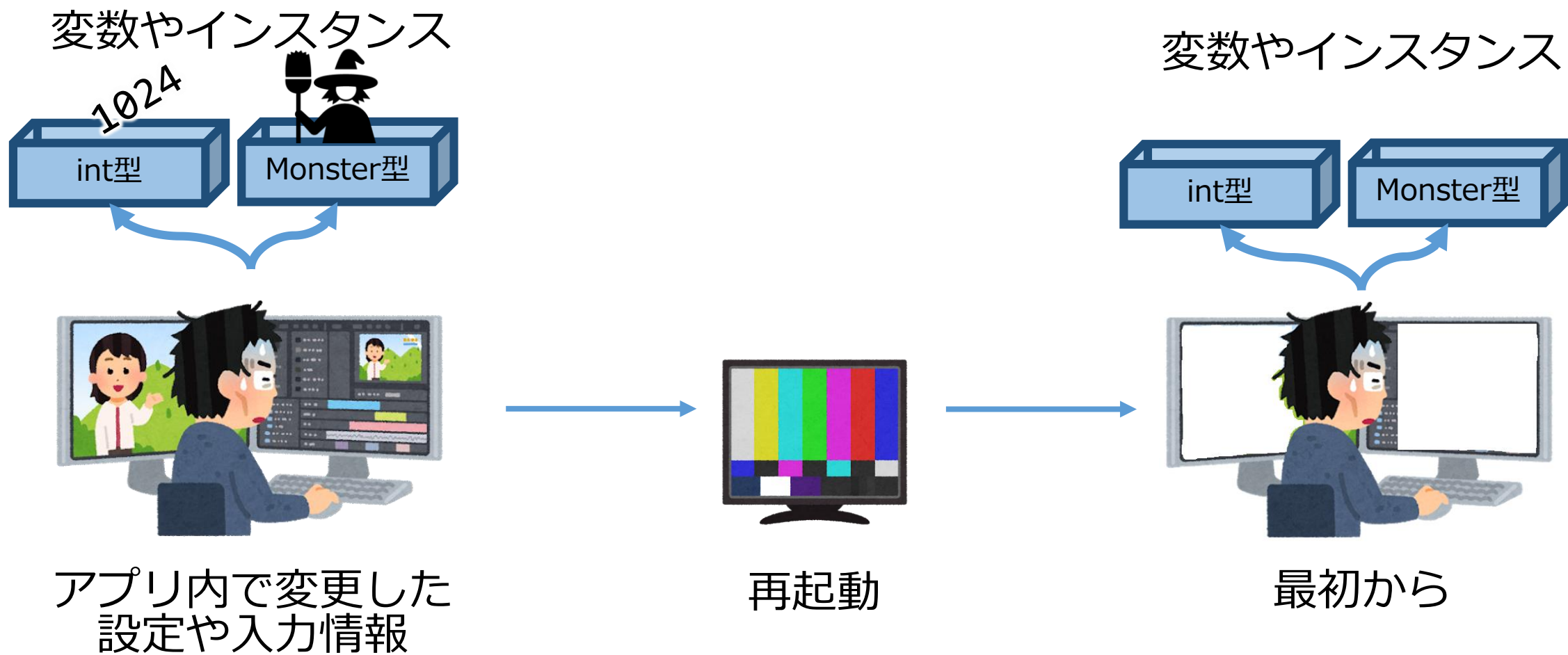
大規模データの
計算結果を出力
して送るとき



入力や設定を外部保存
し、**プログラム再起動**
時に引き継ぐとき

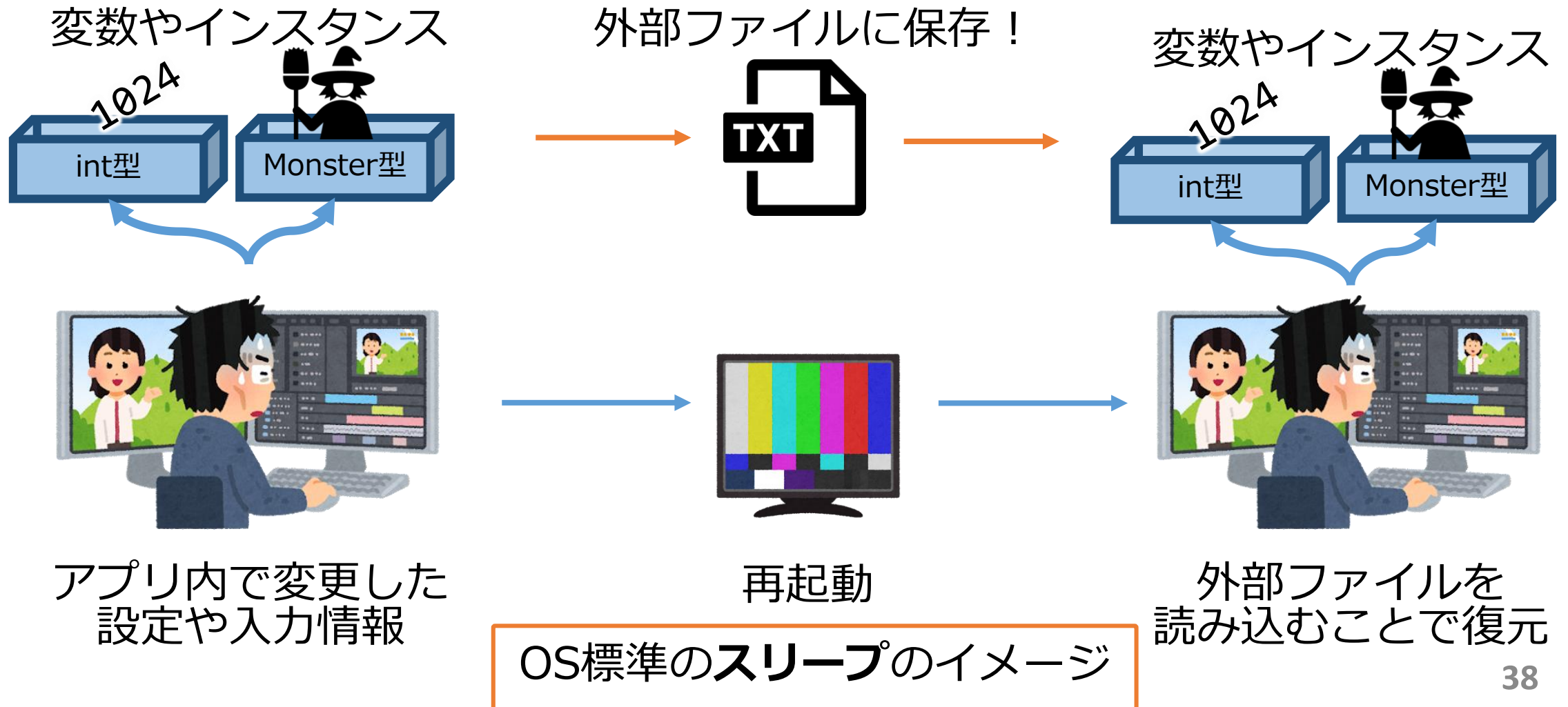
ファイル入出力

シリアライゼーション（バイナリ形式の入出力）



ファイル入出力

シリアライゼーション（バイナリ形式の入出力）



ファイル入出力

シリアライゼーション（バイナリ形式の入出力）



どうやってやるの？一つ一つStringでファイルに出力？
Monster型の場合どうするの？



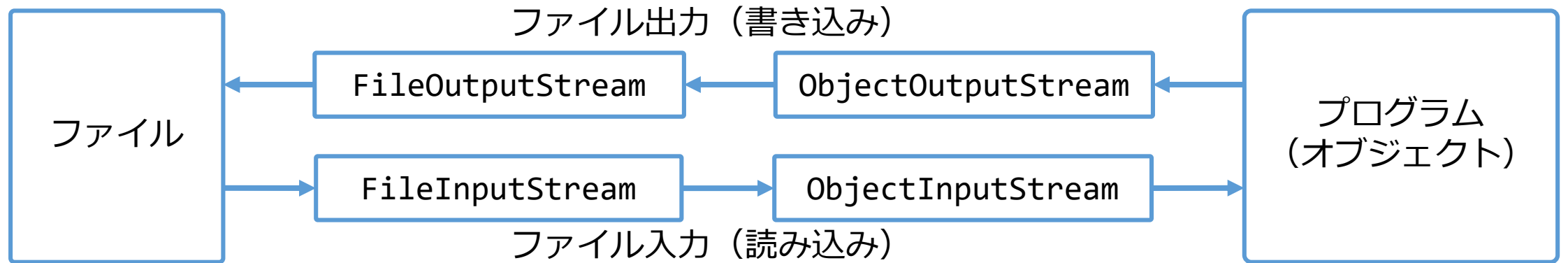
シリアライゼーション

プログラム内のオブジェクトをファイルで保存できる形にする（シリアライズする）こと。ただしString型を出力するのとは異なり、人間が内容を理解できない形式で出力される。

ファイル入出力

シリアライゼーション（バイナリ形式の入出力）

オブジェクトをファイル入出力するためには以下の4つのクラスを使用する.



基本的には，文字列出力のときと同じ.

ファイル入出力

シリアライゼーション（バイナリ形式の入出力）

使い方はほとんど同じ。仲介役のクラスを
FileOutputStreamとObjectOutputStreamにするだけ。

```
Monster m1 = new Monster("hasegawa", 100);
Monster m2 = new Monster("fukuma", 200);
try(FileOutputStream fs = new FileOutputStream("monster.ser");
    ObjectOutputStream os = new ObjectOutputStream(fs) ){
    os.writeObject(m1);
    os.writeObject(m2);
} catch(IOException e){
    e.printStackTrace();
}
```

try-with-resources構文

Monster型のインスタンスをwrite()する

ファイル入出力

シリアライゼーション (バイナリ形式の入出力)

このままでは例外(`NotSerializableException`)が発生
(`Monster`型はシリアライズできないよ！という例外)

```
java.io.NotSerializableException: Monster
at java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java:1184)
at java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java:348)
at Main.main(Main.java:28)
```



右のように`Monster`型に`Serializable`を
`implements`すると
シリアライズできる
ようになる。

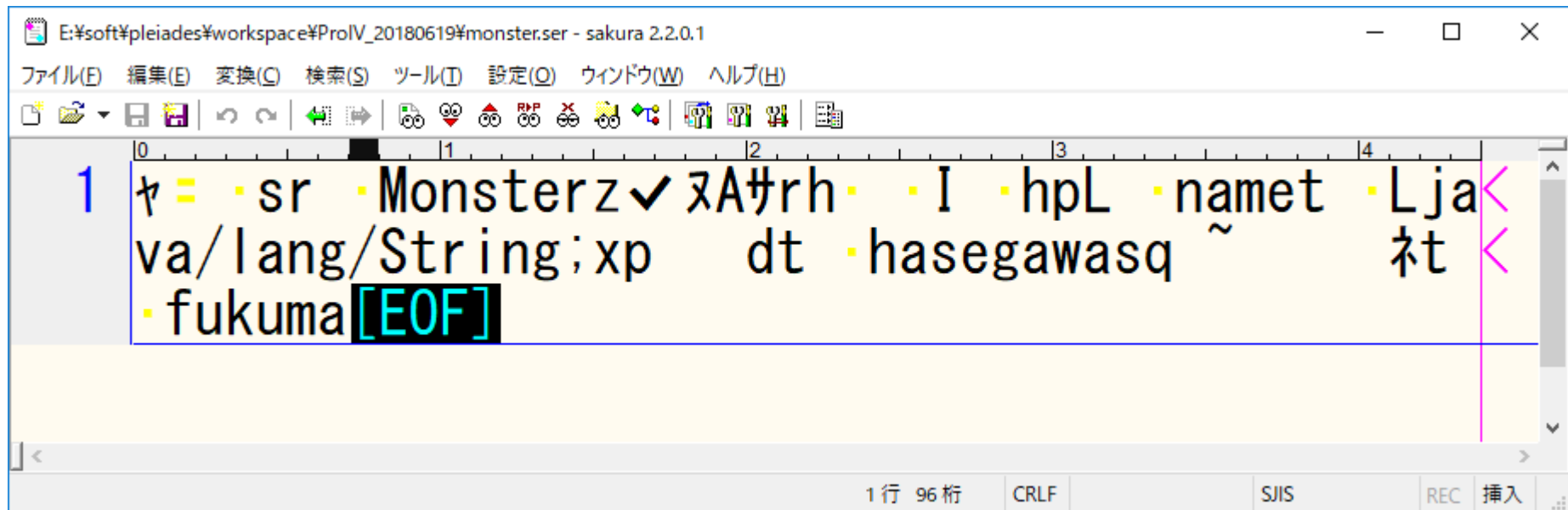
```
public class Monster implements Serializable {
    public String name="";
    public int hp = 0;
    public Monster(String name, int hp){
        this.name = name;
        this.hp = hp;
    }
}
```

メソッドのオーバーライド等は不要！

ファイル入出力

シリアライゼーション（バイナリ形式の入出力）

Monster型にSerializableをimplementsし、
monster.serに出力した結果.



```
E:\soft\pleiades\workspace\ProIV_20180619\monster.ser - sakura 2.2.0.1
ファイル(E) 編集(E) 変換(O) 検索(S) ツール(T) 設定(O) ウィンドウ(W) ヘルプ(H)
1 ヤ = .sr .Monsterz✓ヌAサrh . .I .hpL .namet .Lja<
va/lang/String;xp dt .hasegawasq ~ 社<
.fukuma[E0F]
```

1行 96桁 CRLF SJIS REC 挿入

ファイル入出力

シリアライゼーション (バイナリ形式の入出力)

出力したmonster.serを読み込んでみる.

```
try(FileInputStream fs = new FileInputStream("monster.ser");
    ObjectInputStream os = new ObjectInputStream(fs)) {
    Monster m1 = (Monster)os.readObject();
    Monster m2 = (Monster)os.readObject();
    System.out.println(m1.name + ":" + m1.hp);
    System.out.println(m2.name + ":" + m2.hp);
} catch (IOException e) {
    e.printStackTrace();
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

try-with-resources構文

Monster型のインスタンス
をread()する

hasegawa:100
fukuma:200

まとめ

- ファイルの入出力を行うクラスの使い方を学習した.
 - 書き込み : FileWriter, BufferedWriter

```
FileWriter fw = new FileWriter( [出力先ファイルパス] );  
BufferedWriter bw = new BufferedWriter( [FileWriterインスタンス(この場合 fw)] );  
bw.write( [出力する文字列] );  
bw.newLine();    // 適切な改行コードを自動で生成  
bw.close();
```

- 読み込み : FileReader, BufferedReader

```
FileReader fr = new FileReader( [入力元ファイルパス] );  
BufferedReader br = new BufferedReader( [FileReaderインスタンス(この場合 fr)] );  
String str = br.readLine();    // 1行単位で読み込む  
System.out.println(str);  
br.close();
```

まとめ

- Fileクラスを用いてファイル操作（ディレクトリ確認や生成）を用いた事前準備の手法を学習した.

```
public static void write(String name, String contents){  
    File file = new File(name);  
    if(!file.getParentFile().exists()) file.getParentFile().mkdirs();  
    if(file.exists()){ // 既にファイルが存在する場合  
        if(file.isDirectory()){  
            System.out.println("同名のディレクトリが存在するため別名で保存します. ");  
            write(file.getParent()+"¥¥"+System.currentTimeMillis()+".txt", contents);  
        }else if(file.isFile()){  
            System.out.println("同名のファイルが存在するため別名で保存します. ");  
            write(file.getParent()+"¥¥"+System.currentTimeMillis()+".txt", contents);  
        }  
    }  
}
```

まとめ

- try-with-resources文を用いてclose()を安全に省略する手法を学習した.

```
try(FileReader fr = new FileReader("file.txt")){  
    int ch;  
    while((ch = fr.read())!=-1){  
        System.out.print((char)ch);  
    }  
}catch(IOException e){  
    e.printStackTrace();  
}
```

1. try時にリソースを開くだけ

- オブジェクトをシリアライズしてファイルに保存しておく手法を学習した (**implements Serializable**) .

次週予告

※次週以降も計算機室

前半

ファイル入出力の使い方を学ぶ

後半

講義内容に関するプログラミング演習課題に取り組む。

本日の提出課題

講義パート

課題 1

本日の授業を聞いて、
よくわかったと思う内容を
2点簡潔に述べよ。

課題 2

本日の授業を聞いて、
質問事項または**気になった点**
を1点以上簡潔に述べよ。

課題 3

感想（あれば）

課題 4

なし

演習

- 昼休み, いつものWebページに演習問題をPDFで演習問題をアップロードする. 各自実施してプロII同様のWebページから提出すること.
- 質問は3人体制で受け付けるので遠慮なく申し出る. 質問の際は, どこまでわかっていて何がわからないのかを申し出ること.
- (ないとは思うが) コピペは発覚次第両成敗する.
 - ✓ {コピペ, カンニング} ∈不正行為
- つまらないミスも今回は問答無用で×とするので, 最終チェックを怠らないこと. (去年は目視で甘めに採点していたが, 自動採点を開発している意味がないので. . .)