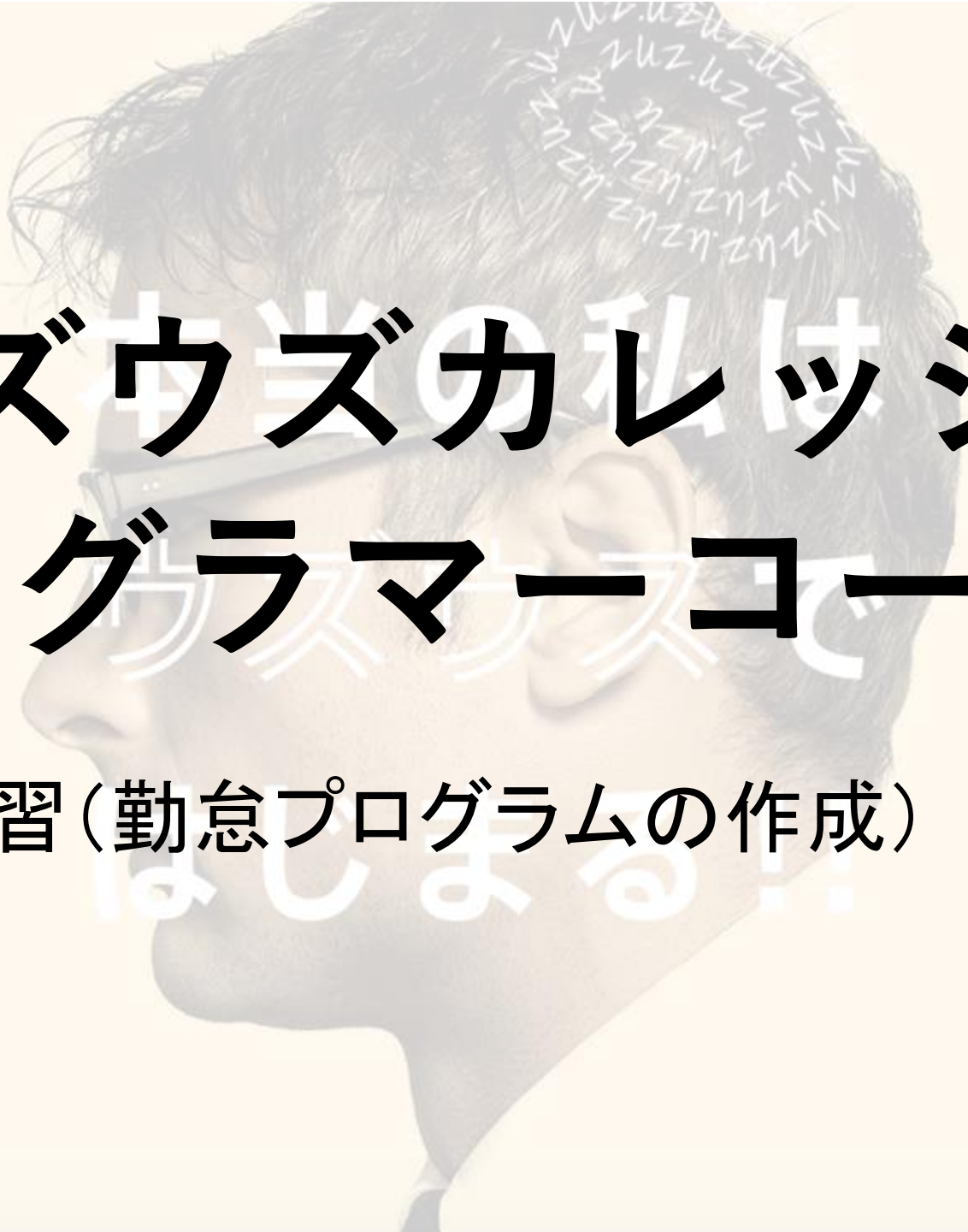
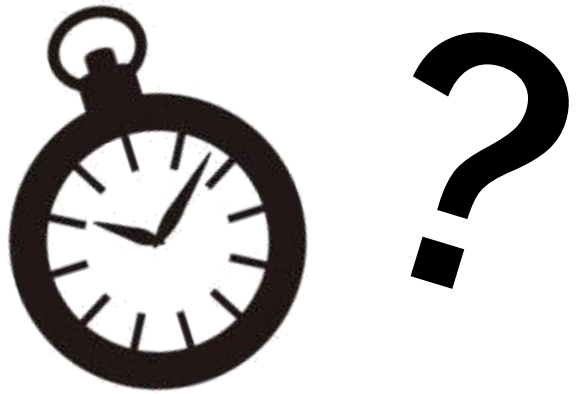


ウズウズカレッジ
プログラマーコース

演習（勤怠プログラムの作成）

【導入①】時刻データの扱い

問題を解く前に、APIを用いた時刻データの扱いについてイメージをつけましょう。
（ 使用するソースコード：CalcWorkingTime.java ）



```java
import java.sql.Time;

public class CalcWorkingTime {

    public static void main(String[] args) {

        // 計算用の数値を定数で用意
        final long ONE_HOUR_BY_MILLI_SEC = 1000 * 60 * 60; // 1時間のミリ秒換算
        final long ONE_MIN_BY_MILLI_SEC  = 1000 * 60;      // 1分のミリ秒換算
        final int  ONE_HOUR_BY_MIN       = 60;             // 1時間の分換算

        // バイトの開始時間と終了時間をコマンドライン引数から受け取る
        Time startTime  = Time.valueOf(args[0]);
        Time finishTime = Time.valueOf(args[1]);

        // getTimeメソッドを使って労働時間をミリ秒（0.001秒単位）で取得する
        // ※getTime()メソッドの戻り値はlong型であることに注意
        long workingTime = finishTime.getTime() - startTime.getTime();

        // ミリ秒で取得した労働時間を○時間△分の形式に直す
        int workingHour = (int)( workingTime / ONE_HOUR_BY_MILLI_SEC );                      // 時間に換算
        int workingMin  = (int)( ( workingTime / ONE_MIN_BY_MILLI_SEC ) % ONE_HOUR_BY_MIN ); // 分に換算

        // 出力
        System.out.println("本日の労働時間は" + workingHour + "時間" + workingMin + "分です。");
    }
}
```

```java
import java.sql.Time;
↵
public class CalcWorkingTime {↵
↵
    public static void main(String[] args) {↵
↵
        // 計算用の数値を定数で用意↵
        final long ONE_HOUR_BY_MILLI_SEC = 1000 * 60 * 60;
        final long ONE_MIN_BY_MILLI_SEC  = 1000 * 60;
        final int  ONE_HOUR_BY_MIN       = 60;
↵
        // バイトの開始時間と終了時間をコマンドライン引
        Time startTime  = Time.valueOf(args[0]);↵
        Time finishTime = Time.valueOf(args[1]);↵
↵
        // getTimeメソッドを使って労働時間をミリ秒（0.001秒
        // ※getTime()メソッドの戻り値はlong型であることに注
        long workingTime = finishTime.getTime() - startTime.getTime();
↵
        // ミリ秒で取得した労働時間を○時間△分の形式に直す↵
        int workingHour = (int)( workingTime / ONE_HOUR_BY_MILLI_SEC );                    // 時間に換算↵
        int workingMin  = (int)( ( workingTime / ONE_MIN_BY_MILLI_SEC ) % ONE_HOUR_BY_MIN ); // 分に換算↵
↵
        // 出力↵
        System.out.println("本日の労働時間は" + workingHour + "時間" + workingMin + "分です。");↵
    }↵
}↵
```

Timeクラスをインポート

出勤時刻を表す文字列（例：『08:00:00』）と
退勤時刻を表す文字列（例：『18:00:00』）を
コマンドライン引数として受け取り、
それぞれTimeオブジェクト（時刻情報を扱うデータ。現時点ではTime型という特殊な型の変数と思っていただければOK）に変換します。

変換したTimeオブジェクトには標準時刻（日本の場合は午前9時）からの経過時間がミリ秒（1000分の1秒）単位で格納されます。

『08:00:00』を入力　→　『-3600000』が格納される
『18:00:00』を入力　→　『32400000』が格納される

なお、Timeオブジェクトを利用するには「java.sql.Time」のインポートが必要です。

```java
import java.sql.Time;

public class CalcWorkingTime {

    public static void main(String[] args) {

        // 計算用の数値を定数で用意
        final long ONE_HOUR_BY_MILLI_SEC = 1000 * 60 * 60; // 1時間のミリ秒換算
        final long ONE_MIN_BY_MILLI_SEC  = 1000 * 60;      // 1分のミリ秒換算
        final int  ONE_HOUR_BY_MIN       = 60;             // 1時間の分換算

        // バイトの開始時間と終了時間をコマンドライン引数から受け取る
        Time startTime  = Time.valueOf(args[0]);
        Time finishTime = Time.valueOf(args[1]);

        // getTimeメソッドを使って労働時間をミリ秒（0.001秒単位）で取得する
        // ※getTime()メソッドの戻り値はlong型であることに注意
        long workingTime = finishTime.getTime() - startTime.getTime();

        // ミリ秒で取得した労働時間を○時間△分の形式に直す
        int workingHour = (int)( workingTime / ONE_HOUR_BY_MILLI_SEC );                    // 時間に換算
        int workingMin  = (int)( ( workingTime / ONE_MIN_BY_MILLI_SEC ) % ONE_HOUR_BY_MIN ); // 分に換算

        // 出力
        System.out.println("本日の労働時間は" + workingHour + "時間" + workingMin + "分です。");
    }
}
```

Timeオブジェクトは getTimeメソッドを使うことで格納されている時間データをlong型で取得できます。

finishTime（退勤時刻データ）からstartTime（出勤時刻データ）を差し引くことでその日の労働時間をミリ秒単位で取得することができます。

```java
import java.sql.Time;

public class CalcWorkingTime {

    public static void main(String[] args) {

        // 計算用の数値を定数で用意
        final long ONE_HOUR_BY_MILLI_SEC = 1000 * 60 * 60; // 1時間のミリ秒換算
        final long ONE_MIN_BY_MILLI_SEC  = 1000 * 60;      // 1分のミリ秒換算
        final int  ONE_HOUR_BY_MIN       = 60;             // 1時間の分換算

        // バイトの開始時間と終了時間をコマンドライン引数から受け取る
        Time startTime  = Time.valueOf(args[0]);
        Time finishTime = Time.valueOf(args[1]);

        // getTimeメソッドを使って労働時間をミリ秒（0.001秒単位）で取得する
        // ※getTime()メソッドの戻り値はlong型であることに注意
        long workingTime = finishTime.getTime() - startTime.getTime();

        // ミリ秒で取得した労働時間を○時間△分の形式に直す
        int workingHour = (int)( workingTime / ONE_HOUR_BY_MILLI_SEC );                          // 時間に換算
        int workingMin  = (int)( ( workingTime / ONE_MIN_BY_MILLI_SEC ) % ONE_HOUR_BY_MIN ); // 分に換算

        // 出力
        System.out.println("本日の労働時間は" + workingHour + "時間" + workingMin + "分です。");
    }
}
```

取得した労働時間（ミリ秒単位）を「○時間△分」の形に変換します。

「○時間」の部分は取得した労働時間をミリ秒から時間単位に直して端数を排除することで、
「△分」の部分は取得した労働時間を分単位に直して60で割った余りを求めることで、
それぞれ取得することが可能です。

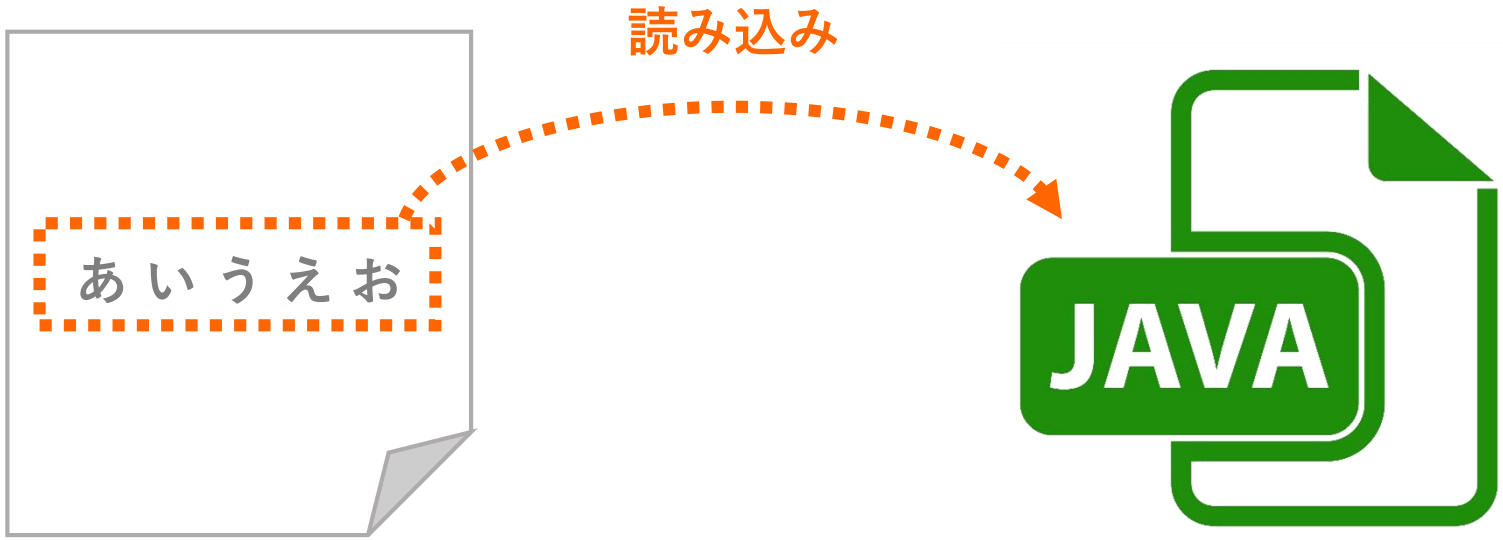
＜演習EX 1-3-1＞

『CalcWorkingTime.java』を参考に、この日の給与を算出して出力するプログラムを作成してください。

＜仕様＞
・時給は900円とし、給与は1分単位で支払われます。
・小数点以下の給与は切り捨てて算出されます。
・労働時間が6時間超～8時間以下の場合は45分の休憩を、8時間を超える場合は1時間の休憩をとるものとします。なお、休憩時間において給与は発生しません。
・休憩時間を差し引いた労働時間（実労働時間）が8時間を超える場合、超過分に限り残業代として1.25倍の給与が支払われるものとします。
・BigDecimalによる誤差の考慮は不要とします。
・クラス名は自由に決めてください。
（必ずクラス名を見ただけで処理内容を想像できるものにすること）

【導入②】外部ファイルの読み込み

問題を解く前に、まずはAPIを用いた外部ファイルの読み込みについてイメージをつけましょう。
（使用するソースコード：ReadFileSample.java ／ WorkingResult.csv ）



```
//  WorkingResult.csvのパス ※「C:\WorkSpace」直下に配置していない場合は適宜変更してください。
final String WORKING_RESULT_FILE_PATH = "C:\\WorkSpace\\WorkingResult.csv";
// コンマ
final String COMMA = ","; 

// 計算用の数値を定数で用意
final long ONE_HOUR_BY_MILLI_SEC = 1000 * 60 * 60; // 1
final long ONE_MIN_BY_MILLI_SEC  = 1000 * 60;      // 1
final int  ONE_HOUR_BY_MIN       = 60;             // 1

List<String> workingResults = new ArrayList<String>(); 

//  WorkingResult.csvを読み込む
try {
    // WorkingResult.csvの読み込み準備
    File workingResultFile = new File(WORKING_RESULT_FILE_PATH);
    BufferedReader br = new BufferedReader(new FileReader(workingResultFile));
    
    // WorkingResult.csvを1行ずつ読み込んでArrayListに格納する
    String recode = br.readLine();
    while (recode != null) {
        workingResults.add(recode);
        recode = br.readLine();
    }
    br.close();
} catch (IOException e) {
    System.out.println(e);
}
```

今回読み込む CSVファイル と呼ばれる外部ファイルです。
（ファイルの拡張子が『 .csv 』のファイル）

CSVは『Comma Separated Value』の略で、複数の値を『 , 』（カンマ）で
区切った表形式のデータを扱います。

```java
//  WorkingResult.csvのパス ※「C:\WorkSpace」直下に配置していない場合は適宜変更してください。
final String WORKING_RESULT_FILE_PATH = "C:\\WorkSpace\\WorkingResult.csv";
// コンマ
final String COMMA = ","; 

// 計算用の数値を定数で用意
final long ONE_HOUR_BY_MILLI_SEC = 1000 * 60 * 60; // 1時間のミリ秒換算
final long ONE_MIN_BY_MILLI_SEC  = 1000 * 60;      // 1分のミリ秒換算
final int  ONE_HOUR_BY_MIN       = 60;             // 1時間の分換算

List<String> workingResults = new ArrayList<String>(); //ファイルから読み込んだデータの格納用

//  WorkingResult.csvを読み込む
try {
    // WorkingResult.csvの読み込み準備
    File workingResultFile = new File(WORKING_RESULT_FILE_PATH);
    BufferedReader br = new BufferedReader(new FileReader(workingResultFile));

    // WorkingResult.csvを1行ずつ読み込んでArrayListに格納する
    String recode = br.readLine();
    while (recode != null) {
        workingResults.add(recode);
        recode = br.readLine();
    }
    br.close();
} catch (IOException e) {
    System.out.println(e);
}
```

プログラムからCSVファイルにアクセスし、1行分ずつ文字列を
読み込んで順次ArrayListに格納していきます。

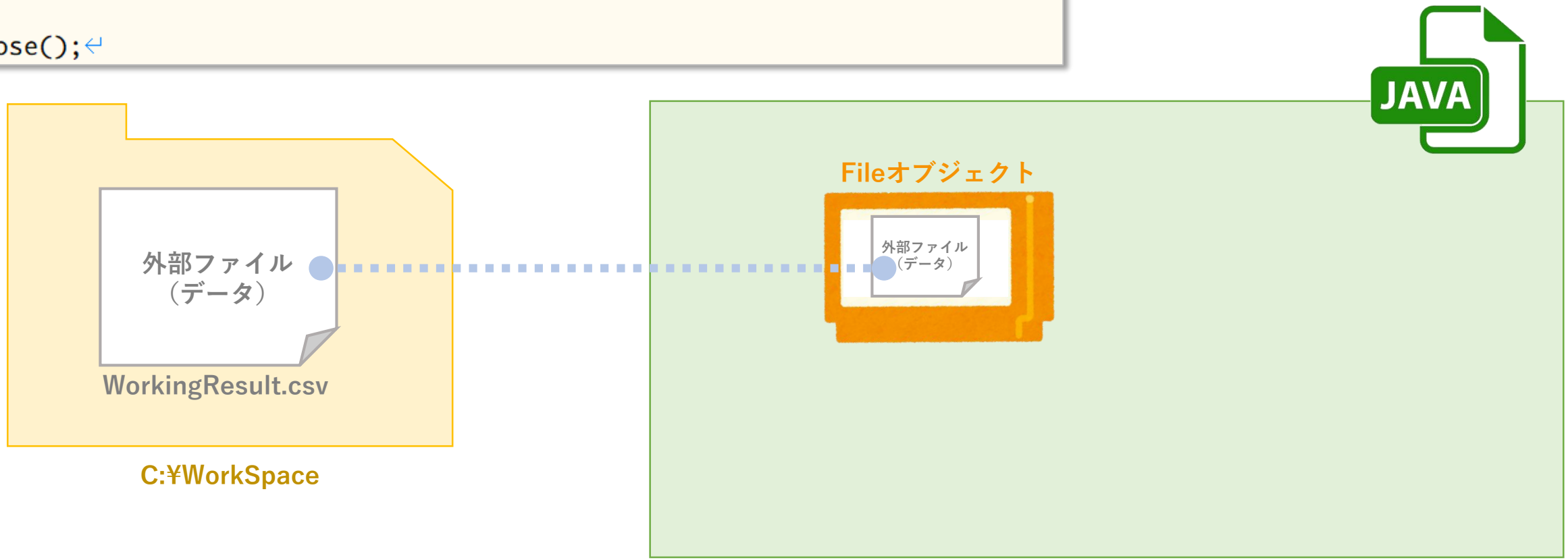
読み込みにはAPIクラス『File』『FileReader』『BufferedReader』が
必要になります。

これらの使用には以下のインポートが必要になります。
- java.io.BufferedReader
- java.io.File
- java.io.FileReader

また、これらを使用するコードは
『try{』と『}catch(IOException e){ System.out.println(e); }』
で囲う必要があります。（詳しくはオブジェクト指向編で学習します）

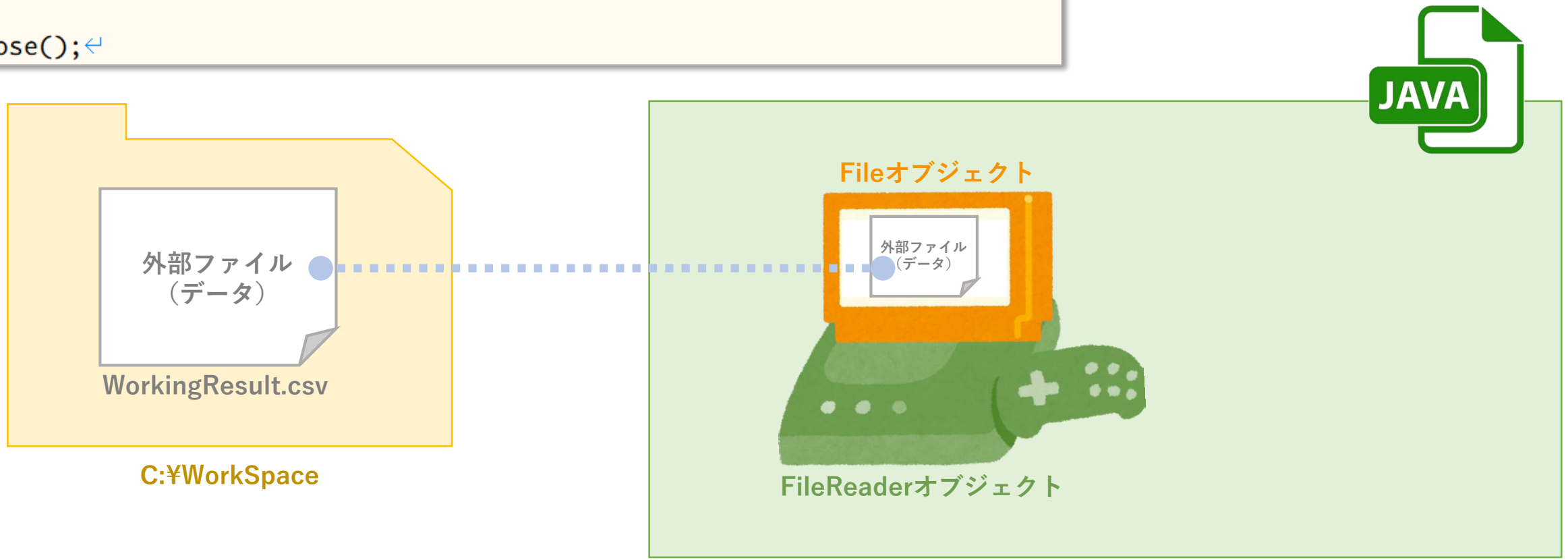
```
//  WorkingResult.csvのパス ※「C:\WorkSpace」直下に配置していない場合は適宜変更してください。
final String WORKING_RESULT_FILE_PATH = "C:\\WorkSpace\\WorkingResult.csv";
```

```
// WorkingResult.csvの読み込み準備
File workingResultFile = new File(WORKING_RESULT_FILE_PATH);
BufferedReader br = new BufferedReader(new FileReader(workingResultFile));

// WorkingResult.csvを1行ずつ読み込んでArrayListに格納する
String recode = br.readLine();
while (recode != null) {
    workingResults.add(recode);
    recode = br.readLine();
}
br.close();
```



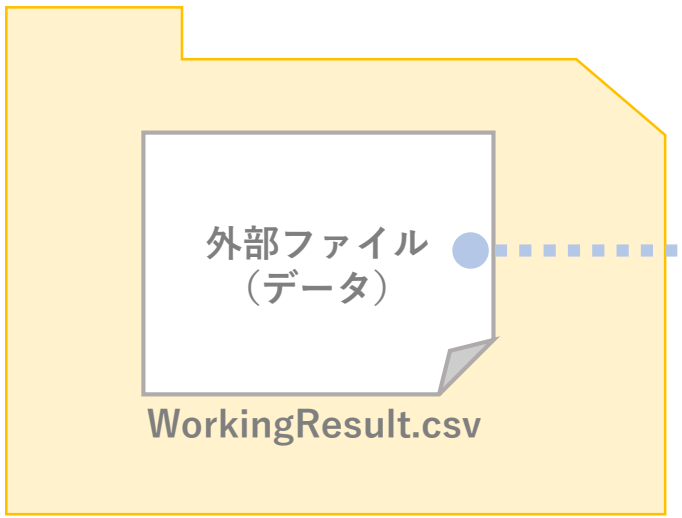
```
//  WorkingResult.csvのパス ※「C:\WorkSpace」直下に配置していない場合は適宜変更してください。
final String WORKING_RESULT_FILE_PATH = "C:\\WorkSpace\\WorkingResult.csv";
```

```
// WorkingResult.csvの読み込み準備
File workingResultFile = new File(WORKING_RESULT_FILE_PATH);
BufferedReader br = new BufferedReader(new FileReader(workingResultFile));

// WorkingResult.csvを1行ずつ読み込んでArrayListに格納する
String recode = br.readLine();
while (recode != null) {
    workingResults.add(recode);
    recode = br.readLine();
}
br.close();
```



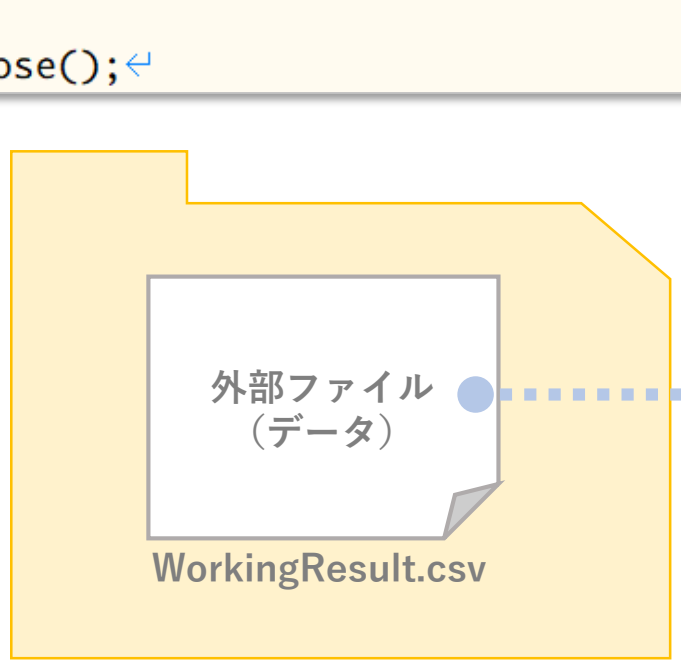
```
//  WorkingResult.csvのパス ※「C:\WorkSpace」直下に配置していない場合は適宜変更してください。
final String WORKING_RESULT_FILE_PATH = "C:\\WorkSpace\\WorkingResult.csv";
```

```
// WorkingResult.csvの読み込み準備
File workingResultFile = new File(WORKING_RESULT_FILE_PATH);
BufferedReader br = new BufferedReader(new FileReader(workingResultFile));

// WorkingResult.csvを1行ずつ読み込んでArrayListに格納する
String recode = br.readLine();
while (recode != null) {
    workingResults.add(recode);
    recode = br.readLine();
}
br.close();
```



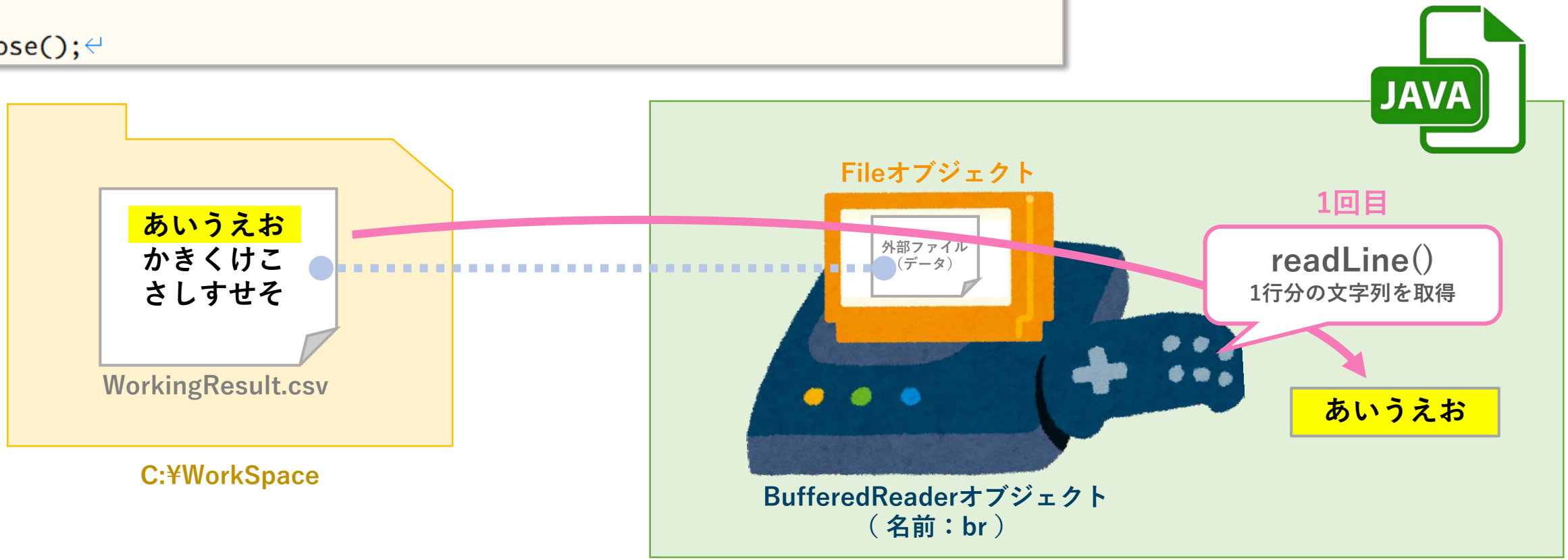
```
//  WorkingResult.csvのパス ※「C:\WorkSpace」直下に配置していない場合は適宜変更してください。
final String WORKING_RESULT_FILE_PATH = "C:\\WorkSpace\\WorkingResult.csv";
```

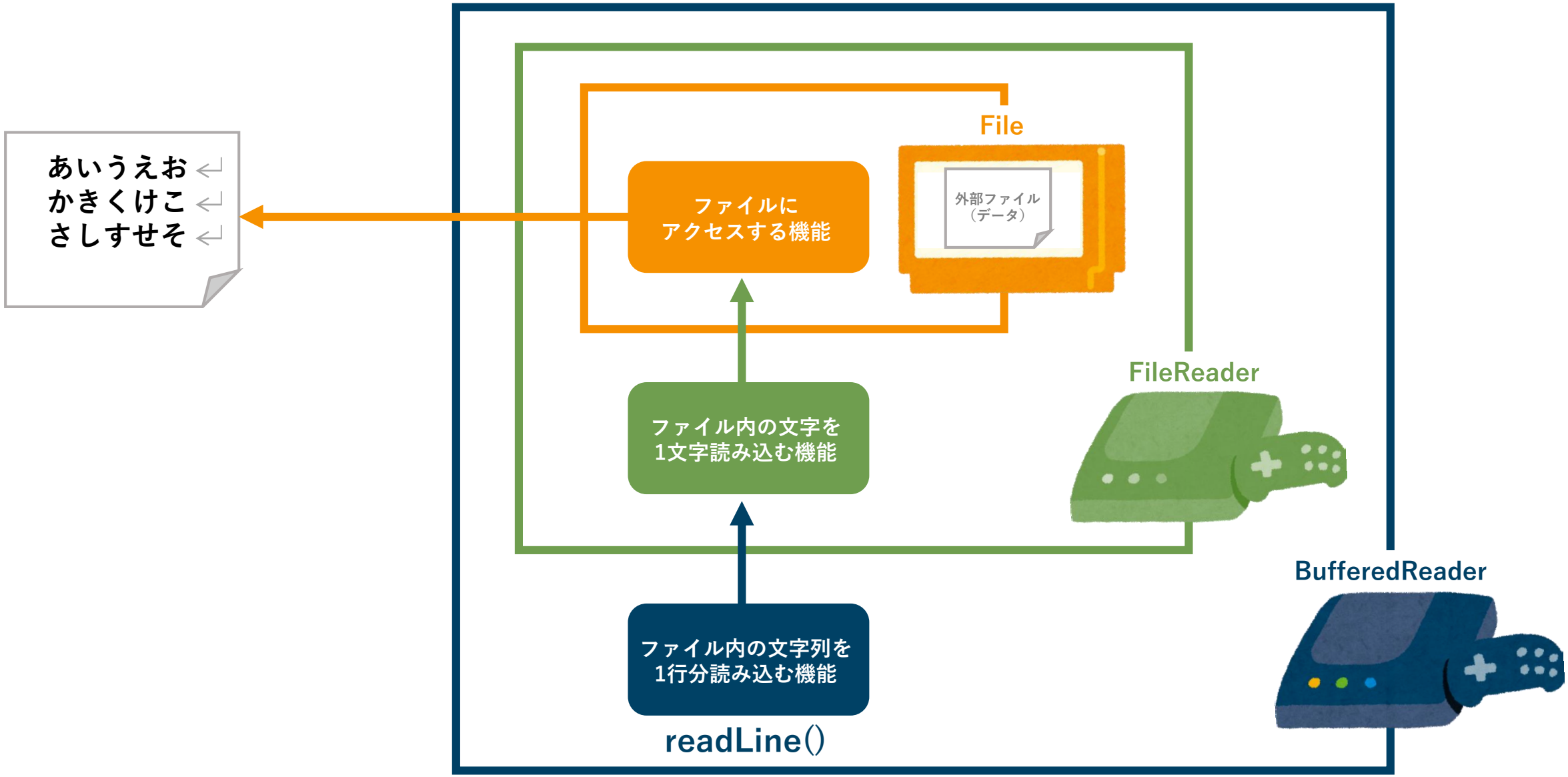
```
// WorkingResult.csvの読み込み準備
File workingResultFile = new File(WORKING_RESULT_FILE_PATH);
BufferedReader br = new BufferedReader(new FileReader(workingResultFile));

// WorkingResult.csvを1行ずつ読み込んでArrayListに格納する
String recode = br.readLine();
while (recode != null) {
    workingResults.add(recode);
    recode = br.readLine();
}
br.close();
```

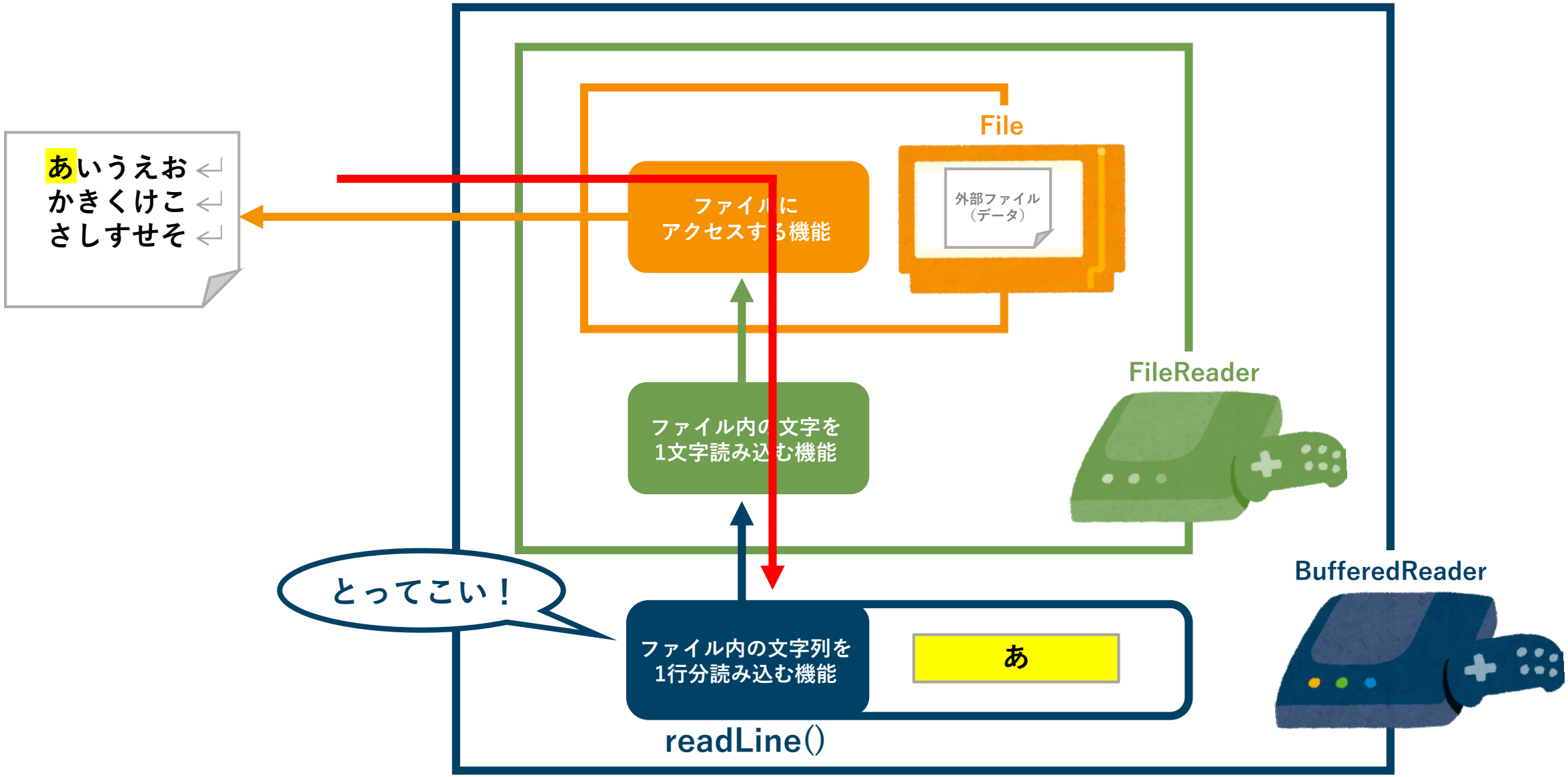


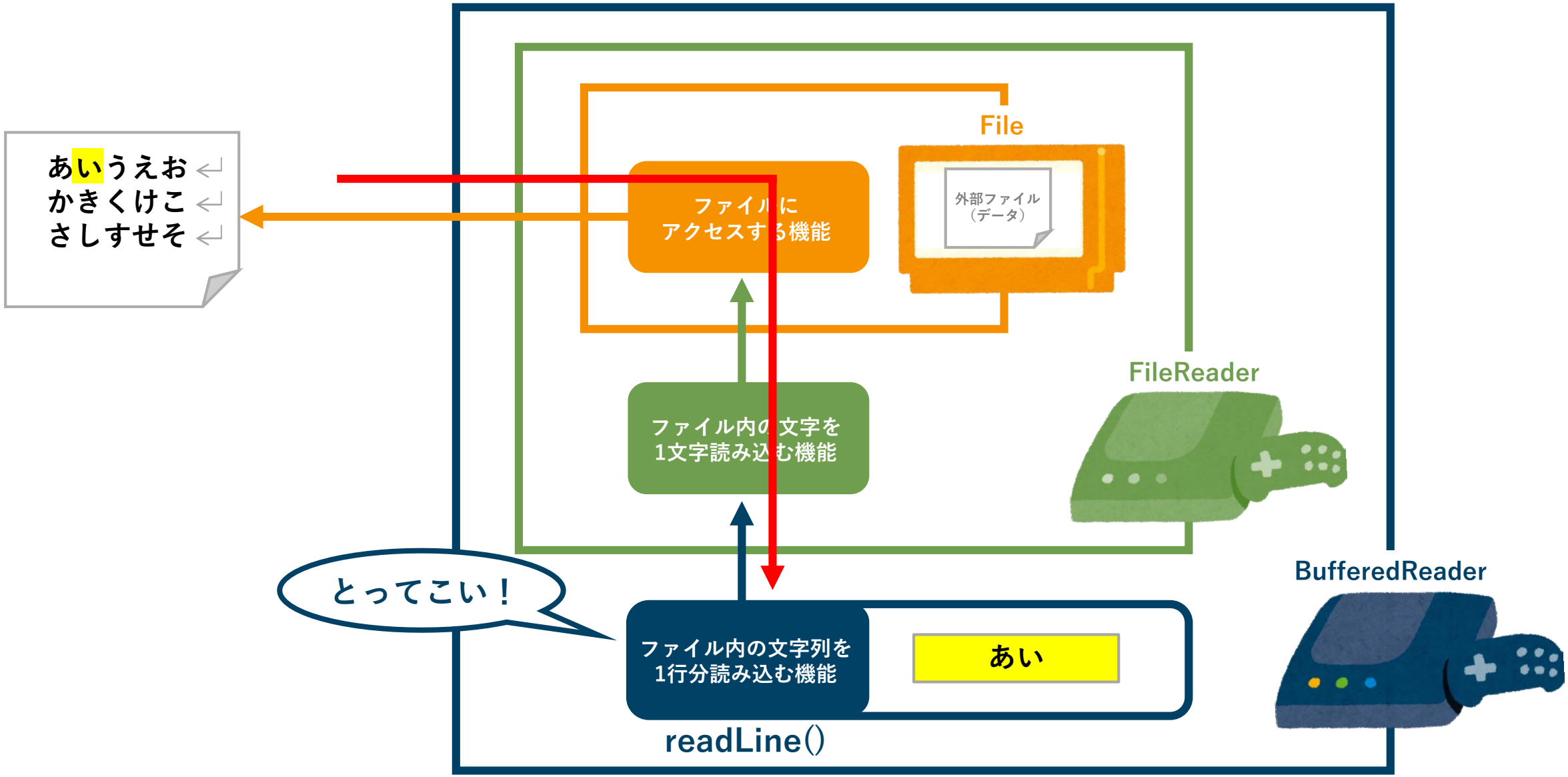
```java
//  WorkingResult.csvのパス ※「C:\WorkSpace」直下に配置していない場合は適宜変更してください。
final String WORKING_RESULT_FILE_PATH = "C:\\WorkSpace\\WorkingResult.csv";
```

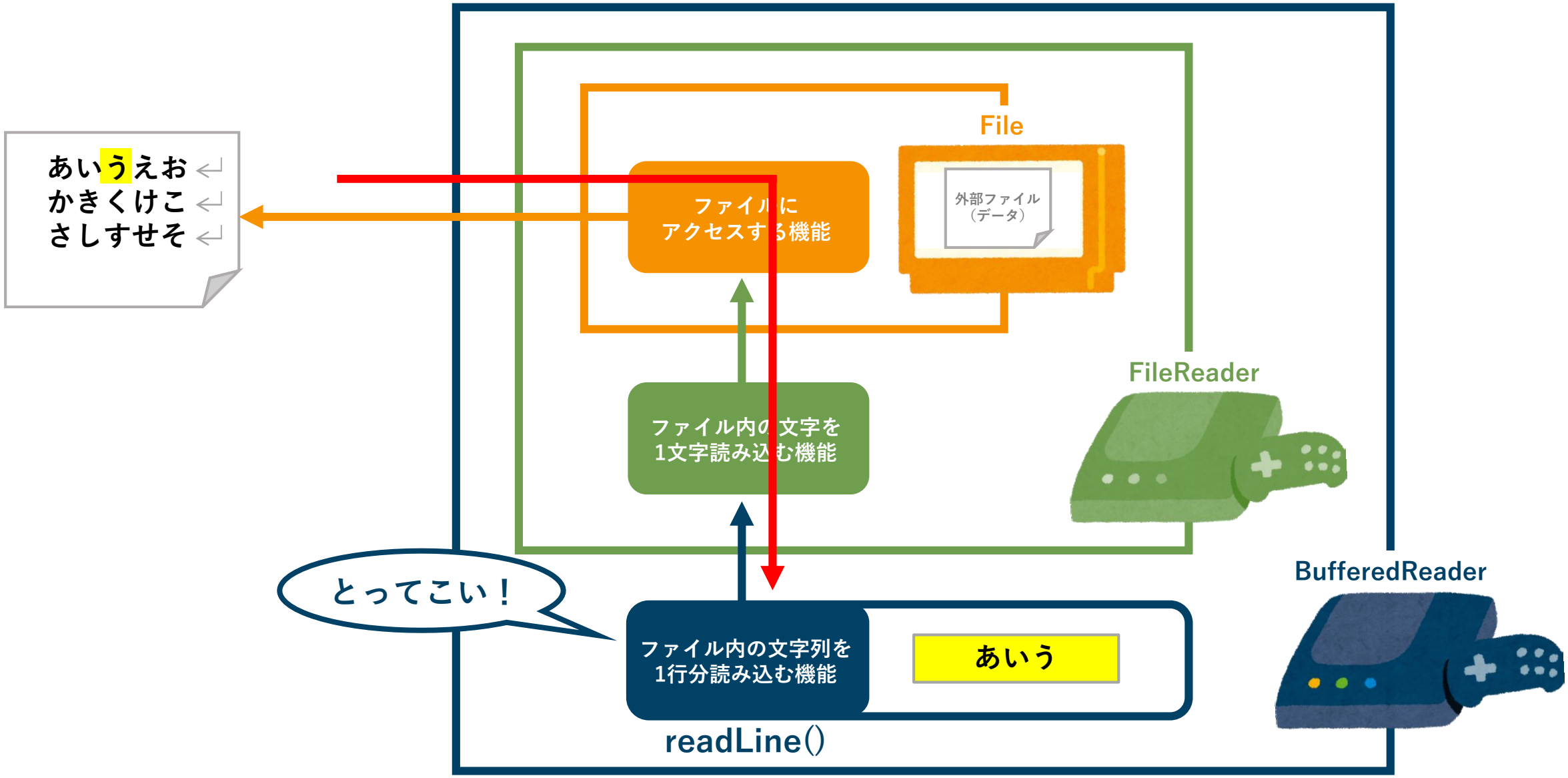
```java
// WorkingResult.csvの読み込み準備
File workingResultFile = new File(WORKING_RESULT_FILE_PATH);
BufferedReader br = new BufferedReader(new FileReader(workingResultFile));

// WorkingResult.csvを1行ずつ読み込んでArrayListに格納する
String recode = br.readLine();
while (recode != null) {
    workingResults.add(recode);
    recode = br.readLine();
}
br.close();
```

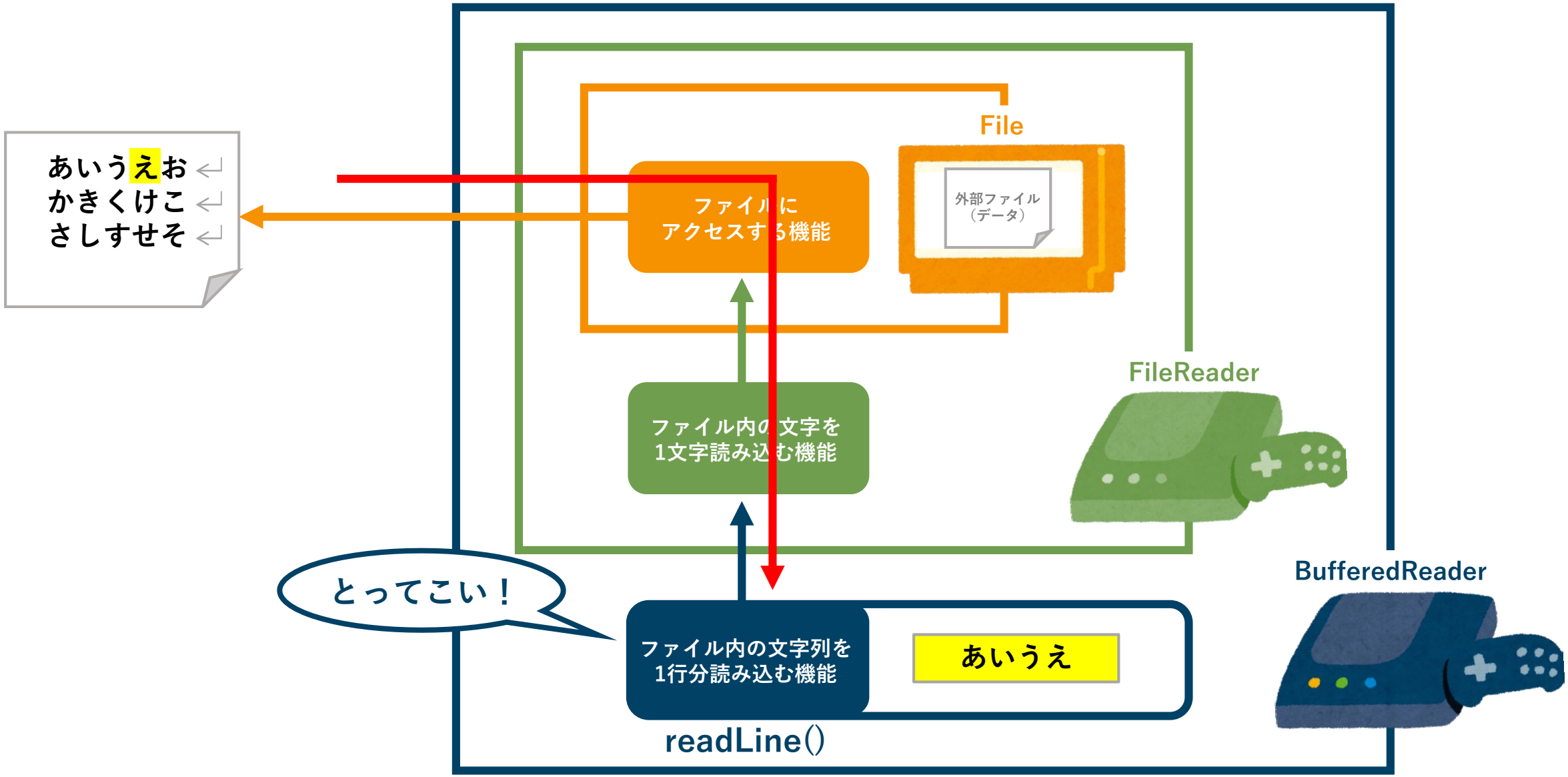


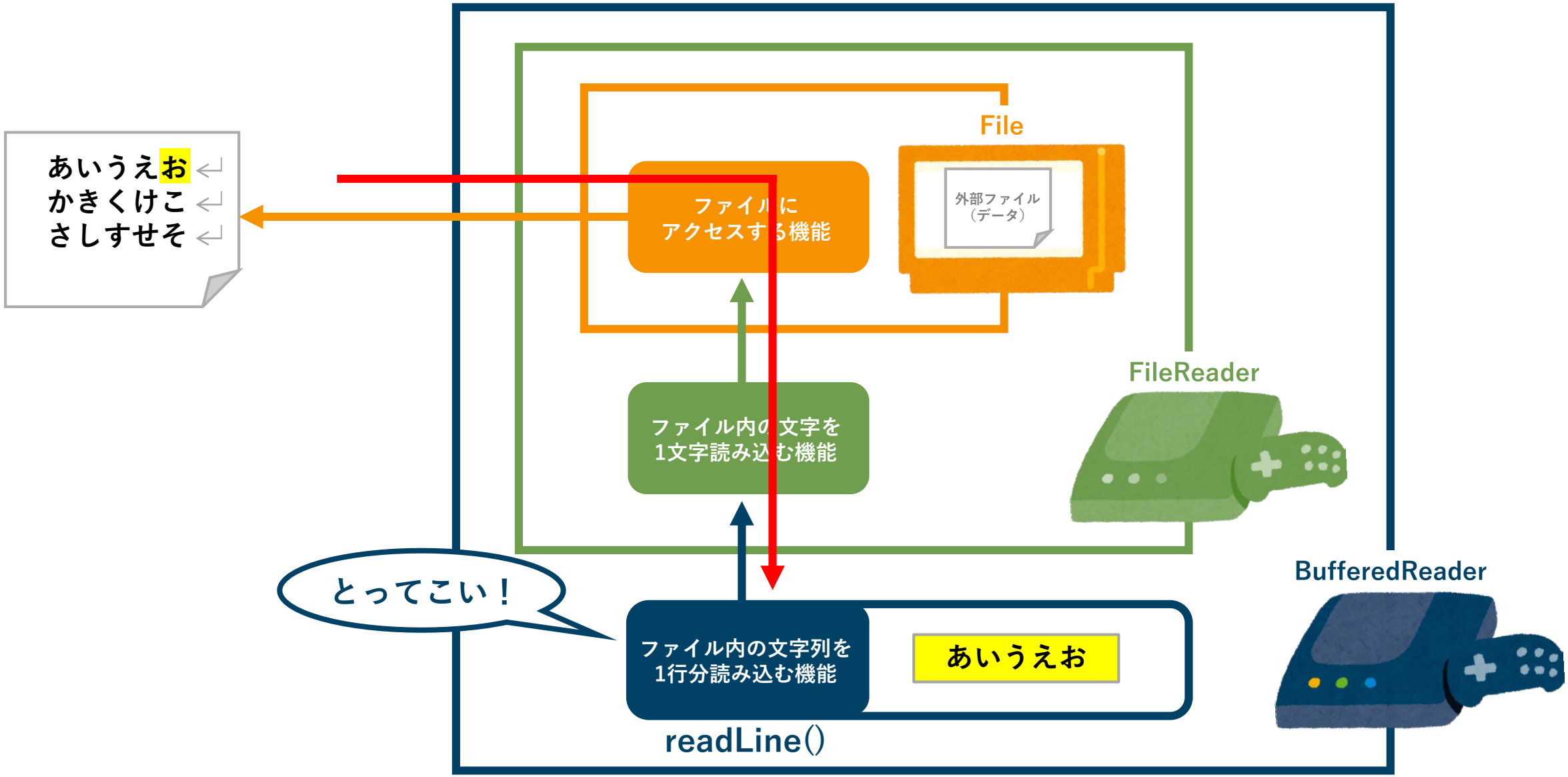

あいうえお
かきくけこ
さしすせそ
File
外部ファイル
（データ）
ファイルに
アクセスする機能
FileReader
ファイル内の文字を
1文字読み込む機能
BufferedReader
ファイル内の文字列を
1行分読み込む機能
readLine()

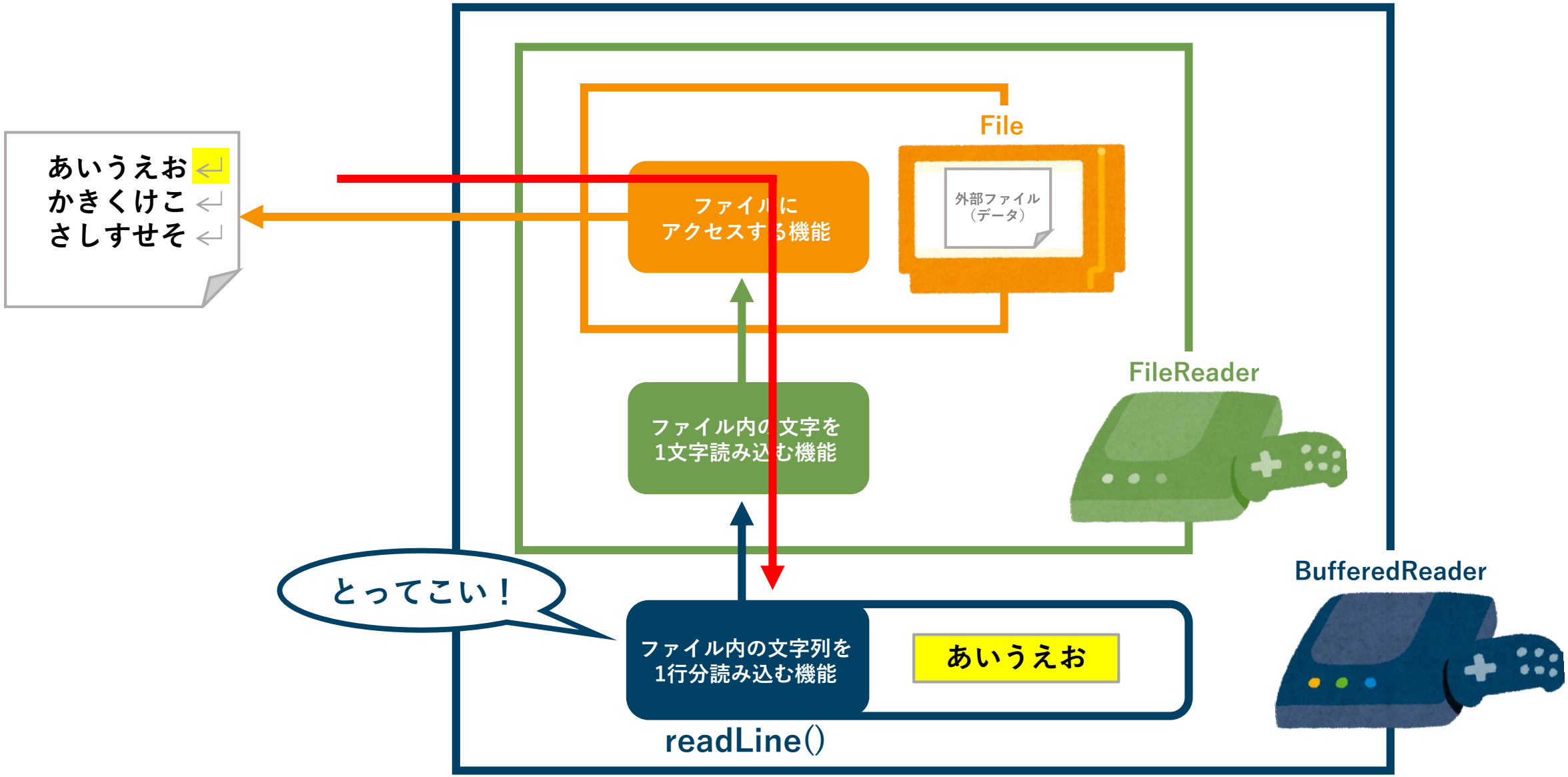

あいうえお
かきくけこ
さしすせそ
File
外部ファイル
（データ）
ファイルに
アクセスする機能
FileReader
ファイル内の文字を
1文字読み込む機能
とってこい！
BufferedReader
ファイル内の文字列を
1行分読み込む機能
あ
readLine()

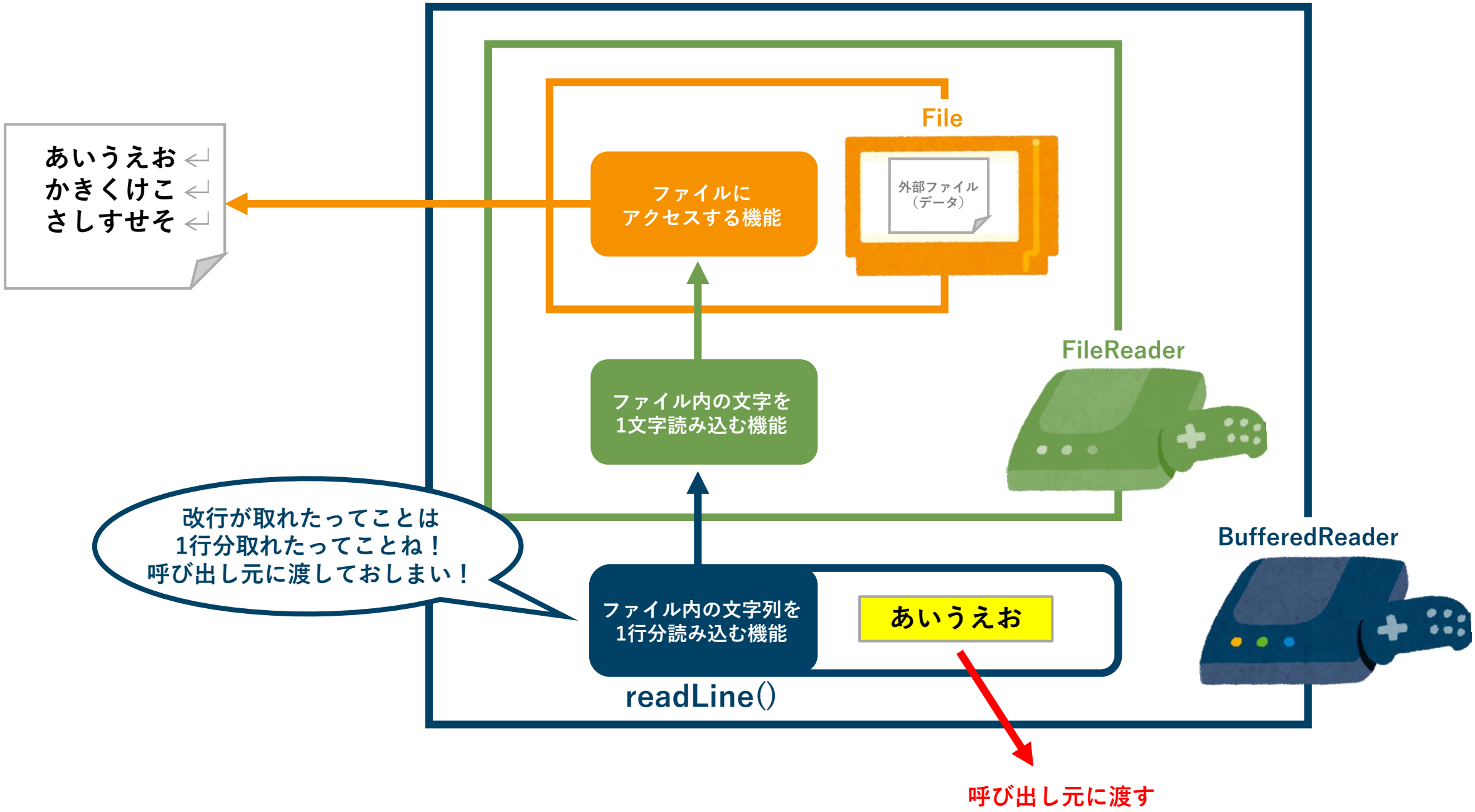

あいうえお
かきくけこ
さしすせそ
File
外部ファイル
（データ）
ファイルに
アクセスする機能
FileReader
ファイル内の文字を
1文字読み込む機能
とってこい！
BufferedReader
ファイル内の文字列を
1行分読み込む機能
あい
readLine()


あいうえお
かきくけこ
さしすせそ
File
外部ファイル
（データ）
ファイルに
アクセスする機能
FileReader
ファイル内の文字を
1文字読み込む機能
とってこい！
ファイル内の文字列を
1行分読み込む機能
あいう
BufferedReader
readLine()


あいうえお
かきくけこ
さしすせそ
File
外部ファイル
（データ）
ファイルに
アクセスする機能
FileReader
ファイル内の文字を
1文字読み込む機能
とってこい！
BufferedReader
ファイル内の文字列を
1行分読み込む機能
あいうえ
readLine()

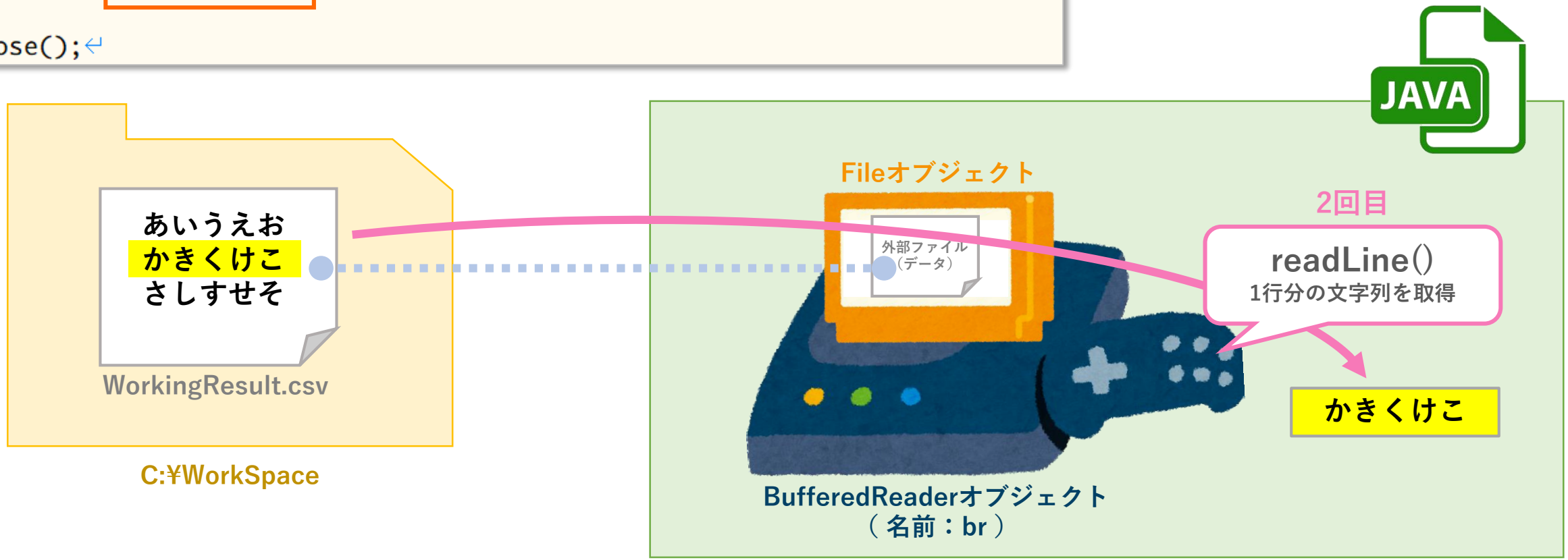

あいうえお
かきくけこ
さしすせそ
File
外部ファイル
（データ）
ファイルに
アクセスする機能
FileReader
ファイル内の文字を
1文字読み込む機能
とってこい！
BufferedReader
ファイル内の文字列を
1行分読み込む機能
あいうえお
readLine()


あいうえお
かきくけこ
さしすせそ
File
外部ファイル
（データ）
ファイルに
アクセスする機能
FileReader
ファイル内の文字を
1文字読み込む機能
とってこい！
BufferedReader
ファイル内の文字列を
1行分読み込む機能
あいうえお
readLine()


あいうえお
かきくけこ
さしすせそ
File
外部ファイル
（データ）
ファイルに
アクセスする機能
FileReader
ファイル内の文字を
1文字読み込む機能
改行が取れたってことは
1行分取れたってことね！
呼び出し元に渡しておしまい！
BufferedReader
ファイル内の文字列を
1行分読み込む機能
あいうえお
readLine()
呼び出し元に渡す

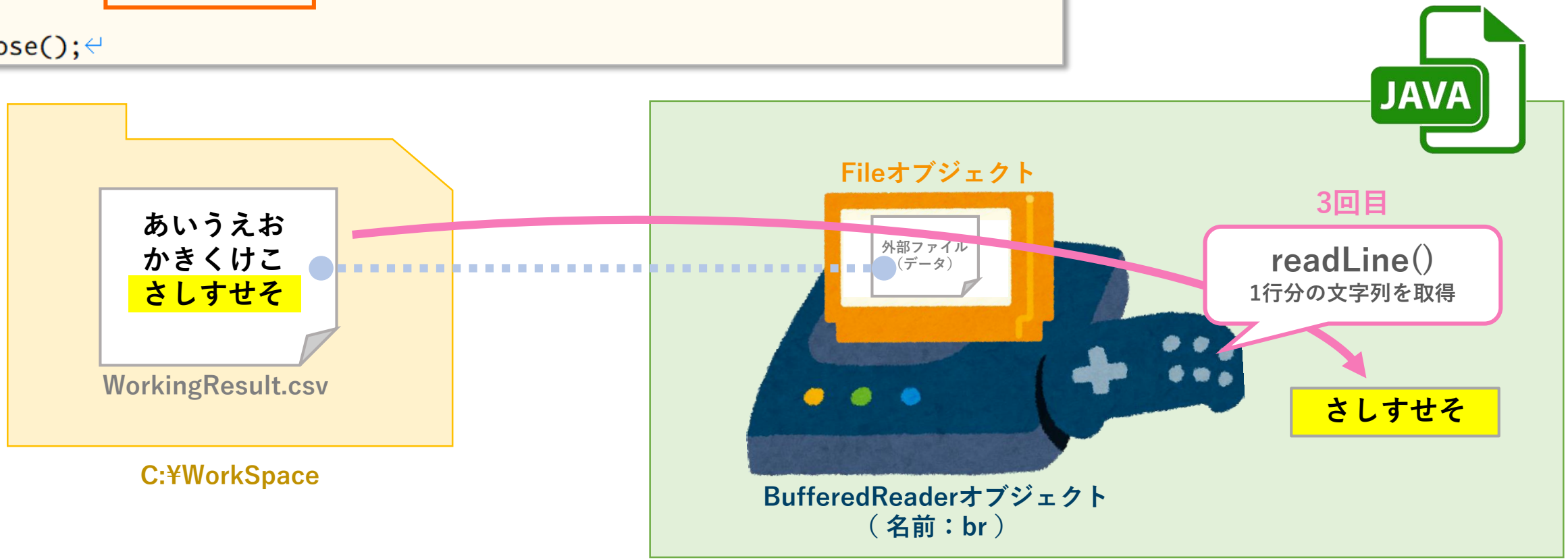
```
//  WorkingResult.csvのパス ※「C:\WorkSpace」直下に配置していない場合は適宜変更してください。
final String WORKING_RESULT_FILE_PATH = "C:\\WorkSpace\\WorkingResult.csv";
```

```
// WorkingResult.csvの読み込み準備
File workingResultFile = new File(WORKING_RESULT_FILE_PATH);
BufferedReader br = new BufferedReader(new FileReader(workingResultFile));

// WorkingResult.csvを1行ずつ読み込んでArrayListに格納する
String recode = br.readLine();
while (recode != null) {
    workingResults.add(recode);
    recode = br.readLine();
}
br.close();
```



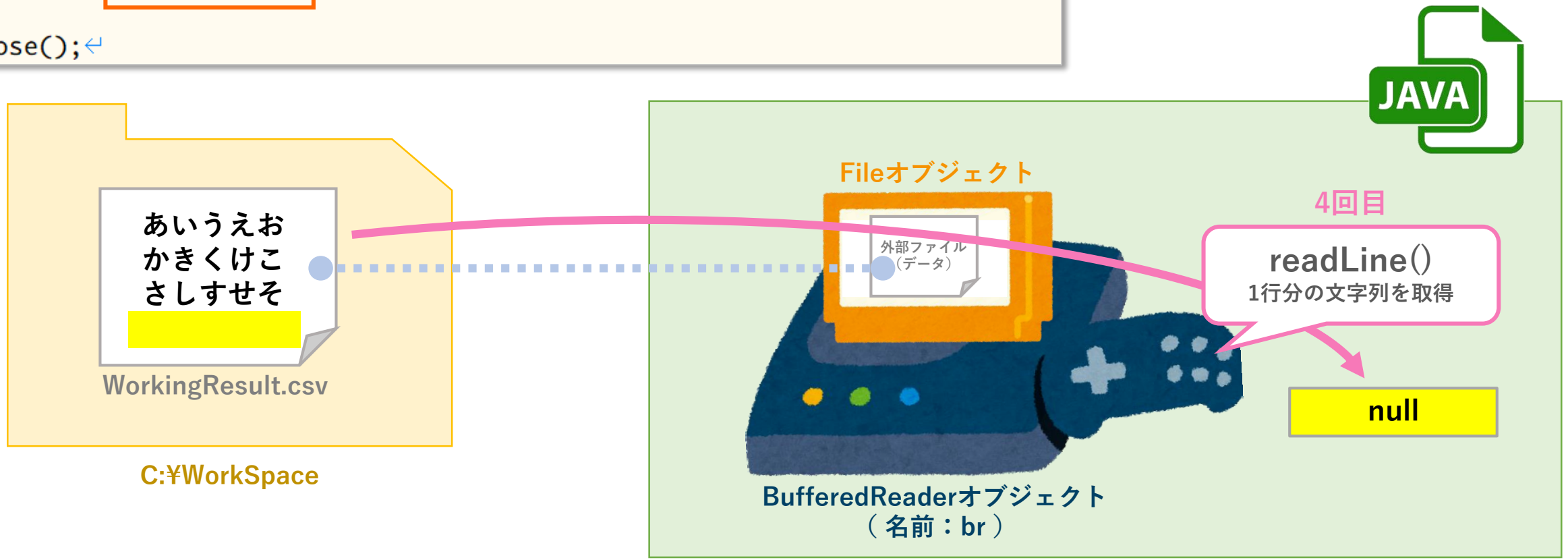
```
//  WorkingResult.csvのパス ※「C:\WorkSpace」直下に配置していない場合は適宜変更してください。
final String WORKING_RESULT_FILE_PATH = "C:\\WorkSpace\\WorkingResult.csv";
```

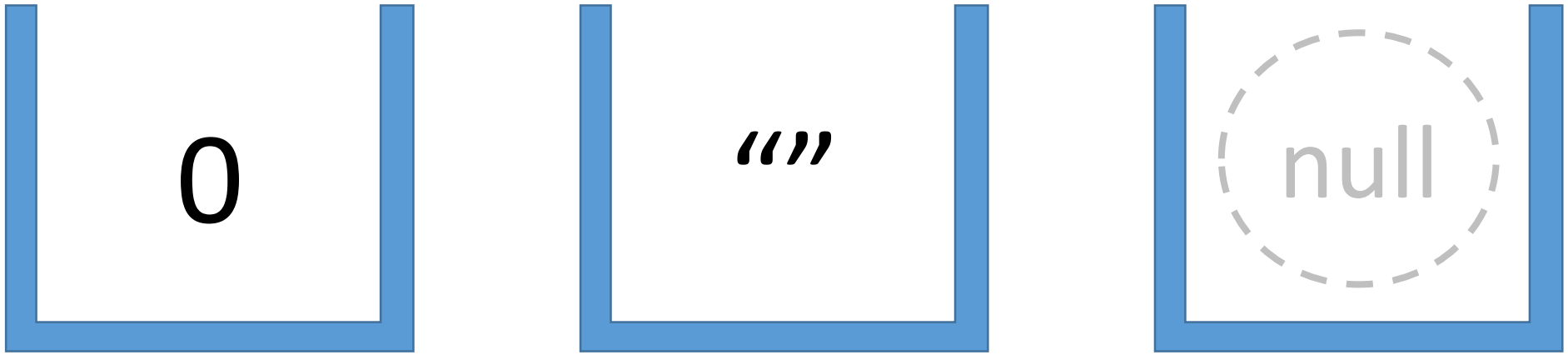
```
// WorkingResult.csvの読み込み準備
File workingResultFile = new File(WORKING_RESULT_FILE_PATH);
BufferedReader br = new BufferedReader(new FileReader(workingResultFile));

// WorkingResult.csvを1行ずつ読み込んでArrayListに格納する
String recode = br.readLine();
while (recode != null) {
    workingResults.add(recode);
    recode = br.readLine();
}
br.close();
```



```java
//  WorkingResult.csvのパス ※「C:\WorkSpace」直下に配置していない場合は適宜変更してください。
final String WORKING_RESULT_FILE_PATH = "C:\\WorkSpace\\WorkingResult.csv";
```

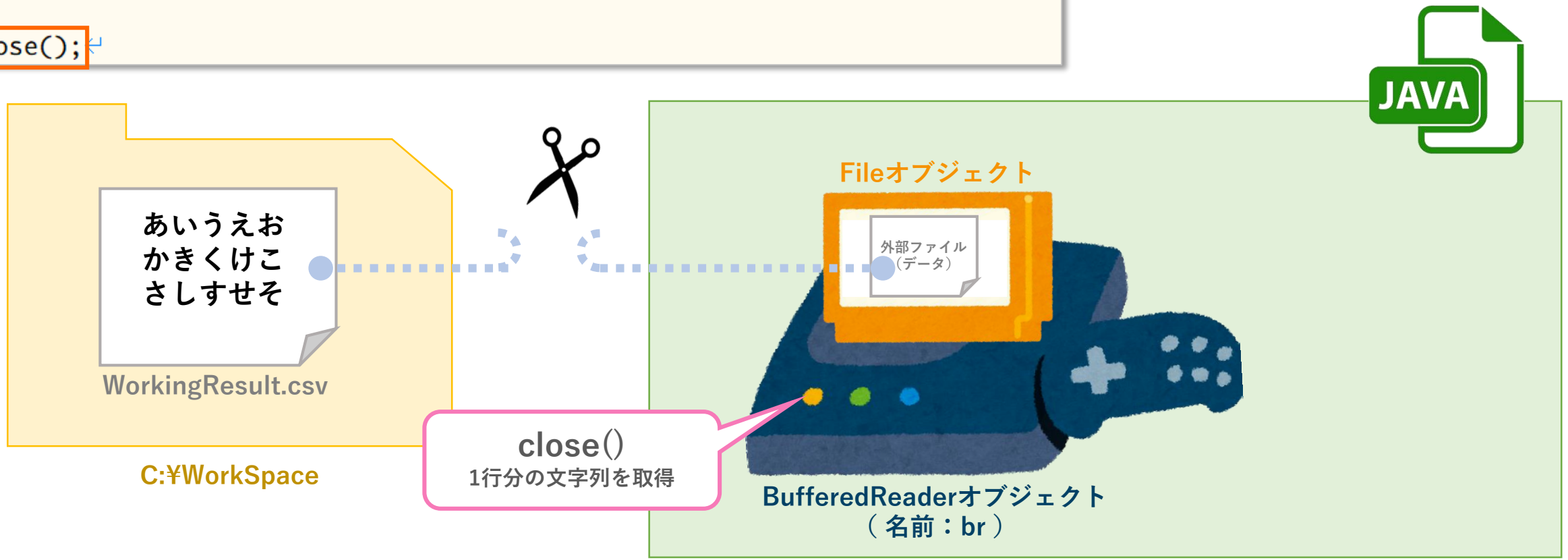
```java
// WorkingResult.csvの読み込み準備
File workingResultFile = new File(WORKING_RESULT_FILE_PATH);
BufferedReader br = new BufferedReader(new FileReader(workingResultFile));

// WorkingResult.csvを1行ずつ読み込んでArrayListに格納する
String recode = br.readLine();
while (recode != null) {
    workingResults.add(recode);
    recode = br.readLine();
}
br.close();
```




0
“”
null

```java
//  WorkingResult.csvのパス ※「C:\WorkSpace」直下に配置していない場合は適宜変更してください。
final String WORKING_RESULT_FILE_PATH = "C:\\WorkSpace\\WorkingResult.csv";
```

```java
// WorkingResult.csvの読み込み準備
File workingResultFile = new File(WORKING_RESULT_FILE_PATH);
BufferedReader br = new BufferedReader(new FileReader(workingResultFile));

// WorkingResult.csvを1行ずつ読み込んでArrayListに格納する
String recode = br.readLine();
while (recode != null) {
    workingResults.add(recode);
    recode = br.readLine();
}
br.close();
```



```java
//  WorkingResult.csvのパス ※「C:\WorkSpace」直下に配置していない場合は適宜変更してください。
final String WORKING_RESULT_FILE_PATH = "C:\\WorkSpace\\WorkingResult.csv";
// コンマ
final String COMMA = ","; 

// 計算用の数値を定数で用意
final long ONE_HOUR_BY_MILLI_SEC = 1000 * 60 * 60; // 1時間のミリ秒換算
final long ONE_MIN_BY_MILLI_SEC  = 1000 * 60;      // 1分のミリ秒換算
final int  ONE_HOUR_BY_MIN       = 60;             // 1時間の分換算

List<String> workingResults = new ArrayList<String>(); //ファイルから読み込んだデータの格納用

//  WorkingResult.csvを読み込む
try {
    // WorkingResult.csvの読み込み準備
    File workingResultFile = new File(WORKING_RESULT_FILE_PATH);
    BufferedReader br = new BufferedReader(new FileReader(workingResultFile));

    // WorkingResult.csvを1行ずつ読み込んでArrayListに格納する
    String recode = br.readLine();
    while (recode != null) {
        workingResults.add(recode);
        recode = br.readLine();
    }
    br.close();
} catch (IOException e) {
    System.out.println(e);
}
```

CSVファイルから1行分ずつ読み込んだデータを格納した
ArrayList「workinResults」が完成！

```
// ArrayListから1行ずつ取り出して日付/出勤時間/退勤時間に振り分け
for (int i = 0; i < workingResults.size() ; i++) {
    
    String workingRecode    = workingResults.get(i);      // 1行ずつ文字列を取り出す
    String[] forSplitRecode = workingRecode.split(COMMA); // splitメソッドを用いてカンマ区切りで文字列を分解＆配列にそれぞれ格納
    
    Date workingDate = Date.valueOf(forSplitRecode[0]); // 出勤日
    Time startTime   = Time.valueOf(forSplitRecode[1]); // 出勤時間
    Time finishTime  = Time.valueOf(forSplitRecode[2]); // 退勤時間
    
    // getTimeメソッドを使って労働時間をミリ秒（0.001秒単位）で取得する
    long workingTime = finishTime.getTime() - startTime.getTime();
    
    // ミリ秒で取得した労働時間を○時間△分の形式に直す
    int workingHour = (int)( workingTime / ONE_HOUR_BY_MILLI_SEC );
    int workingMin  = (int)( ( workingTime / ONE_MIN_BY_MILLI_SEC ) %
    
    // 出力
    System.out.println( "【日付】"     + workingDate + " / " +
                        "【勤務時間】" + startTime + "〜" + finishTime
                        "【労働時間】" + workingHour + "時間" + workingMin + "分"
                      );
}
```

ArryListに格納された1行分の文字列を1つずつ取り出し、
そのデータを元にその日の労働時間を算出・表示していきます。

```
// ArrayListから1行ずつ取り出して日付/出勤時間/退勤時間に振り分け
for (int i = 0; i < workingResults.size() ; i++) {
    String workingRecode    = workingResults.get(i);       // 1行ずつ文字列を取り出す
    String[] forSplitRecode = workingRecode.split(COMMA); // splitメソッドを用いてカンマ区切りで文字列を分解＆配列にそれぞれ格納
    Date workingDate = Date.valueOf(forSplitRecode[0]); // 出
    Time startTime   = Time.valueOf(forSplitReco
    Time finishTime  = Time.valueOf(forSplitReco
    // getTimeメソッドを使って労働時間をミリ秒（
    long workingTime = finishTime.getTime() - sta
    // ミリ秒で取得した労働時間を○時間△分の形式
    int workingHour = (int)( workingTime / ONE_HO
    int workingMin  = (int)( ( workingTime / ONE_
    // 出力
    System.out.println( "【日付】"     + workingD
                        "【勤務時間】" + startTim
                        "【労働時間】" + workingH
                      );
}
```

String型の変数に対して使用できる「split」機能を使用し、文字列を『,』で分割して配列に変換します。

“2017-07-01,09:30:00,15:00:00”

split(“,”)

“2017-07-01”	“09:30:00”	“15:00:00”
[0]	[1]	[2]

```
// ArrayListから1行ずつ取り出して日付/出勤時間/退勤時間に振り分け
for (int i = 0; i < workingResults.size() ; i++) {
    
    String workingRecode    = workingResults.get(i);       // 1行ずつ文字列を取り出す
    String[] forSplitRecode = workingRecode.split(COMMA); // splitメソッドを用いてカンマ区切りで文字列を分解＆配列にそれぞれ格納
    
    Date workingDate = Date.valueOf(forSplitRecode[0]); // 出勤日
    Time startTime   = Time.valueOf(forSplitRecode[1]); // 出勤時間
    Time finishTime  = Time.valueOf(forSplitRecode[2]); // 退勤時間
    
    // getTimeメソッドを使って労働時間をミリ秒（0.001秒単位）で取得する
    long workingTime = finishTime.getTime() - startTime.getTime();
    
    // ミリ秒で取得した労働時間を○時間△分の形式に直す
    int workingHour = (int)( workingTime / ONE_HOUR_BY_MILLI_SEC );                      // 時間に換算
    int workingMin  = (int)( ( workingTime / ONE_MIN_BY_MILLI_SEC ) % ONE_HOUR_BY_MIN ); // 分に換算
    
    // 出力
    System.out.println( "【日付】"     + workingDate + " / " +
                        "【勤務時間】" + startTime + "〜" + finishTime + " / " +
                        "【労働時間】" + workingHour + "時間" + workingMin + "分"
                      );
}
```

String型の変数に対して使用できる「split」機能を使用し、文字列を『,』で分割して配列に変換します。

＜演習EX 1-3-2＞

『ReadFileSample.java』を参考に、 WorkingResult.csv（1ヵ月間の労働実績）を読み込み、
1ヵ月間の給与の総額を算出して出力するプログラムを作成してください。

＜仕様＞ ※EX1-3-1と同じ
・時給は900円とし、給与は1分単位で支払われます。
・小数点以下の給与は切り捨てて算出されます。
・労働時間が6時間超～8時間以下の場合は45分の休憩を、8時間を超える場合は1時間の休憩をとるものとします。なお、休憩時間において給与は発生しません。
・休憩時間を差し引いた労働時間（実労働時間）が8時間を超える場合、
超過分に限り残業代として1.25倍の給与が支払われるものとします。
・BigDecimalによる誤差の考慮は不要とします。
・クラス名は自由に決めてください。
（必ずクラス名を見ただけで処理内容を想像できるものにすること）