# **J2Kad18D「あとはよろしく！（throws）」（実践編P.53「メソッドの外への例外の送出」）**

J2Kad18Dにファイル「test.txt」へキーボードから入力した文字列を書き出したり、「test.txt」の内容を表示したりする処理が作成されている。ファイル入出力処理（**リスト1**の破線で囲まれた部分）をそれぞれメソッドにせよ。

**リスト1：ファイル入出力（ファイル「J2Kad18D.java」）**

public class J2Kad18D {

public static final String FILENAME = "test.txt";

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

while(true) {

System.out.print("どうしますか？（0：書き出す、1：読み込む、-1：終了）＞");

int cmd = Integer.parseInt(in.next());

if (cmd < 0) break;

try {

switch (cmd) {

case 0: // 書き出し

System.out.print("書き出す文字列を入力してください＞");

String str = in.next();

BufferedWriter bw = new BufferedWriter(new FileWriter(FILENAME));

bw.write(str);

bw.close();

break;

case 1:

BufferedReader br = new BufferedReader(new FileReader(FILENAME));

String line;

while((line = br.readLine()) != null) {

System.out.println(line);

}

br.close();

break;

}

} catch(IOException e) {

System.out.println(e);

}

}

}

}

void MyWriter(String str)

void MyReader()

**課題完成時の画面**

どうしますか？（0：書き出す、1：読み込む、-1：終了）＞**0**

書き出す文字列を入力してください＞**ECCCOMP**

どうしますか？（0：書き出す、1：読み込む、-1：終了）＞**1**

ECCCOMP

# **J2Kad18C「例外を投げよう！（throw）」（実践編P.51「例外の作成」、throw）**

以下の仕様で処理を作成せよ。

1. FileNotExceptionとEOFExceptionをスローしてキャッチする処理を作成せよ。
2. tryブロックをメソッド（throwExceptionメソッド）にせよ。

**リスト1：例外のスロー（ファイル「J2Kad18C.java」）**

public class J2Kad18C {

public static void main(String[] args) {

System.out.println("例外をスローしてキャッチします！");

while(true) {

try {

Scanner in = new Scanner(System.in);

System.out.print("どの例外をスローしますか？（0:FileNotFoundException、1:EOFException）＞");

int n = Integer.parseInt(in.next());

**仕様①：FileNotFoundExceptionとEOFExceptionを投げて（スローして）キャッチする処理を作成する**

} catch (Exception e) {

System.out.println(e + "をキャッチしました！");

System.out.println("終了します！");

break;

}

System.out.println();

}

}

**仕様②：throwExceptionメソッドを作る**

}

**課題完成時の画面**

例外をスローしてキャッチします！

どの例外をスローしますか？（0:FileNotFoundException、1:EOFException）＞**0**

FileNotFoundExceptionをキャッチしました！

どの例外をスローしますか？（0:FileNotFoundException、1:EOFException）＞**1**

EOFExceptionをキャッチしました！

どの例外をスローしますか？（0:FileNotFoundException、1:EOFException）＞**ecc**

java.lang.NumberFormatException: For input string: "ecc"をキャッチしました！

終了します！

# **J2Kad18B「例外を作ろう！」（実践編P.51「例外の作成」）※J2Kad18Cをコピーして作成**

J2Kad18Cをコピーし、以下の仕様を追加せよ。

1. 選択肢に「2：IOException」を追加し、IOExceptionのスローとキャッチを追加せよ。
2. 例外クラスとしてNegativeNumberExceptionクラス（負の数が入力されたときに発生）とOverflowExceptionクラス（3以上の数が入力されたときに発生）を追加せよ。

**課題完成時の画面**

例外をスローしてキャッチします！

どの例外をスローしますか？（0:FileNotFoundException、1:EOFException、2：IOException）＞**2**

IOExceptionをキャッチしました！

どの例外をスローしますか？（0:FileNotFoundException、1:EOFException、2：IOException）＞**-1**

NegativeNumberException: 負の数です！

どの例外をスローしますか？（0:FileNotFoundException、1:EOFException、2：IOException）＞**3**

OverflowException: 0から2までの整数を入力してください！

どの例外をスローしますか？（0:FileNotFoundException、1:EOFException、2：IOException）＞**ecc**

java.lang.NumberFormatException: For input string: "ecc"をキャッチしました！

終了します！

NegativeNumberExceptionでは「負の数です！」、

OverflowExceptionでは「0から2までの整数を入力してください」

と表示されるようにすること。

# **J2Kad18A「ECC野球部！」**

あなたは名門、ECC野球部のキャッチャーです。ピッチャーが投げる様々な球種（例外）をキャッチしてください。

**球種（例外クラス）の仕様**

|  |  |  |  |
| --- | --- | --- | --- |
| No. | 球種 | 例外クラス | 例外メッセージ |
| 0 | 直球 | Straight | ストレートを投げた！ |
| 1 | カーブ | Curve | カーブを投げた！ |
| 2 | フォーク | ForkBall | フォークボールを投げた！ |

**例外を投げるメソッド（J2Kad18Aクラスに作成）**

|  |  |
| --- | --- |
| メソッド | 仕様 |
| public static void throwBall(BALL ball) | ballで指定された番号の球種を投げる（例外をスローする）。 |

**リスト1：ECC野球部（ファイル「J2Kad18A.java」）**

public class J2Kad18A {

enum BALL {STRAIGHT, CURVE, FORKBALL}

public static void main(String[] args) {

…

BALL[] balls = BALL.values();

while (true) {

**throwBallメソッドを呼び出して例外をキャッチする処理を作成すること**

}

}

}

**課題完成時の画面**

あなたはECC野球部の名キャッチャーです！

サインを出してピッチャーの球をキャッチしてください！

何のサインを出しますか？（0：直球、1：カーブ、2：フォーク、-1：終了）＞**0**

Straight: ストレートを投げた！

キャッチしました！

何のサインを出しますか？（0：直球、1：カーブ、2：フォーク、-1：終了）＞**1**

Curve: カーブを投げた！

キャッチしました！

何のサインを出しますか？（0：直球、1：カーブ、2：フォーク、-1：終了）＞**2**

ForkBall: フォークボールを投げた！

キャッチしました！

何のサインを出しますか？（0：直球、1：カーブ、2：フォーク、-1：終了）＞**3**

そんな球種はありません！

何のサインを出しますか？（0：直球、1：カーブ、2：フォーク、-1：終了）＞**-1**

ヒント：

Straight、Curve、ForkBallを

まとめるクラスがあると便利

# **J2Kad18S「MyDecorator（Decoratorパターン）」**

文字列を保持するStringHolderクラスが準備されている。コンストラクタで文字列を受け取り、readメソッドで文字列を返す。文字列を装飾するデコレータクラスを作成し、動作確認を行え。作成するデコレータクラスは以下の通り。

**作成するデコレータクラス（ファイル「J2Kad18S.java」に作成）**

|  |  |
| --- | --- |
| クラス | デコレーション |
| StarDecorator | 文字列の前後に「\*\*\*」を付ける。  例：ECCCOMP　→　\*\*\* ECCCOMP \*\*\* |
| BlockDecorator | 文字列を線で囲む。  例： ECCCOMP　→ +-------+  |ECCCOMP|  +-------+ |

**リスト1：MyDecorator（ファイル「J2Kad18S.java」）**

public class J2Kad18S {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

System.out.println("デコレーションします！");

System.out.print("デコレーションする文字列を入力してください＞");

StringHolder holder = new StringHolder(in.next());

while(true) {

System.out.print("どうしますか？（0：Star、1：Block、2：Star＋Block、-1：終了＞");

int n = Integer.parseInt(in.next());

if (n < 0) break;

IReader reader = holder;

**switch(n) {**

**case 0: reader = new StarDecorator(reader); break;**

**case 1: reader = new BlockDecorator(reader); break;**

**case 2: reader = new BlockDecorator(new StarDecorator(reader)); break;**

**}**

System.out.println(reader.read());

}

}

}

追加する

**課題完成時の画面**

デコレーションします！

デコレーションする文字列を入力してください＞**ECCCOMP**

どうしますか？（0：Star、1：Block、2：Star＋Block、-1：終了＞**0**

\*\*\* ECCCOMP \*\*\*

どうしますか？（0：Star、1：Block、2：Star＋Block、-1：終了＞**1**

+-------+

|ECCCOMP|

+-------+

どうしますか？（0：Star、1：Block、2：Star＋Block、-1：終了＞**2**

+---------------+

|\*\*\* ECCCOMP \*\*\*|

+---------------+

どうしますか？（0：Star、1：Block、2：Star＋Block、-1：終了＞**-1**

**課題完成時のクラス図（Decoratorパターン←検索）**

|  |
| --- |
| <<interface>>  ***IReader*** |
|  |
| *+ read() : String* |

追加作成するクラス

J2Kad18S.javaに作成

|  |
| --- |
| **StringHolder** |
| - str : String |
| + StringHolder(str : String)  + read() : String |

|  |
| --- |
| ***MyDecorator*** |
| # src : MyReader |
| + MyDecorator(src : IReader) |

|  |
| --- |
| **StarDecorator** |
|  |
| + StarDecorator(src : IReader)  + read() : String |

|  |
| --- |
| **BlockDecorator** |
|  |
| + BlockDecorator(src : IReader)  + read() : String |

# **J2Kad18X1「新装開店！ECCコーヒー①」**

ECCコーヒーが新装開店した！この機会にドリンクだけでなくトッピングも選べるようにしたい。そこで早速、業者に頼んでドリンク注文プログラムを作ってもらった。ところがこれがとんでもないシロモノだった！！ドリンクとトッピングの組み合わせごとにクラスを作っているため、ドリンクの種類×トッピングの種類だけクラスが存在する！これではドリンクやトッピングの種類が増えるごとにとんでもなくクラスが増えてしまう！！Decoratorパターンを使ってリファクタリングを行え。

**ドリンクの種類と値段 トッピングの種類と値段**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ドリンク | 値段 |  | トッピング | 値段 |
| ブレンドコーヒー（HouseBlend） | 380円 |  | 豆乳（Soy） | +40円 |
| エスプレッソ（Espresso） | 320円 |  | ホイップクリーム（Whip） | +50円 |
| 深煎りコーヒー（DarkRoast） | 400円 |  | ココア（Cocoa） | +30円 |

**業者のクラス設計（ドリンクとトッピングの組み合わせごとにクラスを作成、ファイル「MenuItem.java」）**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ドリンク | トッピングなし | 豆乳 | ホイップクリーム | ココア |
| ブレンドコーヒー | HouseBlendクラス | HouseBlendSoyクラス | HouseBlendWhipクラス | HouseBlendCocoaクラス |
| エスプレッソ | Espressoクラス | EspressoSoyクラス | EspressoWhipクラス | EspressoCocoaクラス |
| 深煎りコーヒー | DarkRoastクラス | DarkRoastSoyクラス | DarkRoastWhipクラス | DarkRoastCocoaクラス |

**リスト1：業者の作った恐ろしいmainメソッド（ファイル「J2Kad18X1.java」）**

public class J2Kad18X1 {

public static void main(String[] args) {

…

// ドリンクの生成

MenuItem item = null;

switch(drink) {

default:

case 0: // ブレンドコーヒー

switch(topping) {

default: item = new HouseBlend(); break;

case 0: item = new HouseBlendSoy(); break;

case 1: item = new HouseBlendWhip(); break;

case 2: item = new HouseBlendCocoa(); break;

}

break;

case 1: // エスプレッソ

…

（恐ろしいので省略、良い子はこんなコードは作らないこと）

…

}

// 会計

System.out.println("お待たせしました！");

System.out.println(item.getName() + "です！");

System.out.println(item.getPrice() + "円になります！");

…

ドリンクごとに場合分けし、さらに

トッピングごとに場合分けしている。

**課題完成時の画面**

ECCコーヒーへようこそ！

飲み物を選んでください（0：ブレンド、1：エスプレッソ、2：深煎りコーヒー、-1：店を出る）＞**0**

トッピングはどうしますか？（0：豆乳、1：ホイップ、2：ココア、-1：いらない）＞**-1**

お待たせしました！

ブレンドコーヒーです！

380円になります！

飲み物を選んでください（0：ブレンド、1：エスプレッソ、2：深煎りコーヒー、-1：店を出る）＞**1**

トッピングはどうしますか？（0：豆乳、1：ホイップ、2：ココア、-1：いらない）＞**0**

豆乳を追加しました！

お待たせしました！

エスプレッソ＋豆乳です！

360円になります！

飲み物を選んでください（0：ブレンド、1：エスプレッソ、2：深煎りコーヒー、-1：店を出る）＞**-1**

ありがとうございました！

# **J2Kad18X2「新装開店！ECCコーヒー②」※J2Kad18X1をコピーして作成**

「お客様のために！」ECCコーヒーはさらに進化します。J2Kad18X1をコピーし、以下の仕様を追加せよ。

1. ドリンク選択後、ラージサイズにするかどうかを選択する（ラージサイズは＋50円）。
2. トッピングを複数追加できるようにする（例：豆乳＋ホイップ、ホイップ＋ホイップ＋ココア、など）。
3. トッピングを値上げする（豆乳：50円、ホイップ：60円：ココア：40円）

**課題完成時の画面（深煎りコーヒー＋ラージ＋ホイップ＋ホイップ＋ココアの場合）**

ECCコーヒーへようこそ！

飲み物を選んでください（0：ブレンド、1：エスプレッソ、2：深煎りコーヒー、-1：店を出る）＞**2**

ラージサイズにしますか？（0：する、-1：しない）＞**0**

ラージを追加しました！

トッピングはどうしますか？（0：豆乳、1：ホイップ、2：ココア、-1：いらない）＞**1**

ホイップを追加しました！

トッピングはどうしますか？（0：豆乳、1：ホイップ、2：ココア、-1：いらない）＞**1**

ホイップを追加しました！

トッピングはどうしますか？（0：豆乳、1：ホイップ、2：ココア、-1：いらない）＞**2**

ココアを追加しました！

トッピングはどうしますか？（0：豆乳、1：ホイップ、2：ココア、-1：いらない）＞**-1**

お待たせしました！

深煎りコーヒー＋ラージ＋ホイップ＋ホイップ＋ココアです！

610円になります！

J2Kad18X1作成前の業者のプログラムで作成した場合、

どんなコードになるのか想像してみること（ああ、恐ろしい）。