## ● J2Kad19D「あとはよろしく！（throws）」

J2Kad19D 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「J2Kad19D.java」）**

```
public class J2Kad19D {
    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
```

## ● J2Kad19C「例外を投げよう！（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「J2Kad19C.java」）**

```
public class J2Kad19C {
    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());
                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();
        }
    }
}
```

**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！
```

## ● J2Kad19B「例外を作ろう！」※J2Kad19C の main と throwException をコピーして作成

J2Kad19C の main メソッドと throwException メソッドをコピーし、さらに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。

## ● J2Kad19A「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（J2Kad19A 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「J2Kad19A.java」）**

```
public class J2Kad19A {
    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

## ● J2Kad19S「MyDecorator」

ファイルからの入力を行うMyFileReaderクラスとキーボードからの入力を行うMyScannerクラスが準備されている。以下のデコレータクラスを追加し動作確認を行え。

**MyFileReader クラスの仕様（MyReader を継承、作成済み）**

| メソッド | 仕様 |
|---|---|
| public MyFileReader(String filename) | フィールド filename に引数 filename を保存する。 |
| public String read() | 「filename＋の中のデータです！」という文字列を返す。<br>(実際のファイル入力は行わない) |

**MyScanner クラスの仕様（MyReader を継承、作成済み）**

| メソッド | 仕様 |
|---|---|
| public MyScanner(InputStream souce) | source（System.in のこと）を入力元とする Scanner クラスを作る。 |
| public String read() | 「何か文字列を入力してください＞」と表示し、入力された文字列を返す。 |

**作成するデコレータクラス（ファイル「J2Kad19S.java」に作成）**

| クラス | デコレーション |
|---|---|
| MyStarDecorator | 文字列の前後に「***」を付ける。文字列　→　***文字列*** |
| MyBracketsDecorator | 文字列をカッコでくくる。文字列　→　［文字列］ |

**デコレーションの仕様**

・MyFileReader①　　MyStarDecorator でデコレーションする
・MyScanner　　MyBracketsDecorator でデコレーションする
・MyFileReader②　　MyBracketsDecorator＋MyBracketsDecorator＋MyStarDecorator でデコレーションする

**リスト１：課題作成前の main メソッド（ファイル「J2Kad19S.java」）**

```
public class J2Kad19S {
    public static void main(String[] args) {
        MyReader in;
        // MyFileReader①
        in = new MyFileReader("test.txt");
        System.out.println(in.read());
        // MyScanner
        in = new MyScanner(System.in);
        System.out.println(in.read());
        // MyFileReader②
        in = new MyFileReader("test2.txt");
        System.out.println(in.read());
    }
}
```

MyStarDecorator でデコレーション

MyBracketsDecorator でデコレーション

MyBracketsDecorator＋MyBracketsDecorator
＋MyStarDecorator でデコレーション

**リスト2：課題完成時のmainメソッド（ファイル「J2Kad19S.java」）**

```
public class J2Kad19S {
    public static void main(String[] args) {
        MyReader in;
        // MyFileReader①
        in = new MyStarDecorator(new MyFileReader("test.txt"));
        System.out.println(in.read());
        // MyScanner
        in = new MyBracketsDecorator(new MyScanner(System.in));
        System.out.println(in.read());
        // MyFileReader②
        in = new MyBracketsDecorator(new MyBracketsDecorator(
                        new MyStarDecorator(new MyFileReader("test2.txt"))));
        System.out.println(in.read());
    }
}
```

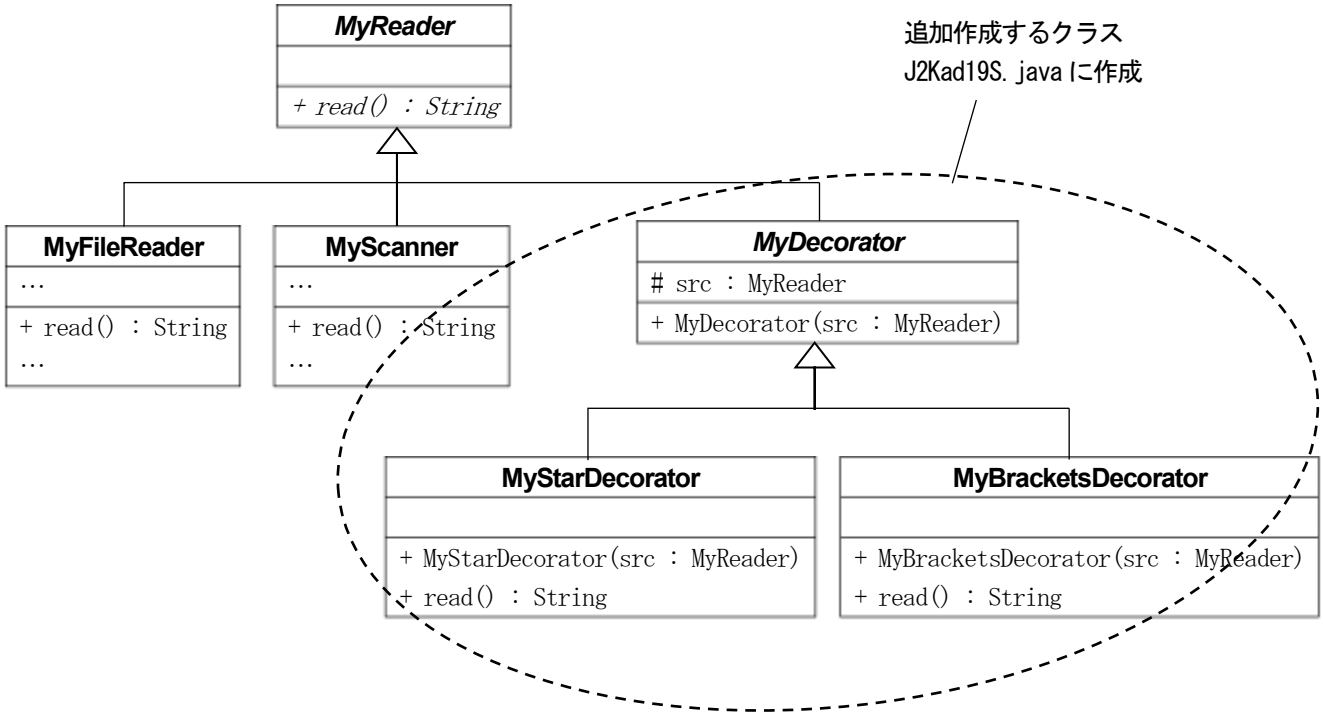
**課題作成前の画面（Before）**

```
test.txt の中のデータです！
何か文字列を入力してください>ECCCOMP
ECCCOMP
test2.txt の中のデータです！
```

**課題完成時の画面（After）**

```
***test.txt の中のデータです！***
何か文字列を入力してください>ECCCOMP
[ECCCOMP]
[[***test2.txt の中のデータです！***]]
```

**課題完成時のクラス図（Decoratorパターン←検索）**



## ● J2Kad19X1「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**

| ドリンク | 値段 |
|---|---|
| ブレンドコーヒー（HouseBlend） | 380 円 |
| エスプレッソ（Espresso） | 320 円 |
| 深煎りコーヒー（DarkRoast） | 400 円 |

**トッピングの種類と値段**

| トッピング | 値段 |
|---|---|
| 豆乳（Soy） | +40 円 |
| ホイップクリーム（Whip） | +50 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 | HouseBlendSoy | HouseBlendWhip | HouseBlendCocoa |
| エスプレッソ | Espresso | EspressoSoy | EspressoWhip | EspressoCocoa |
| 深煎りコーヒー | DarkRoast | DarkRoastSoy | DarkRoastWhip | DarkRoastCocoa |

**リスト１：業者の作った恐ろしい main メソッド（ファイル「J2Kad20X1.java」）**

```
public class J2Kad19X1 {
    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！");
            ⋮
```

**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！
```

## ● J2Kad19X2「新装開店！ECC コーヒー②」※J2Kad19X1 の main メソッドをコピーして改造

J2Kad19X1 の main メソッドをコピーし、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！
```

J2Kad19X1 作成前の業者のプログラムで同じ仕様にする場合、
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）。